Environments for a Networking Laboratory

Morgan Conbere and Michael Erlinger
Computer Science
Harvey Mudd College
Claremont, CA USA
{mconbere,mike}@cs.hmc.edu

Renzo Davoli
Computer Science
University of Bologna
Bologna, Italy
renzo@cs.unibo.it

Michael Goldweber
Computer Science
Xavier University
Cincinnati, OH USA
mikeyg@cs.xu.edu

Abstract

TinkerNet, TinkerNet2, and VDE (Virtual Distributed Ethernet) were all developed as low-cost platforms for teaching bottom-up, hands-on networking at the undergraduate level. In the original TinkerNet "throw away" PCs, cheap components, and free software were used to create a networking laboratory.

TinkerNet was created as a software-only system based on User-Mode Linux (UML). VDE provides a distributed virtual local area Ethernet network based on virtual switches and virtual cables. These environments enable students either to modify an existing network stack or to build their own network stack from the data-link/Ethernet layer up to the application level. This paper discusses all three environments (their design, development, and availability); and our continuing efforts to make the environments and accompanying lab exercises available for classroom use.

Keywords: Networks, Networking education

1 Introduction

As computing grows and matures, we are continually adding layers of abstraction and encapsulation to make our day-to-day usage and programming tasks easier. While abstracting away the growing complexity of a modern computer is a necessary part of computing today, it is occasionally both useful and important to be able to pull back those interfaces and see the actual workings of the systems. Students should understand the workings of some of our more complicated systems. There is a history of labs and programming environments that allow just that (Comer 2002). These systems remove any unnecessary complexity and leave exposed the features most important for students to gain all-important hands-on understanding.

TinkerNet, TinkerNet2, and VDE each provide an environment for student experience in understanding low-level networking. These systems provide direct access to Ethernet packets, and give students the features necessary to implement a network stack from the data link layer all the way up to the application layer. By giving students a hands-on understanding of how the protocols hidden away by the now-universal Berkeley Socket API behave, students will not only have a better grasp of the workings of a network, but perhaps even have a better understanding of the proper usage of sockets.

The 2002 SIGCOMM Workshop on Educational Challenges for Computer Networking (Kurose 2002) exposed many issues related to teaching computer networking: top-down versus bottom-up; one course versus many courses; required course versus elective course; and undergraduate versus graduate. Throughout the workshop discussions one recurring theme emerged: the need for a laboratory to augment lecture. While the principles of networking could be presented in lectures, the group recognized that real understanding occurs when students actively develop and evaluate systems based on those principles -- there is no good substitute for hands-on experience with real networks (ACM 1991). All of the discussed laboratory environments shared two common issues: initial cost of the laboratory and continued cost of maintenance. TinkerNet and VDE mitigate these issues and present novel and powerful environments for teaching undergraduates about the details of networking and network protocols.

The rest of this paper is organized as follows. Section 2 gives a description of the original TinkerNet. Section 3 presents an overview of TinkerNet2, which is followed by a VDE discussion in Section 4. Section 5 presents an prototype set of laboratory exercises which have been designed to be applicable to student use within any of the three environments. This is followed by a discussion of assessment of these laboratories. Section 6. We conclude, Section 7, with a discussion of our efforts in making these environments and exercises available to the community.

2 TinkerNet

At its core, TinkerNet (Erlinger et al. 2004) is a system for letting students insert their own code for processing, generating, and responding to network packets into an OS kernel and booting that kernel on a physical machine. The system is designed to work with very limited hardware resources.

When using TinkerNet, students are provided with a skeleton source tree containing the function prototypes they must implement, as well as a GNU Makefile pre-configured to integrate the student's code into a TinkerNet aware Kernel. Their code both implements a

This quality assured paper appeared at the *20th Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2007), Nelson, New Zealand. Samuel Mann and Noel Bridgeman (Eds). www.naccq.ac.nz

function of the network protocol stack and exercises that code by sending and receiving network traffic. Using tools on the *server* (Figure 1), students can have their kernel remotely booted on one of the *nodes* and view output from that kernel. At no time does the student have to be aware of the existence of the *admin* network or the infrastructure in place to support it. Finally, when the student is done testing a particular build of their kernel, they can simply push a button on the *server* interface *Tinkerboot* and have their *node* reboot.

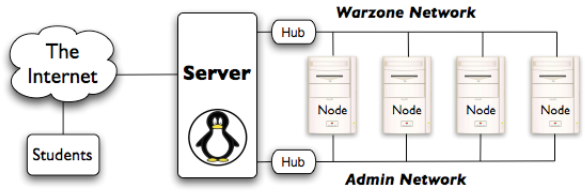

Figure 1: TinkerNet architecture

The *nodes* have a limited processing requirement: at most they need to keep up with incoming network traffic. Kernels are on the order of three to four megabytes. The *nodes* need no hard drive: most *nodes* in our installations have been given a network enabled bootloader via a floppy disk. Thus the total requirements for each *node* in a TinkerNet cluster are: two network cables, two network cards, power, and effectively any PC that will boot from a floppy.

In addition to *nodes* for student kernels, there are a few other hardware requirements. The *server* provides a home for the students and for all the TinkerNet software. The *server* connects to both the *warzone* and *admin* networks, and can also connect to the institution's campus network. Also, two hubs or switches are required to create the *admin* and *warzone* networks. Ideally at least the device for the *warzone* network would be a hub, since that makes all the *warzone* traffic available to all the students, giving them a more robust network experience.

2.1 Student Kernels

The kernel running on each *node* is a modified version of OSKit (Ford et al. 1997). OSKit was developed and distributed by the University of Utah's Flux Group, but is no longer maintained. The needs of TinkerNet are minimal, and thus many OSKit modules are not included in the build process. The modules used include the OSKit Standard C Library implementation, network drivers, and the memory manager.

2.2 Downloading and Managing Student Kernels

To boot the student OSKit kernels, a modified version of the GRUB (v0.97) boot loader resides on each *node*. When a student decides to boot a kernel, that kernel is sent through the student interface *Tinkerboot* to *Tinkercontroller*. *Tinkercontroller* then sends a signal to a waiting *node*, whose GRUB then uses TFTP to retrieve the kernel over the *admin* network.

Tinkercontroller, the heart of the software side of TinkerNet, is responsible for keeping track of which *nodes* are free, waiting, or missing; transfering kernels; logging debug data; and relaying both student and admin commands to the *nodes*.

2.3 Student and Administrative Interfaces

The student interface, *Tinkerboot*, allows students to download and remove kernels, maintain an integrated debug log, run a packet sniffer, and send custom packets (i.e., UDP packets containing student specified data).

Tinkeradmin, the administrative interface, allows an administrative user to reboot each *node*, and more importantly to see the status of each *node*. The administrative user also has access to the debug log of each *node*, making it easy for a lab assistant to help students debug their code.

2.4 TinkerNet Evaluation

In summer 2005 TinkerNet received a major overhaul; it was re-factored, commented, and made publicly available. Also, a step-by-step guide for building a TinkerNet was put onto a wiki (`www.cs.hmc.edu/twiki`). During years of use, several issues have become clear concerning TinkerNet.

2.4.1 Software

TinkerNet relies heavily on several open source projects. Two of the key components, OSKiT (Ford et al. 1997) and GRUB (`www.gnu.org/software/grub/grub-legacy.en.html`) are no longer being maintained. Presently, both are publicly available, compile, and run properly. However, a change in any of the above would severely cripple our ability to make TinkerNet easily deployable.

2.4.2 Hardware

The heart of TinkerNet is the network of *nodes* that students use to run their code. Current *node* implementations use throw-away computers. In effect the total cost of the network has been the two hubs and an extra network card for each *node*. The minimal cost of this system is one of its best features. When TinkerNet was first developed, simulators were neither free nor easy to use. Also many systems using actual computers had strict requirements for their nodes. Our *nodes* are cheaper, but because of their age they are prone to die.

2.4.3 Operation

While TinkerNet has little monetary cost, there is a slight price in terms of time. Booting a new kernel takes approximately 30 seconds and can not be shortened due to the system BIOS. Under normal usage this is not a problem since there are more *nodes* available than necessary.

Even with its warts, the TinkerNet framework lends itself to more than just a networking laboratory. The framework is sophisticated enough to handle distributed computing as well as a laboratory setup for an operating systems class.

2.5 Future TinkerNet Development

No future development of TinkerNet is planned. The system is available and bugs will be fixed as resources permit.

3 TinkerNet2

TinkerNet2 represents a radical departure from the construction of the original TinkerNet system. In TinkerNet2 User-Mode Linux, (UML) (`user-mode-linux.sf.net/`), is used to run the user's networking code, while still maintaining the operational structure of original TinkerNet. This means that there is no longer a need to have a rack of cheap machines as was required in the original TinkerNet. Instead, TinkerNet can be run on a single desktop. This greatly reduces material overhead and makes TinkerNet2 much easier to deploy.

3.1 User-Mode Linux

When considering further revisions to TinkerNet, one of the most appealing ideas was to use virtual operating systems instead of a cluster of old computer parts. Xen, User-Mode Linux, and VMware were all found to be possible options.

An Open Source solution was deemed to be best for an academic application. User-Mode Linux is a virtualization scheme that has documented networking and port communication abilities (very necessary for networking). UML has been incorporated into the Linux Kernel tree and is used for development of new kernel versions. This means that UML is reasonably stable and up to date with current Linux developments.

UML works by changing the memory space that a Linux kernel expects to use such that it will fit in the normal process space inside of another Linux kernel. Thus, User-Mode Linux appears as a process inside of Linux.

For TinkerNet2 a custom User-Mode Linux kernel was built in order to have the smallest memory footprint possible; same approach as was taken with OSKit in original TinkerNet. This allows TinkerNet2 to handle more users with less delay. This also allows removal of many extraneous features that could give student code greater access to networking tools than intended.

3.2 User-space Networking

TinkerNet2 required a system to run a student developed network protocol stack in the UML user-space kernel. A framework that could extract packets from a network device and present them in raw form to the UML user kernel as well as send raw packets out over a network device was developed by combining the abilities of *libnet* and *libpcap*. By combining this user space code and UML, a virtual *node* is created that can safely run student networking code.

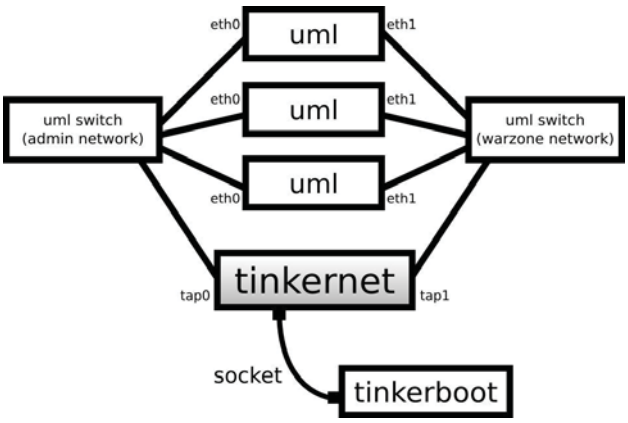

Figure 2: Network topology of the Virtual TinkerNet system

3.3 TinkerNet2 Operation

In TinkerNet2 the user's networking code is run in a kernel that is created within an instance of User-Mode Linux. This gives easy access to a simulated network. This system is controlled by a pair of programs, tinkernet and tinkerboot (Figure 2).

tinkernet, which corresponds to the *Tinkercontroller* in original the TinkerNet, controls the UML based network simulation. tinkernet keeps track of the UML nodes that are being used on the system, relays user commands to the UML nodes, and sends debug data from the nodes to the users.

tinkerboot (corresponding to *Tinkerboot* in the original TinkerNet) is the user interface to the system through which users load their code, send packets, and receive debugging information. One goal of the tinkerboot user interface is to remove the necessity of interacting directly with UML. Ideally, it should not be necessary for users to know that the system is based on UML.

3.4 Future TinkerNet2 Development

While TinkerNet2 is fully functional, there are features that were left out due to time constraints. The most important of these features is packet dropping. In original TinkerNet, *Tinkeradmin* had an option that would force a certain percentage of packets in the *warzone* network to be dropped. This feature is crucial for any realistic transport level data stream lab.

Development on TinkerNet2 continues with a plan to make it available to the community.

4 Virtual Distributed Ethernet

Virtual distributed Ethernet (VDE) (Davoli 2005), developed as part of the "Virtual Square" project (Davoli 2007), provides a distributed virtual local area Ethernet network based on virtual switches and virtual cables. Unlike TinkerNet2, which is a closed network simulator, a VDE can interconnect virtual machines (e.g. UML, Qemu (Ballard 2005), μ/MPS (Goldweber and Davoli 2005, Morsiani and Davoli 1999), other VDEs, or physical machines. Physical machines are connected to a

VDE through a virtual interface, e.g. tun/tap (Krasnyansky 2007).

All the connected machines see the VDE interface as if they were interconnected by a standard Ethernet LAN; even if all the connected "machines" were being run on different real hosts, possibly geographically distributed. VDE is like a multipoint VPN that interconnects both virtual machines and real hosts with a data-link layer abstraction.

The main features of VDE are

- Its behavior is consistent with a real Ethernet network.
- It interconnects virtual machines, applications and virtual connectivity tools to support interoperability with real networks.
- The hosts supporting the virtual machines, applications and virtual connectivity tools may be geographically disperse. (i.e. VDE works in a distributed fashion.)
- It runs completely in user-mode.

The structure of a VDE is itself consistent with the hardware structure of a real modern Ethernet network (Figure 3.)

The VDE switch (`vde_switch`) is the main component in a VDE. A real switch is a tool with several ports that can be used to interconnect computers and other switches. Similarly, a VDE switch has several ports where virtual machines, applications, virtual interfaces or connectivity tools can be virtually "plugged" in.

Two real switches are interconnected by a (cross) cable which is composed of two plugs and a wire between them. A VDE cable is composed of two VDE plugs (`vde_plug`) and a VDE wire. `vde_switch` and `vde_plug` are VDE component programs. A VDE wire is any tool able to transfer a data-stream (e.g. cat, netcat, ssh).

VDEs are particularly useful in the testing of new network protocols or student implementations of existing protocols on existing infrastructure. This would include the testing of IPv6 protocols on IPv4 infrastructures. Like TinkerNet, VDE is used to support a networking course where students build a complete protocol stack layer by layer. A VDE can also be used to build upon an operating systems course. Student written operating systems such as the Kaya project (Goldweber and Davoli 2005) or student experimentations with UML can be seamlessly extended in a networking course; the student written network stack will work on their student written/modified operating systems.

Additionally, VDE's can provide a safe environment where experimental services or student implementations of existing services can be designed, implemented, tested and deployed. This deployment can also optionally extend beyond the confines of the virtual network. Using the `slirpvde` tool one can connect a VDE network with real IP networks.

A key design feature of VDE is that it runs completely as a user process. Root access is only needed when one wishes to connect a VDE with a real IP network. Outside of this, student experimenters are free to design their own network topology, implement their own protocol stacks, modify existing protocol stacks, and implement their own network services.

A VDE is in essence a safe, open-ended "sandbox" (with very high walls) for student experimentation; whatever happens on the VDE is completely disconnected from both the underlying real network and other networks connected to it. VDEs have been used to support projects in network administration and network security in addition to traditional networking projects. For example, a VDE was recently used to support experimentation in detecting and testing countermeasures for denial of service attacks.

4.1 Future VDE Development

VDE is currently supported and distributed (though at differing versions) by Debian SID, Gentoo, FreeBSD, MacOSX, and is available for many other distributions. While VDE development takes place on i386 and powerpc machines, under GNU-Linux, VDE has been ported to the alpha, amd64, arm, hppa, ia64, m68k, mips, mipsel, s390 and sparc architectures.

Future versions will support VLAN and 802.1Q encodings of multiple subnets on the same "port." This will allow a UML host to define several virtual interfaces over one VDE connection.

5 Laboratory Experiments

The prototype Laboratory experiments for all three environments are focused on student development and testing of a fully functional network protocol stack. Each experiment or phase builds on previous experiments. The experiments work up from raw Ethernet packets to a fully functional implementation of IP and then UDP.

Two additional experiments have been developed for TinkerNet where students create their own protocol and implement Blast (Peterson and Davie 2003), a microprotocol which fragments and reassembles large messages. Besides these experiments many others could be created, e.g., advanced courses could implement application protocols or network devices, such as a router.

5.1 Autograder

TinkerNet, which currently has the more mature set of curricular materials and labs, also supports autograder. This tool was developed to address the lack of easy and consistent grading methods, which we perceive as a detriment to TinkerNet's widespread adoption. In its current form, autograder requires a slight modification to each lab, i.e., students are required to implement a call and response with autograder.

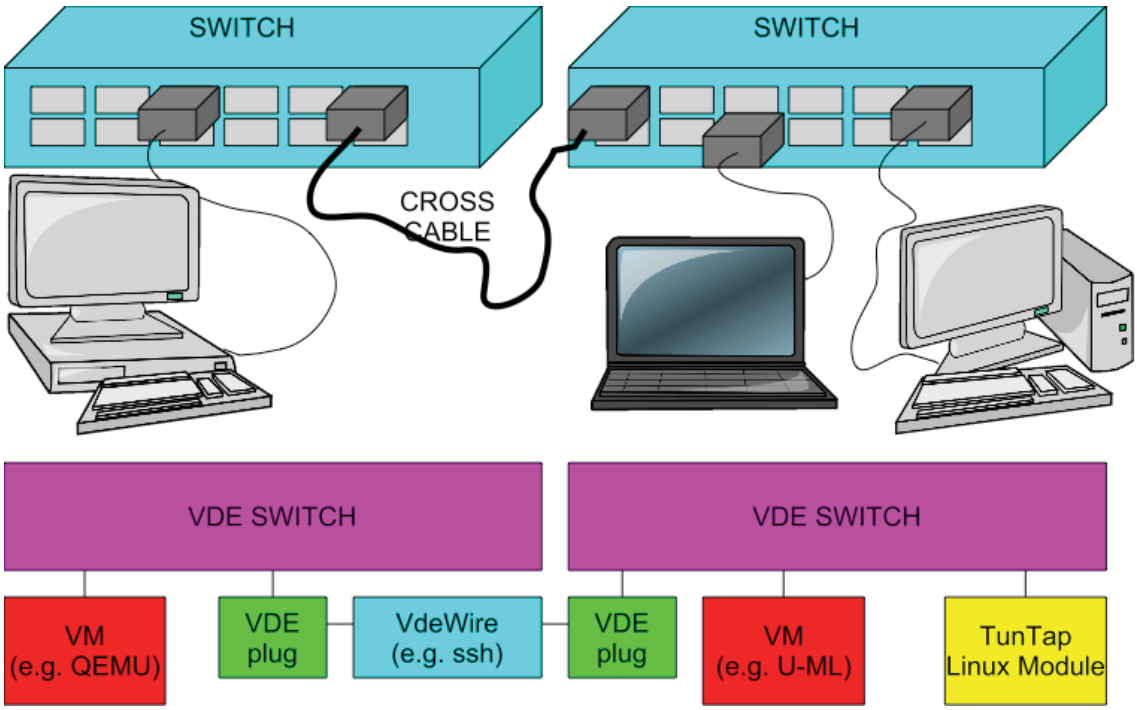

Figure 3. Comparison between a real Ethernet and a VDE

5.2 Future Lab Development

Regardless of which tool one may elect to adopt, TinkerNet, TinkerNet2, or VDE, for the tool to be useful outside of its developing institution(s), it is imperative that there exist high quality curricular materials/lab exercises for use with the tool. It is our intention to continue to create a single set of high quality curricular materials/lab exercises that can operate with any of the three tools/environments. For each exercise we will develop the exercise description, an example solution, and an auto grader script. We will start with the current set of TinkerNet labs, but plan on adding many more exercises.

6 Assessment

Of the three environments, TinkerNet, TinkerNet2, and VDE, only TinkerNet has been used in the classroom - Harvey Mudd College and University of California, Riverside. There are two aspects of TinkerNet that require assessment; creating a TinkerNet environment and use of TinkerNet and the lab exercises in the classroom. The TinkerNet documentation was developed so that any student or faculty member can build a TinkerNet environment.

At the beginning of each networking class a small group of students is selected to build a new TinkerNet environment using the online documentation. This process was never formally assessed. Rather, the group of students (three or four) was selected with one goal being limited experience in system administration. The creation of a new TinkerNet is viewed as a learning experience for these students.

In the four course offerings where TinkerNet was used no formal assessment was made of the development process, but students were asked to provide written comments on the online documentation. This effort has helped remove ambiguity from the documentation. In hindsight we should have created a pre-questionnaire and a post-questionnaire for this group of students. We plan to implement such questionnaires in the Fall '07 networking course. These questionnaires will be focused on system administration experiences, such as: setting up a network, operating system kernels, etc.

6.1 Classroom Assessment

Classroom assessment of TinkerNet based lab exercises has only occurred at the end of the course, i.e., final student course evaluations. These evaluations cover all aspects of the course, i.e., there is no specific set of questions directed at student learning via the labs. This is obviously a mistake and something that we will correct in future offerings. While the course evaluation comments have been very positive on the TinkerNet experience, detailed assessment is required. In particular, we plan to introduce at the beginning of each course offering a student questionnaire related to concepts that the TinkerNet experience will expose, e.g., layering of protocol modules, efficient data organization within packets, micro-protocols, etc. At the end of the semester we will ask students to fill out the same questionnaire. We will then compare the results. We are also considering contacting other schools teaching networking without a laboratory component to see how their students respond at the beginning and end of the course to the same questionnaire.

6.2 Network Laboratory Environment Assessment

In general there are a number of approaches to lab exercises for networking courses. Approaches include

- Socket Level Programming
- Network administration, e.g., capturing packets.
- Network simulation, e.g., NS or Opnet simulations.
- Development of a network tool, e.g., an e-mail client.
- Development of a network device: a router or switch
- Development of network protocols.

TinkerNet and VDE are designed to support most of the above approaches. But currently, only TinkerNet has been used in the classroom and its focus has been network administration and protocol development. There is a need for comprehensive assessment of all these approaches and how they are used at various institutions.

7 Conclusion

Original TinkerNet is a low-cost, flexible, stand-alone laboratory for running networking experiments, which combines ideas from others (Mayo and Kearns 1998, Levin 1997, Chapman and Carlisle 1997, and Rickman et al. 2001) with open source software (Ford et al. 1997, Nelson and Ng 2000). TinkerNet2 takes a software approach, UML, to providing the same environment as the original TinkerNet. VDE provides a virtual Ethernet LAN that can connect real or virtual machines. An advantage of these environments is their accessibility to institutions (e.g., undergraduate institutions) that do not have on-going research environments in the area of computer networks. We will continue to maintain and develop these environments, and make them available to the academic community. We also plan on developing a high quality laboratory experiment environment that would allow the same set of experiments to be run in any of these laboratory environments. These experiments will include solutions and grading scripts. The home of the TinkerNets is `cs.hmc.edu/tinkernet`, and VDE is available at `vde.sourceforge.net`.

8 Acknowledgments

This work was supported in part by the National Science Foundation under grant NSF-DUE-0443012 to Harvey Mudd College.

9 References

Ballard, F. (2005) Qemu, a fast and portable dynamic translator. In USENIX 2005's Annual Technical Conference, FREENIX Track hardware emulator.

Chapman, R. and Carlisle, W. H. (1997) A linux-based lab for operating systems and network courses. In Linux Journal, September 1997.

Comer, D. E. (2003) Hands on Networking with Internet Technologies. Prentice Hall, 2002. ISBN 0-13-048003-7.

Davoli, R. (2007) Virtual square home page. `http://www.virtualsquare.org/`.

Davoli, R. (2005) VDE: Virtual distributed ethernet. In the proceedings of `Trident.com` 2005, Trento, Italy.

Erlinger, M., Molle, M., Winters, T., Shea, R., and Lundberg, C. (2004) Tinkernet: A low-cost networking laboratory. In Computing Education 2004, Sixth Australasian Computing Education Conference. ACM Press, January 2004.

Ford, B., Back, G., Benson, G., Lepreau, J., Lin, A., and Shivers, O. (1997) The flux OSKit: A substrate for kernel and language research. In the symposium on Operating Systems Principles, pages 38–51, 1997.

Goldweber, M. and Davoli, R. (2005) The *kaya* project and the *umps* hardware emulator. In Proceedings of ITiCSE 05. Conference on Innovation and Technology in Computer Science Education, Lisbon, 2005.

ACM Press (1991) Joint Curriculum Task Force. Computing Curricula. ACM Press, 1991.

Krasnyansky, M. (2007) Universal tun/tap device driver. Linux Kernel Documentation: `Documentation / networking / tuntap.txt`.

Kurose, J., et. al. (2002) Workshop on computer networking: Curriculum designs and educational challenges, August 20 2002.

Levin, M. (1997) A prototype for a data communications laboratory or a data comm lab in a closet. In the ACM SIGCSE Bulletin, volume 29, pages 179–183. ACM Press, 1997.

Mayo, J. and Kearns, P. (1998) A secure-networked laboratory for kernel programming. In the ACM SIGCSE Bulletin, volume 30, pages 175–177. ACM Press, September 1998.

Morsiani, M. and Davoli, R. (1999) Learning operating system structure and implementation through the MPS computer system simulator. In Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education, pages 63–67, New Orleans, 1999.

Nelson, D. and Ng, Y. M. (2000) Teaching computer networking using open source software. In the ACM SIGCSE Bulletin, volume 32. ACM Press, July.

Peterson, L. L. and Davie, B. (2003) Computer Networks, A Systems Approach. Morgan Kaufann, 2003. ISBN 1-55860-832-X.

Rickman, J., et. al. (2001) Enhancing the computer networking curriculum. In ACM SIGCSE Bulletin, volume 33. ACM Press, June 2001.