A User Level Framework for Ad hoc Routing

Jeremie Allard Paul Gonin
Minoo Singh Golden G. Richard III
Dept. of Computer Science
University of New Orleans
New Orleans, LA 70148
Contacts: {jallard, paul, minoo, golden}@cs.uno.edu

Abstract

The availability of inexpensive wireless networking hardware (e.g., based on the IEEE 802.11 standards) has generated interest in a large class of wireless applications. Many applications benefit from rapidly deployable networks—for example, collaborative applications to support field research or emergency incident response. The need for networks that can be rapidly deployed has resulted in a substantial body of research in ad hoc routing protocols. Such protocols use intermediate nodes as routers and support highly dynamic network configurations.

We have developed a portable, user-level framework for ad hoc routing in C++. In our current implementation of this framework a tailored SOCKS proxy handles client requests and uses an implementation of an ad hoc routing protocol to provide routing. So far, implementations of DSR and flooding are provided, but other routing protocol implementations can easily be incorporated. An integrated simulator allows new routing protocols to be tested, and the code can be moved to a production ad hoc deployment with no modification. Our framework is suitable for a number of purposes, from ad hoc routing protocol research, where new protocols can be rapidly developed and tested, to the deployment of real ad hoc networks. The system is easily installed on a wide variety of operating systems and requires no kernel hacking.

1. Introduction

Research into ad hoc routing protocols, which have their roots in packet radio networks, has gained momentum recently as wireless networks move into the mainstream. This is due primarily to the availability of inexpensive wireless networking hardware and a range of envisioned applications. Currently, most deployed wireless networks are built using base stations. The disadvantages of such "infrastructure" wireless networks are primarily high cost and the inability to deploy such networks "on the spot" (as in an emergency response scenario). Ad hoc routing protocols use mobile nodes as routers, allowing a network of arbitrary diameter to be deployed without base stations (given sufficient node density).

The IEEE 802.11 standards, on which most currently deployed high bitrate networks are based, specifies an "ad hoc" mode, where mobile units within transmission range of each other (typically within hundreds of feet, though obstructions in the environment drastically reduce the range) can communicate without infrastructure. No routing is supported, however, which severely restricts the diameter of the network. Unfortunately, given current operating systems support for ad hoc networks and the state of current ad hoc routing implementations, configuring nodes in order to deploy a routable ad hoc network is a tedious process, often requiring modification to the operating system kernel.

Because we feel that there is substantial interest in routable ad hoc networks that can be easily deployed, we have developed an ad hoc routing architecture that resides entirely at user level. Although we currently supply implementations of Dynamic Source Routing [1] and flooding, any ad hoc routing protocol can be easily incorporated. Our architecture has been developed in C++ and is designed to be portable to a number of operating systems, including Windows, Mac OS X, and various flavors of Unix.

We envision a number of scenarios where the architecture will prove useful:

- For researchers working on new ad hoc routing protocols, or enhancements to existing protocols, our system allows rapid implementation and testing.
- The integrated simulator makes experimentation with ad hoc routing protocols in a classroom or laboratory setting straightforward.
- The architecture can be used for rapid deployment of ad hoc networks today, for field research, tourist applications, emergency incident response, etc.

2. Previous Work

Flooding, Dynamic Source Routing (DSR) [1], and Ad-Hoc On Demand Distance Vector Routing (AODV) [2] are just a few of the many proposed ad-hoc routing protocols. Though flooding is an attractive option because of its simplicity, it carries a high overhead, particularly for larger ad hoc networks. DSR uses source routing, in which the sender determines the sequence of hops to the destination (the route) and includes this information explicitly in the header of each data packet. AODV, on the other hand, takes a distance vector approach, maintaining routing tables at each node with next-hop information. It determines routes on-demand and maintains only recently used routes. A combination of sequence numbers and packet information is used to forward packets and to avoid routing loops.

There are several user-level and kernel-level implementations of these protocols (and others). These include the mad-hoc AODV implementation for Linux [3], the CMU Monarch implementation of DSR for FreeBSD 3.3 and 3.4 [4], and the INRIA implementation of Optimized Link State Routing (OLSR) [5]. Our goal, rather than implementing a particular ad hoc routing protocol for a particular operating system, has been to develop a *framework* that allows quick implementation of *any* ad hoc routing protocol. Our framework is portable across a variety of operating systems and frees the developer from dealing with a ground-up effort for each protocol.

Routing protocols must be thoroughly tested before deployment. Therefore, simulation is often undertaken before a concrete implementation. Extensive work has been done to develop several simulation environments for wireless (and wired) networks. GloMoSim [6] and ns2 [7] are two such simulation environments. The simulator provided in our framework is not yet as mature as ns2 or GloMoSim, but in contrast to these simulation environments, an implementation of an ad hoc routing protocol can be deployed after simulation w/o modification.

3. Problems

Currently, implementing an ad hoc routing protocol involves extensive work on low-level issues, often requiring modification to an operating system's network stack. Unfortunately, this tedious work must be repeated to port the routing protocol to a different system (and not only between, e.g., Linux and Microsoft Windows, but between different versions of Windows as well). Consequently, implementers of ad-hoc routing protocols cannot concentrate their effort on actual protocol issues until they have solved these low-level issues.

Another waste of effort is caused by the separation between simulators, testbed and deployment platforms. It is quite frustrating, for example, to implement a routing protocol in ns2 or GloMoSim, only to have to start from scratch to deploy it in an actual ad hoc network.

But protocol implementation is not the only area in which we currently identify problems. Present ad hoc implementations can be quite frustrating for those who are interested in developing or using applications for ad hoc networks. Installation and configuration is often extremely difficult and requires extensive system administration skill (such as patching and recompiling the kernel source or installing device drivers).

4. Design Details

4.1. Goals

Our system attempts to provide a flexible, simple to use solution for developing and deploying ad-hoc routing protocols. We wanted to provide the following:

- Support for conventional applications: we should be able to support standard Internet applications (web browsers, mail, ftp, ssh—any TCP or UDP based application) without modification.
- Abstraction of the environment: details of the operating system and the networking hardware are hidden from the protocol implementation. This enables the implementer of a routing protocol to concentrate on protocol details.
- Fast development: adding a routing protocol should be as easy as possible. This implies the availability of facilities for debugging and simulation, and the interfaces the routing protocol must use should be simple and well designed.
- Easy deployment: installing the platform should be possible on as many systems as possible (it should be portable), and should be easy (any user should be able to do it).
- Connectivity to the Internet: it should be possible to access the Internet from the ad-hoc nodes, without modification on the Internet side (we cannot add our system in all Internet's servers).
- Configuration: As little configuration as possible should be necessary (primarily, network environment settings such as DNS and broadcast addresses).

As we describe in the rest of this section, our system uses a user-level proxy to execute the routing protocols, which fulfills the previous conditions. The design of the system is detailed in the following sections.

4.2. Design

The architecture is designed in a layered manner, using the components illustrated in Figure 1.

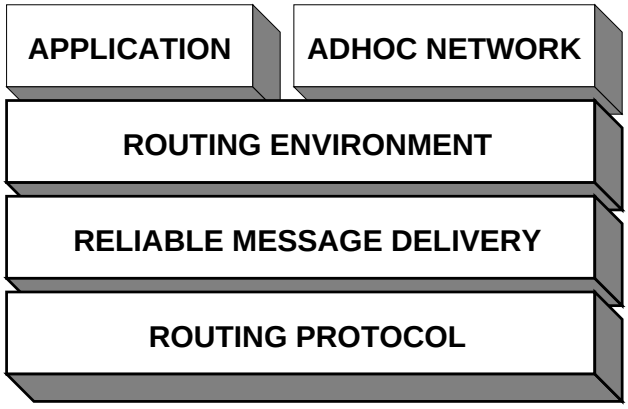

Figure 1. Ad hoc routing architecture.

A message traverses the layers in the following order:

application → routing environment → reliable message delivery → routing protocol → reliable message delivery → routing environment → adhoc network

4.2.1. Application

The current implementation supports all TCP and UDP applications including web browsers, ssh, telnet, ftp etc.

4.2.2. Ad hoc network

The ad hoc network consists of a group of nodes that can transmit messages to each other. Messages can either unicast or broadcast. A broadcast operation involves the emission of a message from a node to all its neighbors.

Since there is no routing support at this layer, nodes may communicate only with their immediate neighbors (i.e., nodes that are accessible without routing). Communication is unreliable—there is no retransmission facility. For simplicity, we only consider bi-directional links.

4.2.3. Routing environment

The routing environment is the layer in charge of abstracting system-dependent issues from the ad hoc routing protocol. A protocol is always initialized with a reference to the routing environment. It provides 4 important methods to the protocol, namely,

- `isLocalClient` is used by the protocol to establish if a destination is accessible from that node (the destination is the node itself or it is accessible through a different interface).
- `sendClientMessage` is used by the protocol to send a message to a client. This client should be accessible to the node (that is, `isLocalClient` returns true).
- `sendRoutingMessage` is used by the protocol to send a routing message to another node (routing environment).
- `broadcastRoutingMessage` is used by the protocol to send a routing message to all the other neighbor nodes.

The routing environment is associated with an ad hoc routing protocol and transmits incoming messages by using the methods of the routing protocol interface discussed in the next section.

Various implementations for a routing environment are imaginable. It would typically be a driver and run at kernel level, but we chose a more flexible and portable approach in our implementation in the form of a SOCKS5 proxy [8].

The rationale behind this choice is our desire to provide an environment that is easy to install and to use. A SOCKS5 proxy approach has many advantages. It runs in user space, and therefore it is easy for a simple user to start, stop or restart it without 'polluting' the kernel space. It is supported by many TCP and UDP applications, either directly or with the use of a SOCKS wrapper [9] and therefore does not require applications to be rewritten for the ad hoc environment. Also, this solution is portable between different operating systems.

As we wanted to provide the nodes on the ad hoc network with connectivity to the Internet through one or more Internet gateway nodes, we had to figure out a way to address DNS name resolution. As a result, our SOCKS5 routing environment acts as a fake DNS server and our nodes are set to use their local address as a DNS server. The proxy is configured with the location of a remote DNS server. DNS requests are forwarded via the routing protocol layer to the remote DNS server. Figure 2 illustrates the sequence of actions that take place when a client asks for DNS name resolution.

4.2.4. Routing protocol

The routing protocol layer is the implementation of the ad hoc routing protocol itself. Using the ad hoc network, it must be able to transmit messages from one node to another, even if the destination is not within the transmission range of the source node. A routing protocol has to implement the `RoutingProtocol` interface, which specifies the following methods:

- `incomingClientMessage` is used by the environment to notify the routing protocol that a message from a client application needs to be routed to its destination.
- `incomingRoutingMessage` is used by the environment to notify the routing protocol that a message from another routing node has been received and needs to be processed.

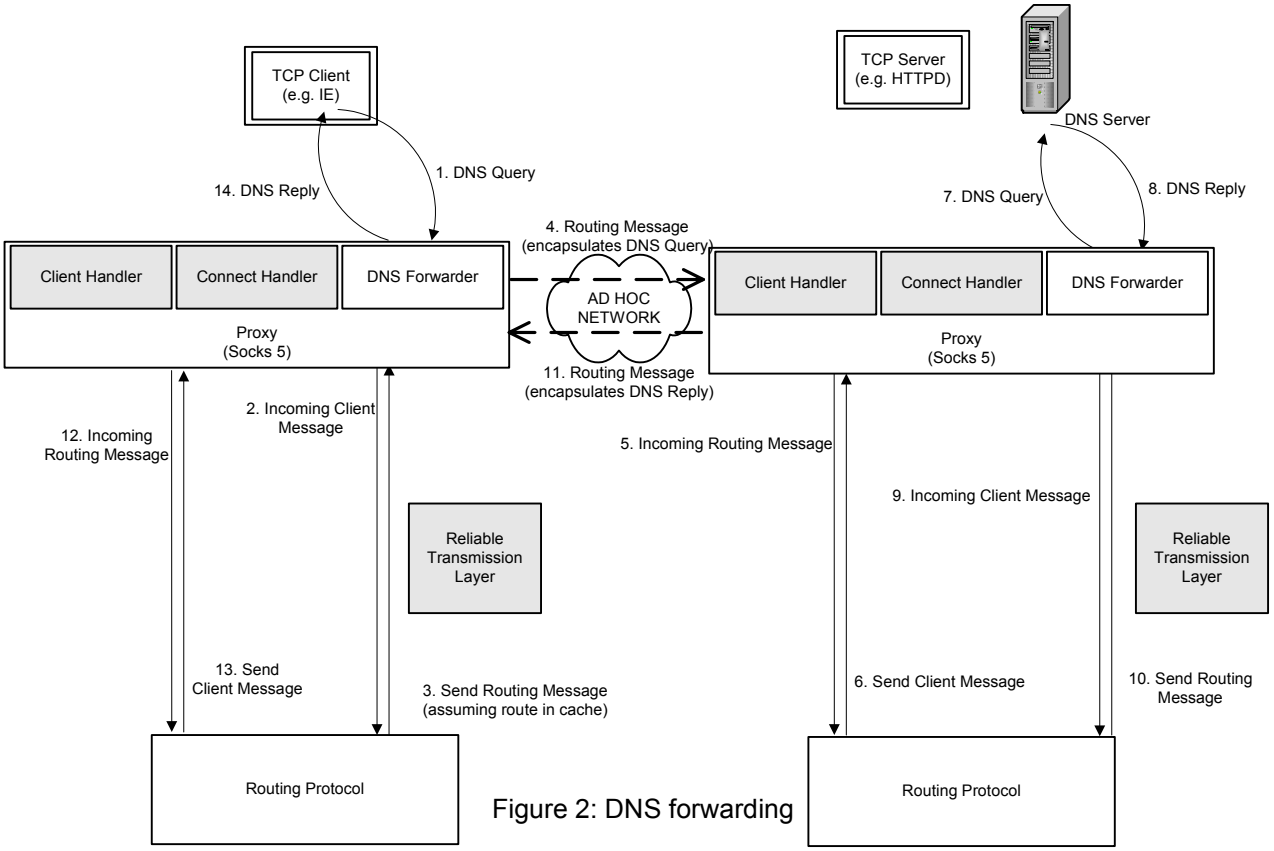

Figure 2: DNS forwarding

4.2.5. Reliable message delivery

The reliable message delivery layer was added to ensure reliable connection-oriented communications on top of the routing protocol implementation. This layer is necessary because we do not use the standard TCP/IP stack to route messages and because the routing protocol implementation is allowed to be unreliable (in fact, most popular ad hoc routing protocols do not address reliable delivery). An alternative would have been to force the routing protocols to implement methods that ensured reliable delivery of packets. But this would contradict one of the major goals of our design—allowing implementations of routing protocols to be developed quickly without modification to the specifications of the routing protocol. We currently use a custom, reliable message delivery protocol that, from the point of view of applications, appears to provide a standard TCP connection.

This layer defines two additional interfaces, namely:

- `ReliableRoutingProtocol` extends `RoutingProtocol` and provides a new method called `incomingReliableMessage`. This method is used by the routing environment to notify that a client request needs to be reliably transmitted.
- `ReliableRoutingEnvironment` extends `RoutingEnvironment` and provides a new method called `sendReliableMessage`. This method is used by the (reliable) routing protocol to reliably route a message toward the destination (therefore, it establishes a TCP connection with the destination).

The reliable message delivery is based on the use of connection identifiers. When a client application wants to establish a TCP connection to a remote host, its request is assigned a connection ID. All messages for the same connection (i.e. having the same connection ID) and for the same destination IP are guaranteed to be delivered in order, except when the `incomingReliableMessage` method returns an error (return value is not zero). In this case, the message could not be reliably transmitted, the connection is considered broken and should not be used anymore.

4.2.6 Implementation of the routing protocol layer

The protocols currently included are flooding and DSR. Other ad hoc routing protocols such as AODV, OLSR, TORA etc. may be easily integrated. In flooding, the message is broadcast to all nodes in the ad hoc network. The nodes, upon reception of the message, take appropriate actions, if any. However, the overhead associated with flooding can be quite high because all nodes receive and process the message, regardless of whether

they are the targets. DSR attempts to decrease this overhead by establishing a route (sequence of hops) to the destination and using this route to send messages to the destination. The implementation of DSR is based on [1].

Figure 3 details the route establishment process for DSR. When a client makes a TCP request, the routing environment forwards this request using `incomingReliableMessage` to the reliable routing environment layer. The reliable routing environment then sends this message to the routing protocol layer using `incomingClientMessage`. The node's cache is searched to find a route to the destination. If a route is not found, a route request is broadcast using the `broadcastRoutingMessage` method. Upon reception of the route request, the node may either forward it (after adding itself to the route) or return a route reply (if it is the destination of the request or it has a cached route to the destination). The route reply is sent using the `sendRoutingMessage` method of the routing environment. When the initiator node receives the route reply, it caches the route and uses it to forward data to the destination. Figure 4 illustrates how data is transferred between the nodes.

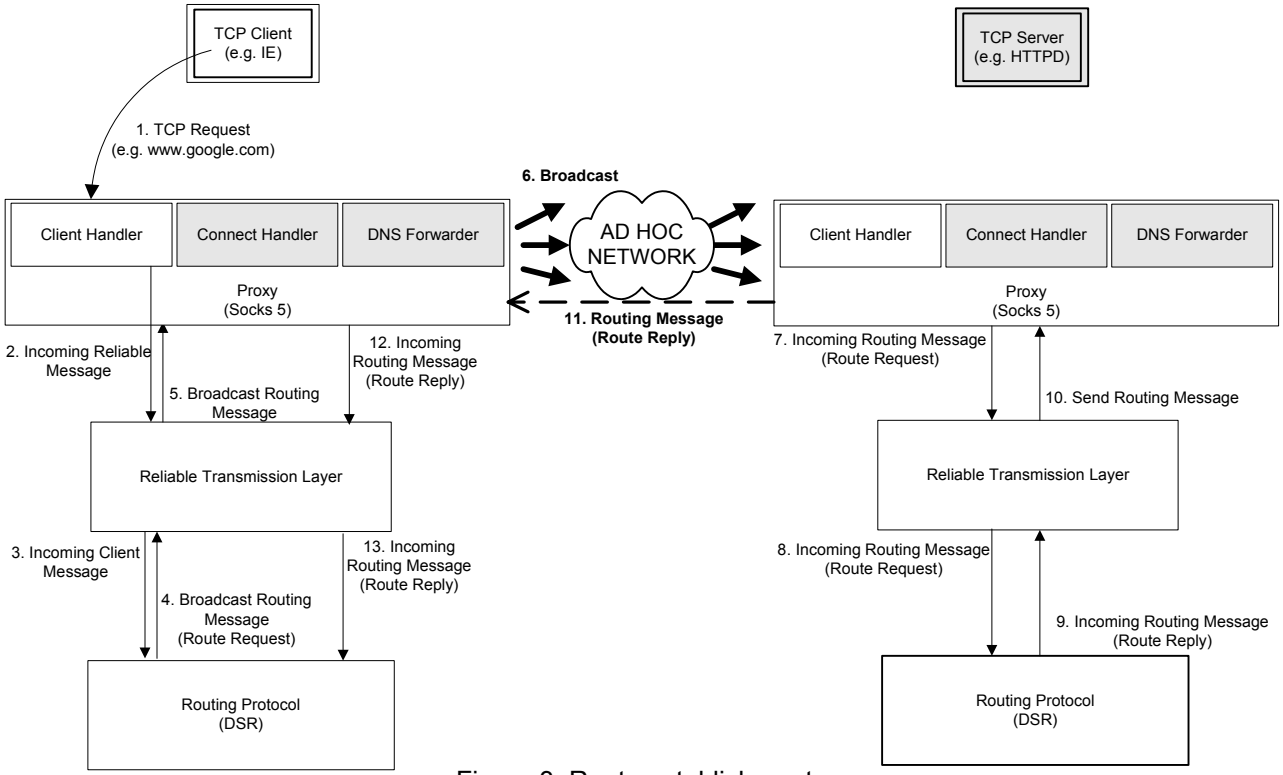

Figure 3: Route establishment

4.3. Testing and Deployment

We have implemented a simulator in order to assist in the development of routing protocols on our platform. This simulator implements the `RoutingEnvironment` interface and thus can be used with protocols that use that interface.

The simulator provides a graphical representation of the simulated network in order to allow an intuitive and quick understanding of the behavior of the protocol. Logs are also available to carefully verify that the implementation of the protocol works correctly.

When the simulator is started, nodes are randomly positioned on a rectangular canvas (the network). We currently support the random direction mobility model. In this model, nodes are assigned an initial direction and speed. When a node hits the network's boundary, it chooses another random direction. The network configuration can be modified by changing the number of nodes, the transmission range, latency and the number of nodes initiating messages. Each of these is a parameter in the initial configuration of the simulator. Other mobility models can be easily incorporated. A snapshot of the simulator in action is presented in Figure 5.

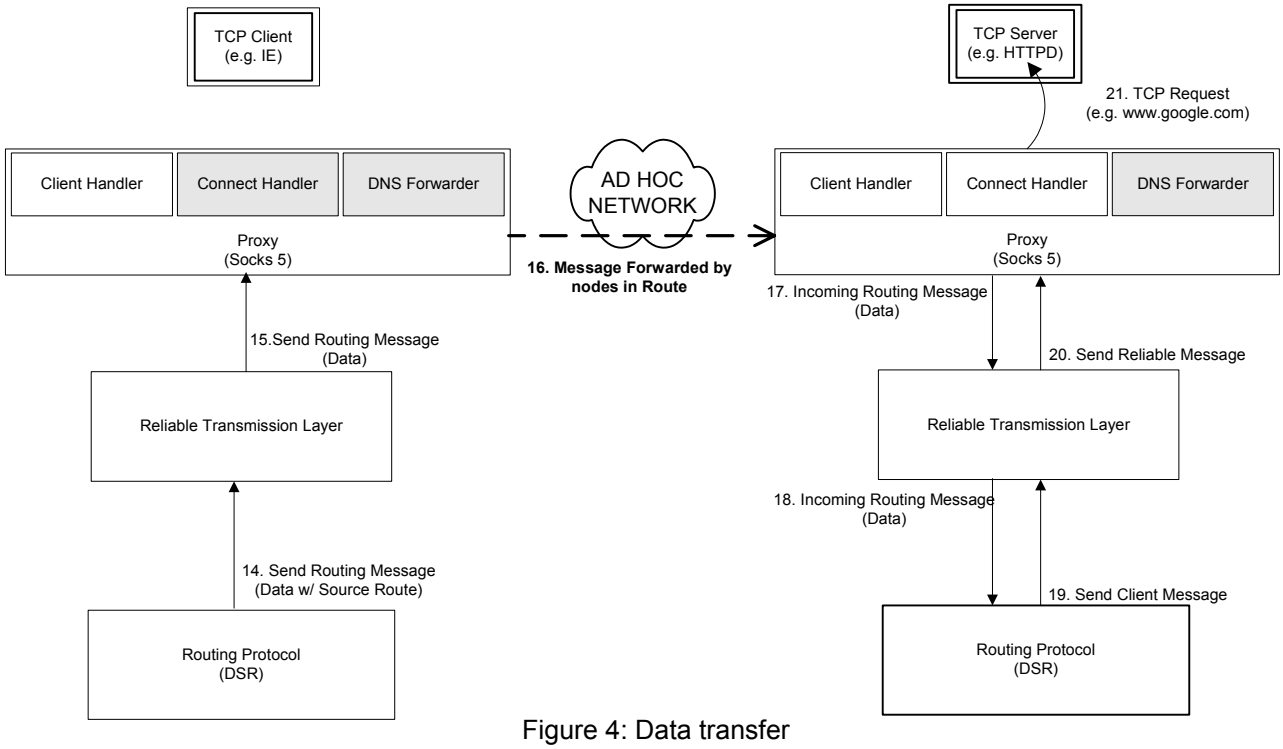

Figure 4: Data transfer

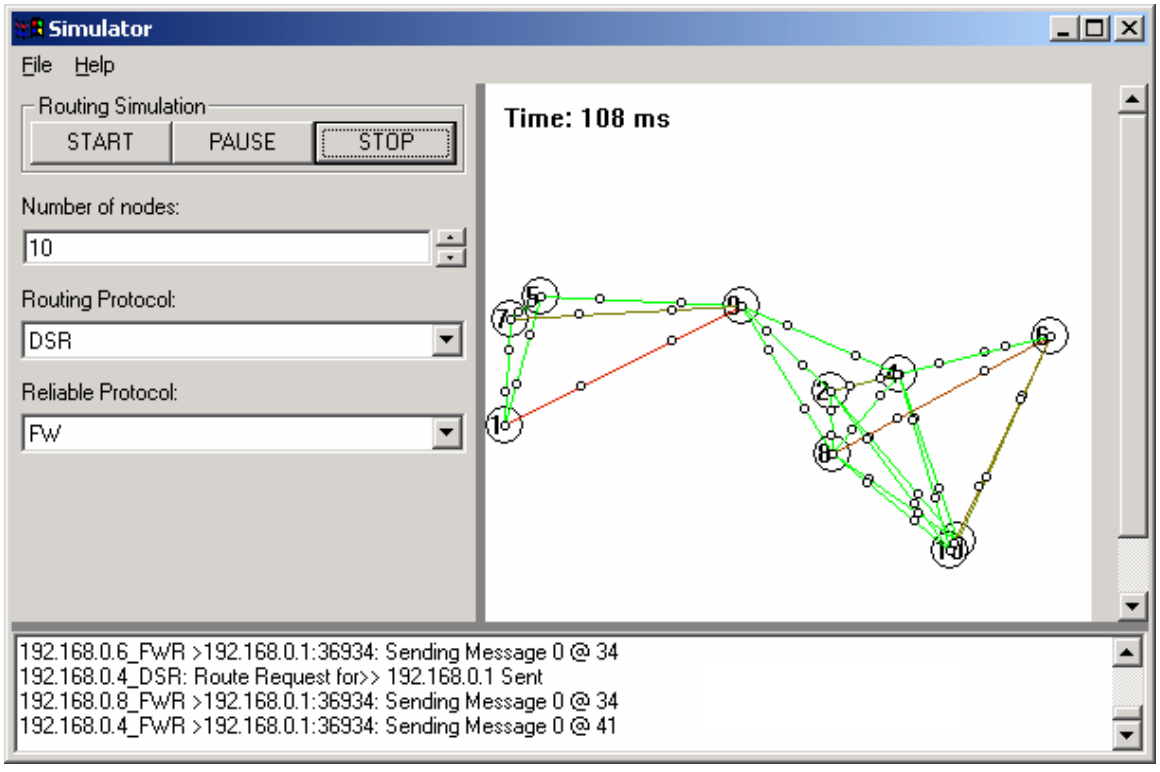

Figure 5: Simulator

Once tested with the simulator, the implementation of the ad hoc routing protocol can be deployed on a wireless network without any modification to its source code. The only requirement is to link the implementation with a routing environment such as our SOCKS proxy. The system was tested using a group of IBM Thinkpad 390X laptops equipped with Orinoco 802.11b wireless cards. The laptops were running different versions of Windows (mainly Me and 2000). The release version of the platform consisted of a single binary executable and a library (DLL) and was distributed to each of the laptops. Each node was set in ad hoc mode with one of them serving as an Internet gateway.

5. Conclusion and Future Work

In this paper, we have described a portable user-level architecture for ad hoc routing. The framework was developed with the following goals in mind: (1) Support all conventional applications; (2) Easy deployment (3) Minimal configuration, (4) Abstraction of the routing environment, and (5) Straightforward Internet connectivity. We proposed a layered framework consisting of a routing environment, routing protocol layer and a reliable message delivery layer. The routing environment abstracts lower-level details of the network and the operating system from the routing protocol layer. This allows the routing protocol to be developed independently and integrated easily.

Our current implementation of the routing environment uses a SOCKS5 interface for maximum portability. The framework currently includes implementations of the Dynamic Source Routing (DSR) and flooding, though other protocols could be easily incorporated. A simulator is provided, which allows testing the routing protocol implementation before deployment. A nice feature of our implementation is that the routing protocols require no modification when moved from the simulation environment to actual deployment.

We imagine that this architecture will be useful for several purposes: for research in designing new ad hoc routing protocols, for educational purposes, allowing students to gain familiarity with ad hoc networking protocols, and for deployment of real ad hoc networks to support emerging applications.

The architecture is still being developed. The next major undertaking is a performance study—while it is clear that the architecture works (we have spent many happy hours surfing in infrastructure-less networks), we currently have no clear picture of how efficiently network resources are used. Other ongoing work includes adding support for multicast routing protocols and IPv6 support.

References

[1] D. Johnson, D. Maltz, Y-C Hu, J. Jetcheva, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," *http://search.ietf.org/internet-drafts/draft-ietf-manet-dsr-07.txt*.

[2] C. Perkins, A. Royer, S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," *http://search.ietf.org/internet-drafts/draft-ietf-manet-aodv-10.txt*.

[3] F. Lilieblad, O. Mattsson, P. Nylund, D. Ouchterlony, A. Roxenhag, madhoc Implementation of AODV for Linux".

[4] Monarch Implementation of DSR, available at *http://www.monarch.cs.rice.edu/*.

[5] INRIA Implementation of Optimized Link State Routing, available at *http://menetou.inria.fr/olsr/*.

[6] X. Zeng, R. Bagrodia, M. Gerla, "GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks," Proceedings of the 12th Workshop on Parallel and Distributed Simulations, 1998.

[7] ns2 website at *http://www.isi.edu/nsnam/ns/*.

[8] M. Leech et al, "SOCKS Protocol Version 5", *http://www.faqs.org/rfcs/rfc1928.html*.

[9] SocksCap, *http://www.socks.nec.com/reference-/sockscap.html.*

[10] R. Caceres, L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," chapter 7, pp. 207-228, Mobile Computing, T. Imelinski and H. Korth, eds. Kluwer Academic.