CO 223 - Computer Communication Networks I

Lab 05 - Socket Programming in Python II

Date - 01/10/2021

Reg No - E/18/397

Task 1

1)

server.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20001
buffersize = 1024
msgFromServer = "Hello UDP Client"
bytesToSend = str.encode(msgFromServer)

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientmsg = "message from client: "+ message.decode()
    clientIP = "client IP address: {}".format(address)

    print(clientmsg)
    print(clientIP)

    # sending reply to client
    UDPServerSocket.sendto(bytesToSend, address)

```

Figure 1.1: server code for task 1

client.py > ...

```python
import socket

msgFromClient = "Helllo UDP Server"
bytesToSend = str.encode(msgFromClient)
serverAddressPort = ("127.0.0.1",20001)
buffersize = 1024

# create UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)

# Send to server using created UDP socket

UDPClientSocket.sendto(bytesToSend,serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(buffersize)
msg = "Message from server: "+msgFromServer[0].decode()

print(msg)
```

Figure 1.2: client code for task 1

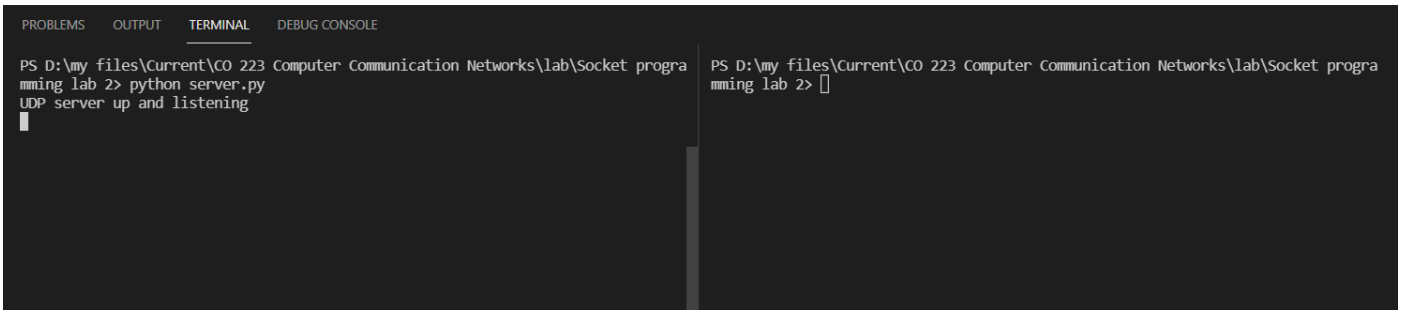

Figure 1.3: both terminals after running only server code

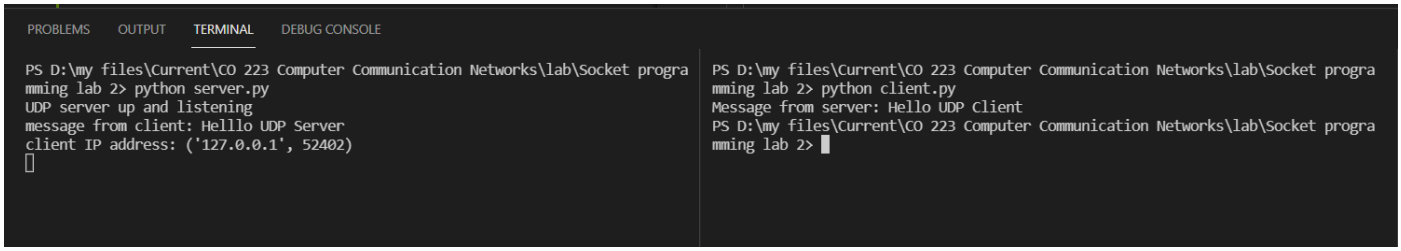

Figure 1.4: Both terminal after running client code

2) Since UDP is connectionless or stateless protocol, server may respond to every request

3) Handling millions of requests depends on the load balancing. Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool. A load balancer is a reverse proxy. It presents a virtual IP address (VIP) representing the application to the client. The client connects to the VIP and the load balancer decides through its algorithms to send the connection to a specific application instance on a server. The load balancer continues to manage and monitor the connection for the entire duration.

Task 2

1)

task 2 > server1.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20001
buffersize = 1024
msgFromServer = "Hello UDP Client(from server 1)"
bytesToSend = str.encode(msgFromServer)

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server 1 up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientmsg = "message from client: "+ message.decode()
    clientIP = "client IP address: {}".format(address)

    print(clientmsg)
    print(clientIP)

    # sending reply to client
    UDPServerSocket.sendto(bytesToSend, address)

```

Figure 2.1: server 1 code

task 2 > server2.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20000
buffersize = 1024
msgFromServer = "Hello UDP Client(from server 2)"
bytesToSend = str.encode(msgFromServer)

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server 2 up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientmsg = "message from client: "+ message.decode()
    clientIP = "client IP address: {}".format(address)

    print(clientmsg)
    print(clientIP)

    # sending reply to client
    UDPServerSocket.sendto(bytesToSend, address)
```

Figure 2.2: server 2 code

task 2 > client.py > ...

```python
import socket

msgFromClient = "Helllo UDP Server"
bytesToSend = str.encode(msgFromClient)
serverAddressPort = ("127.0.0.1",20001)
serverAddressPort2 = ("127.0.0.1",20000)
buffersize = 1024

# create UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)

# Send to server using created UDP socket
UDPClientSocket.sendto(bytesToSend,serverAddressPort)
UDPClientSocket.sendto(bytesToSend,serverAddressPort2)

# receiving and printing the messahge from 1 st server
msgFromServer = UDPClientSocket.recvfrom(buffersize)
msg1 = "Message from server 1: "+msgFromServer[0].decode()
print(msg1)

# receiving and printing the messagee from 2 nd server
msgFromServer = UDPClientSocket.recvfrom(buffersize)
msg2 = "Message from server 2: "+msgFromServer[0].decode()
print(msg2)
```

Figure 2.3: client code

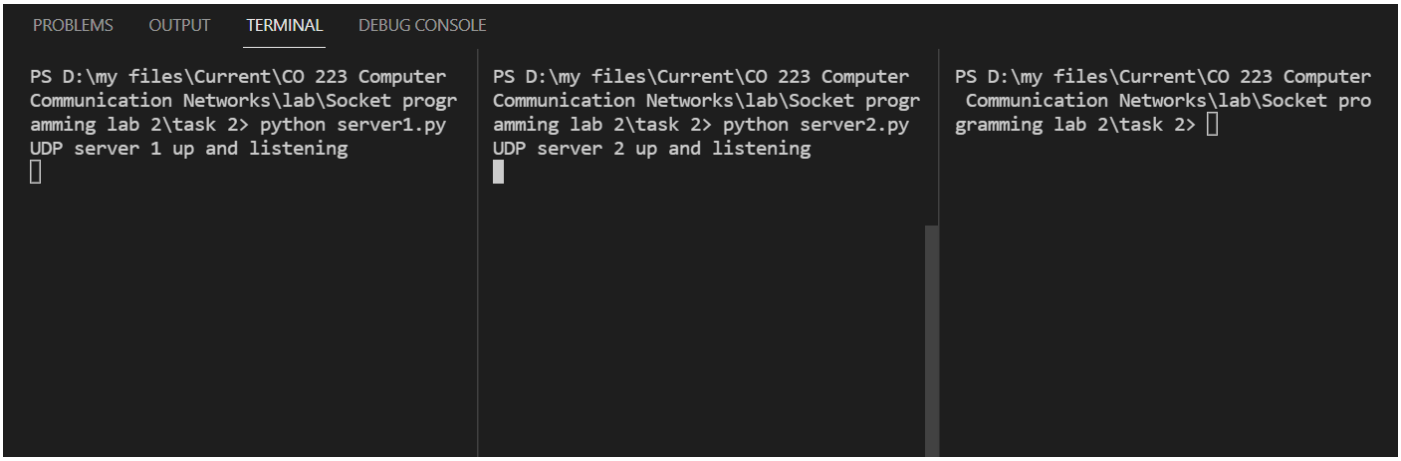

Figure 2.4: Terminals after running both servers

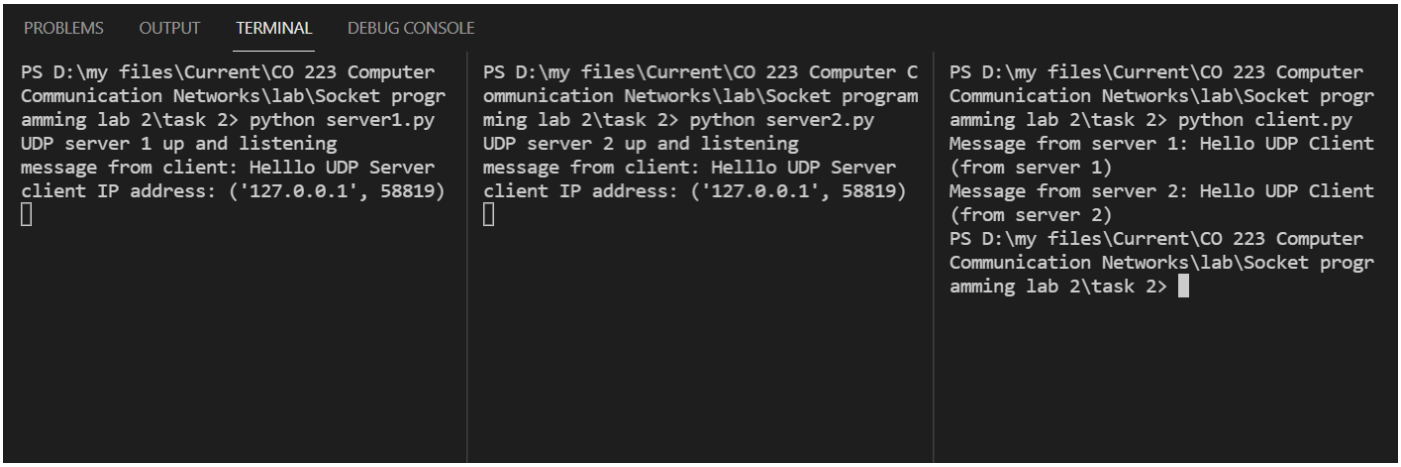

Figure 2.5: Terminals after running client code

2) yes, it's possible.

- Socket specifies both server addresses to send. Since UDP is not connection oriented and UDP sockets are not classified to client socket and server socket, it is able to accept several concurrent connections from different remote hosts. Therefore, it's possible to create multiple servers which listens and communicate with a single client.

Task 3

1) since UDP is connectionless and only sending datagrams usage of connect() won't results anything. Like TCP there isn't any 3-way handshake. For datagram sockets, the connect() call specifies the peer for a socket.

2) As mentioned above connect() on UDP specifies the peer for a socket. Therefore, no need of specify the destination IP address and the port. Thus we can use send() instead of sendto() and we can use recv() instead of recvfrom() in the client code.

server3.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20001
buffersize = 1024
msgFromServer = "Hello UDP Client"
bytesToSend = str.encode(msgFromServer)

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientmsg = "message from client: {}".format(message)
    clientIP = "client IP address: {}".format(address)

    print(clientmsg)
    print(clientIP)

    # sending reply to client
    UDPServerSocket.sendto(bytesToSend,address)

```

Figure 3.1: server code

client3.py > ...

```python
import socket

msgFromClient = "Helllo UDP Server"
bytesToSend = str.encode(msgFromClient)
buffersize = 1024

# create UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)
UDPClientSocket.connect(("127.0.0.1",20001))
# Send to server using created UDP socket

UDPClientSocket.send(bytesToSend)

msgFromServer = UDPClientSocket.recv(buffersize).decode()
msg = "Message from server: "+msgFromServer

print(msg)

```

Figure 3.2: client code

Figure 3.3: both terminals after running server code

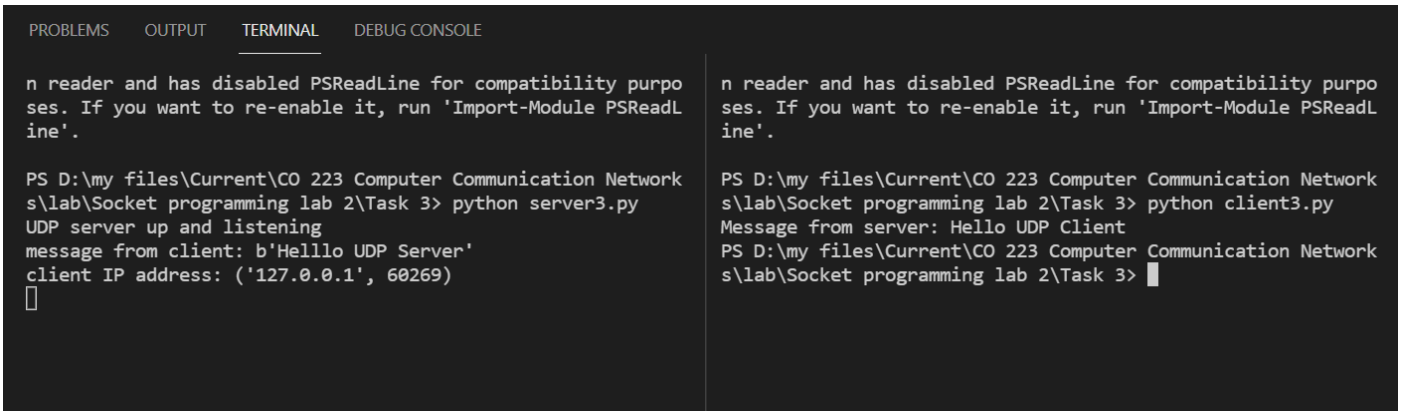

Figure 3.4: both terminals after running client code

Task 4

1)

server.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20001
buffersize = 1024
msgFromServer = "Hello UDP Client"
bytesToSend = str.encode(msgFromServer)

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientmsg = "Number from client: {}".format(message.decode())
    clientIP = "client IP address: {}".format(address)

    print(clientmsg)
    print(clientIP)

    # sending reply to client
    UDPServerSocket.sendto(bytesToSend, address)

```

Figure 4.1.1: server code

client.py > ...

```python
import socket

serverAddressPort = ("127.0.0.1",20001)
buffersize = 1024

# create UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)

# getting number from user
msgFromClient = input("Enter a Number: ")
bytesToSend = str.encode(msgFromClient)

# Send to server using created UDP socket
UDPClientSocket.sendto(bytesToSend,serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(buffersize)
msg = "Message from server: "+msgFromServer[0].decode()

print(msg)

```

Figure 4.1.2: client code

Figure 4.1.3: both terminals after running server code

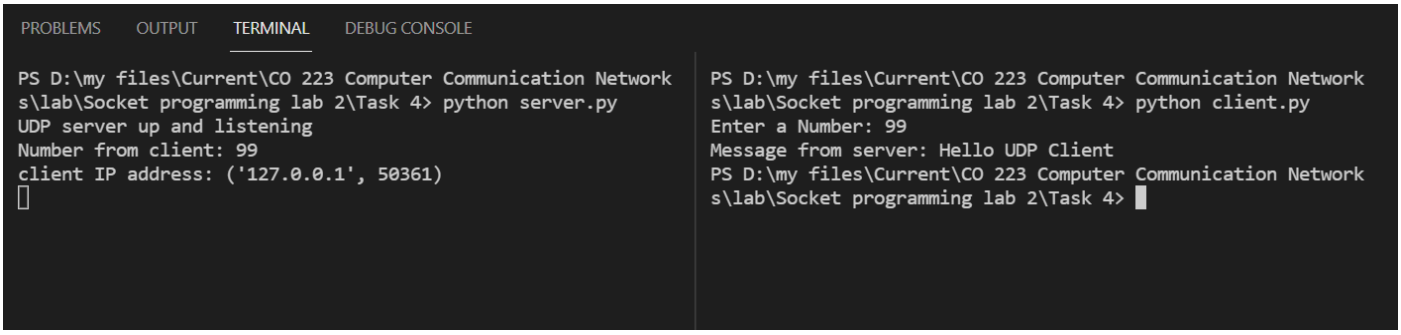

Figure 4.1.4: both terminals after user sending a number

2)

server2.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20001
buffersize = 1024

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]

    clientmsg = "Number from client: {}".format(message.decode())

    print(clientmsg)

    # sending reply to client
    UDPServerSocket.sendto(message, address)
```

Figure 4.2.1: server code

client2.py > ...

```python
import socket

serverAddressPort = ("127.0.0.1",20001)
buffersize = 1024

# create UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)

# getting number from user
n =int(input("Enter a Number: "))

# loop to send numbers to the server
for i in range(0,n):
    bytesToSend = str.encode(str(i+1))

    # Send to server using created UDP socket
    UDPClientSocket.sendto(bytesToSend,serverAddressPort)

# loop to recceive messages from the server
while (True):
    msgFromServer = UDPClientSocket.recvfrom(buffersize)
    msg = "Number from server: "+msgFromServer[0].decode()

    print(msg)


```

Figure 4.2.2: client code

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 4> python server2.py
UDP server up and listening
```

```
PS D:\my files\Current\CO 223 Computer Communication Networks\lab\Socket programming lab 2\Task 4>
```

Figure 4.2.3: both terminals after running server code

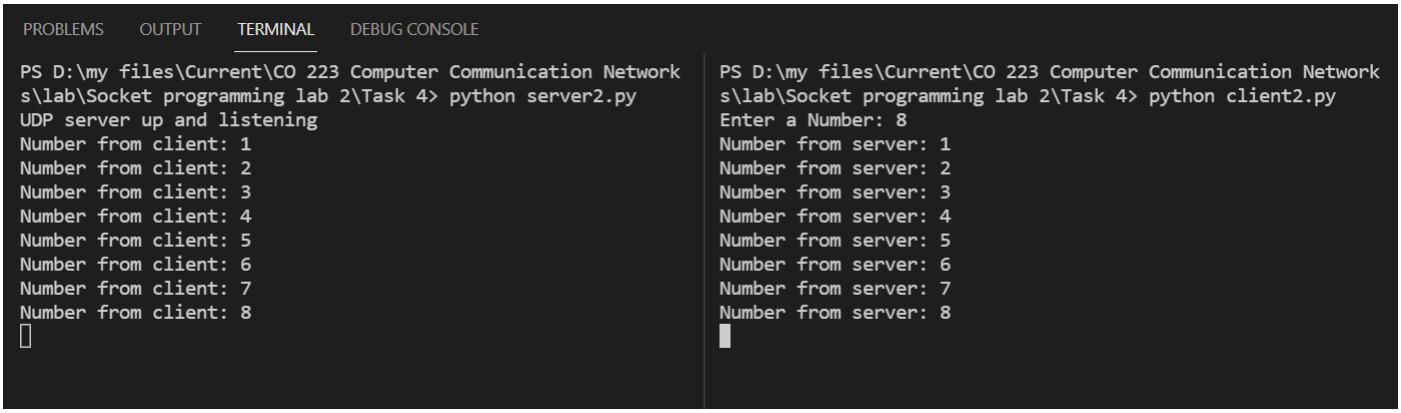

Figure 4.2.4: both terminal after entering a number by client

3) Same code used as the above

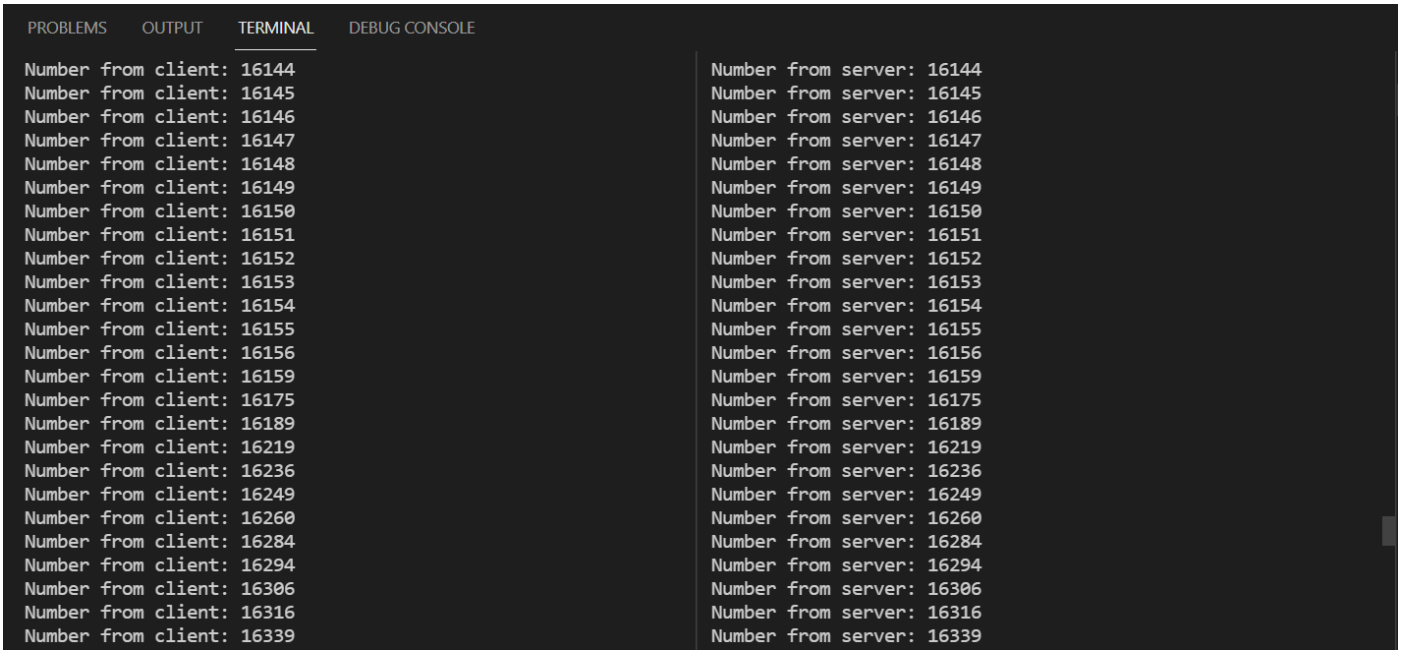

Figure 4.3.1: both terminals after entering a larger n value (after 16156 packet loss can see)

- UDP isn't reliable protocol. The UDP does not require a connection, and it will not resend data packets if there are errors. On every UDP socket, there's a "socket send buffer" that we put packets into. So, if we have a network card that's too slow or something, it's possible that it will not be able to send the packets as fast as we put them in. Therefore, packets will loss.

4)

server4.py > ...

```python
import socket

localIP = "127.0.0.1"
localPort = 20001
buffersize = 1024

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,type = socket.SOCK_DGRAM)

# changing the boffer size
UDPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 200000)

# Bind to address and IP
UDPServerSocket.bind((localIP,localPort))

print("UDP server up and listening")

while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(buffersize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]

    clientmsg = "Number from client: {}".format(message.decode())

    print(clientmsg)

    # sending reply to client
    UDPServerSocket.sendto(message, address)

```

Figure 4.4.1: server code

client4.py > ...

```python
import socket

serverAddressPort = ("127.0.0.1",20001)
buffersize = 1024

# create UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET,type=socket.SOCK_DGRAM)

# getting number from user
n =int(input("Enter a Number: "))

# loop to send numbers to the server
for i in range(0,n):
    bytesToSend = str.encode(str(i+1))

    # Send to server using created UDP socket
    UDPClientSocket.sendto(bytesToSend,serverAddressPort)

# loop to recceive messages from the server
while (True):
    msgFromServer = UDPClientSocket.recvfrom(buffersize)
    msg = "Number from server: "+msgFromServer[0].decode()

    print(msg)

```

Figure 4.4.2: client code

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Number from client: 19989
Number from client: 19990
Number from client: 19991
Number from client: 19992
Number from client: 19993
Number from client: 19994
Number from client: 19995
Number from client: 19996
Number from client: 19997
Number from client: 19998
Number from client: 19999
Number from client: 20000
```

```
Number from server: 19989
Number from server: 19990
Number from server: 19991
Number from server: 19992
Number from server: 19993
Number from server: 19994
Number from server: 19995
Number from server: 19996
Number from server: 19997
Number from server: 19998
Number from server: 19999
Number from server: 20000
```

Figure 4.4.3: end of the both terminals after entering n as 20000

- By using socket.setsockopt() we can increase receive buffer size. By increasing buffer size, we can reduce UDP packet loss. That's why our code is working fine without showing any data loss.