A Network based Chat Server and Client

Student Name

Institutional Affiliations

Course Title

Instructor’s Name

April 2022

# DECLARATION

I certify that the entirety of the files shared contains only my own work.

I have not shared the contents of these files with anyone in any form, nor have I obtained or included information from anyone or any extra sources. Any extra information obtained is referenced.

# 1.0 INTRODUCTION

## 1.1 Problem Description.

Communication innovation alludes to every one of the apparatuses used to send, get, and process data. In the present quick environment, effectiveness and comfort are the keys to fruitful communication innovation. There are a few methods of communication conceived Many years prior. The internet eliminates the requirement for communicators to have a different gadget for each unique kind of correspondence innovation. With the Internet, you can do everything in one spot (K. Ratheeswari, 2018).

However, without proper channels, information cannot be sent over the internet. We need both hardware(computer) and a software for this particular process to be successful. Alongside that, we need a sender(client) and A receiver(client). We also need an intermediary(server) to help pass this information to the intended recipient. This is the problem we are trying to solve. (E. M. Rogers, 1986).

## 1.2 Problem Analysis.

For communication to take place, there must be two or more parties who have a will to communicate. There must be an intermediary for communication. In this case, who is going to convey the message to other party on the other end of the internet.

To come up with a fine strategy of accomplishing this task, we will write a python program which will act as an intermediary between two or more clients. This will act as our server which will be forwarding messages from one client to another.

On the other hand, we will code a client program in python. The client will connect to the server, send a message and expect a response in return. The Server will broadcast the message sent by one client to other clients connected to the Network. If another client is online, it will respond back to the first client via the server.

With this information, our Server will be able to provide a Chatroom where people can share information. This server will not provide file upload or download functionalities. It will be limited to text-based conversations between clients.

The server will also allow clients to connect to it provided the ports specified by the client when attempting connection is open. The IP address of the server should be up and running as well to accept the connection.

With all that, there will be a smooth communication between clients as long as there is a single client who shall have initiated a conversation.

The server Will be able to log clients connected to it and tell the administrator how many clients have joined the chat room.

This application will be of the name ‘Message App’.

## 1.3 Graphical Design

For the implementation of the programs graphically, we will use flow chart to imitate how the server will start, how the clients will attempt connecting to the server, server accepting connections and a single client initiating a connection.

### Flowchart 1.3.1

END

Print(“Server\_response”)

Server.broadcast(message)

Print(“waiting for connection”)

Yes

No

Client Connected

While True

Start Server

### 1.3.2 Explanation

The server should be up and running and wait for the incoming connections. This is achieved using while loop.

This loop will run continuously waiting for a client to connect to it, giving ‘forever’ time for any client who would like to start a conversation to start. It will keep the server continuously running.

If a client connects, The Server will take the client message and Broadcast it to the other Client(s). If a client doesn’t connect, server will display a message, “waiting for connections”.

This will save the server from crashing and breaking out of the loop.

Before the server redo the loop, it will print the output to the server logs and then terminates the loop and the process starts again.

This will allow for a hence forth communication between the clients.

### 1.3.3 Application Testing and Analysis.

To test the Application, we will use Dynamic Testing technique.

1. Dynamic Testing
2. Decision Coverage

Decision coverage Test involves evaluating your code base If it is able to correctly make decisions when given sides to weigh from.

In our code, we will use one of the if statements to evaluate our list of IP addresses.

1. Statement Coverage

The statement coverage tries to check and evaluate statements used in a code if they satisfy conditions for execution without throwing any errors. For our evaluation, we will try to concatenate two strings and expect a formatted string at the terminal. This will imitate statements printed at the server terminal as well as the client terminal.

# 2.0 IMPLEMENTATION

### 2.1 IMPLEMENTATION DESCRIPTION

The implementation of the Application involved programming server side to accept connections and broadcast messages to connected clients and programming client side to be able to send and receive messages from other clients via the server. In this case, we ended up with two python files, server.py and client.py.

2.1.1 Server-Side Implementation and Justification.

This section involved coding the server to fulfil its task. The main aim of the server was to accept connection, receive message from a client and broadcast it.

In a file server.py, we begin by importing required libraries socket and thread which will be used to create a connection and create a server session for each client respectively. We will use sys to terminate program execution when necessary.

We then take the IP address and port number from the user, initialize thread count to zero and the number of connected clients to an empty list.

Thread count will help us keep track on how many people are connected to the server and the empty client list will help keep for us the clients joining the meeting.

We then use the imported socket to establish a connection, by specifying ipv4 connection and TCP protocols. We bind the server IP address to the port, if this becomes unsuccessful, we throw an exception. Finally, we call the listen function to accept a maximum of 10 connections from clients. We then print a message to the terminal that the server is waiting for connections.

We then proceed to code the main part of the program where we start by defining an infinite while loop that will keep waiting for connections. When a client attempts a connection, the server will accept the connection and define a tuple holding the client socket and the address (ip address and port number). We then print the client’s IP address and notify that someone joined the chat. We then add this client to the list by .append() method, start a new thread for this client and print what number they are in the chatroom and increment threads by one.

We then start the client thread.

We proceed to code the listening function. This function runs across the started thread for each and every client. It will try to read from the sockets of the connected clients. If a connection from a specific client is broken, we remove this client from the list of available clients.

We then walk through the list and send the message to every client connected.

Finally, we write the code to close all the connections i.e server connection and the client connections.

#### 2.1.2 Client-Side implementation and justification.

In the file client.py, we import the necessary module socket for establishing a connection, Thread for creating session and sys for terminating execution need be.

We ask for user to input the server IP address and port to connect to. We then try to evaluate these inputs. If the inputs adhere to the defined format, we proceed to create a connection else we throw an error.

We initialize a socket object passing ipv4 class and the TCP connection protocol. We try to connect to the IP address at the specified port printing the program status. If the connection is successful, we print OK. If not, we throw an exception.

It will be better if each connected user has a name to identify themselves in the chatroom. So before opening a thread for a client, we ask for their names for identity.

We define a while loop for continuous sending and receiving of messages without having to force client to create a new connection every time. At the start, we give a user opportunity to type their message, and store it in a variable message\_send. If the user wishes to leave the chatroom, they type shutdown and the execution breaks else, we take the user’s name, combine it with the message, encode it and then send to the server and then close the connection.

We then insert a portion of code to initialize client thread after taking their name giving this user a chance to type a message and another at the same time.

Finally, we write a function which will continuosly listen for any incoming messages from the socket. If there is an incoming message, we take that message, decode it and print it on the client console on a new line.

#### 2.1.3 Challenges and Solutions

1. At first, the server could receive connection from more than one client and a message from either of the clients but wasn’t able to broadcast it. This was solved by importing a utilizing threads.
2. The client wasn’t able to connect to the server even after doing everything as expected. Later on, it came to my attention that the client was trying to access a closed port. I changed the port number and had the problem fixed.
3. Trying to connect to a server which is down. It was a mistake solved when I connected to the wrong IP address.

#### 2.1.4 Data Structures.

The following data structures were utilized in the code:

1. Lists.

List was used to keep the number of clients connected to the server.

This is seen in “list\_of\_people = []”

1. Looping Techniques

Looping techniques are implemented to keep the server up and running and allow client connect to the server continuously. This is seen in the While loops used.

1. Tuples

The socket function created by the server is seen to return a tuple of client socket and the client address.

# 3.0 TESTING

## 3.1 Statement Coverage Testing.

In the Statement Coverage Testing, we will write a test to check if the server prints the statements correctly.

#### Expected Result

We expect a string to be printed at the terminal.

#### Actual Result

The following is the result. This is what we expect.

## 3.2 Decision Coverage Testing

We will write a Test here to check on decisions made by the server on the list of IP addresses.

#### Expected Result

We expect a Boolean value to evaluate to True if an IP address entered by a user is in the list of the IP addresses and false if vice versa.

#### Actual Result

The following is the Result. This is what we expect.

# References

K. Ratheeswari (2018). Information communication technology in education. *Journal of Applied and Advanced research*, *3*(1), 45-47.

E. M. Rogers (1986). *Communication technology*. Simon and Schuster.

R. Zurawski (Ed.). (2014). *Industrial communication technology handbook*. CRC Press.