Lab 2

[Due: your class day of **Sep 25/26/27/21**]

Note: in all the lab questions, please include the screen shots as evidence of your solutions. You can take one screen shot for multiple questions but need reference to that screen shot.

1. Command **dig** is used to find the ip address of a hostname such as www.google.com. To do this, you can simply run **dig www.google.com**. You might see the following result.

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         2694    IN      A       172.217.1.164
```

This is the result returned by your local DNS server when you run **dig** command to request it to find out the ip address of www.google.com. Question section is to repeat your question and **A** stands for the record of IP address. We can see that www.google.com has an ip address 172.217.1.164.

a. Try **$ dig www.example.net** to find out its ip address.

b. run Wireshark on your VM, then **$ dig www.example.net** and stop wireshark. Look at the DNS request packet (using filter DNS to find it easily), confirm that the transport layer protocol is UDP. What are the values of this UDP header (you need to first check the header fields learned in class)?

Note: You might need to clear the DNS cache, using **$ sudo systemd-resolve - - flush-caches**

c. In the DNS request packet in step **b**, the destination IP is your local DNS server's IP. What is this value? As said, DNS is serviced by UDP and has **no** connection setup before sending DNS request. You can confirm this by checking that there is no any packet in Wireshark exchanged between your VM and local DNS server, prior to the DNS request packet (show the screen shot of the window of Wireshark for the list of packets).

2. Run Wireshark and then access **www.example.net** using Firefox (you might need to clear the browser history). Then stop the Wireshark. Check your list of packets in Wireshark window, filtered by the ip address of **www.example.net**. You can see that before the HTTP request to **www.example.net**, there is a connection stage with three packets: SYN packet, SYN-ACK packet and ACK packet. This is to provide the connection setup between your VM and **www.example.net**. Confirm this. Also, confirm that the transport layer protocol in these packets (check one of them is good enough) is TCP. When the message exchange starts, you can

see ACK packet. This is to confirm the receipt of a packet. Find out such a packet. This is to find a packet with flags bit A=1. This provides an evidence that TCP is a reliable protocol. This is different from the UDP protocol. ACK packet might or might not contain the application data. Verify the ACK packet you consider (any of them is ok) to see if it contains application data.

3. Run Wireshark and access **www.example.net** and then close your webpage and stop your Wireshark. Answer the following questions.

a. Find out the first packet from your VM to **www.example.net** (you should know the ip address of www.examplenet now). This should be the SYN-packet (i.e., the first packet of the 3-way handshake protocol). What is source port # and destination port #? Confirm that they are in the TCP header in the Wireshark packet window. What is source IP and destination IP? Confirm that they are in the ip header in the Wireshark packet window.

b. Look at SYN-packet. What is the sequence #? It is a random number. Confirm this.

c. Find out in the TCP header the flag bits U|A|P|R|S|F in the SYN-ACK packet.

d. The receive window field is to tell its partner the current **receive-buffer** size it has. Find out the window size of SYN-ACK packet and that of http response packet. Are they equal?

e. Find out the sequence # of http request packet and its payload size (the **segment len** is the payload size). The next sequence # is the sum of these two numbers. Verify that this is indeed the sequence # of the next packet sent by your VM.

f. Find out the acknowledgement # in http response packet. Is this the same as the next sequence # you calculated above for the request packet? Explain why?

g. What is the flags bits U|A|P|R|S|F in the http response packet?

h. Find out the packet your VM requests to terminate the TCP connection. This packet will be sent when you close the webpage. What is the flags bit U|A|P|R|S|F in this packet?

4. (this problem due with Lab 3)

Write a Python TCP server program that will accept an unlimited number of connections, one at a time, just as in the sample program in the lecture. Upon receiving a TCP connection request, it should reply with the client's IP address and port number. Then, it waits for commands from the client. Valid commands are "TIME", and "EXIT". To the *TIME* command, the server should return the current time (see the example below how to obtain a time string). To an invalid command (.e.g, HELLO), it returns string "Invalid command!". If the client closes the connection or does not send a command in 15 seconds (see below how to set a timeout for a socket), the server closes the current connection and waits for another connection. To the *EXIT* command, your server closes all open sockets (including the welcome socket).

A sample client program is attached to the assignment. You can use it to test your server. You can modify the client with your own sequence of commands.

Submission requirement:

- Your server program
- The sequence of commands from the client sides (no need to include your modified client program)
- Screen shot on the running result at the client side (which should include the printout of all the server messages).
- Screen shot on the **packet list** (no packet details required) between client and server. If you run client server on the same VM, you may need to run Wireshark on the **loopback** interface.
- All the above are put in a **single** assignment solution file.

Example of obtaining a time string

Here is an interactive Python session showing how to come up with a string containing the current time of day using the time.ctime() method from the time module.

```
>>> Import time
>>> now=time.ctime()
>>> now
'Tue jan 10 13:29:47 2023'
>>
```

Set a timeout for a socket

This example shows how to set a timeout on a socket used to accept incoming connections and how to detect a timeout events as an exception.

```
import socket

ss = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ss.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
#Note: SO_REUSEADDR will avoid port number re-use conflict when you
#start the TCP socket program several times in a short time.

ss.bind(('', 4000))
ss.listen(1)

while True:
    print "waiting for connection"
    new,addr = ss.accept()
    new.settimeout(10)
    msg = 'init'
    while len(msg):  #when you close client socket, len(msg)=0.
        try:
            msg = new.recv(30)
        except socket.timeout:
            print "got timeout"
            break
        print msg
    new.close()
ss.close()
```