The Virtual Network System

Martin Casado
Department of Computer Science
Stanford University
Stanford, CA 94305-9030
casado@cs.stanford.edu

Nick McKeown
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-9030
nickm@stanford.edu

ABSTRACT

The goal of our work is to give students a hands-on experience designing, deploying and debugging parts of the Internet infrastructure, such as an Internet router that routes real network traffic, or a security firewall. To do so normally requires that the students have access to snoop and generate raw network traffic, which is a risk to privacy and security. And it normally requires each student to have a dedicated computer, and to modify the kernel. The Virtual Network System (VNS) is a teaching tool designed for undergraduate and graduate networking courses. With VNS, each student can build a router (or any packet-processing device) in user-space, in their own private, protected topology, and process real Internet traffic. VNS has been used by over 500 students at Stanford and remotely from other universities. This paper describes the VNS tool, and our experiences using it in the classroom.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Measurement, Design, Experimentation

Keywords

Network Simulation, TCP, Internet Infrastructure, Router Design, VNS

1. BACKGROUND

The typical undergraduate introductory networking class includes programming assignments. A canonical first assignment gives students experience with API-level sockets programming by implementing a simple web-client or ftp-client. Further assignments typically build on the sockets layer, and the students implement remote procedure calls, network games, distributed file systems and other applications. Yet the sockets layer is quite a high level abstraction of the underlying network, and sits on top of TCP, IP and the link-layer. Projects at this level don't provide students with experience in packet re-transmission, congestion control, routing nor other concepts key to an introductory class.

To provide hands-on access with lower-level networking concepts, the computer science education community has developed a number of project environments to supplement introductory networking courses [6, 7, 15, 8, 11, 16, 18, 10]. These reflect an oft-cited need for active learning environments in networking courses. Active learning in networks gives students experience in the subtleties of the design of a complex system, as well as prepare them for the networking industry.

These environments are typically variants of the following:

- Infrastructure for implementation of transport level protocols over a connection-less protocol such as UDP [15, 8].
- Virtual, simulation or emulation environments for developing simplified link layer or network layer protocols [11, 9, 6, 2, 3].
- Administration and/or kernel level development of network topologies of dedicated hardware or networked virtual machines [14, 16, 18, 12, 17].
- Simplified access to link-layer network traffic for analysis [10]

.

At the introductory level however there is a lack of teaching tools that allow students to develop and deploy Internet infrastructure components such as routers that interact with actual hardware on the Internet. Projects at the network level and below are typically taught theoretically or using special-purpose simulation environments. Simulation environments often require the students to learn and use non-standard scripting interfaces and libraries [9, 2] and are not designed to provide real time access to the Internet. Larger computer science programs may have labs with dedicated hardware in which students can develop network functionality at the kernel level [14, 17]. However, such environments require extensive resources to setup and and manage and are not suitable for introductory courses where the students may not have sufficient systems development expertise.

Giving students experience below the sockets layer with live traffic is quite difficult[5]. At the machine level, this

SIGCSE'05, February 23–27, 2005, St. Louis, Missouri, USA.

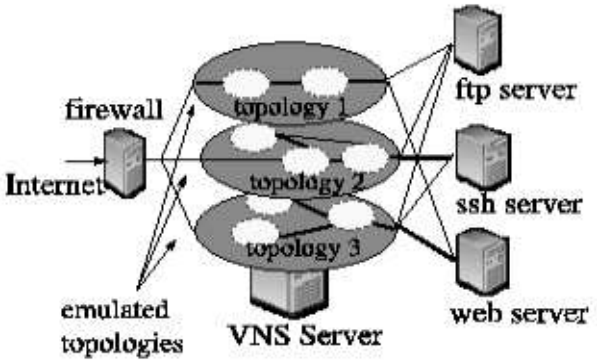

Figure 1: A simple VNS server setup with three virtual topologies and three application servers

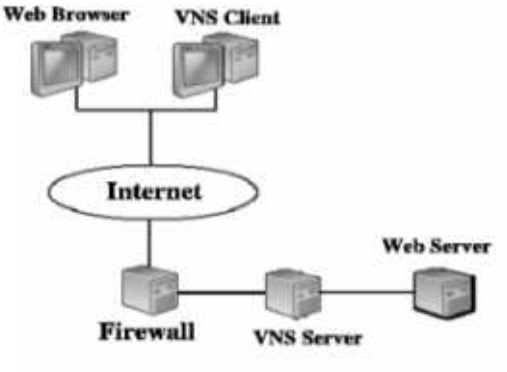

Figure 2: Web browser accessing a web server connected to VNS

usually requires modifying the kernel, which requires a computer per student, and a highly unstable development environment. It also requires great care in the security of the network. Most universities don't allow students to snoop or generate arbitrary raw IP packets on the campus network. And even if it was possible for a student to implement a router, it would be cumbersome to build a realistic topology to test one or more instances.

In this paper we present an educational tool we developed to support projects below the socket layer safely and using few resources. The Virtual Network System (VNS) was designed to allow hundreds of students working remotely (anywhere on the Internet) to develop user space programs that function as network infrastructure components, which operate on the actual Internet handling live, real-time traffic. We have developed a number of programming assignments using VNS that are suitable for undergraduate and graduate level courses. All course materials are publicly available. Fundamental to the design of VNS is the ability to support multiple remote classes working on different projects simultaneously. VNS is a production system currently being used in a number of courses locally and remotely. It has been used in undergraduate and graduate networking courses here at Stanford and at Johns Hopkins to teach the implementation of routers and to support the development of functional TCP compatible stacks that interact with other hosts on the Internet.

2. THE VIRTUAL NETWORK SYSTEM

The core of VNS is a server daemon written in standard C++ that can simulate multiple, virtual network topologies from a single host as shown in figure 1. Each virtual topology is reachable from the Internet and can integrate with actual hardware devices such as commodity PCs running common Internet services. The VNS server daemon can support hundreds of users and topologies from a single machine.

The VNS server accepts connections from VNS clients which are user space programs that will operate as hosts on the virtual topologies. Clients upon connecting can request that the server forward them traffic as seen by a particular host on a given topology. The client may then inspect, discard, modify or inject new traffic into the network, giving it the full capabilities of an actual host on the network.

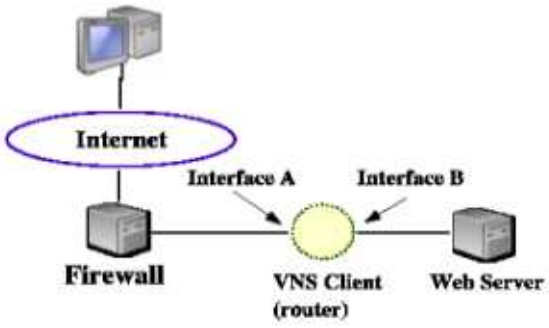

Figure 3: View of virtual topology as seen by web browser

We describe the operation of VNS using a simple example. Figure 2 shows a physical network topology in which a web browser application is trying to access a web server. The VNS server provides a virtual topology such that the web browser sees the network topology shown in figure 3.

In effect, the VNS server inserts the VNS client (the student's code) into the topology, even though the VNS client is physically located somewhere else on the internet (Figure 2). The VNS server achieves this by receiving all packets destined for interfaces A and B and passing them via a separate socket to the VNS client. The VNS client receives each packet exactly as the router would have received it (as raw Ethernet packets) and then determines what action, if any, to take. For example, when the VNS client receives an http (web) request packet on interface A, it will send the packet back to the VNS server with instructions for the VNS server to forward the packet out of interface B to the web server.

Security considerations regarding student access to low-level network functionality are addressed in VNS by placing a firewall with a stringent set of rules between the server and the Internet. The firewall rules only pass valid TCP and UDP packets without forged source addresses (egress filtering). Gateway based rate limiting makes it hard to implement client-initiated DOS style traffic patterns (e.g. SYN floods; although we've yet to experience this type of behaviour).

Projects using VNS typically involve writing a client to perform a specific function on the network. We've devel-

oped simple libraries in C, C++ and Java to serve as the base for clients. These libraries provide low level primitives for interaction with the server such as reserving a host, reading interface information about a particular host, reading a packet from the network and injecting a packet back into the network. All communications between the server and clients are done via a proprietary protocol over a TCP connection. Using a simple, user-level library to handle low level packet access has the advantages of simplicity, portability and simplified debugging. The libraries have the ability to create *pcap* [4] compatible packet trace files of all packets sent to or received by the client for analysis standard tools such as *tcpdump* [4] or *ethereal* [1].

3. REPRESENTATIVE PROJECTS

VNS is a good candidate environment for any project requiring low-level network access. While initially designed to aid students in developing network infrastructure components it is equally useful for demonstrating network concepts such as TCP characteristics using real Internet traffic. In this section we describe two assignments designed around VNS and our current effort to create a visual demonstration tool for use in class lectures and student lab environments.

3.1 Building an Internet Router with VNS

Here at Stanford VNS is used in the introductory networking course to teach the implementation of Internet routers. The assignment is structured around a simple topology consisting of a router with three interfaces connected to two application servers running standard Internet services such as http and ftp. Each student is assigned their own private topology and an IP range to allocate to the router's interfaces. Using the VNS client library as a base, the students develop fully functional routers. The routers must support the following basic functionality:

- ARP query/response capability
- ARP cache with timeout
- IP header checksum calculation
- TTL decrement on forwarded packets
- generate ICMP TTL exceeded when necessary
- NEXT hop lookup in forwarding table
- response to ICMP echo request
- generate ICMP Port Unreach for TCP/UDP packets destined to a local interface

A completed router will be able to route traffic from anywhere on the Internet to the connected Internet servers and will respond correctly to ping and traceroute. During development a student can test his or her router by using a standard web or ftp client directed at one of the application servers connected to their topology. Ambitious students may wish to add support for advanced functionality on their routers. Students have extended their routers to support NAT, web servers and IP filtering.

3.2 Implementing a Full TCP/IP Network Stack

VNS is used by students in our introductory networking course to develop a TCP compliant transport layer called STCP. While STCP can inter-operate with any standard TCP implementation it consists of a subset of TCP functionality leaving out more complex features such as congestion control. Students begin the project by implementing STCP over UDP, using UDP for the network layer. They manage both sides of the TCP connection. When they believe they have working implementations, they migrate their transport code to IP using VNS and test their implementations against standard TCP stacks anywhere on the Internet. The VNS portion of the assignment is done using a simple one host, one interface topology. Testing is done using an ftp client they develop in their direct programming assignment. Armed with an application and a transport layer, they can transfer files from any ftp-server in the Internet.

There are a number of advantages to using an IP based, TCP compatible transport layer. Students get to experience the subtleties of inter-operating with commodity TCP stacks. Exposure to the IP layer is representative of real world transport layer implementations which for example may have to handle checksumming over the IP pseudo-header. Students can extend their code to support standard TCP features such as slow-start, dynamic window-resizing, and the Karn-Partridge algorithm.

3.3 Clack : A GUI Router Builder

We also use VNS in the classroom to graphically demonstrate network concepts. To this end, we developed Clack, a Click-inspired, GUI VNS client written in Java [13]. Clack is a graphical, component-based router builder that can be used to demonstrate various router aspects using live Internet traffic. It currently contains a working subset of Click's basic ipv4 router components and supports ARP handling, IP header checks, IP forwarding, IP TTL decrement and basic ICMP handling and generation. Clack connects to the VNS server just like any other VNS client and can route real network traffic.

Clack is designed to visually illustrate network infrastructure concepts. For example, we show students in real-time how the buffer in a router evolves with network congestion; how performance changes if packets are dropped randomly instead of from the tail of a queue; and what happens if the router mis-sequences the packets. The client contains many user-configurable components, such as a FIFO queue component with configurable size and drop policy that displays its occupancy in real time. The queue also has a graphing feature which plots the absolute and average queue occupancy over a period of time. This can be used to show the TCP slow-start algorithm, or the saw-tooth of a TCP flow in congestion avoidance.

Development on Clack is ongoing. We plan to develop a set of loadable router configurations that can be used for demonstrations during lectures and by students in a lab environment.

4. EXPERIENCES WITH VNS IN THE CLASSROOM

VNS has become an integral part of two annual courses at Stanford and is used remotely by an upper level IP protocols course at Johns Hopkins University. The current con-

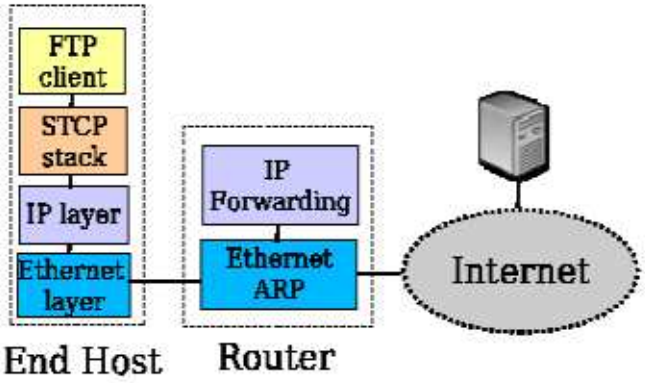

Figure 4: Full network stack and router developed by students in our introductory network course using VNS

figuration has handled tens of students simultaneously and hundreds of students in a quarter.

With VNS we have been able to organize the projects in our introductory networking course so that by the end of the quarter the students have developed a full networking stack that can inter-operate with live hosts on the Internet. The final project for the course requires the students to piece together all of their previously completed programming projects: Their ftp client application operates over their TCP transport layer, from which packets are forwarded through their own routers to connect to the Internet. The VNS topology and components of the final project are shown in figure 4.

Feedback from the students has been very positive. While most admit that the projects are a lot of work, implementations typically range from 1,000 - 2,000 lines of code per project, many find tremendous value in working directly with real Internet traffic using standard protocols. In the words of one student responding to a question on an anonymous, formal evaluation

> "Yes, it greatly helped my understanding. I feel that I could go work commercially on routers now."

We've had numerous requests from students after the quarter is over to allow them to use VNS to experiment with ideas of their own.

We have found that students' experiences are greatly affected by the transparency of the system. That is, a student must have access to all possible paths that traffic might take in order to debug their implementations. Providing *pcap*-compatible dump files is very useful to the students and we are continuing to explore other methods for providing debug access to VNS at runtime.

5. CONCLUSIONS AND FUTURE WORK

VNS gives students the opportunity to process real Internet traffic in user-space from anywhere in the Internet. Our implementation of the VNS server is robust, and can support hundreds or thousands of students from remote schools, with each student having their own private topology.

With support from the National Science Foundation, we are prototyping a national center for internet infrastructure teaching. Our lab setup has the necessary infrastructure, including global IP space, network and server capacity to support thousands of students simultaneously. The idea is to offer full support, curriculum and supplemental source code to institutions interested in using VNS. Technical details and complete assignment materials are posted on the project website at http://yuba.stanford.edu/vns. We are seeking to expand the number of courses that use VNS in remote colleges and universities. We encourage any faculty considering using the project to contact us directly by e-mail.

6. ACKNOWLEDGMENTS

This work was funded by the NSF, grant 02-082. We would like to thank Andreas Terzis of Johns Hopkins for providing very helpful feedback and suggestions, Greg Watson for general guru-ness and Guido Apenzeller and Vikram Vijayaraghavan for help in designing, developing and debugging the next generation virtual router system.

7. REFERENCES

[1] The ethereal network analyzer. http://www.ethereal.com.

[2] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[3] Opnet network simulator. http://www.opnet.com.

[4] tcpdump network sniffer. http://www.tcpdump.org.

[5] S. Akhtar, N. Al-Holou, M. Fienup, G. T. Finley, R. S. Roos, and S. Tannouri. The networks course: Old problems, new solutions. *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 360 – 361, 1999.

[6] I. B. Lewis Barnett. An ethernet performance simulator for undergraduate networking. *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer science education*, pages 145–150, 1993.

[7] N. W. Brad Richards. Illustrating networking concepts with wireless handheld devices. *Proceedings of the 7th annual conference on Innovation and technology in computer science educatio*, pages 29–33, 2002.

[8] P. Dinda. The minet tcp/ip stack. *Northwestern University Department of Computer Science Technical Report NWU-CS-02-08*, January 2002.

[9] R. D. Enrico Carniani. The netwire emulator : a tool for teaching and understanding networks. *ACM SIGCSE Bulletin*, 33(3):153–156, 2001.

[10] M. J. Jipping, A. Bugaj, L. Mihalkova, and D. E. Porter. Using java to teach networking concepts with a programmable network sniffer. *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 120 – 124, 2003.

[11] L. E. Joseph D. Touch, Yu-Shun Wang. Virtual internets for lab and class experiments. *ISI-TR-563*, August 2002.

[12] B. Kneale and L. Box. A virtual learning environment for real-world networking. *Information Science*, page 71, June 2003.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM*

Transactions on Computer Systems, 18(3):263 – 297, August 2000.

[14] J. M. Mayo and P. Kearns. A secure unrestricted advanced systems laboratory. *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 165–169, 1999.

[15] B. Richards. Rtp: A transport layer implementation project. *Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 134 – 141, 2001.

[16] J. Rickman, M. McDonald, G. McDonald, and P. Heeler. Enhancing the computer networking curriculim. *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 157–160, 2001.

[17] P. Steenkiste. A network project course based on network processors. *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 262 – 266, 2003.

[18] S. Yoo and S. Hovis. Remote access internetworking laboratory. *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 311–314, 2004.