Technical Report

Department of Computer Science
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 97-008

Creating a Virtual Network Laboratory

by: Yen-Jen Lee, Wei-hsiu Ma, David H.C. Du, and James A. Schnepf

Creating a Virtual Network Laboratory

Yen-Jen Lee, Wei-hsiu Ma, and **David H.C. Du**[1]
Distributed Multimedia Research Center & Department of Computer Science
University of Minnesota
Minneapolis, Minnesota 55455
{ylee,wma,du}@cs.umn.edu

James A. Schnepf
Department of Computer Science
College of St. Benedict/St. John's University
Collegeville, Minnesota 56321
jschnepf@cs.csbsju.edu

Abstract

Networking technologies have entered an unprecedented era after the explosive growth of the Internet and the roll-out of high speed networks. This paper addresses the concept of using existing multimedia and computer networking technologies to create a remotely accessible, virtual network laboratory that can expand student access and eliminate many of the time, geographical, and cost constraints that currently exist. The authors propose a framework for constructing lab modules for a virtual network laboratory. A prototype has been developed for a series of Java-based modules that allow students to access and interact with the virtual laboratory databases and physical networking devices in a user-friendly manner. It provides a demonstration of networking concepts by using the developed materials in new courses at each of the participating universities.

Keywords: Collaboratory, Virtual Laboratory, Internet, WWW, Java, Network Analyzer, Ethernet, ATM, HIPPI, Fibre Channel

[1]This work is partially supported by the University of Minnesota-IBM Shared Research Project and NSF Grant CDA-9502979.

1 Introduction

As modern society moves into the information age, electronic communication has taken on increased importance in many facets of life. The number of people and machines connected to the global Internet continues to grow exponentially. The government has made the building of the National Information Infrastructure (NII) one of its top priorities as we move into the 21st century. The technology underlying network communications is changing rapidly. After twenty years of relatively stable communications technology, there has been a jump in new protocols and technologies to support data rates in the gigabit per second range over wide geographical areas.

There is increasing recognition of the need to train a more technically qualified workforce with an understanding of the concepts of computer networking and how to effectively apply concepts learned in the classroom to real world problems. In addition, many professionals currently in the networking field are overwhelmed by the changes, and also need further education and training to maintain their mastery of this area.

Science and engineering department have long recognized the need for laboratory experience for their students; it is through these experiences that students deepen their understanding of the conceptual material presented in the classroom. In contrast, networking courses are traditionally conceptual in nature with little opportunity for students to apply what they have learned in order to strengthen their knowledge. This situation does not provide for an adequate understanding of the concepts and the practical applications of those concepts.

To address these urgent needs, and to overcome the shortcomings of current approaches, we are developing a virtual network laboratory environment. The kernel will consist of a collection of multimedia learning modules which can be executed on most workstations and PCs. The modules are designed to allow students carry out hands-on networking labs in a virtual environment. This virtual laboratory will be easily accessible over the Internet.

Individually, several universities such as the University of Minnesota have made efforts to provide laboratory education regarding networking. Three years ago, the University of Minnesota (joined with North Hennepin Community College) established a Bachelors of Information Networking (BIN) based on extensive discussions with industry on what they perceived as needs for future graduates. The program provides students with a fundamental understanding of networking concepts and of distributive programming. This foundation is built on by providing more course work on the implementation and management of these networks and courses that provide exposure and understanding of cutting edge technologies currently being researched that will become the network technologies of the future.

To support these courses, a "hands-on" network lab was established at North Hennepin Community College with some industrial donations and more than a $100,000 budget. However, because of the interconnectivity and interactions, this lab can only support a limited number of students and it is difficult for students outside of the BIN program to access the equipment.

The rapid changes in information technologies challenge educational institutions to change the way in which they carry out their educational responsibilities and the number of students they can reach. The issues involved can be summarized as follows:

- Programs need to bridge the theory and practice of networking technology and provide students

"hands-on" experience with computer network environments.

- These programs need to reach more students with a cost effective delivery that can leverage expensive equipment. Physical laboratory space and communications equipment are expensive to maintain and operate. This limits the number of students that can be provided with a hands-on experience.
- Networking and communications technologies are changing rapidly. To prevent student knowledge from being obsolete as soon as they walk out the door, these programs must include access to state-of-the-art communications equipment and technologies. Access to hardware and software of emerging technologies must be made "available" to students *before* they reach the marketplace.
- These programs must break the geographical and time constraints of traditional educational programs. Ideally, students should be allowed to access course materials and communications equipment at different times and from different places.

Other areas of science and engineering deciplines are also undertaking this route to address electronic educational components, grant scale collaborative work and sharing of unique scientific instrumentation. The *Mocha* model [1] provides algorithm animation using WWW and Java [2] to assist the algorithm designers and students to understand algorithms by visually following their step-by-step execution and demonstration. Mocha is very close to our goal in promoting the use of distributed interactive platform-neutral multimedia applications over the Web for education. Collaboratory [3] coins the term by a prototype implementation that provides a loosely integrated set of Internet capabilities that appear as extensions to the Web to start or join multitool collaborative sessions. Madefast [4] is an early example of collaborative work by defense contractors using WWW. They have been developing Internet-based tools, services, protocols, and design methodologies that will allow contractors to compose teams of specialists from different locations and organizations as project needs arise, and to achive results.

The paper is organized as follows. Section 2 describes the framework to construct virtual network laboratory functional modules. In section 3 and 4, we discuss the system design of a functional prototype with two components: a protocol analyzer and a performance analyzer, respectively. They fit in the NetSniff and NetLoad modules described later. Section 5 outlines the implementation and operation of the lab module prototypes. We conclude the paper and describe programs which put this work into practice in section 6.

2 Framework

To support the development of the virtual lab environment, we proposed a three-stage appraoch to the design and implementation. In each of these stages, we will develop a collection of multimedia learning modules using Java [2], an object-oriented programming language. We will center (but not limit) these modules on the series of instructional lab exercises we have developed for the B.I.N, program. These labs exercises include:

- Apple Computers with Appletalk
- Networking UNIX Workstations with TCP/IP over Ethernet

- Configuring DNS on Unix Workstations
- Networking WindowsNT and Novell PC's over Ethernet
- Interconnecting Heterogeneous Networks
- Configuring and Using Dedicated Routers
- Trouble Shooting Network Problems
- Network Management
- Measuring Network Performance
- Setting Up a Firewall

A key element in the lab agenda is that, almost every lab exercise uses a network analyzer to tap in the network to watch protocol activities and performance. In addition, a few labs offer first-hand installation and configuration experiences to students who have never had a chance to work with dedicated internetworking products, e.g. Lab 6 to configure and use a router. These capabilities need to be supported in our virtual laboratory as well.

2.1 Approach

The stages of development are as follows:

1. Establish an environment where students can remotely access equipment in the lab and carry out experiments without having to physically be located in the lab. This will potentially include limited access to high speed networking devices that students would rarely have an opportunity to access otherwise. For instance, Figure 1 depicts a partial view of the network infrastructure support in the Department of Computer Science at University of Minnesota (the computing nodes are also interconnected by Ethernet not shown). This also includes a collaborative effort with industrial partners to allow students access to the state-of-the-art equipment that educational institutions cannot afford but are available at the industrial sites. This access will occur over the ubiquitous Internet.
2. Develop a set of virtual laboratory experiments that can be performed at sites independent of the network hardware location. These will be developed and carried out by:
   - Physical performing, in the lab, variations of an experiment and capturing the resulting output.
   - Developing interactive multimedia modules that allows the student to establish varying levels of network and device connectivity, and configure network software similarly to what would be done in the physical laboratory.
   - Allowing students to connect to a multimedia server to perform the experiments using a rich set of parameters based upon the actual performance of experiments performed in the laboratory.

3. Based upon the framework that has been developed for part two, develop a set of experiments based on new and developing technologies.

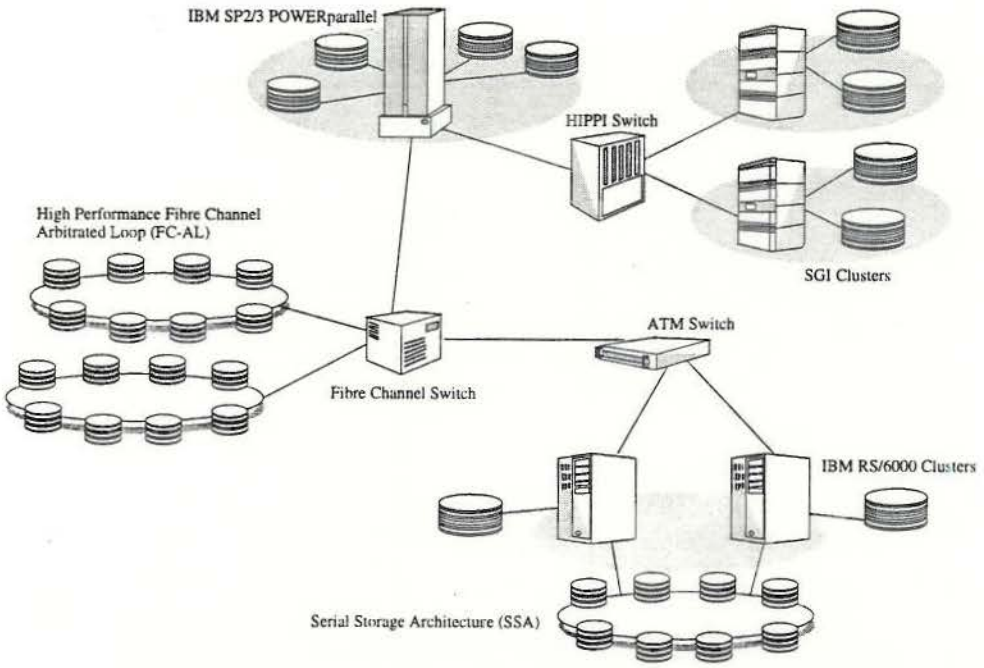

Figure 1: A partial view of the network infrastructure support in the Department of Computer Science at University of Minnesota, Minneapolis campus.

Students will perfom the experiments via a collection of multimedia learning modules. These modules will be developed in a platform-neutral approach in a networked environment. The seamless integration of WWW and Java enables a platform-neutral implementation of the system modules. Remote invocation of the laboratory modules is transparent to end users.

The following modules are planned for development:

- a NetSniffer module that supports a variety of networking standards (including: Shared Ethernet, Switched Ethernet [5], Token Ring, FDDI, ATM [6], HIPPI [7], and Fibre Channel [8, 9]),
- a NetConfig module that allows a network design to be configured and tested in the virtual lab, and
- a NetLoad module that allows captured network traffic patterns to be run on the system designs created by NetConfig.

Existing network monitoring and management tools are typically software driven, with interactions and results made possible through a traditional computer keyboard and monitor interface. With virtual laboratories, we can provide "unlimited" access to "simulated" equipment that is directly under the control of the students using widely available personal computer technology. The student might not even be aware that real hardware is not being used.

It is intended that this access will provide students with the full range of interactions that would be available in the laboratory in real-time with an interface that is similar in functionality to the real interfaces in the lab. It is our hope that by making this flexible environment widely accessible, we can attract students who are traditionally left out of many science and mathematics projects.

Emphasis on access in the third stage will be to provide students with experiences using experimental technologies where access to hardware is not yet feasible. Over the past five years there have been many exciting developments in high speed networks including the development of ATM, HIPPI and Fibre Channel. While these new technologies are likely to come into widespread use in the next few years, computer science students have little, if any, exposure to them. Even those who have some exposure to the concepts, do not have the opportunity to apply those concepts. It is difficult for academic institutions to invest in equipment of new and unproven technologies and incorporate them into their curriculum.

2.2 Module Design using Java

The module design consists of three correlated components: client runtime, server runtime, and physical networking device. The key design principles are ease-of-use and a platform-neutral approach. As shown in Figure 2 using Java for the design framework, client runtime consists of: a graphical user/command *front-end* interface to assimilate the look-and-feel of a physical device, a *back-end* access object to simulate the state and command control of the physical device, and a *communication entity* for the back-end to retrieve pre-orchestrated simulation sequence or communicate with a physical device through a proxy service.

The server runtime provides services to multiple clients concurrently. It uses a *pseudo device* object to respond to clients' back-end access requests for simulated sequences; or, it may access a physical device if the network access mechanism is available through a *proxy agent*. The server's communication entity has dual purposes as it communicates with client and physical devices if applicable.

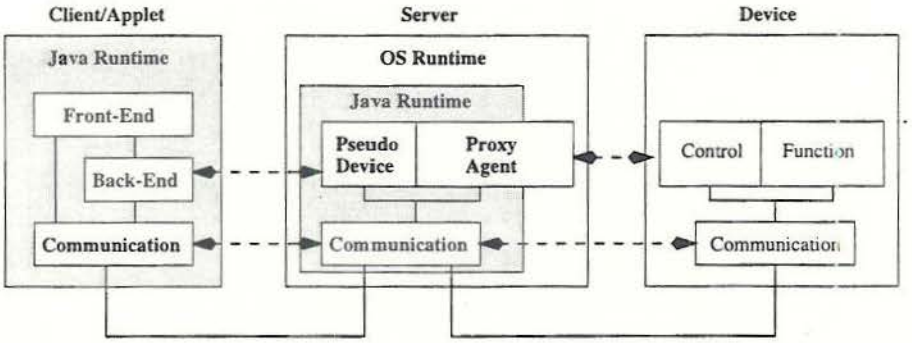

Figure 2: Framework for Virtual Network Lab Module.

Front-End.

The front-end includes a platform-neutral user interface and device-independent primitives which provide general control of executable content in Web page format or as a stand-alone application front-end interface.

Back-End.

The back-end access object coexists with the front-end interface on the client platform to simulate device-dependent portion of the lab, including state transition and operation of the device and protocol decomposition. Generally, a state machine can be formulated for a given physical network device based on its response to user command control. A simulation engine implements the state machine. In response to the user input, the simulation engine generates requests to the server and feeds back responses to the front-end.

Pseudo Device.

Pseudo devices on the server provides the client with simulated sequences and engineering data through databases of experimental traces or synthetic results. A device can be identified by its manufacturer and function. It is the client's responsibility to keep track of the state changes. Pseudo devices simply supply the client's simulation engine with the requested dataset for specific device ID.

Proxy Agent.

Proxy agents bridge the gap between the client and the physical networking device. It provides the client an abstraction to access proprietary management facilities pertaining to the physical networking device. It also has the capability to reorganize the response from physical networking device into presentable information for the client.

Communication Entity.

Communication entities provide remote data access abstraction for the simulation engine to access the backing store (for instance, trace data or proxy service). They also provide server access to physical device using device-specific management protocol.

Physical Networking Device.

Physical networking devices are the target of the concern in the lab experiment. A device generally has integrated control and functional blocks. However, a device may or may not have network capability to respond to queries directly from client back-end access methods. The server runtime provides a proxy agent to address the reachability issue and also monitors the access to the device.

The coherent design of client runtime and server runtime for lab modules promotes common reusable objects and standard implementation paradigm without adhering to a specific hardware platform. Development work can be done on a variety of platforms, and tests can be conducted over the Internet and Web following this design methodology. Users of the developed modules will perceive a working environment as it is physically present.

3 Protocol Analyzer

A protocol analyzer such as Sniffer [10] (which is, in general, a network analyzer with built-in expert system) or Packetman [11] is an indispensable tool to capture and diagnose the bits and bytes going across the network. The analyzer serves several purposes which are directly applicable to a production network for network application developers and operators. First, communication protocol headers are decomposed into individual fields with high-level annotations. Users can conduct a careful inspection and verification of the protocol to understand the meanings of headers and identify implementation flaws. Second, the analyzer catches the timing of protocol interaction. A replay on the captured data based on the timing information truthfully reveals the sequence of protocol operations. Finally, the analyzer can display captured data in different formats and provide mappings among them. Captured data can be stored for later use.

The Java module to simulate the protocol analyzer uses captured data from a network analyzer. The module tailors down the functionality of a protocol analyzer for lab use. The design of a few critical components is as follows.

3.1 Front-End: Protocol Data Display

The front-end interface displays captured data in three views: frame summary, frame detail, and hexadecimal data. These three views mimic the data display in a Sniffer protocol analyzing device. The user controls the focal point by placing the mouse focus to the subwindow of a given view rather than keyboard tabbing done with a Sniffer. The movement can be linear by skimming through the entries in a subwindow, or non-linear by selecting the desired entry. Linear movement resembles sliding a viewing window across a dataset. Non-linear movement causes jumping from one segment to another segment of the dataset.

3.2 Back-End

The back-end simulation engine implements a state machine as shown in Figure 3 (the events to trigger state transition are not explicitly shown). After initialization (*power cycle*), the state changes to *operation selection*. The user has to capture data going across network first (*data collection*), and then view the data in the *data display*.

Under data display, there are three internal states depicted in Figure 4. Each entry in the frame summary subwindow collectively describes the meaning of a frame transferred over the network medium, including timing, source/destination addresses, etc. When frame summary is the active state, the other two subwindows, which provide details for a frame in distinct formats, are in lock-step synchronization frame-by-frame with the frame summary. The frame detail and hex data views have additional lock-step synchronization requirement field by field. This provides a mapping between annotated field data and associated binary information transmitted in the network.

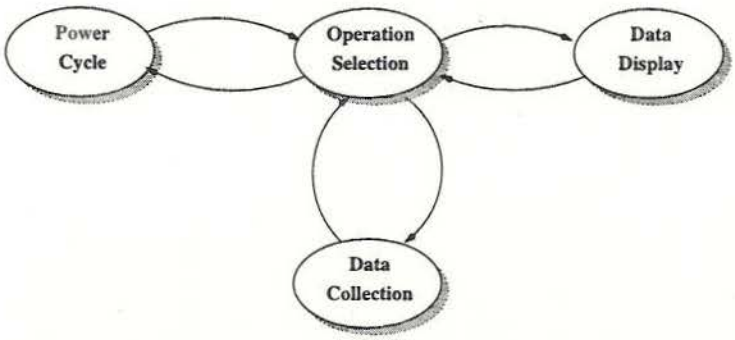

Figure 3: Finite state machine for protocol analyzer.

3.3 Communication Entities

The throughput and variation of network communication over the Internet is unpredictable. Smooth user-perceived operation requires adequate bandwidth and buffer space to retrieve results from server to client. We used a Sniffer and a stop watch to measure how fast that the Sniffer can inactivate an entry in the frame summary and activate the next one (which will be highlighted) by pressing the arrow-down key to move across a range of entries over 10 seconds. Since detail view and hex view provide further details for a frame, the information has to be available before it is being displayed as the user switched to another entry in frame summary. The average moving speed is about 14 frames/sec with a minimum and maximum of 13.8 frames/sec and 14.2 frames/sec. In addition, Sniffer annotates the captured data frame and generates human readable information which is 2 to 3 times of the amount of data contained in a frame. For a one shot test where most of the frames are NFS traffic, average frame information size is around 4.4 KB (833752 bytes/188 frames). The result suggests a measure of speed in human-computer interaction that would ideally be duplicated when using a protocol analyzer module in a virtual environment. Under most stringent requirement, it takes about 500 Kbps channel capacity to retrieve captured data over the network.

In addition to the bandwidth requirement, adequate buffer space is necessary to overcome communication latency for on-the-fly viewing if the dataset can't fit in client's buffer space. While looking at a data frame for a given transmission medium such as Ethernet, the next move of a user is either viewing following frames or viewing previous frames. Jumping to a given frame is similar to re-establishing a new working set of frames for sequential movement. Figure 5 shows the buffer requirements on the client and server. The server may be serving multiple clients with different working sets from the same dataset.

On the other hand, the overhead imposed by Java running on low-end PCs may adversely affect the perceived interactivity and performance. Suppose the rate to inspect frame by frame in the front-end interface of the protocol analyzer is R_f and average amount of data for a frame record is D_f, then the user-perceived display bandwidth is $B_f = D_f \times R_f$. If $B_f \leq B_n$ (data retrieval bandwidth), the continuity of display depends on the overhead in performing memory I/O and display mapping in Java.

If $B_f > B_n$, the amount of memory buffer governs the interactivity. Assume the total amount of a

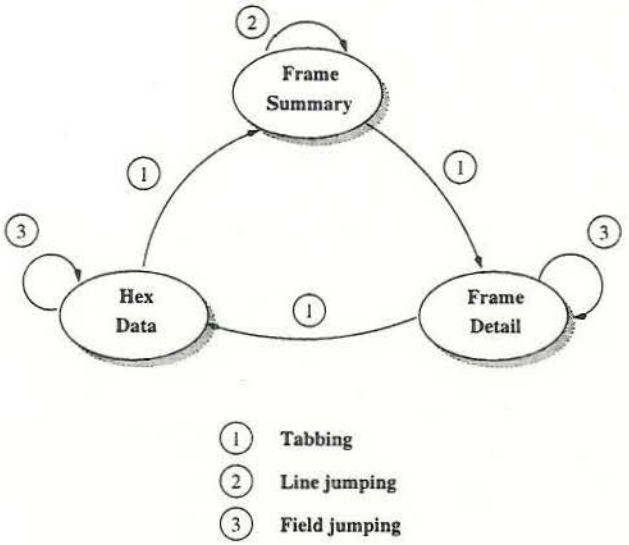

Figure 4: Finite state machine for data display in protocol analyzer.

particular data set is D_T, and the buffer size is D_B which is populated before accepting user command. The minimum pre-populated buffer size to maintain user-perceived continuity is derived as the following:

$$\begin{aligned} \frac{D_T - D_B}{B_n} &\leq \frac{D_T}{B_f} \\ (\textit{Time to retrieve the rest of the data} &\leq \textit{Time for user to reach the end of the data}) \end{aligned}$$

Hence,

$$D_B = (1 - \frac{B_n}{B_f})D_T \quad (1)$$

Coupled with the discussion on network bandwidth earlier, we can determine the client runtime buffer size as well as server's according to the simplified analytical formulation and design criteria.

4 Performance Analyzer

The objective of the performance analyzer is to provide hands-on experience on the performance evaluation over the networks. Performance is a major issue of networks from Ethernet to high speed networks, such as ATM. However, the resources and equipment in a network lab are limited and only a few students can access the network links and test them at the same time. The performance analyzer in the virtual lab can help students access a virtual network remotely to analyze its performance. The virtual network can be a real network link or a data set including pre-stored experimental performance

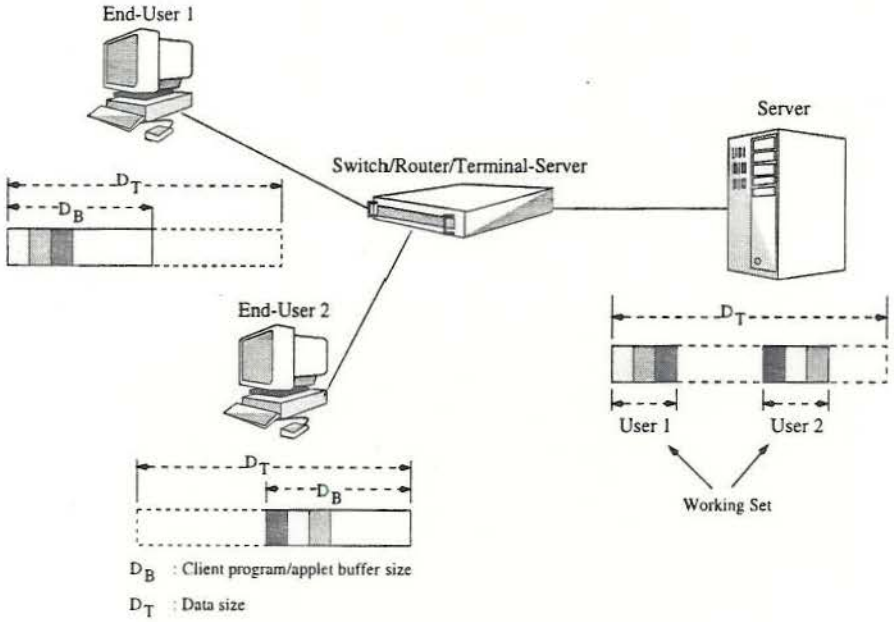

Figure 5: Client-server buffer requirements for protocol analyzer.

results of the network. The students can access different networks with various protocols or traffic patterns in a virtual environment.

Different protocols and networks, have different characteristics. Some benchmark programs, such as Netperf [12], can be used to measure the network throughput under multiple protocols. Netperf provides tests for TCP streams, UDP streams, Fore ATM API streams, etc. The features of different protocols have different effects on the performance of the same network. Some protocols are reliable and others are not. They may use connection-oriented or connectionless services. Packet sizes and protocol overheads are typically different. Hence, the analyzer has to provide the performance outputs with respect to the characteristics of the protocols and their performance data. The parameters for protocols can be adjusted such that students can recognize the effects due to the parameter changes. They can also learn how to tune a network system to get the best throughput.

Additionally, different applications and scenarios generate different traffic patterns. Students can use the analyzer to monitor the effects on performance when they observe traffic patterns of different applications, such as file transfer, short message passing, or multimedia data delivery. Students can use this module to analyze the performance of specific traffic patterns over different protocols or networks. For example, the students may find which protocol is better suited to file transfer. If the file size is variant, different results may occur. The analyzer also provides different probability models, e.g., Poisson distribution, for the traffic patterns so that the students can further explore theoretical concepts.

The critical components of the performance analyzer are described as follows.

4.1 Front-End : Performance Visualization

The performance analyzer provides the user interface to change the parameters according to the protocol and network the students want to learn. The interface of the analyzer helps students visualize the performance outcomes. It is possible to display these outcomes when the performance is evaluated in real-time. Different protocols may need different layouts to illustrate the behaviors which are the major reasons of the performance differences.

4.2 Back-End

Figure 6 illustrates the state machine of the back-end in the performance analyzer. The initial state is protocol selection which waits for the user's selection of a connection-oriented or connectionless protocols. If the student picks a connection-oriented protocol, the back-end first enters the connection setup state and asks the server to setup a connection. The state machine then transfers to the performance measurement state. During this state, the performance evaluation is executed. The student can stop the lab temporarily or reset the lab for another test. For connection-oriented protocols, the back-end can keep the connection in the connection holding state.

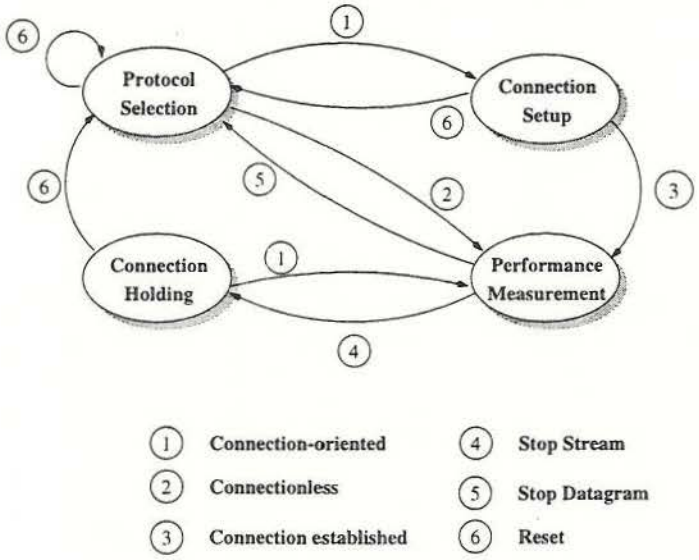

Figure 6: Finite state machine for performance analyzer.

4.3 Communication Entities

The communication entities in the performance analyzer are designed to send and receive information during the performance measurement. Each runtime program treats its entity as a channel to communicate with the other runtime. The entities may need to support multiple connections and handle the requests from different locations.

There are three communication entities on the client, server, and device, respectively. Control and data message passing occur between them. The client applet connects with the server and sends control

messages to request different operations. The communication entity on the server sends control signals to and retrieves performance results from the pseudo device or the entity on the real device. The entity on the server then reports the results to the entity on the client. The messages must be designed to achieve the needs of these communication entities. We designed an application level protocol for the entities of the analyzer. Table 1 lists the categories of the messages, but the list is not exhaustive and can be extended. The performance messages includes latency, throughput and packet status. The control messages control the execution of the measurement. Some parameters can be set according to the parameter messages.

Category	Item
Performance	Latency Throughput Packet Status
Control	Start Stop Reset
Parameter	Protocol Traffic Pattern Message Size Loop Number

Table 1: Message Category

There are two models, the push model and the pull model, for the communication between the client and the server. The push model requires the server to continually send updated results to the client at default time intervals. For the pull model, the client sends a request to server and retrieves information. Due to the high propagation delay of the Internet, the performance analyzer uses the push model to provide best-effort transmission for real-time data without waiting for the request from the client. Even for the pre-stored performance data, the push model can provide the closest data update as on the server side. The student can ask for the pull model to adjust the rate of update by the client applet.

The performance analyzer allows multiple students to retrieve the performance data in the virtual network environment at the same time. The communication entity on the server maintains multiple connections with the clients. Since each client may have a different requests for measurements, the server has to maintain a data structure for each client to record its requests and current status. The items of the structure are listed in Table 2. The status of the client is also maintained on the server such that the server is aware of the current state of the state machine in the client back-end. The connection setup time is stored to help the server check if this connection is holding for too long. The server may force a connection time-out and disconnect the communication. The list also includes the requested protocol, traffic pattern and transmission model.

5 Prototype Implementation

We have developed two functional components according to the framework and design described earlier. One is a simple protocol analyzer to address the NetSniff module. The other is the performance

Item	Possible Value
Client Status	Connection Setup Communication Holding Performance Measurement
Connection Setup Time	Hour:Minute:Second
Protocol Selection	TCP UDP ATM
Traffic Pattern	Poisson Distribution Best Effort
Transmission Model	Push Model Pull Model

Table 2: Data Structure Items for Each Client

analyzer which provides real-time network throughput and latency measurement for the NetLoad module.

5.1 Protocol Analyzer

The Protocol Analyzer implements a protocol data display which is self-executable in a web page with a Java-enabled browser or appletviewer. The appearance of the browser component is depicted in Figure 7. There is a stand-alone server application written in Java to feed simulated sequences to the client. A genuine protocol analyzer *listens* to network activity, but does not generate any traffic on that network. If the network monitored is a shared Ethernet segment, it is put in a promiscuous mode to pick up any, or selected, conversations. For instance, the lab regarding AppleTalk would focus on AppleTalk frames only. The operation of a protocol analyzer has distinct phases of data capture and data display. Hence, the network traffic is captured and stored beforehand rather than displayed on-the-fly. The simulation engine will make a transition from a *fake* capture to a *real* protocol data display.

In Figure 7, there are three text display subwindows. The upper subwindow displays the frame summary. The particular format we have adopted shows sequence number, time difference between frames, destination or source address (either resolved by name or in digits), the application layer protocol, orientation (request or response), and application message summary. The middle and lower subwindows are in sync with upper subwindow and provide the details for the highlighted frame.

The prototype is flexible enough to display different protocol suites as long as they are in a layered structure. Existing protocol families are. The simulation engine in the applet is able to interpret header formats if a protocol family is the focus of the lab exercise. This capability provides a mapping between field information in the detail view and binary information in the hex view when the focus of user interaction is in the middle subwindow.

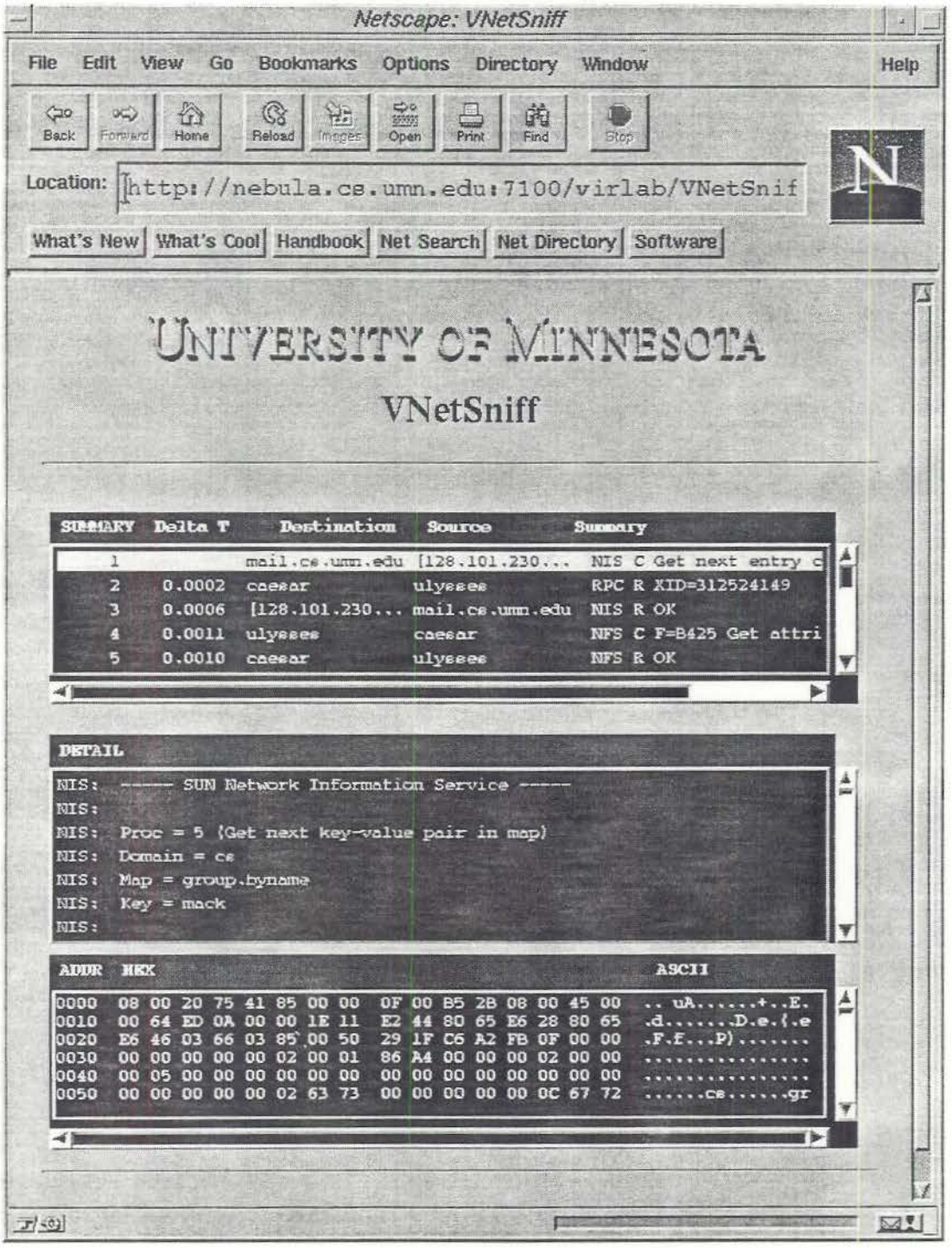

Figure 7: Protocol analyzer screen layout for data display.

5.2 Performance Analyzer

The current implementation of the performance analyzer provides enables monitoring of real-time performance for TCP and UDP protocols. For the TCP protocol, the analyzer measures the round-trip latency over the target network between two machines and calculates its throughput according to the TCP message size. The result is reported and visualized on the user site. The analyzer uses the same mechanism for the UDP protocol. However, in addition to the performance report, the receipt or loss of each UDP packet is also shown to the students.

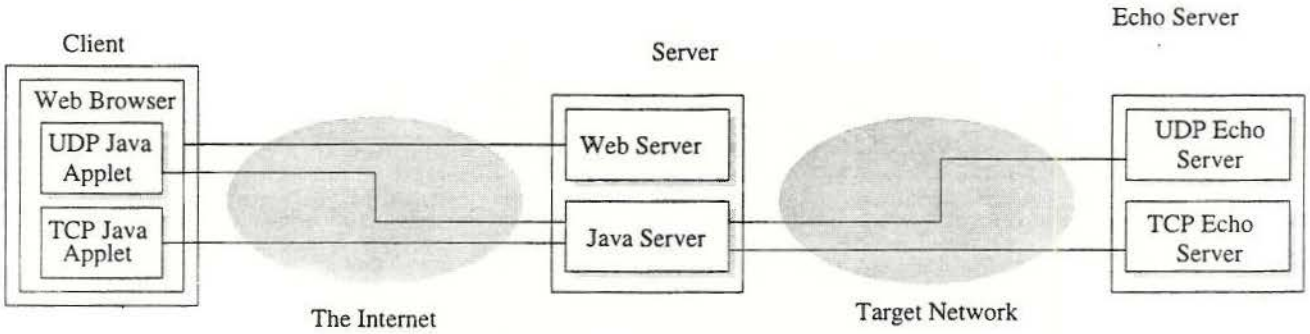

Figure 8: Performance analyzer architecture.

Figure 8 depicts the software architecture of the implementation on the client, server, and the echo server machines. The client executes the virtual lab Java applet which communicates with the web server and Java server. The web server provides the web page and applet downloading to the client. The client applet also exchanges some information with the Java server and updates the current status of the performance evaluation. The real performance test happens between the server and the echo server. The two servers are connected by the target network we want to observe. According to the protocol chosen on the client, the Java server connects with the TCP or UDP echo server.

The analyzer provides the performance data for different message sizes of the TCP protocol. The outcomes of the network throughput between the Java server and the echo server are transmitted to the TCP Java applet. The screen layout for TCP is illustrated in Figure 9. Five different message sizes are evaluated. The Java server sends messages of different sizes in round-robin fashion and gets them back from the echo server. The Java applet retrieves the results of the round-trip latency and throughput of each message and updates the statistic bars on the screen. The height of the bars change dynamically to reflect the current amount of the performance outcomes. The maximal, minimal and average results are also included. The text area in the bottom shows the received message. The start, stop and reset buttons control the state of the back-end in the analyzer.

Figure 10 demonstrates the screen capture for the UDP protocol. Because UDP is an unreliable datagram protocol, some packets may be lost during the transmission. The upper part of the UDP Java applet interface indicates which packets are lost and which packets are transmitted successfully. Each bar represents a UDP packet. The status(ready, sent or received) is shown by a different color and is dynamically changed according to it current status. The underlying mechanism and other layouts are similar to those for TCP

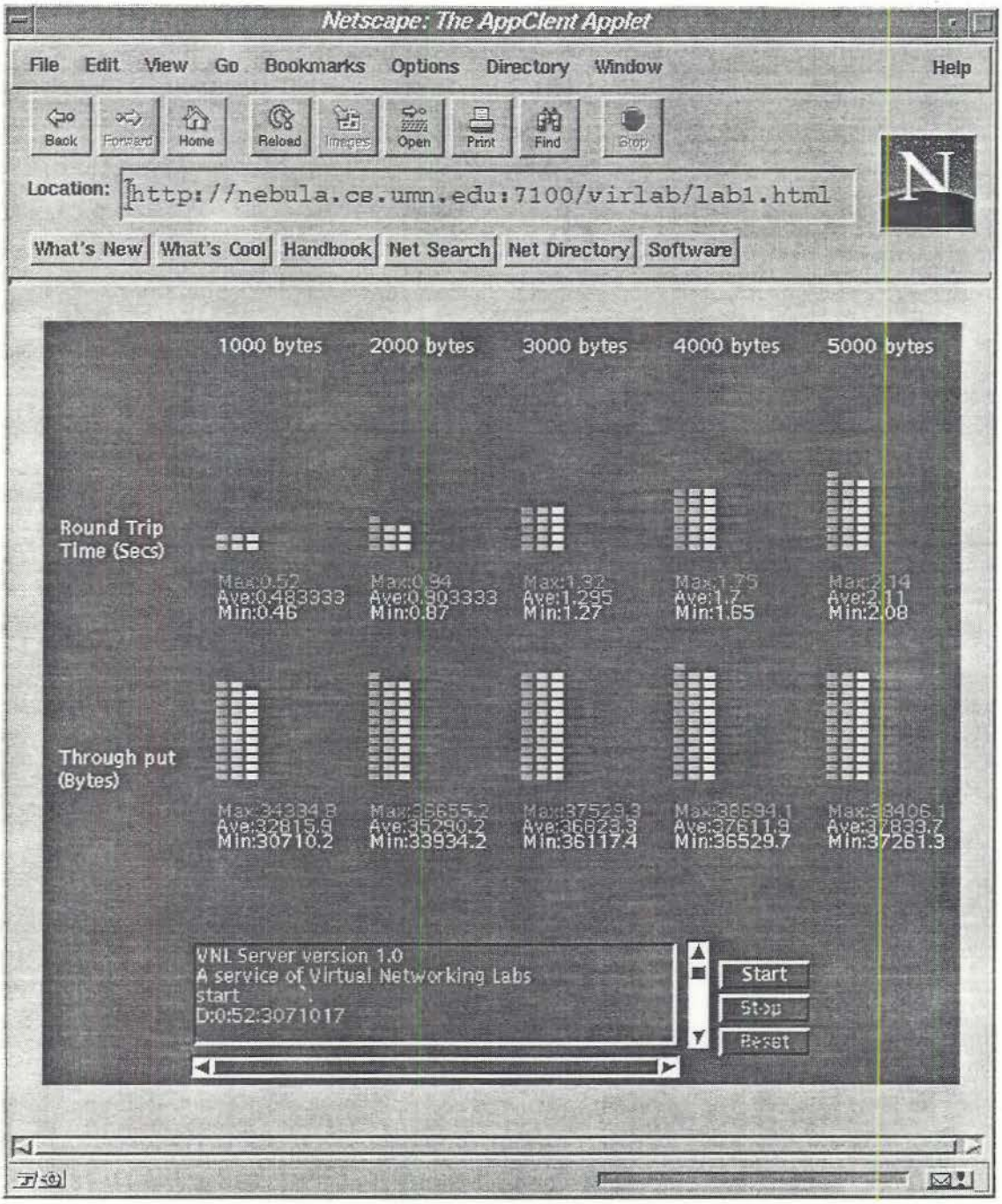

Figure 9: Performance analyzer screen layout for TCP.

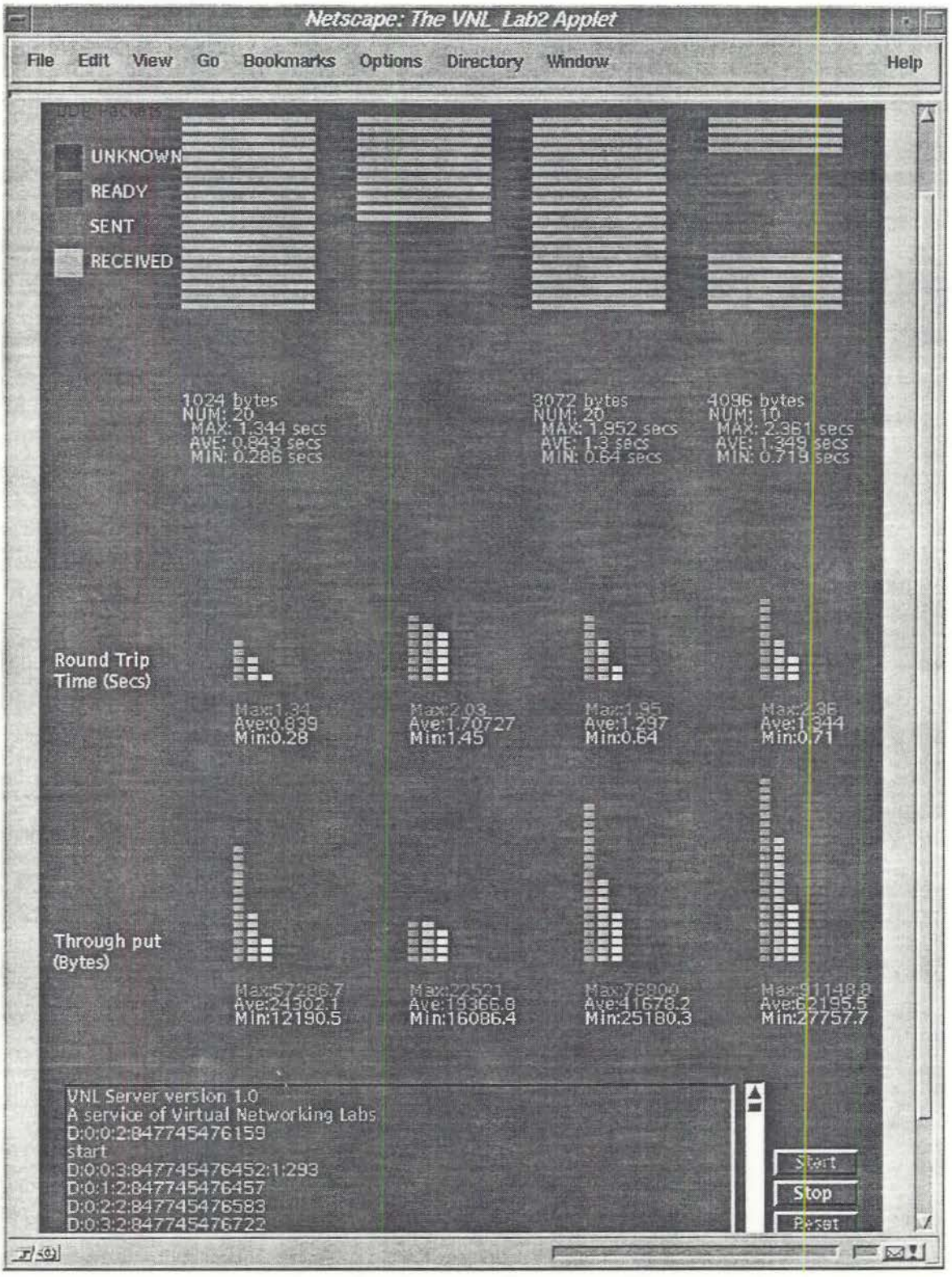

Figure 10: Performance analyzer screen layout for UDP.

6 Conclusion

The need for a virtual network laboratory bridging from concept to application goes far beyond the scope of specialized networking programs. It leverages the high cost of establishing, maintaining, and operating a state-of-art networking lab. With rapidly changing technology, it becomes crucial that faster research to curriculum is achieved. We are using the research facilities at our institutions and the facilities of our industrial partners to design and build modules that include new and developing network technologies. Access to the existing and new networking technologies over the Internet using WWW and Java provides one of the most cost effective methods to maintain and improve our workforce. This work presents an educational prototype to analyze computer communication protocols and visualize network performance. The developed modules will be introduced to the BIN program first in winter 1997 time frame. Future work shall extend the existing base and provide a remote configuration facility with similar design framework.

ACKNOWLEDGMENT

Kung-Feng Chen helped on the implementation of performance analyzer while completing his undergraduate study in Computer Science at University of Minnesota. The systems staff, including James MacDonald, Mike McShane, Andrew Nelson, and Irene Prigge, in our Department provide excellent support to the BIN lab. The presentation of a few diagrams is not possible without the art work contribution from Jenwei Hsieh.

References

[1] James E. Baker, Isabel F. Cruz, Giuseppe Liotta, and Roberto Tamassia. A New Model for Algorithm Animation Over the WWW. *ACM Computing Surveys*, 27(4):568–572, December 1995.

[2] Marc A. Hamilton. Java and the Shift to Net-Centric Computing. *IEEE Computer*, pages 31–39, August 1996.

[3] Richard T. Kouzes, James D. Myers, and William A. Wulf. Collaboratories: Doing Science on the Internet. *IEEE Computer*, pages 40–46, August 1996.

[4] Mark R. Cutkosky, Jay M. Tenenbaum, and Jay Glicksman. Madefast: Collaborative Engineering over the Internet. *Communications of the ACM*, 39(9):78–87, September 1996.

[5] Mart Molle and Greg Watson. 100Base-T/IEEE 802.12/Packet Switching. *IEEE Communications Magazine*, 34(8):64–73, August 1996.

[6] Ronald J. Vetter. ATM Concepts, Architectures, and Protocols. *Communications of the ACM*, 38(2):30–38, February 1995.

[7] Don Tolmie and John Renwick. HIPPI: Smplicity Yields Success. *IEEE Network*, pages 28–32, January 1993.

[8] Clint Jurgens. Fbre Channel: A Connection to the Future. *IEEE Computer*, pages 88–90, August 1995.

[9] Martin W. Sachs and Anujan Varma. Fibre Channel and Related Standards. *IEEE Communications Magazine*, 34(8):40–50, August 1996.

[10] Network General Corporation. *Expert Sniffer Network Analyzer Operations*. Network General Corporation, 1992.

[11] Bradley Williamson. *Packetman v1.2 User Manual.* School of Computing, Curtin University of Technology, Perth, Western Australia, 1995. Available via ftp://ftp.cs.curtin.edu.au/pub/netman/.

[12] Hewlett-Packard Company. *Netperf: A Network Performance Benchmark, Revision 2.1.* Information Networks Division, Hewlett-Packard Company, 1996. Available via http://www.cup.hp.com/netperf/NetperfPage.html.