**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** | **Individual and Team Work** |
|  |  | **10 Marks** | **5 Marks** |
| **Muhammad Ahmed Mohsin** | **333060** |  |  |
| **Amina Bashir** | **343489** |  |  |
| **Imran Haider** | **332569** |  |  |

Table of Contents

[2 Objective of this lab: 3](#_Toc128838987)

[3 Instructions: 4](#_Toc128838988)

[4 Background: 4](#_Toc128838989)

[4.1 Application programming interface: 4](#_Toc128838990)

[4.2 Network Application Programming Interface: 4](#_Toc128838991)

[4.3 Network Sockets: 4](#_Toc128838992)

[4.4 Client-Server Socket Programming 5](#_Toc128838993)

[5 UDP Programming: 5](#_Toc128838994)

[5.1 Client Program: 5](#_Toc128838995)

[5.1.1 Code: 6](#_Toc128838996)

[5.1.2 Screenshot: 7](#_Toc128838997)

[5.2 Server Program: 7](#_Toc128838998)

[5.2.1 Code: 7](#_Toc128838999)

[5.2.2 Screenshot 8](#_Toc128839000)

[6 Lab Task 1 8](#_Toc128839001)

[6.1.1 Code: 9](#_Toc128839002)

[6.1.2 Screenshot 10](#_Toc128839003)

[7 Lab task 2 10](#_Toc128839004)

[7.1 TCP Client Code: 11](#_Toc128839005)

[7.2 TCP Server Code: 12](#_Toc128839006)

[7.2.1 Screenshot: 13](#_Toc128839007)

[8 Lab Task 3 14](#_Toc128839008)

[8.1 Code 14](#_Toc128839009)

[9 Lab Task 4 15](#_Toc128839010)

[9.1 Answer 15](#_Toc128839011)

[10 Lab Task 5 16](#_Toc128839012)

[10.1 Answer: 16](#_Toc128839013)

[10.1.1 Screenshot 16](#_Toc128839014)

[11 Lab task 6 16](#_Toc128839015)

[11.1 UDP Server 17](#_Toc128839016)

[11.2 TCP server: 17](#_Toc128839017)

[12 Conclusion: 17](#_Toc128839018)

**EXPERIMENT NO 5**

**Network Programming**

# 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

# 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.

# Background:

## 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](http://en.wikipedia.org/wiki/Subroutine), [data structures](http://en.wikipedia.org/wiki/Data_structure), [object classes](http://en.wikipedia.org/wiki/Class_(computer_programming)), and variables.

## 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.

## Network Sockets:

A **network socket** is an endpoint of an [inter-process](http://en.wikipedia.org/wiki/Inter-process_communication) [communication flow](http://en.wikipedia.org/wiki/Communication_flow) across a [computer network](http://en.wikipedia.org/wiki/Computer_network). Today, most communication between computers is based on the [Internet Protocol](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/IP_address) and a [port number](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Process_(computing)) or [thread](http://en.wikipedia.org/wiki/Thread_(computer_science)).

## Client-Server Socket Programming

Graphical user interface, diagram

Description automatically generated

# UDP Programming:

## 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.

### 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

### Screenshot:

Graphical user interface, text, application, email

Description automatically generated

## Server Program:

### 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)

### Screenshot

Graphical user interface, text, application, email

Description automatically generated

# 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.

### 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

.

### Screenshot

Graphical user interface, text, application, email

Description automatically generated

# Lab task 2

## 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

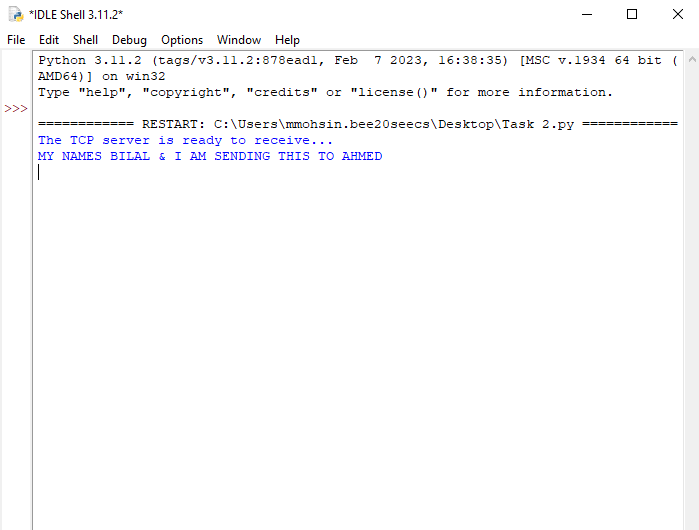
modifiedSentence = clientSocket.recv(1024).decode()

print('From Server:', modifiedSentence)

# Close the client socket

clientSocket.close()

## Screenshot:



## 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()

### Screenshot:

Graphical user interface, text, application, email

Description automatically generated

# 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.

## 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()

# Lab Task 4

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

## 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.

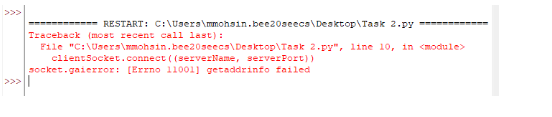
# Lab Task 5

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

## 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.

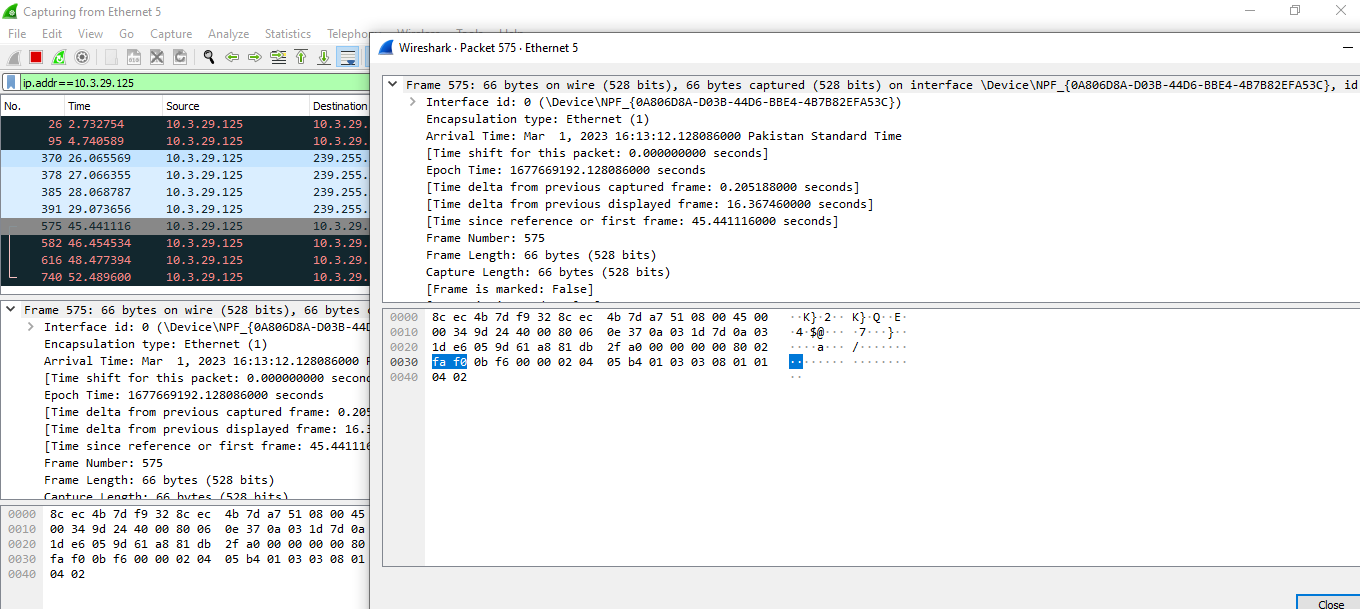
### Screenshot



# 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)

## UDP Server



## TCP server:

Graphical user interface, text, application, email

Description automatically generated

# 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.