Assignment 9: Introduction to TCP

Due the week of November 7/8, 2012

Our goal this week is to study transport protocols. In the process, we will learn a bit about one of the most important protocols used in the Internet, TCP.

To learn the basics of the Internet's transport layer protocols, UDP and TCP, please read §5.1 from the text and §5.2.

In addition, I would like you to read some material that describes how some common application layer protocols use TCP and UDP. From Peterson and Davie, read §9.3.1 which explains the Domain Name Service. I would also like you to read some sections of "Computer Networking: A Top-Down Approach Featuring the Internet" by Kurose and Rose. (You may also want to read §9.1.1 through 9.1.3 for another take on the material you will read from Kurose and Ross)

Before you read Kurose and Ross, I want to warn you not to always do what the authors tell you. The authors suggest some interesting experiments you can do to "experience" the details of some TCP based protocols with the help of a Telnet program. I want you to complete some of these experiments, but not all of them. In particular, I want you to avoid the experiment using SMTP. The problem is that the results of experiments with SMTP tend to show up on error logs in Jesup and lead to suspicion that the experimenter is up to some form of evil. All of the other experiments are much safer.

With that said, please take a look at §2.2 of Kurose and Ross (you can skip §2.2.4 - 2.2.5, but you may find these sections interesting), §2.3 and §2.4 (you can skip §2.4.3). This material is from an older edition of the text. The contents of the latest version of the text is basically the same, but they took out all the cute stories about antique browsers. Don't worry about getting every detail described in this reading. I just want you to get an overview of how TCP is used and enough details to carry out some of the experiments (as part of problem 1 below). The experiment I don't want you to carry out is on page 111 where it says "It is highly recommended that you use Telnet..." Don't!

Please also read parts of Chapter 11 of the text "Exploring Java" by Niemeyer and Peck. This material explains how to write Java programs that send and receive data with TCP using the standard Java libraries.

Exercises

1. As I mentioned in the introduction, the reading from Kurose and Ross describes some experiments one can perform that can give a concrete understanding of how protocols like SMTP, HTTP and POP use TCP. I want to ask you to perform some of these experiments. You are not required to produce anything printed or written to turn in for this exercise, but be prepared to demonstrate the process during our meetings.

   Kurose and Ross give the first hint of how to probe the workings of a TCP based protocol on page 93 of the reading from their text. Basically, the technique they describe takes advantage of the fact that a Telnet client does little more than send whatever you type in as data through a TCP connection and display whatever is sent to you through the connection on your screen. So, if you can just get a Telnet program to connect to some interesting server (a web server, a mail server or whatever), you can see how that server responds to various messages.

   The trick to getting connected to a server has two parts. First you have to know the port number on which the server accepts connections. To spare you looking through the text or a pile of RFCs[1], the interesting port numbers are:

- 80 for HTTP
- 110 for POP

The second part is getting Telnet to use the port number you want rather than the standard Telnet port (23). On Unix systems, this is very easy. You just type the port number as an extra command line argument after the host name. So

```
telnet www.cs.williams.edu 80
```

will get you connected to the department's web server. Try it. After you type the command shown above and press return, type "GET" and press return again. What do you get in response?

Using the same basic approach, experiment with the web server and a POP server.

- Figure out how to fetch the HTML for the 336 web page

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

  using a telnet connection to port 80. The main advantage of using this page as your example is that it is fairly short.

  Use the readings from Kurose and Ross to figure out what commands the server should accept. Fetch pages using GET commands with no protocol specification and GET commands specifying HTTP/1.0 and HTTP/1.1.
- For a POP server, use studentmail.williams.edu and read your own email. Again, Kurose and Ross will provide details on how to talk to a POP server. If their explanation is insufficient, the RFC on POP is fairly easy to read. To find it, just Google "RFC 1939."

2. Want to see some REAL packets?

   There are a number of programs designed to collect packets as they flow by on a local Ethernet and to display their contents. One of these programs, wireshark, has been installed on our Linux systems. Unfortunately, for the sake of other's privacy, it won't actually let you intercept packets on your own. Fortunately, it can be used to examine the contents of packets intercepted earlier and stored in the appropriate format in a file.

   I have borrowed one such file (and the questions below) from an instructor at Cornell University. The file can be accessed using the path:

   ```
   ~tom/shared/336/traces/trace2.cap
   ```

   To view this file, type

   ```
   wireshark ~tom/shared/336/traces/trace2.cap
   ```

   (or just start wireshark and open the file using the "File" menu). It shows the packets that formed a TCP connection used to transmit a stream of packets from a machine with address 132.236.227.44 to a machine with the address 132.236.227.44.

[1]The RFCs (Request for Comment) are a series of technical documents that describe many aspects of Internet operation. All of the basic Internet protocol standard are described in RFCs. They are all easy to find online. Just Google "RFC 2616" or "POP3 RFC".

Play around a bit to learn how wireshark lets you examine the contents of the packets included in the trace. (You may find the user manual under the help menu useful.) Once you get the hang of it, use the program to answer the following questions about the TCP connection shown in the trace.

(a) Using the "TCP Stream Graph" item under the "Statistics" menu, look at the "Stevens" time sequence stream graph for the packets of the TCP stream. You should select the first packet of the TCP connection before selecting the menu item. Look at the stream of packets received in the first 20 ms (that is 0.02 seconds). Do you see any evidence of packets arriving out of order? Where?

(b) Look at the same graph and the first 100 ms. How many of the packets sent were not received the first time; can you point out which ones? What is the approximate value of TCP's timeout used to detect lost packets? How do you know?

(c) Look at the same graph and the first 250 ms. What is the size of the send window (in bytes) during the gap between the first and second flight of packets? How did you determine this?

3. Want to see even more real packets.

From my machine at home, I used the Unix ftp client to transfer a compressed version of one of the trace files in ~tom/shared/336/traces to my machine and then back to bull again. While doing this pointless pair of file transfers, I was running a packet sniffer named tcpdump on both bull and my home system to capture all the packets sent between the two machines. The resulting trace files are named `ftpfrombull.cap` (the trace captured by the sniffer running on bull) and `ftpfromhome.cap` (the file captured at home).

These files can be examined using wireshark. The only odd feature is that my router does something called NAT (Network Address Translation) which translates IP addresses and port numbers used within my home network into different values in the real Internet. So, while my machine thinks it is 10.0.1.2, bull thinks it is 24.15.13.215.

The complete transcript of the ftp session I conducted while collecting these traces is shown below.

```
% ftp cs.williams.edu
Connected to cs.williams.edu.
220-
220-The Department of Computer Science at Williams College.
220-
220-
220 bull FTP server (Version wu-2.6.1(1) Thu Jul 26 14:54:18 EDT 2007) ready.
Name (cs.williams.edu:thomasmu): tom
331 Password required for tom.
Password:
230-
230-This is the Department of Computer Science at Williams College.
230-Please be aware that anonymous ftp is available through our
230-main server, and that daytime access should be kept to a minimum.
230-It is currently Wed Apr 16 17:04:01 2008 in Williamstown.
230-
230-E-mail labstaff@cs.williams.edu with any problems.
230-
```

```
230-
230 User tom logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd shared/336/traces
250 CWD command successful.
ftp> get trace1copy.cap.gz
local: trace1copy.cap.gz remote: trace1copy.cap.gz
500 'EPSV': command not understood.
227 Entering Passive Mode (137,165,8,2,252,136)
150 Opening BINARY mode data connection for trace1copy.cap.gz (40454 bytes).
100% |*************************************| 40454      54.93 KB/s    00:00 ETA
226 Transfer complete.
40454 bytes received in 00:00 (48.48 KB/s)
ftp> ls
227 Entering Passive Mode (137,165,8,2,137,3)
150 Opening ASCII mode data connection for /bin/ls.
total 4544
-rw-r--r--  1 tom  CompSciFaculty     6173 Mar 30 00:25 ftptrace.cap
-rw-r--r--  1 tom  CompSciFaculty    13193 Mar 28 23:49 random.cap
-rw-r--r--  1 tom  CompSciFaculty   519855 Mar 28 15:15 smallhttp.cap
-rw-r--r--  1 tom  CompSciFaculty   195420 Mar 28 13:51 trace1.cap
-rw-r--r--  1 tom  CompSciFaculty    40454 Apr 16 17:03 trace1copy.cap.gz
-rw-r--r--  1 tom  CompSciFaculty    11384 Mar 28 13:51 trace2.cap
-rw-r--r--  1 tom  CompSciFaculty   344168 Mar 28 13:51 trace3.cap
-rw-r--r--  1 tom  CompSciFaculty    64184 Mar 28 13:51 trace4.cap
-rw-r--r--  1 tom  CompSciFaculty  1012004 Mar 28 13:51 trace5.cap
-rw-r--r--  1 tom  CompSciFaculty  1384204 Mar 28 13:51 trace6.cap
-rw-r--r--  1 tom  CompSciFaculty   383340 Mar 28 13:51 trace7.cap
-rw-r--r--  1 tom  CompSciFaculty      649 Mar 28 13:51 traces.txt
-rw-r--r--  1 tom  CompSciFaculty   593119 Apr  7 16:04 twoftps.cap
226 Transfer complete.
ftp> put trace1copy.cap.gz
local: trace1copy.cap.gz remote: trace1copy.cap.gz
227 Entering Passive Mode (137,165,8,2,178,240)
150 Opening BINARY mode data connection for trace1copy.cap.gz.
100% |*************************************| 40454      63.75 KB/s    00:00 ETA
226 Transfer complete.
40454 bytes sent in 00:03 (12.59 KB/s)
ftp> quit
221-You have transferred 80908 bytes in 2 files.
221-Total traffic for this session was 83340 bytes in 3 transfers.
221-Thank you for using the FTP service on bull.
221 Goodbye.
```

Using wireshark, I would like you to determine:

(a) What was the first packet sent as a result of each line that I typed during the FTP session in the ftpfromhome file? (Use the packet numbers displayed on the leftmost column of the wireshark packet display in your answer. I am assuming you are familiar enough with the use of the ftp command to distinguish the lines I typed from the program's responses. This might be foolish since programs that provide nice GUI interfaces to FTP functionality have largely replaced the command line version. So, if you are not familiar with how to use FTP from the command line a) talk to me, or b) notice that except for the log in process, all lines I typed follow the prompt "ftp>".)

(b) How many TCP connections were made during this FTP session?

- Identify the first packet of each connection.
- Who (i.e. my machine or bull) created each of the TCP connections?
- Who closed each of the TCP connections? Identify the packet that initiated the process of closing each connection.

(c) The session involved two file transfers. How many data packets were sent during each of these connections? How many ACKs were sent?

(d) During the two file transfer connections, the ACKs are sent in packets with no data rather than being piggy-backed with data packets. Why? Can you find any examples of ACKs that acknowledge data that the receiver had not previously tried to acknowledge and that are piggy-backed with actual data? If so, identify the first such packet.

(e) What password did I enter while logging into the FTP server?

4. Want to send some real packets?

Based on the information in the reading from "Exploring Java", I would like you to write a very simple Java program that will emulate enough of what a Telnet program does to replace "telnet" as the tool you might have used for the first problem.

This is meant to be a very short programming assignment! One of the nice things about the simple telnet program you used for problem 1 is that it has a very simple user interface. No fancy GUI components, just simple console input and output. As a result, almost all the code you have to write for this will focus on performing network I/O. Real telnet programs know how to negotiate options with a telnet server. Your program does not need to do this. All you need to do is send every line typed to the server and display every line sent to you on standard output.

The program I want you to write should expect two command line arguments: the "hostname" the user wants to talk to and the port number through which the connection should be made. For example, if the main class of your program is named `Mytelnet`, then you might type the command

```
java Mytelnet www.cs.williams.edu 80
GET / HTTP/1.0
```

to get the program to fetch and display the text of the HTML for the CS department's home page.

The only tricky part about this program is that you cannot predict how many lines of a server will send back in response each line the user types. So, your program should first use the Socket class to create a connection to the server. Then, it should create a separate Thread that continuously reads from the input stream associated with the Socket and uses System.out.println to display each line received. At the same time, the main program will read lines from standard input and send each line read through the output stream associated with the Socket.

Your program does not have to be particularly robust. As long as you can use it to reproduce the experiments from problem 1, you are fine (even if it always ends by throwing an exception, etc.). The goal is simply to give you just a little experience with an example of an API that can be used to send network packets (Java's Socket class in this case).

5. What Ethernet packets actually get sent through the campus network when you type

   ```
   java Mytelnet time.nist.gov 13
   ```

   Port 13 is used by servers that support the DAYTIME protocol. The "details" of the DAYTIME protocol are quite simple. Basically, you make a TCP connection to port 13 on the server and it sends you back a string that can be interpreted as the current day and time, and then it closes the connection. If you want more details read RFC 867.

   You could answer this question by running wireshark or some other packet sniffer, but that would take all the fun out of it. Instead, what I want you to do is use you knowledge of how TCP works to predict what packets would actually sent. Include the packets used to establish a connection, terminate the connection, send all the data and any required acknowledgments. You should assume that the entire response from the server fits in one packet.

   Assuming that none of the TCP or IP packets sent include any options, what will be the total number of bytes sent as IP packets to process this single request? (i.e., don't count the overhead of the underlying protocol (Ethernet, 802.11, etc.), just the data generated by TCP, IP, and the DAYTIME protocol.

6. Answer question 5.4 from Peterson and Davie.