Exercise 1 — Introduction & the HTTP Application Layer Protocol

Due: September 17-18, 2015

Our first week of readings and exercises will cover two topics. The first goal is to give you a general (and gentle) introduction to computer networking. As part of this introduction, you will learn about the OSI model for computer networking. This model is an attempt to specify (in the form of a recognized international standard) the facilities that must be provided to support computer communication. The OSI model organizes these facilities into a hierarchy of 7 layers. While these 7 layers do not represent the only way to structure network software, they provide a useful guide to the overall structure of a computer network that we will follow throughout the semester.

The top level of the OSI model's hierarchy is the *application layer* which involves the specification of how network-based application programs (including web browsers, mail client, and many other examples) use the network to communicate with one another. Our second goal for this week will be to learn enough about one application layer protocol for you to write a network application based on that protocol over the next few weeks. The application layer protocol we will study is HTTP, the Hypertext Transfer Protocol, which is the basis for communications between web browsers and web servers.

While we will start our exploration of networks at the top of the OSI hierarchy, we will not be taking a strictly top-down approach to the exploration of networks in the class. We won't be taking a bottom up approach either! Instead, we will start with a middle-up approach followed by a bottom-up approach.

The logic behind this plan is that there is an important logical break between what the OSI model calls the data-link layer and the layers above. The lowest two layers of the OSI model address the problems involved in getting just two computers that are lucky enough to be connected by an appropriate link to communicate effectively. The remaining layers of the OSI model (and to some extent the entire Internet layering model described in the text), take the ability to establish communication between a simple pair of computers as a given and address the complex issue of using this ability to construct a network of many computers that can all communicate with one another.

The purpose of this week's readings (and of your first programming assignment) will be to give you a clear sense of the goal. That is, to understand the functionality the designers and implementors of a network-based/distributed application expect of an operating system's networking primitives. In the weeks that follow, we will explore how to implement such functionality, given the assumption that it is "easy" to exchange messages between a single pair of directly connected computers. Finally, in the last portion of the course we will admit that it is not so easy to establish communication between even a single pair of computers and therefore explore the issues that arise in what the OSI model calls the physical and data-link layers.

The main readings for this week will come from our textbook (Peterson and Davie) and from a text by Kurose and Ross. In addition, I will ask you to read one early paper about the web and its HTTP protocol. I should admit that I am assigning this paper more for its historical interest than for the technical details you will derive from it. It was written by the creators of the web during its infancy. I hope you can appreciate their excitement at the fact that the WWW's growth had been so explosive that there were 829 web servers at the time!

I will put links to PDFs of this paper and the required material from Kurose and Ross on the course web page. You can access the course web page at:

`http://www.cs.williams.edu/~tom/courses/336`

For your introduction to networking, I would like you to read Chapter 1 of Peterson and Davie.

Peterson and Davie include a description of HTTP in their Chapter 9. Unfortunately, since they take a bottom up approach in their text, they assume familiarity with some lower-level details of the network in their description. So, I would instead ask you to read up to but and including §2.2 of Chapter 2 of the text by Kurose and Ross.

Finally, the paper I would like you to read is "The World-Wide Web" by Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret.

Exercises

1. Suppose bits are transmitted from Berkeley, California to Boston over an optical fiber at 1.8 Gbps (1 Gbps = 10^9 bps) going through simple repeaters as needed to amplify the signal. The propagation speed is 2 x 10^5 km/s. The total length of the fiber is 3,200 miles (1 mile = 1.6km). Determine how many bits have been transmitted and are propagating over the fiber when the first bit reaches Boston. That is, determine the bandwidth-delay product for the link. How "long" (in meters) is each bit?

2. In their discussion of resource sharing (§1.2.2), Peterson and Davie explain one advantage of breaking large messages into smaller units called packets. Links can be more easily shared among multiple streams of messages if those messages are broken down into smaller pieces.

   Breaking messages into packets can also be advantageous even if only a single flow of data is using a link. This is because in a switched computer network a receiving switch will usually begin forwarding a message or packet to its eventual destination only after the entire message or packet has been received.

   On the other hand, since each packet transmitted typically requires header bits (holding information like the packet's address and bits used to check for errors) in addition to the data encapsulated in the packet, using packets that are too small increases the total time required to transmit a message. This question (and the next) asks you to explore such tradeoffs.

   (a) Suppose that some message consisting of a large number, K, bits of data is being transmitted from host A to host C through a single switch B. Assume that each packet sent consists of H header bits and P data bits and that the transmission rate is R bits/sec. Assuming propagation delays are negligible, how much time would elapse between the beginning of transmission from A and receipt of the last bit at C if the packet size was so large that the message could be sent as a single packet (i.e. with P > K)?

   (b) If instead K/P >> 1, what would the time required to deliver the message be? (The assumption that K/P >> 1 is intended to mean that you can safely approximate by assuming P divides K evenly.)

   (c) What value of P would minimize the time required in the case of K/P >> 1? (A formula that yields a non-integer result is fine.)

3. Breaking messages into packets also has advantages related to recovery from communication errors. Protocols that guarantee reliable delivery do so by detecting when packets are lost or damaged and automatically retransmitting the damaged packets.

   Suppose that some message consisting of a large number, K, bits of data is being transmitted from host A to B (no intermediate switch this time). Assume that each packet sent consists of H

header bits and P data bits and that the transmission rate is R bits/sec. Further, assume that propagation delays are negligible. Assume that on average 1 in E bits is damaged in transmission and that the error detection information included in the packet header bits are sufficient to enable the system to detect these errors and retransmit the damaged packets. Don't worry about exactly how the detection and retransmission occurs. For the purposes of this question, just assume that the need for retransmission simply increases the total number of packets that need to be sent. Assume that K >> E >> P, meaning that it is safe to assume that no more than one error occurs in any packet and that the number of errors that occur is just the total number of bits sent divided by E. Also assume that E is large enough that you can ignore the possibility that an error will occur in any of the retransmitted packets.

(a) How much time would be required to deliver the message (including retransmissions)?

(b) What value of P would minimize the time required? (A formula that yields a non-integer result is fine.)

4. In §1.4.2 of their text, Peterson and Davie introduce the basics of performing network communications through sockets under Unix in the C programming language. If you think about their example program, however, it is quite lame. It is a one-way chat program. It allows the user running one of two programs they provide (the client) to send several text messages to another user running their server code, but the user of the server code has no way to respond! Imagine if your cell phone provider offered you a discounted texting option which allowed you to send message but never receive them (or the other way around). Would you be interested?

   Why did the authors choose this example over showing code for what would be a much more useful two-way text messaging mechanism? That is, explain why the two-way program would have to be significantly more complicated than the version presented. At the least, explain how any simple attempt to provide two-way service would either fail or only provide a very limited form of two-way service. If possible, outline programming techniques that could be used to provide two-way service. If you don't know the actual Unix mechanisms that could be used to implement a two-way service, feel free to be imaginative and invent new primitives that would provide the flexibility to implement the desired programs.

5. Again, if you think a bit more about the talk program described in §1.4.2 you should realize that the one-way-ness discussed in the previous question is not its only limitation. It also would be a bit odd if the only way to start texting a friend was to first call them up or send them an email asking them what IP address and port number you should use to connect to them.

   Most existing systems for texting/chatting assume that you first log in to an account on some server and then have the ability to talk to anyone else connected to the server using the other individual's name or account ID rather than their machine's IP address and port number.

   For this question, I don't want you to worry about the details of writing any code for such a chat service. Instead, I want you to think a bit about the design of a protocol for such a service. That is, I want you to roughly outline the role's the server and clients in such a system should play and to describe the sorts of messages the clients would exchange with the server and possibly with one another.

   You should assume that there will be a server program running on a fixed IP address and port number that can be incorporated in the client program's code. You do not have to worry about the user interface provided by the client program. You should assume the user running the client will have to enter their own user id (and possibly password) when they first start the program. Don't

worry about how one creates accounts, changes passwords, or other such account management issues. Do think enough about privacy to try to avoid having all messages exchanged by clients pass through the server.

Your answer to this question should describe all connections made between the clients and the server and any connections between pairs of clients as well as describing the different types of messages that will be sent through these connections.