CS2911 Lab Exercise: TCP

Names: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Work with teams of two on this exercise. Fill in your answers within this file. Submit your completed exercise to Canvas by the due date. Only a single copy for your group needs to be submitted.

1. Fire up Wireshark on your computer and start capturing. Browse to a webpage that is not encrypted (e.g. http://apache.org/). Filter with the http filter and select the first packet in the request.  
   Turn off auto-scroll .  
   Now turn off the http filter by clearing the http filter and pressing ENTER. The same packet should be selected, but with other packets around it. If helpful, right-click on the packet and select follow -> TCP Stream to see just the packets that are part of that TCP connection or use a filter like ip.addr == 95.216.24.32, using the IP address of the server to only see packets headed for that destination.  
   1. Identify where the TCP source and destination ports appear within the hexadecimal packet data. Look at a TCP connection to a web server. Write the destination port (on the server) in both decimal and the hexadecimal for the entire field.
   2. Identify the SYN packet sent to the client in response to the first message from the client. Does the packet have the SYN value you expect?
   3. Identify the second packet from the client to the sender. Does it have the SYN and ACK values you expect?
   4. Can you see any other TCP packets to the same server?
   5. Explore the other fields in the packet. What questions do you have about them?

# Download the code provided in Canvas. This code will send data between you and your lab partner. You can also set it up to send through localhost on your machine. To do that you will need to modify the PyCharm configuration to allow multiple instances of the script to run at the same time. This can be done by editing the run configuration for the script in the (upper left of PyCharm). If you need assistance, ask your instructor.

# Fire up Wireshark and select either Wifi (if sending between your computer and your lab partner’s) or loop-back interface (if using localhost) for tracing. There might be other traffic on the interface. It can be beneficial to filter by an IP address. For example, to filter by IP address 9.10.11.12 type the following in the Wireshark filter field. Don’t forget to press enter to activate the filter. ip.addr == 9.10.11.12

# With the capture running, invoke the server (send) and client (receive). Locate the SYN, SYN/ACK, ACK three-way handshake, and the pair of FIN, ACKs at the end of the conversation. Also note the Data portion of the TCP packets and locate the actual messages exchanged.

# Pick out a packet that sent data. Write the sequence number as displayed in Wireshark. Predict the sequence number for the next packet sent from the same source. Find the acknowledgement of this packet. What is its acknowledgement number?

# *Write* the sequence number of the packet you picked: \_\_\_\_\_\_\_\_\_, data len:\_\_\_\_\_\_\_

# *Write* your predicted sequence number for the *next* packet: \_\_\_\_\_\_\_\_\_. Is it correct? If not, fix it: \_\_\_\_\_\_\_\_\_.

# Explain, why is this the sequence number for the next packet?

# *Find* the acknowledgement packet in your Wireshark stream.

# *Write* the acknowledgement number for the packet acknowledging this data: \_\_\_\_\_\_\_\_\_\_\_\_\_\_.

# Explain, why is this the acknowledgement number for the next packet?

# In the segments containing data, note the win=XXXXX. This is the window advertisement and hence the amount of data the sender of that packet is able to receive.

# What is the window size one of the data packets?

# Does this number make sense? Why or why not?

# Now, modify the code on the receiver:

# Set the named constant PRINT\_RESPONSE to True. This causes the receiver to print each byte it receives on a separate line.

# As before, capture packets from this transaction. If you are not seeing any evidence of overwhelming the network or the receiver, set the named constant DO\_SLEEP to True. This causes the receiver to sleep 1 second between each call to recv(1). Run the capture and programs again. At this point you should see the window advertisement reach 0 and effectively halt transmission. Let everything run for a minute or two and observe the capture.

# What is the typical amount of data transferred (Len in Wireshark)?

# What is the window advertisement of the receiver at the start (first data ACK, not in any of the SYN, SYN/ACK, ACK packets)?

# The window advertisement should decrease as data is received. How much does the window size decrease by? Is this what you expect? (If not, what's going on here?)

# TCP Protocol

From what you learned in this exercise fill in the blanks in the following TCP stream (the numbers are above the arrows they describe)

Client

Server

C: SYN, SEQ = 8426

S: ACK, ACK = \_\_\_\_\_

SYN, SEQ = 4029

C: ACK, ACK = 4030

SEQ = \_\_\_\_\_\_\_\_

S: ACK, ACK = 8427

SEQ = \_\_\_\_\_\_\_\_

LEN = 0

C: ACK, ACK = 4030

SEQ = 8427, LEN = 10

C: ACK, ACK = 4030

SEQ = \_\_\_\_\_, LEN = 10

S: ACK, ACK = 8437

SEQ = \_\_\_\_\_, LEN = 0

S: ACK, ACK = \_\_\_\_\_\_\_

SEQ = 4030, LEN = 0

C: FIN, ACK, ACK = \_\_\_\_\_\_\_

SEQ = 8447, LEN = 0

S: ACK, ACK = \_\_\_\_\_\_\_

SEQ = \_\_\_\_\_\_, LEN = 0

S: FIN, ACK = \_\_\_\_\_\_\_

SEQ = \_\_\_\_\_\_, LEN = 0

C: ACK = \_\_\_\_\_\_\_

SEQ = \_\_\_\_\_\_, LEN = 0