Teaching Simplified Network Protocols

Dave Feinberg
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
(412) 268-4731
fberg@cs.cmu.edu

ABSTRACT
We created a course that, beginning from a hypothetical shared light bulb as our physical layer, introduced students to a hierarchy of simplified versions of network protocols, including Ethernet, IP, and TCP. This paper describes those simplified protocols, along with the Java framework students used to implement and simulate them.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education*. C.2.2 [**Computer-Communication Networks**]: Network Protocols – *protocol architecture (OSI model)*.

General Terms
Algorithms, Design

Keywords
Network, Protocol, Simulation, TCP, IP, Ethernet, Programming

1. INTRODUCTION
This paper describes the content of a course entitled "Network Programming," which we taught in the spring semesters of 2006 and 2008 to a total of 48 high school students at The Harker School, an independent school in San Jose, California. In the first half of the course, we introduced students to simplified versions of low-level network protocols (Ethernet, IP, and TCP) which students simulated using a Java framework we provided. At the midpoint of the class, students used their simulated networks to implement their own sockets. In the second half of the course, we switched over to using Java's built-in support for real socket connections. Students worked together in small groups, using sockets to write their own network applications, including networked games, chat programs, etc.

This paper focuses on the first part of the course–particularly the simplified network protocols we developed for teaching Ethernet, IP, and TCP. Lab-based networking courses typically fall into two categories: those in which students configure and experiment with networks, and those in which students implement network protocols [4]. Because we were teaching students who were not likely to continue studying computer networks beyond this course, there was no reason to provide practical training with real network traffic. Rather, we wanted to give our students a hands-on experience with network protocol concepts. We therefore decided to have students implement low-level network protocols on a simulated computer network. Although a number of network simulation frameworks have been developed for use in teaching, none met the precise needs for our course. Some simulation tools required students to implement subsets of the TCP/IP protocols, which could interoperate with industry implementations [1]. However, we felt that this would be unnecessarily detailed and time-consuming work for our students, and that it might obscure the main concepts. Some network simulation frameworks left only one layer for students to implement [3], while others allowed students to implement multiple layers of protocols [5, 7]. But at their lowest level, even the most flexible network simulation tools we looked at were based on the transmitting and receiving of frames. We felt this concealed too much of the physical layer and its relation to the data link layer. To allow students to explore unreliable networks, many tools could be configured to corrupt and drop frames at random. Such a feature, although clearly beneficial, also seemed contrived. We wanted to use a simulation model in which collisions could arise naturally.

In light of all this, we set out to create our own simplified protocol stack, spanning the layers from sockets down to bit detection. To minimize implementation times and remove confusing details, we chose to simplify common protocols so that they became small enough that our students could fully implement them in just half a semester, using our network simulation framework. Students would need to understand the purpose of every bit we used, so we wanted to reduce the protocols to their most essential features, focusing on those bits that really count. We made sure that students understood that we were omitting many details in the real Ethernet and TCP/IP specifications. We believe that our idealized protocols were more effective than the real ones for teaching our students to master fundamental networking concepts.

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

2. PHYSICAL LAYER: LIGHTS
Our physical layer began from perhaps the simplest possible entity that could support communication: a shared bit. This layer took two forms: a story about flashing lights, and a set of black-box Java classes that simulated those flashing lights. In the story, we imagine a dozen people know they will be imprisoned, one person

per cell. Each cell has a single light panel, featuring a light bulb, on/off buttons, and a unique ID number. Pressing the "on" button on one light panel turns on the light bulbs in every cell. Likewise, the "off" button turns off everyone's lights. In other words, all light panels are connected to a single light system, as illustrated in Figure 1.

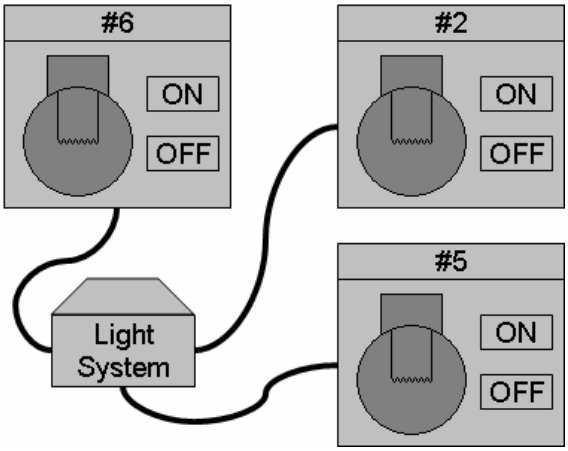

Figure 1. Three hypothetical light panels connected to a light system

Knowing all this, the dozen people conspire to use the light system to communicate with each other. They know they can represent any information as a string of bits. We asked students to consider how the light panels could be used to transmit bits, and what difficulties would arise. The results of that discussion served as the basis for introducing our simplified Ethernet protocol.

3. DATA LINK LAYER: ETHERNET

First we focused on the sending and detecting of bits, using Manchester code (Phase Encoding). To transmit a "0" in *t* seconds, we turned the light on for *t*/2 seconds and then off for *t*/2 seconds, and vice versa for a "1". To eliminate potential ambiguity, we agreed to begin each transmission with an extra "0". Thus, to send the bits "1100", we turned the light on and off in the pattern shown in Figure 2. In between transmissions, the light bulb was left off, indicating network silence.

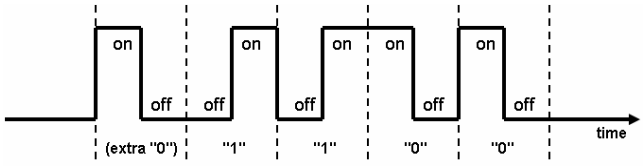

Figure 2. Pattern of on/off transitions to transmit "1100"

We chose Ethernet [6] as our layer 2 protocol, because of its widespread use in homes and schools. Using 48-bit MAC addresses seemed unnecessarily tedious, given that we did not plan to have more than a handful of network interfaces on our simulated network. Therefore, to facilitate learning the fundamentals of the protocol, and to simplify our implementation, we decided to use just 4 bits to represent a MAC address. This allowed us to examine a full Ethernet frame and easily identify the roles played by individual bits. Our 4-bit MAC addresses ranged from #1 to #15, where address #0 was reserved for broadcast frames.

Thus, the Ethernet frames in our simplified protocol consisted only of a 4-bit destination address, 4-bit source address, *n*-bit payload, and 2-bit checksum, as shown in Figure 3. A frame's payload could consist of any number of bits, but in practice, just 23 bits proved to be sufficient for our simplified protocol suite. For the checksum, we simply counted the number of 1s in the addresses and payload, and then found the remainder when this number was divided by 4. (This checksum function is a substantial simplification over the cyclic redundancy check used by the real Ethernet protocol.)

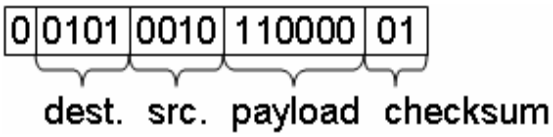

Figure 3. Simplified Ethernet frame for sending "110000" from address #2 to address #5.

To avoid collisions, we waited for silence before transmitting, and watched the state of the light bulb during transmission, to make sure what we set it to matched what we saw. If not, we knew a collision had occurred, in which case we sent a few extra bits to ensure the other transmitting party also detected a collision, before waiting to re-transmit the same frame. On the receiving end, we discarded any frames consisting of fewer than 10 bits, and any frames with invalid checksums.

4. NETWORK LAYER: IP

In our very simplified Internet Protocol, we used two 2-bit values to represent an IP address. For example, the binary value 01.11 represented the IP address 1.3. This was sufficient to let us talk about the value of hierarchical logical addresses. Each of our simplified IP packets consisted only of a destination IP address, a source IP address, and a payload. Each IP packet was transported as the sole payload of an Ethernet frame, as shown in Figure 4.

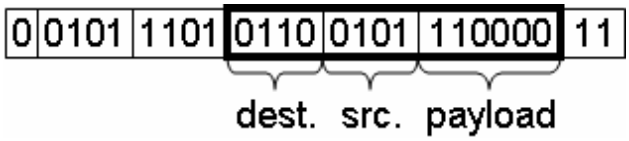

Figure 4. Simplified IP packet carrying "110000" from IP address 1.1 to 1.2. The packet is itself the payload of an Ethernet frame.

For all the technical details we were able to omit in constructing our simplified protocol suite, we found we could not introduce IP packets into our Ethernet simulation without presenting an Address Resolution Protocol (ARP). Using our simplified version of ARP, one could request an unknown MAC address by broadcasting an Ethernet frame to address #0 with a 4-bit payload, corresponding to the requested IP address, as shown in Figure 5. The corresponding ARP reply would contain the same 4-bit payload, with the requested MAC address now appearing as the Ethernet frame's source address, as shown in Figure 6.

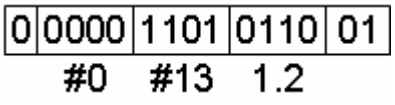

Figure 5. ARP Request for 1.2's MAC Address

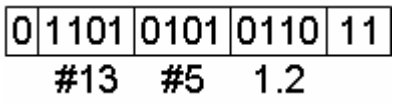

Figure 6. ARP Reply from 1.2

5. TRANSPORT LAYER: TCP

We presented TCP [2] as a completely independent layer from IP, where a TCP packet was the payload of an IP packet, as shown in Figure 7. We first focused on the functionality common to both TCP and UDP: inter-process communication. We chose to represent each process's port number as a 2-bit value. Each TCP packet therefore began with a 2-bit destination port number and a 2-bit source port number.

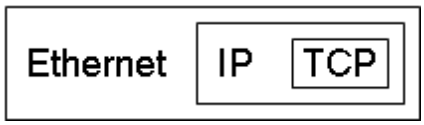

Figure 7. TCP packet embedded inside an IP packet, embedded in an Ethernet frame

We gave each TCP packet a 2-bit sequence number. Thus, in our model, a large message broken down into many TCP packets would begin with sequence #0, then #1, #2, #3, #0, #1, and so on. Next, we used a 1-bit flag to indicate whether this was the final packet in the message. Finally, we chose to have each TCP packet carry the 8-bit ASCII code for a single character as its payload. For example, the message "HELLO" would be broken down into 5 TCP packets, with the first carrying the 8-bit code for the letter "H", and so on. The receiving party would send an ack (acknowledgement packet) for each received letter. Each ack in our simplified TCP model consisted only of the port numbers and sequence number, as shown in Figure 8.

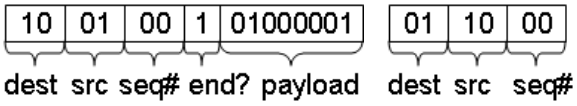

Figure 8. TCP packet #0 (carrying the letter "A") and corresponding ack

In class, we introduced students to the idea of a sliding window, in which a number of TCP packets would be sent before any are acknowledged, and where an ack from the receiving party might acknowledge several packets at once. However, in our implementation, we restricted ourselves to a 1-packet window.

So, in order to transmit "HELLO", we would first send the letter "H", and then wait for an ack. If no ack is received after some timeout, we would re-transmit the "H". Once an ack is received for the "H", we would then go on to transmit the "E". (This is essentially a stop-and-wait protocol.)

6. IMPLEMENTATION

Because the students had already completed an AP Computer Science course, we used Java to implement our network simulation software. Students were provided with two black-box classes: a LightSystem and a LightPanel. Upon running the LightSystem, students could instantiate new LightPanels, which would automatically connect to the LightSystem. Together, these two classes formed the physical layer of our network. (The LightSystem itself was implemented as a server, and could be connected to by LightPanels across the real Internet, using sockets. This allowed us to grade student work by testing that they could interoperate with our correct implementation of the simplified protocols.) The LightPanel API included methods for switching the light on, switching it off, and testing if it is on. The LightPanel could also be asked for its ID, which served as its MAC address.

In implementing our protocol stack, we divided the traditional layers into a number of classes, as shown in Figure 9. Students implemented these classes from the bottom up, starting with the BitHandler class, which could broadcast and detect bit strings.

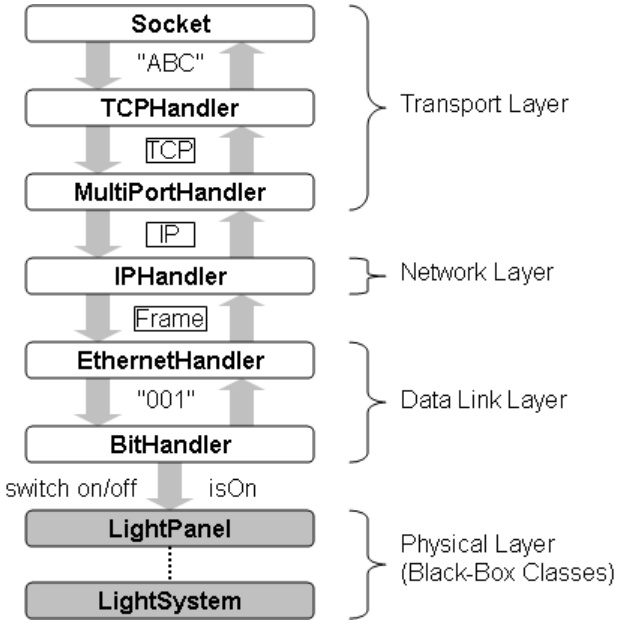

Figure 9. Layers of classes used to implement our network simulation

At the highest level, we created a Socket abstraction (along with a ServerSocket class). The Socket class allowed the user to establish a connection to a given IP address and port, and then send and receive text, one line at a time, until the socket was closed. To send a line of text, the Socket passed that text down to its TCPHandler, whose job it was to wrap each character in its own TCP packet, and to ensure the reliable transmission of each of those packets. To do so, each TCP packet was passed down to a MultiPortHandler, whose job it was to wrap the TCP packet in an IP packet, and pass that down to an IPHandler. The IPHandler would then place the IP packet in an Ethernet frame addressed to the appropriate MAC address (as found in its ARP table or determined from an ARP request), and would pass the frame down to an EthernetHandler. The EthernetHandler would convert the frame into a string of bits, and pass these down to a BitHandler (and handle collisions). The BitHandler's job was to inject those bits into the network by switching the light on and off with the appropriate timing (and to detect collisions).

On the receiving side, the BitHandler used a thread which would regularly poll the LightPanel to determine if the light was on or off. Using a state machine, the BitHandler determined the string

of bits received, and passed these up to an EthernetHandler. The EthernetHandler's job was to parse those bits into a frame, verify its checksum, and pass the frame up to an IPHandler (if the frame was addressed to this LightPanel's MAC address, or to the broadcast address #0). The IPHandler would then extract the frame's payload and handle any received ARP request or response appropriately. If the frame contained an IP packet, that packet would be passed up to a MultiPortHandler. This class would then extract a TCP packet from the payload of the IP packet, and examine the destination port number. The MultiPortHandler maintained a table, with each port number registered to a different TCPHandler. Using this table, the TCP packet would get passed up to the appropriate TCPHandler. If the packet was an acknowledgment, the TCPHandler would send the packet containing the next letter in the message (if any). On the other hand, if the packet contained a letter, that letter would get added to a received message string, and an acknowledgment packet would be sent back. When the entire string was received, it would be passed up to the Socket, and eventually returned to the user as the next received line of text.

7. USER INTERFACE

Students were provided with a LightDisplay class–a user interface for controlling and monitoring a light panel, as shown in Figure 10. The LightDisplay served as an invaluable tool for monitoring network activity.

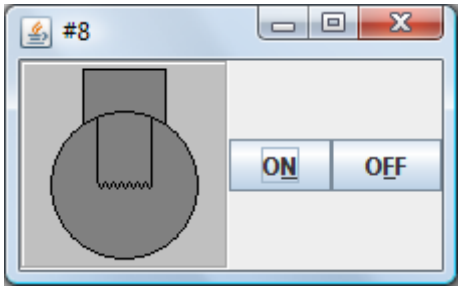

Figure 10. Screenshot of LightDisplay window

Whenever bits were received by a BitHandler, it would pass that bit string up to its registered BitListener. Students were provided with a BitDisplay class (shown in Figure 11), which implemented the BitListener interface. The BitDisplay showed the most recently received bit string, and allowed a user to enter a string of bits to send. The BitDisplay proved to be a valuable debugging tool, allowing us to sniff and spoof Ethernet frames.

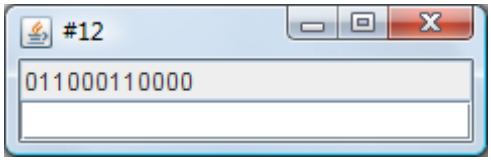

Figure 11. Screenshot of BitDisplay window

Later on, students programmed the EthernetHandler class to implement the BitListener interface, thus allowing an EthernetHandler to register to listen for any string of bits detected by its BitHandler. This architecture repeated itself at each layer, so that an EthernetHandler passed a received frame up to a registered EthernetListener, which might be an EthernetDisplay or an IPHandler, and so on.

8. SWITCHING AND ROUTING

We briefly discussed Ethernet switching in the course. Our more advanced students implemented a simple Ethernet switch, which consisted of several EthernetHandlers–each one on a different LightSystem. The switch snooped on Ethernet traffic to unobtrustively build up a table that kept track of which MAC addresses could be reached from which EthernetHandler. Using this table, the switch would attempt to forward each received frame to the appropriate EthernetHandler(s), if necessary.

We did not go into much detail concerning routing, as our primary goal for the course was to present the workings of the Internet at its endpoints–the students' computers. Thus, we emphasized that each machine knew which range of IP addresses would be found on its local area network (LAN), and that packets destined for all other IP addresses must be sent to the gateway machine (a router on the LAN whose IP address was already known to each machine on the LAN). Advanced students implemented a simple IP router, which consisted of several IPHandlers. The router was configured to know which range of IP addresses could be reached through which IPHandler. Unlike many traditional networking classes, we did not discuss routing algorithms and associated complexities.

9. METAPHORS

In our lessons, we employed a number of metaphors to teach networking concepts. As already emphasized, we used the light system metaphor as a physical layer on which to build our Ethernet protocol. We also compared Ethernet to a room full of people, all trying to talk at once, to motivate solutions to develop strategies for handling collisions. We then extended this metaphor to include multiple rooms of people, where each room played the role of an Ethernet segment, so that a designated person could play the role of an Ethernet switch by running between rooms to relay inter-room messages.

To introduce IP, we compared IP packets to postcards, each carried on its own mail truck (Ethernet frame) destined for some building (host) connected by a network of roads (the physical layer). Here, post offices played the role of routers. This allowed us to emphasize that the IP packet was addressed to the packet's final destination, while the Ethernet frame (truck) carrying that packet (postcard) might be headed to some router (post office) along the way. We extended this postcard metaphor for TCP, by discussing the problem of sending the text of a long book written on a large collection of individual postcards.

Later in the course, we implemented simple applications using client/server architectures. Here, we compared a server to a person waiting to answer calls placed to a published telephone number, and a client to a person calling that number. When a call is established, each person's connected telephone plays the role of a socket. Many students chose to implement network games as their final projects. To teach them how to use sockets to create their own application-specific protocols, we had students act out a 2-player game played over the telephone (simulated with cups and string), where two students acted as clients calling in to play a game, and a third student played the role of a server receiving their calls and passing the necessary information between them. We wrote down the various messages that students spoke into their phones, and used these as the basis for implementing a protocol to play the game over the network.

10. OMITTED TOPICS

In our course, we presented only those areas of computer networking that students needed to know in order to implement socket-based network applications, and to understand how those sockets might be implemented in a simple network model. In choosing to focus on this slice of computer networking, we omitted a number of networking topics that would have been relevant for our students, including HTTP, email, network security, DNS, DHCP, network address translation, home networking, and wireless Ethernet. We also left out a number of topics that are traditionally covered in an introductory computer networking class, including information theory, network topology, routing and inter-router protocols, network flow and congestion, and Internet infrastructure. We only briefly touched on the history of the Internet, and we did not discuss the impact of the Internet on society or related ethical issues.

11. REFLECTIONS

Over all, we felt the course was successful in teaching the key ideas behind network protocols. As we introduced each new topic, the students were engaged in class discussions, anticipating communication difficulties and developing protocols to address them. Ultimately, it wasn't hard to steer the conversation toward the simplified protocols we designed for students to implement. At the midpoint of the course, students performed very well on a comprehensive exam covering the four layers of our simulated network protocol stack.

The implementation of our simulated network required the students to work with several advanced Java programming concepts from the very beginning of the course, including exception handling, multithreaded programming, and the use and implementation of listener objects. Unfortunately, all of these topics were needed for the very first programming assignment.

Although students enjoyed the challenge of running a simplified Ethernet protocol over the light system, their interest declined as we implemented IP and then TCP. Much of the same coding issues repeated themselves at each layer, and therefore later assignments seemed less novel to the students. Not surprisingly, the highlight of the course for our students proved to be the final project, in which students worked in small groups to implement their own network applications.

Our greatest difficulty in the course concerned the first assignment, in which each student's program needed to examine a blinking light panel to determine the received sequence of bits. Unfortunately, later assignments were very sensitive to subtleties in the implementation of the bit detection algorithm. It was not uncommon for a student's algorithm to work well enough for the Ethernet assignment, only to cause problems on the IP assignment. We therefore eventually provided students with an alternative implementation of the physical layer, which transmitted and received bursts of bits, instead of using flashing lights. This implementation solved our reliability issues, but noticeably lost the magic of communicating by flashing light.

We have considered implementing a physical layer that shows flashing lights, but which detects single bits and network silence automatically, and provides a higher level API. Perhaps the light system is too low-level, but it certainly proved to be a helpful metaphor and a great starting point for our discussions. And there was definitely something very satisfying about building so much functionality on top of a flashing light.

12. REFERENCES

[1] Casado, M. and McKeown, N. 2005. The virtual network system. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 76-80. DOI= http://doi.acm.org/10.1145/1047344.1047383

[2] Cerf, V. and Kahn, R. 1974. A protocol for packet network intercommunication. *IEEE Transactions on Communications* 22, 5 (May 1974), 637-648.

[3] Elsharnouby, T. and Shankar, A. U. 2005. Using SeSFJava in teaching introductory network courses. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 67-71. DOI= http://doi.acm.org/10.1145/1047344.1047381

[4] Kurose, J., Liebeherr, J., Ostermann, S., and Ott-Boisseau, T. 2002. Workshop report: ACM SIGCOMM workshop on computer networking: curriculum designs and educational challenges (Pittsburgh, Pennsylvania, USA, August 20, 2002).

[5] McDonald, C. 1991. A network specification language and execution environment for undergraduate teaching. In Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education (San Antonio, Texas, United States, March 07 - 08, 1991). SIGCSE '91. ACM, New York, NY, 25-34. DOI= http://doi.acm.org/10.1145/107004.107012

[6] Metcalfe, R. M. and Boggs, D. R. 1976. Ethernet: distributed packet switching for local computer networks. *Commun. ACM* 19, 7 (Jul. 1976), 395-404. DOI= http://doi.acm.org/10.1145/360248.360253

[7] Tymann, P. 1991. VNET: a tool for teaching computer networking to undergraduates. In Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education (San Antonio, Texas, United States, March 07 - 08, 1991). SIGCSE '91. ACM, New York, NY, 21-24. DOI= http://doi.acm.org/10.1145/107004.107011