Module number: CS3102
Matriculation number: 200017941

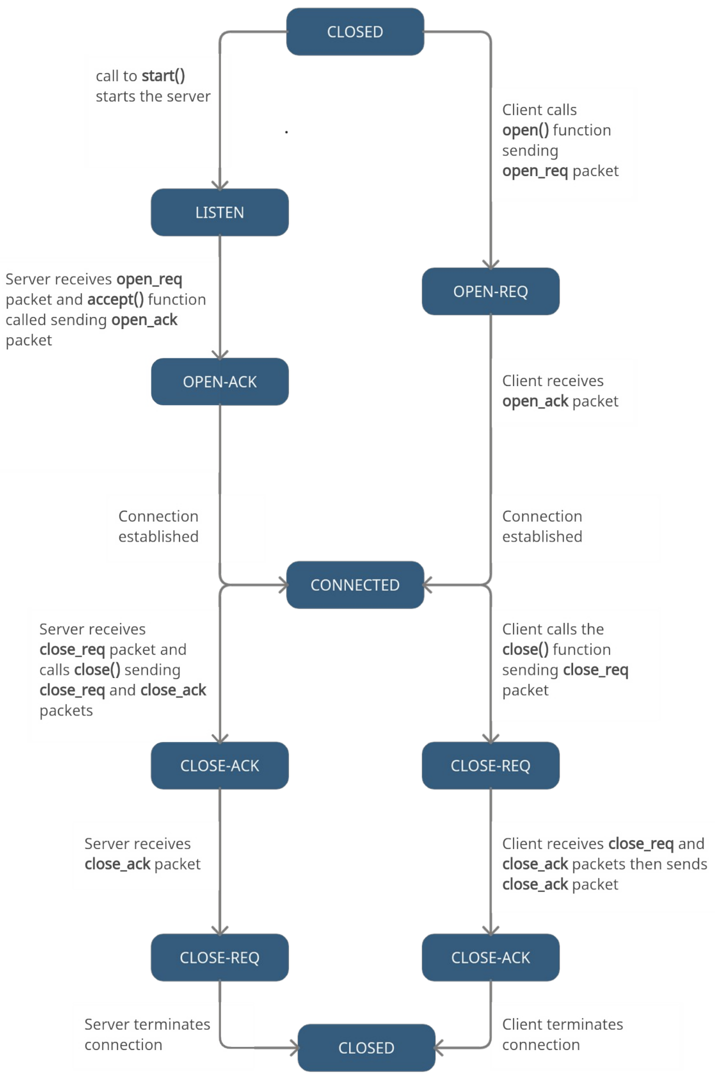
P2 Report

Overview

The objective of this practical was to design and implement a TCP protocol which is built on top of UDP. By doing so reliable and ordered delivery should be ensured. The coursework is split into three parts. I have completed the first two parts but didn't have the time do attempt the third, therefore this report will only be focused on those parts. For part one I have managed to create a FSM to demonstrate the overall operation of the protocol. Alongside this, I created a diagram to show what my design of the protocol looks like. For part two I have managed to make sure the client and server can establish a connection using the handshake protocol. I have also made sure that both end-points can communicate with each other by sending and receiving data. Unfortunately, due to time constraints again I wasn't able to implement the connection termination part.

Design

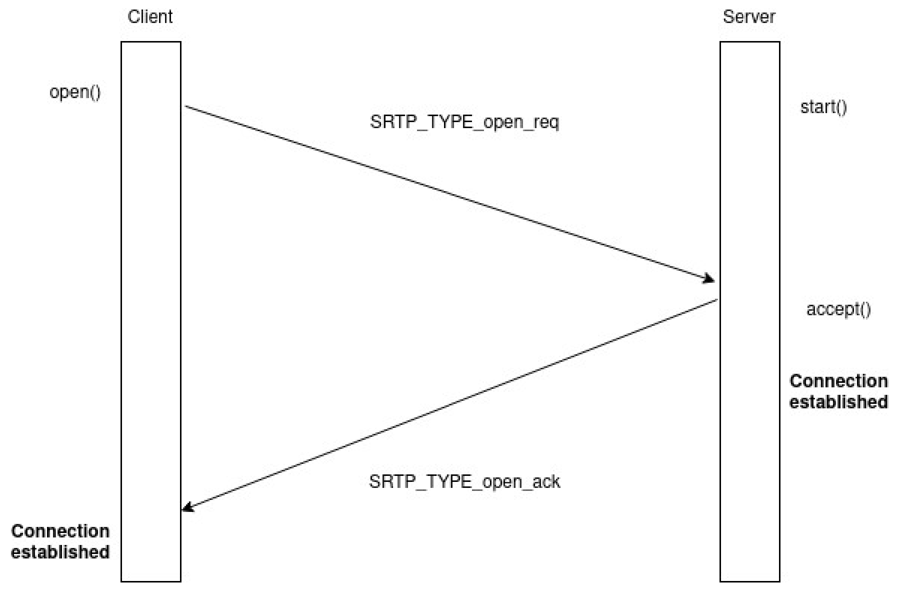
The diagram below shows how the communication is supposed to work when it comes to establishing and terminating a connection:



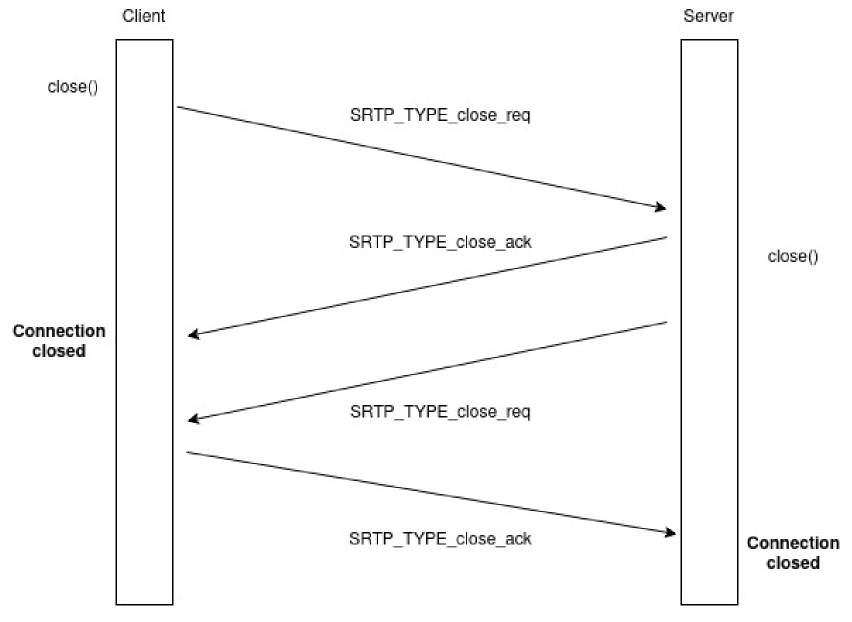
CLOSED
call to start() starts the server
Client calls open() function sending open_req packet
LISTEN
Server receives open_req packet and accept() function called sending open_ack packet
OPEN-REQ
OPEN-ACK
Client receives open_ack packet
Connection established
Connection established
CONNECTED
Server receives close_req packet and calls close() sending close_req and close_ack packets
Client calls the close() function sending close_req packet
CLOSE-ACK
CLOSE-REQ
Server receives close_ack packet
Client receives close_req and close_ack packets then sends close_ack packet
CLOSE-REQ
CLOSE-ACK
Server terminates connection
CLOSED
Client terminates connection

Module number: CS3102
Matriculation number: 200017941

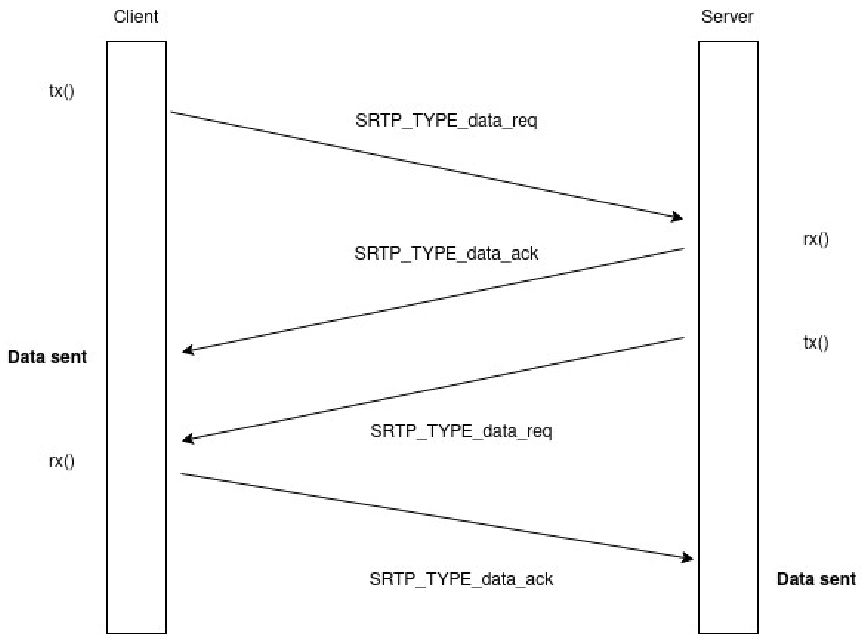
First the server is set up so that it can listen to requests to connect using the start() function. Then the client is set up using start() which sends a SRTP_TYPE_open_req packet to the server requesting to connect. The server receives this request and sends an acknowledgement packet of type SRTP_TYPE_open_ack using the accept() function. This establishes a connection. The protocol exchange message for this part is shown below:



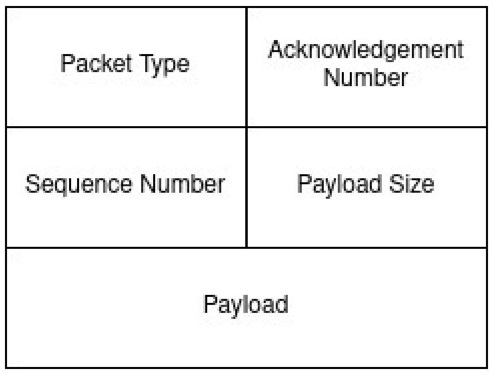
Then when it comes to terminating a connection, the client first sends a request to terminate by calling close(), sending a SRTP_TYPE_close_req packet. The server receives this and sends an acknowledgement packet of type SRTP_TYPE_close_ack packet and the client closes, but still listens. The same series of events happens from the server allowing both end-points to be closed. The protocol exchange message for this part is shown below:



Lastly, both end-points should be able to send and receive data to and from each other which is done using the tx() and rx() functions. The relevant packet types for this functionality are SRTP_TYPE_data_req and SRTP_TYPE_data_req. The protocol exchange message for this part is shown below:



When sending and receiving packets, my implementation of the protocol has the following design:



The ‘Packet Type’ field stores the type of packet being sent, whether it is a request or an acknowledgement, allowing each end-point to send a response packet accordingly. The ‘Acknowledgement Number’ and ‘Sequence Number’ fields are both used to ensure that ordered and reliable transfer of data. Both of the numbers are randomly assigned on from the client-side when open() is used to send a request to connect. The server then takes the sequence number of that packet and adds one to it and assigns it as the server’s acknowledgement number. This is then checked at the client to see if the appropriate packet is sent by the server. The ‘Payload Size’ field simply has the size of the payload and the payload itself is in the ‘Payload’ field.

Module number: CS3102
Matriculation number: 200017941

Implementation

For my implementation, the other files have remained untouched, except for the srtp-packet.h file where I modified the Srtp_Header_t so that it follows my design of the protocol. The srtp_initialise() function was untouched for the most part as I found it unnecessary to implement. The srtp_start() function simply initiates the server by initialising the fields in the Srtp_Pcd struct as well as calling socket() and using bind() which is straightforward.

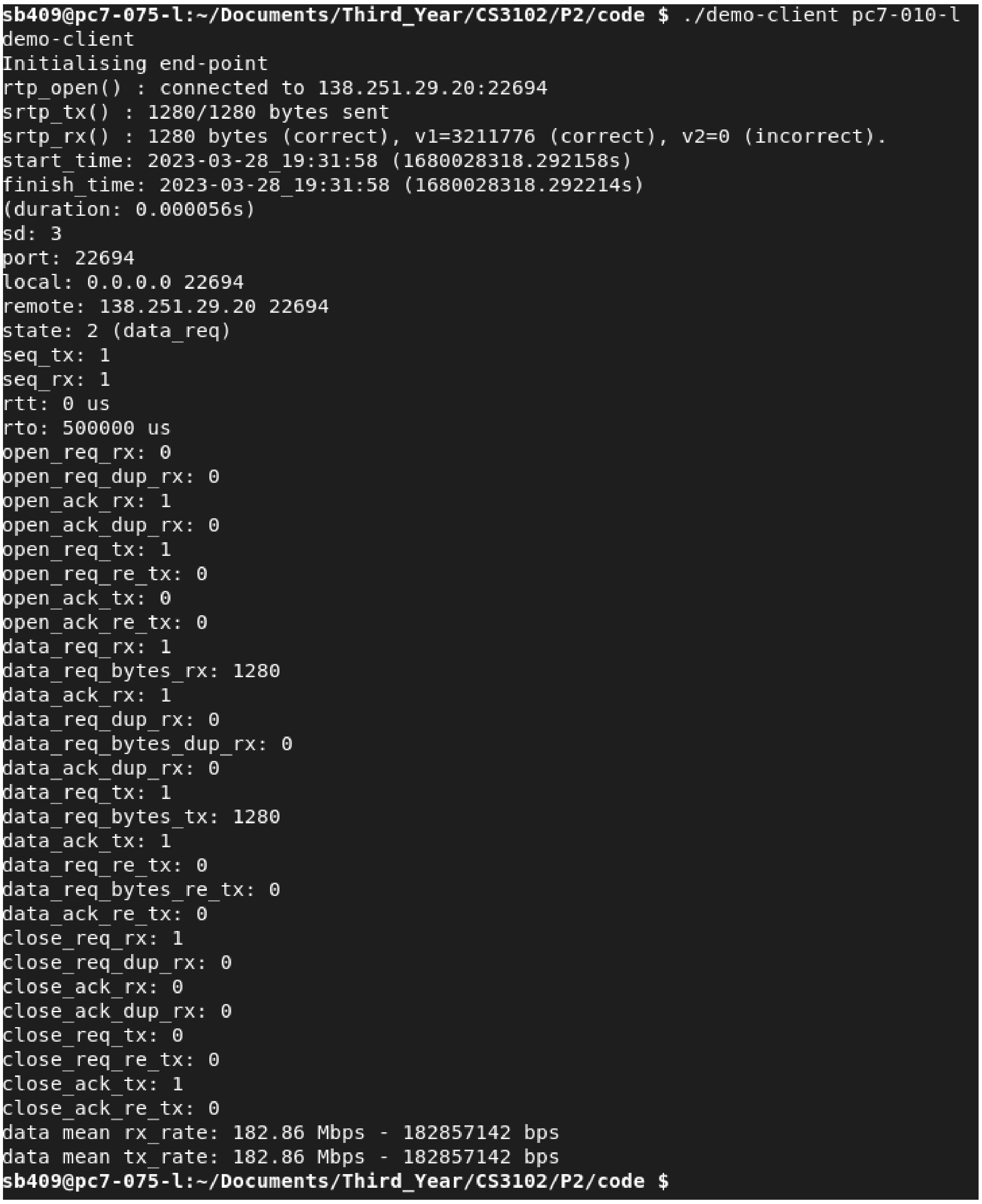
The srtp_open() function on the other hand has a lot more going on. There are three important functionality this function does which allows for the implementation to work. The first is serialising the packet struct so that it can be sent as a byte stream. The way I approached it is by first setting up the header and payload structs individually. Then using memcpy() I individually copied the structs into the byte stream variable. Additionally, it was important to memset() the byte stream at the beginning so that there aren't any unwanted values and for the pointer to where each struct start and end in the stream so that both can be added without any overlaps or gaps. The second functionality is for the client to wait for a acknowledgement packet, and to re-transmit the request packet if it isn't received before the RTO. I decided not to use an event handler such as SIGALARM because that would require additional functions to be defined and declared. So instead of that, I made use of select() which takes a file descriptor (which in this case is a socket descriptor) and checks if it there is any activity that has occurred allowing it to be read. One reason why this choice works is because it takes a timer value which basically acts as an interval which states when the file descriptor is checked for any activity. So all I had to do was set this to 500ms by using timeval struct. Additionally, a for loop wraps around this code so that it loop up to four times before the re-transmission stops. The last functionality is that the function takes the response packet and deserializes it allowing the fields in the packet to be accessed as a struct. This is done in a similar way to when the request packet was serialised. The header and payload were accessed individually again because I was dealing with a nested struct.

For the srtp_accept() function, it does the same things as srtp_open() except in the reverse order since it is receiving order. Also here there is no need for the sever to wait for a response so select() wasn't used. As of the srtp_tx() and srtp_rx() functions, these are mostly a copy and paste from the srtp_open() and srtp_accept() functions since it simply needs to send and receive data that is given to it, so the payload is handled slightly differently so that it sends the given data and that's about it. The close() function wasn't implemented as said earlier.

Testing

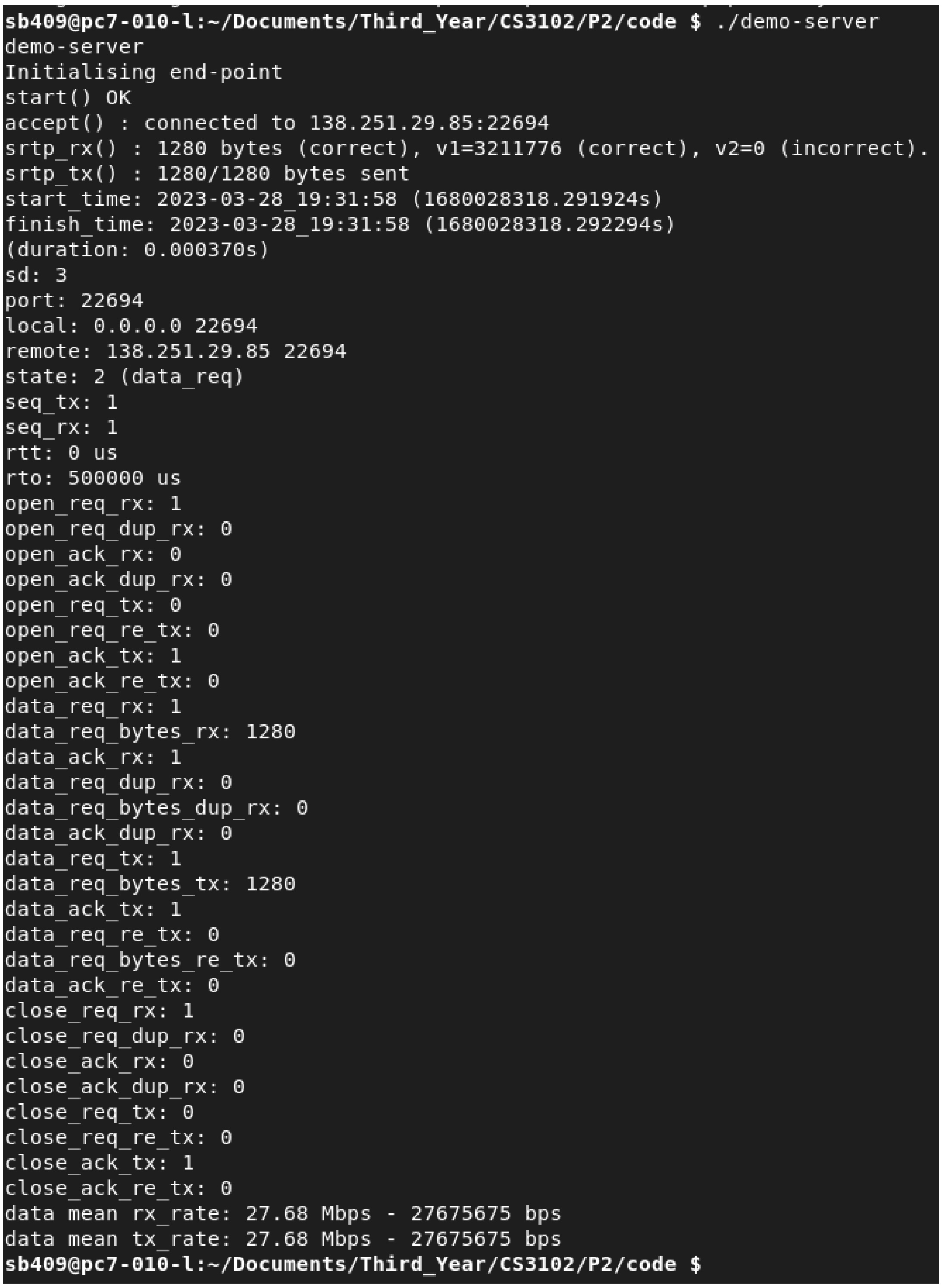
When it came to testing my implementation I mainly used the test files provided to verify that my program works as intended. During the development however, the tests involved placing print statements in the program to identify where the error has occurred and what the error is based on the output of print. Below are the output of from running my final tests using the provided files:

- demo-client.c:



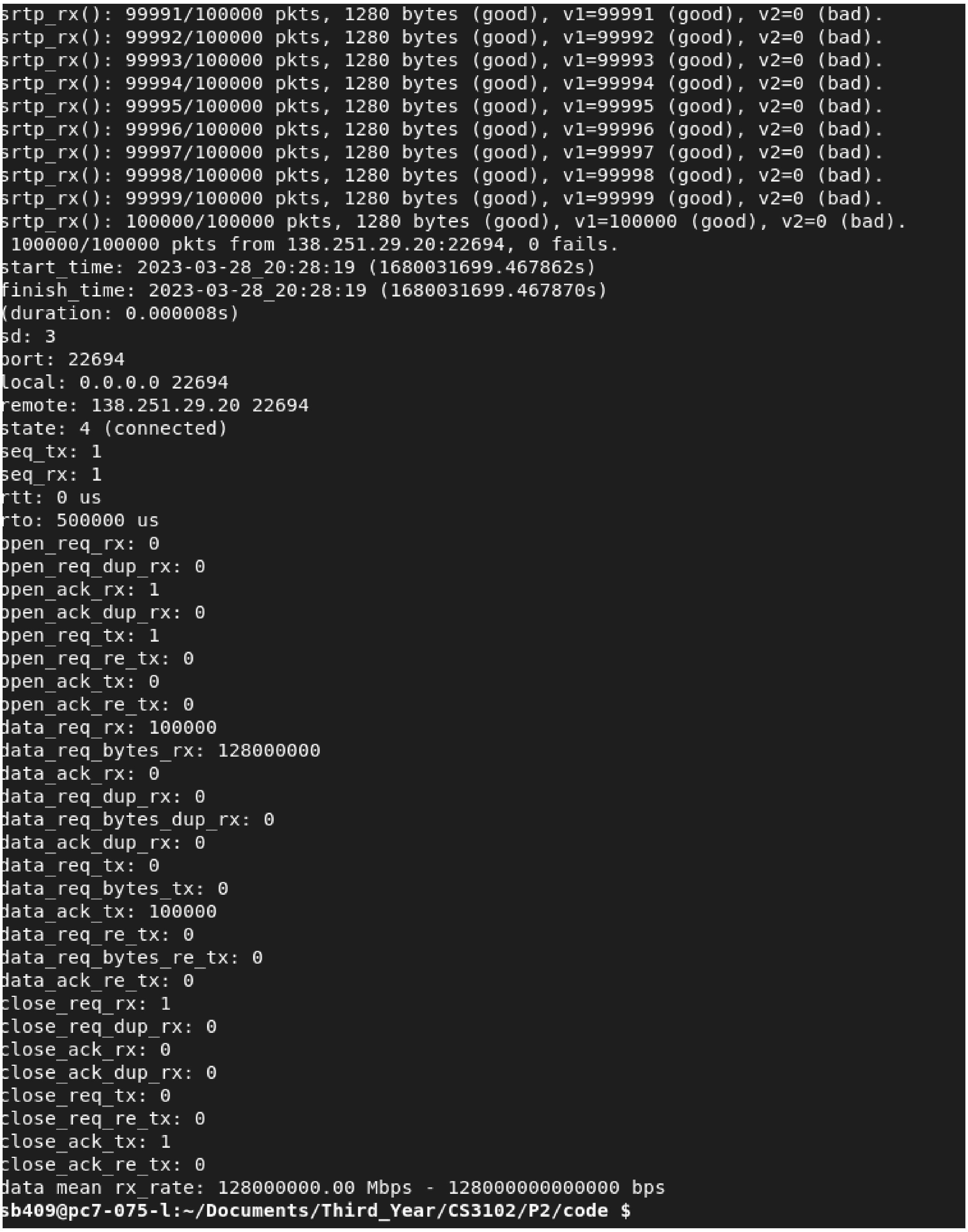
```
sb409@pc7-075-l:~/Documents/Third_Year/CS3102/P2/code $ ./demo-client pc7-010-l
demo-client
Initialising end-point
rtp_open() : connected to 138.251.29.20:22694
srtp_tx() : 1280/1280 bytes sent
srtp_rx() : 1280 bytes (correct), v1=3211776 (correct), v2=0 (incorrect).
start_time: 2023-03-28_19:31:58 (1680028318.292158s)
finish_time: 2023-03-28_19:31:58 (1680028318.292214s)
(duration: 0.000056s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.20 22694
state: 2 (data_req)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 0
open_req_dup_rx: 0
open_ack_rx: 1
open_ack_dup_rx: 0
open_req_tx: 1
open_req_re_tx: 0
open_ack_tx: 0
open_ack_re_tx: 0
data_req_rx: 1
data_req_bytes_rx: 1280
data_ack_rx: 1
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 1
data_req_bytes_tx: 1280
data_ack_tx: 1
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean rx_rate: 182.86 Mbps - 182857142 bps
data mean tx_rate: 182.86 Mbps - 182857142 bps
sb409@pc7-075-l:~/Documents/Third_Year/CS3102/P2/code $
```

- demo-server.c:



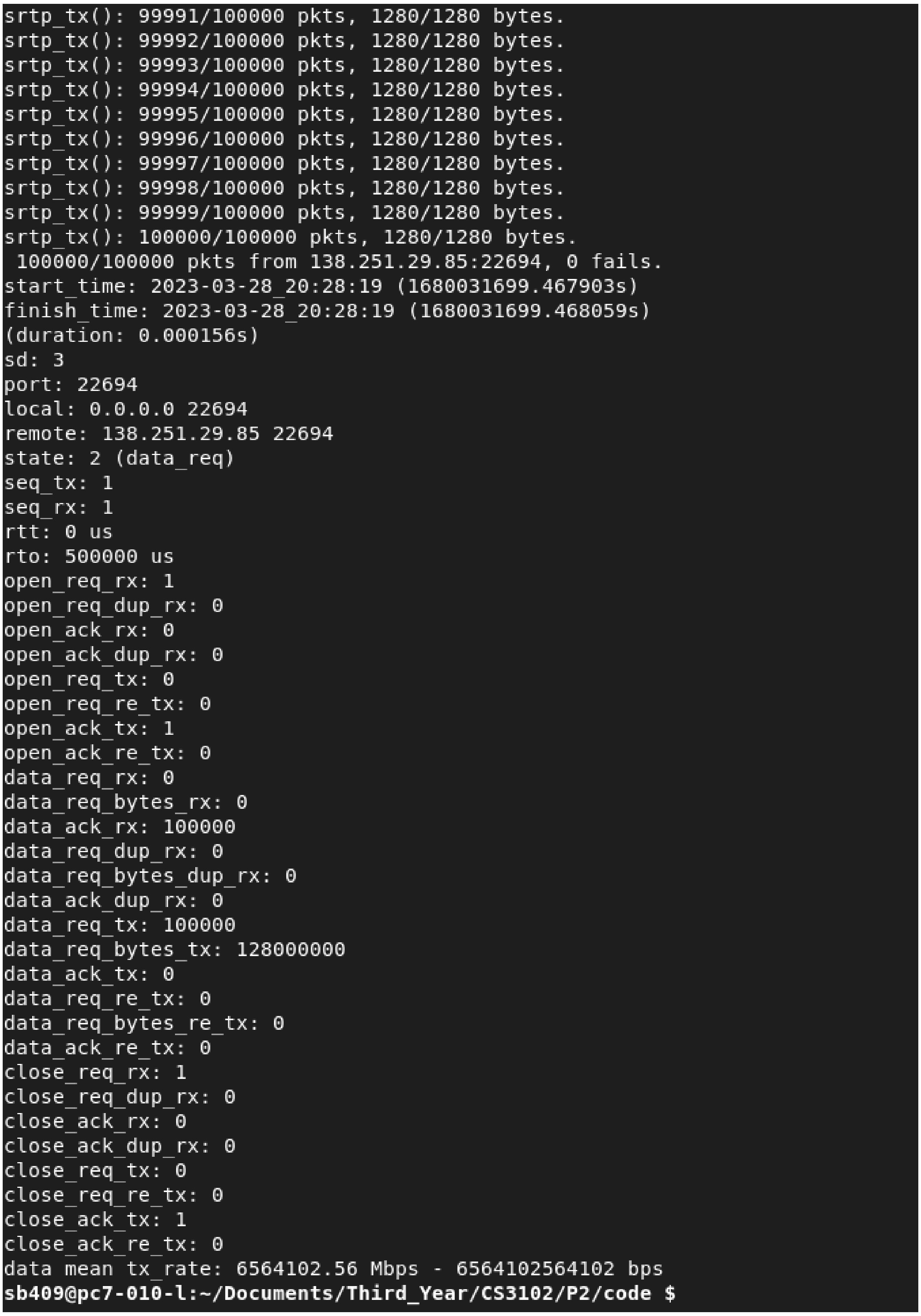
```
sb409@pc7-010-l:~/Documents/Third_Year/CS3102/P2/code $ ./demo-server
demo-server
Initialising end-point
start() OK
accept() : connected to 138.251.29.85:22694
srtp_rx() : 1280 bytes (correct), v1=3211776 (correct), v2=0 (incorrect).
srtp_tx() : 1280/1280 bytes sent
start_time: 2023-03-28_19:31:58 (1680028318.291924s)
finish_time: 2023-03-28_19:31:58 (1680028318.292294s)
(duration: 0.000370s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.85 22694
state: 2 (data_req)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 1
open_req_dup_rx: 0
open_ack_rx: 0
open_ack_dup_rx: 0
open_req_tx: 0
open_req_re_tx: 0
open_ack_tx: 1
open_ack_re_tx: 0
data_req_rx: 1
data_req_bytes_rx: 1280
data_ack_rx: 1
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 1
data_req_bytes_tx: 1280
data_ack_tx: 1
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean rx_rate: 27.68 Mbps - 27675675 bps
data mean tx_rate: 27.68 Mbps - 27675675 bps
sb409@pc7-010-l:~/Documents/Third_Year/CS3102/P2/code $
```

- test-client-1.c:



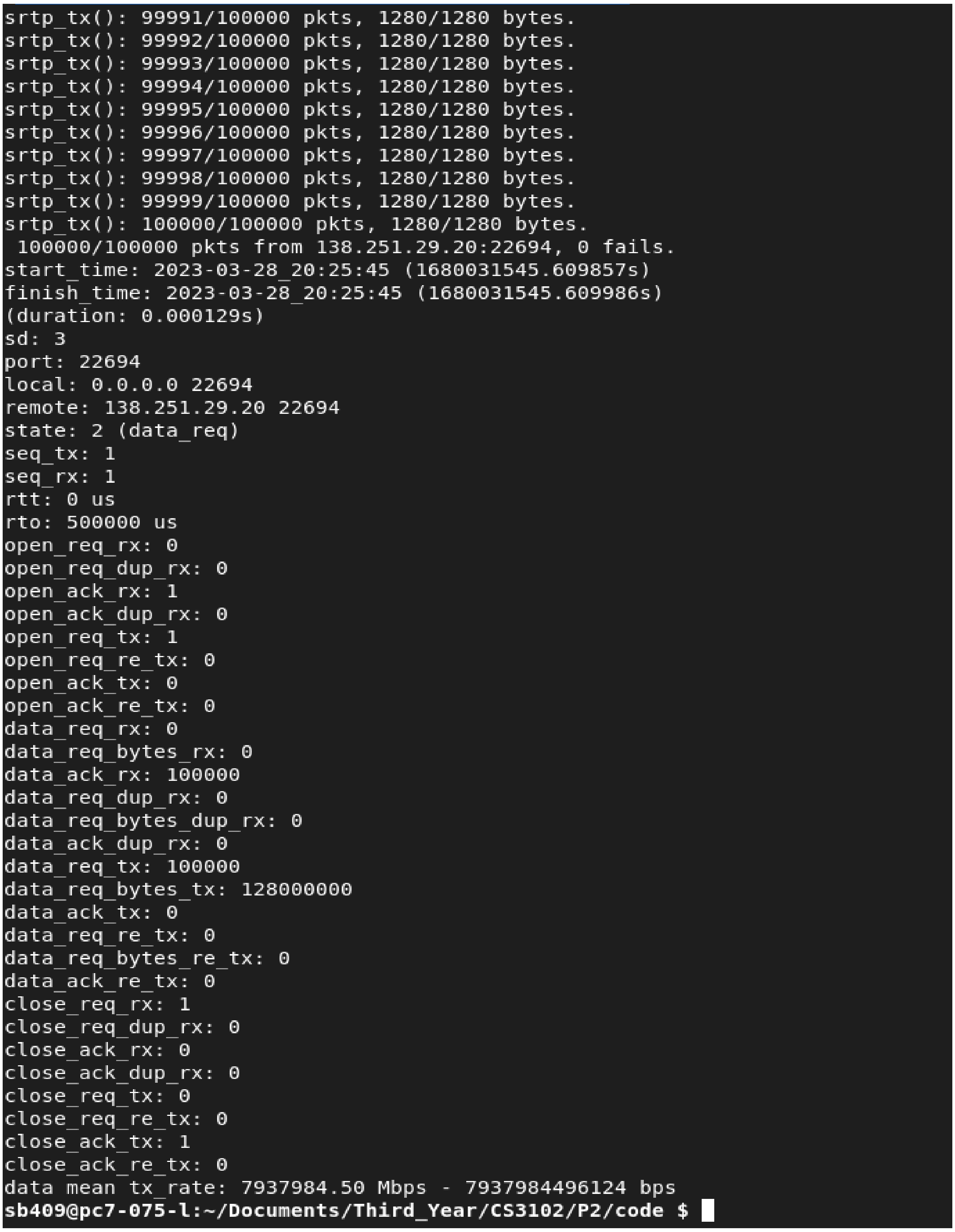
```
srtp_rx(): 99991/100000 pkts, 1280 bytes (good), v1=99991 (good), v2=0 (bad).
srtp_rx(): 99992/100000 pkts, 1280 bytes (good), v1=99992 (good), v2=0 (bad).
srtp_rx(): 99993/100000 pkts, 1280 bytes (good), v1=99993 (good), v2=0 (bad).
srtp_rx(): 99994/100000 pkts, 1280 bytes (good), v1=99994 (good), v2=0 (bad).
srtp_rx(): 99995/100000 pkts, 1280 bytes (good), v1=99995 (good), v2=0 (bad).
srtp_rx(): 99996/100000 pkts, 1280 bytes (good), v1=99996 (good), v2=0 (bad).
srtp_rx(): 99997/100000 pkts, 1280 bytes (good), v1=99997 (good), v2=0 (bad).
srtp_rx(): 99998/100000 pkts, 1280 bytes (good), v1=99998 (good), v2=0 (bad).
srtp_rx(): 99999/100000 pkts, 1280 bytes (good), v1=99999 (good), v2=0 (bad).
srtp_rx(): 100000/100000 pkts, 1280 bytes (good), v1=100000 (good), v2=0 (bad).
 100000/100000 pkts from 138.251.29.20:22694, 0 fails.
start_time: 2023-03-28_20:28:19 (1680031699.467862s)
finish_time: 2023-03-28_20:28:19 (1680031699.467870s)
(duration: 0.000008s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.20 22694
state: 4 (connected)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 0
open_req_dup_rx: 0
open_ack_rx: 1
open_ack_dup_rx: 0
open_req_tx: 1
open_req_re_tx: 0
open_ack_tx: 0
open_ack_re_tx: 0
data_req_rx: 100000
data_req_bytes_rx: 128000000
data_ack_rx: 0
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 0
data_req_bytes_tx: 0
data_ack_tx: 100000
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean rx_rate: 128000000.00 Mbps - 128000000000000 bps
sb409@pc7-075-l:~/Documents/Third_Year/CS3102/P2/code $
```

- test-server-1.c:



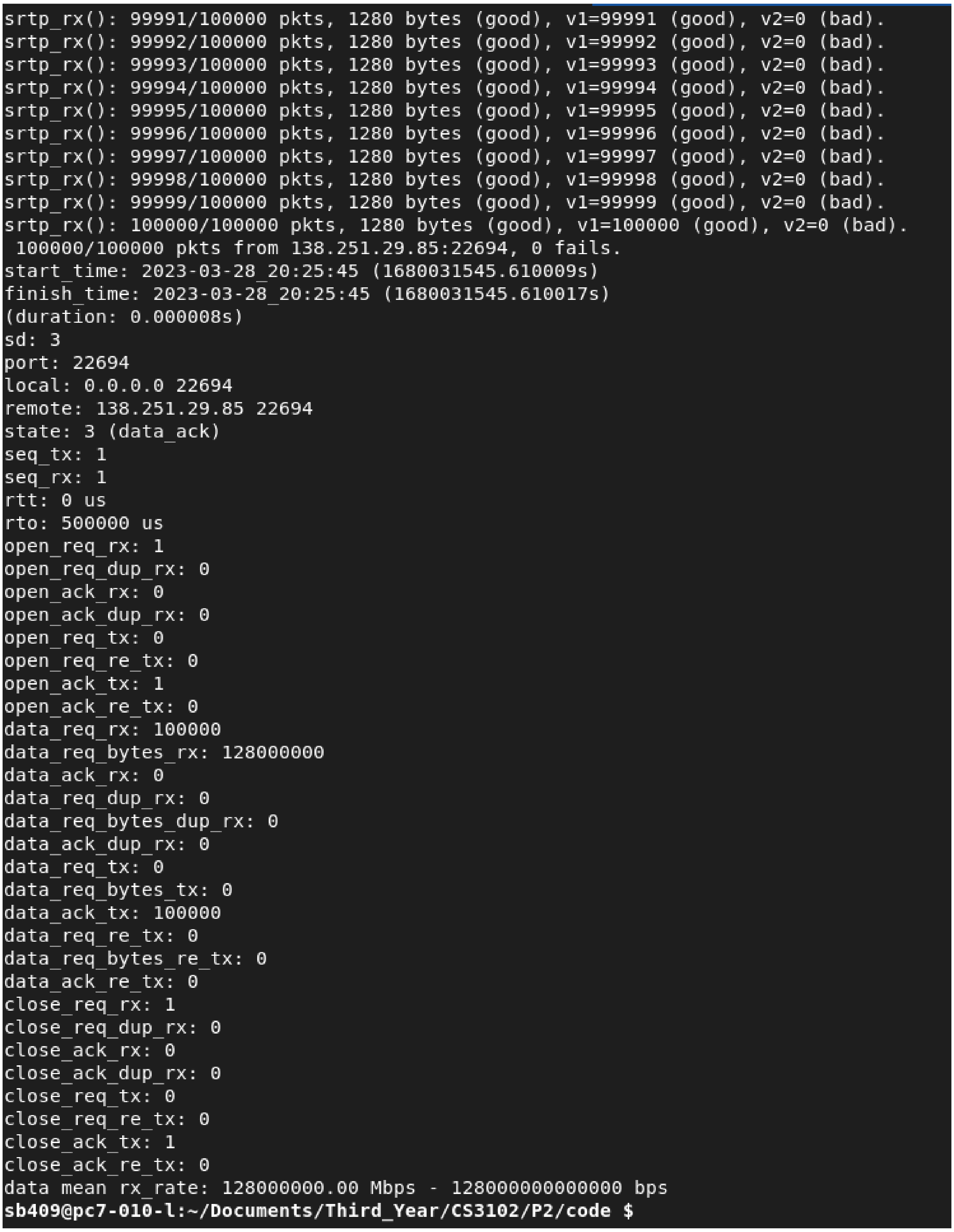
```
srtp_tx(): 99991/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99992/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99993/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99994/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99995/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99996/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99997/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99998/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99999/100000 pkts, 1280/1280 bytes.
srtp_tx(): 100000/100000 pkts, 1280/1280 bytes.
 100000/100000 pkts from 138.251.29.85:22694, 0 fails.
start_time: 2023-03-28_20:28:19 (1680031699.467903s)
finish_time: 2023-03-28_20:28:19 (1680031699.468059s)
(duration: 0.000156s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.85 22694
state: 2 (data_req)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 1
open_req_dup_rx: 0
open_ack_rx: 0
open_ack_dup_rx: 0
open_req_tx: 0
open_req_re_tx: 0
open_ack_tx: 1
open_ack_re_tx: 0
data_req_rx: 0
data_req_bytes_rx: 0
data_ack_rx: 100000
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 100000
data_req_bytes_tx: 128000000
data_ack_tx: 0
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean tx_rate: 6564102.56 Mbps - 6564102564102 bps
sb409@pc7-010-l:~/Documents/Third_Year/CS3102/P2/code $
```

- test-client-2.c:



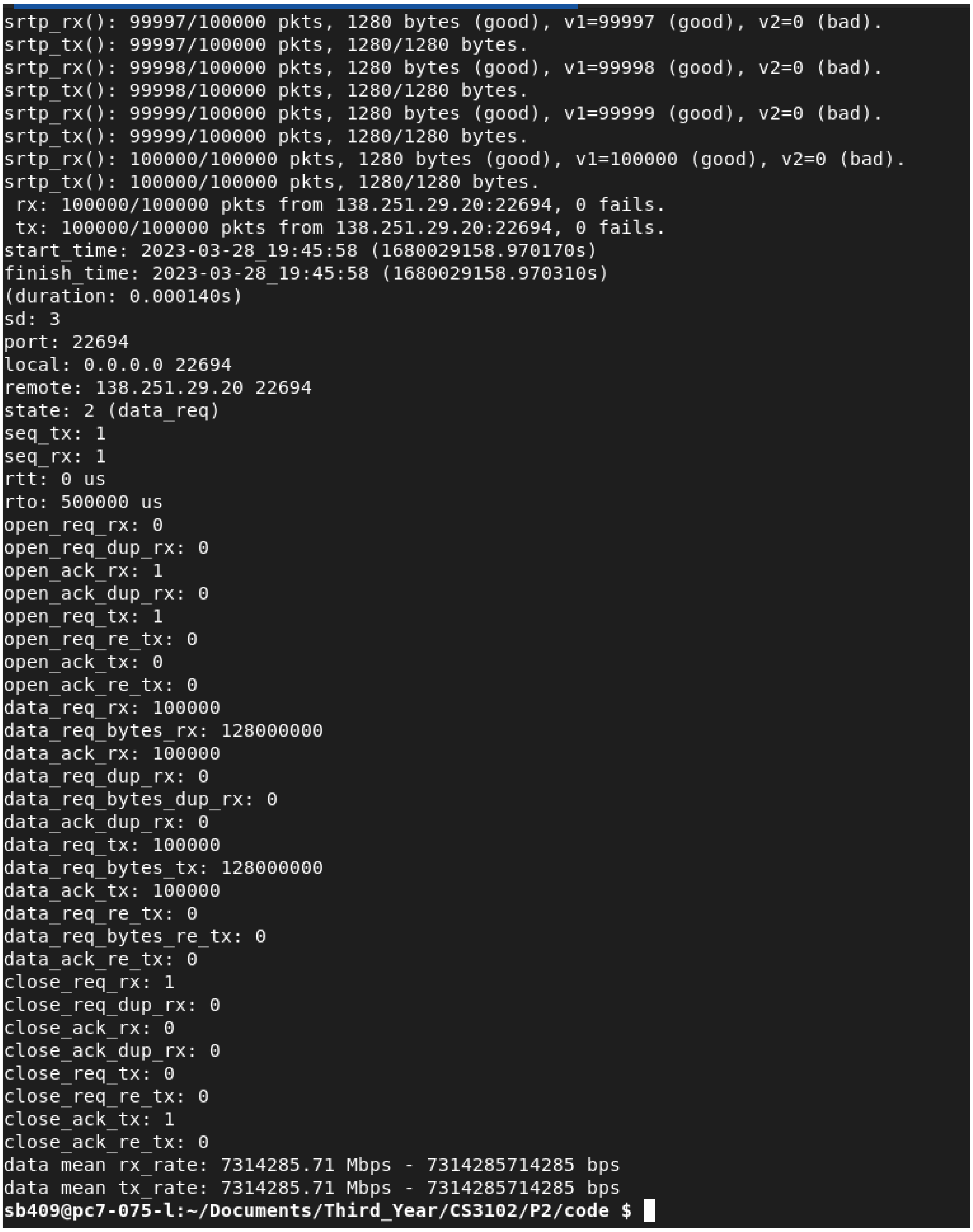
```
srtp_tx(): 99991/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99992/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99993/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99994/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99995/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99996/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99997/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99998/100000 pkts, 1280/1280 bytes.
srtp_tx(): 99999/100000 pkts, 1280/1280 bytes.
srtp_tx(): 100000/100000 pkts, 1280/1280 bytes.
 100000/100000 pkts from 138.251.29.20:22694, 0 fails.
start_time: 2023-03-28_20:25:45 (1680031545.609857s)
finish_time: 2023-03-28_20:25:45 (1680031545.609986s)
(duration: 0.000129s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.20 22694
state: 2 (data_req)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 0
open_req_dup_rx: 0
open_ack_rx: 1
open_ack_dup_rx: 0
open_req_tx: 1
open_req_re_tx: 0
open_ack_tx: 0
open_ack_re_tx: 0
data_req_rx: 0
data_req_bytes_rx: 0
data_ack_rx: 100000
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 100000
data_req_bytes_tx: 128000000
data_ack_tx: 0
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean tx_rate: 7937984.50 Mbps - 7937984496124 bps
sb409@pc7-075-l:~/Documents/Third_Year/CS3102/P2/code $
```

- test-server-2.c:



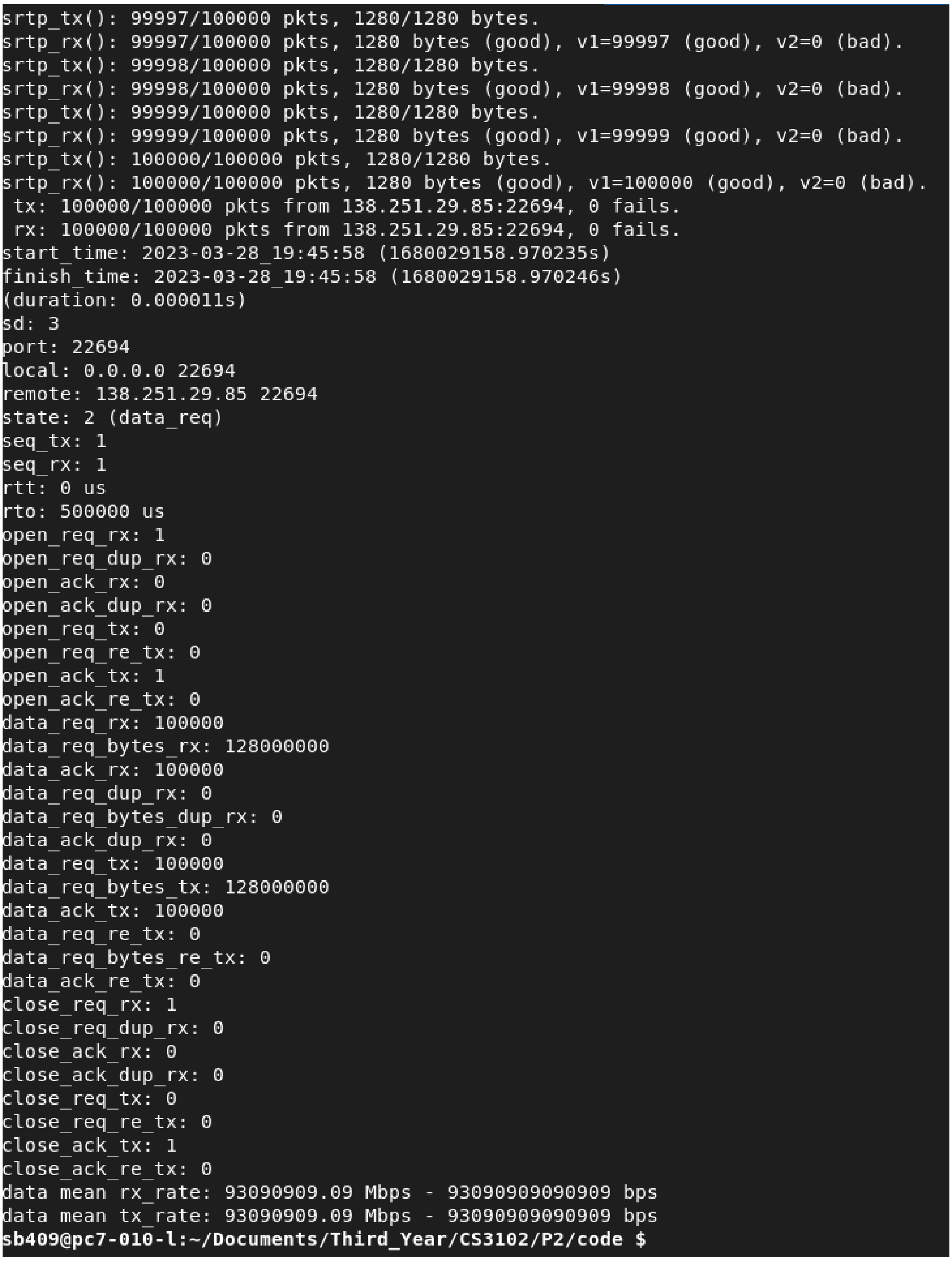
```
srtp_rx(): 99991/100000 pkts, 1280 bytes (good), v1=99991 (good), v2=0 (bad).
srtp_rx(): 99992/100000 pkts, 1280 bytes (good), v1=99992 (good), v2=0 (bad).
srtp_rx(): 99993/100000 pkts, 1280 bytes (good), v1=99993 (good), v2=0 (bad).
srtp_rx(): 99994/100000 pkts, 1280 bytes (good), v1=99994 (good), v2=0 (bad).
srtp_rx(): 99995/100000 pkts, 1280 bytes (good), v1=99995 (good), v2=0 (bad).
srtp_rx(): 99996/100000 pkts, 1280 bytes (good), v1=99996 (good), v2=0 (bad).
srtp_rx(): 99997/100000 pkts, 1280 bytes (good), v1=99997 (good), v2=0 (bad).
srtp_rx(): 99998/100000 pkts, 1280 bytes (good), v1=99998 (good), v2=0 (bad).
srtp_rx(): 99999/100000 pkts, 1280 bytes (good), v1=99999 (good), v2=0 (bad).
srtp_rx(): 100000/100000 pkts, 1280 bytes (good), v1=100000 (good), v2=0 (bad).
 100000/100000 pkts from 138.251.29.85:22694, 0 fails.
start_time: 2023-03-28_20:25:45 (1680031545.610009s)
finish_time: 2023-03-28_20:25:45 (1680031545.610017s)
(duration: 0.000008s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.85 22694
state: 3 (data_ack)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 1
open_req_dup_rx: 0
open_ack_rx: 0
open_ack_dup_rx: 0
open_req_tx: 0
open_req_re_tx: 0
open_ack_tx: 1
open_ack_re_tx: 0
data_req_rx: 100000
data_req_bytes_rx: 128000000
data_ack_rx: 0
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 0
data_req_bytes_tx: 0
data_ack_tx: 100000
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean rx_rate: 128000000.00 Mbps - 128000000000000 bps
sb409@pc7-010-l:~/Documents/Third_Year/CS3102/P2/code $
```

- test-client-3.c:



```
srtp_rx(): 99997/100000 pkts, 1280 bytes (good), v1=99997 (good), v2=0 (bad).
srtp_tx(): 99997/100000 pkts, 1280/1280 bytes.
srtp_rx(): 99998/100000 pkts, 1280 bytes (good), v1=99998 (good), v2=0 (bad).
srtp_tx(): 99998/100000 pkts, 1280/1280 bytes.
srtp_rx(): 99999/100000 pkts, 1280 bytes (good), v1=99999 (good), v2=0 (bad).
srtp_tx(): 99999/100000 pkts, 1280/1280 bytes.
srtp_rx(): 100000/100000 pkts, 1280 bytes (good), v1=100000 (good), v2=0 (bad).
srtp_tx(): 100000/100000 pkts, 1280/1280 bytes.
 rx: 100000/100000 pkts from 138.251.29.20:22694, 0 fails.
 tx: 100000/100000 pkts from 138.251.29.20:22694, 0 fails.
start_time: 2023-03-28_19:45:58 (1680029158.970170s)
finish_time: 2023-03-28_19:45:58 (1680029158.970310s)
(duration: 0.000140s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.20 22694
state: 2 (data_req)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 0
open_req_dup_rx: 0
open_ack_rx: 1
open_ack_dup_rx: 0
open_req_tx: 1
open_req_re_tx: 0
open_ack_tx: 0
open_ack_re_tx: 0
data_req_rx: 100000
data_req_bytes_rx: 128000000
data_ack_rx: 100000
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 100000
data_req_bytes_tx: 128000000
data_ack_tx: 100000
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean rx_rate: 7314285.71 Mbps - 7314285714285 bps
data mean tx_rate: 7314285.71 Mbps - 7314285714285 bps
sb409@pc7-075-l:~/Documents/Third_Year/CS3102/P2/code $
```

- test-server-3.c:



```
srtp_tx(): 99997/100000 pkts, 1280/1280 bytes.
srtp_rx(): 99997/100000 pkts, 1280 bytes (good), v1=99997 (good), v2=0 (bad).
srtp_tx(): 99998/100000 pkts, 1280/1280 bytes.
srtp_rx(): 99998/100000 pkts, 1280 bytes (good), v1=99998 (good), v2=0 (bad).
srtp_tx(): 99999/100000 pkts, 1280/1280 bytes.
srtp_rx(): 99999/100000 pkts, 1280 bytes (good), v1=99999 (good), v2=0 (bad).
srtp_tx(): 100000/100000 pkts, 1280/1280 bytes.
srtp_rx(): 100000/100000 pkts, 1280 bytes (good), v1=100000 (good), v2=0 (bad).
 tx: 100000/100000 pkts from 138.251.29.85:22694, 0 fails.
 rx: 100000/100000 pkts from 138.251.29.85:22694, 0 fails.
start_time: 2023-03-28_19:45:58 (1680029158.970235s)
finish_time: 2023-03-28_19:45:58 (1680029158.970246s)
(duration: 0.000011s)
sd: 3
port: 22694
local: 0.0.0.0 22694
remote: 138.251.29.85 22694
state: 2 (data_req)
seq_tx: 1
seq_rx: 1
rtt: 0 us
rto: 500000 us
open_req_rx: 1
open_req_dup_rx: 0
open_ack_rx: 0
open_ack_dup_rx: 0
open_req_tx: 0
open_req_re_tx: 0
open_ack_tx: 1
open_ack_re_tx: 0
data_req_rx: 100000
data_req_bytes_rx: 128000000
data_ack_rx: 100000
data_req_dup_rx: 0
data_req_bytes_dup_rx: 0
data_ack_dup_rx: 0
data_req_tx: 100000
data_req_bytes_tx: 128000000
data_ack_tx: 100000
data_req_re_tx: 0
data_req_bytes_re_tx: 0
data_ack_re_tx: 0
close_req_rx: 1
close_req_dup_rx: 0
close_ack_rx: 0
close_ack_dup_rx: 0
close_req_tx: 0
close_req_re_tx: 0
close_ack_tx: 1
close_ack_re_tx: 0
data mean rx_rate: 93090909.09 Mbps - 93090909090909 bps
data mean tx_rate: 93090909.09 Mbps - 93090909090909 bps
sb409@pc7-010-l:~/Documents/Third_Year/CS3102/P2/code $
```

Evaluation

However, there I one issues I have noticed emerge from all the tests that I have done which is that the data that is being sent and received in srtp_tx() and srtp_rx() isn’t entirely correct because part of it is being corrupted. In the output from the tests, v1 is always correct but v2 isn’t. This most likely has to do with how the data is being handled which I wasn’t able to debug due to time constraints.

Conclusion

For the most part, my implementation seems to be successful for that parts that I have attempted. The TCP handshake protocol seems to be working as intended allowing a client and server to establish a connection. And both end-points are able to communicate with each other by sending data to and from the end-points. There’s just that one error I have identified from the tests I have conducted which is limiting the performance of my program.