TCP – Congestion Control Algorithms Network programming in C

Authors:

Maor Saadon 318532421

Duvi Amiram 305677494

Contents

1 System Characterization

1.1 System Overview

1.1.1 About the System

'TCP – Congestion Control Algorithms Network programming' - like the name of the assignment our program reflects the communication between sender and receiver according to two congestion control algorithms (cubic, reno) while sending the file.

The Sender will send a message in this message will be a file, with at least 1MB size of a file, and the Receiver will receive it and measure the time it took for his program to receive the file. The receiver doesn't really care about saving the file itself (or its content). He just cares about the Data-Frame that he gets.

The file will be sent in two parts first half and second half, each half will be sent according to one of the CC algorithms we learned in the lectures.
The first half will be sent according to CUBIC algorithms and second half will be sent according to RENO algorithm. By this, and because the size of the two half is the same, we can see based on the result which algorithm is better for this present purpose.

In addition, we will use packet lost tools on Linux which is known as TC. We can see the communication and the packet lost by using Wireshark.

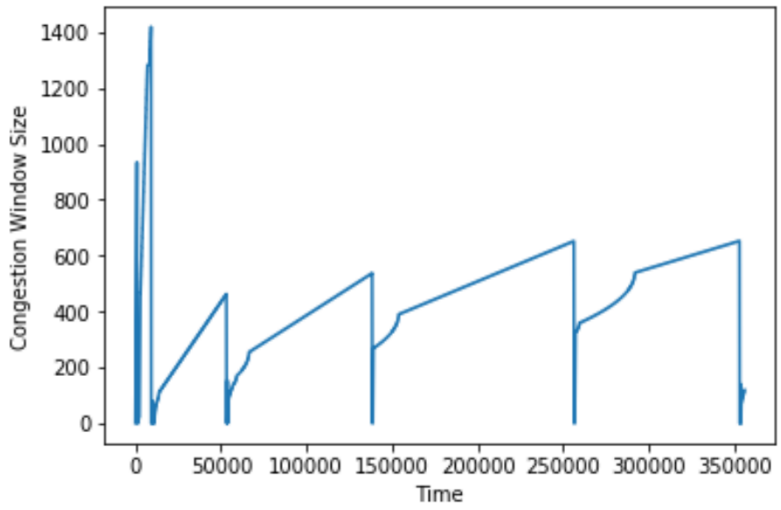
1.1.2 CC – Algorithm

At first let's explain about congestion control, congestion on the internet happens when the computers send more packets to a particular link than that link can handle. If a link is receiving more packets than can fit on the wire, it will begin queueing up those packets, and if the link's queues fill up, the link will begin dropping those packets.
Therefore we have the CC - Algorithm. In this program we use the cubic and reno CC – Algorithm.

RENO:

The traditional approaches to congestion control in TCP, named Tahoe and Reno, operate by increasing the congestion window size, which you can think of as the "rate" at which senders send packets, exponentially until some threshold is reached. After that threshold is reached, they begin increasing linearly.

Once they experience drops, they will aggressively drop the window size, and then enter "slow start" mode again, where they increase exponentially until they hit the threshold. Once the threshold is hit, they will increase linearly again.



The approaches Tahoe and Reno were developed in the late 80s and 90s, and worked then, but the internet has changed a lot since. Notably, there now exist

longer, and higher bandwidth networks (long, fat networks). To fully take advantage of these networks, TCP senders need to send with much higher congestion windows.

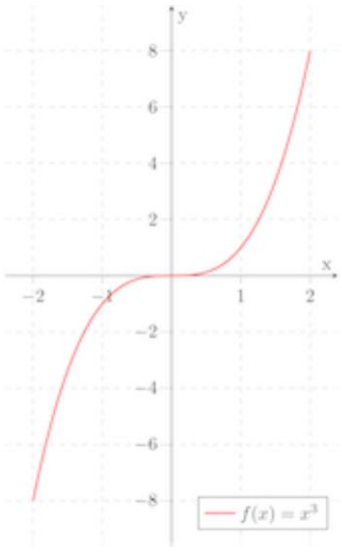
Since Tahoe & Reno grow linearly after crossing the slow start threshold, if a lot more bandwidths become available on a link, it'll take a long time for these algorithms to "discover" that available bandwidth.

CUBIC:

It turns out that to achieve better performance on "long, fat networks", these congestion control algorithms need to do a better job of "exploring" for more bandwidth.

However, we don't want to overload a network, or aggressively steal away available bandwidth from other senders on the same network.

It turns out that there's a mathematical function for growing the congestion window that satisfies both constraints, the cubic function:



Observe a very powerful property of a cubic function: that as x is lower, t function grows very quickly, and then slows down as it approaches a particular point (the inflection point), and then after crossing that point begins growing quickly again.

Therefore, it's took advantage of this property when designing a congestion control algorithm:

If the congestion window is grown as a cubic function of time since the last packet drop, and the inflection point is set to be the size of the congestion window at the last drop, the window has the following behavior:

1. Start growing fast
2. As the algorithm approaches the window at the last drop, the congestion window begins growing more slowly
3. If the congestion window gets to the point at which drops occurred the last time, and does not experience a drop, begin growing slowly again, but then increase more quickly

The concave portion of this, where the window begins growing quickly but then slows down, and the convex portion of this can be thought of two different phases. During the concave phase, the congestion window is catching up to where a packet was lost last time and slows down its growth to be fair to traditional TCP senders. If the algorithm gets there without experiencing drops, it can move into an “exploratory” phase, in which it grows quickly to discover newly available bandwidth.

1.1.3 How to Install and Run the Program

To test the system for yourself, you would need a Linux based operating system.

Instructions:

1. Download the following files:
    a. Sender.c
    b. Receiver.c
    c. Makefile
    d. 1mb.txt
2. Put all of the above files in a single directory.
3. Open said directory in your Linux terminal.
4. Run the following commands:
    a. sudo apt install build-essential
    b. Make
    c. ./Receiver
5. Open the same directory in a second Linux terminal.
6. Run the following command:
    a. ./Sender
7. Once the sending is completed, you'll be asked if you'd like to resend the file. Type "Y" for 'yes' or "N" for 'no' and press Enter.
8. To terminate the program type "N" when a sending session is completed.

1.2 System Functionality

1.2.1 Code Description

IMPORTANT!

We assume that the input is correct and that the user will not repeat sending the file more than 1000 times.

Order of operations: sender's side.

File handling:

1. Open the file in order to perform certain functions on it.

```
// (1)
//open a file
FILE *f = fopen(FILE_TO_SEND, "r");
//checking
if (f == NULL) {
    fprintf(stderr, "Error opening file");
    return 1;
}
```

2. calculate and save file size (used for sending the files in two parts of equal size).

```
// (2)
//calculate the file size
fseek(f, 0, SEEK_END);
int fileSize = (int) ftell(f);
fseek(f, 0, SEEK_SET);
```

3. store file contents in char array - preparing for sending

```
// (3)
//put the file in char array
char filedata[fileSize];
fread(filedata, sizeof(char), fileSize, f);
```

4. close the file

```
// (4)
//close the file
fclose(f);
```

Socket handling:

5. Open a socket from which to send the file.

```
// (5)
//open a socket
int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
//checking
if (sock == -1) {
    printf("Could not create socket : %d", errno);
    return -1;
} else printf("New socket opened\n");

/*
"sockaddr_in" is the "derived" from sockaddr structure
used for IPv4 communication. For IPv6, use sockaddr_in6
*/
struct sockaddr_in serverAddress;

/*
fills the first 'sizeof(serverAddress)' bytes of the memory
area pointed to by &serverAddress with the constant 0.
*/
memset(&serverAddress, 0, sizeof(serverAddress));

//address family, AF_INET(while using TCP) undighned
serverAddress.sin_family = AF_INET;
//change the port number from little endian => network endian(big endian):
serverAddress.sin_port = htons(SERVER_PORT);
//convert IPv4 and IPv6 addresses from text to binary form
int rval = inet_pton(AF_INET, (const char *) SERVER_IP_ADDRESS, &serverAddress.sin_addr);
//checking
if (rval <= 0) {
    printf("inet_pton() failed");
    return -1;
}
```

6. connect said socket to server side using the known server address. (in this case: local host).

```
// (6)
// Make a connection to the server with socket SendingSocket.
int connectResult = connect(sock, (struct sockaddr *) &serverAddress, sizeof(serverAddress));
//checking
if (connectResult == -1) {
    printf("connect() failed with error code : %d", errno);
    close(sock);
    return -1;
} else printf("connected to receiver\n");
```

Preparing for sending:

7. initialize agreed upon sign ("signal") which would indicate that the receiving side (server side) acquired the file size information. Then, send the file size using send() method (see explanation below).

```
// (7)
int signal = 0; //agreed upon sign

send(sock, &fileSize, sizeof(int), 0); //send the filesize to the receiver
```

8. receive the agreed upon sign ("signal") from the receiver's side.

```
// (8)
recv(sock, &signal, sizeof(int), 0);//receiving the the agreed sign from "receiver" side
```

Sending both parts and authenticating:

9. set the CC algorithm to "cubic" using setsockopt() function

```
// (9)
char cc_algo[BUFFER_SIZE]; //char array for changing the algorithem
strcpy(cc_algo, "cubic"); //copy the string "cubic" into cc_algo
socklen_t len = strlen(cc_algo); //saving the size of the str in the cc_algo in socklen_t variable
//checking & defult the cubic algorithem
if (setsockopt(sock, IPPROTO_TCP, TCP_CONGESTION, cc_algo, len) == -1) {
    perror("setsockopt");
    return -1;
}
```

10. send part A of the file (determined by setting the length of data to send to (filesize/2)).

```
// (10)
//send the first half of the file
bytesSent = send(sock, filedata, (fileSize / 2), 0);
//checking
if (bytesSent == -1) printf("send() failed with error code : %d", errno);
else if (bytesSent == 0) printf("peer has closed the TCP connection prior to send().\n");
else if (bytesSent < (fileSize / 2))
    printf("sent only %ld bytes from the required %d.\n", bytesSent, (fileSize / 2));
else printf("Sent %ld bytes\n", bytesSent);
```

11. initialize certain value (in this case: xor(2421, 7494) – last four digits of our ID) then, receive value from receiver and compare for authentication (determining that we sent the first part of the file to the desired receiver).

```
// (11)
int sender_xor = 2421 ^ 7494; //the sender xor for the authentication
int receiver_xor;//the xor that the sender get from the receiver

//receiv the xor from the receiver
recv(sock, &receiver_xor, sizeof(int), 0);
//checking
if (receiver_xor == sender_xor) printf("Part A was successfully sent\n");
else printf("client: something went wrong\n");
```

12.set the CC algorithm to "reno" using setsockopt() function and repeat step

```
// (12)
strcpy(cc_algo, "reno"); //copy the string "reno" into cc_algo
len = strlen(cc_algo); //saving the size of the str in the cc_algo in socklen_t variable
//checking & defult the cubic algorithem
if (setsockopt(sock, IPPROTO_TCP, TCP_CONGESTION, cc_algo, len) != 0) {
    perror("setsockopt");
    return -1;
}

printf("Sending part B using reno CC algorithm\n");

//send the second half of the file
bytesSent = send(sock, filedata + (fileSize / 2) - 1, (fileSize / 2), 0);
//checking
if (bytesSent == -1) printf("send() failed with error code : %d", errno);
else if (bytesSent == 0) printf("peer has closed the TCP connection prior to send().\n");
else if (bytesSent < (fileSize / 2))
    printf("sent only %ld bytes from the required %d.\n", bytesSent, (fileSize / 2));
else printf("Sent %ld bytes\n", bytesSent);

printf("Part B was successfully sent\n");
```

13.ask user if he wants to repeat the sending process or end the connection.

```
// (13)
char choose; //the user decision
printf("Send again? Y = yes, N = no\n");
scanf(" %c", &choose);
```

14.repeat process (while loop) as long as the user answers "yes".

Order of operations: receiver's side.

Handle socket

1. Open a server socket and initialize using sockarddr_in struct (see below functionality of other functions in use). We also set the enable reuse option using setsockopt() function. This allows reusing the same port for receving multiple sending attempts.

```
// (1)
// Open the listening socket
int listeningSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (listeningSocket == -1) {
    printf("Could not create listening socket : %d", errno);
    return 1;
}

// for reusing of port
int enableReuse = 1;
int ret = setsockopt(listeningSocket, SOL_SOCKET, SO_REUSEADDR, &enableReuse, sizeof(int));
if (ret < 0) {
    printf("setsockopt() failed with error code : %d", errno);
    return 1;
}

// "sockaddr_in" is the "derived" from sockaddr structure
// used for IPv4 communication. For IPv6, use sockaddr_in6
struct sockaddr_in serverAddress;

/*
fills the first 'sizeof(serverAddress)' bytes of the memory
area pointed to by &serverAddress with the constant 0.
*/
memset(&serverAddress, 0, sizeof(serverAddress));

// address family, AF_INET(while using TCP) undighned
serverAddress.sin_family = AF_INET;
// any IP at this port (Address to accept any incoming messages)
serverAddress.sin_addr.s_addr = INADDR_ANY;
// change the port number from little endian => network endian(big endian):
serverAddress.sin_port = htons(SERVER_PORT);  // network order (makes byte order consistent)
```

2. Bind the socket

```
    // (2)
    // Bind the socket to the port with any IP at this port
    int bindResult = bind(listeningSocket, (struct sockaddr *) &serverAddress,
sizeof(serverAddress));
    // checking
    if (bindResult == -1) {
        printf("Bind failed with error code : %d", errno);
        close(listeningSocket);
        return -1;
    } else printf("executed Bind() successfully\n");
```

3. Make socket listen to incoming connections.

```
    // (3)
    // Make the socket listen.
    // 500 is a Maximum size of queue connection requests
    // number of concurrent connections = 3
    int listenResult = listen(listeningSocket, 3);
    //checking
    if (listenResult == -1) {
        printf("listen() failed with error code : %d", errno);
        close(listeningSocket);
        return -1;
    } else printf("Waiting for incoming TCP-connections...\n");
```

4. Accept incoming connections using accept() method, on a new socket (file descriptor).

```
    // (4)
    // accept a connection on a socket
    int clientSocket = accept(listeningSocket, (struct sockaddr *) &clientAddress,
&len_clientAddress);
    // checking
    if (clientSocket == -1) {
        printf("listen failed with error code : %d", errno);
        close(listeningSocket);
        return -1;
    } else printf("A new client connection accepted\n");
```

5. Receive file size info from sender and return agreed upon sign which indicates that the file size is known and the file can be sent.

```
    // (5)
    int fileSize; // the file size that we receiv from the sender
    int signal = 0; // agreed sign

    // receive the file size from the sender
    recv(clientSocket, &fileSize, sizeof(int), 0);
    //send the agreed sign to the sender
    send(clientSocket, &signal, sizeof(int), 0);
```

6. Initialize arrays to store the time data of each attempt at receiving the file, and a counter variable to count the number of attempts.

```
// (6)
long timeOfPartA[1000]; // long array to save the run time of sending partA
bzero(timeOfPartA, 1000); // make a zero array
long timeOfPartB[1000]; // long array to save the run time of sending partB
bzero(timeOfPartB, 1000); // make a zero array
long counter = 0; // present the number of sending the whole file


int running = 1; // stop condition
while (running) {
```

Receiving part A:

7. Set the CC algorithm in the listening socket to "cubic" using setsockopt() method (see explanation below) for the retrieval of the first part.

```
// (7)
char cc_algo[BUFFER_SIZE]; // char array for changing the algorithem
printf("Changing to cubic...\n");
strcpy(cc_algo, "cubic"); // copy the string "cubic" into cc_algo
socklen_t len = strlen(cc_algo); // saving the size of the str in the cc_algo in socklen_t variable
// checking & defult the cubic algorithem

if (setsockopt(listeningSocket, IPPROTO_TCP, TCP_CONGESTION, cc_algo, len) == -1) {
    perror("setsockopt");
    return -1;
}
```

8. Initialize buffer with the correct size for receiving the desires part of the file, and initialize a variable to keep track of the number of bytes that were received.

```
// (8)
char buffer[fileSize / 2]; // char array for receiving the half of the file
int totalbytes = 0; // present the bytes that have been received

printf("Waiting for part A...\n");
```

9. Calculate time at the beginning of receiving process.

```
// (9)
struct timeval current_time;//struct for saving current time
gettimeofday(&current_time, NULL);//saving the current time
long before_partA_sec = current_time.tv_sec;//time in second
long before_partA_mic = current_time.tv_usec;//time in microsecond
long total_time_before_partA = before_partA_sec * 1000000 + before_partA_mic;//total time befor
```

10. Receive the bytes sent from "sender" using recv() method (see explanation below. If the connection was lost for some reason than the entire process will be terminated (determined by checking the "running" variable).

```
// (10)
while (totalbytes < (fileSize / 2)) {
    //receive the first part of the file
    int bytesgot = recv(clientSocket, buffer + totalbytes, sizeof(char), 0);
    totalbytes += bytesgot;
    //checking
    if (bytesgot == 0) {
        printf("Connection with sender closed, exiting...\n");
        running = 0;
        break;
    }
}

if (running == 0) break; // quit the program if somthing wrong
```

11. Calculate time at the end of receiving process.

```
// (11)
gettimeofday(&current_time, NULL); // saving the current time
long after_partA_sec = current_time.tv_sec; // time in second
long after_partA_mic = current_time.tv_usec; // time in microsecond
long total_time_after_partA = after_partA_sec * 1000000 + after_partA_mic; // total time after
long total_time_partA = total_time_after_partA - total_time_before_partA; // tatal time that took part A
timeOfPartA[counter] = total_time_partA; // saving the time in the array

printf("Got part A\n");

printf("Sending authntication check\n");
```

12.Send agreed upon sign for authentication xor(2421 ^ 7494).

```
// (12)
int receiver_xor = 2421 ^ 7494; //the receiver xor
send(clientSocket, &receiver_xor, sizeof(int),
     0);//the receiver send his xor to the sender for the authentication

printf("Authontication sent\n");
```

Receiving part B:

13.Changing CC algorithm to Reno in the listening socket, and repeating steps 8 – 11 for the retrieval of part B.

```
// (13)
printf("Changeing to reno..\n");

strcpy(cc_algo, "reno");//copy the string "reno" into cc_algo
len = strlen(cc_algo);//saving the size of the str in the cc_algo in socklen_t variable
//checking & defult the cubic algorithem
if (setsockopt(listeningSocket, IPPROTO_TCP, TCP_CONGESTION, cc_algo, len) == -1) {
    perror("setsockopt");
    return -1;
}
```

14.Close the connection.

```
printf("Exit\n");
close(clientSocket);//close the clientSocket
close(listeningSocket);//listeningSocket
```

15. Calculate the total time it took to receive each part.

```
// (15)
long total_time_of_A = 0;//present the tatal time for receiving part A(for all times)
long total_time_of_B = 0;//present the tatal time for receiving part A(for all times)

for (int i = 0; i < counter; i++) {
    printf("Run time of part A, number %d : (%ld second,%ld microseconds)\n", i + 1, (timeOfPartA[i] / 1000000),
           (timeOfPartA[i] % 1000000));
    printf("Run time of part B, number %d : (%ld second,%ld microseconds)\n", i + 1, (timeOfPartB[i] / 1000000),
           (timeOfPartB[i] % 1000000));

    //sum the time of each receiving (part A/B)
    total_time_of_A += timeOfPartA[i];
    total_time_of_B += timeOfPartB[i];
}
```

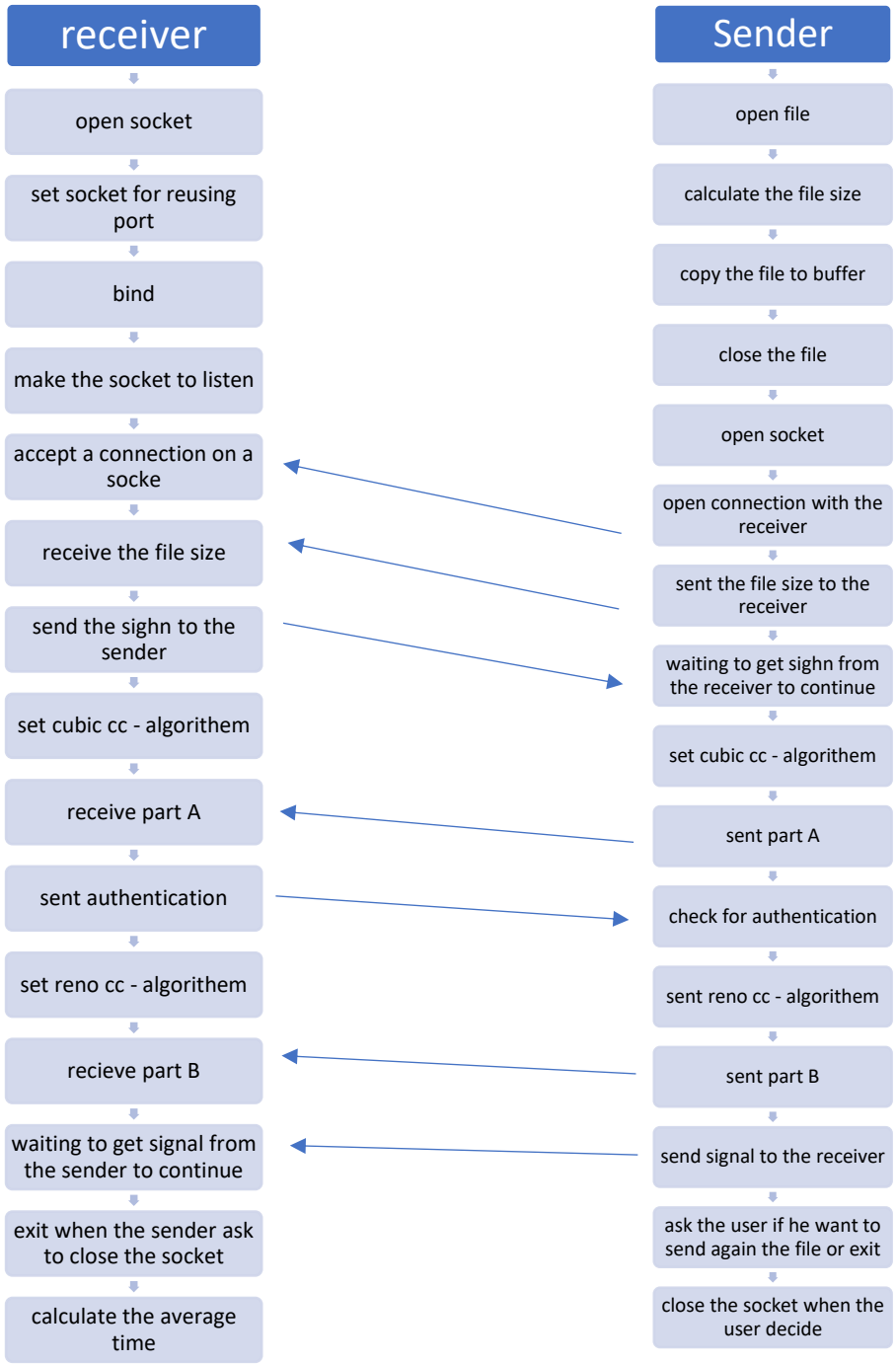
16. Calculate the average receive time for each part using the "counter" variable which indicates the number of times the file was sent.

```
   // (16)
   //calculate the average
   long average_time_of_A = (total_time_of_A / counter);
   long average_time_of_B = (total_time_of_B / counter);

   printf("The average time of part A is: (%ld second,%ld microseconds)\n", (average_time_of_A / 1000000),
          (average_time_of_A % 1000000));
   printf("The average time of part B is: (%ld second,%ld microseconds)\n", (average_time_of_B / 1000000),
          (average_time_of_B % 1000000));

   return 0;
}
```

1.2.2 Flowchart



1.2.3 Output

Sending the file 5 times.

```
executed Bind() successfully
Waiting for incoming TCP-connections...
A new client connection accepted
Changing to cubic...
Waiting for part A...
Got part A
Sending authntication check
Authontication sent
Changeing to reno..
Waiting for part B...
Got part B
Changing to cubic...
Waiting for part A...
Got part A
Sending authntication check
Authontication sent
Changeing to reno..
Waiting for part B...
Got part B
Changing to cubic...
Waiting for part A...
Got part A
Sending authntication check
Authontication sent
Changeing to reno..
Waiting for part B...
Got part B
Changing to cubic...
Waiting for part A...
Got part A
Sending authntication check
Authontication sent
Changeing to reno..
Waiting for part B...
Got part B
Changing to cubic...
Waiting for part A...
Got part A
Sending authntication check
Authontication sent
Changeing to reno..
Waiting for part B...
Got part B
Changing to cubic...
Waiting for part A...
Connection with sender closed, exiting...
Exit
```

```
New socket opened
connected to server
Sending part A using cubic CC algorithm
Sent 528258 bytes
Part A was successfully sent
Sending part B using reno CC algorithm
Sent 528258 bytes
Part B was successfully sent
Send again? Y = yes, N = no
Y
Sending part A using cubic CC algorithm
Sent 528258 bytes
Part A was successfully sent
Sending part B using reno CC algorithm
Sent 528258 bytes
Part B was successfully sent
Send again? Y = yes, N = no
Y
Sending part A using cubic CC algorithm
Sent 528258 bytes
Part A was successfully sent
Sending part B using reno CC algorithm
Sent 528258 bytes
Part B was successfully sent
Send again? Y = yes, N = no
Y
Sending part A using cubic CC algorithm
Sent 528258 bytes
Part A was successfully sent
Sending part B using reno CC algorithm
Sent 528258 bytes
Part B was successfully sent
Send again? Y = yes, N = no
Y
Sending part A using cubic CC algorithm
Sent 528258 bytes
Part A was successfully sent
Sending part B using reno CC algorithm
Sent 528258 bytes
Part B was successfully sent
Send again? Y = yes, N = no
N
```

1.2.4 Functions

Sender function:

1.To read the file we created we used the following functions:

- stream open functions-
  FILE *fopen(const char *pathname, const char *mode);
  The fopen() function opens the file whose name is the string pointed to by pathname and associates a stream with it.
  r - Open text file for reading. The stream is positioned at the beginning of the file.
  Upon successful completion fopen(), return a FILE pointer. Otherwise, NULL is returned and errno is set to indicate the error.
- Reposition a stream-
  int fseek(FILE *stream, long offset, int whence);
  The fseek() function sets the file position indicator for the stream pointed to by stream. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by whence. If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.
  Upon successful completion fseek() return 0.
- Reposition a stream-
  long ftell(FILE *stream);
  The ftell() function obtains the current value of the file position indicator for the stream pointed to by stream.
  Upon successful completion, ftell() returns the current offset. Otherwise, -1 is returned and errno is set to indicate the error.
- Binary stream input/output-
  size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
  The function fread() reads nmemb items of data, each size bytes long, from the stream pointed to by stream, storing them at the location given by ptr.
  On success, fread() return the number of items read or written. This number equals the number of bytes transferred only when size is 1. If an error occurs, or the end of the file is reached, the return value is a short

item count (or zero). The file position indicator for the stream is advanced by the number of bytes successfully read or written.
fread() does not distinguish between end-of-file and error, and callers must use feof(3) and ferror(3) to deter-mine which occurred.

- Close a stream -
  int fclose(FILE *stream);
  The fclose() function flushes the stream pointed to by stream and closes the underlying file descriptor. The behavior of fclose() is undefined if the stream parameter is an illegal pointer, or is a descriptor already passed to a previous invocation of fclose().
  Upon successful completion, 0 is returned. Otherwise, EOF is returned and errno is set to indicate the error. In either case, any further access (including another call to fclose()) to the stream results in undefined behavior.
  The fclose() function may also fail and set errno for any of the errors specified for the routines close(2), write(2), or fflush(3).

2. To create a TCP Connection between the sender and receiver we used the following functions:

- Create an endpoint for communication-
  int socket(int domain, int type, int protocol);
  socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.
  The domain argument specifies a communication domain; this selects the protocol family which will be used for communication:
  AF_INET - IPv4 Internet protocols.
  The socket has the indicated type, which specifies the communication semantics. Currently defined types are:
  SOCK_STREAM - Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
  The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type

within a given protocol family, in which case protocol can be specified as 0. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. In our case is IPPROTO_TCP.

On success, a file descriptor for the new socket is returned. On error, -1 is returned, and errno is set appropriately.

- Fill memory with a constant byte-
  void *memset(void *s, int c, size_t n);
  The memset() function fills the first n bytes of the memory area pointed to by s with the constant byte c.
  The memset() function returns a pointer to the memory area s.
- Convert values between host and network byte order-
  uint16_t htons(uint16_t hostshort);
  The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.
- Convert IPv4 and IPv6 addresses from text to binary form -
  int inet_pton(int af, const char *src, void *dst);
  This function converts the character string src into a network address structure in the af address family, then copies the network address structure to dst. The af argument must be either AF_INET or AF_INET6. dst is written in network byte order.
  In our case the following address families are currently supported:
  AF_INET src points to a character string containing an IPv4 network address in dotted-decimal format, "ddd.ddd.ddd.ddd", where ddd is a decimal number of up to three digits in the range 0 to 255. The address is converted to a struct in_addr and copied to dst, which must be sizeof(struct in_addr) (4) bytes (32bits) long.
  inet_pton() returns 1 on success (network address was successfully converted). 0 is returned if src does not contain a character string representing a valid network address in the specified address family.
  If af does not contain a valid address family, -1 is returned and errno is set to EAFNOSUPPORT.
- Initiate a connection on a socket-
  int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. The addrlen argument specifies the size of addr. The format of the address in addr is determined by the address space of the socket sockfd.
If the socket sockfd is of type SOCK_DGRAM, then addr is the address to which datagrams are sent by default, and the only address from which datagrams are received. If the socket is of type SOCK_STREAM or SOCK_SEQPACKET, this call attempts to make a connection to the socket that is bound to the address specified by addr.
If the connection or binding succeeds, zero is returned. On error, -1 is returned, and errno is set appropriately.

3. To Send the first and the second part of the file and to check for authentication we used the following functions:

- Send a message on a socket-
  ssize_t send(int sockfd, const void *buf, size_t len, int flags);
  send() are used to transmit a message to another socket.
  The send() call may be used only when the socket is in a connected state.
  The argument sockfd is the file descriptor of the sending socket.
  The message is found in buf and has length len.
  If the message is too long to pass atomically through the underlying protocol, the error EMSGSIZE is returned, and the message is not transmitted.
  No indication of failure to deliver is implicit in a send(). Locally detected errors are indicated by a return value of -1.
  When the message does not fit into the send buffer of the socket, send() normally blocks, unless the socket has been placed in nonblocking I/O mode. In nonblocking mode, it would fail with the error EAGAIN or EWOULDBLOCK in this case.
  The flags argument is the bitwise OR of zero or more flags.
  in our case is 0 flag.
  On success, these calls return the number of bytes sent. On error, -1 is returned, and errno is set appropriately.

- Receive a message from a socket -
  ssize_t recv(int sockfd, void *buf, size_t len, int flags);
  recv() used to receive messages from a socket. They may be used to receive data on both connectionless and connection-oriented sockets. This page first describes common features of all three system calls, and then describes the differences between the calls.
  If no messages are available at the socket, the receive calls wait for a message to arrive, unless the socket is nonblocking , in which case the value -1 is returned and the external variable errno is set to EA-GAIN or EWOULDBLOCK. The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.
  These calls return the number of bytes received, or -1 if an error occurred. In the event of an error, errno is set to indicate the error.
  When a stream socket peer has performed an orderly shutdown, the return value will be 0 (the traditional "end-of-file" return).
  Datagram sockets in various domains (e.g., the UNIX and Internet domains) permit zero-length datagrams. When such a datagram is received, the return value is 0.
  The value 0 may also be returned if the requested number of bytes to receive from a stream socket was 0.

4.To change the CC Algorithm -

- Copy a string -
  char *strcpy(char *dest, const char *src);
  The strcpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.
  The strcpy() functions return a pointer to the destination string dest.

- Calculate the length of a string -
  size_t strlen(const char *s);
  The strlen() function calculates the length of the string pointed to by s, excluding the terminating null byte ('\0'). The strlen() function returns the number of bytes in the string pointed to by s.
- Set options on sockets -
  int setsockopt(int sockfd, int level, int optname, const void *optval,
  socklen_t optlen);
  setsockopt() manipulate options for the socket referred to by the file descriptor sockfd. Options may exist at multiple protocol levels.
  When manipulating socket options, the level at which the option resides, and the name of the option must be specified. To manipulate options at the sockets API level, level is specified as SOL_SOCKET. To manipulate options at any other level the proto-col number of the appropriate protocol controlling the option is supplied.
  The arguments optval and optlen are used to access option values.
  The argument should be nonzero to enable a boolean option, or zero if the option is to be disabled.
  On success, zero is returned for the standard options. On error, -1 is returned, and errno is set appropriately.

5.To Close the TCP connection we used the following functions:

- Close a file descriptor-
  close() closes a file descriptor, so that it no longer refers to any file and may be reused.

Receiver function:

1. to Create a TCP Connection between the sender and receiver we used the following functions:

- Create an endpoint for communication-
  int socket(int domain, int type, int protocol);
  (see the explanation above)
- Set options on sockets -
  int setsockopt(int sockfd, int level, int optname, const void *optval,
  socklen_t optlen);
  (see the explanation above)
- Fill memory with a constant byte-
  void *memset(void *s, int c, size_t n);
  (see the explanation above)
- Convert values between host and network byte order-
  uint16_t htons(uint16_t hostshort);
  (see the explanation above)
- Bind a name to a socket -
  When a socket is created with socket(), it exists in a name space (address family) but has no address assigned to it. bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd. addrlen specifies the size, in bytes, of the address structure pointed to by addr. Traditionally, this operation is called “assigning a name to a socket”. It is normally necessary to assign a local address using bind() before a SOCK_STREAM socket may receive connections.
  On success, zero is returned. On error, -1 is returned, and errno is set appropriately.
- Listen for connections on a socket -
  int listen(int sockfd, int backlog);
  listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept().

The sockfd argument is a file descriptor that refers to a socket of type SOCK_STREAM or SOCK_SEQPACKET.
The backlog argument defines the maximum length to which the queue of pending connections for sockfd may grow.
If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection succeeds.
On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

2.To get a connection from the sender we used the following functions:

- accept a connection on a socket
  int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
  The accept() system call is used with connection-based socket types (SOCK_STREAM, SOCK_SEQPACKET). It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket sockfd is unaffected by this call.
  The argument sockfd is a socket that has been created with socket(), bound to a local address with bind(), and is listening for connections after a listen().
  The argument addr is a pointer to a sockaddr structure. This structure is filled in with the address of the peer socket, as known to the communications layer. The exact format of the address returned addr is determined by the socket's address family. When addr is NULL, nothing is filled in; in this case, addrlen is not used, and should also be NULL.
  The addrlen argument is a value-result argument: the caller must initialize it to contain the size (in bytes) of the structure pointed to by addr; on return it will contain the actual size of the peer address.
  The returned address is truncated if the buffer provided is too small; in this case, addrlen will return a value greater than was supplied to the call.
  If no pending connections are present on the queue, and the socket is not marked as nonblocking, accept() blocks the caller until a connection is

present. If the socket is marked nonblocking and no pending connections arenpresent on the queue, accept() fails with the error EAGAIN or EWOULDBLOCK.

3. To receive the first and the second part of the file we used the following functions:

- Receive a message from a socket -
  ssize_t recv(int sockfd, void *buf, size_t len, int flags);

4. To measure the time, it took to receive the first and the second part we used the following functions:

- Get time -
  int gettimeofday(struct timeval *tv, struct timezone *tz);
  The functions gettimeofday() can set the time as well as a timezone.

5. To send back the authentication to the sender we used the following functions:

- Send a message on a socket-
  ssize_t send(int sockfd, const void *buf, size_t len, int flags);

6. To change the CC Algorithm we used the following functions:

- Copy a string -
  char *strcpy(char *dest, const char *src);
  (see the explanation above)
- Calculate the length of a string -
  size_t strlen(const char *s);
  (see the explanation above)
- Set options on sockets -
  int setsockopt(int sockfd, int level, int optname, const void *optval,
  socklen_t optlen);
  (see the explanation above)

4.To Close the TCP connection we used the following functions:

- Close a file descriptor-
  int close(int fd);
  close() closes a file descriptor, so that it no longer refers to any file and may be reused.

1.2.5 Interfaces

- stdio.h is the header file for the standard input/output library in C. It provides functions for reading from and writing to the standard input and output streams (e.g., printf and scanf).
  In this project: Used for printing process steps and output information.

- stdlib.h is the header file for the standard library in C. It provides functions for performing general-purpose tasks (e.g., malloc, free, and exit).
  In this project: Used for handling buffers etc.

- sys/socket.h is the header file for the socket interface in C. It provides functions for creating and manipulating sockets (e.g., socket, bind, listen, and accept).
  In this project: Used for handling sockets and connections.

- arpa/inet.h is the header file for the internet address manipulation library in C. It provides functions for converting internet addresses between their binary and text representations (e.g., inet_aton and inet_ntoa).
  In this project: Used for converting IP from binary to text form.

- netinet/in.h is the header file for the internet address family in C. It provides definitions for internet-related data types, such as struct sockaddr_in which is used to represent internet addresses.
  In this project: Used for initializing socket data using sockaddr_in struct.

- netinet/tcp.h is the header file for the TCP (Transmission Control Protocol) implementation in C. It provides definitions for TCP-related data types and constants, such as TCP_NODELAY which disables the Nagle algorithm for reducing the number of small packets sent over the network.

- string.h is the header file for the string library in C. It provides functions for manipulating strings (e.g., strlen, strcpy, and strcmp).

In this project: Used for setting up CC algorithms variables for the setsockopt() function and more, determining string lengths and more.

- errno.h is the header file for the global errno variable in C. errno is set by various functions to indicate an error, and the errno.h header provides definitions for the different possible values of `.
  In this project: Used for specifying the error tha may have occurred and terminated the program.

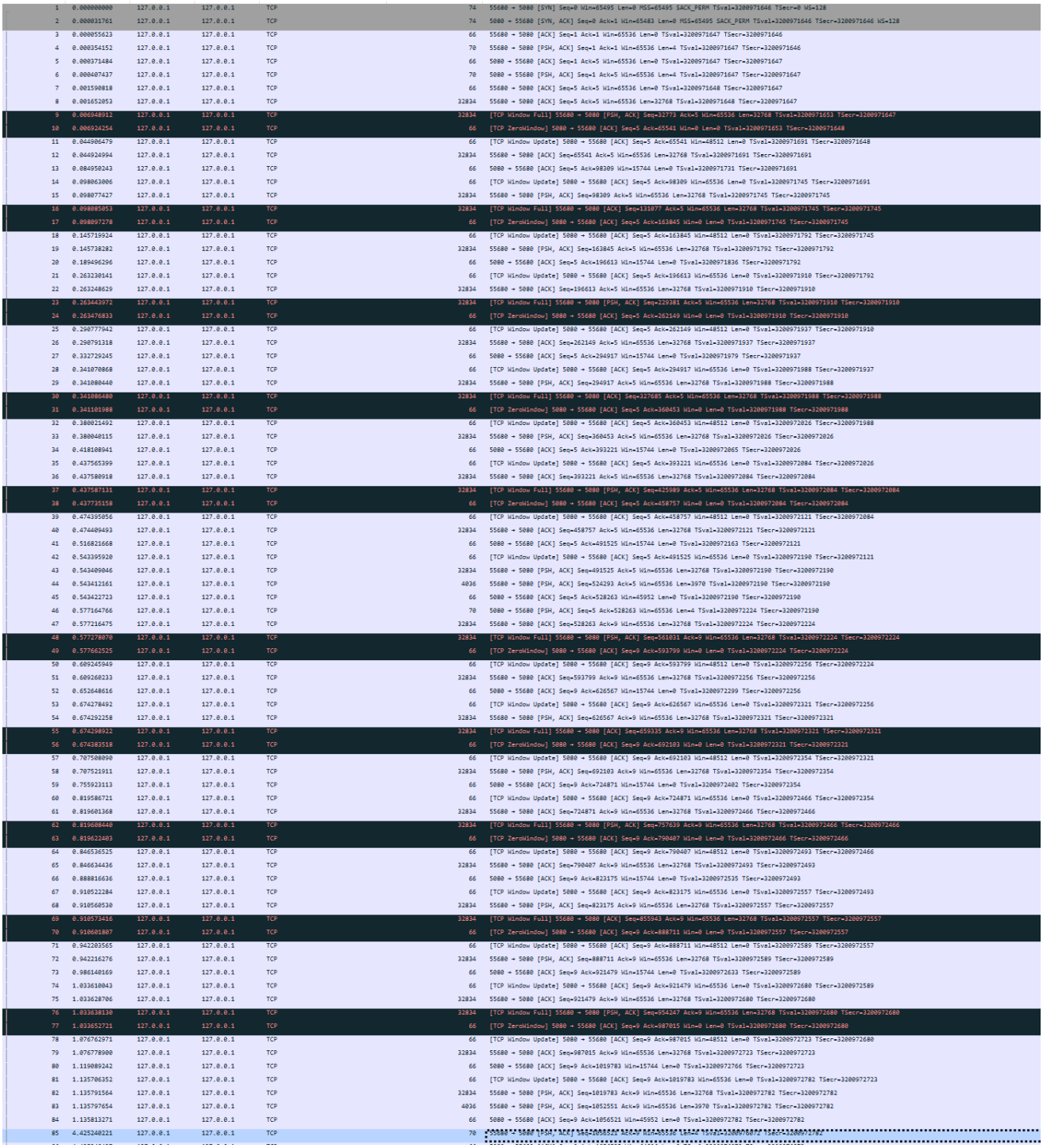
- The time.h: provides functions for manipulating time values. It includes definitions for the time_t data type, which represents a calendar time, as well as functions for converting between time_t values and strings (e.g., ctime and asctime). It also provides functions for measuring the time elapsed between two points in time (e.g., clock and difftime).
  In this project: Used for calculating sending times using gettimeofday() func.

2 Research findings

2.1 Wireshark

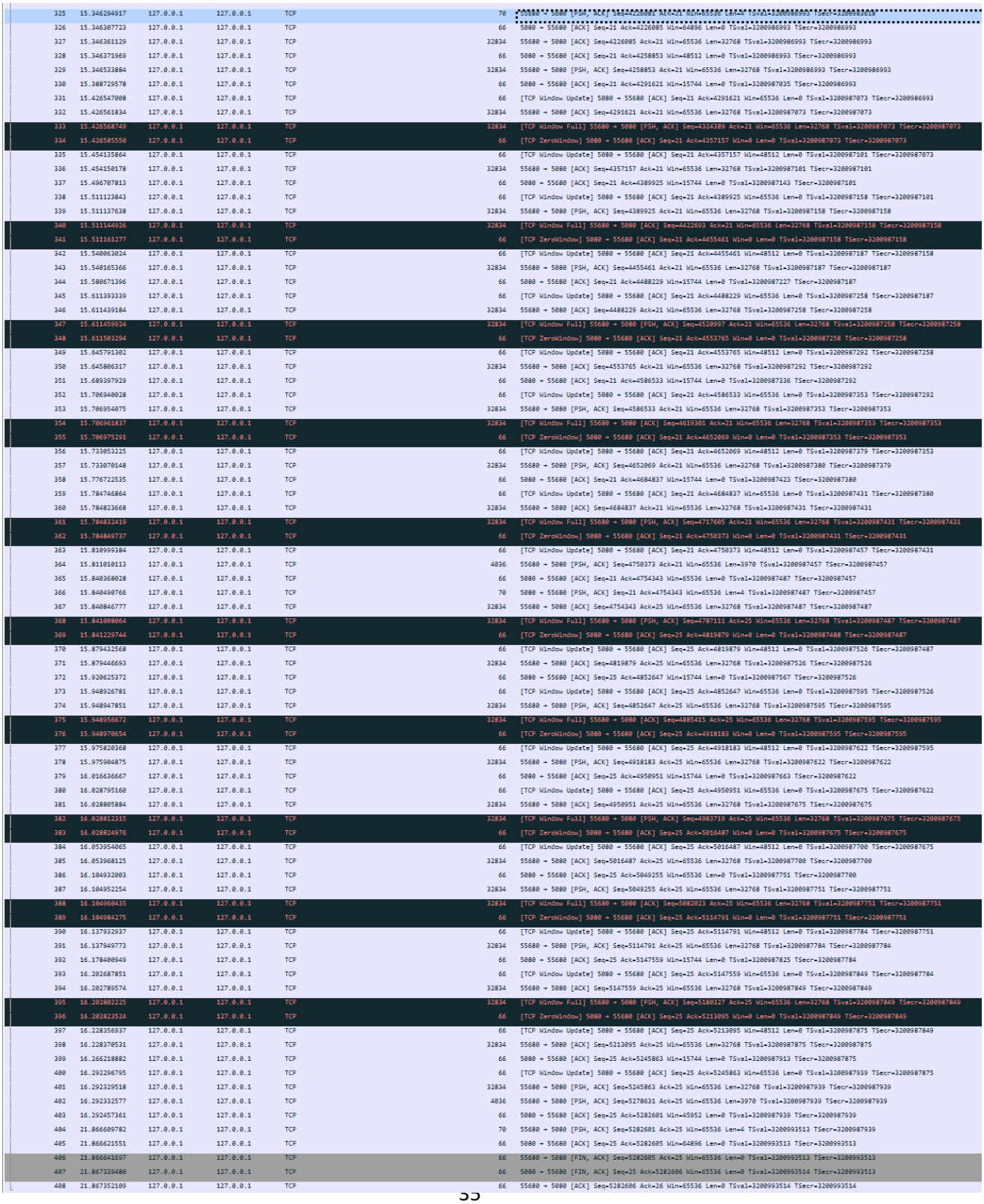
2.1.1 0% lost

Sending number 1:



1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	55680 → 5080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3200971646 TSecr=0 WS=128
2	0.000031761	127.0.0.1	127.0.0.1	TCP	74	5080 → 55680 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3200971646 TSecr=3200971646 WS=128
3	0.000055623	127.0.0.1	127.0.0.1	TCP	66	55680 → 5080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3200971647 TSecr=3200971646
4	0.000354152	127.0.0.1	127.0.0.1	TCP	70	55680 → 5080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=3200971647 TSecr=3200971646
5	0.000371484	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=3200971647 TSecr=3200971647
6	0.000407437	127.0.0.1	127.0.0.1	TCP	70	5080 → 55680 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=4 TSval=3200971647 TSecr=3200971647
7	0.001590818	127.0.0.1	127.0.0.1	TCP	66	55680 → 5080 [ACK] Seq=5 Ack=5 Win=65536 Len=0 TSval=3200971648 TSecr=3200971647
8	0.001652853	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=5 Ack=5 Win=65536 Len=32768 TSval=3200971648 TSecr=3200971647
9	0.006948912	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=32773 Ack=5 Win=65536 Len=32768 TSval=3200971653 TSecr=3200971647
10	0.006924254	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=5 Ack=65541 Win=0 Len=0 TSval=3200971653 TSecr=3200971648
11	0.044906479	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=65541 Win=48512 Len=0 TSval=3200971691 TSecr=3200971648
12	0.044924994	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=65541 Ack=5 Win=65536 Len=32768 TSval=3200971691 TSecr=3200971691
13	0.084958243	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=5 Ack=98309 Win=15744 Len=0 TSval=3200971731 TSecr=3200971691
14	0.098063006	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=98309 Win=65536 Len=0 TSval=3200971745 TSecr=3200971691
15	0.098077427	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=98309 Ack=5 Win=65536 Len=32768 TSval=3200971745 TSecr=3200971745
16	0.098085053	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [ACK] Seq=131077 Ack=5 Win=65536 Len=32768 TSval=3200971745 TSecr=3200971745
17	0.098097278	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=5 Ack=163845 Win=0 Len=0 TSval=3200971745 TSecr=3200971745
18	0.145719924	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=163845 Win=48512 Len=0 TSval=3200971792 TSecr=3200971745
19	0.145738282	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=163845 Ack=5 Win=65536 Len=32768 TSval=3200971792 TSecr=3200971792
20	0.189496296	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=5 Ack=196613 Win=15744 Len=0 TSval=3200971836 TSecr=3200971792
21	0.263230141	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=196613 Win=65536 Len=0 TSval=3200971910 TSecr=3200971792
22	0.263248629	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=196613 Ack=5 Win=65536 Len=32768 TSval=3200971910 TSecr=3200971910
23	0.263443972	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=229381 Ack=5 Win=65536 Len=32768 TSval=3200971910 TSecr=3200971910
24	0.263476833	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=5 Ack=262149 Win=0 Len=0 TSval=3200971910 TSecr=3200971910
25	0.290777942	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=262149 Win=48512 Len=0 TSval=3200971937 TSecr=3200971910
26	0.290791318	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=262149 Ack=5 Win=65536 Len=32768 TSval=3200971937 TSecr=3200971937
27	0.332729245	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=5 Ack=294917 Win=15744 Len=0 TSval=3200971979 TSecr=3200971937
28	0.341070868	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=294917 Win=65536 Len=0 TSval=3200971988 TSecr=3200971937
29	0.341080440	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=294917 Ack=5 Win=65536 Len=32768 TSval=3200971988 TSecr=3200971988
30	0.341086480	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [ACK] Seq=327685 Ack=5 Win=65536 Len=32768 TSval=3200971988 TSecr=3200971988
31	0.341101988	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=5 Ack=360453 Win=0 Len=0 TSval=3200971988 TSecr=3200971988
32	0.380021492	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=360453 Win=48512 Len=0 TSval=3200972026 TSecr=3200971988
33	0.380040115	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=360453 Ack=5 Win=65536 Len=32768 TSval=3200972026 TSecr=3200972026
34	0.418108941	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=5 Ack=393221 Win=15744 Len=0 TSval=3200972065 TSecr=3200972026
35	0.437565399	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=393221 Win=65536 Len=0 TSval=3200972084 TSecr=3200972026
36	0.437580918	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=393221 Ack=5 Win=65536 Len=32768 TSval=3200972084 TSecr=3200972084
37	0.437587131	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=425989 Ack=5 Win=65536 Len=32768 TSval=3200972084 TSecr=3200972084
38	0.437735158	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=5 Ack=458757 Win=0 Len=0 TSval=3200972084 TSecr=3200972084
39	0.474395056	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=458757 Win=48512 Len=0 TSval=3200972121 TSecr=3200972084
40	0.474409493	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=458757 Ack=5 Win=65536 Len=32768 TSval=3200972121 TSecr=3200972121
41	0.516821668	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=5 Ack=491525 Win=15744 Len=0 TSval=3200972163 TSecr=3200972121
42	0.543395920	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=491525 Win=65536 Len=0 TSval=3200972190 TSecr=3200972121
43	0.543409046	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=491525 Ack=5 Win=65536 Len=32768 TSval=3200972190 TSecr=3200972190
44	0.543412161	127.0.0.1	127.0.0.1	TCP	4036	55680 → 5080 [PSH, ACK] Seq=524293 Ack=5 Win=65536 Len=3970 TSval=3200972190 TSecr=3200972190
45	0.543422723	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=5 Ack=528263 Win=45952 Len=0 TSval=3200972190 TSecr=3200972190
46	0.577164766	127.0.0.1	127.0.0.1	TCP	70	5080 → 55680 [PSH, ACK] Seq=5 Ack=528263 Win=65536 Len=4 TSval=3200972224 TSecr=3200972190
47	0.577216475	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=528263 Ack=9 Win=65536 Len=32768 TSval=3200972224 TSecr=3200972224
48	0.577278878	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=561031 Ack=9 Win=65536 Len=32768 TSval=3200972224 TSecr=3200972224
49	0.577662525	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=9 Ack=593799 Win=0 Len=0 TSval=3200972224 TSecr=3200972224
50	0.609245949	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=593799 Win=48512 Len=0 TSval=3200972256 TSecr=3200972224
51	0.609260233	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=593799 Ack=9 Win=65536 Len=32768 TSval=3200972256 TSecr=3200972256
52	0.652648616	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=9 Ack=626567 Win=15744 Len=0 TSval=3200972299 TSecr=3200972256
53	0.674278492	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=626567 Win=65536 Len=0 TSval=3200972321 TSecr=3200972256
54	0.674292258	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=626567 Ack=9 Win=65536 Len=32768 TSval=3200972321 TSecr=3200972321
55	0.674298922	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [ACK] Seq=659335 Ack=9 Win=65536 Len=32768 TSval=3200972321 TSecr=3200972321
56	0.674383518	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=9 Ack=692103 Win=0 Len=0 TSval=3200972321 TSecr=3200972321
57	0.707508090	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=692103 Win=48512 Len=0 TSval=3200972354 TSecr=3200972321
58	0.707521911	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=692103 Ack=9 Win=65536 Len=32768 TSval=3200972354 TSecr=3200972354
59	0.755923113	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=9 Ack=724871 Win=15744 Len=0 TSval=3200972402 TSecr=3200972354
60	0.819586721	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=724871 Win=65536 Len=0 TSval=3200972466 TSecr=3200972354
61	0.819601368	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=724871 Ack=9 Win=65536 Len=32768 TSval=3200972466 TSecr=3200972466
62	0.819608448	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=757639 Ack=9 Win=65536 Len=32768 TSval=3200972466 TSecr=3200972466
63	0.819622403	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=9 Ack=790407 Win=0 Len=0 TSval=3200972466 TSecr=3200972466
64	0.846536525	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=790407 Win=48512 Len=0 TSval=3200972493 TSecr=3200972466
65	0.846634436	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=790407 Ack=9 Win=65536 Len=32768 TSval=3200972493 TSecr=3200972493
66	0.888816636	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=9 Ack=823175 Win=15744 Len=0 TSval=3200972535 TSecr=3200972493
67	0.910522284	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=823175 Win=65536 Len=0 TSval=3200972557 TSecr=3200972493
68	0.910560530	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=823175 Ack=9 Win=65536 Len=32768 TSval=3200972557 TSecr=3200972557
69	0.910573416	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [ACK] Seq=855943 Ack=9 Win=65536 Len=32768 TSval=3200972557 TSecr=3200972557
70	0.910601807	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=9 Ack=888711 Win=0 Len=0 TSval=3200972557 TSecr=3200972557
71	0.942203565	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=888711 Win=48512 Len=0 TSval=3200972589 TSecr=3200972557
72	0.942216276	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=888711 Ack=9 Win=65536 Len=32768 TSval=3200972589 TSecr=3200972589
73	0.986140169	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=9 Ack=921479 Win=15744 Len=0 TSval=3200972633 TSecr=3200972589
74	1.033610043	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=921479 Win=65536 Len=0 TSval=3200972680 TSecr=3200972589
75	1.033628706	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=921479 Ack=9 Win=65536 Len=32768 TSval=3200972680 TSecr=3200972680
76	1.033638130	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=954247 Ack=9 Win=65536 Len=32768 TSval=3200972680 TSecr=3200972680
77	1.033652721	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 55680 [ACK] Seq=9 Ack=987015 Win=0 Len=0 TSval=3200972680 TSecr=3200972680
78	1.076762971	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=987015 Win=48512 Len=0 TSval=3200972723 TSecr=3200972680
79	1.076778900	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [ACK] Seq=987015 Ack=9 Win=65536 Len=32768 TSval=3200972723 TSecr=3200972723
80	1.119089242	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=9 Ack=1019783 Win=15744 Len=0 TSval=3200972766 TSecr=3200972723
81	1.135706352	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 55680 [ACK] Seq=9 Ack=1019783 Win=65536 Len=0 TSval=3200972782 TSecr=3200972723
82	1.135791564	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5080 [PSH, ACK] Seq=1019783 Ack=9 Win=65536 Len=32768 TSval=3200972782 TSecr=3200972782
83	1.135797654	127.0.0.1	127.0.0.1	TCP	4036	55680 → 5080 [PSH, ACK] Seq=1052551 Ack=9 Win=65536 Len=3970 TSval=3200972782 TSecr=3200972782
84	1.135813271	127.0.0.1	127.0.0.1	TCP	66	5080 → 55680 [ACK] Seq=9 Ack=1056521 Win=45952 Len=0 TSval=3200972782 TSecr=3200972782
85	4.425240221	127.0.0.1	127.0.0.1	TCP	70	55680 → 5080 [PSH, ACK] Seq=1056521 Ack=9 Win=65536 Len=4 TSval=3200976074 TSecr=3200972782

Sending number 5:



325	15.346294917	127.0.0.1	127.0.0.1	TCP	70	55680 → 5000 [PSH, ACK] Seq=4226885 Ack=21 Win=65536 Len=4 TSval=3200986993 TSecr=3200983519
326	15.346307723	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4226885 Win=64896 Len=0 TSval=3200986993 TSecr=3200986993
327	15.346361129	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4226885 Ack=21 Win=65536 Len=32768 TSval=3200986993 TSecr=3200986993
328	15.346371969	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4258853 Win=48512 Len=0 TSval=3200986993 TSecr=3200986993
329	15.346533884	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4258853 Ack=21 Win=65536 Len=32768 TSval=3200986993 TSecr=3200986993
330	15.388729578	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4291621 Win=15744 Len=0 TSval=3200987035 TSecr=3200986993
331	15.426547008	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4291621 Win=65536 Len=0 TSval=3200987073 TSecr=3200986993
332	15.426561834	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4291621 Ack=21 Win=65536 Len=32768 TSval=3200987073 TSecr=3200987073
333	15.426568749	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [PSH, ACK] Seq=4324389 Ack=21 Win=65536 Len=32768 TSval=3200987073 TSecr=3200987073
334	15.426585550	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=21 Ack=4357157 Win=0 Len=0 TSval=3200987073 TSecr=3200987073
335	15.454135864	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4357157 Win=48512 Len=0 TSval=3200987101 TSecr=3200987073
336	15.454150178	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4357157 Ack=21 Win=65536 Len=32768 TSval=3200987101 TSecr=3200987101
337	15.496707813	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4389925 Win=15744 Len=0 TSval=3200987143 TSecr=3200987101
338	15.511123843	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4389925 Win=65536 Len=0 TSval=3200987158 TSecr=3200987101
339	15.511137638	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4389925 Ack=21 Win=65536 Len=32768 TSval=3200987158 TSecr=3200987158
340	15.511144926	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [ACK] Seq=4422693 Ack=21 Win=65536 Len=32768 TSval=3200987158 TSecr=3200987158
341	15.511161277	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=21 Ack=4455461 Win=0 Len=0 TSval=3200987158 TSecr=3200987158
342	15.540063824	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4455461 Win=48512 Len=0 TSval=3200987187 TSecr=3200987158
343	15.540165366	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4455461 Ack=21 Win=65536 Len=32768 TSval=3200987187 TSecr=3200987187
344	15.580671396	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4488229 Win=15744 Len=0 TSval=3200987227 TSecr=3200987187
345	15.611393339	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4488229 Win=65536 Len=0 TSval=3200987258 TSecr=3200987187
346	15.611439184	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4488229 Ack=21 Win=65536 Len=32768 TSval=3200987258 TSecr=3200987258
347	15.611459924	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [PSH, ACK] Seq=4520997 Ack=21 Win=65536 Len=32768 TSval=3200987258 TSecr=3200987258
348	15.611503294	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=21 Ack=4553765 Win=0 Len=0 TSval=3200987258 TSecr=3200987258
349	15.645791302	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4553765 Win=48512 Len=0 TSval=3200987292 TSecr=3200987258
350	15.645806317	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4553765 Ack=21 Win=65536 Len=32768 TSval=3200987292 TSecr=3200987292
351	15.689307929	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4586533 Win=15744 Len=0 TSval=3200987336 TSecr=3200987292
352	15.706940028	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4586533 Win=65536 Len=0 TSval=3200987353 TSecr=3200987292
353	15.706954075	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4586533 Ack=21 Win=65536 Len=32768 TSval=3200987353 TSecr=3200987353
354	15.706961837	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [ACK] Seq=4619301 Ack=21 Win=65536 Len=32768 TSval=3200987353 TSecr=3200987353
355	15.706975291	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=21 Ack=4652069 Win=0 Len=0 TSval=3200987353 TSecr=3200987353
356	15.733053225	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4652069 Win=48512 Len=0 TSval=3200987379 TSecr=3200987353
357	15.733070148	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4652069 Ack=21 Win=65536 Len=32768 TSval=3200987380 TSecr=3200987379
358	15.776722535	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4684837 Win=15744 Len=0 TSval=3200987423 TSecr=3200987380
359	15.784746864	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4684837 Win=65536 Len=0 TSval=3200987431 TSecr=3200987380
360	15.784823668	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4684837 Ack=21 Win=65536 Len=32768 TSval=3200987431 TSecr=3200987431
361	15.784832419	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [PSH, ACK] Seq=4717605 Ack=21 Win=65536 Len=32768 TSval=3200987431 TSecr=3200987431
362	15.784849737	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=21 Ack=4750373 Win=0 Len=0 TSval=3200987431 TSecr=3200987431
363	15.810999384	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=21 Ack=4750373 Win=48512 Len=0 TSval=3200987457 TSecr=3200987431
364	15.811010113	127.0.0.1	127.0.0.1	TCP	4036	55680 → 5000 [PSH, ACK] Seq=4750373 Ack=21 Win=65536 Len=3970 TSval=3200987457 TSecr=3200987457
365	15.840368028	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=21 Ack=4754343 Win=65536 Len=0 TSval=3200987487 TSecr=3200987457
366	15.840490766	127.0.0.1	127.0.0.1	TCP	70	5000 → 55680 [PSH, ACK] Seq=21 Ack=4754343 Win=65536 Len=4 TSval=3200987487 TSecr=3200987457
367	15.840846777	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4754343 Ack=25 Win=65536 Len=32768 TSval=3200987487 TSecr=3200987487
368	15.841008064	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [PSH, ACK] Seq=4787111 Ack=25 Win=65536 Len=32768 TSval=3200987487 TSecr=3200987487
369	15.841229744	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=25 Ack=4819879 Win=0 Len=0 TSval=3200987488 TSecr=3200987487
370	15.879432568	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=4819879 Win=48512 Len=0 TSval=3200987526 TSecr=3200987487
371	15.879446693	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4819879 Ack=25 Win=65536 Len=32768 TSval=3200987526 TSecr=3200987526
372	15.920625372	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=4852647 Win=15744 Len=0 TSval=3200987567 TSecr=3200987526
373	15.948926781	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=4852647 Win=65536 Len=0 TSval=3200987595 TSecr=3200987526
374	15.948947851	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4852647 Ack=25 Win=65536 Len=32768 TSval=3200987595 TSecr=3200987595
375	15.948956672	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [ACK] Seq=4885415 Ack=25 Win=65536 Len=32768 TSval=3200987595 TSecr=3200987595
376	15.948970654	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=25 Ack=4918183 Win=0 Len=0 TSval=3200987595 TSecr=3200987595
377	15.975820368	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=4918183 Win=48512 Len=0 TSval=3200987622 TSecr=3200987595
378	15.975904875	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=4918183 Ack=25 Win=65536 Len=32768 TSval=3200987622 TSecr=3200987622
379	16.016636667	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=4950951 Win=15744 Len=0 TSval=3200987663 TSecr=3200987622
380	16.028795160	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=4950951 Win=65536 Len=0 TSval=3200987675 TSecr=3200987622
381	16.028805884	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=4950951 Ack=25 Win=65536 Len=32768 TSval=3200987675 TSecr=3200987675
382	16.028812315	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [PSH, ACK] Seq=4983719 Ack=25 Win=65536 Len=32768 TSval=3200987675 TSecr=3200987675
383	16.028824976	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=25 Ack=5016487 Win=0 Len=0 TSval=3200987675 TSecr=3200987675
384	16.053954065	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=5016487 Win=48512 Len=0 TSval=3200987700 TSecr=3200987675
385	16.053968125	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=5016487 Ack=25 Win=65536 Len=32768 TSval=3200987700 TSecr=3200987700
386	16.104932003	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=5049255 Win=65536 Len=0 TSval=3200987751 TSecr=3200987700
387	16.104952254	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=5049255 Ack=25 Win=65536 Len=32768 TSval=3200987751 TSecr=3200987751
388	16.104960435	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [ACK] Seq=5082023 Ack=25 Win=65536 Len=32768 TSval=3200987751 TSecr=3200987751
389	16.104984275	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=25 Ack=5114791 Win=0 Len=0 TSval=3200987751 TSecr=3200987751
390	16.137932937	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=5114791 Win=48512 Len=0 TSval=3200987784 TSecr=3200987751
391	16.137949773	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=5114791 Ack=25 Win=65536 Len=32768 TSval=3200987784 TSecr=3200987784
392	16.178400949	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=5147559 Win=15744 Len=0 TSval=3200987825 TSecr=3200987784
393	16.202687851	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=5147559 Win=65536 Len=0 TSval=3200987849 TSecr=3200987784
394	16.202789574	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=5147559 Ack=25 Win=65536 Len=32768 TSval=3200987849 TSecr=3200987849
395	16.202802225	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 55680 → 5000 [PSH, ACK] Seq=5180327 Ack=25 Win=65536 Len=32768 TSval=3200987849 TSecr=3200987849
396	16.202823524	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5000 → 55680 [ACK] Seq=25 Ack=5213095 Win=0 Len=0 TSval=3200987849 TSecr=3200987849
397	16.228356937	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=5213095 Win=48512 Len=0 TSval=3200987875 TSecr=3200987849
398	16.228370531	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [ACK] Seq=5213095 Ack=25 Win=65536 Len=32768 TSval=3200987875 TSecr=3200987875
399	16.266218882	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=5245863 Win=15744 Len=0 TSval=3200987913 TSecr=3200987875
400	16.292296795	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5000 → 55680 [ACK] Seq=25 Ack=5245863 Win=65536 Len=0 TSval=3200987939 TSecr=3200987875
401	16.292329518	127.0.0.1	127.0.0.1	TCP	32834	55680 → 5000 [PSH, ACK] Seq=5245863 Ack=25 Win=65536 Len=32768 TSval=3200987939 TSecr=3200987939
402	16.292332577	127.0.0.1	127.0.0.1	TCP	4036	55680 → 5000 [PSH, ACK] Seq=5278631 Ack=25 Win=65536 Len=3970 TSval=3200987939 TSecr=3200987939
403	16.292457361	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=5282601 Win=45952 Len=0 TSval=3200987939 TSecr=3200987939
404	21.866609782	127.0.0.1	127.0.0.1	TCP	70	55680 → 5000 [PSH, ACK] Seq=5282601 Ack=25 Win=65536 Len=4 TSval=3200993513 TSecr=3200987939
405	21.866621551	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [ACK] Seq=25 Ack=5282605 Win=64896 Len=0 TSval=3200993513 TSecr=3200993513
406	21.866641697	127.0.0.1	127.0.0.1	TCP	66	55680 → 5000 [FIN, ACK] Seq=5282605 Ack=25 Win=65536 Len=0 TSval=3200993513 TSecr=3200993513
407	21.867339480	127.0.0.1	127.0.0.1	TCP	66	5000 → 55680 [FIN, ACK] Seq=25 Ack=5282606 Win=65536 Len=0 TSval=3200993514 TSecr=3200993513
408	21.867352109	127.0.0.1	127.0.0.1	TCP	66	55680 → 5000 [ACK] Seq=5282606 Ack=26 Win=65536 Len=0 TSval=3200993514 TSecr=3200993514

Zoom in:

Open connection in line 1, 2.

```
74 55680 → 5080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3200971646 TSecr=0 WS=128
74 5080 → 55680 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3200971646 TSecr=3200971646 WS=128
```

The "[TCP Window Full]" message from Wireshark means that **the system sending this TCP segment has filled up the receive window of the other end with the tcp segment in this packet**. Or put differently: the last received window size of the other end is equal to the length of the TCP segment in this packet

Zero Window means that **the receiver of the packets waves a "white flag" towards the sender, telling it to stop sending because there is no more buffer space for incoming packets**. This is in almost all cases a sign of the receiver being too slow to process the incoming packets in time.

A packet marked "TCP Window Update" simply **indicates that the sender's TCP receive buffer space has increased**. Look at the previous packet from the sender - note the Window Size value in the TCP header.

```
32834 [TCP Window Full] 55680 → 5080 [PSH, ACK] Seq=32773 Ack=5 Win=65536 Len=32768 TSval=3200971653 TSecr=3200971647
   66 [TCP ZeroWindow] 5080 → 55680 [ACK] Seq=5 Ack=65541 Win=0 Len=0 TSval=3200971653 TSecr=3200971648
   66 [TCP Window Update] 5080 → 55680 [ACK] Seq=5 Ack=65541 Win=48512 Len=0 TSval=3200971691 TSecr=3200971648
```

Checking for authentication in line 46, we can see that the receiver sent it after he get the first half of the file, and that the packet length of the is 4 (int)

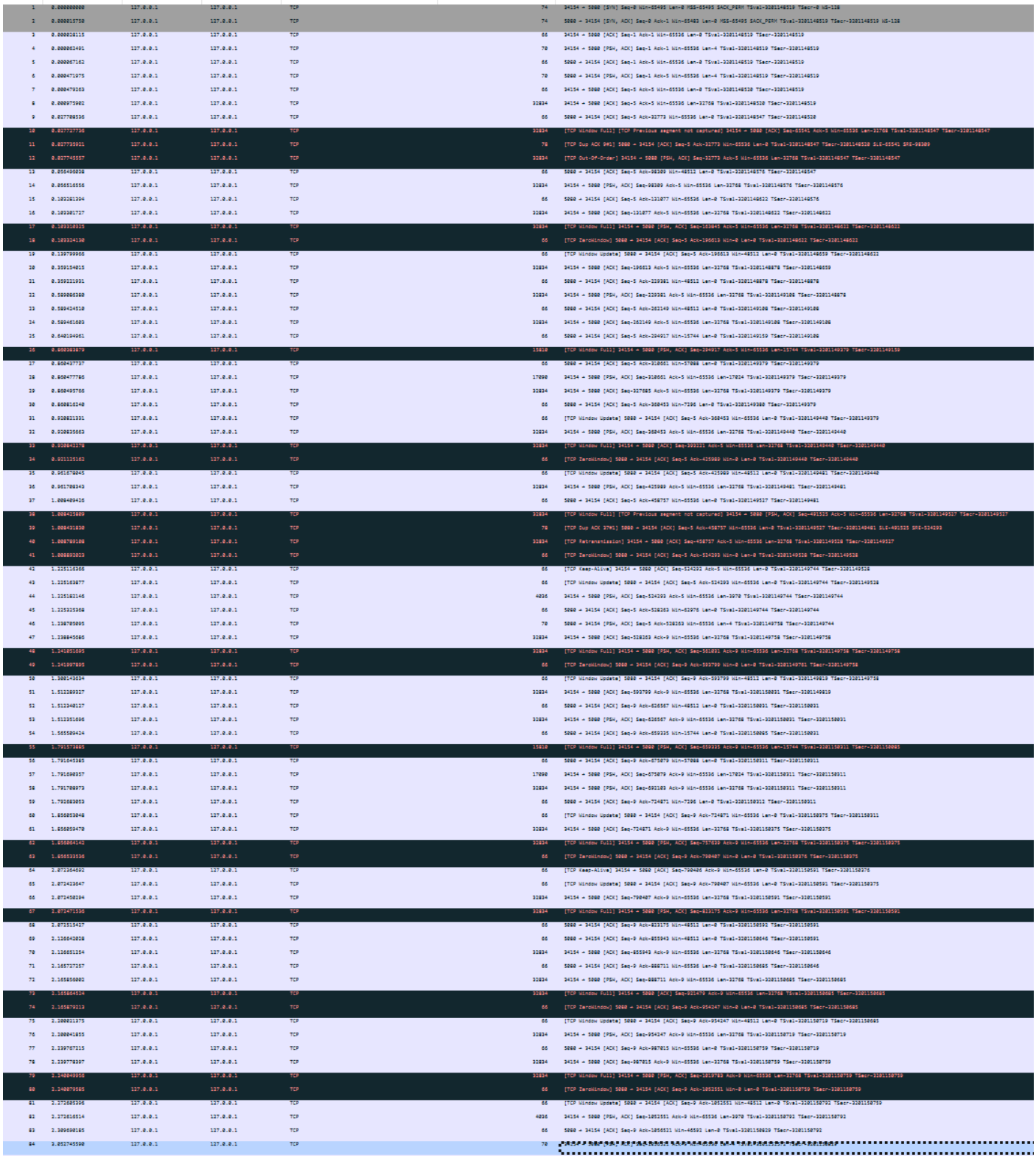
```
70 5080 → 55680 [PSH, ACK] Seq=5 Ack=528263 Win=65536 Len=4 TSval=3200972224 TSecr=3200972190
```

close the socket 406, 407, 408

```
66 55680 → 5080 [FIN, ACK] Seq=5282605 Ack=25 Win=65536 Len=0 TSval=3200993513 TSecr=3200993513
66 5080 → 55680 [FIN, ACK] Seq=25 Ack=5282606 Win=65536 Len=0 TSval=3200993514 TSecr=3200993513
66 55680 → 5080 [ACK] Seq=5282606 Ack=26 Win=65536 Len=0 TSval=3200993514 TSecr=3200993514
```

2.1.2 10% lost

Sending number 1:



Sending number 5:

341	13.165112…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=21 Ack=4389925 Win=48512 Len=0 TSval=3201161684 TSecr=3201161637
342	13.165129…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=4389925 Ack=21 Win=65536 Len=32768 TSval=3201161684 TSecr=3201161684
343	13.210251…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4422693 Win=15744 Len=0 TSval=3201161729 TSecr=3201161684
344	13.230735…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=21 Ack=4422693 Win=65536 Len=0 TSval=3201161750 TSecr=3201161684
345	13.230746…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4422693 Ack=21 Win=65536 Len=32768 TSval=3201161750 TSecr=3201161750
346	13.230753…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 34154 → 5080 [PSH, ACK] Seq=4455461 Ack=21 Win=65536 Len=32768 TSval=3201161750 TSecr=3201161750
347	13.230765…	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 34154 [ACK] Seq=21 Ack=4488229 Win=0 Len=0 TSval=3201161750 TSecr=3201161750
348	13.264693…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=21 Ack=4488229 Win=48512 Len=0 TSval=3201161784 TSecr=3201161750
349	13.264787…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4488229 Ack=21 Win=65536 Len=32768 TSval=3201161784 TSecr=3201161784
350	13.308006…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4520997 Win=15744 Len=0 TSval=3201161827 TSecr=3201161784
351	13.325550…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=21 Ack=4520997 Win=65536 Len=0 TSval=3201161845 TSecr=3201161784
352	13.325583…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=4520997 Ack=21 Win=65536 Len=32768 TSval=3201161845 TSecr=3201161845
353	13.351358…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4553765 Win=65536 Len=0 TSval=3201161870 TSecr=3201161845
354	13.351377…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] [TCP Previous segment not captured] 34154 → 5080 [PSH, ACK] Seq=4586533 Ack=21 Win=65536 Len=32768 TSval=3201161870 TSecr=3201161870
355	13.351386…	127.0.0.1	127.0.0.1	TCP	78	[TCP Dup ACK 353#1] 5080 → 34154 [ACK] Seq=21 Ack=4553765 Win=65536 Len=0 TSval=3201161870 TSecr=3201161845 SLE=4586533 SRE=4619301
356	13.351394…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Out-Of-Order] 34154 → 5080 [ACK] Seq=4553765 Ack=21 Win=65536 Len=32768 TSval=3201161870 TSecr=3201161870
357	13.351497…	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 34154 [ACK] Seq=21 Ack=4619301 Win=0 Len=0 TSval=3201161871 TSecr=3201161870
358	13.568270…	127.0.0.1	127.0.0.1	TCP	66	[TCP Keep-Alive] 34154 → 5080 [ACK] Seq=4619300 Ack=21 Win=65536 Len=0 TSval=3201162087 TSecr=3201161871
359	13.568290…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=21 Ack=4619301 Win=65536 Len=0 TSval=3201162087 TSecr=3201161870
360	13.568302…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4619301 Ack=21 Win=65536 Len=32768 TSval=3201162087 TSecr=3201162087
361	13.568319…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4652069 Win=48512 Len=0 TSval=3201162087 TSecr=3201162087
362	14.006334…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=4652069 Ack=21 Win=65536 Len=32768 TSval=3201162525 TSecr=3201162087
363	14.006350…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4684837 Win=48512 Len=0 TSval=3201162525 TSecr=3201162525
364	14.221704…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4684837 Ack=21 Win=65536 Len=32768 TSval=3201162741 TSecr=3201162525
365	14.221765…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4717605 Win=48512 Len=0 TSval=3201162741 TSecr=3201162741
366	14.221816…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=4717605 Ack=21 Win=65536 Len=32768 TSval=3201162741 TSecr=3201162741
367	14.221844…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4750373 Win=15744 Len=0 TSval=3201162741 TSecr=3201162741
368	14.314139…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=21 Ack=4750373 Win=65536 Len=0 TSval=3201162833 TSecr=3201162741
369	14.432780…	127.0.0.1	127.0.0.1	TCP	4036	34154 → 5080 [PSH, ACK] Seq=4750373 Ack=21 Win=65536 Len=3970 TSval=3201162952 TSecr=3201162833
370	14.432792…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=21 Ack=4754343 Win=62976 Len=0 TSval=3201162952 TSecr=3201162952
371	14.640074…	127.0.0.1	127.0.0.1	TCP	70	5080 → 34154 [PSH, ACK] Seq=21 Ack=4754343 Win=65536 Len=4 TSval=3201163159 TSecr=3201162952
372	14.640354…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4754343 Ack=25 Win=65536 Len=32768 TSval=3201163159 TSecr=3201163159
373	14.640482…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 34154 → 5080 [PSH, ACK] Seq=4787111 Ack=25 Win=65536 Len=32768 TSval=3201163160 TSecr=3201163159
374	14.640484…	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 34154 [ACK] Seq=25 Ack=4819879 Win=0 Len=0 TSval=3201163160 TSecr=3201163159
375	14.705114…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=4819879 Win=48512 Len=0 TSval=3201163224 TSecr=3201163159
376	14.705129…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4819879 Ack=25 Win=65536 Len=32768 TSval=3201163224 TSecr=3201163224
377	14.745593…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=4852647 Win=15744 Len=0 TSval=3201163265 TSecr=3201163224
378	14.766259…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=4852647 Win=65536 Len=0 TSval=3201163285 TSecr=3201163224
379	14.766282…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] [TCP Previous segment not captured] 34154 → 5080 [ACK] Seq=4885415 Ack=25 Win=65536 Len=32768 TSval=3201163285 TSecr=3201163285
380	14.766292…	127.0.0.1	127.0.0.1	TCP	78	[TCP Dup ACK 377#1] 5080 → 34154 [ACK] Seq=25 Ack=4852647 Win=65536 Len=0 TSval=3201163285 TSecr=3201163224 SLE=4885415 SRE=4918183
381	14.791192…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Retransmission] 34154 → 5080 [PSH, ACK] Seq=4852647 Ack=25 Win=65536 Len=32768 TSval=3201163310 TSecr=3201163285
382	14.792078…	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 34154 [ACK] Seq=25 Ack=4918183 Win=0 Len=0 TSval=3201163311 TSecr=3201163310
383	14.853307…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=4918183 Win=48512 Len=0 TSval=3201163372 TSecr=3201163310
384	14.853327…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=4918183 Ack=25 Win=65536 Len=32768 TSval=3201163372 TSecr=3201163372
385	14.871835…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=4950951 Win=15744 Len=0 TSval=3201163391 TSecr=3201163372
386	15.032398…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=4950951 Win=65536 Len=0 TSval=3201163551 TSecr=3201163372
387	15.032414…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=4950951 Ack=25 Win=65536 Len=32768 TSval=3201163551 TSecr=3201163551
388	15.032424…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=4983719 Win=32768 Len=0 TSval=3201163551 TSecr=3201163551
389	15.061972…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=4983719 Win=65536 Len=0 TSval=3201163581 TSecr=3201163551
390	15.061991…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] [TCP Previous segment not captured] 34154 → 5080 [ACK] Seq=5016487 Ack=25 Win=65536 Len=32768 TSval=3201163581 TSecr=3201163581
391	15.062001…	127.0.0.1	127.0.0.1	TCP	78	[TCP Dup ACK 388#1] 5080 → 34154 [ACK] Seq=25 Ack=4983719 Win=65536 Len=0 TSval=3201163581 TSecr=3201163551 SLE=5016487 SRE=5049255
392	15.062012…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Out-Of-Order] 34154 → 5080 [PSH, ACK] Seq=4983719 Ack=25 Win=65536 Len=32768 TSval=3201163581 TSecr=3201163581
393	15.062022…	127.0.0.1	127.0.0.1	TCP	66	[TCP ZeroWindow] 5080 → 34154 [ACK] Seq=25 Ack=5049255 Win=0 Len=0 TSval=3201163581 TSecr=3201163581
394	15.094606…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=5049255 Win=48512 Len=0 TSval=3201163614 TSecr=3201163581
395	15.094631…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=5049255 Ack=25 Win=65536 Len=32768 TSval=3201163614 TSecr=3201163614
396	15.308659…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Retransmission] 34154 → 5080 [PSH, ACK] Seq=5049255 Ack=25 Win=65536 Len=32768 TSval=3201163828 TSecr=3201163614
397	15.308683…	127.0.0.1	127.0.0.1	TCP	78	5080 → 34154 [ACK] Seq=25 Ack=5082023 Win=65536 Len=0 TSval=3201163828 TSecr=3201163828 SLE=5049255 SRE=5082023
398	15.308697…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=5082023 Ack=25 Win=65536 Len=32768 TSval=3201163828 TSecr=3201163828
399	15.308707…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=5114791 Win=48512 Len=0 TSval=3201163828 TSecr=3201163828
400	15.531347…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=5114791 Ack=25 Win=65536 Len=32768 TSval=3201164050 TSecr=3201163828
401	15.531408…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=5147559 Win=48512 Len=0 TSval=3201164050 TSecr=3201164050
402	15.531456…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [ACK] Seq=5147559 Ack=25 Win=65536 Len=32768 TSval=3201164050 TSecr=3201164050
403	15.531481…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=5180327 Win=15744 Len=0 TSval=3201164050 TSecr=3201164050
404	15.663073…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=5180327 Win=65536 Len=0 TSval=3201164182 TSecr=3201164050
405	15.663091…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=5180327 Ack=25 Win=65536 Len=32768 TSval=3201164182 TSecr=3201164182
406	15.663100…	127.0.0.1	127.0.0.1	TCP	32834	[TCP Window Full] 34154 → 5080 [ACK] Seq=5213095 Ack=25 Win=65536 Len=32768 TSval=3201164182 TSecr=3201164182
407	15.686262…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=5245863 Win=48512 Len=0 TSval=3201164205 TSecr=3201164182
408	15.686278…	127.0.0.1	127.0.0.1	TCP	32834	34154 → 5080 [PSH, ACK] Seq=5245863 Ack=25 Win=65536 Len=32768 TSval=3201164205 TSecr=3201164205
409	15.686281…	127.0.0.1	127.0.0.1	TCP	4036	34154 → 5080 [PSH, ACK] Seq=5278631 Ack=25 Win=65536 Len=3970 TSval=3201164205 TSecr=3201164205
410	15.686291…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=5282601 Win=11776 Len=0 TSval=3201164205 TSecr=3201164205
411	15.801116…	127.0.0.1	127.0.0.1	TCP	66	[TCP Window Update] 5080 → 34154 [ACK] Seq=25 Ack=5282601 Win=65536 Len=0 TSval=3201164320 TSecr=3201164205
412	16.904880…	127.0.0.1	127.0.0.1	TCP	70	34154 → 5080 [PSH, ACK] Seq=5282601 Ack=25 Win=65536 Len=4 TSval=3201165424 TSecr=3201164320
413	16.904893…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [ACK] Seq=25 Ack=5282605 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424
414	16.904913…	127.0.0.1	127.0.0.1	TCP	66	34154 → 5080 [FIN, ACK] Seq=5282605 Ack=25 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424
415	16.905253…	127.0.0.1	127.0.0.1	TCP	66	5080 → 34154 [FIN, ACK] Seq=25 Ack=5282606 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424
416	16.905266…	127.0.0.1	127.0.0.1	TCP	66	34154 → 5080 [ACK] Seq=5282606 Ack=26 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424

Zoom in:

Now we will see more packets that gets lost.

Open connection

```
 74 34154 → 5080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3201148519 TSecr=0 WS=128
 74 5080 → 34154 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3201148519 TSecr=3201148519 WS=128
 66 34154 → 5080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3201148519 TSecr=3201148519
```

Lost packets:
"tcp previous segment not captured" is an expert message created by Wireshark when it didn't see a packet that should have been in the trace; this warning was previously called "tcp previous segment lost".

Dup ACK means that you capture at the source of the data (not the receiving side). That is quite normal if the packet loss occurs somewhere in the path to the receiver.

TCP out-of-order means that particular frame was received in a different order from which it was sent (after a later packet in the sequence). It is not generally a problem. It probably indicates there are multiple paths between source and destination - and one travels a through a longer path.

```
32834 [TCP Window Full] [TCP Previous segment not captured] 34154 → 5080 [ACK] Seq=65541 Ack=5 Win=65536 Len=32768 TSval=3201148547 TSecr=3201148547
   78 [TCP Dup ACK 9#1] 5080 → 34154 [ACK] Seq=5 Ack=32773 Win=65536 Len=0 TSval=3201148547 TSecr=3201148520 SLE=65541 SRE=98309
32834 [TCP Out-Of-Order] 34154 → 5080 [PSH, ACK] Seq=32773 Ack=5 Win=65536 Len=32768 TSval=3201148547 TSecr=3201148547
```

TCP Retransmission occurs when the sender retransmits a packet after the expiration of the acknowledgement

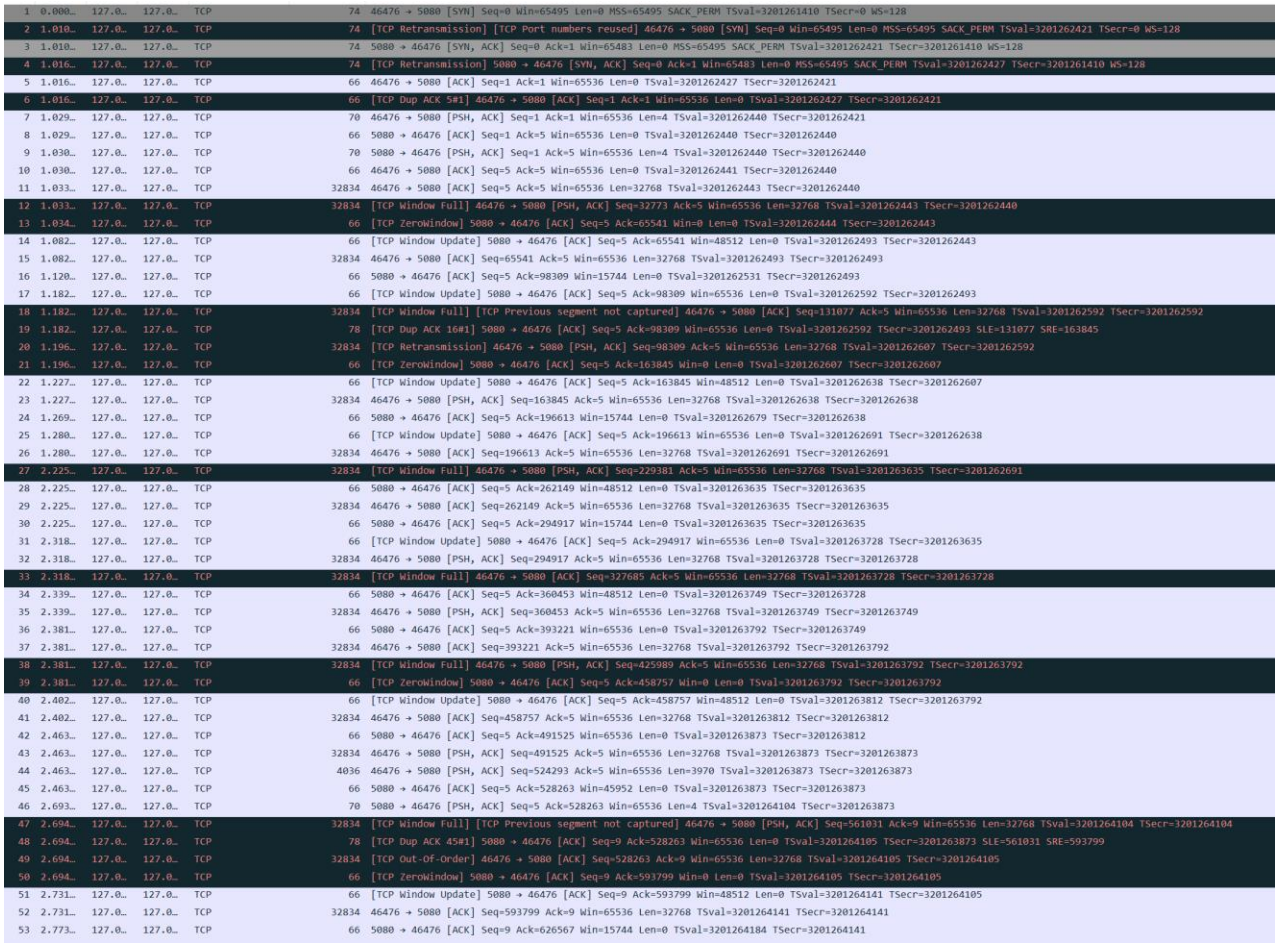
```
396 15.… 127… 127… TCP        32834 [TCP Retransmission] 34154 → 5080 [PSH, ACK] Seq=5049255 Ack=25 Win=65536 Len=32768 TSval=3201163828 TSecr=3201163614
```

Close connection lines 414, 415, 416

```
414 16.… 127… 127… TCP          66 34154 → 5080 [FIN, ACK] Seq=5282605 Ack=25 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424
415 16.… 127… 127… TCP          66 5080 → 34154 [FIN, ACK] Seq=25 Ack=5282606 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424
416 16.… 127… 127… TCP          66 34154 → 5080 [ACK] Seq=5282606 Ack=26 Win=65536 Len=0 TSval=3201165424 TSecr=3201165424
```

2.1.3 15% lost

open connection in lines 1 2



1	0.000…	127.0…	127.0…	TCP	74	46476 → 5080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3201261410 TSecr=0 WS=128
2	1.010…	127.0…	127.0…	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 46476 → 5080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3201262421 TSecr=0 WS=128
3	1.010…	127.0…	127.0…	TCP	74	5080 → 46476 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3201262421 TSecr=3201261410 WS=128
4	1.016…	127.0…	127.0…	TCP	74	[TCP Retransmission] 5080 → 46476 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3201262427 TSecr=3201261410 WS=128
5	1.016…	127.0…	127.0…	TCP	66	46476 → 5080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3201262427 TSecr=3201262421
6	1.016…	127.0…	127.0…	TCP	66	[TCP Dup ACK 5#1] 46476 → 5080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3201262427 TSecr=3201262421
7	1.029…	127.0…	127.0…	TCP	70	46476 → 5080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=3201262440 TSecr=3201262421
8	1.029…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=3201262440 TSecr=3201262440
9	1.030…	127.0…	127.0…	TCP	70	5080 → 46476 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=4 TSval=3201262440 TSecr=3201262440
10	1.030…	127.0…	127.0…	TCP	66	46476 → 5080 [ACK] Seq=5 Ack=5 Win=65536 Len=0 TSval=3201262441 TSecr=3201262440
11	1.033…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=5 Ack=5 Win=65536 Len=32768 TSval=3201262443 TSecr=3201262440
12	1.033…	127.0…	127.0…	TCP	32834	[TCP Window Full] 46476 → 5080 [PSH, ACK] Seq=32773 Ack=5 Win=65536 Len=32768 TSval=3201262443 TSecr=3201262440
13	1.034…	127.0…	127.0…	TCP	66	[TCP ZeroWindow] 5080 → 46476 [ACK] Seq=5 Ack=65541 Win=0 Len=0 TSval=3201262444 TSecr=3201262443
14	1.082…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=5 Ack=65541 Win=48512 Len=0 TSval=3201262493 TSecr=3201262443
15	1.082…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=65541 Ack=5 Win=65536 Len=32768 TSval=3201262493 TSecr=3201262493
16	1.120…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=98309 Win=15744 Len=0 TSval=3201262531 TSecr=3201262493
17	1.182…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=5 Ack=98309 Win=65536 Len=0 TSval=3201262592 TSecr=3201262493
18	1.182…	127.0…	127.0…	TCP	32834	[TCP Window Full] [TCP Previous segment not captured] 46476 → 5080 [ACK] Seq=131077 Ack=5 Win=65536 Len=32768 TSval=3201262592 TSecr=3201262592
19	1.182…	127.0…	127.0…	TCP	78	[TCP Dup ACK 16#1] 5080 → 46476 [ACK] Seq=5 Ack=98309 Win=65536 Len=0 TSval=3201262592 TSecr=3201262493 SLE=131077 SRE=163845
20	1.196…	127.0…	127.0…	TCP	32834	[TCP Retransmission] 46476 → 5080 [PSH, ACK] Seq=98309 Ack=5 Win=65536 Len=32768 TSval=3201262607 TSecr=3201262592
21	1.196…	127.0…	127.0…	TCP	66	[TCP ZeroWindow] 5080 → 46476 [ACK] Seq=5 Ack=163845 Win=0 Len=0 TSval=3201262607 TSecr=3201262607
22	1.227…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=5 Ack=163845 Win=48512 Len=0 TSval=3201262638 TSecr=3201262607
23	1.227…	127.0…	127.0…	TCP	32834	46476 → 5080 [PSH, ACK] Seq=163845 Ack=5 Win=65536 Len=32768 TSval=3201262638 TSecr=3201262638
24	1.269…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=196613 Win=15744 Len=0 TSval=3201262679 TSecr=3201262638
25	1.280…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=5 Ack=196613 Win=65536 Len=0 TSval=3201262691 TSecr=3201262638
26	1.280…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=196613 Ack=5 Win=65536 Len=32768 TSval=3201262691 TSecr=3201262691
27	2.225…	127.0…	127.0…	TCP	32834	[TCP Window Full] 46476 → 5080 [PSH, ACK] Seq=229381 Ack=5 Win=65536 Len=32768 TSval=3201263635 TSecr=3201262691
28	2.225…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=262149 Win=48512 Len=0 TSval=3201263635 TSecr=3201263635
29	2.225…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=262149 Ack=5 Win=65536 Len=32768 TSval=3201263635 TSecr=3201263635
30	2.225…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=294917 Win=15744 Len=0 TSval=3201263635 TSecr=3201263635
31	2.318…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=5 Ack=294917 Win=65536 Len=0 TSval=3201263728 TSecr=3201263635
32	2.318…	127.0…	127.0…	TCP	32834	46476 → 5080 [PSH, ACK] Seq=294917 Ack=5 Win=65536 Len=32768 TSval=3201263728 TSecr=3201263728
33	2.318…	127.0…	127.0…	TCP	32834	[TCP Window Full] 46476 → 5080 [ACK] Seq=327685 Ack=5 Win=65536 Len=32768 TSval=3201263728 TSecr=3201263728
34	2.339…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=360453 Win=48512 Len=0 TSval=3201263749 TSecr=3201263728
35	2.339…	127.0…	127.0…	TCP	32834	46476 → 5080 [PSH, ACK] Seq=360453 Ack=5 Win=65536 Len=32768 TSval=3201263749 TSecr=3201263749
36	2.381…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=393221 Win=65536 Len=0 TSval=3201263792 TSecr=3201263749
37	2.381…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=393221 Ack=5 Win=65536 Len=32768 TSval=3201263792 TSecr=3201263792
38	2.381…	127.0…	127.0…	TCP	32834	[TCP Window Full] 46476 → 5080 [PSH, ACK] Seq=425989 Ack=5 Win=65536 Len=32768 TSval=3201263792 TSecr=3201263792
39	2.381…	127.0…	127.0…	TCP	66	[TCP ZeroWindow] 5080 → 46476 [ACK] Seq=5 Ack=458757 Win=0 Len=0 TSval=3201263792 TSecr=3201263792
40	2.402…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=5 Ack=458757 Win=48512 Len=0 TSval=3201263812 TSecr=3201263792
41	2.402…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=458757 Ack=5 Win=65536 Len=32768 TSval=3201263812 TSecr=3201263812
42	2.463…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=491525 Win=65536 Len=0 TSval=3201263873 TSecr=3201263812
43	2.463…	127.0…	127.0…	TCP	32834	46476 → 5080 [PSH, ACK] Seq=491525 Ack=5 Win=65536 Len=32768 TSval=3201263873 TSecr=3201263873
44	2.463…	127.0…	127.0…	TCP	4036	46476 → 5080 [PSH, ACK] Seq=524293 Ack=5 Win=65536 Len=3970 TSval=3201263873 TSecr=3201263873
45	2.463…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=5 Ack=528263 Win=45952 Len=0 TSval=3201263873 TSecr=3201263873
46	2.693…	127.0…	127.0…	TCP	70	5080 → 46476 [PSH, ACK] Seq=5 Ack=528263 Win=65536 Len=4 TSval=3201264104 TSecr=3201263873
47	2.694…	127.0…	127.0…	TCP	32834	[TCP Window Full] [TCP Previous segment not captured] 46476 → 5080 [PSH, ACK] Seq=561031 Ack=9 Win=65536 Len=32768 TSval=3201264104 TSecr=3201264104
48	2.694…	127.0…	127.0…	TCP	78	[TCP Dup ACK 45#1] 5080 → 46476 [ACK] Seq=9 Ack=528263 Win=65536 Len=0 TSval=3201264105 TSecr=3201263873 SLE=561031 SRE=593799
49	2.694…	127.0…	127.0…	TCP	32834	[TCP Out-Of-Order] 46476 → 5080 [ACK] Seq=528263 Ack=9 Win=65536 Len=32768 TSval=3201264105 TSecr=3201264105
50	2.694…	127.0…	127.0…	TCP	66	[TCP ZeroWindow] 5080 → 46476 [ACK] Seq=9 Ack=593799 Win=0 Len=0 TSval=3201264105 TSecr=3201264105
51	2.731…	127.0…	127.0…	TCP	66	[TCP Window Update] 5080 → 46476 [ACK] Seq=9 Ack=593799 Win=48512 Len=0 TSval=3201264141 TSecr=3201264105
52	2.731…	127.0…	127.0…	TCP	32834	46476 → 5080 [ACK] Seq=593799 Ack=9 Win=65536 Len=32768 TSval=3201264141 TSecr=3201264141
53	2.773…	127.0…	127.0…	TCP	66	5080 → 46476 [ACK] Seq=9 Ack=626567 Win=15744 Len=0 TSval=3201264184 TSecr=3201264141

Close the connection in lines 256, 257, 258

203	13.0176…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=3763306 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274426
204	13.0179…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=3828789 Win=1964032 Len=0 TSval=3201274428 TSecr=3201274427
205	13.0179…	127.0.0…	127.0.0…	TCP	65549	[TCP Previous segment not captured] 46476 → 5080 [ACK] Seq=3894272 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274428
206	13.0179…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=3959755 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274428
207	13.0180…	127.0.0…	127.0.0…	TCP	78	[TCP Dup ACK 204#1] 5080 → 46476 [ACK] Seq=21 Ack=3828789 Win=1964032 Len=0 TSval=3201274428 TSecr=3201274427 SLE=3894272 SRE=3959755
208	13.0180…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4025238 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274428
209	13.0180…	127.0.0…	127.0.0…	TCP	78	[TCP Dup ACK 204#2] 5080 → 46476 [ACK] Seq=21 Ack=3828789 Win=1964032 Len=0 TSval=3201274428 TSecr=3201274427 SLE=3894272 SRE=4025238
210	13.0180…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4090721 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274428
211	13.0180…	127.0.0…	127.0.0…	TCP	78	[TCP Dup ACK 204#3] 5080 → 46476 [ACK] Seq=21 Ack=3828789 Win=1964032 Len=0 TSval=3201274428 TSecr=3201274427 SLE=3894272 SRE=4090721
212	13.0181…	127.0.0…	127.0.0…	TCP	65549	[TCP Fast Retransmission] 46476 → 5080 [ACK] Seq=3828789 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274428
213	13.0184…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4156204 Win=1636736 Len=0 TSval=3201274428 TSecr=3201274428
214	13.0184…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4156204 Ack=21 Win=65536 Len=65483 TSval=3201274429 TSecr=3201274428
215	13.0184…	127.0.0…	127.0.0…	TCP	4460	46476 → 5080 [PSH, ACK] Seq=4221687 Ack=21 Win=65536 Len=4394 TSval=3201274429 TSecr=3201274428
216	13.0444…	127.0.0…	127.0.0…	TCP	4460	[TCP Retransmission] 46476 → 5080 [PSH, ACK] Seq=4221687 Ack=21 Win=65536 Len=4394 TSval=3201274455 TSecr=3201274428
217	13.2490…	127.0.0…	127.0.0…	TCP	65549	[TCP Retransmission] 46476 → 5080 [ACK] Seq=4156204 Ack=21 Win=65536 Len=65483 TSval=3201274659 TSecr=3201274428
218	13.2490…	127.0.0…	127.0.0…	TCP	78	5080 → 46476 [ACK] Seq=21 Ack=4226081 Win=1798144 Len=0 TSval=3201274659 TSecr=3201274455 SLE=4156204 SRE=4221687
219	14.7118…	127.0.0…	127.0.0…	TCP	70	46476 → 5080 [PSH, ACK] Seq=4226081 Ack=21 Win=65536 Len=4 TSval=3201276122 TSecr=3201274659
220	14.7118…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4226085 Win=2028032 Len=0 TSval=3201276122 TSecr=3201276122
221	14.7121…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4226085 Ack=21 Win=65536 Len=65483 TSval=3201276122 TSecr=3201276122
222	14.7121…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4291568 Win=1997440 Len=0 TSval=3201276122 TSecr=3201276122
223	14.7121…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4291568 Ack=21 Win=65536 Len=65483 TSval=3201276122 TSecr=3201276122
224	14.7123…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4357051 Win=1964672 Len=0 TSval=3201276122 TSecr=3201276122
225	14.7123…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4357051 Ack=21 Win=65536 Len=65483 TSval=3201276123 TSecr=3201276122
226	14.7123…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4422534 Win=1932032 Len=0 TSval=3201276123 TSecr=3201276123
227	14.7124…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4422534 Ack=21 Win=65536 Len=65483 TSval=3201276123 TSecr=3201276122
228	14.7125…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4488017 Win=1899264 Len=0 TSval=3201276123 TSecr=3201276123
229	14.7125…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4488017 Ack=21 Win=65536 Len=65483 TSval=3201276123 TSecr=3201276122
230	14.7125…	127.0.0…	127.0.0…	TCP	65549	[TCP Previous segment not captured] 46476 → 5080 [ACK] Seq=4618983 Ack=21 Win=65536 Len=65483 TSval=3201276123 TSecr=3201276123
231	14.7125…	127.0.0…	127.0.0…	TCP	78	5080 → 46476 [ACK] Seq=21 Ack=4553500 Win=1866496 Len=0 TSval=3201276123 TSecr=3201276123 SLE=4618983 SRE=4684466
232	14.7125…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4684466 Ack=21 Win=65536 Len=65483 TSval=3201276123 TSecr=3201276123
233	14.7125…	127.0.0…	127.0.0…	TCP	4460	46476 → 5080 [PSH, ACK] Seq=4749949 Ack=21 Win=65536 Len=4394 TSval=3201276123 TSecr=3201276123
234	14.7126…	127.0.0…	127.0.0…	TCP	78	[TCP Dup ACK 231#1] 5080 → 46476 [ACK] Seq=21 Ack=4553500 Win=1866496 Len=0 TSval=3201276123 TSecr=3201276123 SLE=4618983 SRE=4749949
235	14.7126…	127.0.0…	127.0.0…	TCP	65549	[TCP Out-Of-Order] 46476 → 5080 [ACK] Seq=4553500 Ack=21 Win=65536 Len=65483 TSval=3201276123 TSecr=3201276123
236	14.7126…	127.0.0…	127.0.0…	TCP	78	[TCP Dup ACK 231#2] 5080 → 46476 [ACK] Seq=21 Ack=4553500 Win=1866496 Len=0 TSval=3201276123 TSecr=3201276123 SLE=4618983 SRE=4754343
237	14.7126…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=21 Ack=4754343 Win=1665664 Len=0 TSval=3201276123 TSecr=3201276123
238	15.2343…	127.0.0…	127.0.0…	TCP	70	5080 → 46476 [PSH, ACK] Seq=21 Ack=4754343 Win=2028032 Len=4 TSval=3201276644 TSecr=3201276123
239	15.2350…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4754343 Ack=25 Win=65536 Len=65483 TSval=3201276645 TSecr=3201276644
240	15.2350…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=25 Ack=4819826 Win=1997440 Len=0 TSval=3201276645 TSecr=3201276645
241	15.2357…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4819826 Ack=25 Win=65536 Len=65483 TSval=3201276646 TSecr=3201276644
242	15.2359…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=25 Ack=4885309 Win=1964032 Len=0 TSval=3201276646 TSecr=3201276646
243	15.2361…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4885309 Ack=25 Win=65536 Len=65483 TSval=3201276646 TSecr=3201276645
244	15.2362…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=4950792 Ack=25 Win=65536 Len=65483 TSval=3201276646 TSecr=3201276645
245	15.2362…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=25 Ack=5016275 Win=1897344 Len=0 TSval=3201276646 TSecr=3201276646
246	15.2363…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=5016275 Ack=25 Win=65536 Len=65483 TSval=3201276646 TSecr=3201276646
247	15.2363…	127.0.0…	127.0.0…	TCP	65549	46476 → 5080 [ACK] Seq=5081758 Ack=25 Win=65536 Len=65483 TSval=3201276646 TSecr=3201276646
248	15.2363…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=25 Ack=5081758 Win=1831936 Len=0 TSval=3201276646 TSecr=3201276646
249	15.2363…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=25 Ack=5147241 Win=1798528 Len=0 TSval=3201276646 TSecr=3201276646
250	15.2365…	127.0.0…	127.0.0…	TCP	4460	[TCP Previous segment not captured] 46476 → 5080 [PSH, ACK] Seq=5278207 Ack=25 Win=65536 Len=4394 TSval=3201276646 TSecr=3201276646
251	15.2365…	127.0.0…	127.0.0…	TCP	78	[TCP Dup ACK 249#1] 5080 → 46476 [ACK] Seq=25 Ack=5147241 Win=1798528 Len=0 TSval=3201276647 TSecr=3201276646 SLE=5278207 SRE=5282601
252	15.2365…	127.0.0…	127.0.0…	TCP	65549	[TCP Out-Of-Order] 46476 → 5080 [ACK] Seq=5147241 Ack=25 Win=65536 Len=65483 TSval=3201276647 TSecr=3201276647
253	15.2365…	127.0.0…	127.0.0…	TCP	78	5080 → 46476 [ACK] Seq=25 Ack=5212724 Win=1733120 Len=0 TSval=3201276647 TSecr=3201276647 SLE=5278207 SRE=5282601
254	15.2366…	127.0.0…	127.0.0…	TCP	65549	[TCP Out-Of-Order] 46476 → 5080 [ACK] Seq=5212724 Ack=25 Win=65536 Len=65483 TSval=3201276647 TSecr=3201276647
255	15.2367…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [ACK] Seq=25 Ack=5282601 Win=1663360 Len=0 TSval=3201276647 TSecr=3201276647
256	17.1930…	127.0.0…	127.0.0…	TCP	70	46476 → 5080 [FIN, PSH, ACK] Seq=5282601 Ack=25 Win=65536 Len=4 TSval=3201278603 TSecr=3201276647
257	17.1933…	127.0.0…	127.0.0…	TCP	66	5080 → 46476 [FIN, ACK] Seq=25 Ack=5282606 Win=2028032 Len=0 TSval=3201278603 TSecr=3201278603
258	17.1933…	127.0.0…	127.0.0…	TCP	66	46476 → 5080 [ACK] Seq=5282606 Ack=26 Win=65536 Len=0 TSval=3201278603 TSecr=3201278603

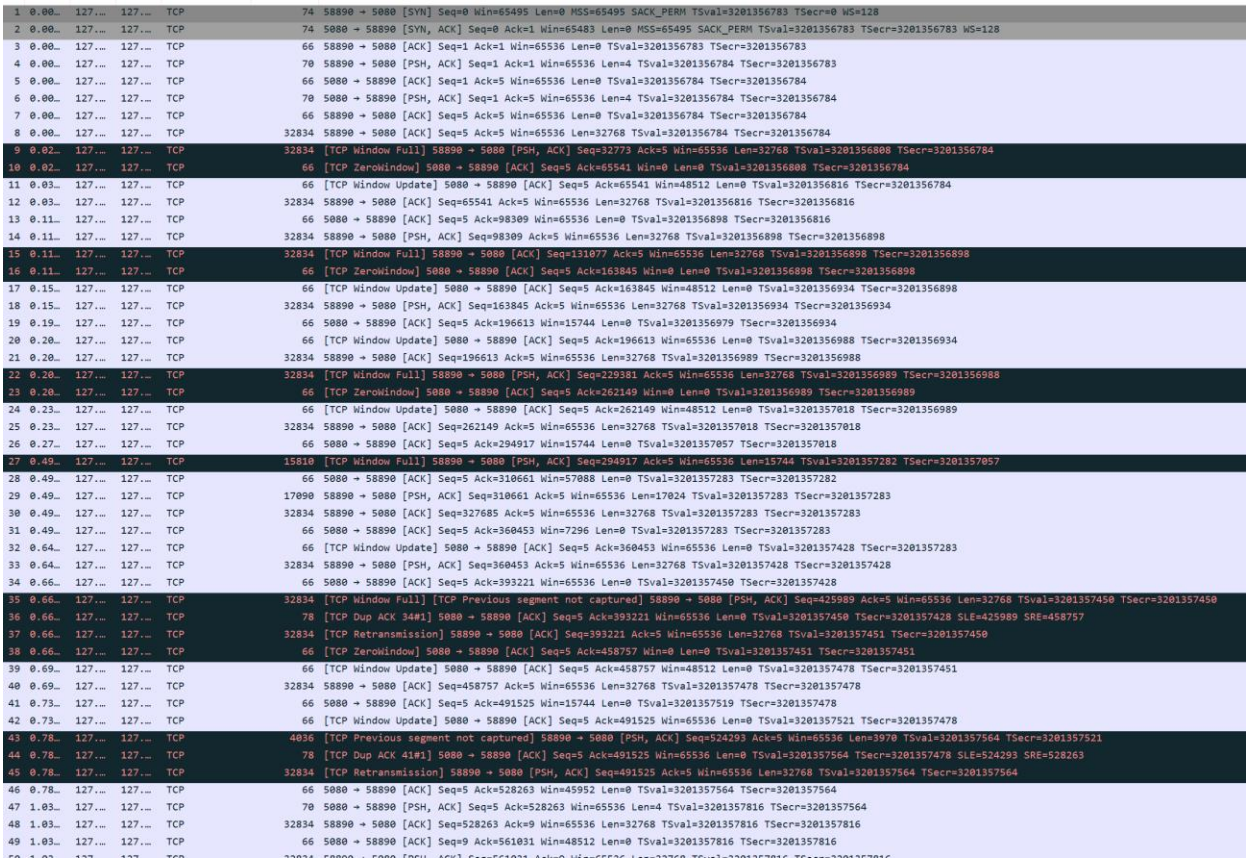
Here we can see one more problem that accrue:

TCP Fast Retransmission occurs when the sender retransmits a packet before the expiration of the acknowledgement timer

212	13.…	127…	127…	TCP	65549	[TCP Fast Retransmission] 46476 → 5080 [ACK] Seq=3828789 Ack=21 Win=65536 Len=65483 TSval=3201274428 TSecr=3201274428

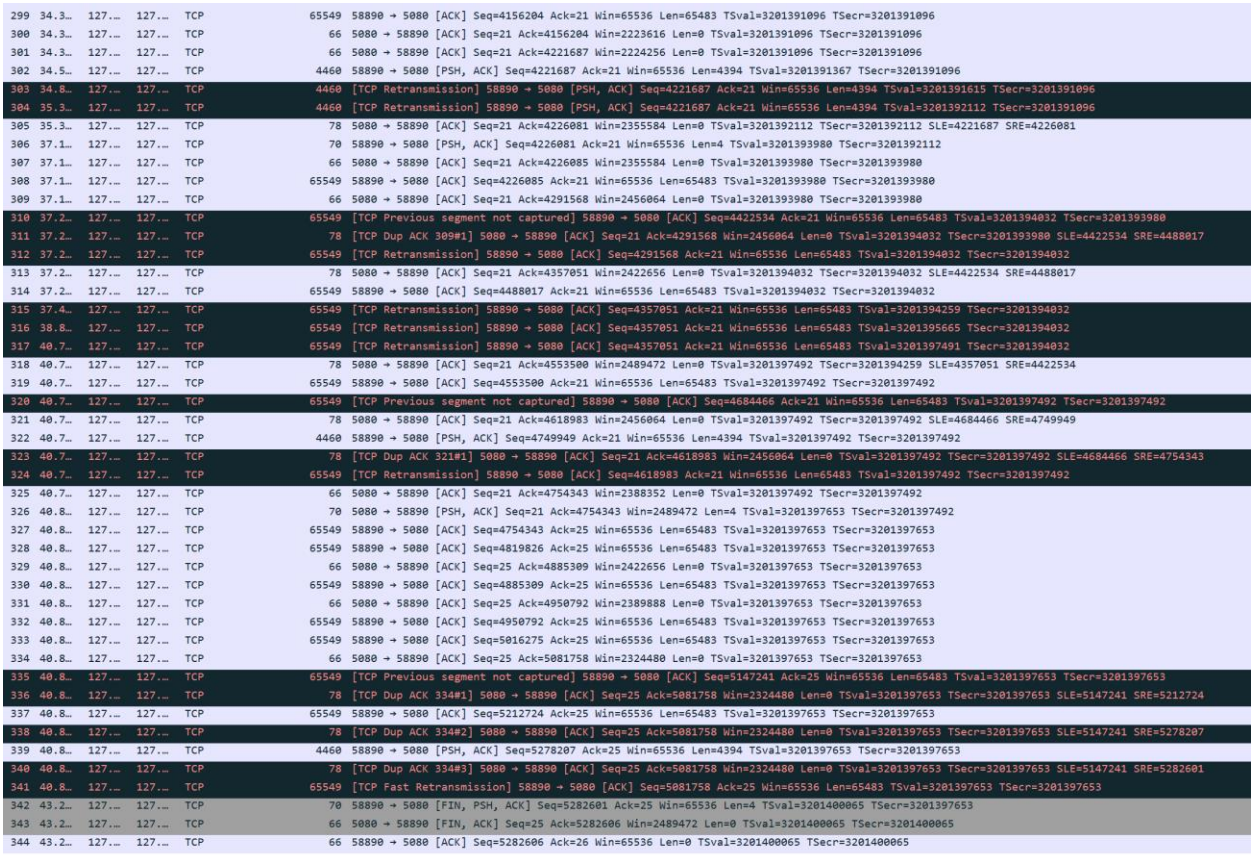
2.1.4 20% lost

open connection in lines 1 2



1	0.00…	127.…	127.…	TCP	74	58890 → 5080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3201356783 TSecr=0 WS=128
2	0.00…	127.…	127.…	TCP	74	5080 → 58890 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3201356783 TSecr=3201356783 WS=128
3	0.00…	127.…	127.…	TCP	66	58890 → 5080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3201356783 TSecr=3201356783
4	0.00…	127.…	127.…	TCP	70	58890 → 5080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=3201356784 TSecr=3201356783
5	0.00…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=3201356784 TSecr=3201356784
6	0.00…	127.…	127.…	TCP	70	5080 → 58890 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=4 TSval=3201356784 TSecr=3201356784
7	0.00…	127.…	127.…	TCP	66	58890 → 5080 [ACK] Seq=5 Ack=5 Win=65536 Len=0 TSval=3201356784 TSecr=3201356784
8	0.00…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=5 Ack=5 Win=65536 Len=32768 TSval=3201356784 TSecr=3201356784
9	0.02…	127.…	127.…	TCP	32834	[TCP Window Full] 58890 → 5080 [PSH, ACK] Seq=32773 Ack=5 Win=65536 Len=32768 TSval=3201356808 TSecr=3201356784
10	0.02…	127.…	127.…	TCP	66	[TCP ZeroWindow] 5080 → 58890 [ACK] Seq=5 Ack=65541 Win=0 Len=0 TSval=3201356808 TSecr=3201356784
11	0.03…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=65541 Win=48512 Len=0 TSval=3201356816 TSecr=3201356784
12	0.03…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=65541 Ack=5 Win=65536 Len=32768 TSval=3201356816 TSecr=3201356816
13	0.11…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=98309 Win=65536 Len=0 TSval=3201356898 TSecr=3201356816
14	0.11…	127.…	127.…	TCP	32834	58890 → 5080 [PSH, ACK] Seq=98309 Ack=5 Win=65536 Len=32768 TSval=3201356898 TSecr=3201356898
15	0.11…	127.…	127.…	TCP	32834	[TCP Window Full] 58890 → 5080 [ACK] Seq=131077 Ack=5 Win=65536 Len=32768 TSval=3201356898 TSecr=3201356898
16	0.11…	127.…	127.…	TCP	66	[TCP ZeroWindow] 5080 → 58890 [ACK] Seq=5 Ack=163845 Win=0 Len=0 TSval=3201356898 TSecr=3201356898
17	0.15…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=163845 Win=48512 Len=0 TSval=3201356934 TSecr=3201356898
18	0.15…	127.…	127.…	TCP	32834	58890 → 5080 [PSH, ACK] Seq=163845 Ack=5 Win=65536 Len=32768 TSval=3201356934 TSecr=3201356934
19	0.19…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=196613 Win=15744 Len=0 TSval=3201356979 TSecr=3201356934
20	0.20…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=196613 Win=65536 Len=0 TSval=3201356988 TSecr=3201356934
21	0.20…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=196613 Ack=5 Win=65536 Len=32768 TSval=3201356989 TSecr=3201356988
22	0.20…	127.…	127.…	TCP	32834	[TCP Window Full] 58890 → 5080 [PSH, ACK] Seq=229381 Ack=5 Win=65536 Len=32768 TSval=3201356989 TSecr=3201356988
23	0.20…	127.…	127.…	TCP	66	[TCP ZeroWindow] 5080 → 58890 [ACK] Seq=5 Ack=262149 Win=0 Len=0 TSval=3201356989 TSecr=3201356989
24	0.23…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=262149 Win=48512 Len=0 TSval=3201357018 TSecr=3201356989
25	0.23…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=262149 Ack=5 Win=65536 Len=32768 TSval=3201357018 TSecr=3201357018
26	0.27…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=294917 Win=15744 Len=0 TSval=3201357057 TSecr=3201357018
27	0.49…	127.…	127.…	TCP	15810	[TCP Window Full] 58890 → 5080 [PSH, ACK] Seq=294917 Ack=5 Win=65536 Len=15744 TSval=3201357282 TSecr=3201357057
28	0.49…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=310661 Win=57088 Len=0 TSval=3201357283 TSecr=3201357282
29	0.49…	127.…	127.…	TCP	17090	58890 → 5080 [PSH, ACK] Seq=310661 Ack=5 Win=65536 Len=17024 TSval=3201357283 TSecr=3201357283
30	0.49…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=327685 Ack=5 Win=65536 Len=32768 TSval=3201357283 TSecr=3201357283
31	0.49…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=360453 Win=7296 Len=0 TSval=3201357283 TSecr=3201357283
32	0.64…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=360453 Win=65536 Len=0 TSval=3201357428 TSecr=3201357283
33	0.64…	127.…	127.…	TCP	32834	58890 → 5080 [PSH, ACK] Seq=360453 Ack=5 Win=65536 Len=32768 TSval=3201357428 TSecr=3201357428
34	0.66…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=393221 Win=65536 Len=0 TSval=3201357450 TSecr=3201357428
35	0.66…	127.…	127.…	TCP	32834	[TCP Window Full] [TCP Previous segment not captured] 58890 → 5080 [PSH, ACK] Seq=425989 Ack=5 Win=65536 Len=32768 TSval=3201357450 TSecr=3201357450
36	0.66…	127.…	127.…	TCP	78	[TCP Dup ACK 34#1] 5080 → 58890 [ACK] Seq=5 Ack=393221 Win=65536 Len=0 TSval=3201357450 TSecr=3201357428 SLE=425989 SRE=458757
37	0.66…	127.…	127.…	TCP	32834	[TCP Retransmission] 58890 → 5080 [ACK] Seq=393221 Ack=5 Win=65536 Len=32768 TSval=3201357451 TSecr=3201357450
38	0.66…	127.…	127.…	TCP	66	[TCP ZeroWindow] 5080 → 58890 [ACK] Seq=5 Ack=458757 Win=0 Len=0 TSval=3201357451 TSecr=3201357451
39	0.69…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=458757 Win=48512 Len=0 TSval=3201357478 TSecr=3201357451
40	0.69…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=458757 Ack=5 Win=65536 Len=32768 TSval=3201357478 TSecr=3201357478
41	0.73…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=491525 Win=15744 Len=0 TSval=3201357519 TSecr=3201357478
42	0.73…	127.…	127.…	TCP	66	[TCP Window Update] 5080 → 58890 [ACK] Seq=5 Ack=491525 Win=65536 Len=0 TSval=3201357521 TSecr=3201357478
43	0.78…	127.…	127.…	TCP	4036	[TCP Previous segment not captured] 58890 → 5080 [PSH, ACK] Seq=524293 Ack=5 Win=65536 Len=3970 TSval=3201357564 TSecr=3201357521
44	0.78…	127.…	127.…	TCP	78	[TCP Dup ACK 41#1] 5080 → 58890 [ACK] Seq=5 Ack=491525 Win=65536 Len=0 TSval=3201357564 TSecr=3201357478 SLE=524293 SRE=528263
45	0.78…	127.…	127.…	TCP	32834	[TCP Retransmission] 58890 → 5080 [PSH, ACK] Seq=491525 Ack=5 Win=65536 Len=32768 TSval=3201357564 TSecr=3201357564
46	0.78…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=5 Ack=528263 Win=45952 Len=0 TSval=3201357564 TSecr=3201357564
47	1.03…	127.…	127.…	TCP	70	5080 → 58890 [PSH, ACK] Seq=5 Ack=528263 Win=65536 Len=4 TSval=3201357816 TSecr=3201357564
48	1.03…	127.…	127.…	TCP	32834	58890 → 5080 [ACK] Seq=528263 Ack=9 Win=65536 Len=32768 TSval=3201357816 TSecr=3201357816
49	1.03…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=9 Ack=561031 Win=48512 Len=0 TSval=3201357816 TSecr=3201357816

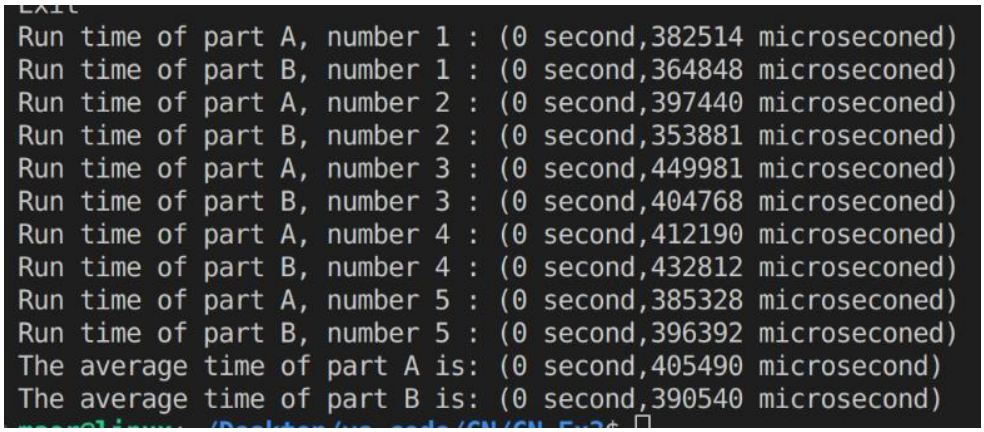
Close the connection in lines 342, 343, 344



299	34.3…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4156204 Ack=21 Win=65536 Len=65483 TSval=3201391096 TSecr=3201391096
300	34.3…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=21 Ack=4156204 Win=2223616 Len=0 TSval=3201391096 TSecr=3201391096
301	34.3…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=21 Ack=4221687 Win=2224256 Len=0 TSval=3201391096 TSecr=3201391096
302	34.5…	127.…	127.…	TCP	4460	58890 → 5080 [PSH, ACK] Seq=4221687 Ack=21 Win=65536 Len=4394 TSval=3201391367 TSecr=3201391096
303	34.8…	127.…	127.…	TCP	4460	[TCP Retransmission] 58890 → 5080 [PSH, ACK] Seq=4221687 Ack=21 Win=65536 Len=4394 TSval=3201391615 TSecr=3201391096
304	35.3…	127.…	127.…	TCP	4460	[TCP Retransmission] 58890 → 5080 [PSH, ACK] Seq=4221687 Ack=21 Win=65536 Len=4394 TSval=3201392112 TSecr=3201391096
305	35.3…	127.…	127.…	TCP	78	5080 → 58890 [ACK] Seq=21 Ack=4226081 Win=2355584 Len=0 TSval=3201392112 TSecr=3201392112 SLE=4221687 SRE=4226081
306	37.1…	127.…	127.…	TCP	70	58890 → 5080 [PSH, ACK] Seq=4226081 Ack=21 Win=65536 Len=4 TSval=3201393980 TSecr=3201392112
307	37.1…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=21 Ack=4226085 Win=2355584 Len=0 TSval=3201393980 TSecr=3201393980
308	37.1…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4226085 Ack=21 Win=65536 Len=65483 TSval=3201393980 TSecr=3201393980
309	37.1…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=21 Ack=4291568 Win=2456064 Len=0 TSval=3201393980 TSecr=3201393980
310	37.2…	127.…	127.…	TCP	65549	[TCP Previous segment not captured] 58890 → 5080 [ACK] Seq=4422534 Ack=21 Win=65536 Len=65483 TSval=3201394032 TSecr=3201393980
311	37.2…	127.…	127.…	TCP	78	[TCP Dup ACK 309#1] 5080 → 58890 [ACK] Seq=21 Ack=4291568 Win=2456064 Len=0 TSval=3201394032 TSecr=3201393980 SLE=4422534 SRE=4488017
312	37.2…	127.…	127.…	TCP	65549	[TCP Retransmission] 58890 → 5080 [ACK] Seq=4291568 Ack=21 Win=65536 Len=65483 TSval=3201394032 TSecr=3201394032
313	37.2…	127.…	127.…	TCP	78	5080 → 58890 [ACK] Seq=21 Ack=4357051 Win=2422656 Len=0 TSval=3201394032 TSecr=3201394032 SLE=4422534 SRE=4488017
314	37.2…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4488017 Ack=21 Win=65536 Len=65483 TSval=3201394032 TSecr=3201394032
315	37.4…	127.…	127.…	TCP	65549	[TCP Retransmission] 58890 → 5080 [ACK] Seq=4357051 Ack=21 Win=65536 Len=65483 TSval=3201394259 TSecr=3201394032
316	38.8…	127.…	127.…	TCP	65549	[TCP Retransmission] 58890 → 5080 [ACK] Seq=4357051 Ack=21 Win=65536 Len=65483 TSval=3201395665 TSecr=3201394032
317	40.7…	127.…	127.…	TCP	65549	[TCP Retransmission] 58890 → 5080 [ACK] Seq=4357051 Ack=21 Win=65536 Len=65483 TSval=3201397491 TSecr=3201394032
318	40.7…	127.…	127.…	TCP	78	5080 → 58890 [ACK] Seq=21 Ack=4553500 Win=2489472 Len=0 TSval=3201397492 TSecr=3201394259 SLE=4357051 SRE=4422534
319	40.7…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4553500 Ack=21 Win=65536 Len=65483 TSval=3201397492 TSecr=3201397492
320	40.7…	127.…	127.…	TCP	65549	[TCP Previous segment not captured] 58890 → 5080 [ACK] Seq=4684466 Ack=21 Win=65536 Len=65483 TSval=3201397492 TSecr=3201397492
321	40.7…	127.…	127.…	TCP	78	5080 → 58890 [ACK] Seq=21 Ack=4618983 Win=2456064 Len=0 TSval=3201397492 TSecr=3201397492 SLE=4684466 SRE=4749949
322	40.7…	127.…	127.…	TCP	4460	58890 → 5080 [PSH, ACK] Seq=4749949 Ack=21 Win=65536 Len=4394 TSval=3201397492 TSecr=3201397492
323	40.7…	127.…	127.…	TCP	78	[TCP Dup ACK 321#1] 5080 → 58890 [ACK] Seq=21 Ack=4618983 Win=2456064 Len=0 TSval=3201397492 TSecr=3201397492 SLE=4684466 SRE=4754343
324	40.7…	127.…	127.…	TCP	65549	[TCP Retransmission] 58890 → 5080 [ACK] Seq=4618983 Ack=21 Win=65536 Len=65483 TSval=3201397492 TSecr=3201397492
325	40.7…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=21 Ack=4754343 Win=2388352 Len=0 TSval=3201397492 TSecr=3201397492
326	40.8…	127.…	127.…	TCP	70	5080 → 58890 [PSH, ACK] Seq=21 Ack=4754343 Win=2489472 Len=4 TSval=3201397653 TSecr=3201397492
327	40.8…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4754343 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
328	40.8…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4819826 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
329	40.8…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=25 Ack=4885309 Win=2422656 Len=0 TSval=3201397653 TSecr=3201397653
330	40.8…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4885309 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
331	40.8…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=25 Ack=4950792 Win=2389888 Len=0 TSval=3201397653 TSecr=3201397653
332	40.8…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=4950792 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
333	40.8…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=5016275 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
334	40.8…	127.…	127.…	TCP	66	5080 → 58890 [ACK] Seq=25 Ack=5081758 Win=2324480 Len=0 TSval=3201397653 TSecr=3201397653
335	40.8…	127.…	127.…	TCP	65549	[TCP Previous segment not captured] 58890 → 5080 [ACK] Seq=5147241 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
336	40.8…	127.…	127.…	TCP	78	[TCP Dup ACK 334#1] 5080 → 58890 [ACK] Seq=25 Ack=5081758 Win=2324480 Len=0 TSval=3201397653 TSecr=3201397653 SLE=5147241 SRE=5212724
337	40.8…	127.…	127.…	TCP	65549	58890 → 5080 [ACK] Seq=5212724 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
338	40.8…	127.…	127.…	TCP	78	[TCP Dup ACK 334#2] 5080 → 58890 [ACK] Seq=25 Ack=5081758 Win=2324480 Len=0 TSval=3201397653 TSecr=3201397653 SLE=5147241 SRE=5278207
339	40.8…	127.…	127.…	TCP	4460	58890 → 5080 [PSH, ACK] Seq=5278207 Ack=25 Win=65536 Len=4394 TSval=3201397653 TSecr=3201397653
340	40.8…	127.…	127.…	TCP	78	[TCP Dup ACK 334#3] 5080 → 58890 [ACK] Seq=25 Ack=5081758 Win=2324480 Len=0 TSval=3201397653 TSecr=3201397653 SLE=5147241 SRE=5282601
341	40.8…	127.…	127.…	TCP	65549	[TCP Fast Retransmission] 58890 → 5080 [ACK] Seq=5081758 Ack=25 Win=65536 Len=65483 TSval=3201397653 TSecr=3201397653
342	43.2…	127.…	127.…	TCP	70	58890 → 5080 [FIN, PSH, ACK] Seq=5282601 Ack=25 Win=65536 Len=4 TSval=3201400065 TSecr=3201397653
343	43.2…	127.…	127.…	TCP	66	5080 → 58890 [FIN, ACK] Seq=25 Ack=5282606 Win=2489472 Len=0 TSval=3201400065 TSecr=3201400065
344	43.2…	127.…	127.…	TCP	66	58890 → 5080 [ACK] Seq=5282606 Ack=26 Win=65536 Len=0 TSval=3201400065 TSecr=3201400065

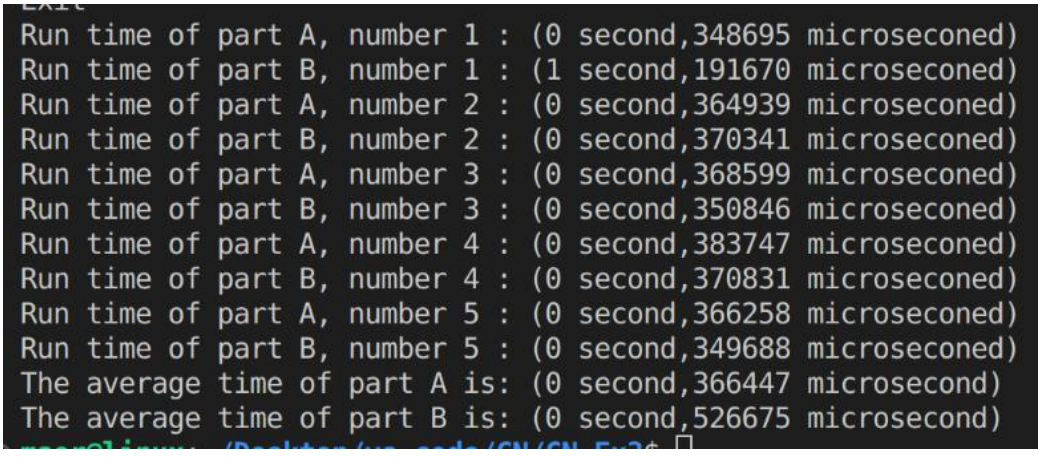
2.2 Average sending times comparisons

2.2.1 0% lost



we can see in this picture that the average time of sending part A according to CUBIC algorithm and the average time of sending part B according to RENO algorithm is very tight because we have 0% of packets, so the difference is minor.

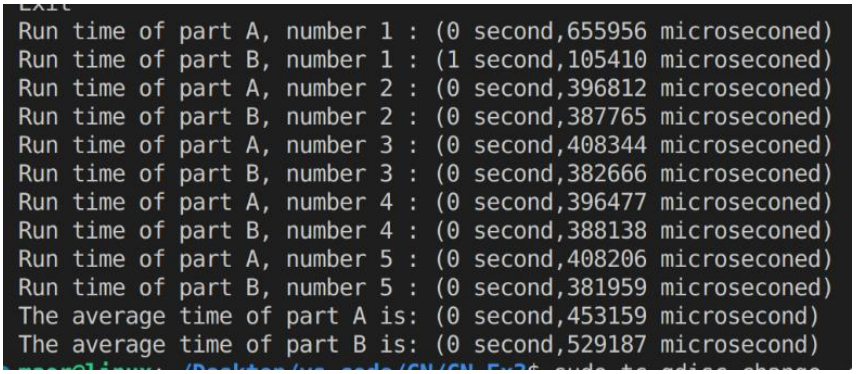
2.2.2 10% lost



we can see in this picture that the average time of sending part A according to CUBIC algorithm is smaller than average time of sending part B according to RENO algorithm. Now we have 10% packets lost

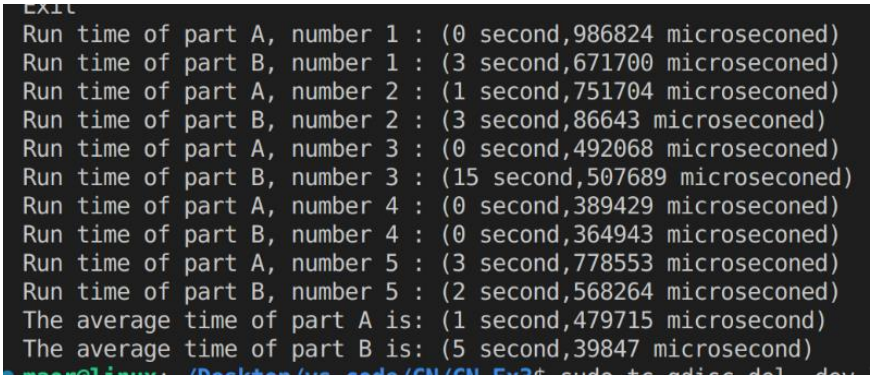
therefore the gap is bigger than in 0% packets lost.

2.2.3 15% lost



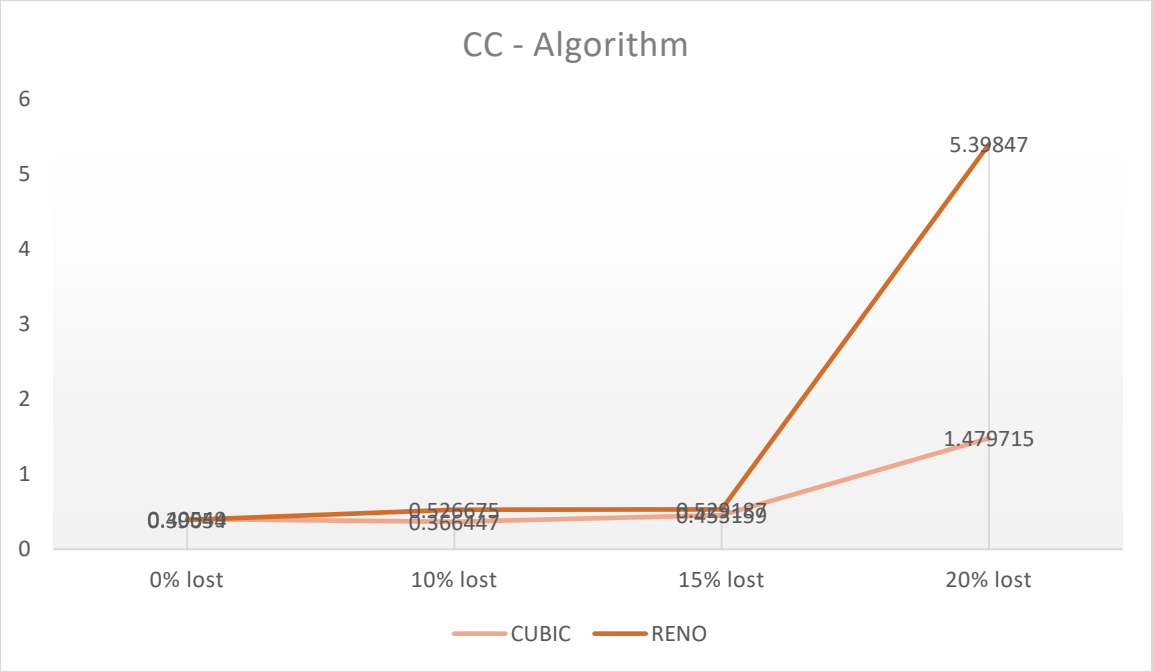
We can see the difference of the average time

2.2.4 20% lost



We can see the difference of the average time

2.3 Diagram


CC - Algorithm
6
5
4
3
2
1
0
5.39847
1.479715
0.526675
0.366447
0% lost
10% lost
15% lost
20% lost
CUBIC
RENO

2.4 Conclusions

We can understand from the previous pages that sending the first half of the file according to CUBIC CC – Algorithm is faster than sending the second half of the file according to RENO CC – Algorithm.

This can be seen in all five sending of the file, and this is the reason we send five times -> to be assured that our results are reliable.

According to the diagram: the more packets we have that are lost, then the gap between the times of sending first half to sending the second half increases. Also, we can see that the more packets we have that are lost, then the time to send the file is getting longer.

In Wireshark's recording we can see the difference between the 0% lost to 10% lost to 15% lost and to 20% lost. The more losses we have the more 'black packets' appear in Wireshark.

In addition, it is possible to see in Wireshark's recording the principles that we learned in class about the TCP algorithm actualize, such as:

Increasing the receiver's window when it is full and decreasing the window when it reaches the thresholdץ

And we can see in the receiver's ACK messages, the principle of Cumulativeness, which means that the receiver sends the last ACK that he received, which contains all the ACK before it.

2.5 Bibliography

Cubic and Reno CC – Algorithm
https://squidarth.com/rc/programming/networking/2018/08/01/congestion-cubic.html

How to calculate time
https://www.youtube.com/watch?v=cunJcNgtxMk&feature=youtu.be

About the functions
'man' command on vs terminal

For many things

ChatGPT: Optimizing Language Models for Dialogue (openai.com)

https://www.google.com/search?gs_ssp=eJzj4tTP1TcwMU02T1JgNGB0YPBiS8_PT89JBQBASQXT&q=google&oq=googlr&aqs=chrome.1.69i57j46i10i131i199i433i465i512j0i10i131i433i512l4j69i60j69i65.3825j0j4&sourceid=chrome&ie=UTF-8