A TCP/UDP PROTOCOL VISUALIZATION TOOL: VISUAL TCP/UDP ANIMATOR (VTA)[1]

Chunhua Zhao[2] *and Jean Mayo*[3]

Abstract *- The TCP/UDP/IP/Ethernet protocol suite is commonly covered within an undergraduate curriculum. Often, this is the first exposure students have to a network protocol. We have found that students have difficulty understanding the unique functions of these protocols and how the protocols are related. This is due at least in part to the fact that packet specification and construction takes place within the operating system, while covering a typical operating system protocol stack implementation would be beyond the scope or time constraints of many of these undergraduate courses. In order to address this problem, we have written a visualization tool that illustrates the operation of TCP and UDP over the IP and Ethernet protocols using real packets captured live from the network or stored in a file. The tool provides several views of captured data that, when used individually or in combination, illustrate the operation of these protocols.*

Index terms - Network protocols, visualization, socket programming

MOTIVATION

An undergraduate curriculum is likely to include coverage of the TCP/UDP/IP/Ethernet protocol suite. These protocols are studied because they are in widespread use, because coverage exposes students to the layered nature of network protocols, and because knowledge of the protocols helps students to better understand related issues in security, operating system implementation, system administration, and network programming.

In our experience, students have difficulty understanding how the TCP/UDP/IP/Ethernet protocols are related, the unique functions of each protocol, and the format and content of data packets that travel over the network. This is at least in part due to the fact that implementation of the protocol takes place within the operating system, while studying the operating system protocol stack code is beyond the scope and time constraints of many courses that might include coverage of these protocols. In order to help our students better understand the operation of these protocols, we have written a visualization tool, the Visual TCP/UDP Animator (VTA).

VTA displays packets captured from the network in any of several *views*. These include

- the *Packet List View*, that displays the contents of each field of captured packets,
- the *Connection Packet View*, which displays all packets passing over a TCP connection,
- the *Connection Reconstruction View*, which displays the application data exchanged along a TCP connection as a conversation with two participants,
- the *Topology View*, which displays an undirected graph in which nodes correspond to machines and edges correspond to the existence of a packet in which the connected machines are source and destination,
- the *Timeline View*, which displays a space-time diagram [8] between each pair of machines that communicate,and
- the *TCP Status View*, which depicts the current status of a connection within a TCP state diagram.

The tool takes input either from file or from the network. A user may specify that all data from the network be captured, that data from selected active sockets be captured, or that all data from a chosen application be captured. Regardless, an additional filter, by source or destination IP address or port, may be applied to the packet stream prior to display. VTA can save a session for later replay. No special privileges are required, and a user may see only data that passes over sockets which she owns. VTA has been designed to run over the Linux operating system.

RELATED WORK

There has been little work in the academic community on the visualization of TCP and related protocols. Barr et al. developed the XSNIFF network monitoring tool with a graphical X-Window based display [1]. XSNIFF displays statistical information (e.g. which stations are sending the most data and what types of services and protocols are being used most frequently) and a TCP connection graph.

The VINT project is developing a network simulator that will allow the study of scale and protocol interaction in current and future network protocols [6]. A network animator (Nam) has been developed that provides packet-level animation, protocol graphs, and traditional time-event plots of protocol actions [4]. The scope of VTA is much smaller and its use is correspondingly less complex than the use of Nam.

[1]Supported in part by the National Science Foundation under grant DUE-9952509 and CAREER award CCR-9984682.

[2]Michigan Technological University, Houghton, MI 49931, czhao@mtu.edu

[3]Michigan Technological University, Houghton, MI 49931, jmayo@mtu.edu

Tcpdump [9] prints out the headers of selected packets on a network interface. Many tools are available for traffic analysis. These tools include Multi Router Traffic Grapher (MRTG) [12], IPTraf [7], IP Flow Meter (IPFM), Tcptrace [13] and Ethereal [3]. Ethereal also provides the ability to reconstruct (in ASCII format) the stream of data that flows over a TCP connection. These tools, like XSNIFF, focus on analysis of the traffic flowing over a network. The goal of our tool is to illustrate execution of the TCP and UDP protocols over IP and Ethernet. This is accomplished in part by providing connection-oriented views, rather than views that attempt to analyze all data passing through a network interface.

VISUAL TCP/UDP/IP ANIMATOR

VTA displays data in any of six views. The stream of data displayed in these views begins from a user-specified source. This stream then passes through an optional filter, and then feeds into the selected views. In this section, we describe configuration of the data stream and the six available views.

FIGURE 1. DATA SPECIFICATION WITH WIZARD

Input Data Specification

When VTA is first run, the user wizard, depicted in Figure 1, guides users through the input data specification process. A user may select one from among the options *Disk File*, *All Realtime Traffic*, *Current Active Sockets*, or *Specific Application* as the initial source of the packet stream which feeds into the VTA views.

Selection of the *Disk File* option prompts the user for the name of an input file. VTA can save the data of any session, and this option can be used to replay an earlier session. File input can also be easily generated using output from *tcpdump* [9].

When the *All Realtime Traffic* option is chosen, VTA will collect all data that passes through the network interface.

When a user selects the *Current Active Sockets* option, she is presented with a list of all active sockets. She may select any number of these active sockets, and all passing through one of these sockets will be displayed. Figure 2 depicts the active socket window.

VTA provides a library containing routines that wrapper around the Linux socket system calls. These routines are designed to allow a user to capture all the data flowing to and

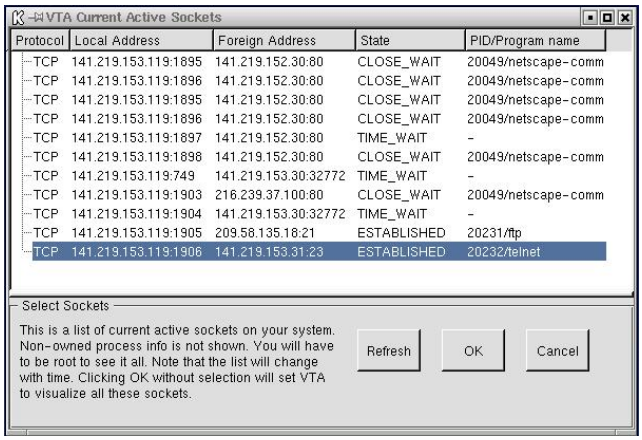

FIGURE 2. ACTIVE SOCKET SELECTION

from a particular application, without requiring that the user be aware in advance of port numbers assigned within the application. Selection of the *Specific Application* option causes VTA to inter-operate with the socket wrappers (through the `/tmp` directory). This provides students a mechanism both for understanding the operation of the socket system calls, as well as debugging their socket-based network applications.

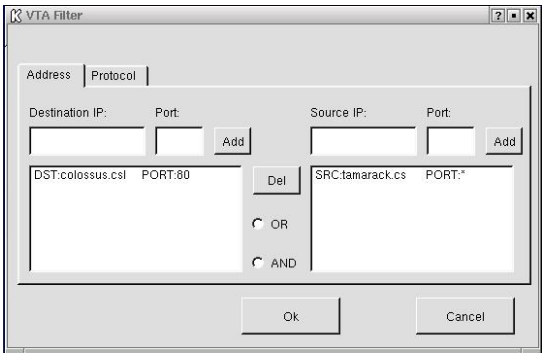

FIGURE 3. SOCKET FILTER SPECIFICATION

Whichever of these sources is selected, a subsequent filter can optionally be applied to the stream prior to its display by VTA. After selecting any of the available data sources, the filter specification window (depicted in Figure 3) is presented to the user. If no filter is applied, all collected packets will be displayed. Data can be filtered by source IP address or port, destination IP address or port, and protocol (TCP or UDP).

FIGURE 4. VTA MAIN WINDOW

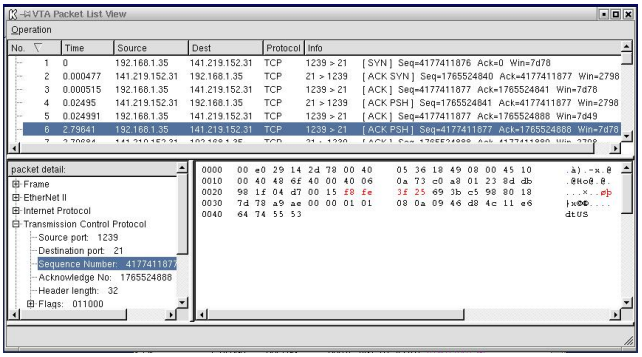

FIGURE 5. PACKETLIST VIEW

VTA Views

Once the input packet stream has been configured, any of six views can be selected for display of the stream. Capture and display begins upon selecting *Start* from the main window, shown in Figure 4, and continues until *Stop* is selected or no additional data is available. A step mode, in which a single packet is displayed for each step, is also available.

Processing of the input data stream can be stopped, and the input stream reinitialized, at any time. When a new run begins, all open views are cleared and display of new data begins.

Following is a description of the six available views.

Packetlist View

The packetlist view window is depicted in Figure 5. A summary line is displayed for each captured packet. The summary line contains:

- a packet number, indicating its position in the sequence of captured packets,
- a time value, indicating seconds that have elapsed between the capture of the first packet and the capture of the displayed packet,
- the source and destination IP addresses,
- the protocol (TCP or UDP), and
- protocol related information.

Selecting a single summary line displays the contents of the packet in two formats. The *packet detail* displays a text description of, and value for, each field of each header (Ethernet, IP, TCP/UDP) in the packet. The second depiction contains the hexadecimal representation of the packet contents. Selecting a field within the packet detail highlights the corresponding packet bytes within the hexadecimal representation.

The packet view abstracts the received data stream as a pile of packets. Data is viewed from the perspective of the interface; there is no attempt to collect packets together in any way, e.g. , packets that pass between the same two endpoints.

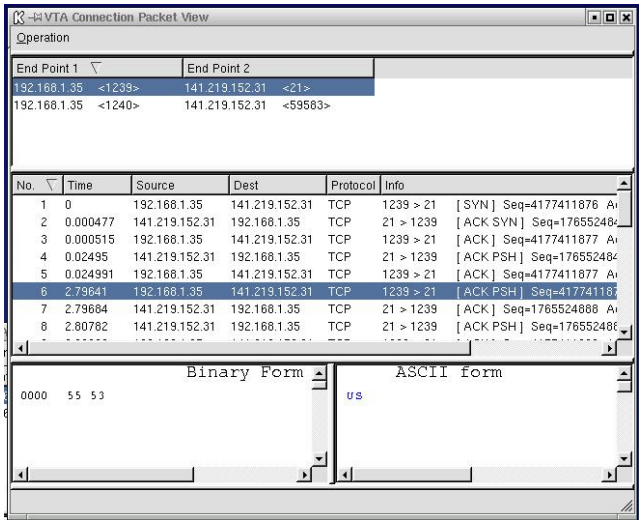

FIGURE 6. CONNECTION PACKET VIEW

Connection Packet View

The connection packet view is shown in Figure 6. A summary line appears for each unique (source, destination) pair. The summary line contains the source and destination addresses (<IP address,port>). Selecting a particular pair displays a summary line, similar to that of the packet view, for each packet that has been sent or received, by the host, along the selected path. Selecting the summary line for a particular packet displays the application data contained in that packet in binary and ASCII format.

This view separates the received packet stream into a number of smaller streams, distinguished by communication endpoints (same source <IP address,port> and destination <IP address,port>). Unlike the packet view, a user can easily trace all data exchanged along a given path, including the exchange of data required to establish and maintain a TCP connection.

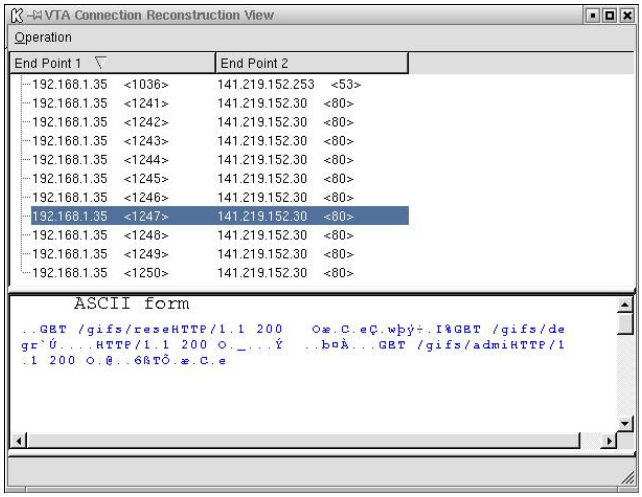

FIGURE 7. CONNECTION RECONSTRUCTION VIEW

Connection Reconstruction View

The connection reconstruction view is shown in Figure 7. This view attempts to depict data transmitted along the connection as a conversation between the communication endpoints. A summary line is displayed for each TCP connection. Selecting a single connection displays the data, in ASCII format, that has flowed across the connection. Different text colors denote the direction of the data transmission. For example, data transmitted from the VTA host to a receiver always appears in a single color that is different from the single color used to depict data received by the VTA host.

No data associated with connection maintenance is displayed in this view. It is suited to visualization of the data exchanged by protocols that sit above TCP.

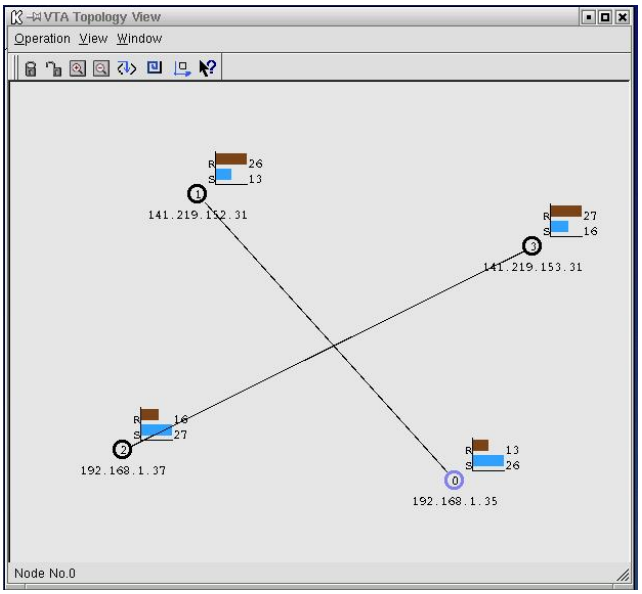

FIGURE 8. TOPOLOGY VIEW

Topology View

The topology view displays an undirected graph where edges correspond to source/destination pairs in a captured packet and nodes correspond to IP addresses. The topology view window is depicted in Figure 8. For each node, an IP address and number of packets sent and received is displayed.

In order to display the network topology, an automatic layout algorithm based on a spring-embedder model is used [5]. Attractive forces are assigned on all links and repulsive forces are assigned between nodes. Iteration is used in an attempt to achieve balance. This technique can produce reasonable layouts of many networks, but may not produce satisfactory results of complicated networks. As a remedy, VTA allows the user to adjust the resulting layout by moving nodes within the window.

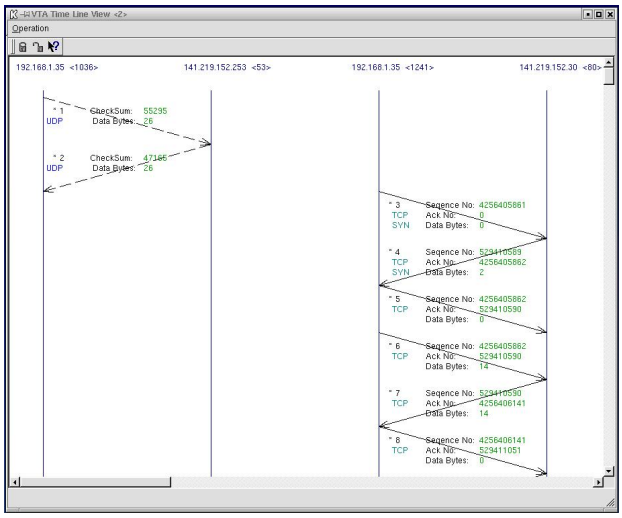

FIGURE 9. TIMELINE VIEW

Timeline View

The timeline view is depicted in Figure 9. In the timeline view, an axis appears for each new socket (<IP,port> pair). Each sent or received packet results in an arrow between the axes corresponding to the source and destination. Both UDP and TCP communications are displayed. (If the transmission is based on UDP, the arrow appears dashed; if the transmission is based on TCP the arrow appears solid.)

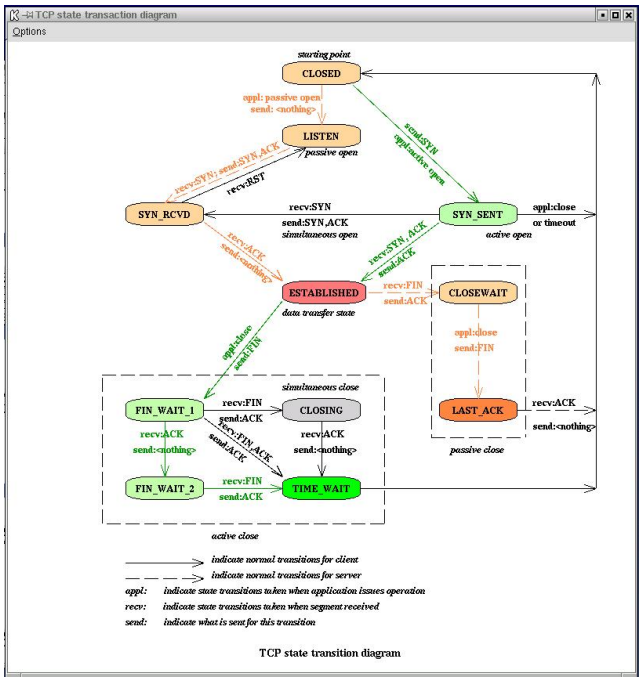

FIGURE 10. TCP STATUS VIEW

TCP Status View

The TCP Status view is shown in Figure 10. This view depicts the state of a TCP connection within the protocol state transition diagram [15]. Different colors, yellow or green, mark the state in which the two connection endpoints currently reside. A third color marks states through which the connection has passed.

IMPLEMENTATION

To capture packets transferred by a network, a capture application needs to interact directly with the network hardware. VTA operates by putting the Ethernet network interface card into promiscuous mode so that every packet going across the network is captured. The underlying operating system must provide support so that these frames can be captured.

Providing access to the datalink layer for an application is a powerful feature that is available with most current operating systems. VTA uses *libpcap* [9], a publicly available packet capture library. This library provides VTA with a powerful high-level capture interface. VTA does not interact with the hardware directly, but uses the functions exported by *libpcap* to capture packets, set packet filters, and communicate with the network adapter.

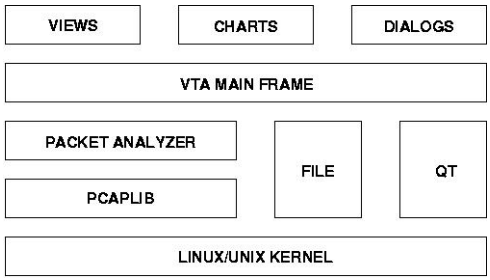

FIGURE 11. SYSTEM ARCHITECTURE

Figure 11 shows the layered system architecture of VTA. The lowest level is the datalink layer access primitive that is provided by the operating system. It supplies the application a set of functions used to read and write data from the network at the datalink layer. The VTA main frame takes the raw packet traffic from the network, encapsulates the packets into a sequence of packets, and distributes the packets to various views for display.

The graphical user interface manages the interaction with the user and displays the capture results. It is constructed with *Qt*[16], which is a C++ GUI software toolkit. *Qt* was selected because it is highly portable, being currently supported on Microsoft Windows 95/98/2000, Microsoft Windows NT, Linux, Solaris, HP-UX, Digital UNIX (OSF/1, Tru64), AIX and others.

Figure 12 shows the packet flow among different layers of the VTA system. The raw packets are captured from the Ethernet adapter. Then the packets are filtered by the libpcap library. After that, the packet traffic specified by the user is

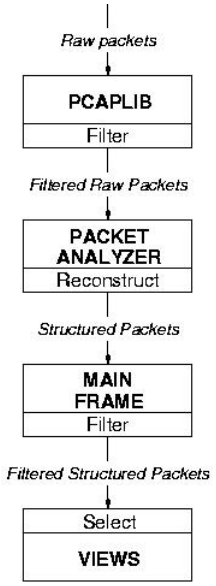

FIGURE 12. DATA FLOW ARCHITECTURE

reconstructed into a sequence of packets by the packet analyzer and delivered to the main frame, which then distributes the packets to different views specified by the user.

Unprivileged Access

VTA provides for access by unprivileged users (those who are not *root*). Any ordinary user can run VTA, and she will have access only to data transmitted through a socket which she owns.

This is accomplished through use of a small, root-owned setuid (SUID) program and the Linux `/proc` file system. Within UNIX, a process has two userids (UIDs): a *real* UID and an *effective* UID. The *effective* UID is used to determine whether or not a process has permission to access a resource. Ordinarily, the real and effective UIDs are those corresponding to the user executing a program. When a program is SUID, the effective UID is set to the owner of the program, rather than the UID that corresponds to the user executing the program.

A root-owned SUID program performs all necessary privileged operations for capturing data from the network. A check is made to ensure the socket associated with a given packet, to be displayed by VTA, is owned by the real userid of the VTA process before the data is given to VTA for display.

The use of SUID programs provides the potential for buffer overflow exploits. In order to address this concern, the code which runs with an effective userid of root has been minimized, and made available for (further) examination prior to use. Still, system administrators may not be comfortable adding an SUID binary onto their system. In this case, VTA may be used within a laboratory which securely provides students root access. (Examples of such laboratories, designed to allow students root access, are given by Mayo and Kearns [10], Schafer et al. [14], and Clark [2])

Packets captured live from the network are processed by VTA periodically, to apply the user-specified filter and to restrict displayed data to that passing through a socket owned by the user running VTA. When VTA is not run by root and the socket wrapper library is not used, certain (UDP) packets may be dropped. This arises when ownership of these packets cannot be ascertained through the `/proc` file system.

CONCLUSIONS AND FUTURE WORK

The developed visualization tool illustrates the operation of TCP and UDP over the IP and Ethernet protocols using packets captured live from the network or stored in a file. Several views of captured data are provided that, when used individually or in combination, illustrate the operation of these protocols.

Privileged access is not required. When the supplied socket function wrapper library is used, all packets that pass over a connection are captured. When the library is not used, capture of all data is ensured only if VTA is run by the root user.

The NIST Net Network Emulator [11] provides a mechanism for examination of TCP (and UDP) responses to various network conditions. NIST Net is a general purpose tool for emulating performance dynamics in IP networks. NIST Net can emulate end-to-end performance characteristics imposed by various wide area network situations or by various underlying subnetwork technologies.

Based on our use of the tool, several additional features are planned. Currently, VTA captures data from a single machine. While this is adequate for illustrating the characteristics of TCP and UDP, we feel that distributing VTA (automatically), so that data is captured at both ends of a connection, will enhance its operation. Additionally, we are developing program pre-processor directives to support two new operation modes. The first mode will allow monitoring of only user-specified socket routines. For example, a user may then only see traffic generated by one, among several, send operations contained in her source code. The second mode will allow a user to step through the socket routines contained in her code, viewing the packets generated by the execution of each socket routine. Finally, we are planning to allow application of a user-specified format to application data prior to its display. This will allow a user to view the packet's application in a user-prescribed format, rather than only ASCII or binary, as in the current version of VTA.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the work of Ping Chen, Rong Ge, Yin Ma, and Yuping Zhang in development of a preliminary version of VTA. Their efforts made this work possible.

References

[1] Bob Barr, Sung Yoo, and Tom Cheatham. Network monitoring system design. In *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, pages 102–106, Atlanta, Georgia, Feburary 26 – March 1 1998.

[2] P.C. Clark. Supporting the education of information assurance with a laboratory environment. In *Proceedings of the Fifth National Colloquium for Information Systems Security Education*, May 2001.

[3] Gerald Combs. The ethereal network analyzer. `http://www.ethereal.com/`, 2001.

[4] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with nam, the vint network animator. *IEEE Computer*, 33(11):63–68, 2000.

[5] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164, 1991.

[6] Ahmed Helmy and Satish Kulmar. Vint project. `http://www.isi.edu/nsnam/vint/`, 1997.

[7] Gerard Paul Java. Iptraf - an ip network monitor. `http://cebu.mozcom.com/riker/iptraf/`, 2001.

[8] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[9] Lawrence Berkeley National Laboratory Network Research Group. Lbnl's network research group. `http://www-nrg.ee.lbl.gov/`, 2001.

[10] J. Mayo and P. Kearns. A secure unrestricted advanced systems laboratory. In *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, pages 165–169, March 1999.

[11] National Institute of Standards and Technology Internetworking Group. Nist net home page. `http://snad.ncsl.nist.gov/itg/nistnet/`, 2001.

[12] Tobias Oetiker and Dave Rand. Mrtg: The multi router traffic grapher. `http://www.mrtg.org/`, 2001.

[13] Shawn Ostermann. Tcptrace - official homepage. `http://www.tcptrace.org/`, 2001.

[14] J. H. Schafer, D. J. Ragsdale, J. R. Surdu, and Jr. C.A. Carver. The iwar range: A laboratory for undergraduate information assurance education. *The Journal of Computing in Small Colleges*, 16(1):223–232, 2001.

[15] W. Richard Stevens. *Unix Network Programming: Networking APIs: Sockets and XTI*, volume 1. Prentice Hall, second edition, 1998.

[16] Trolltech AS. Trolltech. `http://www.trolltech.com/`, 2001.