The University of Calgary

Department of Electrical & Computer Engineering

ENSF 462 Networked Systems

(Fall 2023)

Lab 1 - Wireshark Lab and Socket Programming[1]

Lab Section	Section Date	Location
B01	September 21st, 2023	ENA 305
B02	September 26th, 2023	ENG 24
B03	September 20th, 2023	ICT 319

Part 1 – Wireshark Lab

1.1 Introduction

In this first Wireshark lab, you'll get acquainted with Wireshark, and make some simple packet captures and observations.

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or email client)

Acknowledgement:
Part of the content in Lab 1 manual is modified based on the material provided on the authors' website for the textbook, *Computer Networks, A Top-down Approach, 8h ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.*

that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer over a given interface (link layer, such as Ethernet or WiFi). Recall that messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable or an 802.11 WiFi radio. Capturing all link-layer frames thus gives you all messages sent/received across the monitored link from/by all protocols and applications executing in your computer. The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message.

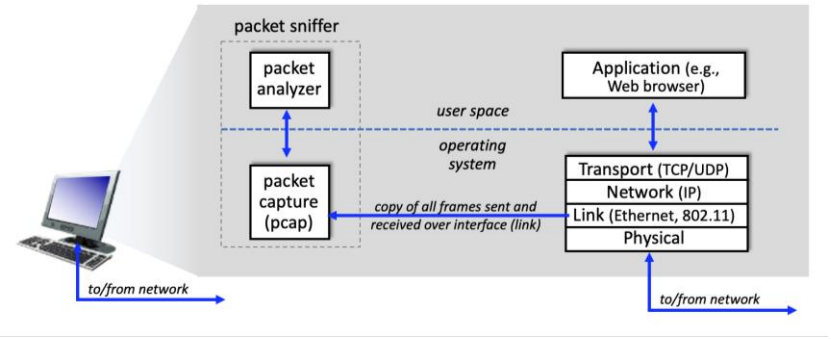

Figure 1: packet sniffer structure

1.2 Getting Wireshark

We will be using the **Wireshark packet sniffer** for this lab, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. In order to run Wireshark, you'll need to have access to a computer that supports both Wireshark and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed for you, if it is not installed within your operating system, when you install Wireshark.

Download and install the Wireshark software:

- Go to http://www.wireshark.org/download.html and download and install the Wireshark binary for your computer.

The Wireshark FAQ has a number of helpful hints and interesting tidbits of information, particularly if you have trouble installing or running Wireshark.

1.3 Running Wireshark

When you run the Wireshark program, you'll get a startup screen that looks something like the screen below. Different versions of Wireshark will have different startup screens – so don't panic if yours doesn't look exactly like the screen below!

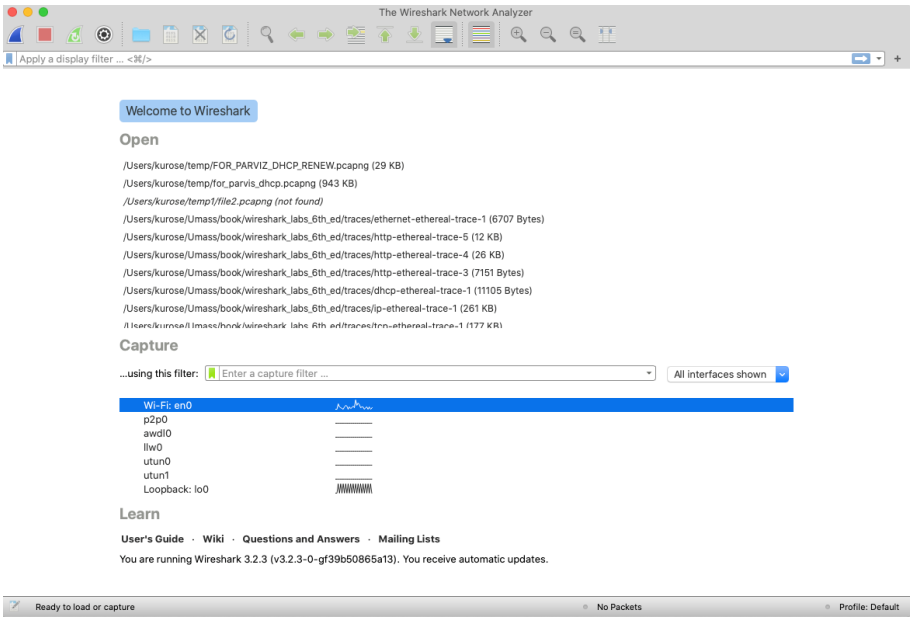

Figure 2: Initial Wireshark Screen

There's not much that's very interesting on this screen. But note that under the Capture section, there is a list of so-called interfaces. The Mac computer we're taking these screenshots from has just one interface – "Wi-Fi en0," (shaded in blue in Figure 2) which is the interface for Wi-Fi access. All packets to/from this computer will pass through the Wi-Fi interface, so it's here where we'll want to capture packets. On a Mac, double click on this interface (or on another computer locate the interface on startup page through which you are getting Internet connectivity, e.g., mostly likely a WiFi or Ethernet interface, and select that interface).

Once you click on one of these interfaces to start packet capture (i.e., for Wireshark to begin capturing all packets being sent to/from that interface), a screen like the one below will be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the Capture pull down menu and selecting Stop (or by clicking on the red square button next to the Wireshark fin in Figure 2).

This looks more interesting! The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the Wireshark window (and on a Mac at the top of the screen as well; the screenshot in Figure 3 is from a Mac). Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data and exit the Wireshark application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; note that this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing

can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.

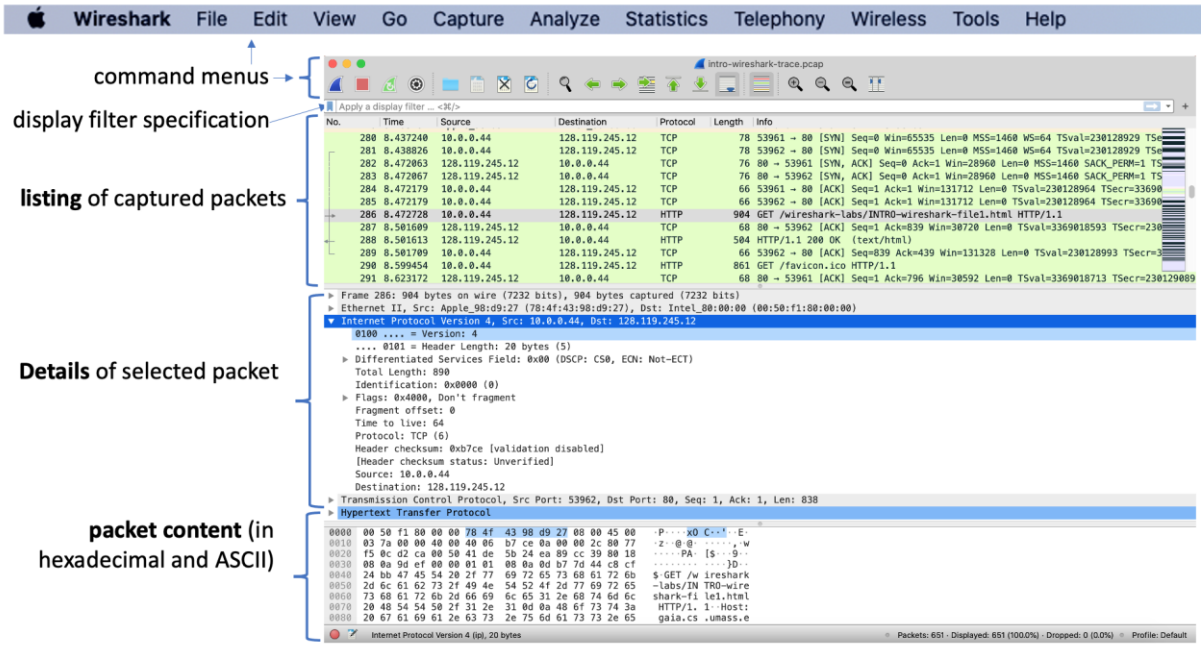

Figure 3: Wireshark window, during and after capture

- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus/minus boxes or right/downward-pointing triangles to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.
- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field,** into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows).

1.4 Packet Capture Using Wireshark

1. Start up your favorite web browser, which will display your selected homepage.

2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 2. Wireshark has not yet begun capturing packets.

3. To begin packet capture, select the Capture pull down menu and select *Interfaces*. This will cause the "Wireshark: Capture Interfaces" window to be displayed (on a PC) or you can choose Options on a Mac. You should see a list of interfaces, as shown in Figures 4a (Windows) and 4b (Mac).

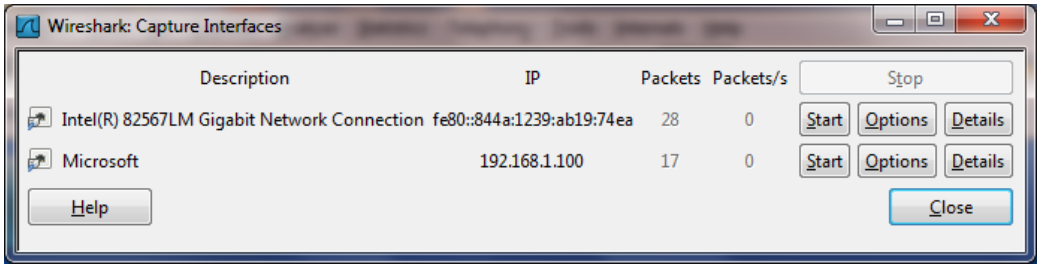

Figure 4a: Wireshark Capture interface window, on a Windows computer

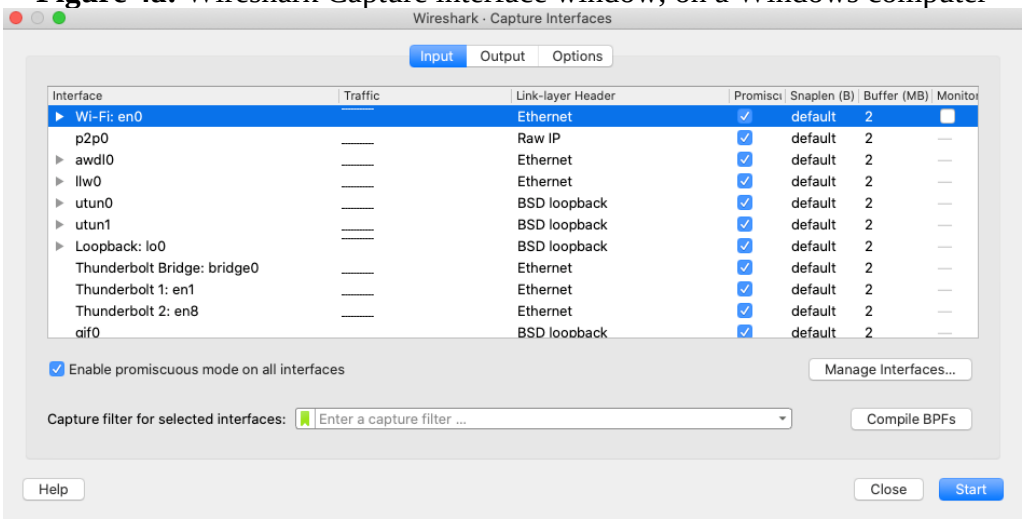

Figure 4b: Wireshark Capture interface window, on a Mac computer

4. You'll see a list of the interfaces on your computer as well as a count of the packets that have been observed on that interface. On a Windows machine, click on *Start* for the interface on which you want to begin packet capture (in the case in Figure 4a, the Gigabit network Connection). Packet capture will now begin - Wireshark is now capturing all packets being sent/received from/by your computer!
5. Once you begin packet capture, a window similar to that shown in Figure 3 will appear. This window shows the packets being captured. While Wireshark is running, enter the URL: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html and have that page displayed in your browser. Your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page. The Ethernet or WiFi frames containing these HTTP messages will be captured by Wireshark.

6. After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop Wireshark packet capture by selecting *Capture* pulldown menu and selecting *Stop*, or by click on the red Stop square. The main Wireshark window should now look similar to Figure 3. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well. Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user.

7. Type in "http" (without the quotes) into the display filter specification window at the top of the main Wireshark window. Then select *Apply* (to the right of where you entered "http") or just hit return. This will cause only HTTP message to be displayed in the packet-listing window. Figure 5 below shows a screenshot after the http filter has been applied to the packet capture window shown earlier in Figure 3.

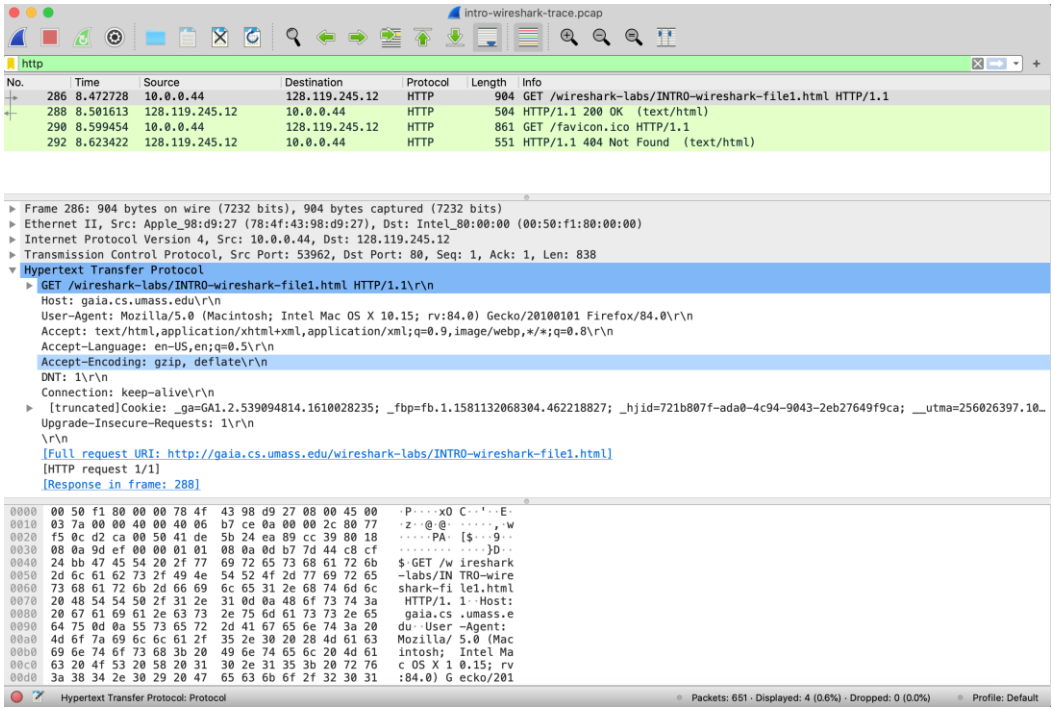

Figure 5: looking at the details of the HTTP message that contained a GET of http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

8. Find the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. (Look for an HTTP GET message in the "listing of captured packets" portion of the Wireshark window that shows "GET" followed by the gaia.cs.umass.edu URL that you entered.) When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window. By clicking on '+' and '-' or right-pointing and down-pointing arrowheads to the left side of the packet details

window, *minimize* the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. *Maximize* the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 5.

9. Print the two HTTP messages (GET and OK) into a PDF file. To do so, select Print from the Wireshark File command menu, and select the "Selected Packet Only" and "Print all expanded", and then click "Print". In the pop-up window, select "Save as PDF", as shown in Figure 6 below.

10. Exit Wireshark

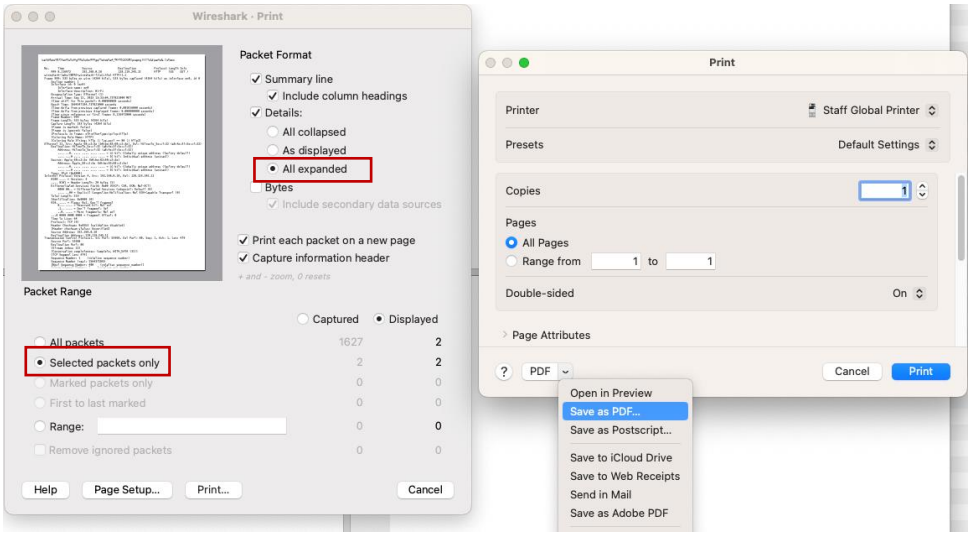

Figure 6: Print the selected messages.

1.5 Wireshark Lab Questions

Q1. Which of the following protocols are shown as appearing (i.e., are listed in the Wireshark "protocol" column) in your trace file: TCP, QUIC, HTTP, DNS, UDP, TLSv1.2?

Q2. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. (If you want to display the Time field in time-of-day format, select the Wireshark *View* pull down menu, then select Time *Display Format*, then select *Time-of-day*.)

Q3. What is the Internet address of the gaia.cs.umass.edu? What is the Internet address of your computer that sent the HTTP GET message?

Q4. Expand the information on the HTTP message in the Wireshark "Details of selected packet" window so you can see the fields in the HTTP GET request message. What type of Web browser issued the HTTP request? The answer is shown at the right end of the information following the "User-Agent:" field in the expanded HTTP message display. [This field value in the HTTP message is how a web server learns what type of browser you are using.]
    - Example: Firefox, Safari, Microsoft Internet Edge, Other

Q5. Expand the information on the Transmission Control Protocol for this packet in the Wireshark "Details of selected packet" window so you can see the fields in the TCP segment carrying the HTTP message. What is the destination port number (the number following "Dest Port:" for the TCP segment containing the HTTP request) to which this HTTP request is being sent?

Q6. If you enter the URL: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html multiple times during Wireshark packet capture, are there any difference in the sent HTTP message and the received HTTP reply?

Part 2 – Socket Programming

2.1 Programming requirement

In this part, you'll write a simple chat program relying on TCP as the transport protocol.

There are two users in this system, one in the client end (user 1) and one in the server end (user 2). Here's what your client and server should do:

- Your client should first read the user 1'name from the input console and store it in a variable. Then it will open a TCP socket to your server and send a message which contains user 1's name.

- Your server will first read user 2's name from the input console, store it in a variable, and then begin accepting connections. On receipt of the first message (the name of user 1) from the client, your server should extract the name and store it in a variable and send user 2's name to the client.

- After receiving user 2's name, the client extracts and saves it in a variable. Then, they start chatting in turn, i.e., one side sends a message, waits for a reply from the other side, and then proceeds to send the next message. We assume that the first message (after the names) is sent by the client. The output should show whose turn it is on both sides. For instance, if user 1's name is "Bob" and user 2's name is "Alice", when it is Alice's turn to write, both client and server sides should show this by printing Alice's name at the start of the line.

- The chat finishes once one side sends or receives a "bye" message.

Except for the sending and the receiving messages, you should program your client and server to **print informative statements whenever they take an action** (e.g., open a socket, close a socket, detect termination of input, etc.). This not only helps you monitor whether your processes are working correctly (or not!) but also enables TAs to determine from this output if your processes are working correctly.

2.2 Programming Languages and Operating Systems

Python will be used for programming in this course on any platform (Linux/Unix, Mac, PC) you want. The textbook is a good starting point for socket programming in Python. You can find Python instructions for simple socket programming in Chapter 2.

A Python socket tutorial is http://docs.python.org/howto/sockets.html

If you find better pages for Python, please share with the class!

2.3 Programming notes

Here are a few tips/thoughts to help you with the assignment:

- You must choose a server port number greater than 1023 (to be safe, choose a server port number larger than 5000). If you want to explicitly choose you client-side port, also choose a number larger than 5000.
- You may need to know your machine's IP address, when one process connects to another. You can telnet to your own machine and see the dotted decimal address displayed by the telnet program. You can also use the UNIX nslookup command. On Windows, see the ipconfig utility. On a Mac, you can run the terminal program and use the ifconfig command (just type in `ifconfig` or `ifconfig | grep "inet ")`.
- Many of you may be running the clients and servers on the same machine (e.g., by starting up the server and running it in the background, and then starting up the client. This is fine, since you are using sockets for communication, these processes can run on the same machine or different machines without modification. You can use either "localhost" or IP address of your machine as the server address in this case.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error.

Submit a lab report that includes the following:

- **Your name and UCID #**
- **Your answer to questions Q1 – Q6**
- **The PDF file with the printed HTTP messages. Mark/Highlight where in the message you've found the information that answers a question.**
- **Python files for client and server for Part 2, and also the screenshots of the output of the client and server.**