Teaching Networking Hardware

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

Gregory Watson
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-9030
gwatson@stanford.edu

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

ABSTRACT

We present our experience with the design and teaching of a graduate-level networking hardware course in which students design and build an Internet router. Each team of two students (one proficient in hardware and one proficient in software) design and develop a fully functional router that routes live Internet traffic and inter-operates with other students' routers via a simple routing protocol. Hardware is designed in Verilog using an industry-standard design flow on a specially designed platform, called NetFPGA. Software is written in user-space using a high-level language. Software and hardware are combined and tested using real network traffic over arbitrary private topologies using a custom tool, called VNS. Our approach is distinguished in that both hardware and software can be designed, tested and deployed remotely over the Internet. Our goal is to give students experience in the design of complex networking systems. In our initial course offering in Spring 2004, all teams successfully implemented fully functional routers in less than ten weeks. We will pilot courses outside of Stanford using the remote teaching infrastructure presented in this paper.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education; C.2.1 [**Computer Communications Networks**]: Network Architecture and Design - packet-switching networks, Network communications

General Terms

Design, Experimentation

Keywords

Network project, Network internals, Pedagogy.

1. MOTIVATION

An increasing number of our graduating computer science students work in the networking industry, where they work on the design of complex hardware-software systems that must interoperate with existing systems and infrastructure. Our goal is to give students a relevant project experience, in which they design and debug a real piece of networking hardware and then deploy it into the Internet. Our traditional classroom and lab teaching doesn't provide students with hands-on experience in the subtleties associated with complex system design. This is in part because of the difficulty of creating meaningful learning exercises. Labs and projects based on commercial tools tend to become bogged down by the vagaries of a particular environment. Normally, network and link layer functions are buried in kernel space or specialized hardware. Even with a good user-level kernel debugging environment, development at this level degenerates into advanced kernel hacking. Furthermore, access to live Internet traffic has considerable safety and privacy concerns [6]. For these reasons, many projects rely on simulation and emulation environments [4, 5, 3].

We set out to develop an environment that allows students to implement complex systems built from both hardware and software, and then to test the system in arbitrary topologies while processing real Internet traffic. Our environment needed to be simple to use: for example, the hardware design environment should be based on an industry-standard design flow, and give students access to raw packets to process in any way they choose, yet be simple to learn, and enable designs to be completed in a short time period.

In this paper we describe our experiences with the design and pilot offering of a graduate networking course on "Networking Hardware". During the course, two-student teams designed fully functional, integrated hardware/software routers that process live Internet traffic. The course is based on two tools we developed: NetFPGA is a hardware design platform that was used to implement the main datapath through the router, and VNS (Virtual Network System) was used to implement the routing protocols and control software. The students designed their own interoperability tests, and used VNS to implement arbitrary private topologies that connect to the Internet via a secure firewall. Both NetFPGA and VNS can be used remotely over the Internet, allowing students to deploy and debug their designs remotely, without needing to see or touch the hardware platform. We believe that our environment overcomes many of the hurdles that make low-level network projects in hardware and software difficult to support in the classroom. We describe the tools we have developed and how they are used in the classroom,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ITiCSE '05 Universidade Nova de Lisboa
Copyright 2004 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

course details and feedback both from the students and industry experts regarding the usefulness and relevance of the course.

2. COURSE ENVIRONMENT

Our course was developed as part of a larger research effort to explore educational tools and methodologies for bringing network infrastructure technologies into the classroom. The goal is to develop a remote teaching lab that can support thousands of students in many, geographically distributed universities, providing them with direct, link-layer access to the Internet. Providing efficient and safe access to the Internet to a large and geographically disparate group of students is challenging. For this purpose we've developed two new technologies NetFGPA and VNS, which provide support for developing networking infrastructure by simplifying access to the Internet. While the focus of this paper is not to explain the inner-workings of either tool, we present a high level overview of their function in supporting the course.

2.1 The Virtual Network System

The Virtual Network System (VNS) enables the development of routers purely in software as user space processes. With VNS, an educator can create a virtual network topology at the edge of the Internet and assign each student (or group of students) a node in that topology. VNS provides the infrastructure for tunneling raw packets for each node to a student-written, user-space program which makes routing decisions and hands packets back for injection into the Internet. VNS has been successfully used in undergraduate networking courses here at Stanford and at Johns Hopkins to teach router implementation in software.

2.2 NetFPGA

NetFPGA is a platform where students can build networking hardware remotely. Each NetFPGA board (figure 1) provides eight Ethernet channels, two programmable logic devices, and some fast memory for network packet storage. Students create their designs using industry-standard EDA tools, and can then download their designs to the NetFPGA board over the network via a web interface. Once their design is running in hardware the VNS system is used to route real network traffic to and from their board such that it forms part of the campus network.

2.3 Combining VNS and NetFPGA

VNS and NetFPGA were designed so that they can be used individually and in tandem. When used together, NetFPGA and VNS provide seamless communication between the hardware and software portions of a project. Real-world networking devices are designed to use fast low-level packet processing implemented directly in hardware that communicates with proprietary software to offload complex operations. Projects which use both NetFPGA and VNS together allow students to develop the lowest levels of packet handling in hardware and the higher level functions in software. Both tools provide a simple raw packet interface to the student, with very few constraints or system dependencies.

3. COURSE COMPONENTS

Using VNS and NetFPGA, the hardware and software portions of the router are initially developed separately with periodic integration and testing. There is an interoperability requirement for the course; i.e. router must work with all other routers in the class, for example, by correctly exchanging routing tables according to a prescribed protocol. The course is structured to put the onus on the students to organize and plan interoperability testing.

Figure 1: NetFPGA Board: an 8 port, programmable network device

3.1 Router Development Overview

Router development during the course can be roughly broken down into three stages, development of the software and hardware base, integration and inter-operability and advanced functionality. The first two stages focus on creating a functional router base. In the last stage, students can optionally extend the functionality of their router. Each stage is composed of a number of milestones and deliverables that the students must complete and submit by predetermined deadlines. All hardware development is done in Verilog and industry-standard EDA tools. The software portions are written in C and can be developed on any standard *nix system. The groups are also responsible for maintaining two design documents that detail the design and implementation decisions for both the hardware and software respectively.

3.2 Deliverables and Testing

Deliverables at each milestone may be conceptual designs submitted as portions of the design document, code implementations or both. All code submissions are cross-checked against the design document to ensure the students are representing a consistent view of the router within the design document. Implementation deliverables are tested at each milestone to ensure correct functionality. Testing is cumulative when applicable. That is, each progressive submissions must be able to operate successfully on tests for the current milestone as well as all previous tests. Cumulative testing

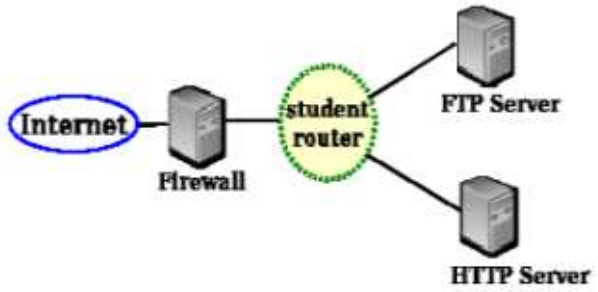

Figure 2: Single router topology used to develop basic router

is done to verify that continued development does not break existing functionality.

An important skill in networking is to be able to anticipate the many possible configurations and environments in which an implementation may be run. While we provide the students with functionality requirements for each deliverable we don't provide specifics as to what conditions the submissions will be tested. At the start of the course, each group is given their own, single router network topology for development and testing. The topology consists of a single router and two application servers as shown in figure 2. With VNS, we can construct network topologies of arbitrary complexity. We encourage the students to think about topology configurations that would be useful for testing their implementations, and upon request we set up topologies of their design. Students must support a routing protocol which requires that they request at least one topology with multiple routers.

3.3 Basic Router Functionality Requirements

The first two thirds of the quarter are spent developing a basic Internet router. We provide a minimal code base for both the software and hardware sub-groups to extend and integrate into their designs. For the hardware portion, the students are given a single port ethernet switch which demonstrates basic packet processing in Verilog. For software, we provide code to interface with VNS and a port of the TCP portion of the *lwip* [1] protocol stack. Both the hardware and software portion of the projects have incremental, progressive steps towards the full design. A basic router must support the following:

- IP forwarding in hardware
- Hardware ARP cache managed by software
- Software TCP stack which can support multiple network applications
- Management of routing table, arp cache and interfaces through a command line interface (CLI) reachable by telneting into the router
- IP decoding in hardware including
  - Basic header sanity check (version, length etc.)
  - TTL decrement on forwarded packets
  - IP checksum verification

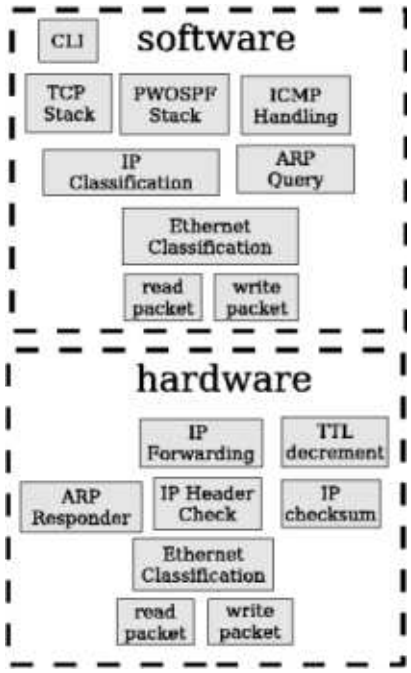

Figure 3: Functional components in hardware and software of completed basic router

- Update forwarding table via simplified inter-domain routing protocol

During development students can test virtually all functional components using standard network debugging tools and practices. This includes, interacting with the CLI via *telnet*, exercising ICMP support with *ping* and *traceroute* and handling traffic by using standard ftp and http clients to access content on the application servers.

3.4 Integration

The project is structured so that all router functionality is developed initially in software and then moved to hardware. This approach has the advantage that the students themselves develop a functional base reference in software that they can then use to verify correctness of their hardware design. The movement of functionality from software to hardware necessitates strong communication within the team regarding functional components, interfaces and semantics. We require the groups to document their integration strategy prior to attempting it. The functionality breakdown of a completed, fully integrated basic router is illustrated in figure 3.

3.5 Interoperability

It is essential that routers operate cooperatively in a complex, arbitrary and heterogenous environment. A router must not only strictly adhere to public standards but must be able to correctly handle traffic generated by varying interpretations of those standards, or packets with errors. A network pioneer Jon Postel put it:

> "Be conservative in what you do, be liberal in what you accept from others."

We believe this is a crucial lesson to take away from network system design. From the first day of class, students are told that their finished routers are expected to interoperate to build forwarding tables and route traffic on a complex and dynamic network topology. Furthermore, the students are responsible for developing the inter-operability testing plan and executing it. The students must therefore deal with the problems caused by varying implementations of a single set of standards. Differences and errors are overcome by communicating between groups. We also make available a complete, compiled reference solution for the groups to test against during development.

3.6 Advanced Functionality

After completing the basic router, students can extend their implementations to support additional functionality of their choosing. Providing an open-ended design problem allows ambitious and creative students to distinguish themselves. There are no specific functional requirements for this portion of the project, however the designs must have both a hardware and software component. Unlike the base router design where placement of functionality is rigidly mandated by the project specifications, the students now are forced to confront design issues inherent in integrated implementation. Each group must decide how much functionality to put in hardware vs software, what interfaces to expose for communications and what protocol properties the communications require. Typical functionality includes:

- Adding a managed firewall for network or transport layer filtering
- Adding NAT support to hosts behind the router
- Building a simple DHCP server
- Providing VPN-like support between two routers

4. CLASSROOM EXPERIENCE

The course was first offered in Spring 2004 at Stanford university. Seven MS/PhD students registered for the course, three with hardware experience and four with software backgrounds. Enrollment was limited for the first offering.

4.1 Basic Router Designs

All groups successfully completed the basic router. Final designs ranged from 7,000 to 12,000 lines of C, and 4,000 to 7,000 lines of Verilog. Despite the relatively specific functional requirements for the deliverables, each group uniquely approached the problems in their designs. For example, each router used a different mechanism for handling asynchronous packets, such as multiple threads, polling and signals.

A major component of succesful system design is anticipating the conditions under which an implementation may be used. At the start of the course, the students were informed that their finished routes would be tested on a large toplogy, but we did not tell them the size or configuration. To prepare for the final test, students could ask us to create a topology with a specific configuration. During the course, the groups requested three to four topologies each ranging from one to four routers with various subnet sizes on the connecting links.

The final testing topology consisted of seven routers and four end hosts. Multiple instances of the students routers were run on the topology interspersed with a reference solution. During testing we used VNS to virtually disconnect links between routers in order to determine the convergence time of the students' inter-domain routing protocol implementations. The routers also had to display stability downloading over 400 megabytes worth of data across 7 router instances.

All finished routers were able to perform the basic functions on three host topologies however only one of the groups routers was able to scale to the seven router topology without fault. The other two groups had minor shortcomings and were then given access to the testing topology to fix their implementations.

4.2 Advanced Functionality

All groups implemented advanced functionality in their routers. The final designs were presented to a panel of industry experts (router architects from local networking companies) who voted on the best overall router. Two of the teams based their extensions on ideas offered by us, but transformed them into their own designs that went far beyond what we had intended. The third team created an original project. Completed routers with advanced functionality ranged from 9,000 to 18,000 lines of C and 5,000 to 10,000 lines of Verilog.

4.2.1 Team 1: IP-in-IP tunneling and hardware ACLs

The team implemented a subset of a network firewall, by extending their router to support software configurable, hardware access control lists (ACLs). Access controls could be based on source IP, destination IP, source port, destination port and transport layer protocol. The group also implemented VPN-like IP tunnels between a pair of cooperating routers.

4.2.2 Team 2: Hardware flow control and web server

The team implemented source/destination filtering in hardware for flow control between IP address pairs. The flow control mechanism allowed software to set a maximum rate for each flow. The team also extended their router to support a functional web server from that allowed access and configuration of all modifiable internal parameters.

4.2.3 Ethernet encryption

Perhaps the most novel functionality involved link-layer encryption between cooperating routers. The implementing group developed an integrated hardware/software solution for encrypting the data portion of Ethernet packets prior to transfer between interfaces. The software was responsible for establishing a shared secret between the routers which was then used by the hardware as a DES key for symmetric encryption of the Ethernet data.

4.3 Student Response

We solicited feedback through an anonymous online questionnaire. Most students found the workload to be heavy (average 12 hours per week, peak 25 hours for the most complex portions). Overall, feedback was positive, and we plan to offer the class every year, with larger enrollment.

4.4 Response from Industry Experts

Students presented their routers to a panel of five industry experts with backgrounds in hardware and software router

design (thanks to architects from Cisco Systems and the XORP project[2]). The panelists were impressed by the level of sophistication of the base routers and advanced functionality. They found the material to be highly applicable to network system design and implementation. A number of the panelists commented on the lack of graduating students with sufficient backgrounds in network infrastructure technologies and inquired as to the availability of the students for positions after the course ended!

5. CONCLUSIONS AND APPLICATION

We presented our experiences designing and teaching a projects course in integrated router development. We believe that the course bridges a gap in current CS curricula by providing real-life hands-on experience with the design of network systems.

Offering the course at Stanford was a first step towards our goal of supporting remote classes at other colleges and universities. We are currently looking for schools who may be interested in offering this course. To support remote classes we have available all curriculum, specifications and reference implementations, host all support technologies (VNS and NetFPGA) and will offer full support for the duration of the course. To learn more about the course, VNS or NetFPGA, please visit their respective websites or contact us directly by e-mail.

- **Course Website:** http://yuba.stanford.edu/cs344_public/
- **NetFPGA Website:** http://yuba.stanford.edu/NetFPGA/
- **VNS Website:** http://yuba.stanford.edu/vns/

6. ACKNOWLEDGMENTS

This work was funded by the NSF, grant 02-082. We would like to thank the students who took CS344 during Spring Quarter 2004. We would also like to thank Dan Wendlandt and Guido Apenzeller for help in developing and maintaining NetFGPA and VNS.

7. REFERENCES

[1] lwip - a lightweight tcp/ip stack. www.sics.se/ adam/lwip/.

[2] Xorp the extensible open router platform. http://www.xorp.org/.

[3] C. N. Davis, C. S. Ransbottom, and L. C. D. Hamilton. Teaching computer networks through modelling. *ACM SIGAda Ada Letters*, XVIII(5):104–110, Sept/Oct 1998.

[4] M. W. Dixon and T. W. Koziniec. Using opnet to enhance student learning in a data communication course. *IS2002 Proceedings of the Informing Science + IT Education Conference*, pages 0349 – 0355, July 2002.

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

[6] 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.