18CSC302J- COMPUTER NETWORKS

MINOR PROJECT REPORT

Submitted by

Harshit Sharma [RA2011030010206]

Sooraj Tomar [RA2011030010224]

Pankaj Bhattarai [RA2011030010230]

Under the Guidance of

Dr. N. Krishnaraj

Associate Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE ENGINEERING

with specialization in Cyber Security

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

November 2022

Simple Chat Room Using TCP/IP

Aim

To write a python code to develop a simple Chat Room using sockets and the TCP/IP Protocol which can be used to chat with our friends.

Figure 1: Chat Room Logo

Introduction

The term chat room, or chatroom, is primarily used to describe any form of synchronous conferencing, occasionally even asynchronous conferencing. The term can thus mean any technology ranging from real-time online chat and online interaction with strangers to fully immersive graphical social environments. In this project however, we shall focus on messaging chat/conversion amongst people. We are to implement the aforementioned chat room using the Python Programming Language with the help of sockets and the TCP/IP Protocol.

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes. Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. A socket is a software abstraction for an input or output medium of communication. Sockets allow communication between processes that lie on the same machine, or on different machines working in diverse environment and even across different continents. A socket is the most vital and fundamental entity. Sockets are the end-point of a two-way communication link. An endpoint is a combination of IP address and the port number. We can use TCP protocol as well as UDP protocol. The basic terms in sockets are: Client and Server.

For Client-Server communication, Sockets are to be configured at the two ends to initiate a connection:

- Listen for incoming messages
- Send the responses at both ends
- Establishing a bidirectional communication.

In the below figure you can see the basic functions done by Client-Server programs using TCP protocol for establishment of a successful connection, communication and finally connection teardown or termination.

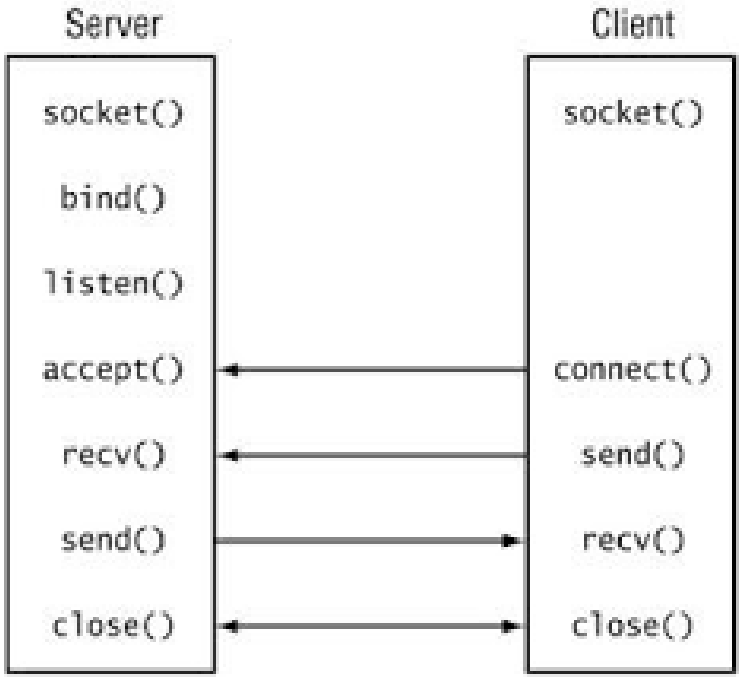

Figure 2: Client – Server Socket Function Flow

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP, which is part of the Transport Layer of the TCP/IP suite. SSL/TLS often runs on top of TCP. TCP is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening (passive open) for connection requests from clients before a connection is established. Three-way handshake (active open), retransmission, and error detection adds to reliability but lengthens latency.

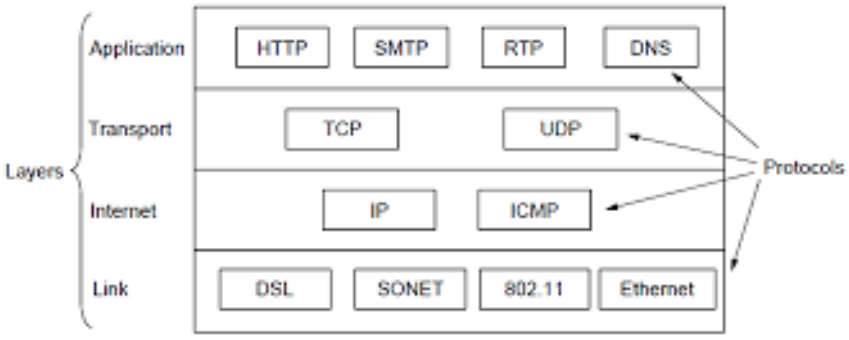

Figure 3: TCP Protocol's Location in TCP/IP Framework

TCP Protocol Header and 3-Way Handshake Mechanism

TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
60	480																																

Figure 4: TCP Header

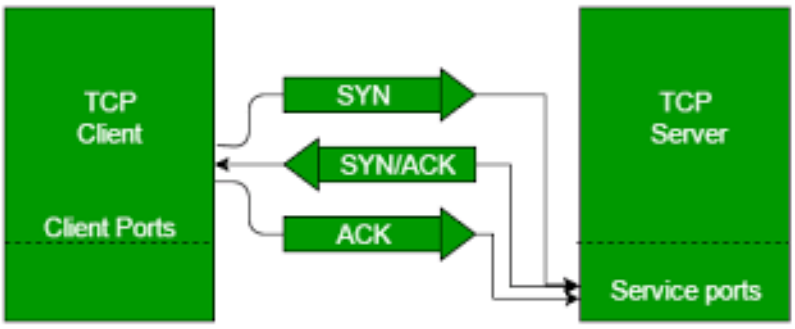

Figure 5: TCP Protocol 3-Way Handshake

Objectives

The objectives of this project are to:

- Create a Chat Room using TCP protocol to chat amongst friends and which also can be used for other professional tasks as well.
- To test our knowledge of Computer Networking concepts regarding sockets and TCP/IP.
- To gain new knowledge so as to grow our professional skill set.

Description

Socket programming

Sockets can be thought of as endpoints in a communication channel that is bi-directional and establishes communication between a server and one or more clients. Here, we set up a socket on each end and allow a client to interact with other clients via the server. The socket on the server side associates itself with some hardware port on the server-side. Any client that has a socket associated with the same port can communicate with the server socket.

Multi-Threading

A thread is a sub-process that runs a set of commands individually of any other thread. So, every time a user connects to the server, a separate thread is created for that user, and communication from the server to the client takes place along individual threads based on socket objects created for the sake of the identity of each client.

We will require two scripts to establish this chat room. One to keep the serving running, and another that every client should run in order to connect to the server.

Server-Side Script

The server-side script will attempt to establish a socket and bind it to an IP address and port specified by the user (windows users might have to make an exception for the specified port number in their firewall settings, or can rather use a port that is already open). The script will then stay open and receive connection requests and will append respective socket objects to a list to keep track of active connections. Every time a user connects, a separate thread will be created for that user. In each thread, the server awaits a message and sends that message to other users currently on the chat. If the server encounters an error while trying to receive a message from a particular thread, it will exit that thread.

Client-Side Script

The client-side script will simply attempt to access the server socket created at the specified IP address and port. Once it connects, it will continuously check as to whether the input comes from the server or from the client, and accordingly redirects output. If the input is from the server, it displays the message on the terminal. If the input is from the user, it sends the message that the user enters to the server for it to be broadcasted to other users. This is the client-side script, that each user must use in order to connect to the server.

Implementation

Code

Chat Server

```
# Python program to implement server side of chat room.
import socket
import select
import sys
from _thread import *
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
if len(sys.argv) != 3:
        print ("Correct usage: script, IP address, port number")
        exit()
IP_address = str(sys.argv[1])
Port = int(sys.argv[2])
server.bind((IP_address, Port))
"""
listens for 100 active connections. This number can be
increased as per convenience.
"""
server.listen(100)
list_of_clients = []
def clientthread(conn, addr):
        # sends a message to the client whose user object is conn
        conn.send("Welcome to this chatroom!".encode())
```

```
        while True:
                try:
                        message = conn.recv(2048).decode()
                        if message:
                                """prints the message and address of the
                                user who just sent the message on the server
                                terminal"""
                                print ("<" + addr[0] + "> " + message)
                                # Calls broadcast function to send message to all
                                message_to_send = "<" + addr[0] + "> " + message
                                broadcast(message_to_send, conn)
                        else:
                                """message may have no content if the connection
                                is broken, in this case we remove the connection"""
                                remove(conn)
                except:
                        continue
"""Using the below function, we broadcast the message to all
clients who's object is not the same as the one sending
the message """
def broadcast(message, connection):
        for clients in list_of_clients:
                if clients!=connection:
                        try:
                                clients.send(message.encode())
                        except:
                                clients.close()
```

```
                                # if the link is broken, we remove the client
                                remove(clients)
"""The following function simply removes the object
from the list that was created at the beginning of
the program"""
def remove(connection):
        if connection in list_of_clients:
                list_of_clients.remove(connection)
while True:
        conn, addr = server.accept()
        """Maintains a list of clients for ease of broadcasting
        a message to all available people in the chatroom"""
        list_of_clients.append(conn)
        # prints the address of the user that just connected
        print (addr[0] + " connected")
        # creates and individual thread for every user
        # that connects
        start_new_thread(clientthread,(conn,addr))
conn.close()
server.close()
```

Chat Client

```
# Python program to implement client side of chat room.
import socket
import select
import sys
```

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
if len(sys.argv) != 3:
        print ("Correct usage: script, IP address, port number")
        exit()
IP_address = str(sys.argv[1])
Port = int(sys.argv[2])
server.connect((IP_address, Port))
while True:
        # maintains a list of possible input streams
        sockets_list = [sys.stdin, server]
        """ There are two possible input situations. Either the user wants to give manual input to send to other people,or the server is sending a message to be printed on the screen. Select returns from sockets_list, the stream that is reader for input. So for example, if the server wants to send a message, then the if condition will hold true below.If the user wants to send a message, the else condition will evaluate as true"""
        read_sockets,write_socket, error_socket = select.select(sockets_list,[],[])
        for socks in read_sockets:
                if socks == server:
                        message = socks.recv(2048)
                        print (message)
                else:
                        message = sys.stdin.readline()
                        server.send(message.encode())
                        sys.stdout.write("<You>")
                        sys.stdout.write(message)
                        sys.stdout.flush()
server.close()
```

Output

Server and Client

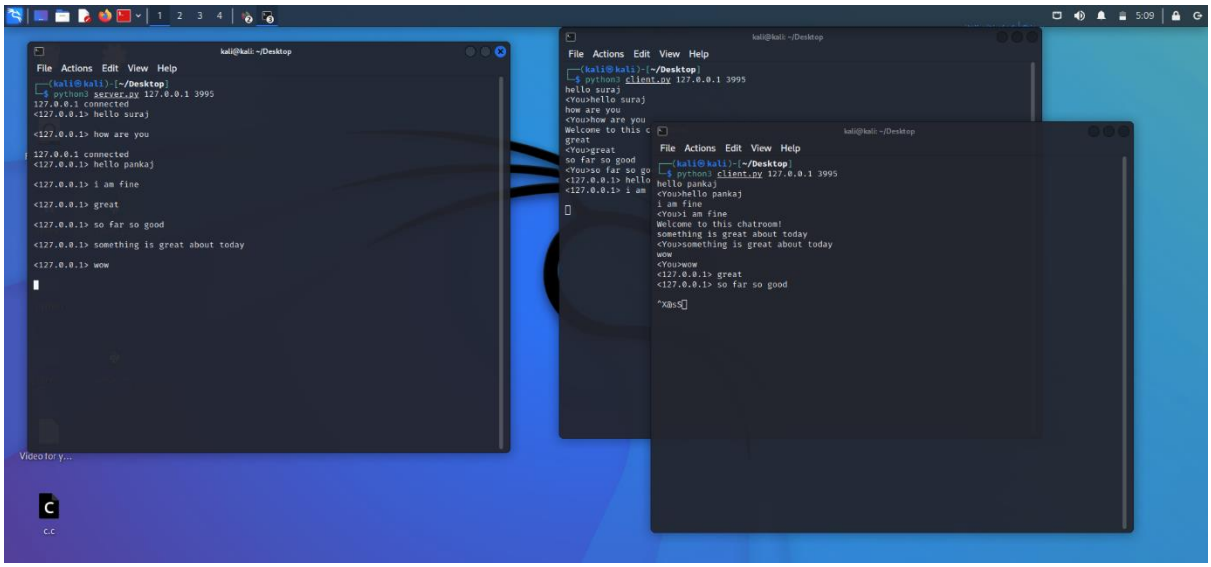

Figure 6: Server and Client 1

Client and Client

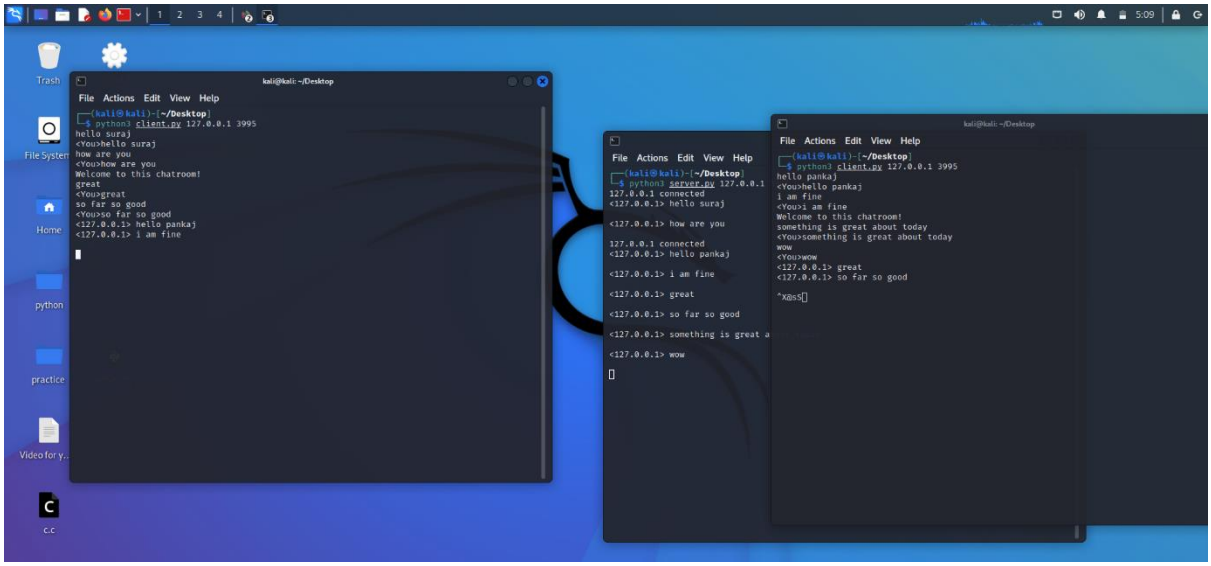

Figure 7: Client 1 and Client 2

Server and Client

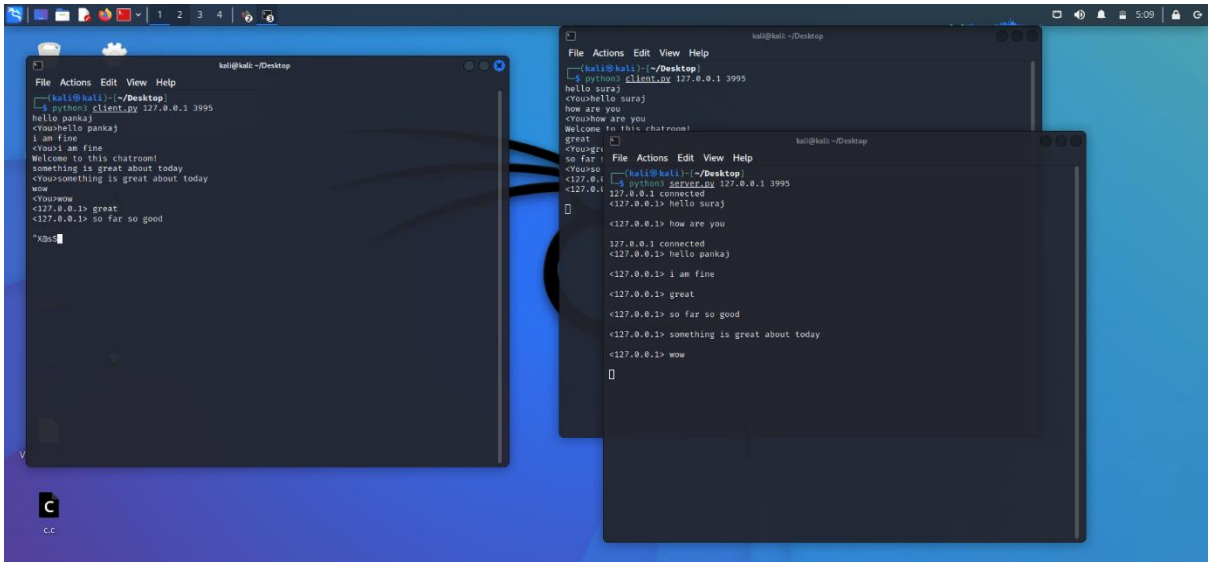

Figure 8: Server and Client 2

Result

This server can be set up on a local area network by choosing any on the computer to be a server node, and using that computer's private IP address as the server IP address. Thus, the Client Server Simple Chat Room Using TCP/IP was successfully implemented. The objectives of this project were met and lots of lessons were learnt. This implementation gave us a hands-on experience over how to clearly implement sockets and use the advantages of the Transmission Control Protocol to our use. Overall, the project can be termed as a success for meeting all its objectives, however, there is always scope for improvement to have bug fixes, make the experience more aesthetic to the user and finally, improve to an extent to even have a commercial release.

References / Websites

[1] TCP/IP Model basics from GeeksforGeeks

[2] Socket Programming in Python GeeksForGeeks

[3] Simple Chat Room Code Reference

[4] Journal on TCP Protocol

[5] Client – Server Model Basics on Wikipedia