TinkerNet: A Low-Cost Networking Laboratory

Mike Erlinger, mike@cs.hmc.edu
Computer Science Department
Harvey Mudd College
301 East 12th Street
Claremont, CA 91711

Mart Molle, mart@cs.ucr.edu
Computer Science and Engineering Department
University of California Riverside
SURGE Building, room 310
Riverside, CA 92521

Titus Winters, tdw@cs.hmc.edu
Computer Science Department
Harvey Mudd College
301 East 12th Street
Claremont, CA 91711

Chris Lundberg, cdl@cs.hmc.edu
Computer Science Department
Harvey Mudd College
301 East 12th Street
Claremont, CA 91711

Roy Shea, rshea@cs.hmc.edu
Computer Science Department
Harvey Mudd College
301 East 12th Street
Claremont, CA 91711

Abstract

The 2002 SIGCOMM Workshop on Educational Challenges for Computer Networking [Kur02a] exposed many issues related to teaching computer networking with the need for a laboratory in parallel with lecture a recurring theme. We have created TinkerNet, a low cost (mostly throw-away PCs), laboratory environment for networking experiments focused on the Build-Your-Own-Stack (BYOS) activity. This paper describes the hardware and software environment that is TinkerNet together with a set of student experiments. We discuss our plans for formal assessment to determine how the addition of laboratory experiments affects student learning. Finally the status and availability of TinkerNet is noted.

1 Overview

The 2002 SIGCOMM Workshop on Educational Challenges for Computer Networking [Kur02a] exposed many issues related to teaching computer networking with the need for a laboratory in parallel with lecture a recurring theme. We, (Computer Science at Harvey Mudd College and Computer Science at University of California, Riverside) have created TinkerNet a low-cost, flexible, stand-alone laboratory for running networking experiments and assignments focused on the Build-Your-Own-Stack (BYOS) activity.

TinkerNet allows students the opportunity to experience networking protocols by having direct access to an Ethernet network, with raw frame data arriving and simple functions for sending raw data out. The experiments are focused on students building and testing pieces of the network protocol stack, from OSI layer 2 (Link) up to layer 7 (Application). Because all of the nuances of kernel design are hidden from students, they can concentrate on the particular task at hand: building a network stack.

TinkerNet, Figure 1, is a stand-alone laboratory consisting of an array of back-end nodes, two networks, and a controller. The back-end node array is a set of machines onto which students load their OS kernels. These machines have limited physical requirements - in our prototype the backend nodes are all Pentium 200s cast off by other departments as obsolete. Back-end nodes each contain a CPU, RAM, and two 10/100Mbit Ethernet interfaces (no monitors, keyboards, or disks), and are accessed entirely via their network interfaces. One of these interfaces is attached to the *test* network (e.g., 192.168.100/24). The other interface is connected to the *admin* network (e.g., 192.168.200/24).

The *admin* network provides connectivity to each of the nodes and provides the services needed to administer those nodes, i.e., sending kernel images, remote control of the bootloader, etc. The *test* network provides students with a network to evaluate their network kernels, i.e., a network where student packets flow. An important security feature of TinkerNet is that neither the *admin* network nor the *test* network is connected to the institution's production network and/or the public Internet.

The TinkerNet controller connects to both the *test* and *admin* networks, just like a back-end node, but

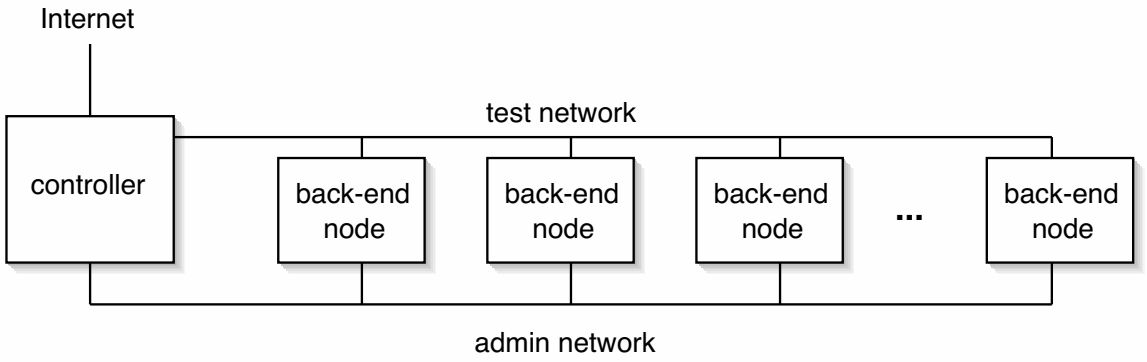

Figure 1: TinkerNet Architecture

also connects to the institution's production network on a third interface. The controller provides a home for the students and for all the software to make TinkerNet operational.

2 Implementation

The TinkerNet hardware infrastructure is based on our own experiences and interpretation of designs found in the literature. Comer's [Com02] networking laboratory description (discovered after we had prototyped TinkerNet) is similar to TinkerNet, but differs in significant ways. The most significant differences being Comer's need for special hardware (Console Multiplexor and Reset Controller) and his use of an operating system with limited features and accessibility, XINU [Com84]. TinkerNet is based on commodity hardware and the readily-available OSKit[Flu02][For97], Linux, and GNU software. Our software choices are more widespread within the computing community, and thus TinkerNet will both benefit from the symbiotic relationship with these other projects and have more acceptance because it involves well known and easily available technology. An additional advantage of the TinkerNet approach is that it is accessible even to those institutions (e.g., undergraduate institutions) that do not have on-going research in the area of computer networks. Our prototyping effort and Comer's laboratory description indicate that the TinkerNet design is reasonable and fully implementable.

Although students never have to be aware of it, their kernels are built using OSKit. OSKit is a project from the University of Utah's Flux group [Flu02], [For97] which makes prototyping and development of operating systems and OS kernels easier by providing a set of component libraries that can be added or removed from the kernel as necessary. TinkerNet uses the OSKit *libc* implementation, network drivers, and memory manager. TinkerNet does not need most of the features common to fully functional operating systems (file-system, TCP/IP stack, etc), so most of the OSKit provided modules are not utilized or linked into the student kernels. However, the presence of these modules does mean that TinkerNet can be easily expanded to teach principles based on these additional modules, e.g., Operating System fundamentals.

In prototyping TinkerNet we found the need for some custom code: a simple network stack, an ARP table, a simple scheduler, initialization functions, and a *printf* function that works over a network connection instead of to a terminal. All of these modules have been prototyped and are functional.

2.1 Administrative Network Stack

The administrative network stack is used for communicating information between a back-end node and the controller without interfering with traffic on the *test* network or relying on the proper implementation of *test* network protocols. This stack supports remote debugging, messages to boot and release kernels, as well as providing control over auxiliary devices, such as power distribution units and terminal servers. The administrative network stack, in essence, replaces the traditional I/O interface for the back-end nodes.

2.2 ARP Table

Proper implementation of a network stack implies that ARP requests will not be made for every outgoing packet. MAC addresses are stored in a hash table keyed on IP address for quick access. This ARP table is only used in the administrative portion of the kernel code; students must still implement their own ARP table for the *test* network.

2.3 Scheduler

To reduce the chances that student projects create infinite loops or crash their kernel so that the machine does not reboot properly, all student code is executed outside of the networking interrupt structure. The frames read by the test network are scheduled by a primitive scheduler and placed in a queue. When the task execution function (the only portion of the project running outside of interrupts) gets to this packet, it will dequeue it and call the **ethrecv** function. This scheduler is a non-interrupting scheduler in that it will never terminate a task. However, system level interrupts are capable of interrupting the scheduled tasks. Thus, it is still possible to reboot the

machine remotely, query for debug output, etc., using the *admin* network. This approach has the added benefit of providing debugging information even if the student's code is buggy and has crashed.

2.4 netprintf

This function is used to print debugging output from the back-end node over the *admin* network rather than to the console. It functions exactly like *printf*, since it is implemented using the same *libc* internals and argument lists.

3 Operational Overview:

When using TinkerNet, students are provided with a skeleton source tree containing the function prototypes they must implement, as well as a GNU Makefile preconfigured: to build the student's source, to link the student object code to the existing object code for handling the *admin* network, and to prepare the image to be sent to a back-end node. Using tools on the controller, students can have their kernel remotely booted and view output from it. 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 controller and have the node their kernel was booted on reboot and rejoin the pool of available nodes.

4 Laboratory Experiments

In developing our prototype we have created sketches of a semester-long set of laboratory experiments focused on student development of a fully functional network protocol stack. In this set of experiments, each new experiment builds on the previous experiments and is keyed to lecture material. Currently we use Peterson and Davie [PD03] as the course text, but having previously used Comer, [Com03] we believe the experiments can be easily matched to almost any computer networking text.

We begin with an experiment to review some issues around programming in C[1], and then work our way up from raw Ethernet packets to fully functional IP and then UDP. Later assignments build on what students have already experienced in the course by having them create some new protocols to handle issues in networking, e.g., finding other hosts supporting a protocol. We believe that there are many other experiments which could be created, e.g., a protocol to handle parts of TCP such as recovering dropped packets. We also believe a full implementation of TCP would require much more time then is available in a semester. We envision TinkerNet being used in advanced courses to implement application protocols and/or network devices, such as a router.

[1]In our standard undergraduate curriculum the majority of the programming is in C++ and many students will be unfamiliar with the differences between C and C++.

4.1 Example Set of Laboratory Experiments

- Lab 1: The goal of this assignment is to gain proficiency with C programming, and to (begin to) learn the differences between C and C++. There are also a few exercises that address networking in general.
- Lab 2: The goals of this assignment are: to gain familiarity with the lab environment, to successfully compile a TinkerNet kernel, and to implement functions that send and receive Ethernet packets.
- Lab 3: The goals of this assignment are: to gain more familiarity with the lab environment, to successfully compile a TinkerNet kernel, and to implement functions that send and receive ARP packets.
- Lab 4: In this assignment a simple, end-host version of the Internet Protocol is implemented. IP is the most basic protocol for transmitting data across heterogeneous networks.
- Lab 5: In this assignment sending and receiving of UDP datagrams is implemented, as well as a simple service to test this functionality.
- Lab 6: In this assignment the opportunity to design, create, and implement a peer-to-peer protocol is provided. In particular, this protocol can be used to locate other hosts on the network running the same protocol. Using this protocol, two machines will simultaneously boot on TinkerNet, locate each other, and then transmit data between themselves.

4.2 Detailed Description of a Laboratory Experiment

In this assignment, students gain an understanding of how a data transport protocol can be used to transmit more advanced network protocols from point to point. In particular, they are given raw Ethernet frames off of an Ethernet network. They then need to examine and to correctly categorize these frames.

> In the sample file **experiment1.c** there is a function **ethrecv** which you must write. Your function must keep the same declaration (name, return value, and parameters) or the project will **not** link properly. This function is given two parameters, a pointer and a length. This pointer points to a full Ethernet frame of the given length. All fields in this frame remain in network byte-order.
>
> You are to perform the following tasks:
>
> 1. Check the frame to see if it is addressed to your system, or to Ethernet broadcast. If the frame is not destined for either address, discard it. (For this project, discarding a packet involves returning from **ethrecv** without processing the packet.)

2. Check the protocol type field of the Ethernet frame. If the protocol contained in this Ethernet frame is IP or ARP, print out an appropriate message. If the encapsulated protocol is not ARP or IP, discard the frame.
3. Implement a function that takes a hardware address (array of 6 bytes), a pointer to some data, length of that data, and a protocol number. The function should then compose the proper Ethernet frame and send it on the network using the function **ethsend**.

Notes

The Linux kernel source (`linux/include/linux/if_ether.h`) contains both a *struct* that captures the fields of the Ethernet frame, as well as numeric constants for demultiplexing the encapsulated protocol.

Also, you should include the header file *machineInfo.h*. Information that is specific to the back-end node your kernel boots on will be filled in here automatically. This *struct* includes strings containing the node name, MAC address, and IP address.

5 Analysis: Practical and Pedagogical

At the 2002 SIGCOMM Workshop, Jorg Liebeherr presented *10 Thoughts on Networking Labs* [Kur02b]. As a way to position TinkerNet within the myriad of proposals for computer networking laboratories, we present Jorg's list modified with comments related to TinkerNet as well as three additional thoughts of our own. These comments reflect our views related to the focus of a networking laboratory and student learning. Our comments on student learning are currently anecdotal. Section 6 describes our plan for doing formal assessment of the TinkerNet Laboratory.

1. **Try to make education in computer networking more concrete.** Lectures cannot ever fully convey to the student the complexity and interactions of a system. TinkerNet provides the access needed for concrete experiences in networking. A key TinkerNet objective is to make this access inexpensive, easy to use, and easy to maintain.
2. **Don't teach a vendor-specific course on router configuration.** TinkerNet focuses on basic networking functionality. It provides a vehicle for a non-vendor approach to experiencing network development. While the current experiments emphasize TCP/IP, there is no reason other protocols could not be investigated and/or students could develop their own protocols.
3. **Use science labs as model.** TinkerNet provides an infrastructure around which experiments can be organized. Students create, observe, and measure using existing tools such as `tcpdump` and `ethereal`.
4. **Build on prior knowledge.** TinkerNet assumes a certain level of programming expertise which is usually obtained in earlier courses. Also, the experiments are keyed to lecture activities. Students have a certain knowledge from lecture that is put into practice using TinkerNet.
5. **De-emphasize skill - emphasize learning.** The primary purpose of TinkerNet is to study networking protocols, in particular, those implemented by the student. The ability to configure any particular network subsystem, e.g., a Cisco router, is not part of the activity. However, we do foresee future experiments where certain operational aspects of routing are implemented as a TinkerNet experiment.
6. **Students should feel in control of the equipment.** One of the purposes of TinkerNet is to give students total control over their kernels. They build these kernels, upload them, and control their execution. The TinkerNet software hides many of the details of student kernel management, but the primary aspects of building the network protocol stack are under the student's control.
7. **Keep it real.** TinkerNet is as real as it gets. Students build their own protocol stack, execute that stack on their own machine, and measure and evaluate the results. All this happens in an environment which is well controlled for the benefit of students.
8. **Organize a lab in 3 phases: Prelab, lab session, lab report.** Prelab consists of discussion of the experiment in class - during the lecture on that specific area. Lab sessions are then held at a later time. All students are presented with help in the laboratory in the form of student assistants. Each experiment requires a lab report, the contents of which are well defined.
9. **Leverage time investment.** "Designing, writing, and testing a single lab are a substantial investment of time. Therefore a lab course should stay relevant for several years." TinkerNet will last as long as there are operating systems and network protocol stacks. The material covered in TinkerNet experiments is timely now and will be timely for many years in the future.
10. **Control the need for supervision.** TinkerNet, like all laboratory environments, will require some supervision. This supervision can be at the level of student assistants rather then faculty. There are two time frames for TinkerNet experiments. First, there is a common laboratory time for all students. It is during this period that the majority of the experiment is completed.

Second, since TinkerNet is a dedicated environment, there is no reason why students could not use the laboratory at off hours to complete their experiments.

11. **Control the costs of creation, operating, and maintenance.** The TinkerNet environment is inexpensive with respect to hardware, as most of it is available for free. We believe that costs are less than $1000 for all parts other than the controller. The controller can be a reasonably configured PC. Once TinkerNet is developed, operation and maintenance are again inexpensive because the majority of the software modifications can be done by students and in reality will be available from the community (i.e., open source CVS tree). Overall TinkerNet is also inexpensive, because it can provide a machine for each student without dedicating that machine. The back-end node pool of machines is dynamically assignable to students as needed. Of great value is the fact that our initial experience with TinkerNet indicates that n nodes are sufficient for $2n$ students under average working conditions, and perhaps as much as $2.5n$ if students are aggressively reminded to not needlessly leave nodes booted while developing code.

12. **Create a laboratory environment that can be used by multiple courses.** TinkerNet is described as an environment for teaching computer networking, but there is no reason it cannot be expanded to teach other subjects, e.g., Operating Systems. Once the TinkerNet environment is created, additional experiments will require few changes to TinkerNet.

13. **Create a laboratory environment that is isolated from the normal computing activities.** TinkerNet can be both physically and logically isolated from department computing resources. This is particularly advantageous as students are given access to read all network traffic, and it is possible for students to accidently "destroy" the performance of a network. The isolated nature of TinkerNet eliminates any administration concerns with possible campus network or operating system accidents. Basically, TinkerNet allows student code to "crash and burn" without affecting computing resources outside of TinkerNet.

6 Status and Availability

We have prototyped TinkerNet by prototyping both the hardware and software environments. We have successfully used TinkerNet in one course offering at Harvey Mudd. We believe that TinkerNet has many advantages for teaching networking both pedagogical and administrative. In terms of pedagody we currently only have anecdotal evidence - student course comments indicate that students have been excited and enthusiastic about the laboratory and the amount of synthesis with material presented in lecture.

We are just beginning a formal assessment process. At HMC (where the networking course is taught once a year) we are developing a set of questions related to each experiment to be provided to students prior to the laboratory and after the corresponding lecture material. After the experiment students will be asked to reply to the same questions and to provide comments on how the experiment increased their knowledge (also any suggested changes to the experiment). At UCR where a number of sections of the networking course are taught each semester, it will be possible for TinkerNet use to be isolated to certain sections. UCR will then use a questionnaire given to all students taking networking to determine whether the TinkerNet experience has provided students with a better understanding of networking.

¿From the administrative viewpoint, TinkerNet is cheap to purchase and cheap to maintain. We have applied to the NSF (National Science Foundation) for support in developing and making available a full TinkerNet implementation - hardware design, robust software, and complete experiment set. Those interested in using TinkerNet in its current state can contact either Mike Erlinger (mike@cs.hmc.edu) or Titus Winters (tdw@cs.hmc.edu).

References

[Com84] Douglas E. Comer. *Operating System Design, The XINU Approach.* Prentice Hall, 1984. ISBN 0-13-637539-1.

[Com02] Douglas E. Comer. *Hands on Networking with Internet Technologies - Part V, Protocol Stack Implementation in A Special-Purpose Lab.* Prentice Hall, 2002. ISBN 0-13-048003-7.

[Com03] Douglas E. Comer. *Computer Networks and Internets.* Prentice Hall, 2003. ISBN 0-13-143351-2.

[Flu02] Flux Group. Utah The OSKit Project, June 2002.

[For97] .et .al Ford, Bryan. The Flux OSKit: A Substrate for Kernel and Language Research, October 1997.

[Kur02a] et. al. Kurose, J. Workshop on computer networking: Curriculum designs and educational challenges, August 20 2002.

[Kur02b] J. et. at Kurose. Workshop on computer networking: Curriculum designs and educational challenges, August 20 2002. Comments of Jorg Liebeherr.

[PD03] Larry L. Peterson and Bruce S. Davie. *Computer Networks, A Systems Approach.* Morgan Kaufmann, 2003. ISBN 1-55860-832-X.