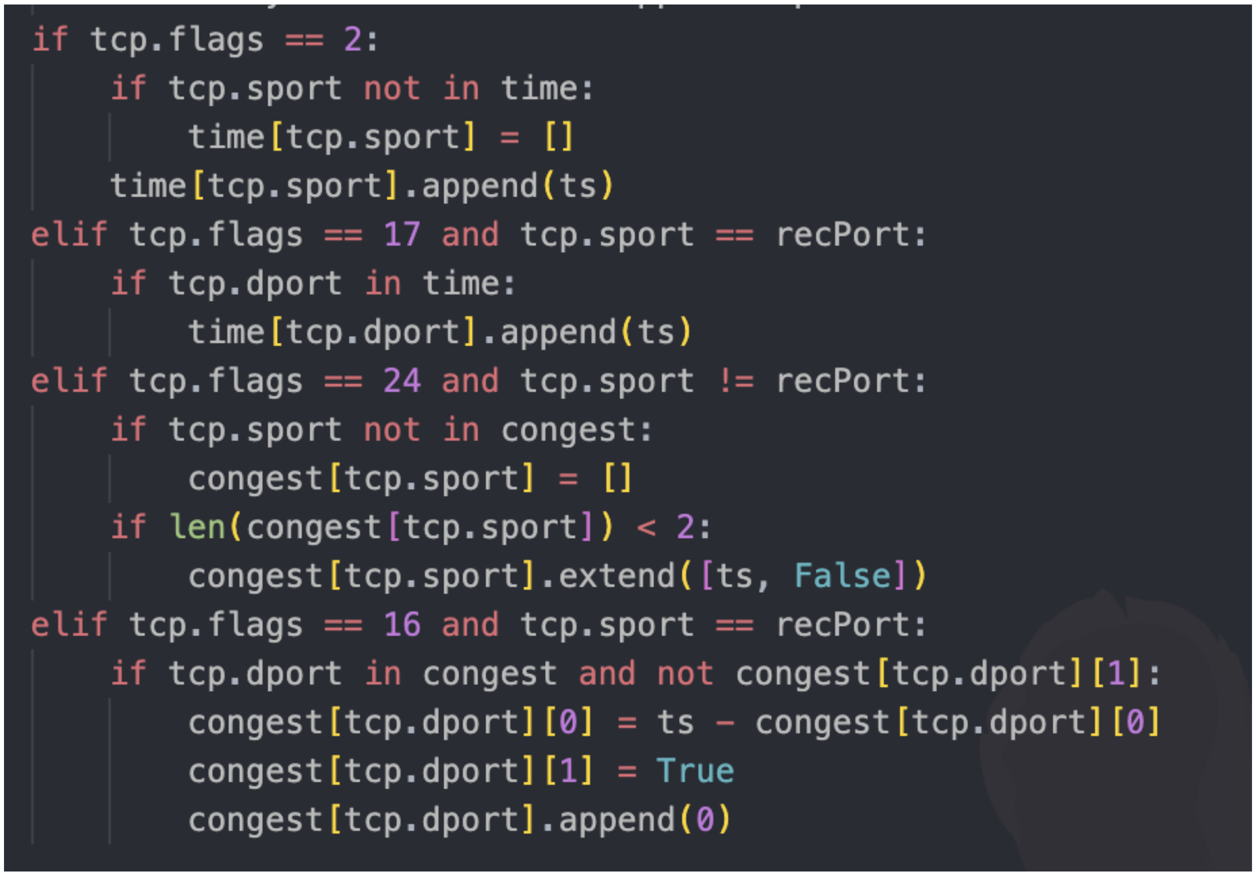
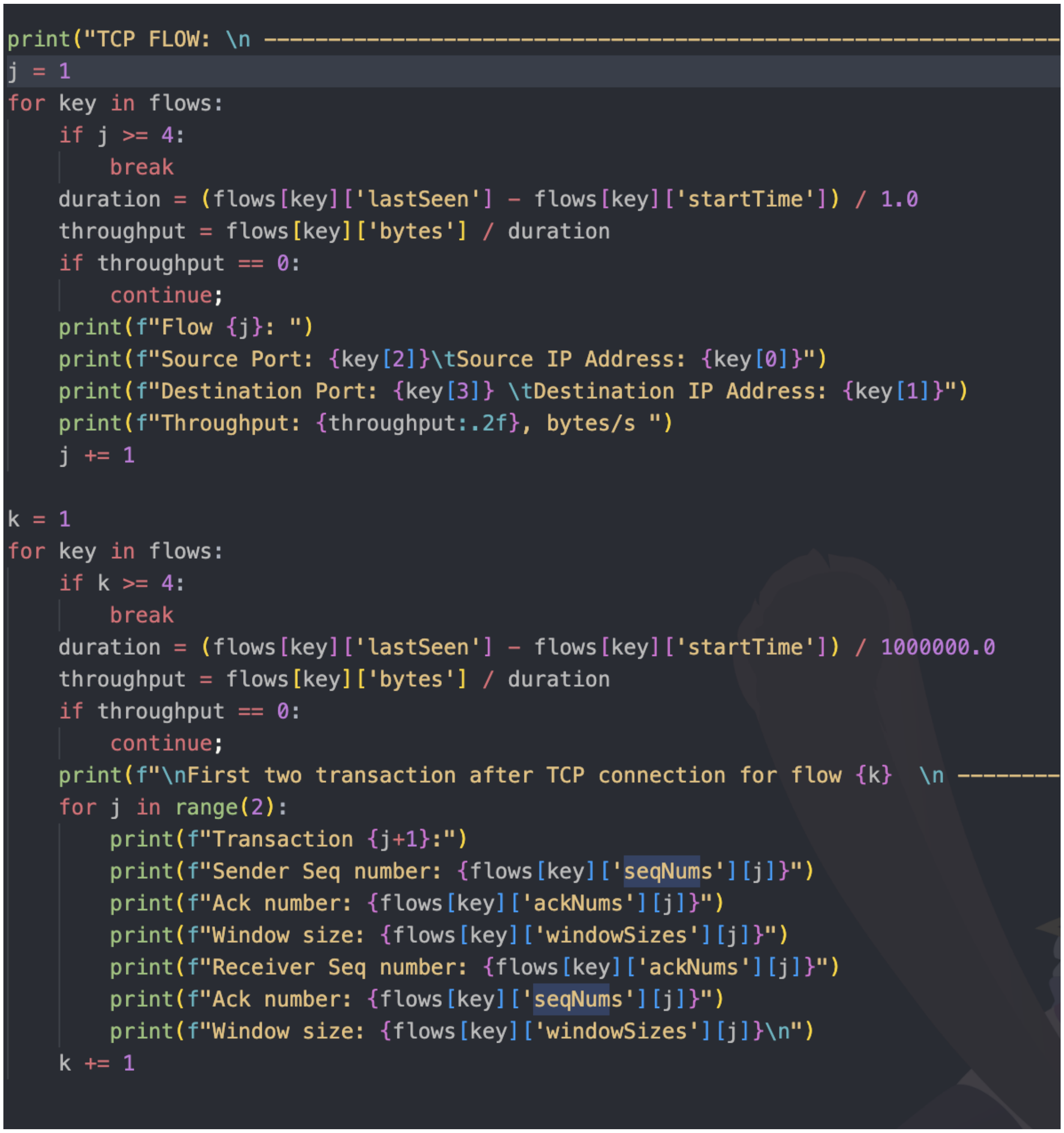
High Level Summary

The analysis_pcap_tcp.py program uses the dpkt library to analyze the info inside the assignment2.pcap file and performs calculations related to the network flows and congestion. The first steps shown in my program is opening the pcap file to read/parse. The method convMac converts the MAC addresses to a more readable string format. The program initializes variables like recPort, flows, time, and congest which are dictionaries that will be used to store information for their respective areas. The main loop of the program iterates through each packet in the pcap file. For each packet, the program checks if it is a TCP packet by checking for the eth.data attribute. If the packet is a tcp packet, the source/destination ip address, source/destination port number, tcp seq number, ack number, and window size are extracted as a tuple. The program then uses the tuple as a key to identify unique tcp flows. If it is unique, the program updates the flow information. If the packet is the first packet in the flow, the program initializes the flow information. The program checks for a specific tcp and performs additional calculations based on the flag. If the packet has a SYN flag, the program adds the timestamp of the time dictionary with the source port number as a key. If the packet has a FIN flag, the program adds the timestamp to the time dictionary with the destination port number as the key. If the packet has the CWR flag set, the program adds the timestamp to the congest dictionary with the source port as the key and sets the second element of the corresponding list to False. If this is the first time experiencing congestion, the program stores the timestamp and sets the second element of the list to false. If the packet has the ACK flag, the program checks if the source port number equals the receiving port. If the key exists in the congest dictionary and the second element of the list is false, then the program calculates the RTT as the difference between the current timestamp and the timestamp stored in the list, sets the second element to true, and adds the RTT to the list.



```
if tcp.flags == 2:
    if tcp.sport not in time:
        time[tcp.sport] = []
    time[tcp.sport].append(ts)
elif tcp.flags == 17 and tcp.sport == recPort:
    if tcp.dport in time:
        time[tcp.dport].append(ts)
elif tcp.flags == 24 and tcp.sport != recPort:
    if tcp.sport not in congest:
        congest[tcp.sport] = []
    if len(congest[tcp.sport]) < 2:
        congest[tcp.sport].extend([ts, False])
elif tcp.flags == 16 and tcp.sport == recPort:
    if tcp.dport in congest and not congest[tcp.dport][1]:
        congest[tcp.dport][0] = ts - congest[tcp.dport][0]
        congest[tcp.dport][1] = True
        congest[tcp.dport].append(0)
```

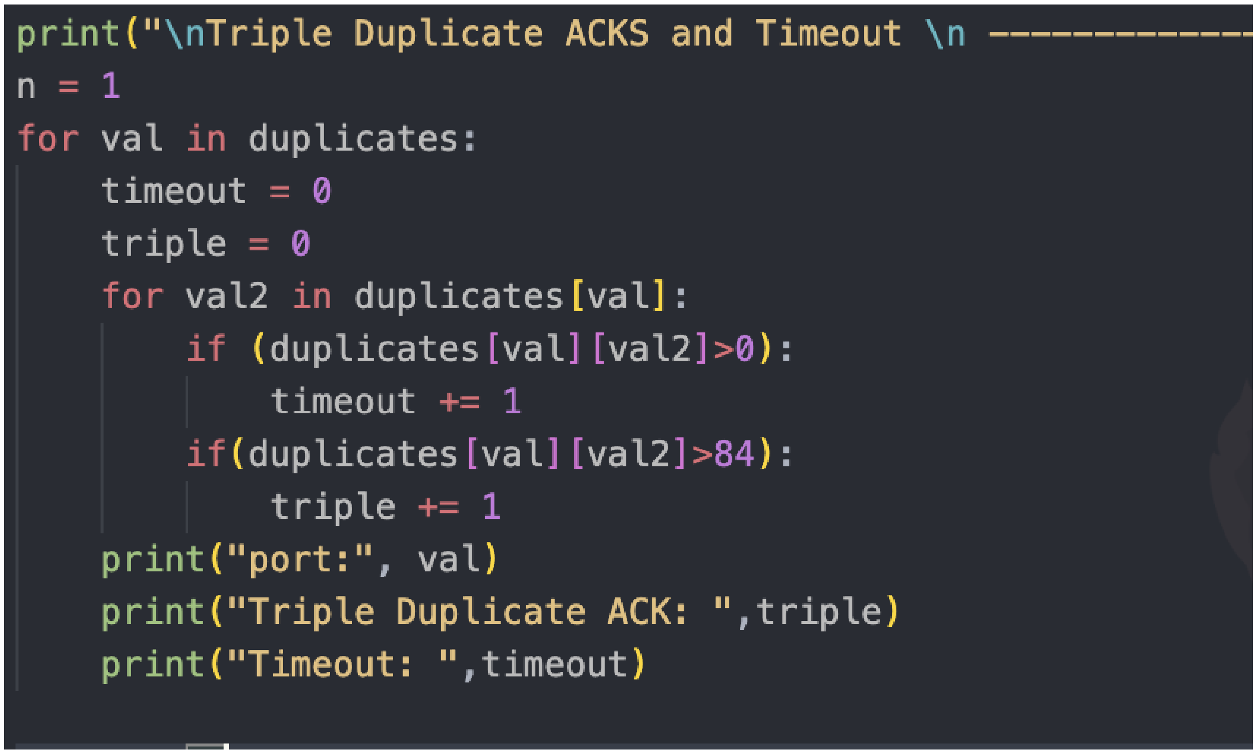
The next section of code is responsible for printing out the information about TCP flows and transactions. The first loop prints information about the first 3 TCP flows and the first 2 transactions. The for loop iterates through the flows dictionary and calculates the duration of the flow and its corresponding throughput. If the throughput is 0, the loop continues to the next iteration. Otherwise, the throughput (bits per second) gets printed out with the flow number, source, destination port, and ip address. The second for loop prints out the transactions for the 3 tcp flows. If the throughput is 0 like above, the loop continues to the next iteration. Otherwise, the flow number, details of the first 2 transactions, seq number, ack number, and window size for sender/ receiver are printed out.



```
print("TCP FLOW: \n --------------------------------------------------------------
j = 1
for key in flows:
    if j >= 4:
        break
    duration = (flows[key]['lastSeen'] - flows[key]['startTime']) / 1.0
    throughput = flows[key]['bytes'] / duration
    if throughput == 0:
        continue;
    print(f"Flow {j}: ")
    print(f"Source Port: {key[2]}\tSource IP Address: {key[0]}")
    print(f"Destination Port: {key[3]} \tDestination IP Address: {key[1]}")
    print(f"Throughput: {throughput:.2f}, bytes/s ")
    j += 1

k = 1
for key in flows:
    if k >= 4:
        break
    duration = (flows[key]['lastSeen'] - flows[key]['startTime']) / 1000000.0
    throughput = flows[key]['bytes'] / duration
    if throughput == 0:
        continue;
    print(f"\nFirst two transaction after TCP connection for flow {k}  \n --------
    for j in range(2):
        print(f"Transaction {j+1}:")
        print(f"Sender Seq number: {flows[key]['seqNums'][j]}")
        print(f"Ack number: {flows[key]['ackNums'][j]}")
        print(f"Window size: {flows[key]['windowSizes'][j]}")
        print(f"Receiver Seq number: {flows[key]['ackNums'][j]}")
        print(f"Ack number: {flows[key]['seqNums'][j]}")
        print(f"Window size: {flows[key]['windowSizes'][j]}\n")
    k += 1
```

The next section of the code is part b which prints out the congestion window size and triple duplicate ack/ timeout. The first part of part b loops through the congestion dictionary and prints the first 3 window sizes for each flow. The next section creates a dictionary called duplicate and adds information about the tcp packets that have the SYN flag. For each packet containing this flag, it creates a new entry in the dictionary. For each packet with the ACK flag and the same destination port as the receiver port specified earlier, it checks for duplicate ack numbers. If there is a duplicate, it increments the count by 1 for that acknowledgement number in the corresponding dictionary entry. The same will be done for the timeouts. We print this out for each port number.



```
print("\nTriple Duplicate ACKS and Timeout \n -------------
n = 1
for val in duplicates:
    timeout = 0
    triple = 0
    for val2 in duplicates[val]:
        if (duplicates[val][val2]>0):
            timeout += 1
        if(duplicates[val][val2]>84):
            triple += 1
    print("port:", val)
    print("Triple Duplicate ACK: ",triple)
    print("Timeout: ",timeout)
```

<u>Instructions to run the code</u>
To run this program, we use the dpkt and socket module. If you don't have dpkt installed you can use the command pip3 install dpkt. The pcap file needs to be downloaded in order to run the code. There is no input needed for this program unless a different file is used in which the code can be easily adjusted.