Summary

Part 1:

The number of TCP flows initiated from the sender :
Every time we see a syn that sends from sender to receiver, we create a new TCP_flow object, and then we add 1 to a global counter, in my case, it is called TCP_flow_count.

For each TCP flow:
(a) Write down the (source port, source IP address, destination port, destination IP address)

Source IP and destination IP can be obtained by using the dpkt library, but they are not readable when we get them by using the library, so I used socket.inet_ntop method in the Socket library to convert the unreadable form to readable strings. Then we can get the readable source port and the destination port by just using the dpkt library without converting.

b) For the first two transactions after the TCP connection is set up (from sender to receiver), the values of the Sequence number, Ack number, and Receive Window size.

What I did is In my TCP_flow class, I put an empty list there, which is used to store all the transactions, and I also made a transaction class that contains its seq, ack, win, and some other properties that are used for other purposes. After the SYN packet are sent, and the sender and receiver are connected, we then check if the type of the transaction is ACK by the flags of the transaction, if it is ACK, then we check if there are any duplicate transactions in the same ports, if there are duplicates, we will not add it to the transaction list, if there are not duplicates, we just add it to the transaction list.
Once we go through all the transactions, we can then just print out the first two transactions of each flow, their seq, ack, and win can all be printed easily since each transaction is an object that contains all the information we need.

(c) The sender throughput.

Sender throughput is the total bytes from sender to receiver divided by the total amount of time from the start of the sender to the end of the sender. For the start time, I just save the timestamp when the SYN packet is sent from sender to receiver to a variable in the flow class called start_time, and then the end_time is saved when the sender received one more ack from the receiver after the sender sends a FIN to the receiver, so I also saved that timestamp to the variable in the flow class called end_time. The total bytes part is very simple, I have a pkt_length variable in my flow class, which will contain the total bytes from sender to receiver, so every time when we have one ack packet from sender to receiver, if it is not duplicate, we add its payload length to the pkt_length variable, and finally, we calculate the throughput at the end, using the total bytes from sender to receiver divided by the total amount of time from the start of the sender to the end of the sender.

Part 2:

Congestion Window

The way I calculate the congestion window is by having a counter for the congestion window called cwnd_count and a list for storing the counts called cwnds in the flow class, whenever I see an ACK that is from sender to receiver that is not duplicate, I add one to the counter cwnd_count, and if I see a FIN packet or an ACK packet from the receiver to sender, then I append the counter into the list cwnds, then I double the counter. In the end, we just print the first three numbers in the cwnds list.

Total retransmission

Every time we see a duplicate ACK packet that we already have in the transactions list from sender to receiver, we add one to the variable retransmission in the flow class.

Timeout retransmission

I calculate the RTT for the SYN packet, then double it and use it as an estimation to the RTO, then the start time of retransmission minus the start time of the original packet is the time period we want to check, if it is greater than RTO then it is time out retransmission, if it is not, then it should be other types of retransmission. Also, every time when I have a duplicate, we need to reset the original transaction's start time to the retransmission start time, this is done after checking if it is greater than RTO or not.

Triple Duplicate ACK retransmission

For Three Duplicate ACK, first, we check that each ACK packet from receiver to sender has the same ack number as the previous ACK packet from receiver to sender, I used a variable in the TCP_flow class to save the previous ACK packet from receiver to sender, which called receiver_prev_transaction, if it is the same, we add 1 to a counter called receiver_duplicate_ack in the TCP_flow class, we use this number to determine when the number of duplicates is equal or higher than 3. If it is not the same, we will reset the receiver_duplicate_ack counter to 0, which means that no triple duplicates appear. Once receiver_duplicate_ack reaches 3, we know there is a triple duplicate, but we still don't know if there is retransmission, if it reaches 3, we set flag duplicating to true, this flag is also in the TCP_flow class, and then we go back to the sender side after we finish checking timeouts retransmission, then we check if the ack number from the receiver side is equal to the new ACK packet's seq number that is from sender to receiver, and if they are equal, then this new ACK packet is the fast retransmission that caused by the triple duplicate ACK.