Overview

The analysis_pcap_tcp program analyzes a PCAP file to generate information about the different TCP Flows in the trace. Though the code utilizes the provided “assignment2.pcap” file for the analysis, one can easily make changes for it to work with another PCAP file, subject to a wireshark trace that was captured between two hosts.

A copy of the code can be found at the github repository here.

Part A

In the first part of the assignment, we were interested in capturing the TCP flows between nodes 130.245.145.12 (sender) and 128.208.2.198 (receiver) [ip addresses that we hard coded into the program]. The library of choice for us was **dpkt**, and as such, after opening the file in “rb” (read binary) mode, we created a **dpkt** reader object **pcap** which enabled us to iterate through each of the packets, extracting both the timestamp and buffer (buffer in this case means the packet or frame that was captured). We then passed the buffer object into: **eth = dpkt.ethernet.Ethernet(buffer)**, which took the raw bytes of the buffer and returned an ethernet frame object, on which we can extract relevant information.

With the ethernet frame object (link layer), we were able to fetch a network layer object through **ip = eth.data**, and as well, a transport layer object through **tcp = ip.data**, giving us an abstraction that mimics the layers of the OSI model studied in class. With these objects, we did checks to make sure the ethernet frame had an ip component, in addition, checking to make sure the ip object had a tcp component. From there we made use of these objects along with the **socket.inet_ntoa()** function to extract the source/destination ip addresses as well as the source/destination port numbers for the flow, in addition to the transport layer protocol in use. With these values, we defined what distinguished a given TCP flow from another, with the tuple (Source IP Address, Source Port, Destination IP Address, Destination Port), but we made sure to sort the (ip_addr_1:port_1, ip_addr_2:port:2) combination such that ip_addr_1 is smaller than ip_addr_2, because packets could be sent in both directions, and we wanted to associated all relevant packets to a given flow.

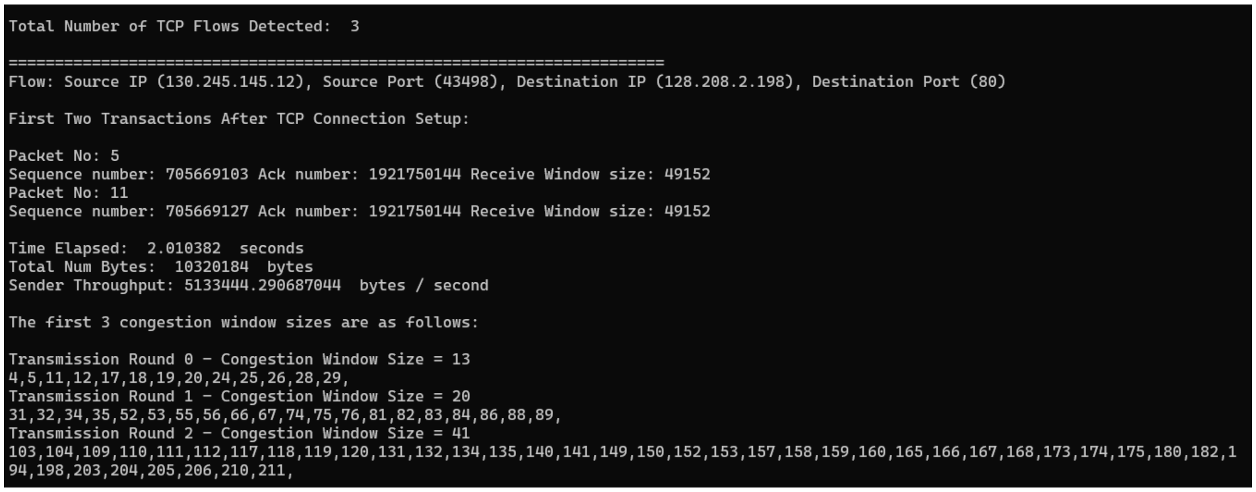
We then made sure to define what information we wanted to store about each packet in the trace. We called this variable **packet_info** in the code, and some fields worth including were the packet_number, the time_stamp, values from the four tuples as shown above, sequence and acknowledgement numbers, details about which flags were set, and many more. Though we didn’t include the entire buffer data, we did make sure to include at least the size of the tcp segment. From there, since we were only interested in capturing sender initiated TCP flows, we did a check for this, and then we did a lookup of a particular packet's tcp flow id by the tuple, and then added the packet to the corresponding flow. It is worth noting that along with the list of packet details associated for a given flow, we made sure to store what we referred to as flow **meta_data**, and more specifically the window scale values in both directions as established during the SYN and SYN/ACK packets exchanged as part of the TCP connection setup. For part 1b of the assignment, among the reporting of "Sequence number, Ack number, and Receive Window size" for the first two transactions after the TCP connection setup, while sequence number and ack number were easily fetched from the buffer, the receive window size was less straightforward. Still, we understood that by "receive window size", this meant the "**calculated window size**" as reported in wireshark. To fetch for these, when parsing each packet, we made sure to check if was a SYN or a SYN/ACK, and if so, we looked at the **TCP Options**, and associated **window scale** values 'receiver_to_sender_window_scale' for SYN-ACK and 'sender_to_receiver_window_scale' in case of SYN. After writing logic to capture the first two sender transactions after the TCP connection setup, we reported for each packet, the packet number, sequence number, ack number, and received window size (aka how much space does the sender have in its buffer). This was computed through the formula **calculated_window_size = trans['window_size'] * (2 ** ws)**, where trans['window_size'] is the window size as reported by tcp.win and is the captured direction specific window scale value, giving as a result, a number to indicate to the recipient how much space it has in its buffer. With regards to the calculation for total sender throughput, when parsing each packet, we made sure to include use a variable **sender_tcp_bytes_total** to keep track of this, and divided by the total time elapsed (the time the sender first sent a SYN packet to the time it last received an ack, which is typically the FIN ACK packet last received by sender).

Part B

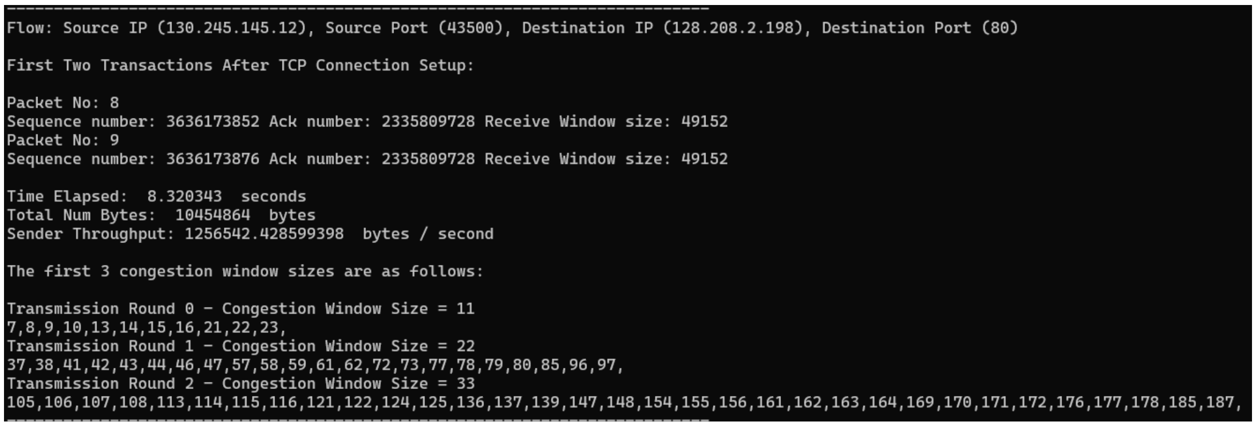
The logic for part b of the assignment is found in **run_congestion_control,** a function we invoked in our run function for each flow. Here, we first iterated through the packets of the flow, fetching the SYN and SYN-ACK packets, from which we were able to determine the estimated round trip time via **estimated_RTT = (syn_ack_time - syn_time).total_seconds().** From there, we used two pointers to keep track of the congestion window boundaries, and specifically we used the time stamps to demarcate these, in variables **lower_congestion_window_time** and **upper_congestion_window_time.** Since we were only interested in the first three congestion window sizes, we used a variable **transmission_round_number** to indicate which RTT were currently in, and reported the packets for that round in

window_packets[]. We used **lower_congestion_window_time** to mark the lower end for a window, and incremented **upper_congestion_window_time** one packet at a time and determined whether the difference in times were at most the estimated RTT computed earlier. We then moved the markers to the next “window” via lower_congestion_window_time = upper_congestion_window_time once a particular packet was not within that RTT, and reported the results.

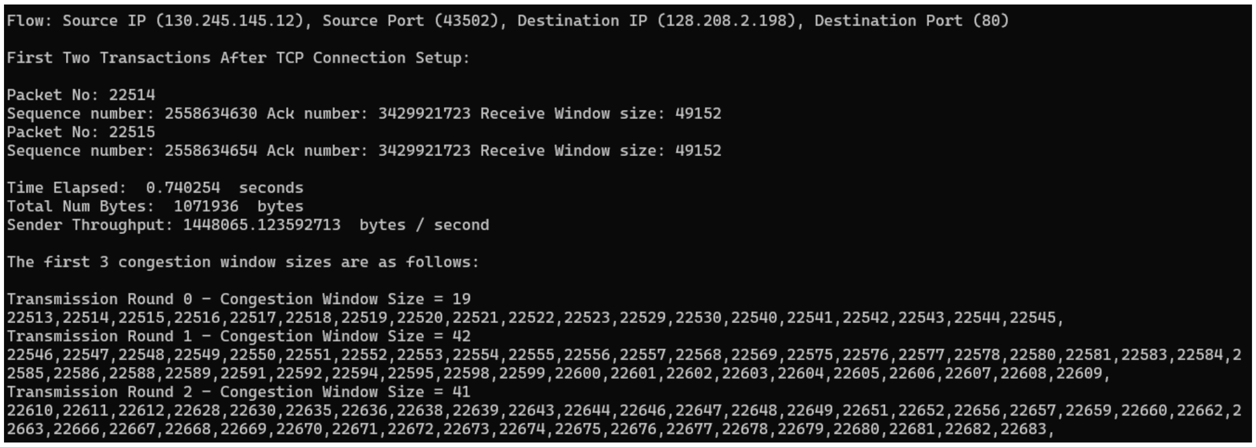
Example output is below



```
Total Number of TCP Flows Detected:  3

======================================================================
Flow: Source IP (130.245.145.12), Source Port (43498), Destination IP (128.208.2.198), Destination Port (80)

First Two Transactions After TCP Connection Setup:

Packet No: 5
Sequence number: 705669103 Ack number: 1921750144 Receive Window size: 49152
Packet No: 11
Sequence number: 705669127 Ack number: 1921750144 Receive Window size: 49152

Time Elapsed:  2.010382  seconds
Total Num Bytes:  10320184  bytes
Sender Throughput: 5133444.290687044  bytes / second

The first 3 congestion window sizes are as follows:

Transmission Round 0 - Congestion Window Size = 13
4,5,11,12,17,18,19,20,24,25,26,28,29,
Transmission Round 1 - Congestion Window Size = 20
31,32,34,35,52,53,55,56,66,67,74,75,76,81,82,83,84,86,88,89,
Transmission Round 2 - Congestion Window Size = 41
103,104,109,110,111,112,117,118,119,120,131,132,134,135,140,141,149,150,152,153,157,158,159,160,165,166,167,168,173,174,175,180,182,1
94,198,203,204,205,206,210,211,
```



```
----------------------------------------------------------------------------
Flow: Source IP (130.245.145.12), Source Port (43500), Destination IP (128.208.2.198), Destination Port (80)

First Two Transactions After TCP Connection Setup:

Packet No: 8
Sequence number: 3636173852 Ack number: 2335809728 Receive Window size: 49152
Packet No: 9
Sequence number: 3636173876 Ack number: 2335809728 Receive Window size: 49152

Time Elapsed:  8.320343  seconds
Total Num Bytes:  10454864  bytes
Sender Throughput: 1256542.428599398  bytes / second

The first 3 congestion window sizes are as follows:

Transmission Round 0 - Congestion Window Size = 11
7,8,9,10,13,14,15,16,21,22,23,
Transmission Round 1 - Congestion Window Size = 22
37,38,41,42,43,44,46,47,57,58,59,61,62,72,73,77,78,79,80,85,96,97,
Transmission Round 2 - Congestion Window Size = 33
105,106,107,108,113,114,115,116,121,122,124,125,136,137,139,147,148,154,155,156,161,162,163,164,169,170,171,172,176,177,178,185,187,
----------------------------------------------------------------------------
```



```
Flow: Source IP (130.245.145.12), Source Port (43502), Destination IP (128.208.2.198), Destination Port (80)

First Two Transactions After TCP Connection Setup:

Packet No: 22514
Sequence number: 2558634630 Ack number: 3429921723 Receive Window size: 49152
Packet No: 22515
Sequence number: 2558634654 Ack number: 3429921723 Receive Window size: 49152

Time Elapsed:  0.740254  seconds
Total Num Bytes:  1071936  bytes
Sender Throughput: 1448065.123592713  bytes / second

The first 3 congestion window sizes are as follows:

Transmission Round 0 - Congestion Window Size = 19
22513,22514,22515,22516,22517,22518,22519,22520,22521,22522,22523,22529,22530,22540,22541,22542,22543,22544,22545,
Transmission Round 1 - Congestion Window Size = 42
22546,22547,22548,22549,22550,22551,22552,22553,22554,22555,22556,22557,22568,22569,22575,22576,22577,22578,22580,22581,22583,22584,2
2585,22586,22588,22589,22591,22592,22594,22595,22598,22599,22600,22601,22602,22603,22604,22605,22606,22607,22608,22609,
Transmission Round 2 - Congestion Window Size = 41
22610,22611,22612,22628,22630,22635,22636,22638,22639,22643,22644,22646,22647,22648,22649,22651,22652,22656,22657,22659,22660,22662,2
2663,22666,22667,22668,22669,22670,22671,22672,22673,22674,22675,22676,22677,22678,22679,22680,22681,22682,22683,
```