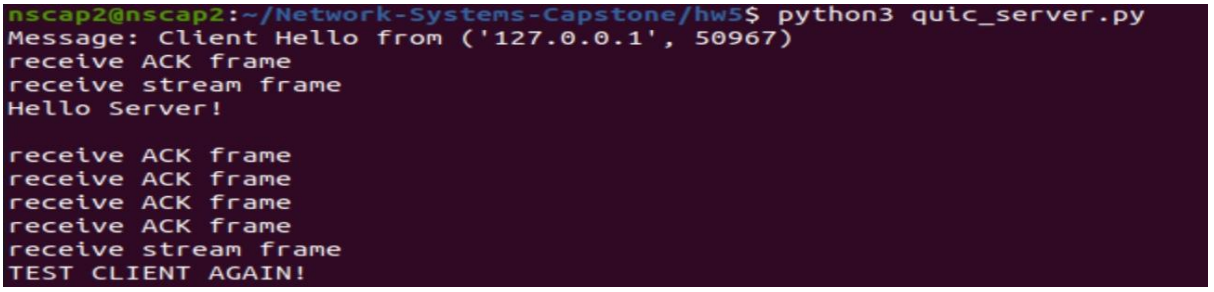
NSCap report

1. **Explain how you implement error control**

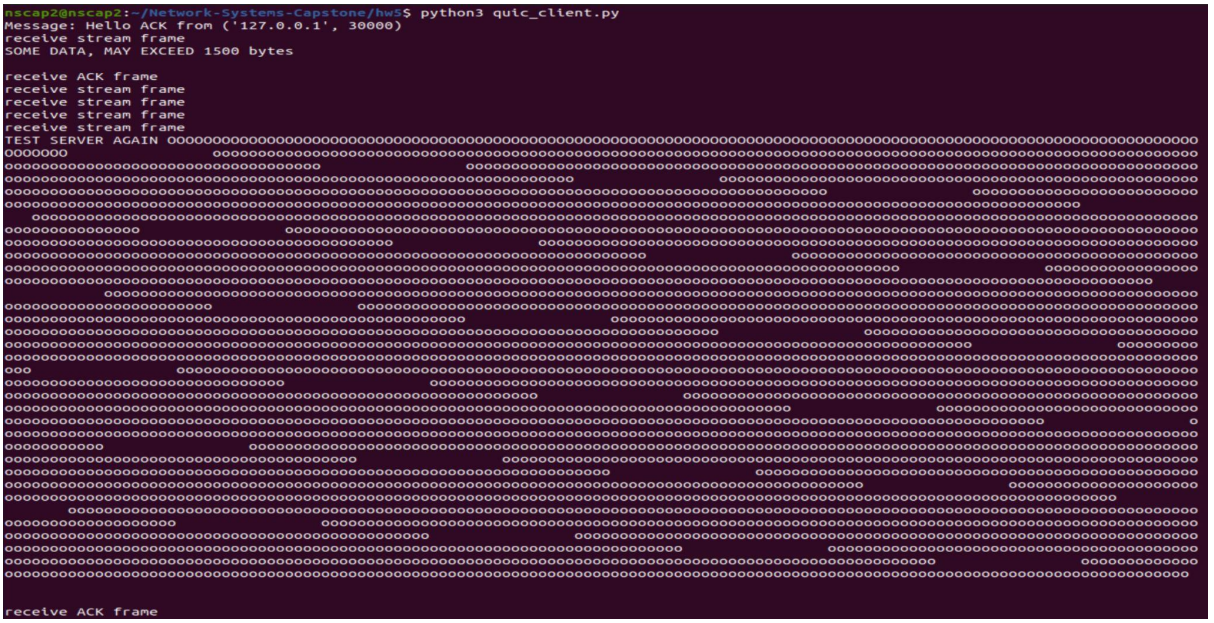
   I use a dictionary to record if the packet receives an ACK or not. As long as the packet in the stream was sent and it didn't receive an ACK in time, it should be retransmitted. Moreover, in the recv function, I will check if all the packets were received and re-order the packets before passing them to the application layer.

   The image below shows the server sending the data to the client, and the server receiving the ACK after sending the packets. Also, the receiver can re-order the packets.

   (server)



   (client)



2. **Explain how you implement flow control**

   In this part, I set the maximum size of the receive window to 30 packets. After receiving the packet, I check the size of the receiver buffer. If the receiver buffer is full, send this information when sending an ACK to the sender. Make sure that the faster sender doesn't overwhelm the slower receiver.

   For example, the receive window size is 5 and there are eight packets to send. I print out the number of packets received each time on the receiver end. After receiving six packets, the transmission will stop on the sender end because the receiver buffer is full.

   (sender)

```
nscap2@nscap2:~/Network-Systems-Capstone/hw5$ python3 quic_client.py
Message: Hello ACK from ('127.0.0.1', 30000)
sending flag:  True
sending now
sending flag:  True
sending now
sending flag:  False
sending flag:  False
sending flag:  False
sending flag:  False
```

   (receiver)

```
nscap2@nscap2:~/Network-Systems-Capstone/hw5$ python3 quic_server.py
Message: Client Hello from ('127.0.0.1', 47334)
3
3
```

3. **Explain how you implement congestion control**

   To ensure that network resources are used efficiently and that data is transmitted at a rate that is both fast and reliable, I calculate the lost packet rate (the ACK that is actually received / the ACK that should be received) to confirm whether too many packets are sent at one time. If the lost packet rate is lower than 0.5, make the congestion window half.

```
nscap2@nscap2:~/Network-Systems-Capstone/hw5$ python3 quic_server.py
Message: Client Hello from ('127.0.0.1', 38659)
window size: 32
window size: 16
window size: 8
window size: 8
window size: 8
window size: 8
window size: 8
window size: 8
```

4. **If you use two streams to send data simultaneously from the client to the server or in the other direction, what will happen if one packet of a stream gets lost? Is the behavior of QUIC different from that of TCP? Why?**

   When two streams send data simultaneously from the client to the server or in the other direction, if one packet of a stream gets lost, only the packets of that particular stream will be affected, and the packets of the other stream will continue to be transmitted. QUIC can adapt to the conditions of each individual stream and avoid the unnecessary reduction of sending rates for other streams that are not affected by packet loss. However, the behavior of TCP is different from that of QUIC. In TCP, there is only a single congestion control mechanism for the entire connection. Therefore, when a packet is lost, the congestion control mechanism will reduce the sending rate for all streams which share the same connection.