

# **VOICE CHAT**

(Server-Client Program Using Python)

## **COMPUTER NETWORKS PROJECT REPORT**

**By**

**Pavithran K (RA1911003010135)**

**Benak Raj D (RA1911003010136)**

**Raj Srivastava (RA1911003010139)**

**Under the guidance of**

**Dr. G. Manju**

**In the partial fulfilment for the course of**

**18CSC302J - COMPUTER NETWORKS**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND**

**TECHNOLOGY**

**Kattankulathur , Chengalpattu**

NOVEMBER 2021

## **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

(Under Section 3 of UGC Act, 1956)

### **BONAFIDE CERTIFICATE**

Certified that this project report "Voice Chat" is the bonafide work of  
**Pavithran K (RA1911003010135) , Benak Raj D (RA1911003010136)**  
**and Raj Srivastava (RA1911003010139)** who carried out the project  
work under my supervision.

#### **SIGNATURE**

Dr.G. Manju  
Associate Professor,  
Computer Science and Engineering  
SRM Institute of Science and Technology  
Potheri , SRM Nagar , Kattankulathur  
TamilNadu-603203

#### **SIGNATURE**

Dr.E. Sasikala  
Course Coordinator  
Associate Professor,  
Data Science and Business Systems  
SRMIST KTR , Potheri  
TamilNadu-603203

## ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable Vice Chancellor Dr. C. MUTHAMIZHCHELVAN, for being the beacon in all our endeavors. We would like to express my warmth of gratitude to our Registrar Dr. S.Ponnusamy, for his encouragement. We express our profound gratitude to our Dean (College of Engineering and Technology) Dr. T. V.Gopal, for bringing out novelty in all executions. We would like to express my heartfelt thanks to Chairperson, School of Computing Dr. Revathi Venkataraman, for imparting confidence to complete my course project. We wish to express my sincere thanks to Course Audit Professor Dr.M.LAKSHMI, Professor and Head, Data Science and Business Systems and Course Coordinator Dr.E. Sasikala, Associate Professor, Data Science and Business Systems for their constant encouragement and support.

We are highly thankful to our Course project Internal guide Dr.G . Manju , Associate Professor , Computer Science and Engineering, for his assistance, timely suggestion and guidance throughout the duration of this course project. We extend my gratitude to HOD, Computer Science and Engineering and my Departmental colleagues for their Support. Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on us to complete our course project.

# Table of Contents

CHAPTERS	CONTENTS
1.	ABSTRACT
2.	INTRODUCTION
3.	REQUIREMENT ANALYSIS
4.	ARCHITECTURE & DESIGN
5.	IMPLEMENTATION
6.	EXPERIMENT RESULTS & ANALYSIS
	6.1. RESULTS
	6.2. RESULT ANALYSIS
	6.3. CONCLUSION & FUTURE WORK27
7.	REFERENCES
8.	APPENDIX

# ABSTRACT

This report presents a detail overview in developing a client-server based voice chat application using socket programming. The application is built using Python and is using UDP datagram. The primary objective of this report is to present the principles behind socket programming and the libraries available for socket programming applications in Python. The main objective of this project is to develop a program that allows multiple people to communicate over the internet using their microphones for voice chat. This application enables the multiple user to communicate each other hassle-free. It also enables the user to send high quality audio and also to record calls if needed. We can also explore Dolby.io Spatial Audio feature and play with it .An encrypted voice chat app, written in Python, that uses web-sockets to send data between a multi-threaded client and server.

# INTRODUCTION

Client-server program is no longer a foreign concept in computer networks. Instead, the Internet to date is composed of a series of various client-server applications. The client and server refer to the role whereby the client program is the entity that initiates a communication and server program is the one that waits passively for and eventually respond to the client that is trying to initiate communication with it. There are several advantages of client-server model which includes centralization of resources, flexibility, scalability and interoperability. A socket on the other hand, is an abstraction in which a program may send and receive data similar to the way input-output of a system is handled. An important characteristic that has driven programmers to program using socket based programming is due to its transparency. This means that, regardless if a socket program is written in Python language, it will still be able to communicate with other socket program which was built using other languages such as C or C++. This voice chat application using socket programming is closely related to distributed computing whereby the client and server paradigm is a distributed application in which the workload are distributed among the nodes namely the client and the server. These nodes serves the same purpose makes it resembles the distributed computing application characteristics.

# REQUIREMENT ANALYSIS

## 3.1 Hardware Requirements Development and Implementation

- Processor : 2.4 GHz
- RAM : 2 GB
- Free storage : 100 MB (Min)

## 3.2 Software Requirements for Development and Implementation

- Platform : Any platform like Visual studio code or Jupyter or Pycharm
- Operating System : Mac/Windows 10
- Programming language : Python 3

## 3.3 Required Libraries:

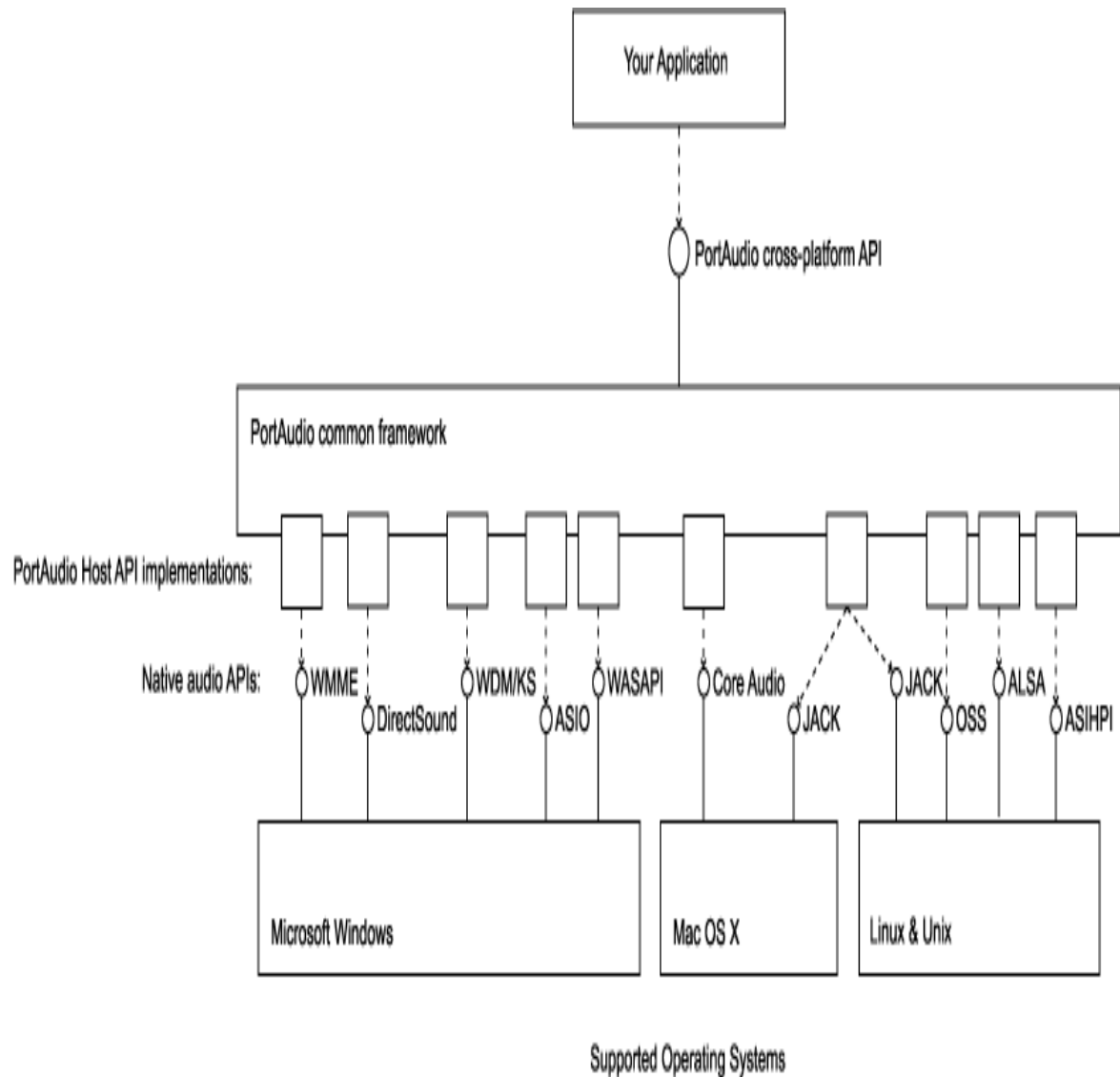
- PyAudio
- Socket Module (standard library)
- Threading Module (standard library)

## 3.4 Dependency Installation:

- Windows: pip install pyaudio
- Linux/Mac: 

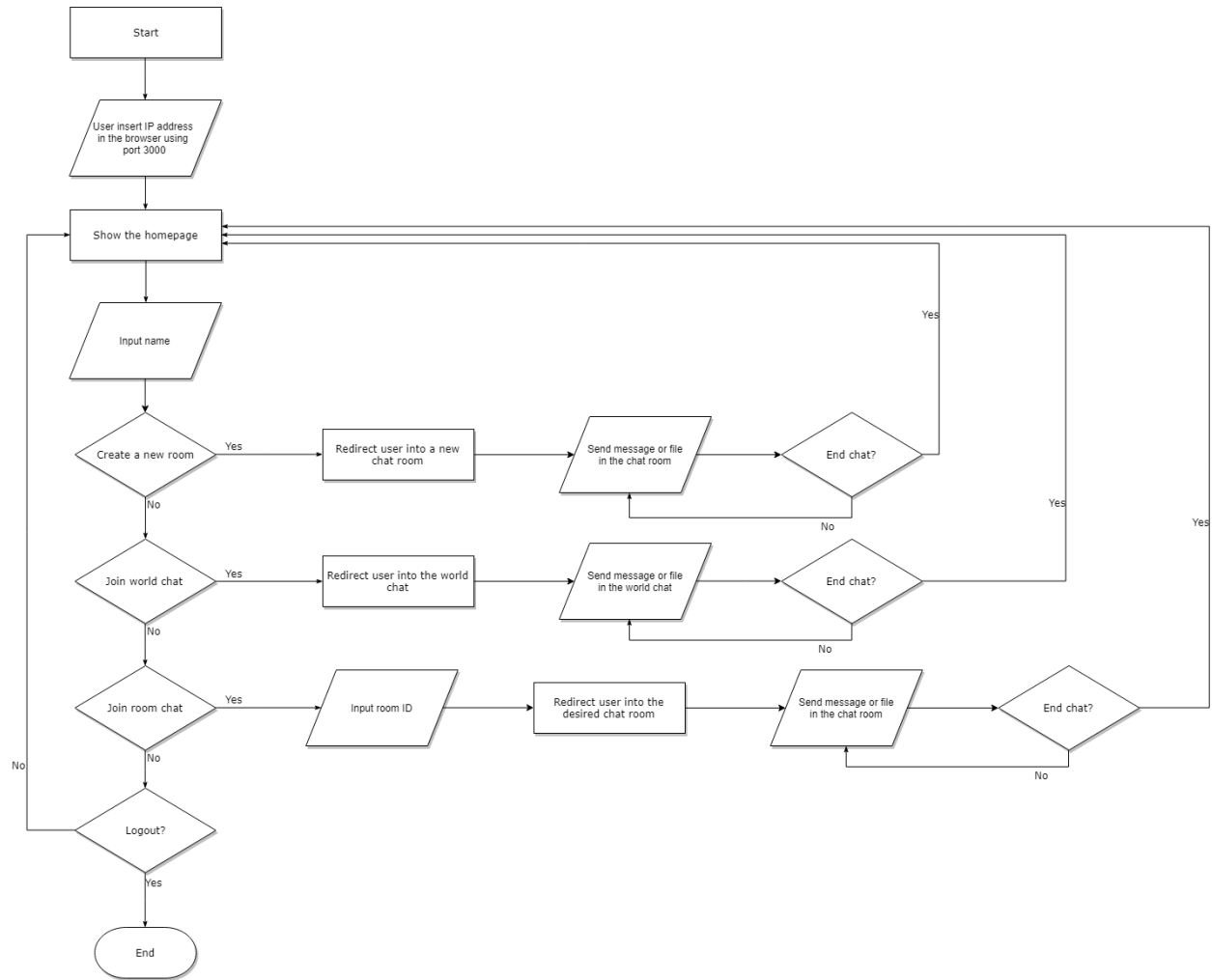
```
sudo apt install -y portaudio19-dev  
sudo apt install -y pyaudio  
pip install -r requirements.txt
```

# FLOWCHART DIAGRAM

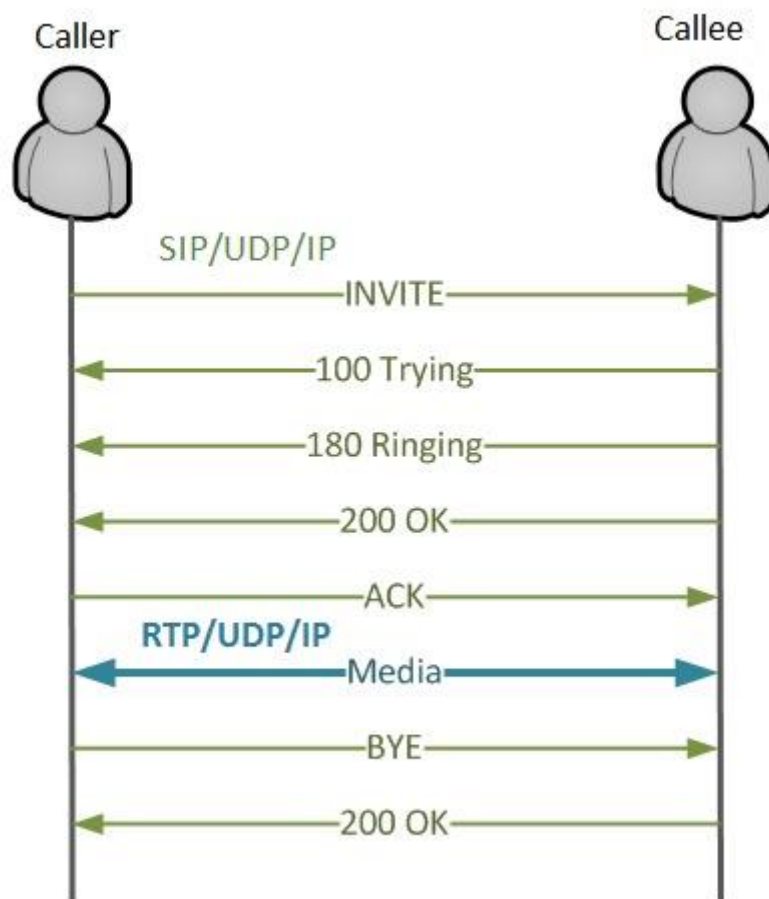




# CONCEPT BEHIND PORT AUDIO



# MESSAGE SEQUENCE DIAGRAM



# CODE

## Server:

```
import socket
import threading

class Server:
    def __init__(self):
        self.ip = socket.gethostbyname(socket.gethostname())
        while 1:
            try:
                self.port = int(input('Enter port number to run on --> '))

                self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.s.bind((self.ip, self.port))

                break
            except:
                print("Couldn't bind to that port")

        self.connections = []
        self.accept_connections()

    def accept_connections(self):
        self.s.listen(100)

        print('Running on IP: '+self.ip)
        print('Running on port: '+str(self.port))

        while True:
            c, addr = self.s.accept()

            self.connections.append(c)

            threading.Thread(target=self.handle_client,args=(c,addr,)).start()

    def broadcast(self, sock, data):
        for client in self.connections:
            if client != self.s and client != sock:
                try:
                    client.send(data)
                except:
                    pass

    def handle_client(self,c,addr):
        while 1:
            try:
                data = c.recv(1024)
                self.broadcast(c, data)

            except socket.error:
                c.close()

server = Server()
```

## Client

```
import socket
import threading
import pyaudio

class Client:
    def __init__(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        while 1:
            try:
                self.target_ip = input('Enter IP address of server --> ')
                self.target_port = int(input('Enter target port of server --> '))

                self.s.connect((self.target_ip, self.target_port))

                break
            except:
                print("Couldn't connect to server")

        chunk_size = 1024 # 512
        audio_format = pyaudio.paInt16
        channels = 1
        rate = 20000

        # initialise microphone recording
        self.p = pyaudio.PyAudio()
        self.playing_stream = self.p.open(format=audio_format, channels=channels, rate=rate, output=True, frames_per_buffer=chunk_size)
        self.recording_stream = self.p.open(format=audio_format, channels=channels, rate=rate, input=True, frames_per_buffer=chunk_size)

        print("Connected to Server")

        # start threads
        receive_thread = threading.Thread(target=self.receive_server_data).start()
        self.send_data_to_server()

    def receive_server_data(self):
        while True:
            try:
                data = self.s.recv(1024)
                self.playing_stream.write(data)
            except:
                pass

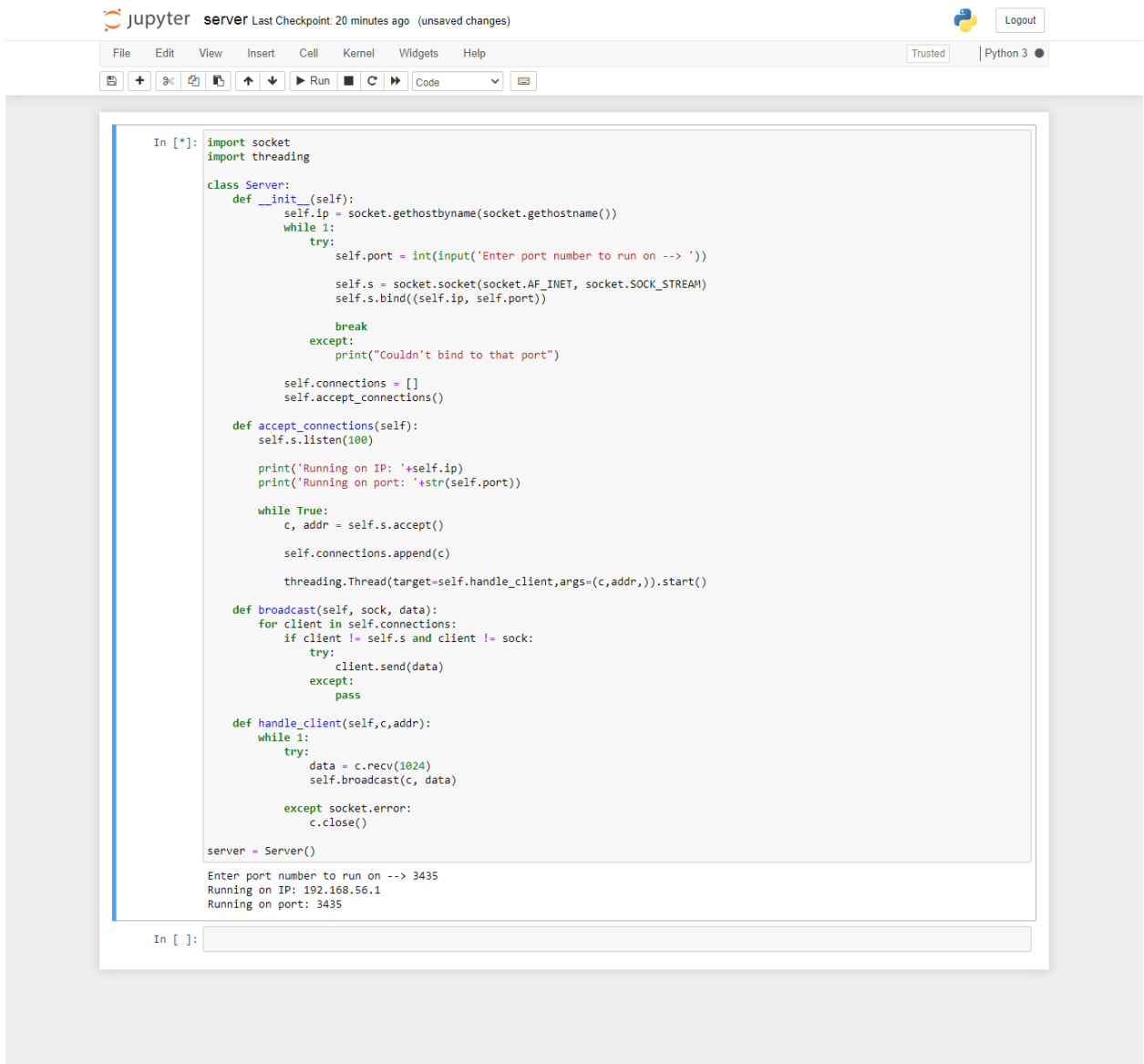
    def send_data_to_server(self):
        while True:
            try:
                data = self.recording_stream.read(1024)
                self.s.sendall(data)
            except:
                pass

client = Client()
```

## How to run?

- Run `server.py` or `server.exe` specifying the port you want to bind to.
- If you intend to use this program across the internet, ensure you have port forwarding that is forwarding the port the server is running on to the server's machine local IP (the IP displayed on the server program) and the correct port.
- Clients can connect across the internet by entering your public IP (as long as you have port forwarding to your machine) and the port the machine is running on or in the same network by entering the IP displayed on the server.
- If the client displays "Connected to Server", you can now communicate with others in the same server by speaking into a connected microphone.

# RESULT:



The image shows a JupyterLab interface with a code editor and a terminal. The code defines a `Server` class that listens for incoming connections and broadcasts data to all connected clients. The terminal output shows the server running on IP 192.168.56.1 and port 3435.

```
In [*]: import socket
import threading

class Server:
    def __init__(self):
        self.ip = socket.gethostname(socket.gethostname())
        while 1:
            try:
                self.port = int(input('Enter port number to run on --> '))

                self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.s.bind((self.ip, self.port))

                break
            except:
                print("Couldn't bind to that port")

        self.connections = []
        self.accept_connections()

    def accept_connections(self):
        self.s.listen(100)

        print('Running on IP: '+self.ip)
        print('Running on port: '+str(self.port))

        while True:
            c, addr = self.s.accept()

            self.connections.append(c)

            threading.Thread(target=self.handle_client,args=(c,addr,)).start()

    def broadcast(self, sock, data):
        for client in self.connections:
            if client != self.s and client != sock:
                try:
                    client.send(data)
                except:
                    pass

    def handle_client(self, self, c, addr):
        while 1:
            try:
                data = c.recv(1024)
                self.broadcast(c, data)

            except socket.error:
                c.close()

server = Server()

Enter port number to run on --> 3435
Running on IP: 192.168.56.1
Running on port: 3435
```

In [ ]:

```
In [*]: import socket
import threading
import pyaudio

class Client:
    def __init__(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        while 1:
            try:
                self.target_ip = input('Enter IP address of server --> ')
                self.target_port = int(input('Enter target port of server --> '))

                self.s.connect((self.target_ip, self.target_port))

                break
            except:
                print("Couldn't connect to server")

        chunk_size = 1024 # 512
        audio_format = pyaudio.paInt16
        channels = 1
        rate = 20000

        # initialise microphone recording
        self.p = pyaudio.PyAudio()
        self.playing_stream = self.p.open(format=audio_format, channels=channels, rate=rate, output=True, frames_per_buffer=chunk_size)
        self.recording_stream = self.p.open(format=audio_format, channels=channels, rate=rate, input=True, frames_per_buffer=chunk_size)

        print("Connected to Server")

        # start threads
        receive_thread = threading.Thread(target=self.receive_server_data).start()
        self.send_data_to_server()

    def receive_server_data(self):
        while True:
            try:
                data = self.s.recv(1024)
                self.playing_stream.write(data)
            except:
                pass

    def send_data_to_server(self):
        while True:
            try:
                data = self.recording_stream.read(1024)
                self.s.sendall(data)
            except:
                pass

client = Client()

Enter IP address of server --> 192.168.56.1
Enter target port of server --> 3435
Connected to Server
```

In [ ]:

In [ ]:

## Literature Review

The socket programming concept has benefited many areas of computer networks today. The socket application developed in this work is based on the concept of distributed computing. This is relevant as to that it allows for resources to be distributed among several node in the network. Socket programming helps to implement the bottom level of network communication, using Application Programming Interface (API). A similar application is built by other researchers as presented in. This application is also using a multicast datagram socket class which is useful for sending and receiving IP multicast packets. Furthermore, a multicast socket is also a UDP datagram socket which is capable for joining "group". This additional functionality is an added advantage for using multicast socket .



## Conclusion and Discussion

This paper presents the detail implementation of a voice chat application that is using socket programming method. In this work, we have chosen Python as the programming language as it covers an adequate range of functions and classes to develop this socket-based programming application. The application is subject to further improvement in the future in which other functionalities may be included to enhance its overall function.

# REFERENCES

[1] Law, K.L.E., Leung, R. 2002. "A design and implementation of active network socket programming." *Computer Communications and Networks*. 78 41-41 ,38-.

[2] Malhotra, A., Sharma, V., Gandhi, P., Purohit, N. "UDP based chat application." *Computer Engineering and Technology (ICCET)*, 2010 2nd International Conference vol.6: pp.V6-374-V6-377.

Kaur, D., Dhanda, P., Mirchandani, M. 2000. "Development of a real time chat application on intelligent network based on fuzzy logic," *Circuits and Systems*, 2000." *Proceedings of the 43rd IEEE Midwest Symposium* vol.3, pp.1376- 1380.

[3] Shirali-Shahreza, M.H., Shirali-Shahreza, M. 2007. "Text Steganography in chat." *Internet. ICI. 3rd IEEE/IFIP International Conference in Central Asia* pp.1-5, 26-28.

[4] Wang, Ho Leung, Tsuhan Chen, Hendriks, F., Xiping Wang, and Zon-Yin Shae. "eMeeting: a multimedia application for interactive meeting and seminar." *Global Telecommunications Conference, GLOBECOM. IEEE* 3, pp. 2994- 2998, 17-21

# APPENDIX

## SERVER:

```
import socket
```

```
import threading
```

```
class Server:
```

```
    def __init__(self):
```

```
        self.ip = socket.gethostbyname(socket.gethostname())
```

```
        while 1:
```

```
            try:
```

```
                self.port = int(input('Enter port number to run on --> '))
```

```
                self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
                self.s.bind((self.ip, self.port))
```

```
                break
```

```
            except:
```

```
                print("Couldn't bind to that port")
```

```
self.connections = []
```

```
self.accept_connections()
```

```
def accept_connections(self):
```

```
    self.s.listen(100)
```

```
    print('Running on IP: '+self.ip)
```

```
    print('Running on port: '+str(self.port))
```

```
    while True:
```

```
        c, addr = self.s.accept()
```

```
        self.connections.append(c)
```

```
        threading.Thread(target=self.handle_client,args=(c,addr,)).start()
```

```
def broadcast(self, sock, data):
```

```
    for client in self.connections:
```

```
        if client != self.s and client != sock:
```

```
            try:
```

```
                client.send(data)
```

```
except:
```

```
    pass
```

```
def handle_client(self,c,addr):
```

```
    while 1:
```

```
        try:
```

```
            data = c.recv(1024)
```

```
            self.broadcast(c, data)
```

```
        except socket.error:
```

```
            c.close()
```

```
server = Server()
```

CLIENT:

```
import socket
```

```
import threading
```

```
import pyaudio
```

```
class Client:
```

```
def __init__(self):

    self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    while 1:

        try:

            self.target_ip = input('Enter IP address of server --> ')

            self.target_port = int(input('Enter target port of server --> '))

            self.s.connect((self.target_ip, self.target_port))

            break

        except:

            print("Couldn't connect to server")

    chunk_size = 1024 # 512

    audio_format = pyaudio.paInt16

    channels = 1

    rate = 20000

    # initialise microphone recording
```

```
self.p = pyaudio.PyAudio()
```

```
self.playing_stream = self.p.open(format=audio_format, channels=channels,  
rate=rate, output=True, frames_per_buffer=chunk_size)
```

```
self.recording_stream = self.p.open(format=audio_format, channels=channels,  
rate=rate, input=True, frames_per_buffer=chunk_size)
```

```
print("Connected to Server")
```

```
# start threads
```

```
receive_thread = threading.Thread(target=self.receive_server_data).start()
```

```
self.send_data_to_server()
```

```
def receive_server_data(self):
```

```
    while True:
```

```
        try:
```

```
            data = self.s.recv(1024)
```

```
            self.playing_stream.write(data)
```

```
        except:
```

```
            pass
```

```
def send_data_to_server(self):
```

```
    while True:
```

```
        try:
```

```
            data = self.recording_stream.read(1024)
```

```
            self.s.sendall(data)
```

```
        except:
```

```
            pass
```

```
client = Client()
```