Mini-Project (2)

Step 1: Design & Implement our protocol

Introduction:
Our group built the protocol on the top of UDP socket connection. We implemented connection-oriented, reliable, pipelined protocol which includes flow control and congestion control mechanisms as well. Since UDP does not inherently provide these features, we added mechanisms to ensure reliable communication, similar to TCP.

Connection-oriented and reliable communication: We simulate a **3-way TCP handshake** on top of the UDP protocol to make our protocol connection-oriented. We have a packet class which simlates the TCP packet structure.
Firstly, we create a UDP socket between the client and the server. Then, the client will make a packet with initial sequence number x, set the TCP SYN bit to true, and send the packet to the server. Once the server receives this packet, the server will make a packet with setting TCP SYN bit and ACK bit to true, indicating it is SYNACK, and an initial sequence number decided by the server, an ack number which is x+1.
Next, the server will send the packet it made to the client. When the client receives this packet indicating SYNACK, it shows the server is alive, the connection on the client side is established. The client then will make a packet with ack number to ack the SYNACK, send this packet to the server. If the server receives the correct ACK number, it indicates the client is alive and the connection is established.
Additionally, a timer is set for the SYN packet sent by the client and the SYNACK packet sent by the server. If either of these packets is not acknowledged within the timeout period, it will be resent for maximum 3 times, ensuring reliable communication during the handshake.

Pipelined protocol:
We chose to implement **Go-back-N** to ensure data reliability.
Sender:
1. Sender can have up to N (window size) un-acked packets in pipeline, representing the number of unacked packets in the pipeline.
2. Sender sets timer for oldest un-acket packet. When timer expires, sender will retransmit all un-acked packets.
3. Sends packets sequentially, keep track of the sequence number. The next sequence number is the current sequence number plus the length of its payload.

Receiver:
1. Only sends cumulative ack for the last packet received in order.
2. Does not ack packet if there’s a gap, discard out of order packets.
2. Will resend the last ack number to sender if there is a gap.

Flow control:
Our protocol implements flow control using receiver window size (rwnd). The receiver controls sender, and sender will not overflow. The receiver advertises its available buffer size (rwnd) in every acknowledgment packet header.The sender adjusts its transmission rate to make sure the number of outstanding packets does not exceed the receiver’s rwnd, so that we ensure the sender does not overwhelm the receiver.

Congestion control:
We simulate the **TCP Tahoe** congestion control.
Slow start: The congestion window (cwnd) starts at 1 packet and doubles the size with every successful acknowledgment until it reaches the threshold (ssthresh) or a packet loss is detected.
Congestion avoidance: When cwnd exceeds ssthresh, the window size grows linearly instead of exponentially.
Loss indicated by timeout or 3 duplicate acks: When detect packet loss, the cwnd is reset to 1 and ssthresh is set to half of the current cwnd.

Socket type: UDP socket, based on UDP, build TCP like protocol.

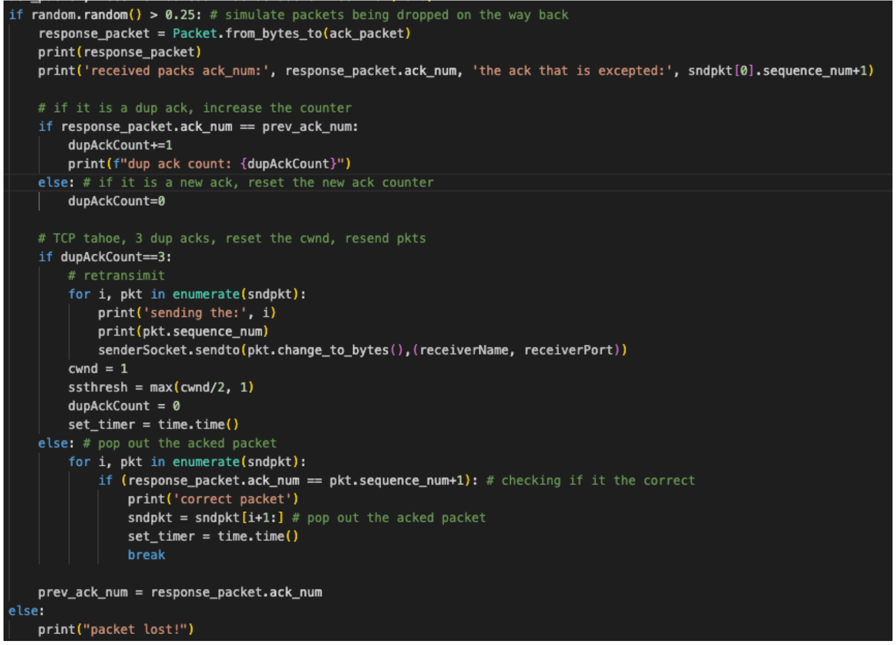
Packet structure: A Packet class was implemented to simulate a TCP-like packet.
Each packet contains:

- Sequence number: For tracking order.
- Acknowledgment number: For reliable communication.
- Flags: SYN, ACK, FIN flags for connection control.
- Payload: the data being transmitted. We will get the payload data from a text file.
- Receiver window size (rwnd): Advertised by the receiver for flow control.

Step 2: Analyze Traffic

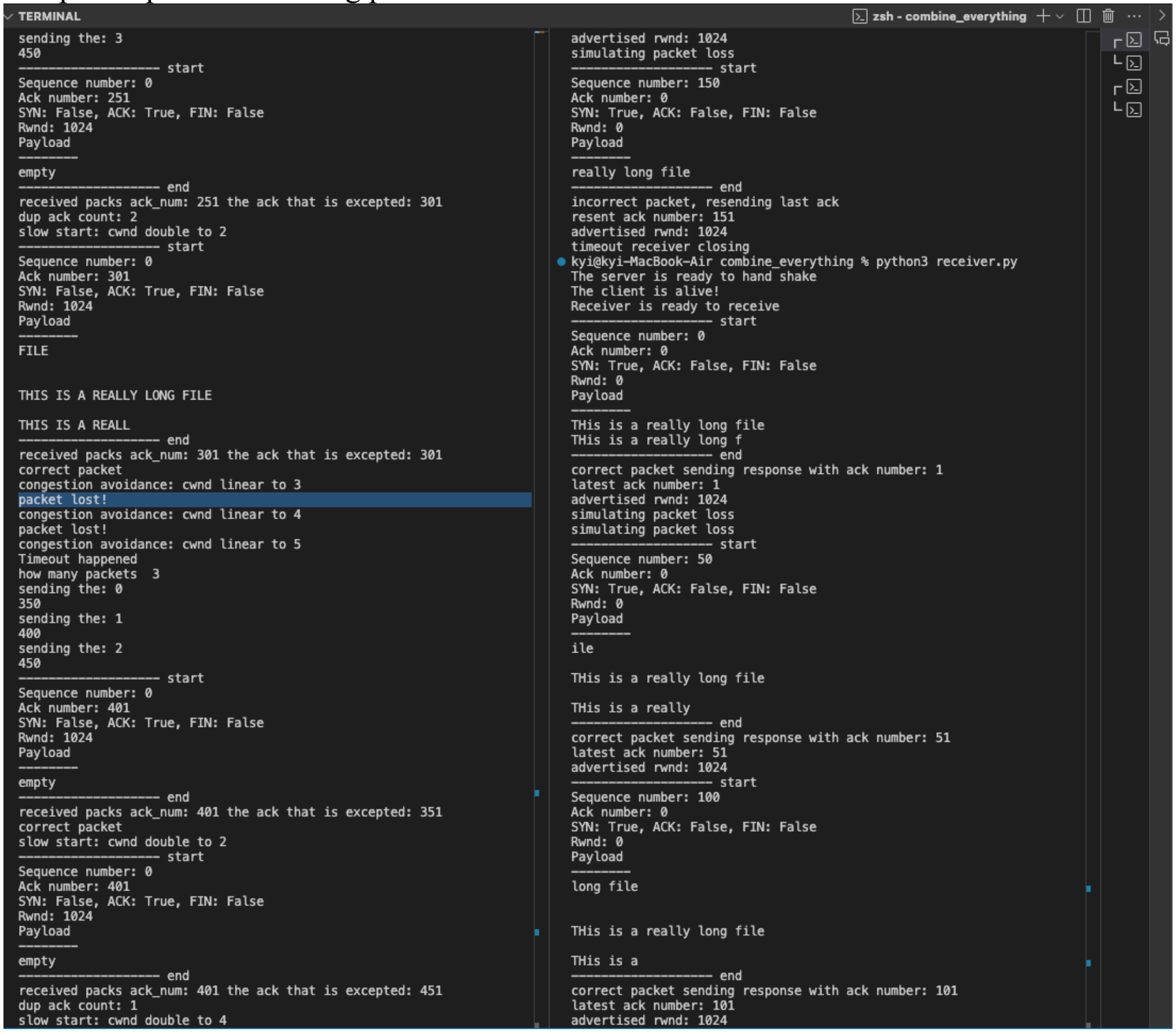
Simulate packet loss:

Where to do it: in the function that sends packets, before sending packets to the receiver.
How to do it: we used a random function to simulate packet loss by randomly decide whether or not to drop a packet. We set the loss rate is 25%, if generating a random number that is less than 0.25, we will drop this packet. If the generated number is greater than 0.25, than we will send this packet over network.



```
if random.random() > 0.25: # simulate packets being dropped on the way back
    response_packet = Packet.from_bytes_to(ack_packet)
    print(response_packet)
    print('received packs ack_num:', response_packet.ack_num, 'the ack that is excepted:', sndpkt[0].sequence_num+1)

    # if it is a dup ack, increase the counter
    if response_packet.ack_num == prev_ack_num:
        dupAckCount+=1
        print(f"dup ack count: {dupAckCount}")
    else: # if it is a new ack, reset the new ack counter
        dupAckCount=0

    # TCP tahoe, 3 dup acks, reset the cwnd, resend pkts
    if dupAckCount==3:
        # retransimit
        for i, pkt in enumerate(sndpkt):
            print('sending the:', i)
            print(pkt.sequence_num)
            senderSocket.sendto(pkt.change_to_bytes(),(receiverName, receiverPort))
        cwnd = 1
        ssthresh = max(cwnd/2, 1)
        dupAckCount = 0
        set_timer = time.time()
    else: # pop out the acked packet
        for i, pkt in enumerate(sndpkt):
            if (response_packet.ack_num == pkt.sequence_num+1): # checking if it the correct
                print('correct packet')
                sndpkt = sndpkt[i+1:] # pop out the acked packet
                set_timer = time.time()
                break

    prev_ack_num = response_packet.ack_num
else:
    print("packet lost!")
```

Sample output of simulating packet loss:



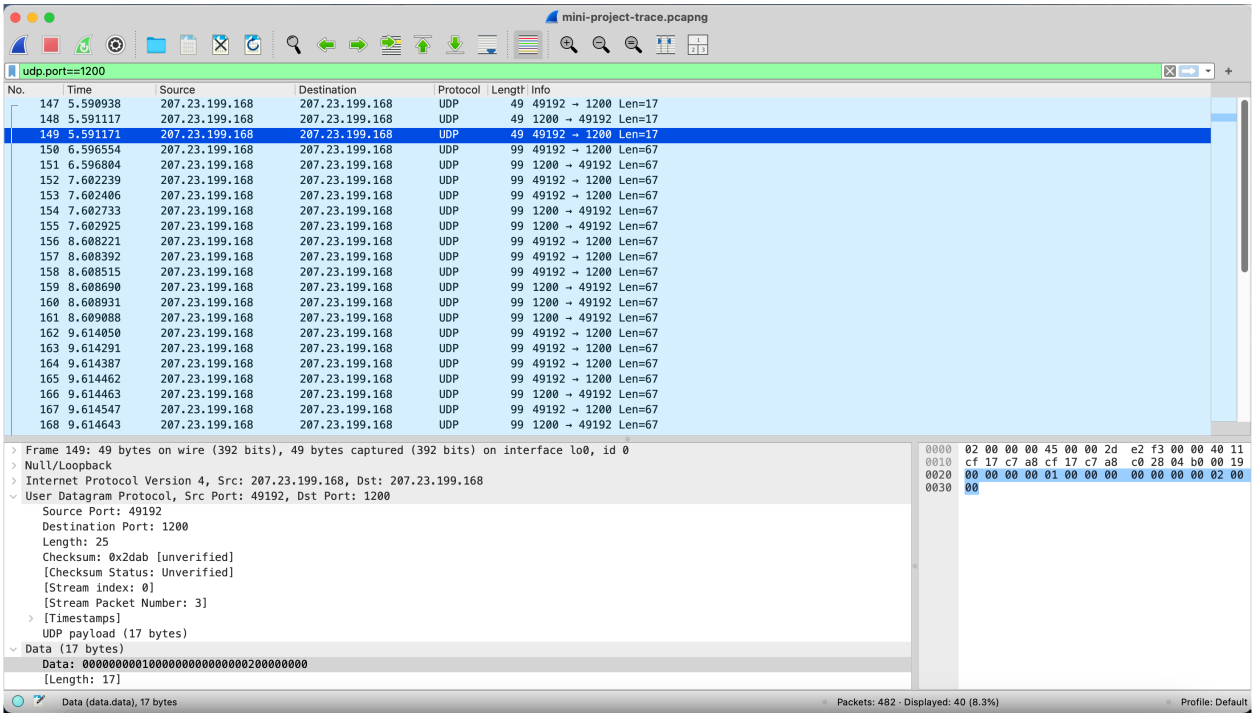
```
sending the: 3
450
------------------- start
Sequence number: 0
Ack number: 251
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 251 the ack that is excepted: 301
dup ack count: 2
slow start: cwnd double to 2
------------------- start
Sequence number: 0
Ack number: 301
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
FILE


THIS IS A REALLY LONG FILE

THIS IS A REALL
------------------- end
received packs ack_num: 301 the ack that is excepted: 301
correct packet
congestion avoidance: cwnd linear to 3
packet lost!
congestion avoidance: cwnd linear to 4
packet lost!
congestion avoidance: cwnd linear to 5
Timeout happened
how many packets  3
sending the: 0
350
sending the: 1
400
sending the: 2
450
------------------- start
Sequence number: 0
Ack number: 401
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 401 the ack that is excepted: 351
correct packet
slow start: cwnd double to 2
------------------- start
Sequence number: 0
Ack number: 401
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 401 the ack that is excepted: 451
dup ack count: 1
slow start: cwnd double to 4
```

```
advertised rwnd: 1024
simulating packet loss
------------------- start
Sequence number: 150
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
really long file
------------------- end
incorrect packet, resending last ack
resent ack number: 151
advertised rwnd: 1024
timeout receiver closing
kyi@kyi-MacBook-Air combine_everything % python3 receiver.py
The server is ready to hand shake
The client is alive!
Receiver is ready to receive
------------------- start
Sequence number: 0
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
THis is a really long file
THis is a really long f
------------------- end
correct packet sending response with ack number: 1
latest ack number: 1
advertised rwnd: 1024
simulating packet loss
simulating packet loss
------------------- start
Sequence number: 50
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
ile

THis is a really long file

THis is a really
------------------- end
correct packet sending response with ack number: 51
latest ack number: 51
advertised rwnd: 1024
------------------- start
Sequence number: 100
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
long file


THis is a really long file

THis is a
------------------- end
correct packet sending response with ack number: 101
latest ack number: 101
advertised rwnd: 1024
```

Proof of connection-oriented:

In Wireshark:

The first 3 packets are the packets being sent in the handshake part, these packets only have 17 bytes (header) and no payload. The first packet is from sender and includes TCP SYN message, the second packet is from receiver and includes SYNACK, and the third packet is from sender and includes ACK number.



```
udp.port==1200
```

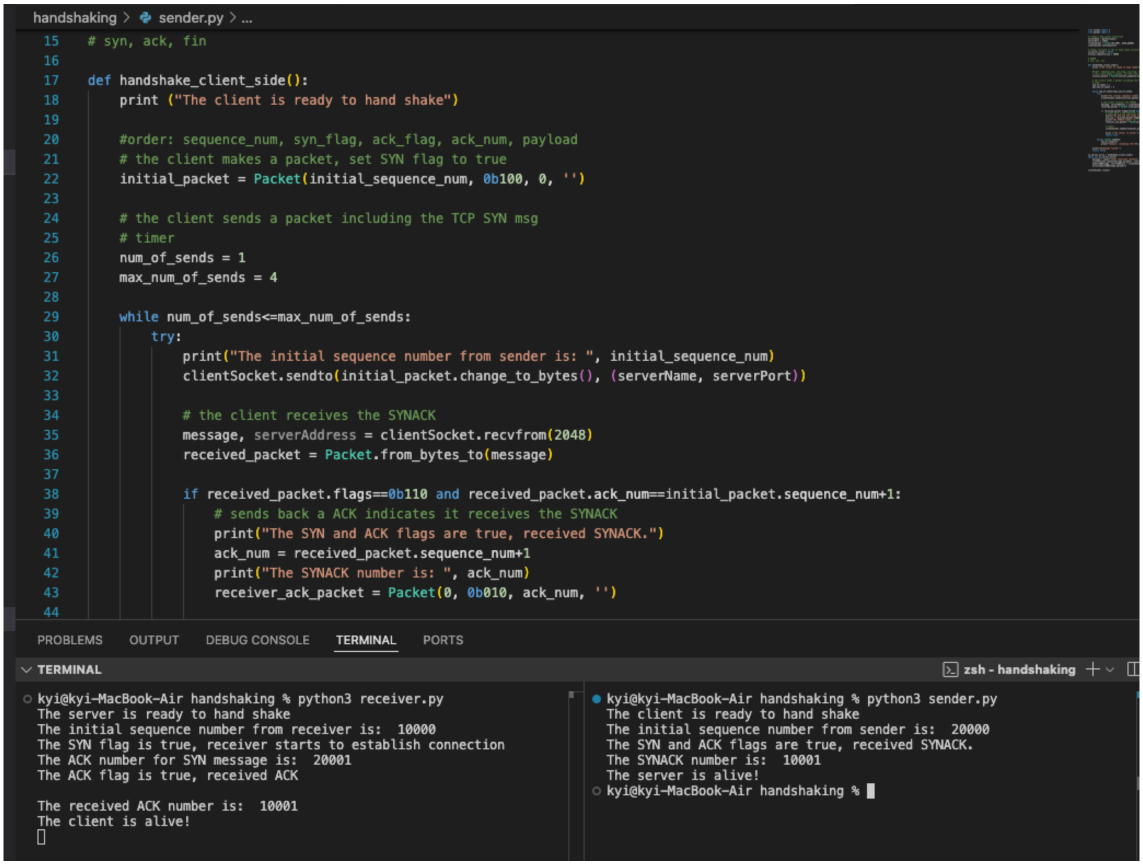
```
No.  Time      Source          Destination     Protocol Length Info
147 5.590938  207.23.199.168  207.23.199.168  UDP   49 49192 → 1200 Len=17
148 5.591117  207.23.199.168  207.23.199.168  UDP   49 1200 → 49192 Len=17
149 5.591171  207.23.199.168  207.23.199.168  UDP   49 49192 → 1200 Len=17
150 6.596554  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
151 6.596804  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
152 7.602239  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
153 7.602406  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
154 7.602733  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
155 7.602925  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
156 8.608221  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
157 8.608392  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
158 8.608515  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
159 8.608690  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
160 8.608931  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
161 8.609088  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
162 9.614050  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
163 9.614291  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
164 9.614387  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
165 9.614462  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
166 9.614463  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
167 9.614547  207.23.199.168  207.23.199.168  UDP   99 49192 → 1200 Len=67
168 9.614643  207.23.199.168  207.23.199.168  UDP   99 1200 → 49192 Len=67
```

```
> Frame 149: 49 bytes on wire (392 bits), 49 bytes captured (392 bits) on interface lo0, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 207.23.199.168, Dst: 207.23.199.168
v User Datagram Protocol, Src Port: 49192, Dst Port: 1200
    Source Port: 49192
    Destination Port: 1200
    Length: 25
    Checksum: 0x2dab [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 3]
  > [Timestamps]
    UDP payload (17 bytes)
v Data (17 bytes)
    Data: 0000000001000000000000000200000000
    [Length: 17]
```

```
0000  02 00 00 00 45 00 00 2d  e2 f3 00 00 40 11
0010  cf 17 c7 a8 cf 17 c7 a8  c0 28 04 b0 00 19
0020  00 00 00 00 01 00 00 00  00 00 00 00 02 00
0030  00
```

```
Data (data.data), 17 bytes          Packets: 482 · Displayed: 40 (8.3%)          Profile: Default
```

The screenshot shows a connection-oriented protocol through the exchange of ACK numbers during the handshake process. The sender sends a SYN packet with a sequence number, and the receiver replies with an ACK (SYNACK), acknowledging the client’s sequence number and sending its own. Then, sender sends an ACK to acknowledge SYNACK. This back-and-forth of sequence and acknowledgment numbers confirms reliable communication, typical of connection-oriented protocols like TCP.



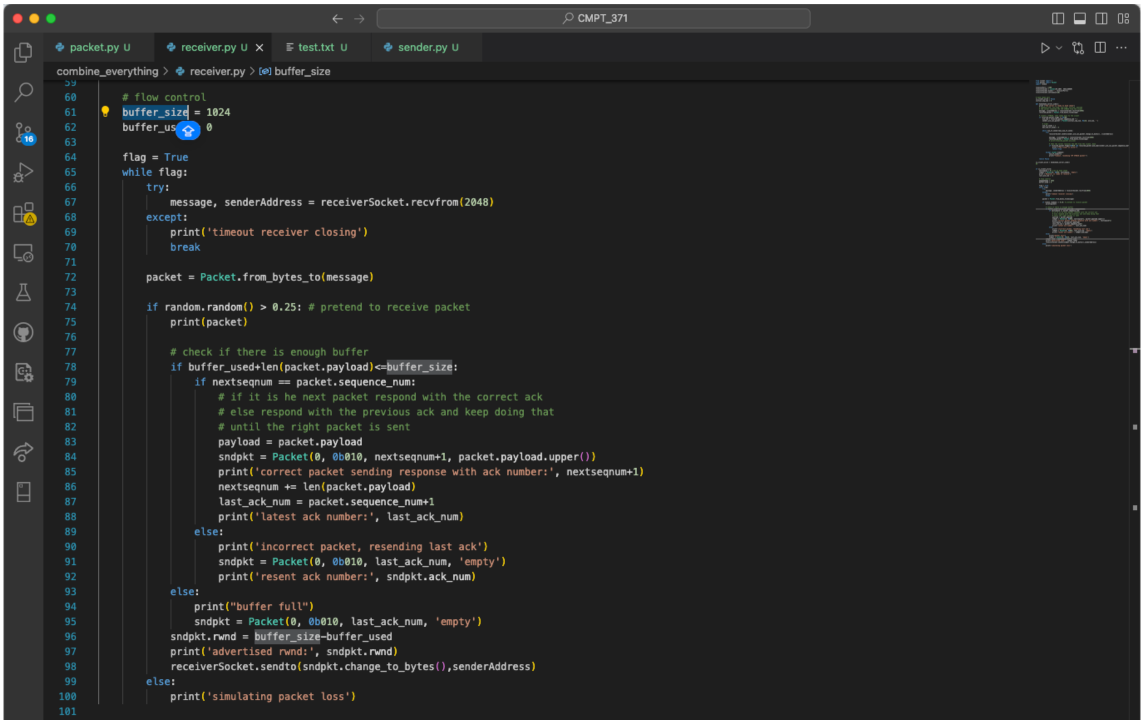
```python
# syn, ack, fin

def handshake_client_side():
    print ("The client is ready to hand shake")

    #order: sequence_num, syn_flag, ack_flag, ack_num, payload
    # the client makes a packet, set SYN flag to true
    initial_packet = Packet(initial_sequence_num, 0b100, 0, '')

    # the client sends a packet including the TCP SYN msg
    # timer
    num_of_sends = 1
    max_num_of_sends = 4

    while num_of_sends<=max_num_of_sends:
        try:
            print("The initial sequence number from sender is: ", initial_sequence_num)
            clientSocket.sendto(initial_packet.change_to_bytes(), (serverName, serverPort))

            # the client receives the SYNACK
            message, serverAddress = clientSocket.recvfrom(2048)
            received_packet = Packet.from_bytes_to(message)

            if received_packet.flags==0b110 and received_packet.ack_num==initial_packet.sequence_num+1:
                # sends back a ACK indicates it receives the SYNACK
                print("The SYN and ACK flags are true, received SYNACK.")
                ack_num = received_packet.sequence_num+1
                print("The SYNACK number is: ", ack_num)
                receiver_ack_packet = Packet(0, 0b010, ack_num, '')
```

```
kyi@kyi-MacBook-Air handshaking % python3 receiver.py
The server is ready to hand shake
The initial sequence number from receiver is:  10000
The SYN flag is true, receiver starts to establish connection
The ACK number for SYN message is:  20001
The ACK flag is true, received ACK

The received ACK number is:  10001
The client is alive!
```

```
kyi@kyi-MacBook-Air handshaking % python3 sender.py
The client is ready to hand shake
The initial sequence number from sender is:  20000
The SYN and ACK flags are true, received SYNACK.
The SYNACK number is:  10001
The server is alive!
kyi@kyi-MacBook-Air handshaking %
```

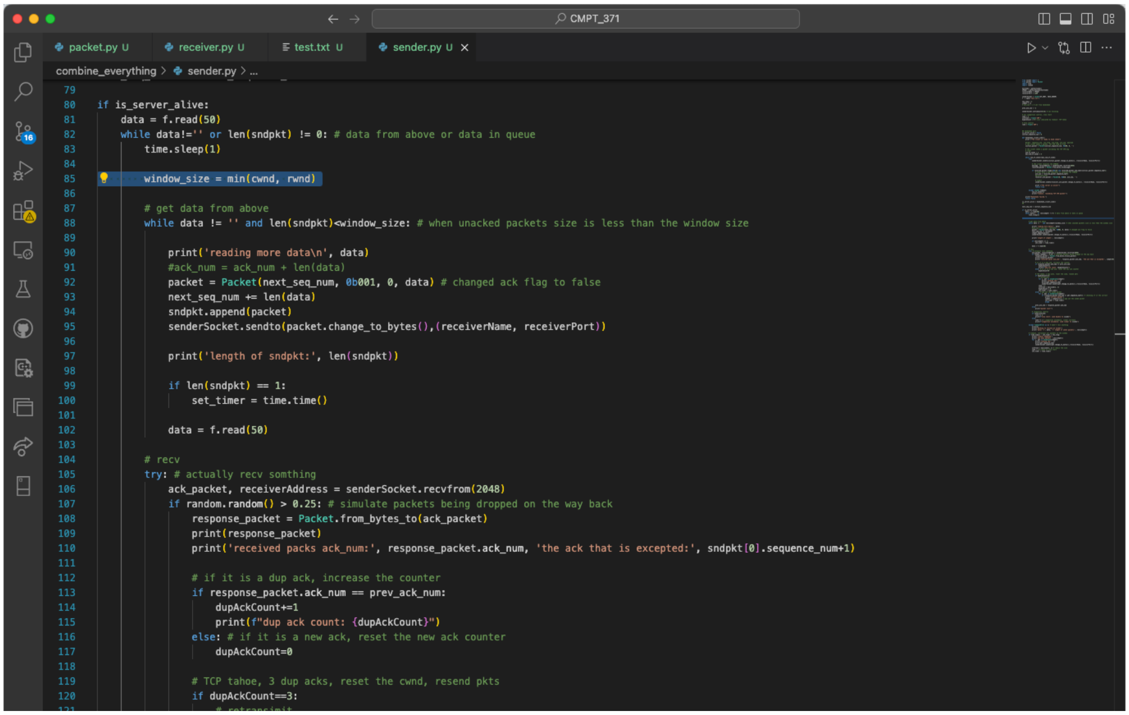
Proof of flow control:

In receiver: set buffer size and used buffer size, put rwnd in the packet header and send it to sender:



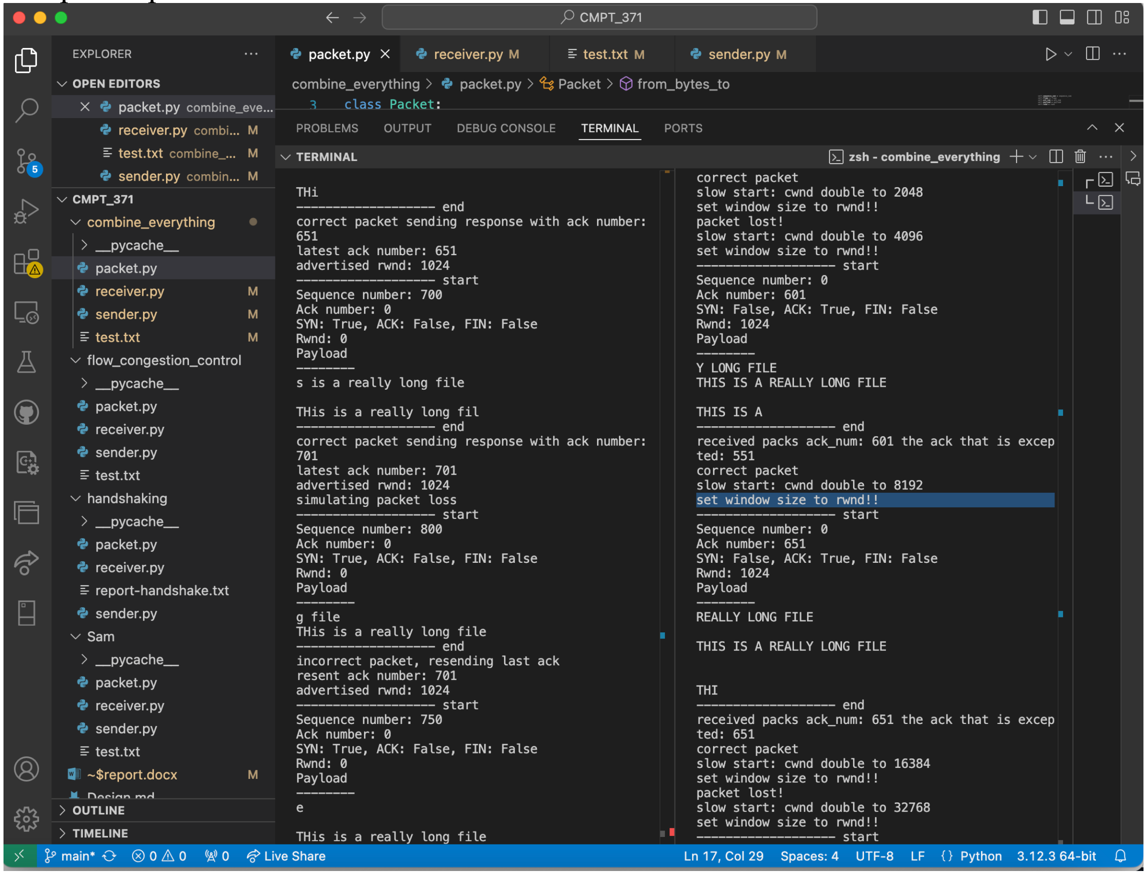
```python
    # flow control
    buffer_size = 1024
    buffer_us 0

    flag = True
    while flag:
        try:
            message, senderAddress = receiverSocket.recvfrom(2048)
        except:
            print('timeout receiver closing')
            break

        packet = Packet.from_bytes_to(message)

        if random.random() > 0.25: # pretend to receive packet
            print(packet)

            # check if there is enough buffer
            if buffer_used+len(packet.payload)<=buffer_size:
                if nextseqnum == packet.sequence_num:
                    # if it is he next packet respond with the correct ack
                    # else respond with the previous ack and keep doing that
                    # until the right packet is sent
                    payload = packet.payload
                    sndpkt = Packet(0, 0b010, nextseqnum+1, packet.payload.upper())
                    print('correct packet sending response with ack number:', nextseqnum+1)
                    nextseqnum += len(packet.payload)
                    last_ack_num = packet.sequence_num+1
                    print('latest ack number:', last_ack_num)
                else:
                    print('incorrect packet, resending last ack')
                    sndpkt = Packet(0, 0b010, last_ack_num, 'empty')
                    print('resent ack number:', sndpkt.ack_num)
            else:
                print("buffer full")
                sndpkt = Packet(0, 0b010, last_ack_num, 'empty')
            sndpkt.rwnd = buffer_size-buffer_used
            print('advertised rwnd:', sndpkt.rwnd)
            receiverSocket.sendto(sndpkt.change_to_bytes(),senderAddress)
        else:
            print('simulating packet loss')
```

In sender: get the rwnd from the receiver, set the window size to rwnd or cwnd (smaller one), so that the sender won’t overwhelm receiver:



```python

if is_server_alive:
    data = f.read(50)
    while data!='' or len(sndpkt) != 0: # data from above or data in queue
        time.sleep(1)

        window_size = min(cwnd, rwnd)

        # get data from above
        while data != '' and len(sndpkt)<window_size: # when unacked packets size is less than the window size

            print('reading more data\n', data)
            #ack_num = ack_num + len(data)
            packet = Packet(next_seq_num, 0b001, 0, data) # changed ack flag to false
            next_seq_num += len(data)
            sndpkt.append(packet)
            senderSocket.sendto(packet.change_to_bytes(),(receiverName, receiverPort))

            print('length of sndpkt:', len(sndpkt))

            if len(sndpkt) == 1:
                set_timer = time.time()

            data = f.read(50)

        # recv
        try: # actually recv somthing
            ack_packet, receiverAddress = senderSocket.recvfrom(2048)
            if random.random() > 0.25: # simulate packets being dropped on the way back
                response_packet = Packet.from_bytes_to(ack_packet)
                print(response_packet)
                print('received packs ack_num:', response_packet.ack_num, 'the ack that is excepted:', sndpkt[0].sequence_num+1)

                # if it is a dup ack, increase the counter
                if response_packet.ack_num == prev_ack_num:
                    dupAckCount+=1
                    print(f"dup ack count: {dupAckCount}")
                else: # if it is a new ack, reset the new ack counter
                    dupAckCount=0

                # TCP tahoe, 3 dup acks, reset the cwnd, resend pkts
                if dupAckCount==3:
                    # retransimit
```

Sample output: the sender sets the window size to receiver window size

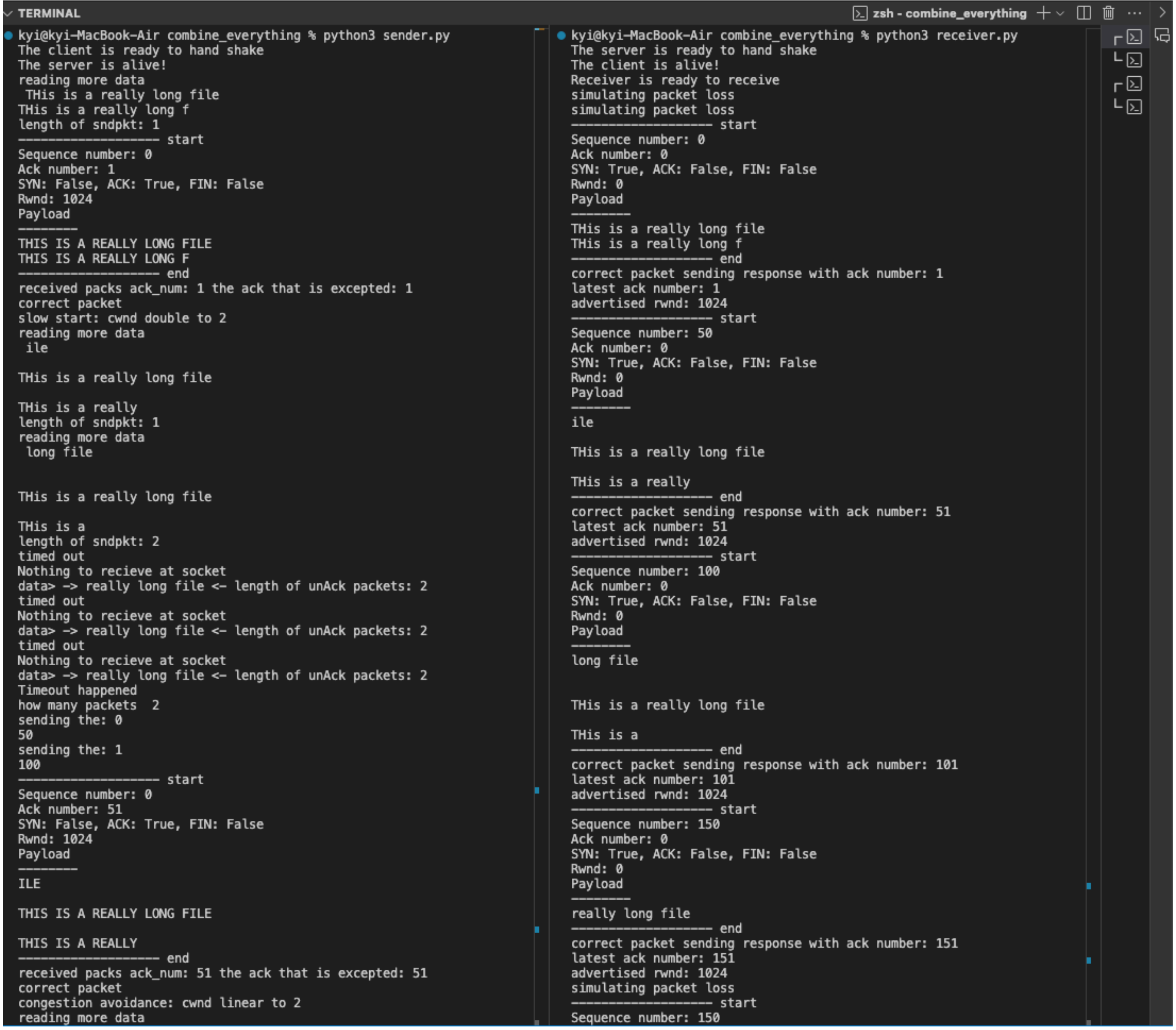


```
THi
------------------- end
correct packet sending response with ack number:
651
latest ack number: 651
advertised rwnd: 1024
------------------- start
Sequence number: 700
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
s is a really long file

THis is a really long fil
------------------- end
correct packet sending response with ack number:
701
latest ack number: 701
advertised rwnd: 1024
simulating packet loss
------------------- start
Sequence number: 800
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
g file
THis is a really long file
------------------- end
incorrect packet, resending last ack
resent ack number: 701
advertised rwnd: 1024
------------------- start
Sequence number: 750
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
e

THis is a really long file
```

```
correct packet
slow start: cwnd double to 2048
set window size to rwnd!!
packet lost!
slow start: cwnd double to 4096
set window size to rwnd!!
------------------- start
Sequence number: 0
Ack number: 601
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
Y LONG FILE
THIS IS A REALLY LONG FILE

THIS IS A
------------------- end
received packs ack_num: 601 the ack that is excep
ted: 551
correct packet
slow start: cwnd double to 8192
set window size to rwnd!!
------------------- start
Sequence number: 0
Ack number: 651
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
REALLY LONG FILE

THIS IS A REALLY LONG FILE


THI
------------------- end
received packs ack_num: 651 the ack that is excep
ted: 651
correct packet
slow start: cwnd double to 16384
set window size to rwnd!!
packet lost!
slow start: cwnd double to 32768
set window size to rwnd!!
------------------- start
```

Proof of congestion control:

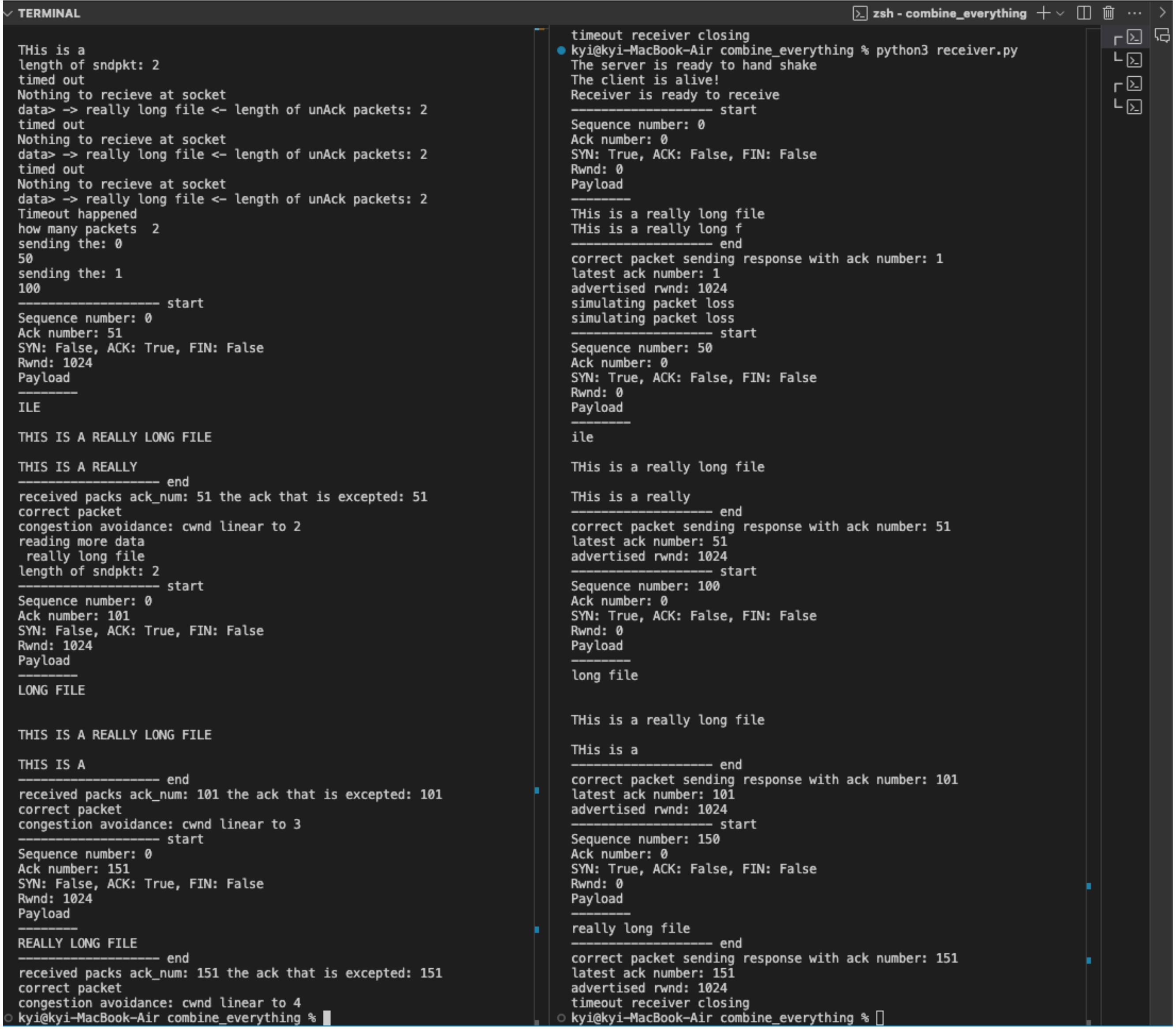
At first, sender adjust the window size by slow start



```
kyi@kyi-MacBook-Air combine_everything % python3 sender.py
The client is ready to hand shake
The server is alive!
reading more data
 THis is a really long file
THis is a really long f
length of sndpkt: 1
------------------- start
Sequence number: 0
Ack number: 1
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
THIS IS A REALLY LONG FILE
THIS IS A REALLY LONG F
------------------- end
received packs ack_num: 1 the ack that is excepted: 1
correct packet
slow start: cwnd double to 2
reading more data
 ile

THis is a really long file

THis is a really
length of sndpkt: 1
reading more data
 long file


THis is a really long file

THis is a
length of sndpkt: 2
timed out
Nothing to recieve at socket
data> -> really long file <- length of unAck packets: 2
timed out
Nothing to recieve at socket
data> -> really long file <- length of unAck packets: 2
timed out
Nothing to recieve at socket
data> -> really long file <- length of unAck packets: 2
Timeout happened
how many packets  2
sending the: 0
50
sending the: 1
100
------------------- start
Sequence number: 0
Ack number: 51
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
ILE

THIS IS A REALLY LONG FILE

THIS IS A REALLY
------------------- end
received packs ack_num: 51 the ack that is excepted: 51
correct packet
congestion avoidance: cwnd linear to 2
reading more data
```

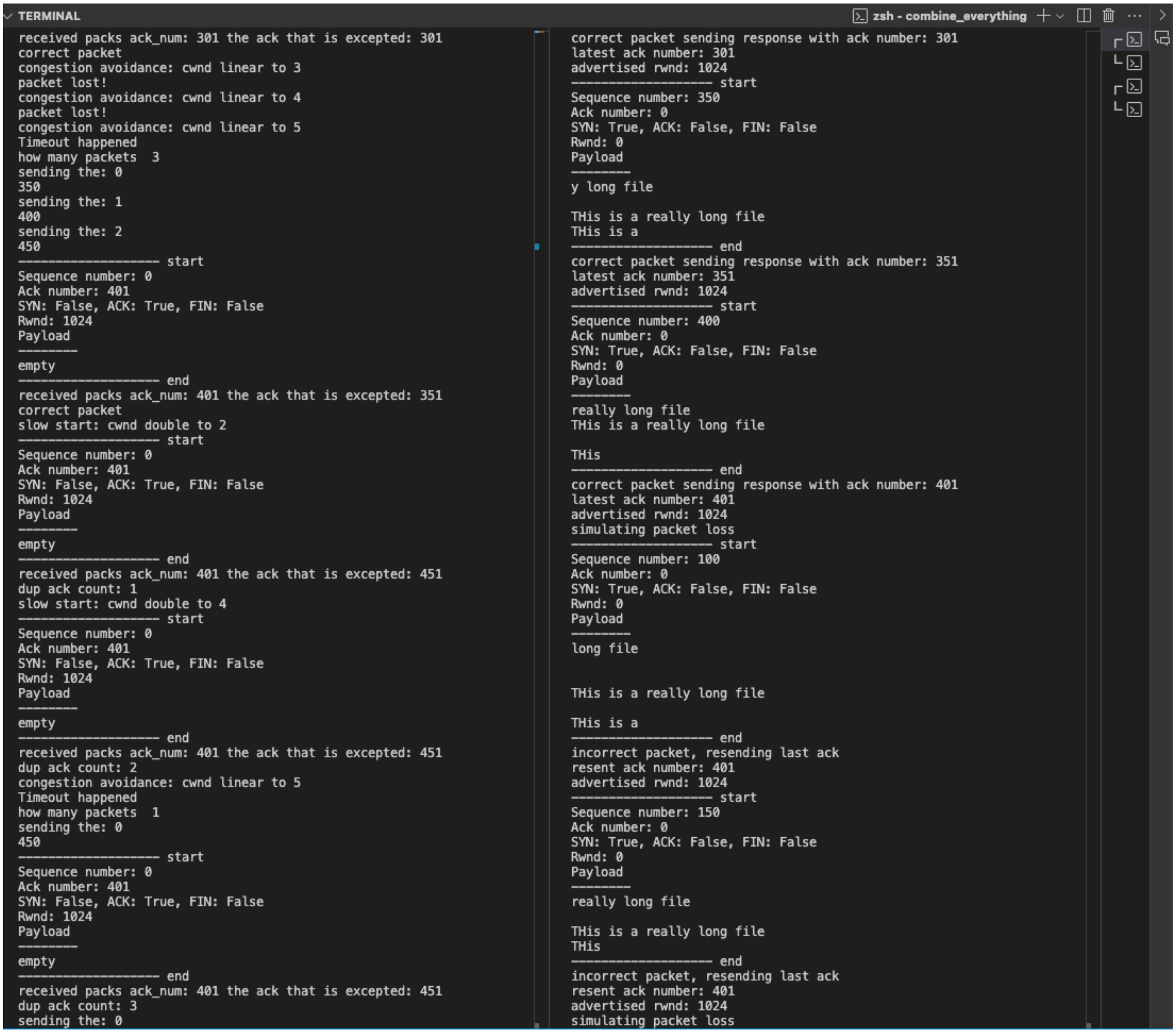
```
kyi@kyi-MacBook-Air combine_everything % python3 receiver.py
The server is ready to hand shake
The client is alive!
Receiver is ready to receive
simulating packet loss
simulating packet loss
------------------- start
Sequence number: 0
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
THis is a really long file
THis is a really long f
------------------- end
correct packet sending response with ack number: 1
latest ack number: 1
advertised rwnd: 1024
------------------- start
Sequence number: 50
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
ile

THis is a really long file

THis is a really
------------------- end
correct packet sending response with ack number: 51
latest ack number: 51
advertised rwnd: 1024
------------------- start
Sequence number: 100
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
long file


THis is a really long file

THis is a
------------------- end
correct packet sending response with ack number: 101
latest ack number: 101
advertised rwnd: 1024
------------------- start
Sequence number: 150
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
really long file
------------------- end
correct packet sending response with ack number: 151
latest ack number: 151
advertised rwnd: 1024
simulating packet loss
------------------- start
Sequence number: 150
```

When the window size is greater or equal to ssthresh, the congestion avoidance happens, window size changes to increase linearly



```
THis is a
length of sndpkt: 2
timed out
Nothing to recieve at socket
data> -> really long file <- length of unAck packets: 2
timed out
Nothing to recieve at socket
data> -> really long file <- length of unAck packets: 2
timed out
Nothing to recieve at socket
data> -> really long file <- length of unAck packets: 2
Timeout happened
how many packets  2
sending the: 0
50
sending the: 1
100
------------------- start
Sequence number: 0
Ack number: 51
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
ILE

THIS IS A REALLY LONG FILE

THIS IS A REALLY
------------------- end
received packs ack_num: 51 the ack that is excepted: 51
correct packet
congestion avoidance: cwnd linear to 2
reading more data
 really long file
length of sndpkt: 2
------------------- start
Sequence number: 0
Ack number: 101
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
LONG FILE


THIS IS A REALLY LONG FILE

THIS IS A
------------------- end
received packs ack_num: 101 the ack that is excepted: 101
correct packet
congestion avoidance: cwnd linear to 3
------------------- start
Sequence number: 0
Ack number: 151
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
REALLY LONG FILE
------------------- end
received packs ack_num: 151 the ack that is excepted: 151
correct packet
congestion avoidance: cwnd linear to 4
kyi@kyi-MacBook-Air combine_everything %
```

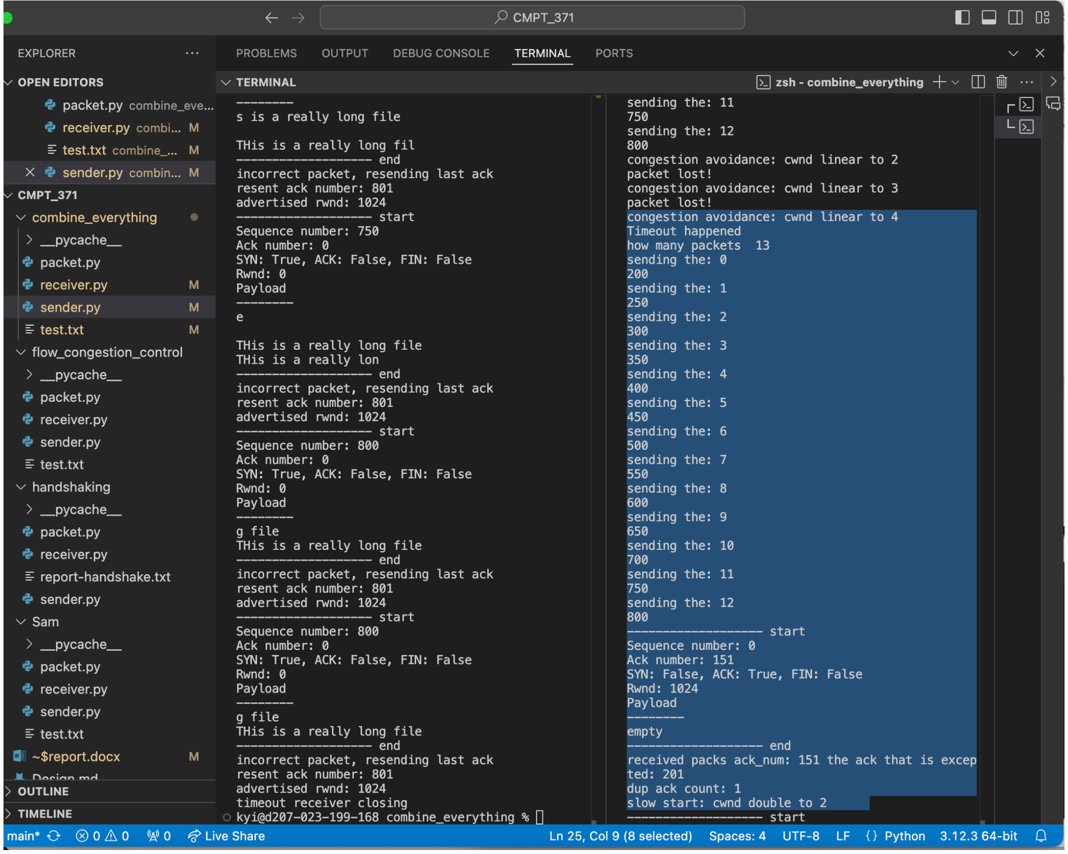
```
timeout receiver closing
kyi@kyi-MacBook-Air combine_everything % python3 receiver.py
The server is ready to hand shake
The client is alive!
Receiver is ready to receive
------------------- start
Sequence number: 0
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
THis is a really long file
THis is a really long f
------------------- end
correct packet sending response with ack number: 1
latest ack number: 1
advertised rwnd: 1024
simulating packet loss
simulating packet loss
------------------- start
Sequence number: 50
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
ile

THis is a really long file

THis is a really
------------------- end
correct packet sending response with ack number: 51
latest ack number: 51
advertised rwnd: 1024
------------------- start
Sequence number: 100
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
long file


THis is a really long file

THis is a
------------------- end
correct packet sending response with ack number: 101
latest ack number: 101
advertised rwnd: 1024
------------------- start
Sequence number: 150
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
really long file
------------------- end
correct packet sending response with ack number: 151
latest ack number: 151
advertised rwnd: 1024
timeout receiver closing
kyi@kyi-MacBook-Air combine_everything %
```



```
received packs ack_num: 301 the ack that is excepted: 301
correct packet
congestion avoidance: cwnd linear to 3
packet lost!
congestion avoidance: cwnd linear to 4
packet lost!
congestion avoidance: cwnd linear to 5
Timeout happened
how many packets  3
sending the: 0
350
sending the: 1
400
sending the: 2
450
------------------- start
Sequence number: 0
Ack number: 401
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 401 the ack that is excepted: 351
correct packet
slow start: cwnd double to 2
------------------- start
Sequence number: 0
Ack number: 401
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 401 the ack that is excepted: 451
dup ack count: 1
slow start: cwnd double to 4
------------------- start
Sequence number: 0
Ack number: 401
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 401 the ack that is excepted: 451
dup ack count: 2
congestion avoidance: cwnd linear to 5
Timeout happened
how many packets  1
sending the: 0
450
------------------- start
Sequence number: 0
Ack number: 401
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
------------------- end
received packs ack_num: 401 the ack that is excepted: 451
dup ack count: 3
sending the: 0
```

```
correct packet sending response with ack number: 301
latest ack number: 301
advertised rwnd: 1024
------------------- start
Sequence number: 350
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
y long file

THis is a really long file
THis is a
------------------- end
correct packet sending response with ack number: 351
latest ack number: 351
advertised rwnd: 1024
------------------- start
Sequence number: 400
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
really long file
THis is a really long file

THis
------------------- end
correct packet sending response with ack number: 401
latest ack number: 401
advertised rwnd: 1024
simulating packet loss
------------------- start
Sequence number: 100
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
long file


THis is a really long file

THis is a
------------------- end
incorrect packet, resending last ack
resent ack number: 401
advertised rwnd: 1024
------------------- start
Sequence number: 150
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
really long file

THis is a really long file
THis
------------------- end
incorrect packet, resending last ack
resent ack number: 401
advertised rwnd: 1024
simulating packet loss
```

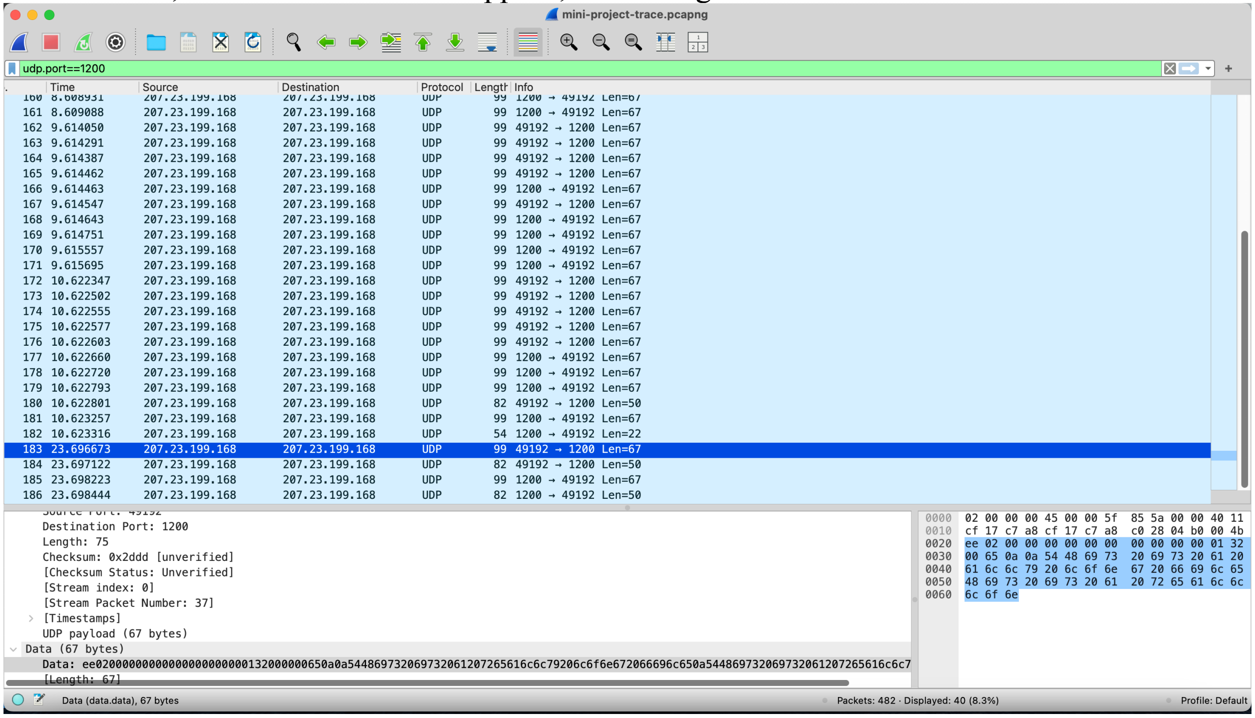
When timeout or sender receive duplicate acks for 3 times, the ssthresh will be halved down, and the window size will be reset to 1, and do slow start again:



```
--------
s is a really long file

THis is a really long fil
-------------------- end
incorrect packet, resending last ack
resent ack number: 801
advertised rwnd: 1024
-------------------- start
Sequence number: 750
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
e

THis is a really long file
THis is a really lon
-------------------- end
incorrect packet, resending last ack
resent ack number: 801
advertised rwnd: 1024
-------------------- start
Sequence number: 800
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
g file
THis is a really long file
-------------------- end
incorrect packet, resending last ack
resent ack number: 801
advertised rwnd: 1024
-------------------- start
Sequence number: 800
Ack number: 0
SYN: True, ACK: False, FIN: False
Rwnd: 0
Payload
--------
g file
THis is a really long file
-------------------- end
incorrect packet, resending last ack
resent ack number: 801
advertised rwnd: 1024
timeout receiver closing
kyi@d207-023-199-168 combine_everything % 
```

```
sending the: 11
750
sending the: 12
800
congestion avoidance: cwnd linear to 2
packet lost!
congestion avoidance: cwnd linear to 3
packet lost!
congestion avoidance: cwnd linear to 4
Timeout happened
how many packets  13
sending the: 0
200
sending the: 1
250
sending the: 2
300
sending the: 3
350
sending the: 4
400
sending the: 5
450
sending the: 6
500
sending the: 7
550
sending the: 8
600
sending the: 9
650
sending the: 10
700
sending the: 11
750
sending the: 12
800
-------------------- start
Sequence number: 0
Ack number: 151
SYN: False, ACK: True, FIN: False
Rwnd: 1024
Payload
--------
empty
-------------------- end
received packs ack_num: 151 the ack that is excep
ted: 201
dup ack count: 1
slow start: cwnd double to 2
-------------------- start
```

In Wireshark, it shows the timeout happens, it did congestion control:



.	Time	Source	Destination	Protocol	Length	Info
160	8.608931	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
161	8.609088	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
162	9.614050	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
163	9.614291	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
164	9.614387	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
165	9.614462	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
166	9.614463	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
167	9.614547	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
168	9.614643	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
169	9.614751	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
170	9.615557	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
171	9.615695	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
172	10.622347	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
173	10.622502	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
174	10.622555	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
175	10.622577	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
176	10.622603	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
177	10.622660	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
178	10.622720	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
179	10.622793	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
180	10.622801	207.23.199.168	207.23.199.168	UDP	82	49192 → 1200 Len=50
181	10.623257	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
182	10.623316	207.23.199.168	207.23.199.168	UDP	54	1200 → 49192 Len=22
183	23.696673	207.23.199.168	207.23.199.168	UDP	99	49192 → 1200 Len=67
184	23.697122	207.23.199.168	207.23.199.168	UDP	82	49192 → 1200 Len=50
185	23.698223	207.23.199.168	207.23.199.168	UDP	99	1200 → 49192 Len=67
186	23.698444	207.23.199.168	207.23.199.168	UDP	82	1200 → 49192 Len=50

```
    Source Port: 49192
    Destination Port: 1200
    Length: 75
    Checksum: 0x2ddd [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 37]
  > [Timestamps]
    UDP payload (67 bytes)
v Data (67 bytes)
    Data: ee020000000000000000000001320000000650a0a544869732069732061207265616c6c79206c6f6e672066696c650a5448697320697320612072656616c6c7
    [Length: 67]
```

```
0000  02 00 00 00 45 00 00 5f  85 5a 00 00 40 11
0010  cf 17 c7 a8 cf 17 c7 a8  c0 28 04 b0 00 4b
0020  ee 02 00 00 00 00 00 00  00 00 00 00 01 32
0030  00 65 0a 0a 54 48 69 73  20 69 73 20 61 20
0040  61 6c 6c 79 20 6c 6f 6e  67 20 66 69 6c 65
0050  48 69 73 20 69 73 20 61  20 72 65 61 6c 6c
0060  6c 6f 6e
```

Data (data.data), 67 bytes
Packets: 482 · Displayed: 40 (8.3%)
Profile: Default