National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

**Department of Electrical Engineering**

**Faculty Member: Sir Haasaan Khaliq** **Dated: 2/24/2023**

**Semester:6th** **Section: C**

EE-357 Computer and Communication Networks

Experiment – 4

# Network Programming in python

| | | **PLO5/ CLO4** | **PLO9/ CLO5** |
|---|---|---|---|
| **Name** | **Reg. No** | **Modern Tool Usage**<br><br>**10 Marks** | **Individual and Team Work**<br><br>**5 Marks** |
| **Muhammad Ahmed Mohsin** | **333060** | | |
| **Amina Bashir** | **343489** | | |
| **Imran Haider** | **332569** | | |

## 1 TABLE OF CONTENTS

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

**EXPERIMENT NO 5**

# Network Programming

## 2 OBJECTIVE OF THIS LAB:

After this lab, the students should be able to

- Explain the concepts of client server communication
- Setup client/server communication
- Use the sockets interface of Python programming language
- Implement simple Client / Server applications using UDP and TCP

## 3 INSTRUCTIONS:

- Read carefully before starting the lab.
- These exercises are to be done individually.
- To obtain credit for this lab, you are supposed to complete the lab tasks and provide the source codes and the screen shot of your output in this document (please use red font color) and upload the completed document to your course's LMS site.
- Avoid plagiarism by copying from the Internet or from your peers. You may refer to source/ text but you must paraphrase the original work.

## 4 BACKGROUND:

### 4.1 APPLICATION PROGRAMMING INTERFACE:

An application programming interface (API) is a specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

### 4.2 NETWORK APPLICATION PROGRAMMING INTERFACE:

The place to start when implementing a network application is the interface exported by network'. Generally, all operating systems provide an interface to its networking sub system. This interface is called as the 'Network Application Programming Interface' (Network API) or socket interface.

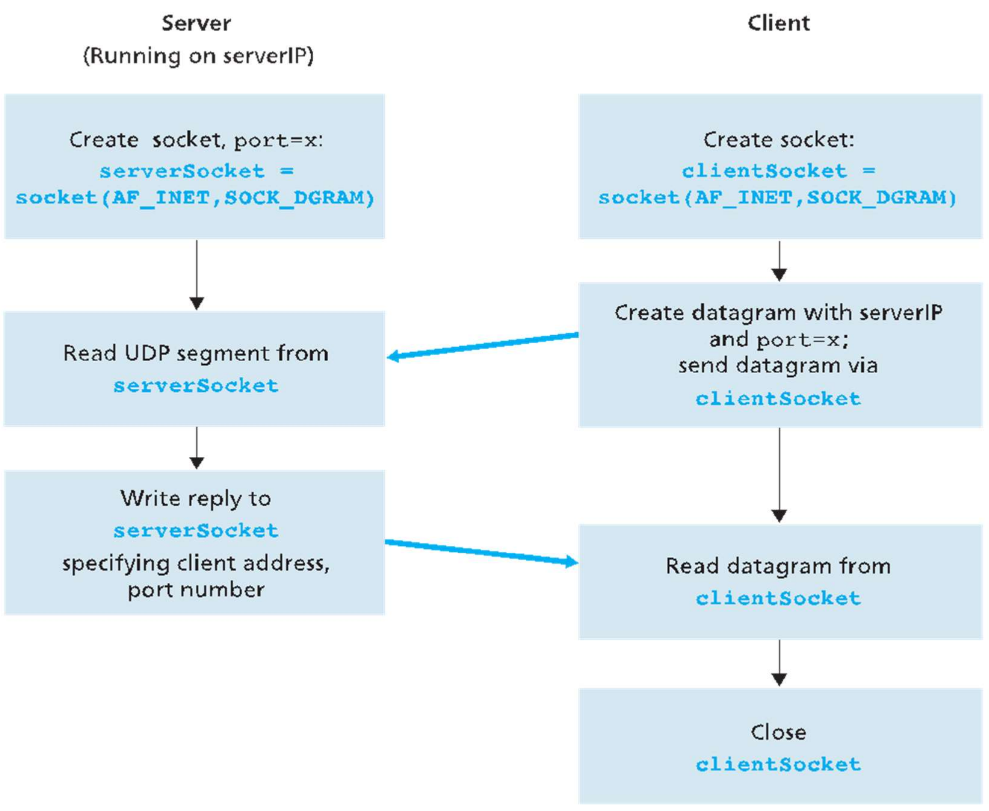
### 4.3 NETWORK SOCKETS:

A **network socket** is an endpoint of an inter-process communication flow across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are **Internet sockets**.

The socket is a special file in UNIX. The socket interface defines various operations for creating a socket, attaching the socket to the network, sending/receiving messages through the socket and so on. Any application uses a socket primitive to establish a connection between client and server.

A **socket address** is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, Internet sockets deliver incoming data packets to the appropriate application process or thread.

### 4.4 Client-Server Socket Programming


Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket = socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket = socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP and port=x; send datagram via clientSocket
Read UDP segment from serverSocket
Write reply to serverSocket specifying client address, port number
Read datagram from clientSocket
Close clientSocket

## 5 UDP Programming:

### 5.1 Client Program:

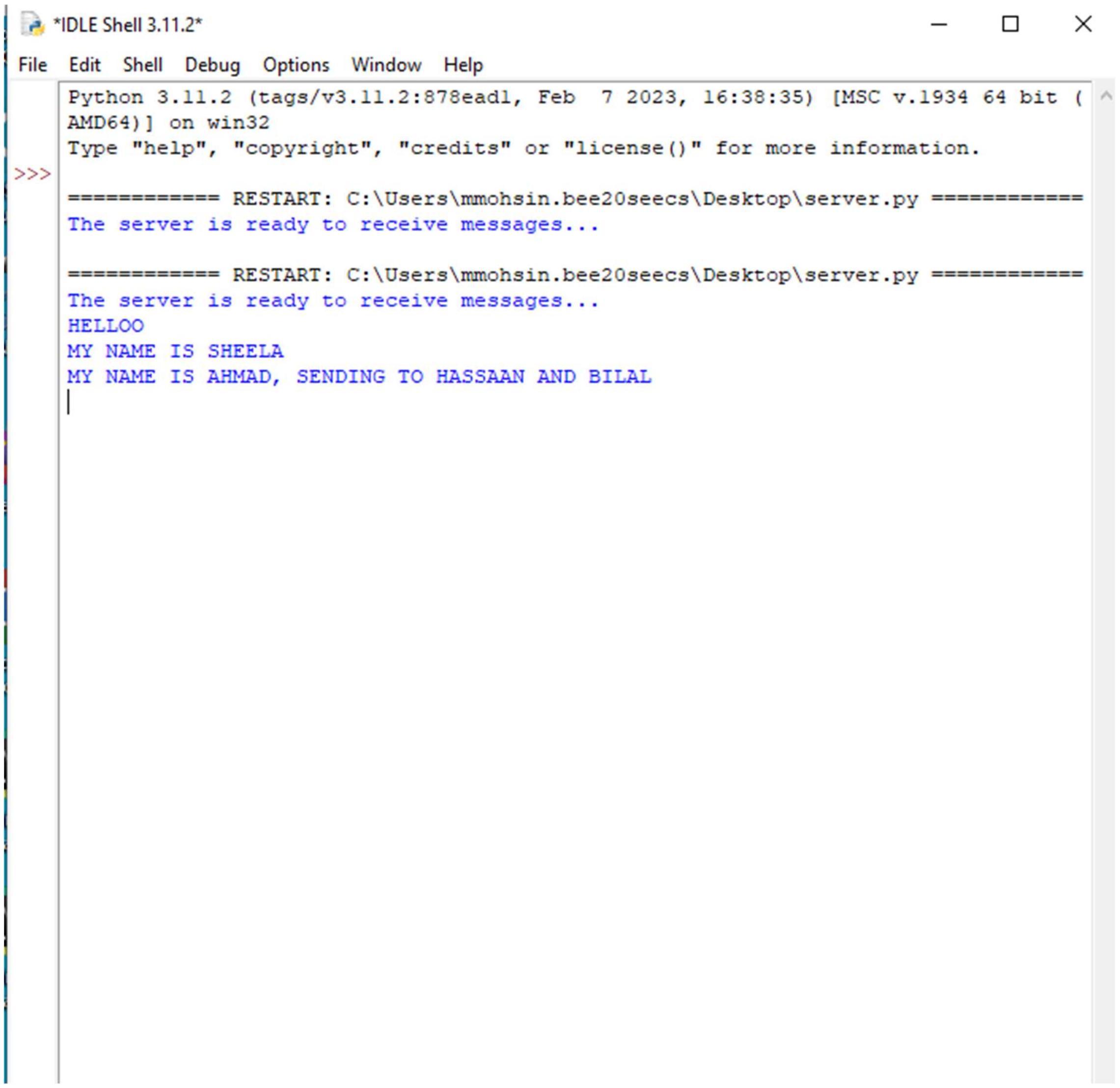
The client program is called UDPClient.py, and the server program is called UDPServer.py. In order to emphasize the key issues, we intentionally provide code that is minimal. “Good code” would certainly have a few more auxiliary lines, in particular for handling error cases. For this application, we have arbitrarily chosen 12000 for the server port number.

#### 5.1.1 Code:

```
from socket import *

serverIP = 'hostname' # replace with IP address of the server
serverPort = 25000 # port where server is listening
```

```
clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input('Input lowercase sentence:').encode('utf-8')
clientSocket.sendto(message, (serverIP, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode('utf-8')) # print the received message

clientSocket.close() # close the socket
```

### 5.1.2 Screenshot:



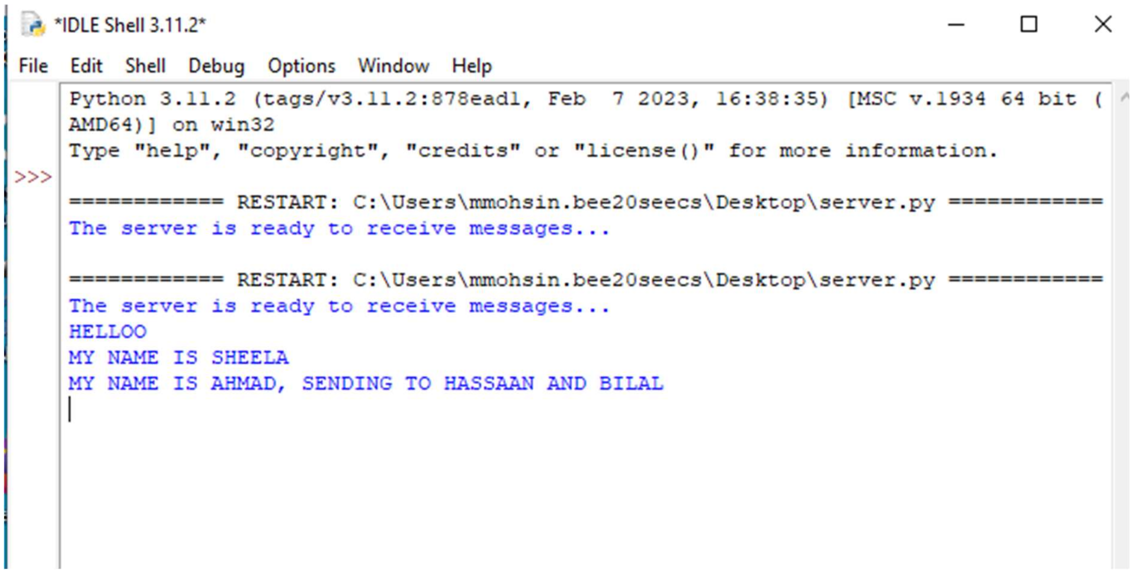
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\server.py ============
The server is ready to receive messages...

============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\server.py ============
The server is ready to receive messages...
HELLOO
MY NAME IS SHEELA
MY NAME IS AHMAD, SENDING TO HASSAAN AND BILAL
```

## 5.2 Server Program:

### 5.2.1 Code:

```
from socket import *
serverPort = 25000 # port where the server will listen for incoming messages
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print('The server is ready to receive messages...')
while True:
   message, clientAddress = serverSocket.recvfrom(2048)
   modifiedMessage = message.decode('utf-8').upper().encode('utf-8')
   serverSocket.sendto(modifiedMessage, clientAddress)
```

### 5.2.2 Screenshot



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\server.py ============
The server is ready to receive messages...

============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\server.py ============
The server is ready to receive messages...
HELLOO
MY NAME IS SHEELA
MY NAME IS AHMAD, SENDING TO HASSAAN AND BILAL
```

# 6 LAB TASK 1

Modify the UDPClient program such that the UDPClient is able to calculate the Application level Round Trip Time (RTT) for the communication between the Client and the Server. The Client should also print the time when Request is send and time when the Reply is received in human readable form.

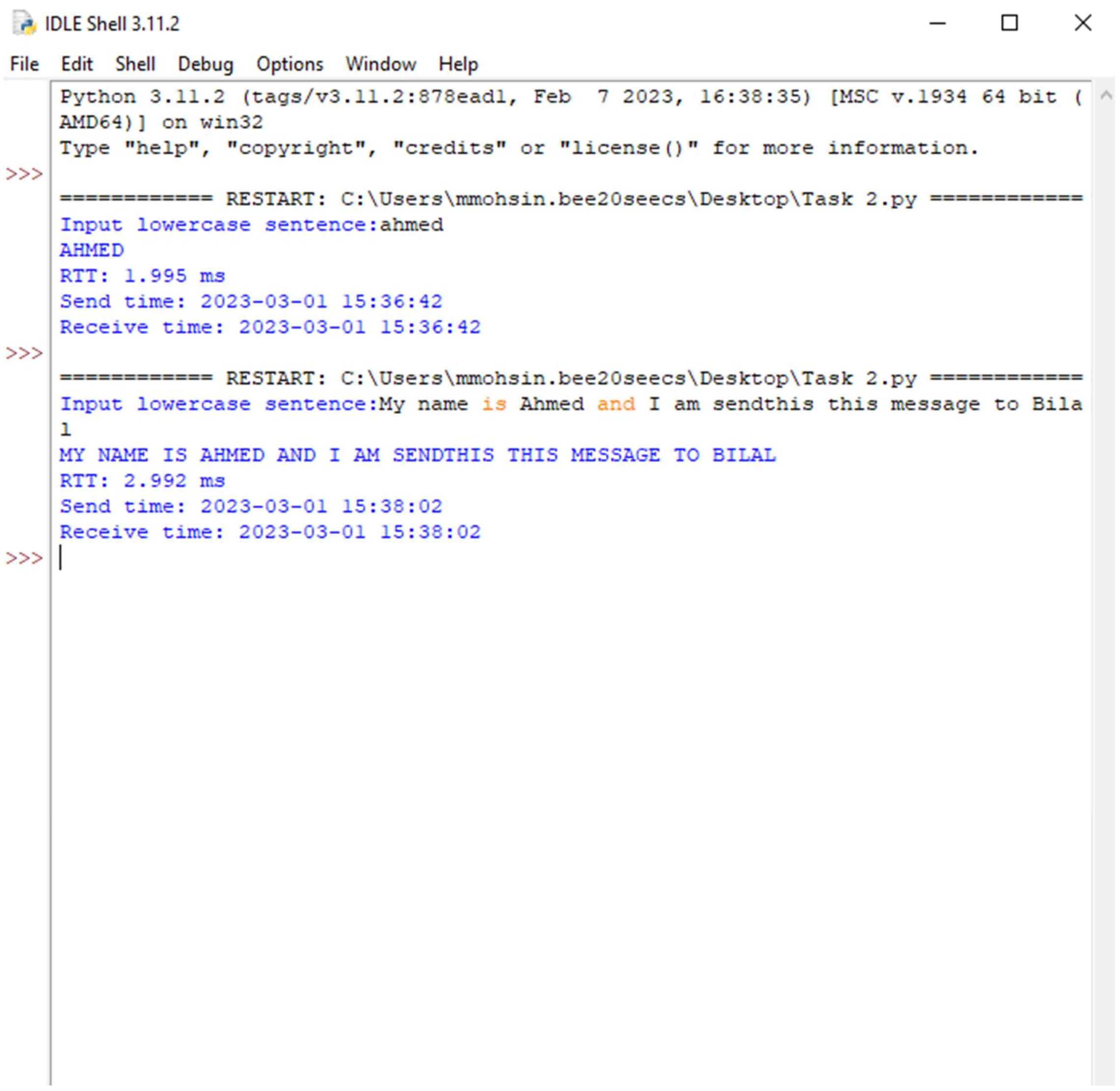
### 6.1.1 Code:

```
from socket import *
import time

serverIP = 'hostname' # replace with IP address of the server
serverPort = 25000 # port where server is listening

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input('Input lowercase sentence:').encode('utf-8')
send_time = time.time() # record the time when the request is sent
clientSocket.sendto(message, (serverIP, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
recv_time = time.time() # record the time when the reply is received

print(modifiedMessage.decode('utf-8')) # print the received message

# calculate the Application level Round Trip Time (RTT) and print it
rtt = (recv_time - send_time) * 1000 # convert to milliseconds
print(f"RTT: {rtt:.3f} ms")

# print the send and receive times in human-readable form
print(f"Send time: {time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime(send_time))}")
```

```
print(f"Receive time: {time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(recv_time))}")

clientSocket.close() # close the socket
```

.

### 6.1.2 Screenshot



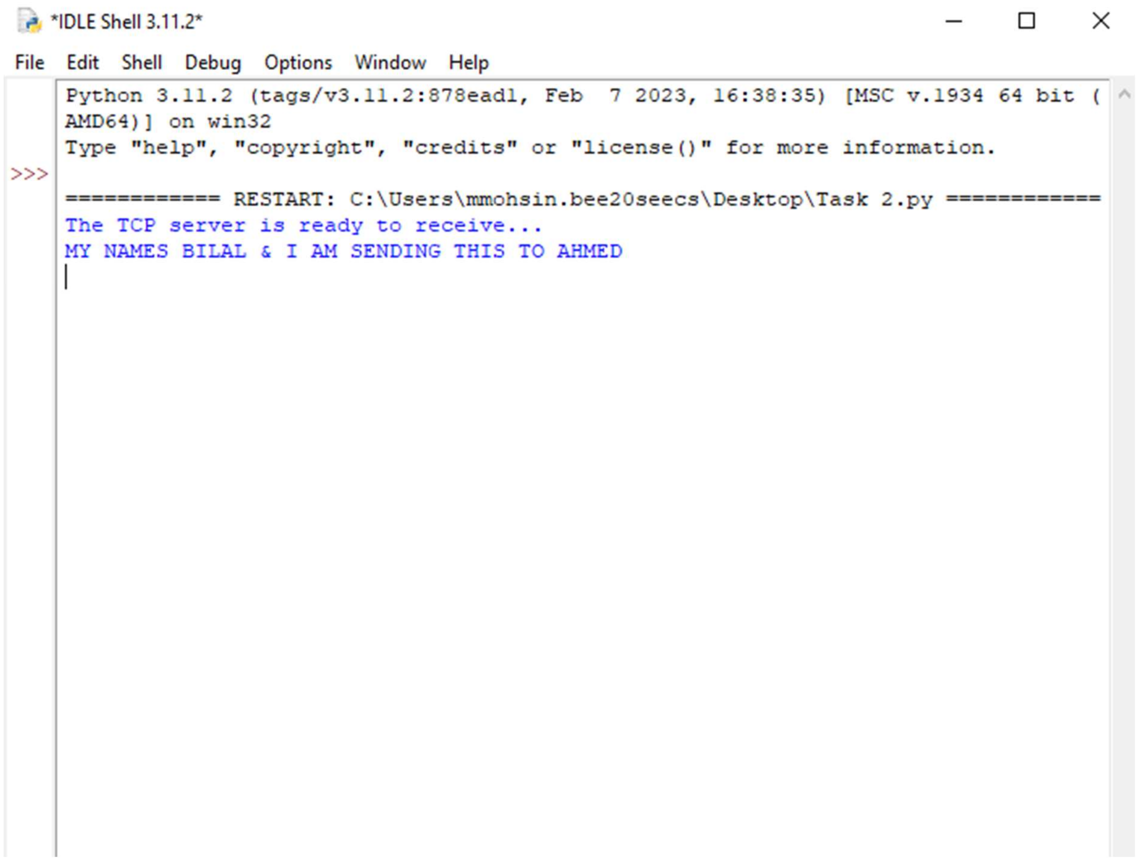
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\Task 2.py ============
Input lowercase sentence:ahmed
AHMED
RTT: 1.995 ms
Send time: 2023-03-01 15:36:42
Receive time: 2023-03-01 15:36:42
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\Task 2.py ============
Input lowercase sentence:My name is Ahmed and I am sendthis this message to Bila
l
MY NAME IS AHMED AND I AM SENDTHIS THIS MESSAGE TO BILAL
RTT: 2.992 ms
Send time: 2023-03-01 15:38:02
Receive time: 2023-03-01 15:38:02
>>>
```

# 7 LAB TASK 2

## 7.1 TCP CLIENT CODE:

```
from socket import *

# Define server host name and port number
serverName = 'localhost'
serverPort = 12000

# Create client socket and initiate TCP connection with server
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

# Get sentence input from user and send it to the server
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())

# Receive modified sentence from the server and print it
modifiedSentence = clientSocket.recv(1024).decode()
print('From Server:', modifiedSentence)

# Close the client socket
clientSocket.close()
```

## 7.2 SCREENSHOT:



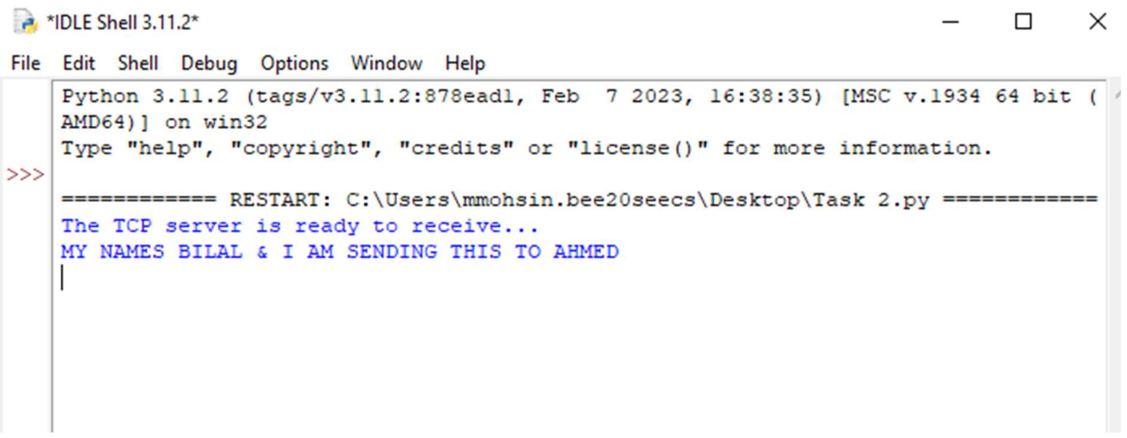
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\Task 2.py ============
The TCP server is ready to receive...
MY NAMES BILAL & I AM SENDING THIS TO AHMED
```

## 7.3 TCP SERVER CODE:

```
from socket import *

# Define server port number and create server socket
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))

# Listen for incoming TCP connection requests
serverSocket.listen(1)
print('The server is ready to receive...')
```

```
while True:
    # Accept a new TCP connection request
    connectionSocket, addr = serverSocket.accept()

    # Receive sentence from the client
    sentence = connectionSocket.recv(1024).decode()

    # Capitalize the sentence
    capitalizedSentence = sentence.upper()

    # Send the modified sentence back to the client
    connectionSocket.send(capitalizedSentence.encode())

    # Close the connection socket
    connectionSocket.close()
```

### 7.3.1 Screenshot:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\Task 2.py ============
The TCP server is ready to receive...
MY NAMES BILAL & I AM SENDING THIS TO AHMED
```

## 8 LAB TASK 3

Modify the TCPClient program such that the TCPClient is able to calculate the Application level Round Trip Time (RTT) for the communication between the Client and the Server. The Client should also print the time when connection request is send and time when the Reply (capitalized words) is received in human readable form.

### 8.1 CODE

```
from socket import *
from datetime import datetime

# Define server host name and port number
serverName = 'localhost'
serverPort = 12000

# Create client socket and initiate TCP connection with server
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

# Get sentence input from user and send it to the server, measuring the send
time
sentence = input('Input lowercase sentence:')
sendTime = datetime.now()
clientSocket.send(sentence.encode())

# Receive modified sentence from the server and print it, measuring the receive
time
modifiedSentence = clientSocket.recv(1024).decode()
receiveTime = datetime.now()
print('From Server:', modifiedSentence)

# Calculate the Round Trip Time (RTT) and print it
rtt = receiveTime - sendTime
print('RTT:', rtt.total_seconds(), 'seconds')
```

```
# Print the send time and receive time in human-readable form
print('Send Time:', sendTime.strftime('%Y-%m-%d %H:%M:%S.%f'))
print('Receive Time:', receiveTime.strftime('%Y-%m-%d %H:%M:%S.%f'))

# Close the client socket
clientSocket.close()
```

## 9 LAB TASK 4

Compare the values of the RTT for both the UDP and TCP. Which one has got higher RTT? Why?

### 9.1 ANSWER

RTT (Round-Trip Time) is a metric that measures the time taken for a packet to travel from the sender to the receiver and back again. The RTT for TCP and UDP differ due to the congestion control mechanisms employed by each protocol. TCP is a reliable and connection-oriented protocol that establishes a virtual connection between two endpoints and uses congestion control to ensure guaranteed delivery of data. The congestion control algorithm in TCP monitors the network and adjusts the sending rate to avoid congestion, which can increase the RTT. In contrast, UDP is a connectionless protocol that does not provide congestion control or guarantee the delivery of data. UDP sends packets as quickly as possible without waiting for acknowledgment, resulting in a lower RTT. However, this can lead to higher packet loss and lower reliability compared to TCP. Therefore, the RTT for TCP and UDP differ due to the trade-off between reliability and speed in network communication.

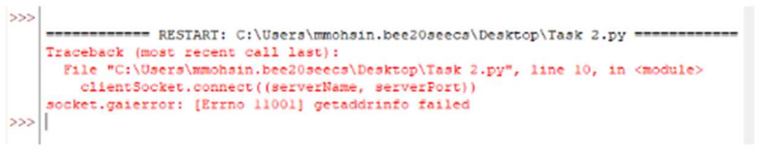
## 10 LAB TASK 5

What happens when your client (both UDP and TCP) tries to send data to a non-existent server

### 10.1 ANSWER:

When a client (both UDP and TCP) tries to send data to a non-existent server, the client will receive an error message indicating that the destination server is unreachable or that the connection has been refused. In TCP, the client will send a SYN packet to initiate a connection with the server, but since the server does not exist, it will not respond with a SYN-ACK packet. The client will then receive a "connection refused" error message. In UDP, since there is no connection setup, the client will simply receive an error message indicating that the destination server is unreachable.

#### 10.1.1 Screenshot

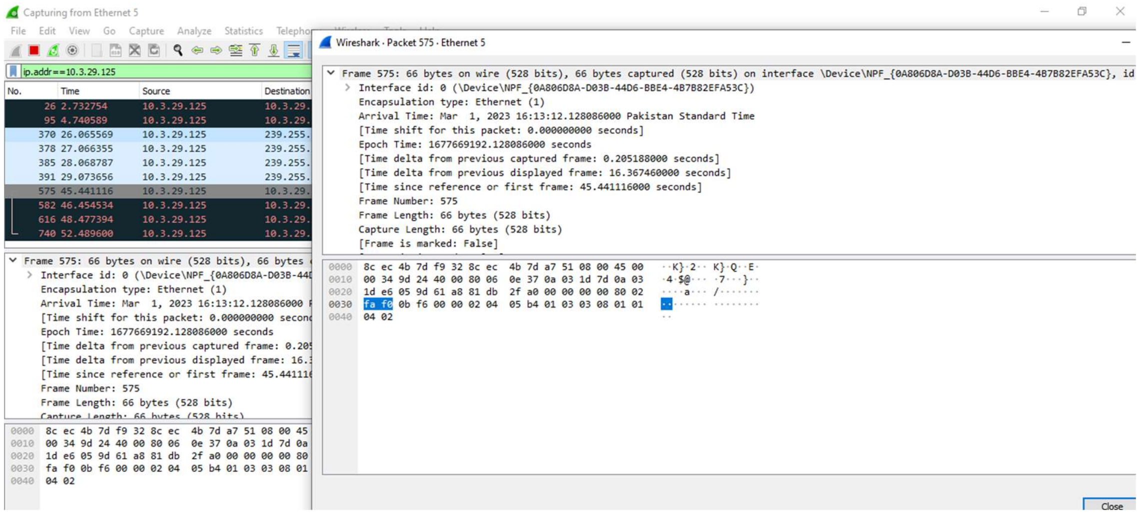


```
>>>
============ RESTART: C:\Users\mmohsin.bee20seecs\Desktop\Task 2.py ============
Traceback (most recent call last):
  File "C:\Users\mmohsin.bee20seecs\Desktop\Task 2.py", line 10, in <module>
    clientSocket.connect((serverName, serverPort))
socket.gaierror: [Errno 11001] getaddrinfo failed
>>>
```

## 11 LAB TASK 6

Trace and identify the packets for both TCP and UDP client server communication using Wireshark and highlight the difference.(Note: you do not need to write details about each difference)

## 11.1 UDP Server



Capturing from Ethernet 5

ip.addr==10.3.29.125

```
No.   Time          Source         Destination
 26   2.732754      10.3.29.125    10.3.29.
 95   4.740589      10.3.29.125    10.3.29.
370   26.065569     10.3.29.125    239.255.
378   27.066355     10.3.29.125    239.255.
385   28.068787     10.3.29.125    239.255.
391   29.073656     10.3.29.125    239.255.
575   45.441116     10.3.29.125    10.3.29.
582   46.454534     10.3.29.125    10.3.29.
616   48.477394     10.3.29.125    10.3.29.
740   52.489600     10.3.29.125    10.3.29.
```

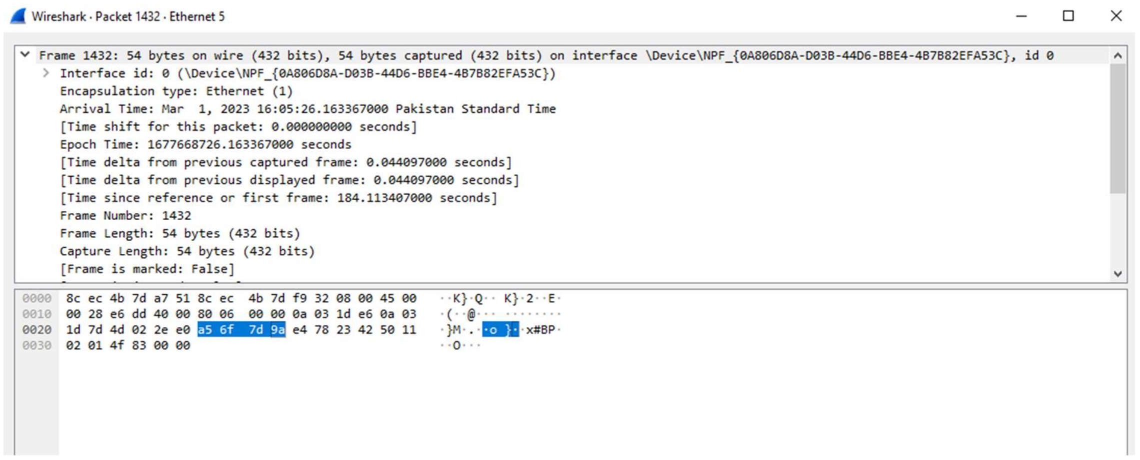
```
Frame 575: 66 bytes on wire (528 bits), 66 bytes
  Interface id: 0 (\Device\NPF_{0A806D8A-D03B-44D
  Encapsulation type: Ethernet (1)
  Arrival Time: Mar  1, 2023 16:13:12.128086000 P
  [Time shift for this packet: 0.000000000 second
  Epoch Time: 1677669192.128086000 seconds
  [Time delta from previous captured frame: 0.205
  [Time delta from previous displayed frame: 16.3
  [Time since reference or first frame: 45.441116
  Frame Number: 575
  Frame Length: 66 bytes (528 bits)
  Capture Length: 66 bytes (528 bits)

0000  8c ec 4b 7d f9 32 8c ec  4b 7d a7 51 08 00 45
0010  00 34 9d 24 40 00 80 06  0e 37 0a 03 1d 7d 0a
0020  1d e6 05 9d 61 a8 81 db  2f a0 00 00 00 00 80
0030  fa f0 0b f6 00 00 02 04  05 b4 01 03 03 08 01
0040  04 02
```

Wireshark · Packet 575 · Ethernet 5

```
Frame 575: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{0A806D8A-D03B-44D6-BBE4-4B7B82EFA53C}, id
  Interface id: 0 (\Device\NPF_{0A806D8A-D03B-44D6-BBE4-4B7B82EFA53C})
  Encapsulation type: Ethernet (1)
  Arrival Time: Mar  1, 2023 16:13:12.128086000 Pakistan Standard Time
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1677669192.128086000 seconds
  [Time delta from previous captured frame: 0.205188000 seconds]
  [Time delta from previous displayed frame: 16.367460000 seconds]
  [Time since reference or first frame: 45.441116000 seconds]
  Frame Number: 575
  Frame Length: 66 bytes (528 bits)
  Capture Length: 66 bytes (528 bits)
  [Frame is marked: False]

0000  8c ec 4b 7d f9 32 8c ec  4b 7d a7 51 08 00 45 00   ··K}·2·· K}·Q··E·
0010  00 34 9d 24 40 00 80 06  0e 37 0a 03 1d 7d 0a 03   ·4·$@··· ·7···}··
0020  1d e6 05 9d 61 a8 81 db  2f a0 00 00 00 00 80 02   ····a··· /·······
0030  fa f0 0b f6 00 00 02 04  05 b4 01 03 03 08 01 01   ········ ········
0040  04 02                                              ··
```

Close

## 11.2 TCP server:



Wireshark · Packet 1432 · Ethernet 5

```
Frame 1432: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{0A806D8A-D03B-44D6-BBE4-4B7B82EFA53C}, id 0
  Interface id: 0 (\Device\NPF_{0A806D8A-D03B-44D6-BBE4-4B7B82EFA53C})
  Encapsulation type: Ethernet (1)
  Arrival Time: Mar  1, 2023 16:05:26.163367000 Pakistan Standard Time
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1677668726.163367000 seconds
  [Time delta from previous captured frame: 0.044097000 seconds]
  [Time delta from previous displayed frame: 0.044097000 seconds]
  [Time since reference or first frame: 184.113407000 seconds]
  Frame Number: 1432
  Frame Length: 54 bytes (432 bits)
  Capture Length: 54 bytes (432 bits)
  [Frame is marked: False]

0000  8c ec 4b 7d a7 51 8c ec  4b 7d f9 32 08 00 45 00   ··K}·Q·· K}·2··E·
0010  00 28 e6 dd 40 00 80 06  00 00 0a 03 1d e6 0a 03   ·(··@··· ········
0020  1d 7d 4d 02 2e e0 a5 6f  7d 9a e4 78 23 42 50 11   ·}M·.··o }··x#BP·
0030  02 01 4f 83 00 00                                  ··O···
```

## 12 Conclusion:

In conclusion, this lab provided an opportunity to design and implement the UDP and TCP protocols using Python programming language. Through this exercise, we gained a deeper understanding of how these protocols operate at the network level and the differences between them in terms of reliability and speed. By implementing these protocols ourselves, we were able to see how the network stack handles data transmission and how packet loss and congestion can impact network performance. Overall, this lab was a valuable learning experience in networking and provided practical experience in designing and implementing network protocols.