

מטלה 3 - רשתות תקשורת

Table of Content

Background	2
Congestion control	2
Reno	2
Cubic	3
The process	4
Running the program Packet Loss:	7
0% Packet Loss:	7
10% Packet Loss:	8
15% Packet Loss:	9
20% Packet Loss:	10
25% Packet Loss:	11
Conclusion	12
Packet loss - transferring time	12

Background

This exercise aims to implement two classes “Sender” and “Receiver” in Python using PyCharm. The main goal of this code is simulating a text file of 2MB transferring between a sender and client. This is achieved using congestion control algorithms: “Cubic” and “Reno” (learnt in class) and using the TCP as the Transport protocol of the packets .

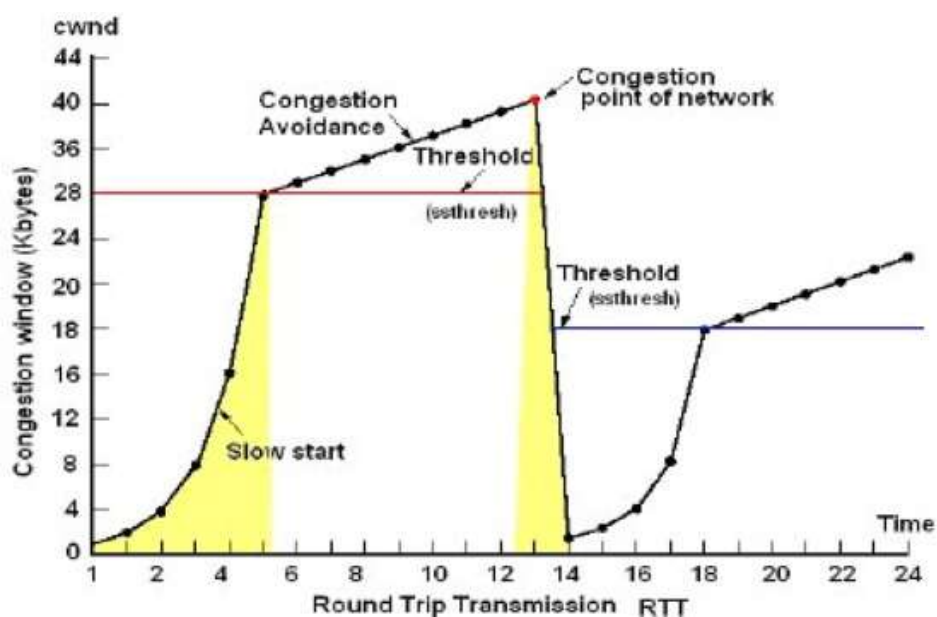
Congestion control

TCP is a classic standard which enables application programs and computing devices to exchange messages over a network. The congestion control ensures efficient packet sending and prevents congestion (traffic) in the process and by that decreases the number of packets lost.

Reno

The Reno TCP implements the additive increase multiplicative decrease (AIMD) algorithm for congestion control.¹ This algorithm uses “slow start” which increases the size of the window by 1 at first (so its current size is 2), if it gets ACK which means the transfer of packets was successful it continues and increases the window's size by 2 (window size = 4). This process goes on and doubles the window's size every time ACK is received (Slope #1 - slow start) in the image below² until it faces a threshold in which case it begins to increase the window linearly (Slope #2 - congestion avoidance). Finally when reaching a “problem” we halve the size of the current window.

³



¹ <https://www.linkedin.com/advice/0/what-pros-cons-using-tcp-reno-vs-cubic>

² <https://media.geeksforgeeks.org/wp-content/uploads/20220202203124/Renocongestionwindow.png>

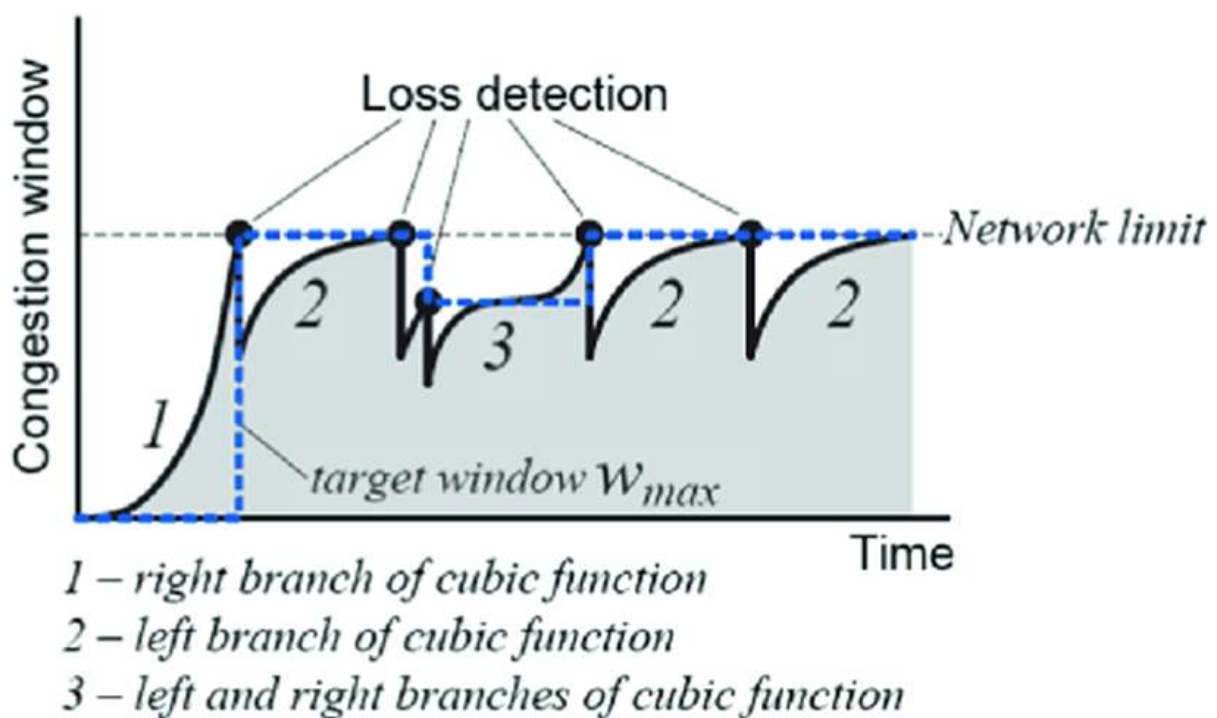
³ <https://media.geeksforgeeks.org/wp-content/uploads/20220202203124/Renocongestionwindow.png>

Cubic

TCP Cubic is a newer TCP variant that is designed for high-speed networks with a high bandwidth-delay product.⁴ The algorithm also uses "Fast retransmission" which is a TCP extension that allows saving time when a packet is lost, the main difference is that instead of waiting the timeout (which could be a waste of time) the extension assumes that if there are 3 duplicate ACK, the packet was lost and it resends it. "Fast Recovery" is also used, which helps dealing with lost packets more quickly.

The way Cubic works is to initialize the window size to a small value, the window's size increases by a cubic function (hence the name of the algorithm) the window increases until reaching a threshold, this is similar to the "slow start" in Reno and is marked in the following image as Slope 1, the only difference here is that instead of multiplying the window size by 2 we increase it by x^3 .

When there is load on the network (which is likely to cause congestion and loss of packets) the sending window needs to be reduced, as explained in Reno the window size halves.



5

⁴ <https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf>

⁵ <https://www.researchgate.net/publication/341995073/figure/fig1/AS:1019836431863809@1620159361369/Cwnd-behavior-in-CUBIC-TCP-14.png>

The process

The sender class begins by opening and reading the text file (in binary):

```
def send_file(filename, host, port):  
    file_size = os.path.getsize(filename)  
    part_size = file_size // 2  
    with open(filename, 'rb') as f:  
        data = f.read()
```

After doing so the code continues with an infinite loop unless the sender wants to stop sending the file. Then we declare “socket.socket(socket.AF_INET, socket.SOCK_STREAM)” which is used to create a TCP/IP socket, which can be used to establish a network connection between the sender and receiver, socket.AF_INET and socket.SOCK_STREAM is used to create the socket using the IPv4 protocol.

```
while True:  
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    s.connect((host, port))
```

Next we're setting the cc algorithm using the s.setsockopt() which sets the socket options. Note the encode() function in the sender class which is necessary to send information between the two as when sending data over a network, it has to be in the form of bytes. After setting the congestion control algorithm to “cubic”, the code sends the first half of a file over the TCP connection using the sendall() function. Once the process is over we close the connection using socket.close() method.

```
s.setsockopt(socket.IPPROTO_TCP, socket.TCP_CONGESTION, "cubic".encode())  
# Send first half of file  
s.sendall(data[:part_size])  
s.close()
```

Next, we move into the receiver's class where we use the bind() which assigns an IP address and a port number to a socket instance and listen() function in order to listen for incoming connections, once a connection is found it is accepted by the socket and returns a connection as well as an address. The receiver receives the first part of the text using the recv() function and then writes it into the disk (in binary) and calculates the time it takes to receive that part of the text.

```

def receive_file(filename, host, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        while True:
            conn, addr = s.accept()
            with open(filename, 'wb') as f:
                conn.setsockopt(socket.IPPROTO_TCP, socket.TCP_CONGESTION, "cubic".encode())
                # Receive first half of file
                start_time = time.time()
                data = ""
                while True:
                    chunk = conn.recv(1000000).decode()
                    print(chunk.__sizeof__())
                    if len(chunk) <= 0:
                        break
                    data += chunk
                f.write(data.encode())
                end_time = time.time()
                print(f'Received first half in {end_time - start_time} seconds')
                time_pt1.append(end_time - start_time)
            |
            conn, addr = s.accept()

```

Finally, an authentication (which is a xor between the two IDs) is sent to the sender just before the connection is closed.

```

# Send authentication:
conn.sendall(str(ID1 ^ ID2).encode())

conn.close()

```

Moving back to the sender class, the authentication is received and being compared with the $ID_1 \wedge ID_2$ to check if there isn't a match, we terminate the collection and placing a message to the screen, else, a socket is being set up, tries to connect to the receiver and sends the second and last part of the text file and terminates later. **We are then closing the connection to let the server know that the sender stopped sending data.**(The while loop in the receiver will break since)

```

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))

# Receive the authentication from the server
auth = s.recv(5)
if auth.decode() == str(ID1 ^ ID2):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    # Send second half of file
    s.setsockopt(socket.IPPROTO_TCP, socket.TCP_CONGESTION, "reno".encode())
    s.sendall(data[part_size:])
    s.close()
else:
    s.close()
    print("Connection was cut")
    break

```

Meanwhile, the receiver just like in the first part tries to make a connection with the sender and receive the rest of the file, write it in binary into the disk and of course records the amount of time taken to receive that information.

Last but not least, a question appears on the screen asking whether we'd like to resend the file, if we do we must press 'y' and the file is being resent, once we've had enough we press 'n' and the socket is closed, the server gets a notification that the process is DONE!

```

# Ask user if they want to send the file again
send_again = input('Send file again? (y/n): ')
if send_again.lower() != 'y':
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.sendall("_done_sending_all_".encode())
    s.close()
    break
else:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.sendall("_keep_sending_".encode())
    s.close()

```

Sender

```

conn, addr = s.accept()
is_done = conn.recv(50).decode()
if is_done == "_done_sending_all_":
    break
else:
    conn.close()

```

Receiver

Running the program Packet Loss:

Note that if we get one the following: retransmission, fast transmission, dup ack, out of order messages when “standing” on one of the packets with the mouse, it means that the current packet was lost.

0% Packet Loss:

In the Sender.py we sent the file 15 times. The output is on the Reciever.py, with two Time lists, each list contains the times it took for the corresponding half of the file. We also calculated the average of each part's time:

```
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$ python3 Reciever.py
Received first half in 0.08959555625915527 seconds
Received second half in 0.14330530160625977 seconds
Received first half in 0.013621807098388672 seconds
Received second half in 0.015645742416381836 seconds
Received first half in 0.007117509841918945 seconds
Received second half in 0.013411521911621094 seconds
Received first half in 0.00948953028540039 seconds
Received second half in 0.04862618446350098 seconds
Received first half in 0.006170749664306641 seconds
Received second half in 0.009211063385009766 seconds
Received first half in 0.012079238891601562 seconds
Received second half in 0.01346278190612793 seconds
Received first half in 0.0034759844647216797 seconds
Received second half in 0.00439080129699787 seconds
Received first half in 0.0048596858978271484 seconds
Received second half in 0.011658430099487305 seconds
Received first half in 0.005337953567504883 seconds
Received second half in 0.05243825912475586 seconds
Received first half in 0.00910043716430664 seconds
Received second half in 0.00922942161500586 seconds
Received first half in 0.005564689636230469 seconds
Received second half in 0.05639982223510742 seconds
Received first half in 0.011380672454833984 seconds
Received second half in 0.05073118209838867 seconds
Received first half in 0.018256187438964844 seconds
Received second half in 0.015533685684204102 seconds
Received first half in 0.018079042434692383 seconds
Received second half in 0.01066549682617188 seconds
Received first half in 0.018444299697875977 seconds
Received second half in 0.05318021774291992 seconds
[0.08959555625915527, 0.013621807098388672, 0.007117509841918945, 0.00948953028540039, 0.006170749664306641, 0.012079238891601562, 0.0034759844647216797, 0.0048596858978271484, 0.005337953567504883, 0.00910043716430664, 0.005564689636230469, 0.011380672454833984, 0.018256187438964844, 0.018079042434692383, 0.018444299697875977]
Avg time for part 1 is: 0.015504884719848632
[0.14330530160625977, 0.015645742416381836, 0.013411521911621094, 0.04862618446350098, 0.009211063385009766, 0.01346278190612793, 0.00439080129699787, 0.011658430099487305, 0.05243825912475586, 0.00922942161500586, 0.05639982223510742, 0.05073118209838867, 0.015533685684204102, 0.01066549682617188, 0.05318021774291992]
Avg time for part 2 is: 0.03992594083150228
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$
```

```
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): n
```

0_packet_loss.pcapng

Time	Source	Destination	Protocol	Length	Info
1326	12.218114514	127.0.0.1	TCP	66	46278 → 8000 [ACK] Seq=20 Ack=2 Win=65536 Len=0 TSval=1720363536 TSecr=1720363536
1325	12.218088032	127.0.0.1	TCP	66	8000 → 46278 [FIN, ACK] Seq=1 Ack=20 Win=65536 Len=0 TSval=1720363536 TSecr=1720363531
1324	12.213580163	127.0.0.1	TCP	66	[TCP Previous segment not captured] 46270 → 8000 [ACK] Seq=1083871 Ack=2 Win=65536 Len=0 TSval=1720363531 TSecr=1720363531
1323	12.213568775	127.0.0.1	TCP	66	[TCP ACKed unseen segment] 8000 → 46270 [FIN, ACK] Seq=1 Ack=1083871 Win=2421632 Len=0 TSval=1720363531 TSecr=1720362251
1322	12.213497922	127.0.0.1	TCP	66	46278 → 8000 [FIN, ACK] Seq=19 Ack=1 Win=65536 Len=0 TSval=1720363531 TSecr=1720363531
1321	12.213385181	127.0.0.1	TCP	66	8000 → 46278 [ACK] Seq=1 Ack=19 Win=65536 Len=0 TSval=1720363531 TSecr=1720363531
1320	12.213376901	127.0.0.1	TCP	84	46278 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=18 TSval=1720363531 TSecr=1720363531
1319	12.213366280	127.0.0.1	TCP	66	46278 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1720363531 TSecr=1720363531
1318	12.213294438	127.0.0.1	TCP	74	8000 → 46278 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1720363531 TSecr=1720363531 WS=128
1317	12.213280423	127.0.0.1	TCP	74	46278 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1720363531 TSecr=0 WS=128
1316	10.934116773	127.0.0.1	TCP	66	[TCP ACKed unseen segment] 8000 → 46278 [ACK] Seq=1 Ack=1083871 Win=2421632 Len=0 TSval=1720362251 TSecr=1720362251
1315	10.933442458	127.0.0.1	TCP	66	8000 → 46278 [ACK] Seq=1 Ack=523857 Win=1113216 Len=0 TSval=1720362251 TSecr=1720362251
1314	10.933439242	127.0.0.1	TCP	32807	46270 → 8000 [ACK] Seq=982231 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1313	10.933432821	127.0.0.1	TCP	32807	46270 → 8000 [PSH, ACK] Seq=949499 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1312	10.933406012	127.0.0.1	TCP	66	8000 → 46278 [ACK] Seq=1 Ack=491116 Win=982272 Len=0 TSval=1720362251 TSecr=1720362251
1311	10.933402636	127.0.0.1	TCP	32807	46270 → 8000 [ACK] Seq=910749 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1310	10.933395063	127.0.0.1	TCP	32807	46270 → 8000 [PSH, ACK] Seq=884008 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1309	10.933366030	127.0.0.1	TCP	66	8000 → 46278 [ACK] Seq=1 Ack=458375 Win=851328 Len=0 TSval=1720362251 TSecr=1720362251
1308	10.933362885	127.0.0.1	TCP	32807	46270 → 8000 [ACK] Seq=851267 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1307	10.933356552	127.0.0.1	TCP	32807	46270 → 8000 [PSH, ACK] Seq=818526 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1306	10.933327060	127.0.0.1	TCP	66	8000 → 46278 [ACK] Seq=1 Ack=425634 Win=720384 Len=0 TSval=1720362251 TSecr=1720362251
1305	10.933323774	127.0.0.1	TCP	32807	46270 → 8000 [ACK] Seq=785785 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1304	10.933315880	127.0.0.1	TCP	32807	46270 → 8000 [PSH, ACK] Seq=753644 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1303	10.933287928	127.0.0.1	TCP	66	8000 → 46278 [ACK] Seq=1 Ack=392093 Win=569440 Len=0 TSval=1720362251 TSecr=1720362251
1302	10.933272762	127.0.0.1	TCP	32807	46270 → 8000 [ACK] Seq=726393 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251
1301	10.933265750	127.0.0.1	TCP	32807	46270 → 8000 [PSH, ACK] Seq=687562 Ack=1 Win=65536 Len=32741 TSval=1720362251 TSecr=1720362251

```
1009 ... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window: 512
[Calculated window size: 65536]
[Window size scaling factor: 128]
Checksum: 0xadfa [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
```


10% Packet Loss:

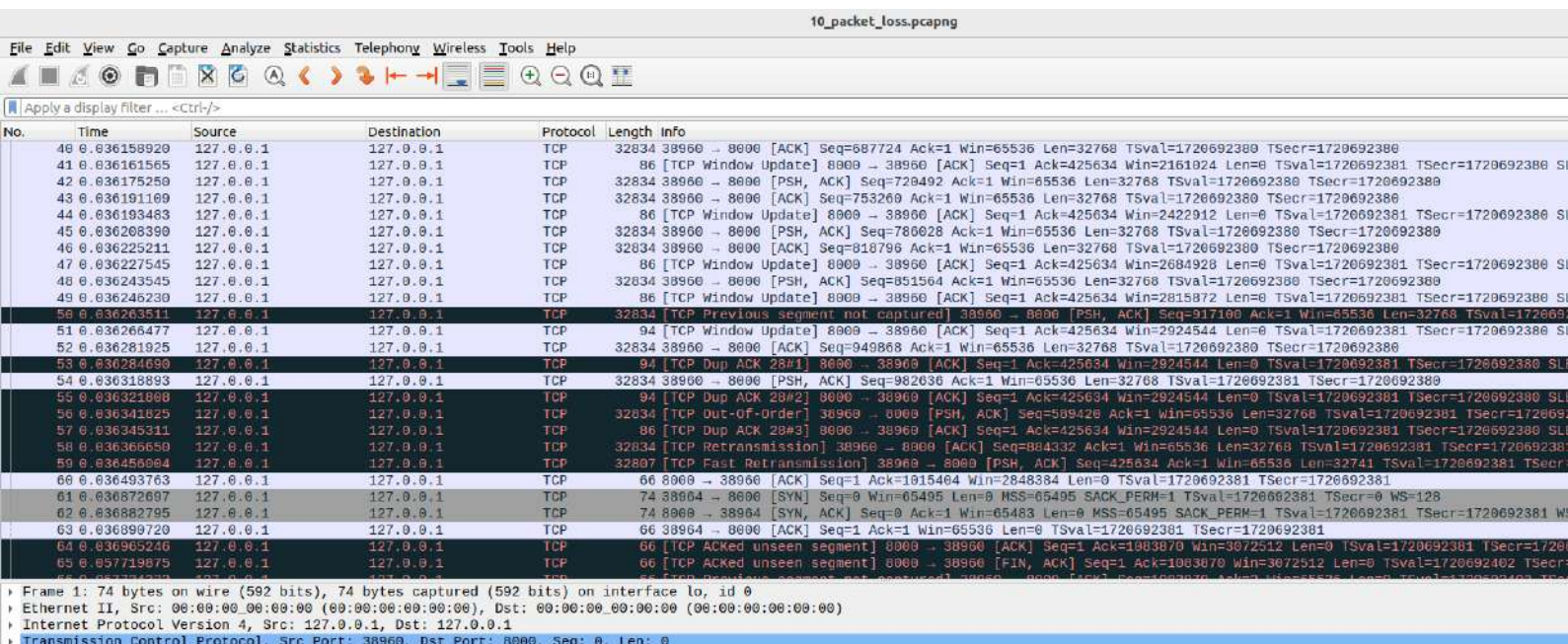
In the Sender.py we sent the file 15 times with 10% packet loss. The output is on the Reciever.py, with two Time lists, each list contains the times it took for the corresponding half of the file. We also calculated the average of each part's time. We can see in the wireshark's screenshot that the packets in black are lost, duplicate acks, out of order packets, etc.

```

yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$ python3 Reciever.py
Received first half in 0.05643343925476074 seconds
Received second half in 0.16506576538005938 seconds
Received first half in 0.03851056090937988 seconds
Received second half in 0.04074539375305176 seconds
Received first half in 0.010231404903564453 seconds
Received second half in 0.050568342208862305 seconds
Received first half in 0.006062030792236328 seconds
Received second half in 0.026092052459716797 seconds
Received first half in 0.048486948013305664 seconds
Received second half in 0.050725460052490234 seconds
Received first half in 0.2277982234954834 seconds
Received second half in 0.02234625816345215 seconds
Received first half in 0.24602794647216797 seconds
Received second half in 0.02848672866821280 seconds
Received first half in 0.028843029022216797 seconds
Received second half in 0.031054258346557617 seconds
Received first half in 0.004027366638183594 seconds
Received second half in 0.03165173530578613 seconds
Received first half in 0.004185199737548828 seconds
Received second half in 0.2548079490661621 seconds
Received first half in 0.0030013126373291016 seconds
Received second half in 0.45357275009155273 seconds
Received first half in 0.22075700759887695 seconds
Received second half in 0.4342679977416992 seconds
Received first half in 0.003516674041748047 seconds
Received second half in 0.06717991828918457 seconds
Received first half in 0.06174659729003906 seconds
Received second half in 0.03650188446044922 seconds
Received first half in 0.05237245559692383 seconds
Received second half in 0.042871713638305664 seconds
[0.05643343925476074, 0.03851056090937988, 0.010231404903564453, 0.006062030792236328, 0.048486948013305664, 0.050725460052490234, 0.2277982234954834, 0.24602794647216797, 0.02848672866821280, 0.028843029022216797, 0.031054258346557617, 0.004027366638183594, 0.004185199737548828, 0.2548079490661621, 0.0030013126373291016, 0.45357275009155273, 0.22075700759887695, 0.4342679977416992, 0.003516674041748047, 0.06717991828918457, 0.06174659729003906, 0.03650188446044922, 0.05237245559692383]
Avg time for part 1 is: 0.06697335243225067
[0.16506576538005938, 0.04874539375305176, 0.050568342208862305, 0.026092052459716797, 0.050725460052490234, 0.02234625816345215, 0.02848672866821280, 0.031054258346557617, 0.03165173530578613, 0.2548079490661621, 0.45357275009155273, 0.4342679977416992, 0.06717991828918457, 0.03650188446044922, 0.042871713638305664]
Avg time for part 2 is: 0.1162625471508952
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$

Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): n
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$

```



15% Packet Loss:

In the Sender.py we sent the file 15 times with 15% packet loss. The output is on the Reciever.py, with two Time lists, each list contains the times it took for the corresponding half of the file. We also calculated the average of each part's time:

```
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject$ python3 Reciever.py
Received first half in 0.23745965957641602 seconds
Received second half in 0.4323761463165283 seconds
Received first half in 0.45241498947143555 seconds
Received second half in 1.0726017951965332 seconds
Received first half in 0.2518388102689209 seconds
Received second half in 0.04768991470336914 seconds
Received first half in 0.6782848834991455 seconds
Received second half in 0.5822141170501709 seconds
Received first half in 0.49785542488089145 seconds
Received second half in 1.4852008019580078 seconds
Received first half in 0.45798349380493164 seconds
Received second half in 0.011821746826171875 seconds
Received first half in 0.5185422897338867 seconds
Received second half in 0.027641773223876953 seconds
Received first half in 0.018190622329711914 seconds
Received second half in 3.253708839416504 seconds
Received first half in 0.5415024757385254 seconds
Received second half in 0.49889765548706055 seconds
Received first half in 0.47856903076171875 seconds
Received second half in 0.28269076347351074 seconds
Received first half in 0.26219828976257324 seconds
Received second half in 0.4221152534080633 seconds
Received first half in 0.23019647598266602 seconds
Received second half in 0.031673431396484375 seconds
Received first half in 0.7235367298126221 seconds
Received second half in 0.00340289115905762 seconds
Received first half in 0.7630398273468018 seconds
Received second half in 0.36771488189697266 seconds
Received first half in 0.4334249496459961 seconds
Received second half in 7.499902009963989 seconds
[0.23745965957641602, 0.45241498947143555, 0.2518388102689209, 0.6782848834991455, 0.49785542488089145, 0.45798349380493164, 0.5185422897338867, 0.011821746826171875, 0.5415024757385254, 0.47856903076171875, 0.26219828976257324, 0.23019647598266602, 0.7235367298126221, 0.7630398273468018, 0.4334249496459961]
Avg time for part 1 is: 0.43633532524108887
[0.4323761463165283, 1.0726017951965332, 0.04768991470336914, 0.5822141170501709, 1.4852008019580078, 0.011821746826171875, 0.027641773223876953, 3.253708839416504, 0.49889765548706055, 0.28269076347351074, 0.4221152534080633, 0.23019647598266602, 0.031673431396484375, 0.7235367298126221, 0.7630398273468018, 0.4334249496459961]
Avg time for part 2 is: 1.071510140101115
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject$

Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): n
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject$
```

15_packet_loss.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter (e.g. <Ctrl>F)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	35788 → 8000 [SYN, Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1739549336 TSecr=0 WS=128
2	0.001343952	127.0.0.1	127.0.0.1	TCP	74	8000 → 35788 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1739549337 TSecr=1739549336 WS=128
3	0.002510963	127.0.0.1	127.0.0.1	TCP	66	35788 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1739549337 TSecr=1739549337
4	0.007355150	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=32741 TSval=1739549343 TSecr=1739549337
5	0.007377612	127.0.0.1	127.0.0.1	TCP	66	8000 → 35788 [ACK] Seq=1 Ack=32742 Win=48040 Len=0 TSval=1739549343 TSecr=1739549343
6	0.007416715	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [PSH, ACK] Seq=32742 Ack=1 Win=65536 Len=32741 TSval=1739549343 TSecr=1739549337
7	0.014976847	127.0.0.1	127.0.0.1	TCP	74	35792 → 8000 [SYN, Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1739549351 TSecr=0 WS=128
8	0.015904493	127.0.0.1	127.0.0.1	TCP	74	8000 → 35792 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1739549352 TSecr=1739549351 WS=128
9	0.015939949	127.0.0.1	127.0.0.1	TCP	66	35792 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1739549352 TSecr=1739549352
10	0.215186476	127.0.0.1	127.0.0.1	TCP	32807	[TCP Retransmission] 35788 → 8000 [PSH, ACK] Seq=32742 Ack=1 Win=65536 Len=32741 TSval=1739549551 TSecr=1739549343
11	0.215201999	127.0.0.1	127.0.0.1	TCP	78	8000 → 35788 [ACK] Seq=1 Ack=65483 Win=65536 Len=0 TSval=1739549551 TSecr=1739549551 SLE=32742 SRE=65483
12	0.215231634	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [ACK] Seq=65483 Ack=1 Win=65536 Len=32741 TSval=1739549551 TSecr=1739549551
13	0.215294671	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [PSH, ACK] Seq=98224 Ack=1 Win=65536 Len=32741 TSval=1739549551 TSecr=1739549551
14	0.215710726	127.0.0.1	127.0.0.1	TCP	66	8000 → 35788 [ACK] Seq=1 Ack=130965 Win=65536 Len=0 TSval=1739549552 TSecr=1739549551
15	0.215816002	127.0.0.1	127.0.0.1	TCP	32807	[TCP Previous segment not captured] 35788 → 8000 [PSH, ACK] Seq=163706 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549551
16	0.215841349	127.0.0.1	127.0.0.1	TCP	78	[TCP Window Update] 8000 → 35788 [ACK] Seq=1 Ack=130965 Win=196480 Len=0 TSval=1739549552 TSecr=1739549551 SLE=163706 SRE=1739549552
17	0.215897323	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [ACK] Seq=196447 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
18	0.215905268	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [PSH, ACK] Seq=229188 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
19	0.215917811	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [ACK] Seq=261929 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
20	0.215926976	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [PSH, ACK] Seq=294670 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
21	0.215942156	127.0.0.1	127.0.0.1	TCP	78	[TCP Window Update] 8000 → 35788 [ACK] Seq=1 Ack=130965 Win=327424 Len=0 TSval=1739549552 TSecr=1739549551 SLE=163706 SRE=1739549552
22	0.215968055	127.0.0.1	127.0.0.1	TCP	32807	[TCP Out-Of-Order] 35788 → 8000 [ACK] Seq=130965 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
23	0.215977462	127.0.0.1	127.0.0.1	TCP	78	[TCP Window Update] 8000 → 35788 [ACK] Seq=1 Ack=130965 Win=589440 Len=0 TSval=1739549552 TSecr=1739549551 SLE=163706 SRE=1739549552
24	0.216008009	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [ACK] Seq=327411 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
25	0.216014050	127.0.0.1	127.0.0.1	TCP	32807	35788 → 8000 [PSH, ACK] Seq=360152 Ack=1 Win=65536 Len=32741 TSval=1739549552 TSecr=1739549552
26	0.216021745	127.0.0.1	127.0.0.1	TCP	78	[TCP Window Update] 8000 → 35788 [ACK] Seq=1 Ack=130965 Win=726384 Len=0 TSval=1739549552 TSecr=1739549551 SLE=163706 SRE=1739549552

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 35788, Dst Port: 8000, Seq: 0, Len: 0

20% Packet Loss:

In the Sender.py we sent the file 15 times with 20% packet loss. The output is on the Reciever.py, with two Time lists, each list contains the times it took for the corresponding half of the file. We also calculated the average of each part's time:

```
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject$ python3 Reciever.py
Received first half in 4.874563932418823 seconds
Received second half in 0.27122020721435547 seconds
Received first half in 0.07108831405639648 seconds
Received second half in 0.029778003692620953 seconds
Received first half in 0.25115156173706055 seconds
Received second half in 0.05012518501281738 seconds
Received first half in 1.056328535079956 seconds
Received second half in 0.4343841075897217 seconds
Received first half in 0.44153285026550293 seconds
Received second half in 1.4342188835144043 seconds
Received first half in 1.2929260730743409 seconds
Received second half in 2.7633707523345947 seconds
Received first half in 8.538524627685547 seconds
Received second half in 48.353251218795776 seconds
Received first half in 0.005272388458251953 seconds
Received second half in 5.5080366134643555 seconds
Received first half in 1.3732750415802062 seconds
Received second half in 0.483461856842041 seconds
Received first half in 3.249495029449463 seconds
Received second half in 0.01530313491821289 seconds
Received first half in 0.0054776608548583984 seconds
Received second half in 0.8907284736633301 seconds
Received first half in 4.174912691116333 seconds
Received second half in 0.25340819358825684 seconds
Received first half in 1.044593095779419 seconds
Received second half in 0.7915122509002686 seconds
Received first half in 4.5270915031433105 seconds
Received second half in 33.7935106754303 seconds
Received first half in 0.2290797233581543 seconds
Received second half in 0.644705723999023 seconds
[4.874563932418823, 0.07108831405639648, 0.25115156173706055, 1.056328535079956, 0.44153285026550293, 1.2929260730743409, 8.538524627685547, 0.005272388458251953, 1.3732750415802062, 3.249495029449463, 0.0054776608548583984, 4.174912691116333, 1.044593095779419, 4.5270915031433105, 0.2290797233581543]
Avg time for part 1 is: 2.0750675350603841
[0.27122020721435547, 0.029778003692620953, 0.05012518501281738, 0.4343841075897217, 1.4342188835144043, 2.7633707523345947, 48.353251218795776, 5.5080366134643555, 0.483461856842041, 0.01530313491821289, 0.8907284736633301, 0.25340819358825684, 0.7915122509002686, 33.7935106754303, 0.644705723999023]
Avg time for part 2 is: 6.381734355290731
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject$

Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): n
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject$
```

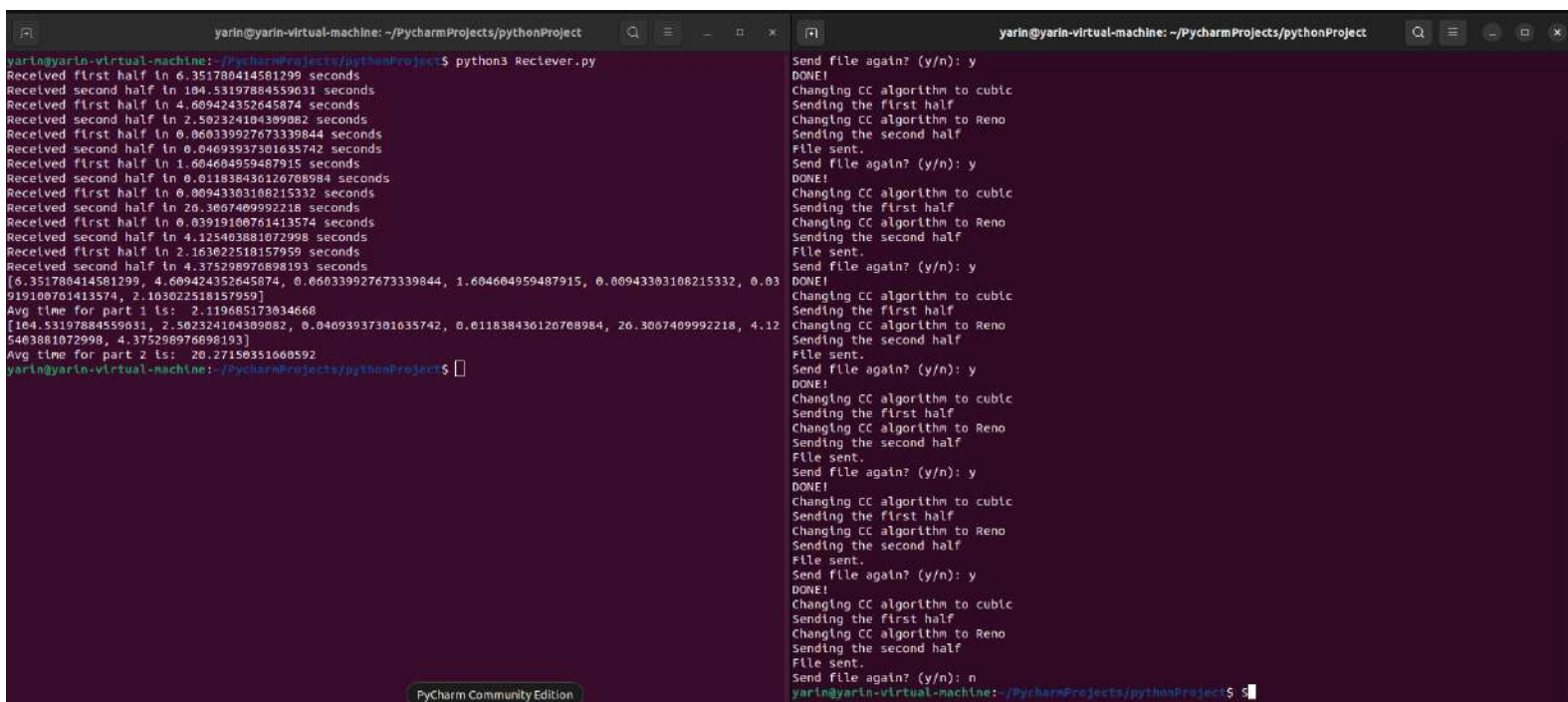
20_packet_loss.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
0.	1.0.000000000	127.0.0.1	127.0.0.1	TCP	74	41714 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1721529002 TSecr=0 WS=128
1.	2.1.029067892	127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 41714 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1721530031 TSecr=0 WS=128
2.	3.1.029098051	127.0.0.1	127.0.0.1	TCP	74	8000 → 41714 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1721530031 TSecr=1721529002 WS=128
3.	4.1.029143302	127.0.0.1	127.0.0.1	TCP	66	41714 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1721530031 TSecr=1721530031
4.	5.1.030255899	127.0.0.1	127.0.0.1	TCP	32807	41714 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=32741 TSval=1721530032 TSecr=1721530031
5.	6.1.030319918	127.0.0.1	127.0.0.1	TCP	66	8000 → 41714 [ACK] Seq=1 Ack=32742 Win=48640 Len=0 TSval=1721530032 TSecr=1721530032
6.	7.1.030343131	127.0.0.1	127.0.0.1	TCP	32807	41714 → 8000 [PSH, ACK] Seq=32742 Ack=1 Win=65536 Len=32741 TSval=1721530032 TSecr=1721530031
7.	8.1.030462132	127.0.0.1	127.0.0.1	TCP	66	8000 → 41714 [ACK] Seq=1 Ack=65483 Win=65536 Len=0 TSval=1721530032 TSecr=1721530032
8.	9.1.030498780	127.0.0.1	127.0.0.1	TCP	32807	[TCP Previous segment not captured] 41714 → 8000 [PSH, ACK] Seq=98224 Ack=1 Win=65536 Len=32741 TSval=1721530032 TSecr=1721530031
9.	10.1.030504011	127.0.0.1	127.0.0.1	TCP	78	[TCP Dup ACK #1] 8000 → 41714 [ACK] Seq=1 Ack=65483 Win=65536 Len=0 TSval=1721530033 TSecr=1721530032 SLE=98224 SRE=130905
10.	11.1.045516308	127.0.0.1	127.0.0.1	TCP	32807	[TCP Retransmission] 41714 → 8000 [ACK] Seq=65483 Ack=1 Win=65536 Len=32741 TSval=1721530047 TSecr=1721530033
11.	12.4.060940585	127.0.0.1	127.0.0.1	TCP	74	41720 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1721533071 TSecr=0 WS=128
12.	13.4.060976652	127.0.0.1	127.0.0.1	TCP	74	8000 → 41720 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1721533071 TSecr=1721530071 WS=128
13.	14.4.133998359	127.0.0.1	127.0.0.1	TCP	32807	[TCP Retransmission] 41714 → 8000 [ACK] Seq=65483 Ack=1 Win=65536 Len=32741 TSval=1721533135 TSecr=1721530033
14.	15.4.133121131	127.0.0.1	127.0.0.1	TCP	78	8000 → 41714 [ACK] Seq=1 Ack=130965 Win=65536 Len=0 TSval=1721533135 TSecr=1721530047 SLE=65483 SRE=98224
15.	16.4.133177696	127.0.0.1	127.0.0.1	TCP	32807	41714 → 8000 [ACK] Seq=130965 Ack=1 Win=65536 Len=32741 TSval=1721533135 TSecr=1721533135
16.	17.4.133330430	127.0.0.1	127.0.0.1	TCP	32807	41714 → 8000 [PSH, ACK] Seq=163706 Ack=1 Win=65536 Len=32741 TSval=1721533135 TSecr=1721533135
17.	18.4.133700933	127.0.0.1	127.0.0.1	TCP	66	8000 → 41714 [ACK] Seq=1 Ack=196447 Win=65536 Len=0 TSval=1721533136 TSecr=1721533135
18.	19.4.133809374	127.0.0.1	127.0.0.1	TCP	32807	41714 → 8000 [ACK] Seq=196447 Ack=1 Win=65536 Len=32741 TSval=1721533136 TSecr=1721533136
19.	20.4.176835475	127.0.0.1	127.0.0.1	TCP	66	8000 → 41714 [ACK] Seq=1 Ack=229188 Win=65536 Len=0 TSval=1721533179 TSecr=1721533136
20.	21.4.176839941	127.0.0.1	127.0.0.1	TCP	32807	[TCP Previous segment not captured] 41714 → 8000 [ACK] Seq=261929 Ack=1 Win=65536 Len=32741 TSval=1721533179 TSecr=1721533179
21.	22.5.002934688	127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] 8000 → 41720 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1721534095 TSecr=17215330
22.	23.5.061193855	127.0.0.1	127.0.0.1	TCP	32807	[TCP Retransmission] 41714 → 8000 [ACK] Seq=261929 Ack=1 Win=65536 Len=32741 TSval=1721534095 TSecr=1721533179
23.	24.5.061214333	127.0.0.1	127.0.0.1	TCP	80	[TCP Dup ACK 20#1] 8000 → 41714 [ACK] Seq=1 Ack=229188 Win=65536 Len=0 TSval=1721534095 TSecr=1721533136 SLE=261929 SRE=294070
24.	25.5.061236045	127.0.0.1	127.0.0.1	TCP	32807	[TCP Retransmission] 41714 → 8000 [PSH, ACK] Seq=229188 Ack=1 Win=65536 Len=32741 TSval=1721534095 TSecr=1721534095
25.	26.5.061254237	127.0.0.1	127.0.0.1	TCP	66	8000 → 41714 [ACK] Seq=1 Ack=294070 Win=128 Len=0 TSval=1721534095 TSecr=1721534095

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo, 0 Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 41714, Dst Port: 8000, Seq: 0, Len: 0

25% Packet Loss:

In the Sender.py we sent the file 7 times with 25% packet loss, in this run we chose to send it 7 times instead of 15 as the time it took was very long. The output is on the Reciever.py, with two Time lists, each list contains the times it took for the corresponding half of the file. We also calculated the average of each part's time:



```
yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$ python3 Reciever.py
Received first half in 6.351780414581299 seconds
Received second half in 104.53197884559631 seconds
Received first half in 4.609424352645874 seconds
Received second half in 2.502324104399082 seconds
Received first half in 0.060339927673339844 seconds
Received second half in 0.04093937301635742 seconds
Received first half in 1.604604959487915 seconds
Received second half in 0.011838436126708984 seconds
Received first half in 0.00943303108215332 seconds
Received second half in 26.3067409992218 seconds
Received first half in 0.03919100761413574 seconds
Received second half in 4.125403881072998 seconds
Received first half in 2.163022518157959 seconds
Received second half in 4.375298970898193 seconds
[6.351780414581299, 4.609424352645874, 0.060339927673339844, 1.604604959487915, 0.00943303108215332, 0.03919100761413574, 2.163022518157959]
Avg time for part 1 is: 2.119685173034668
[104.53197884559631, 2.502324104399082, 0.04093937301635742, 0.011838436126708984, 26.3067409992218, 4.125403881072998, 4.375298970898193]
Avg time for part 2 is: 20.27150351000592
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$

yarin@yarin-virtual-machine: ~/PycharmProjects/pythonProject
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): y
DONE!
Changing CC algorithm to cubic
Sending the first half
Changing CC algorithm to Reno
Sending the second half
File sent.
Send file again? (y/n): n
yarin@yarin-virtual-machine:~/PycharmProjects/pythonProject$
```

Conclusion

Packet loss - transferring time

In the last part of the exercise we've tested our code on 5 different packet loss percentages: 0%, 10%, 15%, 20% and 25%.

Note that the first half of the text file is sent faster than the second part in all cases, another thing to notice is that the first half is sent using the Cubic algorithm and the second is sent using Reno. This makes sense as Cubic uses the function x^3 and Reno x after congestion, let us look at the difference between the two functions:

x	1	2	3	4	5	6	7
$f(x) = x^3$	1	8	27	64	125	126	343
$f(x) = x$	1	2	3	4	5	6	7

Note how much faster the cubic function grows rather than linear, this helps us make sense out of the results received above. The Cubic algorithm is known to be better for handling packet loss as it recovers faster than Reno as it uses a faster growing function.

Another important thing to notice is how the difference between the two grows as the packet loss percentage increases, in the 0% packet loss we have a tiny difference of 0.015 seconds, as the number of packets lost increased to 20% so did the time difference which became more than 4 seconds.