CSE310 Programming assignment 2 Summary/Explanation
Jacky Chow 113268425 03-11-2022 Submitted around 11:59 PM 3-11-2022
This is submission attempt number TWO, grace period used for this is 27 hours from 3-10-2022 9:00 pm. Please grade the latest submission, thanks.
(Total grace hours used: 1 Hour from PA1 and 27 hours from PA2 = Total: 28 hours used)

I have tried my very best to explain how I got my results and my thought process for this assignment and looking for possible partial credits since the code can possibly be confusing. Even though I was not able to complete part b, but I hope my explanations for Part B helps.

Part A) a
In general, a complete flow is identified by the SYN and FIN packets within the PCAP file. So every SYN packet signifies a new flow that has been created. So in order to abstract the idea of finding completed flow, I have only used the SYN packets to show a flow that existed in the PCAP file. Because there will be a possibility that the FIN packet being lost, and not exist due to different reasons. As a result, the assumption and my definition of a flow depend on the number of SYN packets that exist in the PCAP file.

I have used a dictionary to store all the sender source ports as keys, so the dictionary keys will consist of 43498, 43500, 43502 for the three flows, in order to achieve that I have stored all the SYN packet's source ports as the key of my dictionary. Since the SYN packets indicate the general flow information, which includes the source port, receiving port, source IP, sender IP. Also, we are only interested in TCP packets, so, therefore, internet (HTTP) packets, so the receiver end of the port will always be 80, so port 80 is also included in the dictionary key.

In order to show the general information about the flow, I iterated the dictionary keys to obtain its information, such as the ports and Ip addresses. I have collected the packet IP addresses in a separate list for future use (part A (b)).

```
print("Source port: " + str(key) + " Destination port: " + str(tcp.dport)
+ " Sender IP: " + str(sip[i]) +
                  " Receiver IP: " + str(rip[i]) + ' Timestamp: ' +
str(times[i]))
```

Part A) b
In order to retrieve the first two transactions. I have to loop through all the packets within each key in the dictionary I have created in part A a). After analyzing the PCAP behavior using WireShark, the first send packet of the send transaction will be the 3rd send (130.245.145.12) to receive (128.208.2.198) packet of the entire PCAP file for all three flows. To accommodate for the three-way handshake - (SYN, SYN/ACK, ACK). Because the first send to receive packet will always be the SYN packet, which is the start of the flow, then the second send to receive packet will be the ACK packet. Since the SYN/ACK packet will be in the port 80 key because it is a packet is is from the receiver to the sender.

As the three-way handshake is established, so the following send packet will be the 3rd packet as previously mentioned. Consequently, the 4th packet will be the second send packet used for the second transaction. Then I will get the appropriate attributes of the packet which include the sequence number and ack number.

Then I will get the corresponding returning packet by matching the sequence number and its port numbers. So if the receive to send packet has the same sequence number as the sender to receive packet's acknowledgment number, and if the send to receive packet's source port is equal to the destination port number, then it will be the correct corresponding packet. However, there are many packets that have the same sequence number and the same port numbers, so we do not want to use the very first returning packet as we go through the dictionary. So every time we match the packet. We place them into a separate list and use that list to cross-check for the second transaction. So there won't be any duplicate return packets being shown by mistake.

Finally, as we get all the send and received packets for each flow, we will print the appropriate attribute for each packet, including sequence number, acknowledgment number, sour and destination ports, and its window size.

The following loop is used to find the receiving packet for transaction one.

```
            tempcount = 0
            templist = []
            for (ts, ip) in port_dict[80]:
                tempcount += 1
                if ip.data.seq == port_dict[key][2][1].data.ack and key ==
ip.data.dport:
                    new = ip
                    templist.append(ip.data.ack)
                    break
```

Part A) c
I did not use the dictionary from parts A) a, b. Since it was rather difficult to iterate through all the send packets in each flow in the dictionary. So instead, I created a list that contains all of the send packets and a list of each key (the three ports), and an index variable to iterate through each port/flow, and the resulting packet starting with the value 0 with the size of the number of flows (3).

Part A) c continued next page.

I will iterate through the list with all the send packets, and start to add the bytes or their len(TCP) to the corresponding keys/flow. We only want to add the packets after the connection and before the connection tear down. So the code is as follows.

```
        j = 0
        i = 0
        for i in lista:
            if(tcp.sport == lista[j] and not ((tcp.flags &
dpkt.tcp.TH_FIN)) and not ((tcp.flags & dpkt.tcp.TH_SYN))):
                listb[j] = listb[j] + len(tcp)

            j += 1
```

After obtaining the resulting bytes for the throughput, I have to find the time it took to send all of them for each flow. So I have created a new dictionary for the time that went by. Then within the same loop, we will add the timestamp value for the SYN packet into the dictionary. Finally, we want to subtract the FIN packet's timestamp from the SYN packet, ultimately getting the time that went by for the flow.

```
if tcp.flags & dpkt.tcp.TH_FIN and tcp.sport in lista:
            elapsed_times[tcp.sport] = timestamp -
elapsed_times[tcp.sport]
            print(elapsed_times)
```

Part B)
Unfortunately, I was not able to implement part B at this submission attempt.

My thought process for getting the congestion window size is to obtain the RTT by finding the time difference between the SYN and the SYN-ACK packet. Which I was able to do so with the timestamp difference.

```
# Adding the RTTs into the RTTlist
    RTTlist = []
    o = 1
    for o in range(len(SYNACKtimes)):
        RTTlist.append(SYNACKtimes[o] - SYNtimes[o])
    RTTlist.remove(0)
    print(RTTlist)
```

Then check how many packets were sent for each RTT, then repeat this process 3 times to obtain the 3 congestion windows for each flow.