

## 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 simulates 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-acked 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 (sssthresh) or a packet loss is detected.

**Congestion avoidance:** When cwnd exceeds sssthresh, 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 sssthresh 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 expected:', sndpkt[0].sequence_num+1)

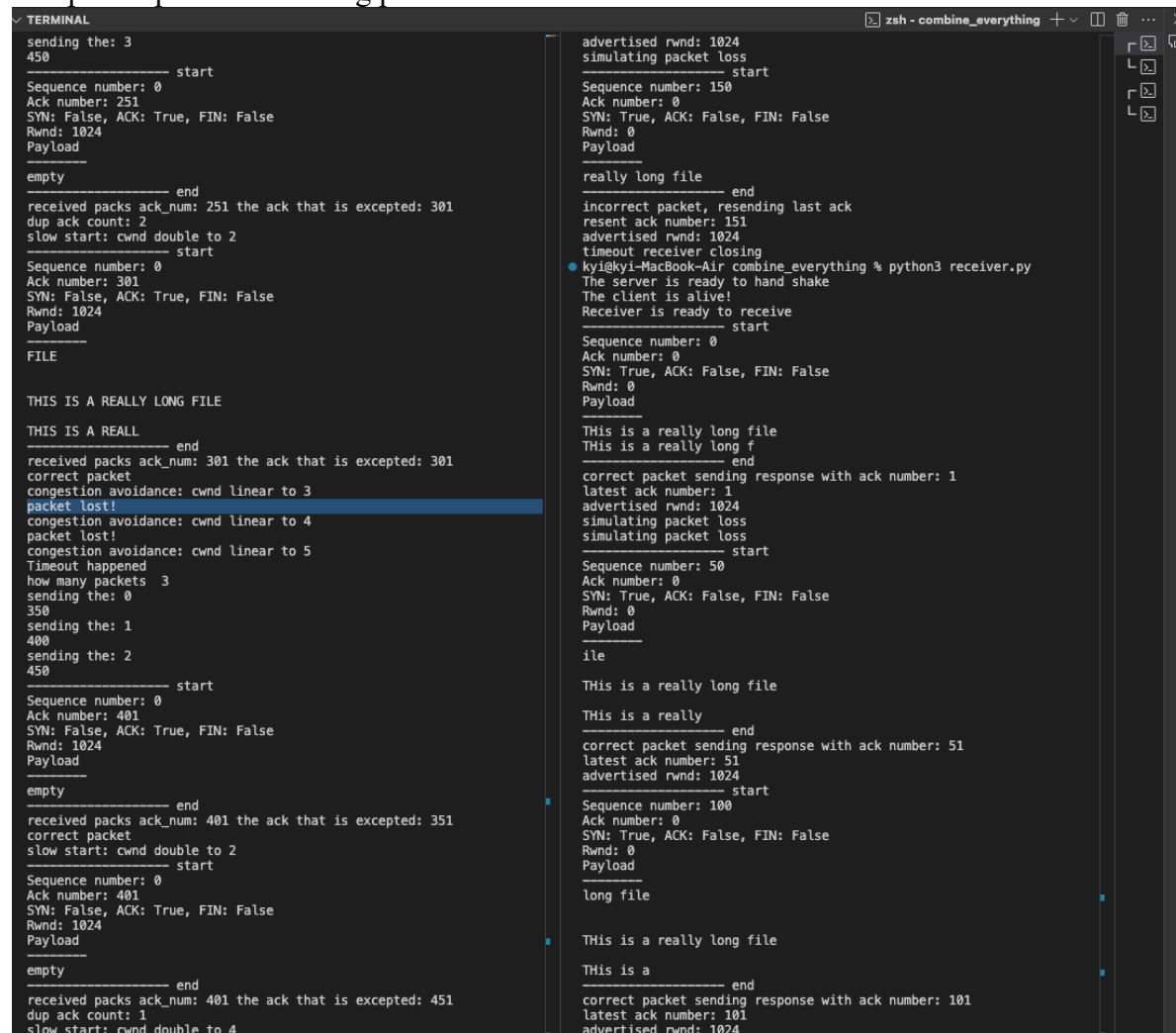
    # if it is a dup ack, increase the counter
    if response_packet.ack_num == prev_ack_num:
        dupAckCount+=1
        print(f'dup ack count: {dupAckCount}')
    else: # if it is a new ack, reset the new ack counter
        dupAckCount=0

    # TCP Tahoe, 3 dup acks, reset the cwnd, resend pkts
    if dupAckCount==3:
        # retransmit
        for i, pkt in enumerate(sndpkt):
            print('sending the:', i)
            print(pkt.sequence_num)
            senderSocket.sendto(pkt.change_to_bytes(), (receiverName, receiverPort))

        cwnd = 1
        sssthresh = 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:



```

TERMINAL
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
-----
advised 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
advised 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
advised 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
advised 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
advised rwnd: 1024

```

### Proof of connection-oriented:

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.

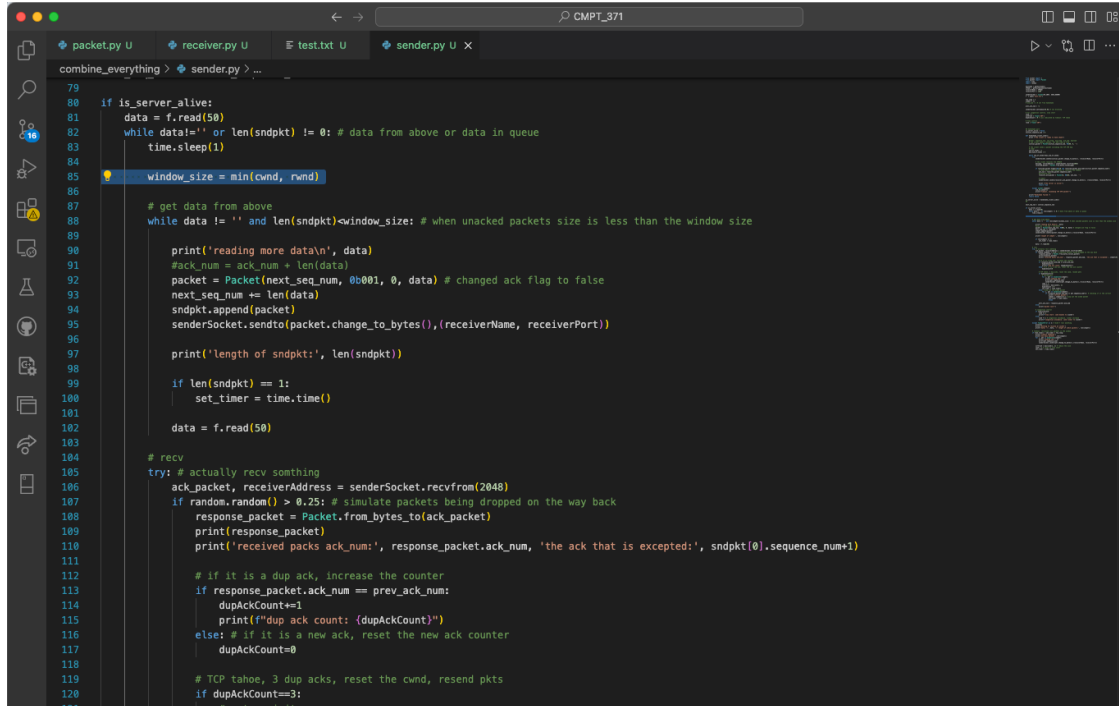
```
handshaking > sender.py > ...
15 # syn, ack, fin
16
17 def handshake_client_side():
18     print("The client is ready to hand shake")
19
20     #order: sequence_num, syn_flag, ack_flag, ack_num, payload
21     # the client makes a packet, set SYN flag to true
22     initial_packet = Packet(initial_sequence_num, 0b100, 0, '')
23
24     # the client sends a packet including the TCP SYN msg
25     # timer
26     num_of_sends = 1
27     max_num_of_sends = 4
28
29     while num_of_sends<=max_num_of_sends:
30         try:
31             print("The initial sequence number from sender is: ", initial_sequence_num)
32             clientSocket.sendto(initial_packet.change_to_bytes(), (serverName, serverPort))
33
34             # the client receives the SYNACK
35             message, serverAddress = clientSocket.recvfrom(2048)
36             received_packet = Packet.from_bytes_to(message)
37
38             if received_packet.flags==0b110 and received_packet.ack_num==initial_packet.sequence_num+1:
39                 # sends back a ACK indicates it receives the SYNACK
40                 print("The SYN and ACK flags are true, received SYNACK.")
41                 ack_num = received_packet.sequence_num+1
42                 print("The SYNACK number is: ", ack_num)
43                 receiver_ack_packet = Packet(0, 0b010, ack_num, '')
44
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
▼ TERMINAL
o 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!
[]
o 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!
o 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:

```
combine_everything > receiver.py > [0] buffer_size
39
60 # flow control
61 buffer_size = 1024
62 buffer_used = 0
63
64 flag = True
65 while flag:
66     try:
67         message, senderAddress = receiverSocket.recvfrom(2048)
68     except:
69         print('timeout receiver closing')
70         break
71
72     packet = Packet.from_bytes_to(message)
73
74     if random.random() > 0.25: # pretend to receive packet
75         print(packet)
76
77     # check if there is enough buffer
78     if buffer_used+len(packet.payload)<=buffer_size:
79         if nextseqnum == packet.sequence_num:
80             # if it is he next packet respond with the correct ack
81             # else respond with the previous ack and keep doing that
82             # until the right packet is sent
83             payload = packet.payload
84             sndpkt = Packet(0, 0b010, nextseqnum+1, packet.payload.upper())
85             print('correct packet sending response with ack number:', nextseqnum+1)
86             nextseqnum += len(packet.payload)
87             last_ack_num = packet.sequence_num+1
88             print('latest ack number:', last_ack_num)
89         else:
90             print('incorrect packet, resending last ack')
91             sndpkt = Packet(0, 0b010, last_ack_num, 'empty')
92             print('resent ack number:', sndpkt.ack_num)
93     else:
94         print("buffer full")
95         sndpkt = Packet(0, 0b010, last_ack_num, 'empty')
96         sndpkt.rwnd = buffer_size-buffer_used
97         print('advertised rwnd:', sndpkt.rwnd)
98         receiverSocket.sendto(sndpkt.change_to_bytes(), senderAddress)
99     else:
100         print('simulating packet loss')
101
```

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:



```
79
80 if is_server_alive:
81     data = f.read(50)
82     while data!='' or len(sndpkt) != 0: # data from above or data in queue
83         time.sleep(1)
84
85     window_size = min(cwnd, rwnd)
86
87     # get data from above
88     while data != '' and len(sndpkt) < window_size: # when unacked packets size is less than the window size
89
90         print('reading more data\n', data)
91         #ack_num = ack_num + len(data)
92         packet = Packet(next_seq_num, 0b001, 0, data) # changed ack flag to false
93         next_seq_num += len(data)
94         sndpkt.append(packet)
95         senderSocket.sendto(packet.change_to_bytes(), (receiverName, receiverPort))
96
97     print('length of sndpkt:', len(sndpkt))
98
99     if len(sndpkt) == 1:
100         set_timer = time.time()
101
102     data = f.read(50)
103
104 # rcv
105 try: # actually rcv something
106     ack_packet, receiverAddress = senderSocket.recvfrom(2048)
107     if random.random() > 0.25: # simulate packets being dropped on the way back
108         response_packet = Packet.from_bytes_to(ack_packet)
109         print(response_packet)
110         print('received packs ack_num:', response_packet.ack_num, 'the ack that is expected:', sndpkt[0].sequence_num+1)
111
112     # if it is a dup ack, increase the counter
113     if response_packet.ack_num == prev_ack_num:
114         dupAckCount+=1
115         print(f'dup ack count: {dupAckCount}')
116     else: # if it is a new ack, reset the new ack counter
117         dupAckCount=0
118
119     # TCP Tahoe, 3 dup acks, reset the cwnd, resend pkts
120     if dupAckCount==3:
121         # reset cwnd
```

## Proof of congestion control:

```
✓ TERMINAL
● 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
```

```
✓ TERMINAL zsh - combine_everything + ▢ ▢ ... >

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 %
```

```
✓ TERMINAL zsh - combine_everything + - ... >
received packs ack_num: 301 the ack that is expected: 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 expected: 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 expected: 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 expected: 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 expected: 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
```