TCP, UDP Lab

Tom Kelliher, CS 325

Mar. 7, 2011

1 Introduction

In this lab, you'll be experimenting with the TCP and UDP client/server code we just finished discussing in class. You'll use Wireshark to observe the packet/datagram traffic between client and server. I'll also ask you to make some modifications to the client and server code to prepare you for later projects.

Open an NX connection to merlin and login. All the source code you'll need is on the course web site. You can copy/paste the code from a web browser into your editor. You'll need to modify the server code to select a unique port number, preferably ≥ 30000 (and < 65536). You have two options for the location of your server:

1. Run the server on merlin. This is the easiest thing to do, but I don't care for the "overly orderly" way that the TCP packets travel back and forth between the client and server — it's not very realistic. The source and destination IP addresses will both be the same — `127.0.0.1` — which can get confusing in a big hurry.

   If you choose this option, you'll need to change the hostname in the client source code to `localhost` and set the port number to whatever port number you chose for the server. When you use Wireshark, you'll need to start it on the loopback interface (`lo`) rather than the physical interface (`eth0`).

   Use two copies of the terminal, one for running the server and the other for running the client. Similarly, depending upon your work flow, you may find it easiest to use two copies of an editor for client/source code modification.

2. Run the server on phoenix. This is a bit more complicated, but you'll see more realistic results in Wireshark.

   If you choose this option, you'll need to change the hostname in the client source code to `phoenix.goucher.edu` and set the port number to whatever port number you chose for the server. When you use Wireshark, you'll need to start it on the physical interface (`eth0`), as you've done previously. Remember, set your capture filter to `not tcp portrange 6000-6016`.

   From merlin, you'll need to ssh from a terminal window to phoenix. Depending upon your work flow, you may need two ssh sessions to phoenix. Locally on merlin, you'll be editing and running the client code.

You compile and run your code from a terminal shell. Compiling a Java application looks like this:

```
javac TCPClient.java
```

Running a Java application looks like this:

```
java TCPClient
```

Hand in answers to each of the questions asked below.

2 TCP

1. Compile the TCP client and server. Run the server; type `Ctrl-c` to it to terminate it.

   Run the client, typing a line of text to it once it's running for it to pass along to the server.

2. Use Wireshark to observe the data passed back and forth between the client and server. Hint: Set the Wireshark display filter to `tcp.port == <port>`, where `<port>` is the port number you're using, to restrict the display to your specific TCP session. Answer these questions:

   (a) The TCP connection initiation handshake requires that three packets be exchanged. What are the flags set in the handshake packets?

   (b) Is any application-level data exchanged during the handshake?

   (c) What is the total number of packets exchanged?

   (d) How many packets actually contain application-level data, whether from client to server or vice versa?

   (e) Is there any particular flag set when application-data is contained in a packet?

   (f) How many steps are in the connection tear-down handshake?

3. Run the client, but don't type a line of text to it. Instead, start another instance of the client in another terminal window and type a line of text to it. Then, type a line of text to the first client instance. Explain the result.

4. Create a large file (`largeFile`, > 1024 bytes) with a single newline character as the final character of the file. The file should contain **no** other newline characters. Feed the file to the client program like this:

   ```
   java TCPClient < largeFile
   ```

   (a) How many packets were exchanged?

   (b) Did the server return the entire file?

3 UDP

1. Compile the UDP client and server. Run the server; type `Ctrl-c` to it to terminate it.

   Run the client, typing a line of text to it once it's running for it to pass along to the server.

2. Use Wireshark to observe the data passed back and forth between the client and server. Hint: Set the Wireshark display filter to `udp.port == <port>`, where `<port>` is the port number you're using, to restrict the display to your specific UDP datagrams. Answer these questions:

(a) Does UDP use anything at all resembling a handshake prior to exchanging data?

(b) What is the total number of packets exchanged?

(c) How many packets actually contain application-level data, whether from client to server or vice versa?

3. Run the client, but don't type a line of text to it. Instead, start another instance of the client in another terminal window and type a line of text to it. Then, type a line of text to the first client instance. Explain the result.

4. Use `largeFile` from the TCP client/server experiment with your UDP client/server:

```
java UDPClient < largeFile
```

(a) How many packets were exchanged?

(b) Did the server return the entire file?

4 Client/Server Modifications

1. Modify the TCP client/server so that you can type several lines of text to the client and have them echoed back from the server, line by line. The client should gracefully exit, closing the socket, when you type the end-of-file character (`Ctrl-d`) at the beginning of a line.

   You can test your modifications by feeding a Java source file to the client:

```
java TCPClient < TCPClient.java
```

2. Repeat for the UDP client/server.