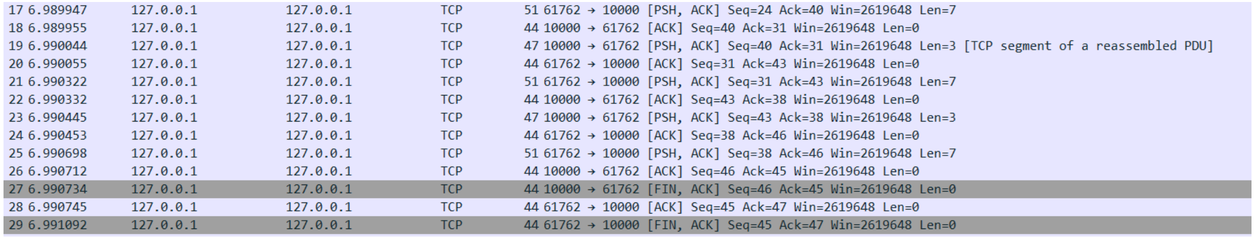
ASSIGNMENT-3

Siddharth Gupta

Roll: 12041450

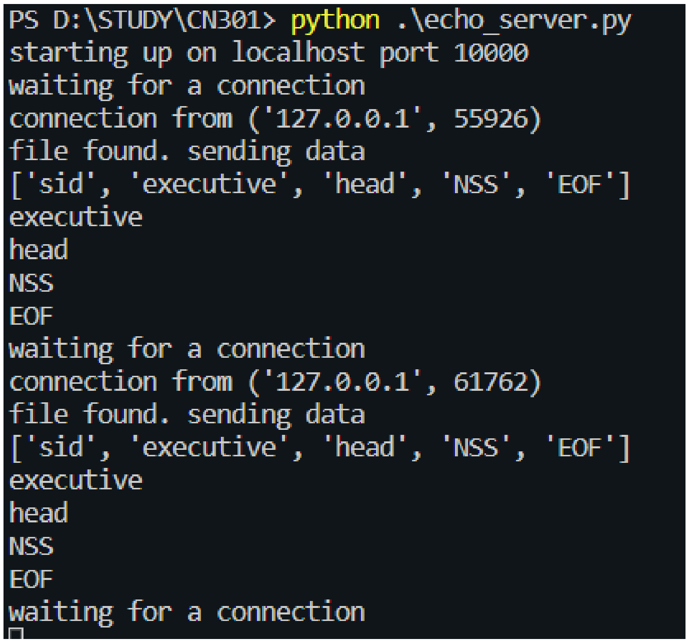
PART 1:

1. Create 2 files for server and client respectively
2. Client asks for the file from the server
3. Server search in the local directory
4. If found return the words from the file
5. If not found sends an error of 404: File not Found



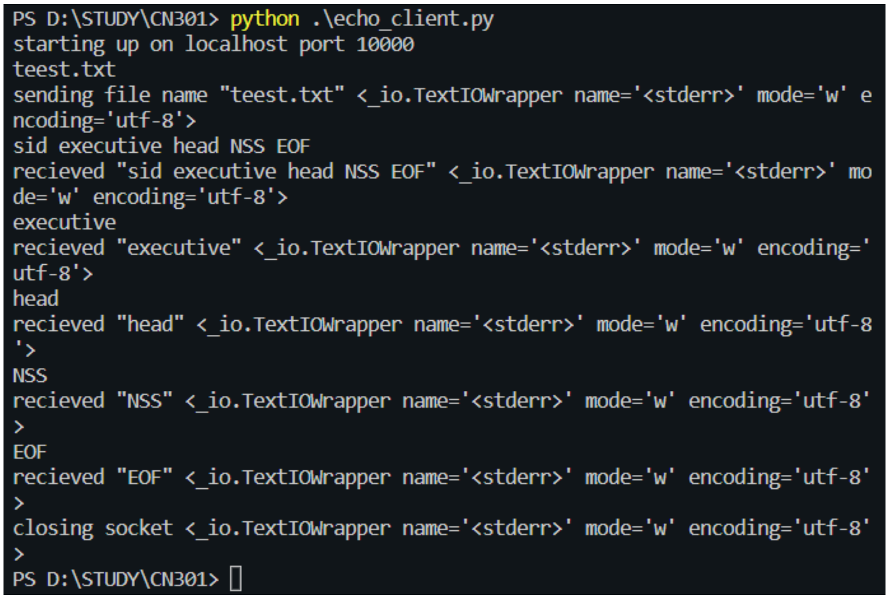
```
17 6.989947   127.0.0.1   127.0.0.1   TCP   51 61762 → 10000 [PSH, ACK] Seq=24 Ack=40 Win=2619648 Len=7
18 6.989955   127.0.0.1   127.0.0.1   TCP   44 10000 → 61762 [ACK] Seq=40 Ack=31 Win=2619648 Len=0
19 6.990044   127.0.0.1   127.0.0.1   TCP   47 10000 → 61762 [PSH, ACK] Seq=40 Ack=31 Win=2619648 Len=3 [TCP segment of a reassembled PDU]
20 6.990055   127.0.0.1   127.0.0.1   TCP   44 61762 → 10000 [ACK] Seq=31 Ack=43 Win=2619648 Len=0
21 6.990322   127.0.0.1   127.0.0.1   TCP   51 61762 → 10000 [PSH, ACK] Seq=31 Ack=43 Win=2619648 Len=7
22 6.990332   127.0.0.1   127.0.0.1   TCP   44 10000 → 61762 [ACK] Seq=43 Ack=38 Win=2619648 Len=0
23 6.990445   127.0.0.1   127.0.0.1   TCP   47 10000 → 61762 [PSH, ACK] Seq=43 Ack=38 Win=2619648 Len=3
24 6.990453   127.0.0.1   127.0.0.1   TCP   44 61762 → 10000 [ACK] Seq=38 Ack=46 Win=2619648 Len=0
25 6.990698   127.0.0.1   127.0.0.1   TCP   51 61762 → 10000 [PSH, ACK] Seq=38 Ack=46 Win=2619648 Len=7
26 6.990712   127.0.0.1   127.0.0.1   TCP   44 10000 → 61762 [ACK] Seq=46 Ack=45 Win=2619648 Len=0
27 6.990734   127.0.0.1   127.0.0.1   TCP   44 10000 → 61762 [FIN, ACK] Seq=46 Ack=45 Win=2619648 Len=0
28 6.990745   127.0.0.1   127.0.0.1   TCP   44 61762 → 10000 [ACK] Seq=45 Ack=47 Win=2619648 Len=0
29 6.991092   127.0.0.1   127.0.0.1   TCP   44 61762 → 10000 [FIN, ACK] Seq=45 Ack=47 Win=2619648 Len=0
```

Terminal arguments at the server side:



```
PS D:\STUDY\CN301> python .\echo_server.py
starting up on localhost port 10000
waiting for a connection
connection from ('127.0.0.1', 55926)
file found. sending data
['sid', 'executive', 'head', 'NSS', 'EOF']
executive
head
NSS
EOF
waiting for a connection
connection from ('127.0.0.1', 61762)
file found. sending data
['sid', 'executive', 'head', 'NSS', 'EOF']
executive
head
NSS
EOF
waiting for a connection
```

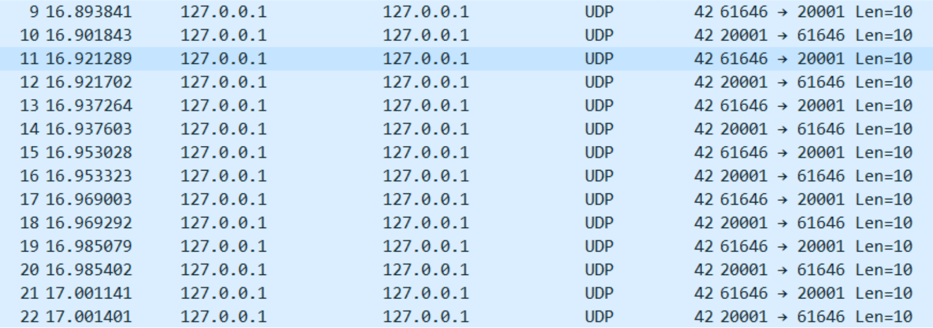
Terminal arguments at the client side:



```
PS D:\STUDY\CN301> python .\echo_client.py
starting up on localhost port 10000
teest.txt
sending file name "teest.txt" <_io.TextIOWrapper name='<stderr>' mode='w' e
ncoding='utf-8'>
sid executive head NSS EOF
recieved "sid executive head NSS EOF" <_io.TextIOWrapper name='<stderr>' mo
de='w' encoding='utf-8'>
executive
recieved "executive" <_io.TextIOWrapper name='<stderr>' mode='w' encoding='
utf-8'>
head
recieved "head" <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8
'>
NSS
recieved "NSS" <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'
>
EOF
recieved "EOF" <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'
>
closing socket <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'
>
PS D:\STUDY\CN301>
```

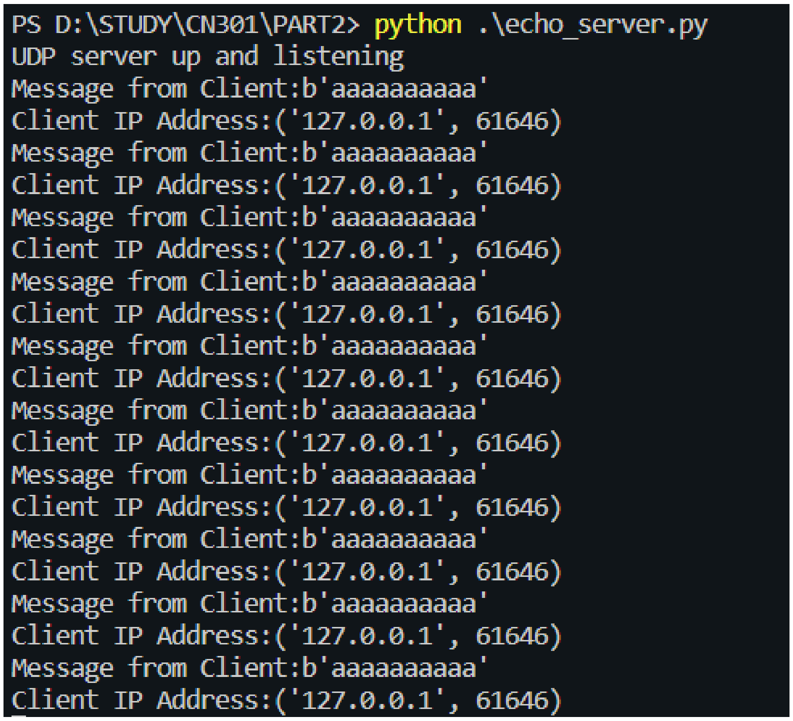
PART 2_1:

1. Create a UDP client server
2. Create 2 files for server and client respectively
3. Calculate the RTT for each of the packets send
4. Waiting 0.01 sec for receiving the return packet from the server
5. Server sends the packets as soon as it receives one.
6. If return packet not received then the packet lost error.
7. Every time the client sends data of fixed datasize



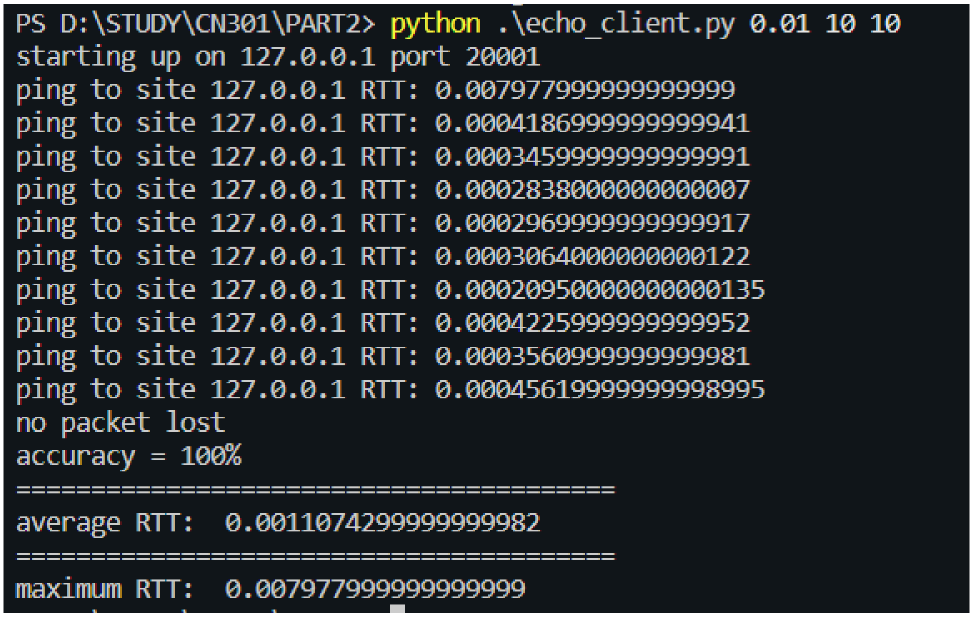
```
 9 16.893841   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
10 16.901843   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
11 16.921289   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
12 16.921702   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
13 16.937264   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
14 16.937603   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
15 16.953028   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
16 16.953323   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
17 16.969003   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
18 16.969292   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
19 16.985079   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
20 16.985402   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
21 17.001141   127.0.0.1    127.0.0.1    UDP    42 61646 → 20001 Len=10
22 17.001401   127.0.0.1    127.0.0.1    UDP    42 20001 → 61646 Len=10
```

Server side traffic shown below.



```
PS D:\STUDY\CN301\PART2> python .\echo_server.py
UDP server up and listening
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 61646)
```

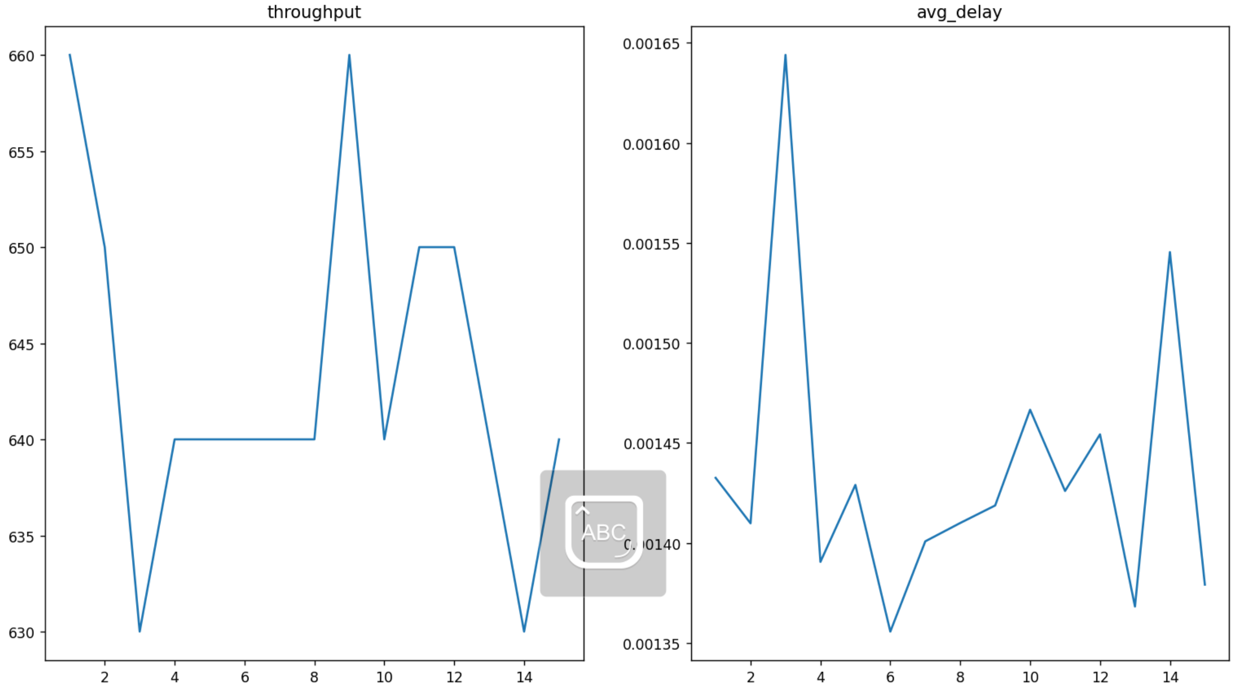
Client side traffic shown below:



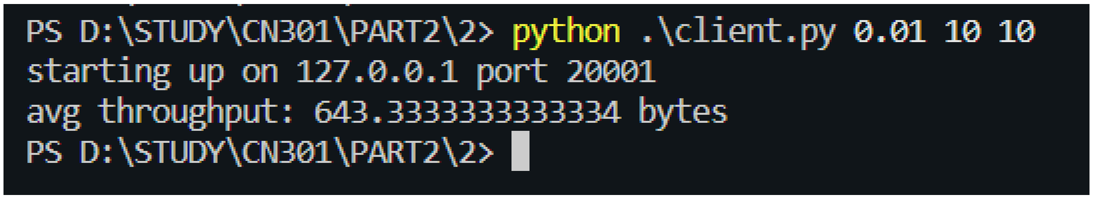
```
PS D:\STUDY\CN301\PART2> python .\echo_client.py 0.01 10 10
starting up on 127.0.0.1 port 20001
ping to site 127.0.0.1 RTT: 0.007977999999999999
ping to site 127.0.0.1 RTT: 0.00041869999999999941
ping to site 127.0.0.1 RTT: 0.00034599999999999991
ping to site 127.0.0.1 RTT: 0.00028380000000000007
ping to site 127.0.0.1 RTT: 0.00029699999999999917
ping to site 127.0.0.1 RTT: 0.00030640000000000122
ping to site 127.0.0.1 RTT: 0.000209500000000000135
ping to site 127.0.0.1 RTT: 0.00042259999999999952
ping to site 127.0.0.1 RTT: 0.00035609999999999981
ping to site 127.0.0.1 RTT: 0.000456199999999998995
no packet lost
accuracy = 100%
========================================
average RTT:  0.0011074299999999982
========================================
maximum RTT:  0.007977999999999999
```

PART 2_2:

1. Calculating the throughput and the average delay for each second for 15 times
2. Throughput can be done just by summing the packet size for each second as we are calculating for 1 second.
3. Calculate the delay of all packets sent in one second then calculating the average delay for that second.
4. Finally plotting the graph using matplotlib library in python

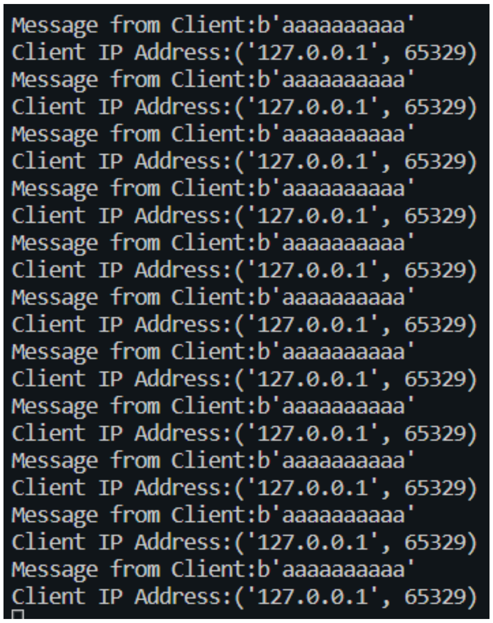


Client side terminal shown below:



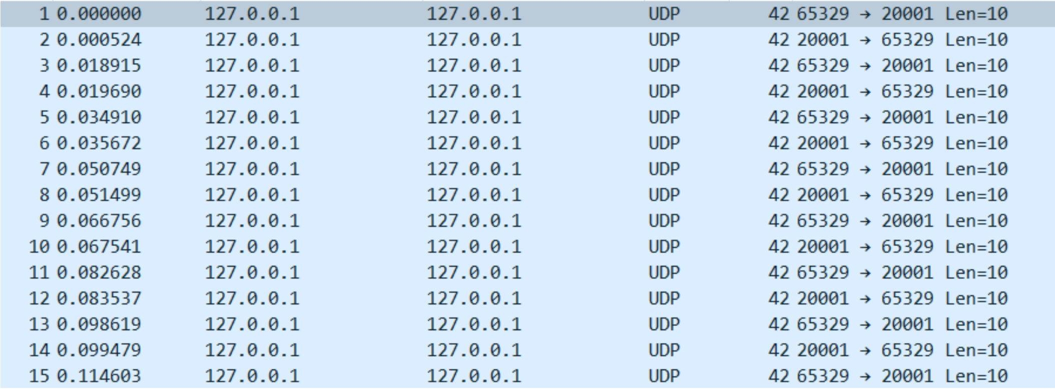
```
PS D:\STUDY\CN301\PART2\2> python .\client.py 0.01 10 10
starting up on 127.0.0.1 port 20001
avg throughput: 643.3333333333334 bytes
PS D:\STUDY\CN301\PART2\2>
```

Server side traffic shown below:



```
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('127.0.0.1', 65329)
```

Wireshark packet capture shown below:



```
 1 0.000000   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
 2 0.000524   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
 3 0.018915   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
 4 0.019690   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
 5 0.034910   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
 6 0.035672   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
 7 0.050749   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
 8 0.051499   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
 9 0.066756   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
10 0.067541   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
11 0.082628   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
12 0.083537   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
13 0.098619   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
14 0.099479   127.0.0.1   127.0.0.1   UDP   42 20001 → 65329 Len=10
15 0.114603   127.0.0.1   127.0.0.1   UDP   42 65329 → 20001 Len=10
```

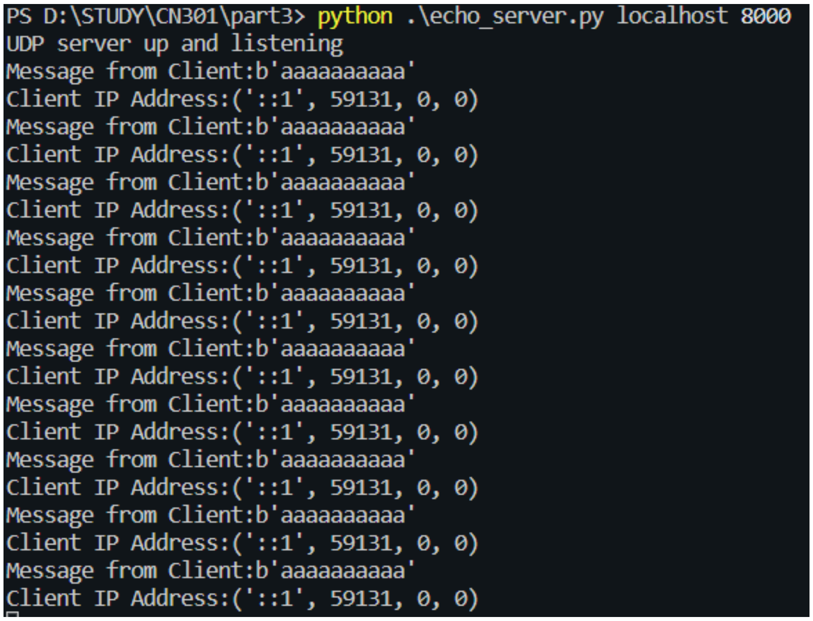
PART3:

For localhost:

1. Create a UDP client server
2. Create 2 files for server and client respectively
3. Both the files take 2 command line arguments <localhost_type> and the <port>
4. For ipv4 :<localhost_type> = ‘localhost’
5. For ipv6:<localhost_type> = ‘ip6-localhost’
6. Pings the desired server with packets of same type of ip version.

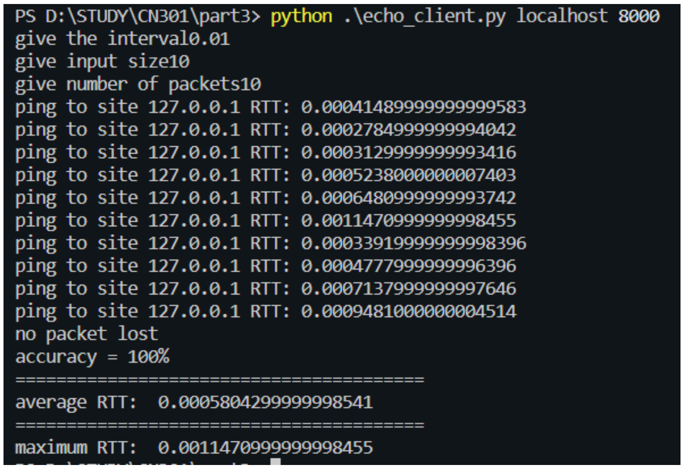
**Using the part 2 ping function to do some actions in this question

Server side traffic shown below:



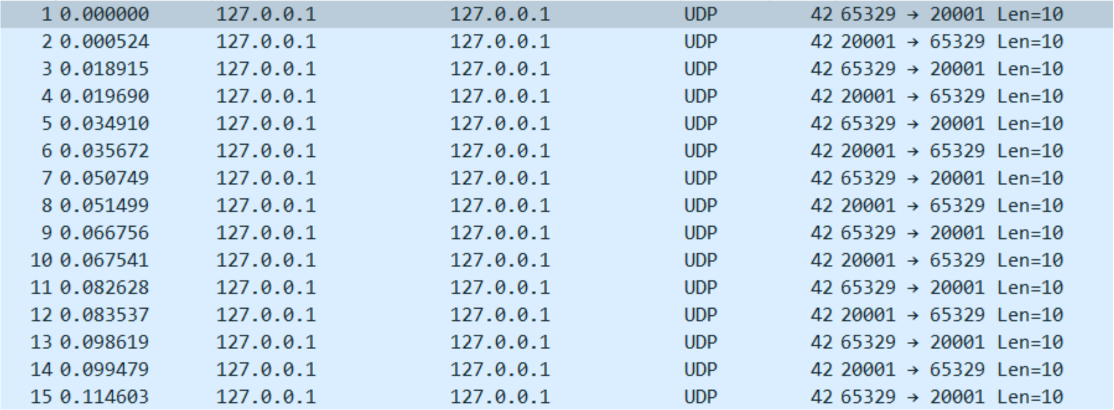
```
PS D:\STUDY\CN301\part3> python .\echo_server.py localhost 8000
UDP server up and listening
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 59131, 0, 0)
```

Client side terminal server shown below.



```
PS D:\STUDY\CN301\part3> python .\echo_client.py localhost 8000
give the interval0.01
give input size10
give number of packets10
ping to site 127.0.0.1 RTT: 0.00041489999999999583
ping to site 127.0.0.1 RTT: 0.0002784999999994042
ping to site 127.0.0.1 RTT: 0.0003129999999993416
ping to site 127.0.0.1 RTT: 0.0005238000000007403
ping to site 127.0.0.1 RTT: 0.0006480999999993742
ping to site 127.0.0.1 RTT: 0.0011470999999998455
ping to site 127.0.0.1 RTT: 0.00033919999999998396
ping to site 127.0.0.1 RTT: 0.0004777999999996396
ping to site 127.0.0.1 RTT: 0.0007137999999997646
ping to site 127.0.0.1 RTT: 0.0009481000000004514
no packet lost
accuracy = 100%
========================================
average RTT:  0.0005804299999998541
========================================
maximum RTT:  0.0011470999999998455
```

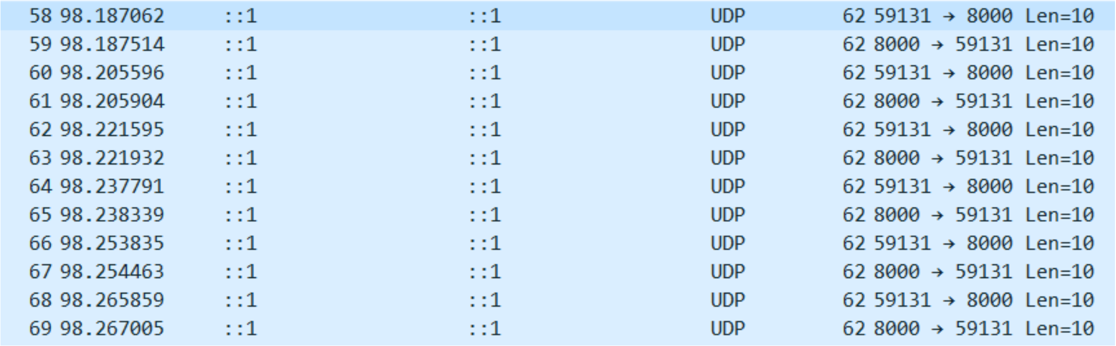
Wireshark packet capture : 127.0.0.1 defines that its a ipv4 packet



1	0.000000	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
2	0.000524	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
3	0.018915	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
4	0.019690	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
5	0.034910	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
6	0.035672	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
7	0.050749	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
8	0.051499	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
9	0.066756	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
10	0.067541	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
11	0.082628	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
12	0.083537	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
13	0.098619	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10
14	0.099479	127.0.0.1	127.0.0.1	UDP	42	20001 → 65329 Len=10
15	0.114603	127.0.0.1	127.0.0.1	UDP	42	65329 → 20001 Len=10

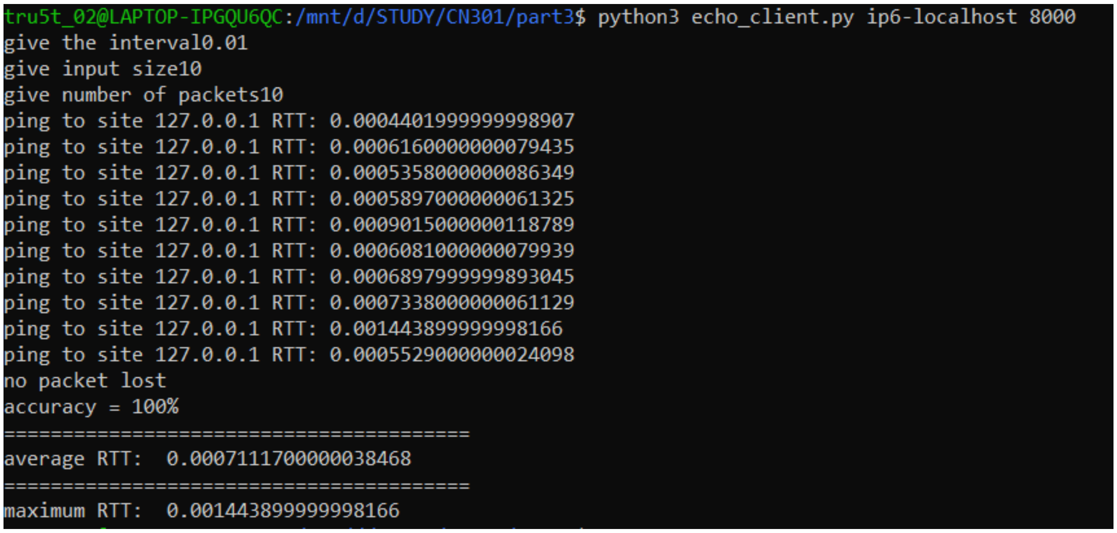
For ip6-localhost:

Wireshark packet capture: ::1 shows its a ipv6 packet



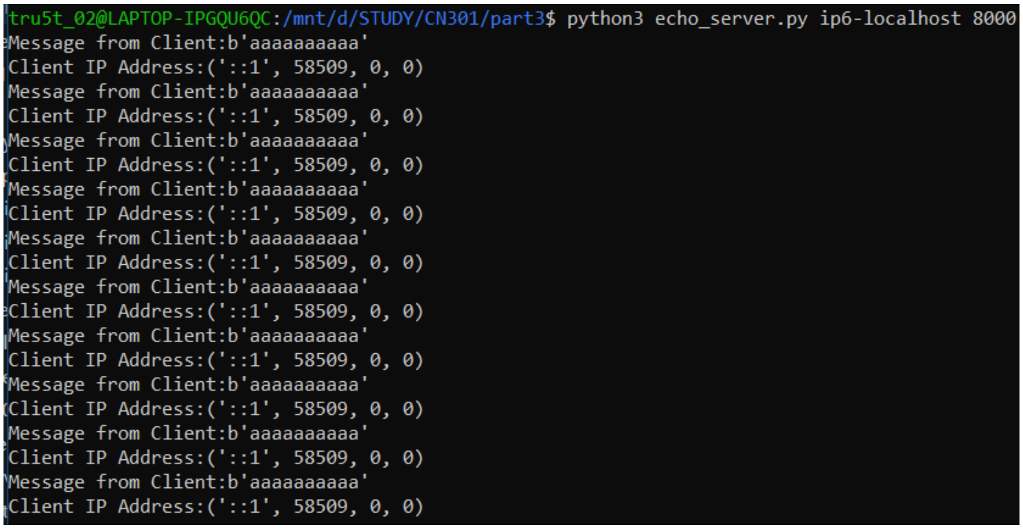
58	98.187062	::1	::1	UDP	62	59131 → 8000 Len=10
59	98.187514	::1	::1	UDP	62	8000 → 59131 Len=10
60	98.205596	::1	::1	UDP	62	59131 → 8000 Len=10
61	98.205904	::1	::1	UDP	62	8000 → 59131 Len=10
62	98.221595	::1	::1	UDP	62	59131 → 8000 Len=10
63	98.221932	::1	::1	UDP	62	8000 → 59131 Len=10
64	98.237791	::1	::1	UDP	62	59131 → 8000 Len=10
65	98.238339	::1	::1	UDP	62	8000 → 59131 Len=10
66	98.253835	::1	::1	UDP	62	59131 → 8000 Len=10
67	98.254463	::1	::1	UDP	62	8000 → 59131 Len=10
68	98.265859	::1	::1	UDP	62	59131 → 8000 Len=10
69	98.267005	::1	::1	UDP	62	8000 → 59131 Len=10

Client side terminal shown below:



```
tru5t_02@LAPTOP-IPGQU6QC:/mnt/d/STUDY/CN301/part3$ python3 echo_client.py ip6-localhost 8000
give the interval0.01
give input size10
give number of packets10
ping to site 127.0.0.1 RTT: 0.00044019999999998907
ping to site 127.0.0.1 RTT: 0.0006160000000079435
ping to site 127.0.0.1 RTT: 0.0005358000000086349
ping to site 127.0.0.1 RTT: 0.0005897000000061325
ping to site 127.0.0.1 RTT: 0.0009015000000118789
ping to site 127.0.0.1 RTT: 0.0006081000000079939
ping to site 127.0.0.1 RTT: 0.0006897999999893045
ping to site 127.0.0.1 RTT: 0.0007338000000061129
ping to site 127.0.0.1 RTT: 0.001443899999998166
ping to site 127.0.0.1 RTT: 0.0005529000000024098
no packet lost
accuracy = 100%
========================================
average RTT:  0.0007111700000038468
========================================
maximum RTT:  0.001443899999998166
```

Server Side traffic shown below:

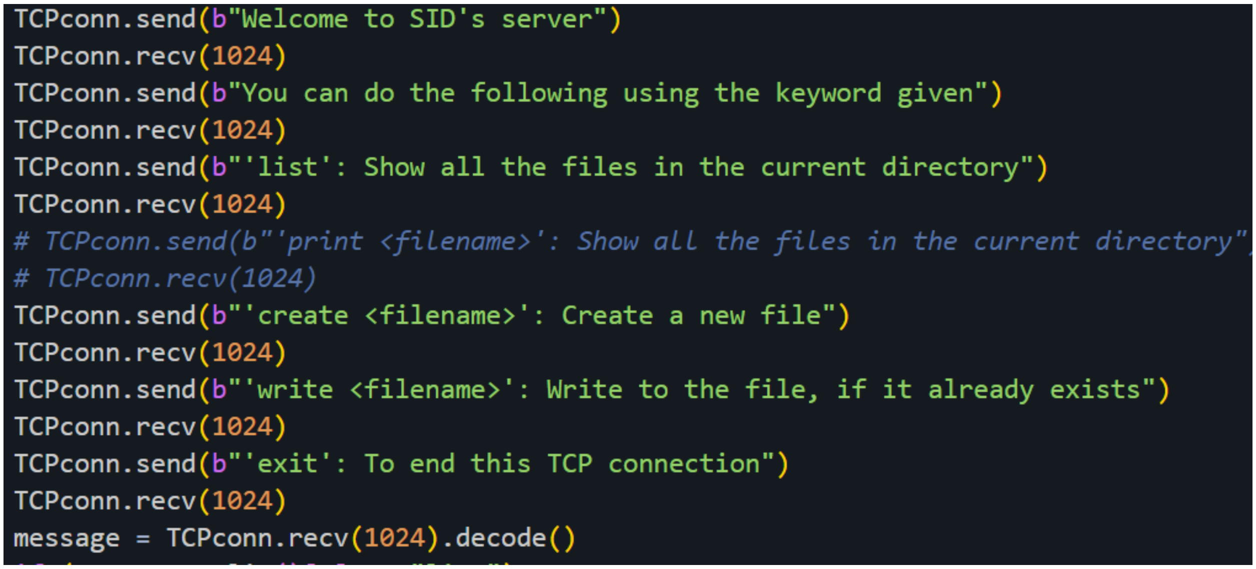


```
tru5t_02@LAPTOP-IPGQU6QC:/mnt/d/STUDY/CN301/part3$ python3 echo_server.py ip6-localhost 8000
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
Message from Client:b'aaaaaaaaaa'
Client IP Address:('::1', 58509, 0, 0)
```

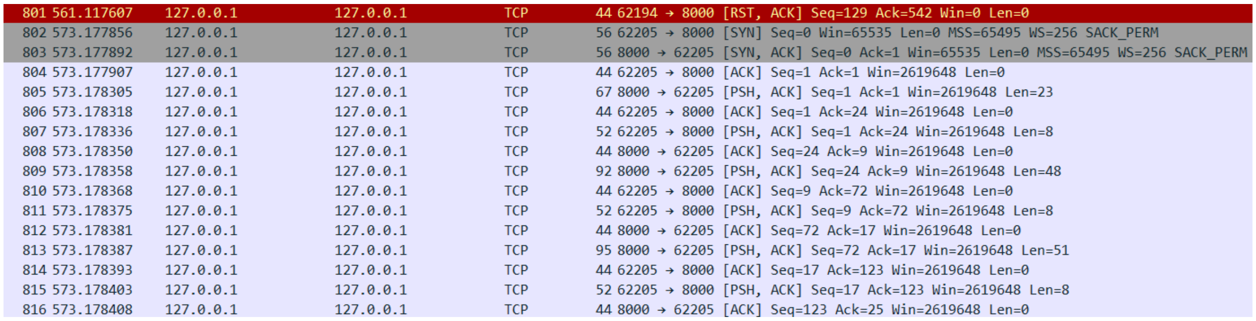
PART 4:

Creating a system command line arguments in which the server responds to various command line features like list , create , write or exit.
For each argument the server interacts with the client and responds to the query

How to send query to the server.

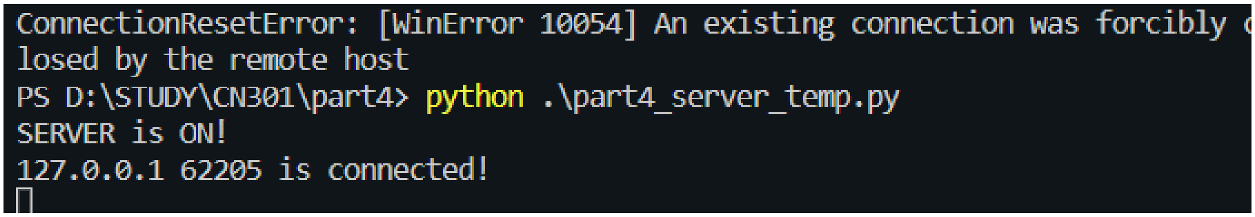


```
TCPconn.send(b"Welcome to SID's server")
TCPconn.recv(1024)
TCPconn.send(b"You can do the following using the keyword given")
TCPconn.recv(1024)
TCPconn.send(b"'list': Show all the files in the current directory")
TCPconn.recv(1024)
# TCPconn.send(b"'print <filename>': Show all the files in the current directory"
# TCPconn.recv(1024)
TCPconn.send(b"'create <filename>': Create a new file")
TCPconn.recv(1024)
TCPconn.send(b"'write <filename>': Write to the file, if it already exists")
TCPconn.recv(1024)
TCPconn.send(b"'exit': To end this TCP connection")
TCPconn.recv(1024)
message = TCPconn.recv(1024).decode()
```



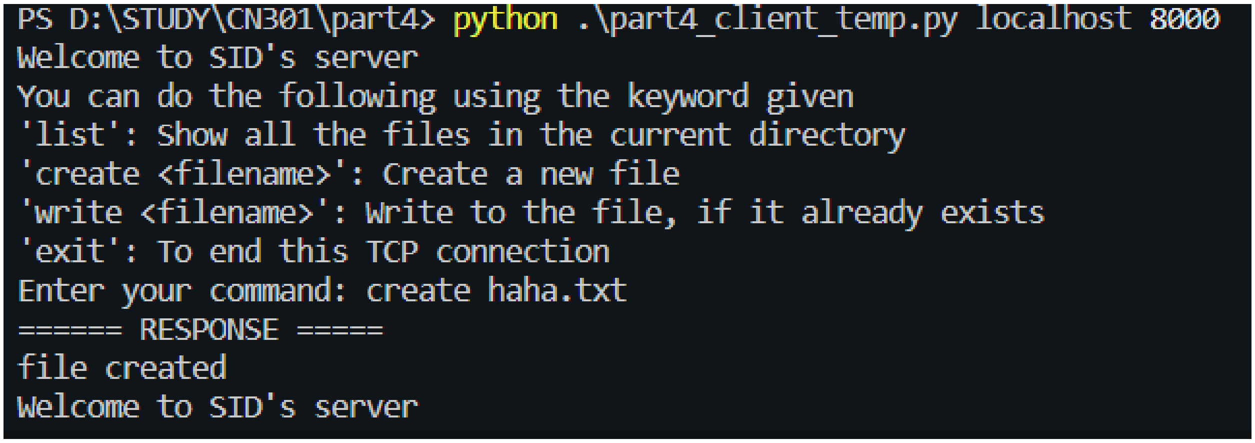
```
801 561.117607    127.0.0.1          127.0.0.1          TCP       44 62194 → 8000 [RST, ACK] Seq=129 Ack=542 Win=0 Len=0
802 573.177856    127.0.0.1          127.0.0.1          TCP       56 62205 → 8000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
803 573.177892    127.0.0.1          127.0.0.1          TCP       56 8000 → 62205 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
804 573.177907    127.0.0.1          127.0.0.1          TCP       44 62205 → 8000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
805 573.178305    127.0.0.1          127.0.0.1          TCP       67 8000 → 62205 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=23
806 573.178318    127.0.0.1          127.0.0.1          TCP       44 62205 → 8000 [ACK] Seq=1 Ack=24 Win=2619648 Len=0
807 573.178336    127.0.0.1          127.0.0.1          TCP       52 62205 → 8000 [PSH, ACK] Seq=1 Ack=24 Win=2619648 Len=8
808 573.178350    127.0.0.1          127.0.0.1          TCP       44 8000 → 62205 [ACK] Seq=24 Ack=9 Win=2619648 Len=0
809 573.178358    127.0.0.1          127.0.0.1          TCP       92 8000 → 62205 [PSH, ACK] Seq=24 Ack=9 Win=2619648 Len=48
810 573.178368    127.0.0.1          127.0.0.1          TCP       44 62205 → 8000 [ACK] Seq=9 Ack=72 Win=2619648 Len=0
811 573.178375    127.0.0.1          127.0.0.1          TCP       52 62205 → 8000 [PSH, ACK] Seq=9 Ack=72 Win=2619648 Len=8
812 573.178381    127.0.0.1          127.0.0.1          TCP       44 8000 → 62205 [ACK] Seq=72 Ack=17 Win=2619648 Len=0
813 573.178387    127.0.0.1          127.0.0.1          TCP       95 8000 → 62205 [PSH, ACK] Seq=72 Ack=17 Win=2619648 Len=51
814 573.178393    127.0.0.1          127.0.0.1          TCP       44 62205 → 8000 [ACK] Seq=17 Ack=123 Win=2619648 Len=0
815 573.178403    127.0.0.1          127.0.0.1          TCP       52 62205 → 8000 [PSH, ACK] Seq=17 Ack=123 Win=2619648 Len=8
816 573.178408    127.0.0.1          127.0.0.1          TCP       44 8000 → 62205 [ACK] Seq=123 Ack=25 Win=2619648 Len=0
```

Server side terminal shown below.



```
ConnectionResetError: [WinError 10054] An existing connection was forcibly c
losed by the remote host
PS D:\STUDY\CN301\part4> python .\part4_server_temp.py
SERVER is ON!
127.0.0.1 62205 is connected!
```

Client side terminal shown below.



```
PS D:\STUDY\CN301\part4> python .\part4_client_temp.py localhost 8000
Welcome to SID's server
You can do the following using the keyword given
'list': Show all the files in the current directory
'create <filename>': Create a new file
'write <filename>': Write to the file, if it already exists
'exit': To end this TCP connection
Enter your command: create haha.txt
====== RESPONSE =====
file created
Welcome to SID's server
```