# <u>Summary</u>

## Part A

**1. The number of TCP flows initiated from the sender**

For assignment2.pcap, I calculated 3 flows being initiated from the sender.

** I implemented my analysis_pcap_tcp program by firstly collecting all the packets in a list. Packets[ ]

The way I found the number of tcp flows was by looping through the packets list and checking which packets had a SYN flag, without an ACK flag. The presence of the syn flag without the ack flag represents the first packet of the three way handshake. It also represents a packet from the 'sender', since in a three way handshake:

1. The **sender** firstly sends a packet with SYN flag(without an Ack), then

2. The **receiver** sends a packet with SYN and ACK, then

3. The **sender** finally sends a packet with ACK (without or without piggybacking the 1st data packet)

Therefore by identifying the first packet (1) of the three way handshake, I was able to get the number of three way handshakes, and that in turn indicates the number of tcp flows being initiated from the sender.

**2. a) Source port, source IP address, destination port, destination IP address**

The source and destination, port and IP addresses are information that is present in every packet.

To get the unique values for each flow, I extracted this information from the first packet of the flow. Since the first packet is the SYN packet from the three way handshake (as described above), I was able to identify the sender and receiver. Once I had identified the sender and receiver, it was pretty simple to extract the actual sender and destination port and address numbers from the packet using the buffer.

In my implementation, I made a Packet structure and was able to extract all the information about that packet using the buffer. I got the source and destination IP addresses from the Internet Protocol (IP) layer at buffer[26:30] and buffer[30:34] And I got the source and destination ports from the TCP layer at buffer[34:36] and buffer[36:38]

**2. b) First two transactions after the TCP connection is set up**

It was fairly easy to get the first two ‘sent’ packets in a flow by going through the list of packets in each flow and getting the first two packets that had a payload (data is being sent over) and had the same source and destination ports as the first SYN packet.

Having the same source and destination port as the first SYN packet verified that the packet is being sent from the sender to the receiver. (Just like the first packet in the three way handshake). And by checking the payload I was able to ignore the first two packets of the three way handshake and possibly check if the last ACK from the sender had piggybacked data with it.

In my program, I implemented a transaction map to keep track of every transaction. The ‘received’ packet, which is basically an acknowledgment back from the receiver was found by adding the sequence number and the size/length of payload of the sent packet.

I only took the first two sent packets that had received an ack back (received packet in its transaction map), therefore accounting for lost packets.

Receive window size was calculated by multiplying the relative window size of the packet with the scaling factor (which is information already available in the syn packet).

**c) Sender throughput**

While sorting the packets into different respective flows, I added the lengths of the tcp segments of all data packets (This includes the tcp header and the tcp payload).

I was able to find the first data packet being sent after the connection setup (three way handshake) whether it was piggy backed in the handshake or otherwise, and started adding the tcp segment length from there onward all the way till the last data packet sent from the sender to the receiver. This was the total number of bytes sent from the sender.

The time was calculated from the timestamp of the first data packet sent till the last acknowledgment received from the receiver. Total bytes / total time = total throughput.

## Part B

### 1. First 3 Congestion window size

The congestion window size was calculated by looping through all the packets of a flow and if I encountered a sender packet I stored its acknowledgement number in a list. (Acknowledgment number was once again found by adding the packet’s sequence number and length of its payload).

And if I found a receiver packet I would check if its acknowledgment number was equal to any one in the list of acknowledgment numbers I had stored. If yes I would count the length of the list of acknowledgment numbers and print that. Then I would clear the list and continue looping, again adding sender packets' acknowledgment numbers to a list and clearing it once I found a corresponding receiver packet acknowledging it.

The length of the acknowledgment numbers list represented the number of packets sent without being acknowledged yet. This would be the number of packets in the congestion window size.

In assignment2.pcap, the first three congestion window sizes are [10, 20, 33]. This is the TCP tahoe approach. The initial congestion window size is 10, then it increases to 20 packets. This is a 'slow start' where the congestion window size is doubled until it reaches a preset threshold. From there on, the congestion window would be linearly increased by a small factor (additive increase) which is the congestion control phase. However, since the 3rd congestion window is not the double of the last cwnd, this could indicate either that the threshold was met or maybe enough bandwidth was not available. If a packet is lost, then the cwnd would start again from the initial cwnd value and threshold would be halved.

## 2. Retransmissions due to triple duplicate acks and timeout

As mentioned above, I implemented a transaction map to keep track of each sent packet and its ack received back from the receiver. This was done by a transaction object, which also had a counter inside it. Every time I got an ack from the receiver, I found the corresponding transaction object in the map using the ack number and increment the counter.

Keeping a track of the number of acks received by each sent packet led me to recognize when this counter exceeded 3. When there were 4 acks in the counter (implying three duplicates) i set a flag in this transaction object and was able to identify if the sender sent the packet again (retransmission)

I was able to verify the sender retransmitted the packet again by checking if the packet had a corresponding transaction in the map already. (The transaction would be mapped at the ack number of this packet with a flag set if it had received more than 3 acks) Therefore getting the number of retransmissions from triple dup acks.

Lastly, for retransmissions due to timeout, I subtracted the retransmissions due to triple duplicate acks from the total number of retransmissions.