Computer Communication Networking
UE21EC351A

Real-time audio communication system

Team DANK

Anirudh N S PES2UG21EC015

K Dharshan PES2UG21EC059

Nithin Chowdary K PES2UG21EC063

Karthik M PES2UG21EC064

Abstract

This project focuses on creating a real-time audio communication system using a client-server architecture. The server listens for incoming connections, handles multiple client connections concurrently, and facilitates communication between clients. Clients, when connected, can send and receive audio data, allowing for real-time voice communication. The system uses the socket library for network communication and threading to manage multiple client connections simultaneously.

Introduction

The provided code implements a basic client-server communication system for real-time audio streaming using Python's socket and threading libraries. The system enables clients to connect to a server, send audio data while pressing a designated key, and receive and play back audio data from other connected clients. The server manages multiple client connections and facilitates communication between them.

Socket programming

- Socket Creation: Socket programming involves creating communication endpoints (sockets) that enable data transfer between different processes or devices over a network.
- Connection Establishment: Sockets facilitate the establishment of connections between a client and a server, allowing bidirectional communication by sending and receiving data through the network.

Transmission Control Protocol

In the project here, TCP (Transmission Control Protocol) is used for communication between the server and clients.

Here's how TCP is employed:

1. **Socket Initialization**: Both the server and the clients create TCP sockets using the `socket.socket` method with the argument `socket.SOCK_STREAM`. This indicates that the sockets will use TCP for communication.
2. **Connection Establishment**: The server binds to a specific IP address and port and listens for incoming TCP connections. Clients connect to

the server using their TCP sockets. The server.listen(5) line in the server script sets the server to a listening state, ready to accept incoming TCP connections.

3. **Data Transmission**: Serialized data, including audio information, is sent and received through the TCP connections using the send and recv methods.

Working

Server:

- The server initializes a socket and listens for incoming connections on a specified port.
- Each client connection is handled in a separate thread (handle_client function) to allow for concurrent communication with multiple clients.
- The server receives audio data from one client and broadcasts it to all other connected clients.

Client:

- The client connects to the server using a socket connection.
- It captures audio input from the microphone using the pyaudio library.
- When the designated key (right shift key) is pressed, the client starts sending audio data to the server.
- The client also listens to incoming audio data from the server and plays it back.

Code

server.py

```
import socket
import threading
import pickle

HEADER = 1024
PORT = 19699
SERVER = "" #Replace with your server machine ip
ADDR = (SERVER, PORT)
PAYLOAD_BITS = 8

connections = set()

def handle_client(conn, addr):
```

```
    print(f"[NEW CONNECTION] {addr} connected")
    connected = True
    connections.add(conn)
    print(f"CONNECTIONS: {connections}")
    data = b''
    while connected:
        data += conn.recv(HEADER)
        msg_length = int(data[:PAYLOAD_BITS])#.decode()
        data = data[PAYLOAD_BITS:]
        # print(f"\x1b[31mSERVER MESSAGE LENGTH:
{msg_length}\x1b[0m")
        while len(data)<msg_length:
            data += conn.recv(HEADER)

        message=pickle.loads(data)
        if(message == "!DISCONNECT"):
            print(f"\x1b[31m{conn} DISCONNETED\x1b[0m")
            connected=False
            connections.discard(conn)
        else:
            send(conn, data[:msg_length],msg_length)
            data = data[msg_length:]
    conn.close()

def send(conn, data, msg_length):
    for c in connections:
        if c is not conn:
            send_length = str(msg_length).encode('utf-8')
            send_length = b' '*(PAYLOAD_BITS-len(send_length)) + send_length
            message = send_length+data
            # print(f"\x1b[34mSERVER SIDE SENDING MESSAGE:
{message}\x1b[0m")
            c.send(message)


def start():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind(ADDR)
        server.listen(5)
        print(f"[LISTENING] server is listening on {SERVER, PORT}")
        threading.Thread(target=send)
        while True:
            conn, addr = server.accept()
            thread = threading.Thread(target=handle_client, args=(conn, addr))
            thread.start()
            print(f"[ACTIVE CONNECTIONS] {threading.active_count()-1}")
```

```
print("[STARTING] server is starting...")
start()
```

Client.py

```
import socket
import threading
import pickle
from pynput import keyboard
import pyautogui
import pyaudio
import time

global connected

FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
HEADER = 1024
PORT = 19699
SERVER = "" #Replace with your service machine ip
ADDR = (SERVER, PORT)
DISCONNECT_MESSAGE = "!DISCONNECT"
connected = True
PAYLOAD_BITS = 8
talking = False

audio = pyaudio.PyAudio()
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)

input_stream = audio.open(format=FORMAT,
                    channels = CHANNELS,
                    rate = RATE,
                    input = True)

output_stream = audio.open(format=FORMAT,
                    channels = CHANNELS,
                    rate = RATE,
```

```
                    output = True)

def stopRecord(key):
    global talking
    try:
        talking = False
        print("STOPPING")
    except AttributeError:
        pass


def record(key):
    global talking
    global connected
    try:
        if(key == keyboard.Key.shift_r):
            talking=True
            with keyboard.Listener(on_release=stopRecord) as listener:
                while talking:
                    data = input_stream.read(HEADER, exception_on_overflow = False)
                    data = pickle.dumps(data)

                    send_length = str(len(data)).encode('utf-8')
                    print(f"\x1b[31mSending {send_length} bytes\x1b[0m")
                    send_length = b' '*(PAYLOAD_BITS-len(send_length)) + send_length
                    message = send_length+data
                    print(f"\x1b[31mSending message: {message}\x1b[0m")
                    client.sendall(message)

        elif(key.char == 'x'):
            print("DISCONNECTING")
            data = pickle.dumps(DISCONNECT_MESSAGE)
            send_length = str(len(data)).encode('utf-8')
            send_length = b' '*(PAYLOAD_BITS-len(send_length)) + send_length
            message = send_length+data
            client.send(message)
            connected=False
            talking=False

            input_stream.stop_stream()
            input_stream.close()
            output_stream.stop_stream()
            output_stream.close()
            output_wavefile.close()
            audio.terminate()
    except:
```

```
        print("Press Left Shift To Talk")

def setup():
    global connected
    with keyboard.Listener(on_press=record) as listener:
        while connected:
            pass
        client.close()

def listenToServer():
    global connected

    data = b''
    while connected:
        try:
            data += client.recv(HEADER)
            msg_length = int(data[:PAYLOAD_BITS])
            data = data[PAYLOAD_BITS:]
            while len(data)<msg_length:
                data += client.recv(HEADER)

            sample = pickle.loads(data)
            data = data[msg_length:]
            print(f"\x1b[32mReceived Message: {sample}\x1b[0m")
            output_stream.write(sample)
        except ValueError:
            print(f"DATA RECIEVED IS {data}")

thread = threading.Thread(target=setup)
thread2 = threading.Thread(target=listenToServer)
thread.start()
thread2.start()
```

Observation:

The below is the server terminal which shows 2 Active connections.

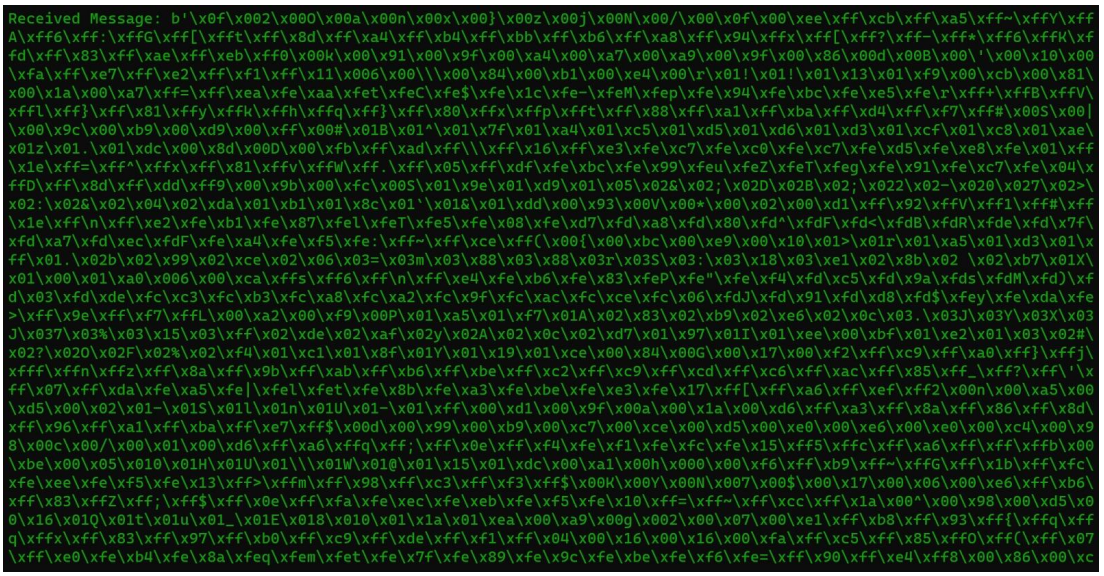
```
PS C:\Python310> python server1.py
[STARTING] server is starting...
[LISTENING] server is listening on 192.168.136.230, 19699
[NEW CONNECTION] ('192.168.136.230', 54542) connected
[ACTIVE CONNECTIONS] 1
CONNECTIONS: {<socket.socket fd=364, family=AddressFamily.AF_INET, type=Sock
etKind.SOCK_STREAM, proto=0, laddr=('192.168.136.230', 19699), raddr=('192.1
68.136.230', 54542)>}
[NEW CONNECTION] ('192.168.136.212', 1321) connected
[ACTIVE CONNECTIONS] 2
CONNECTIONS: {<socket.socket fd=412, family=AddressFamily.AF_INET, type=Sock
etKind.SOCK_STREAM, proto=0, laddr=('192.168.136.230', 19699), raddr=('192.1
68.136.212', 1321)>, <socket.socket fd=364, family=AddressFamily.AF_INET, ty
pe=SocketKind.SOCK_STREAM, proto=0, laddr=('192.168.136.230', 19699), raddr=
('192.168.136.230', 54542)>}
```

Sending message

```
Sending message: b'    2066\x80\x04\x95\x07\x08\x00\x00\x00\x00\x00B\x00
\x08\x00\x00;\xfc\x7f\xfdC\xfeL\xfe\xba\xfd\xe1\xfc$\xfc\xcc\xfb\xf7\xfb\x9a
\xfc\x88\xfd\x84\xfeP\xff\xc5\xff\xe0\xff\xbd\xff\x8e\xff\x80\xff\xa7\xff\r\
x00\xb0\x00\x94\x01\xaa\x02\xc0\x03\x82\x04\xa4\x04\x18\x04\'\x03K\x02\xf1\x
014\x02\xd9\x02\x81\x03\xd6\x03\xbd\x03H\x03\xaa\x02\x16\x02\xb3\x01\x8d\x01
\x9a\x01\xba\x01\xca\x01\xb4\x01v\x01\x1a\x01\xab\x004\x00\xb8\xffB\xff\xd5\
xfew\xfe$\xfe\xdb\xfd\x9f\xfdv\xfdg\xfdm\xfdw\xfdy\xfdm\xfdT\xfd5\xfd\x1b\xf
d\x18\xfdG\xfd\xbc\xfdr\xfeD\xff\x03\x00\x92\x00\xfc\x00]\x01\xcb\x01B\x02\x
b2\x02\x16\x03}\x03\xf8\x03r\x04\xb9\x04\x8b\x04\xce\x03\xac\x02y\x01\x85\x0
0\xea\xff\x8a\xff(\xff\x95\xfe\xdc\xfd!\xfd\x89\xfc\x1d\xfc\xc7\xfbs\xfb\'\x
fb\x00\xfb\x1e\xfb\x81\xfb\x0e\xfc\xa2\xfc%\xfd\x8d\xfd\xe5\xfd:\xfe\xa2\xfe
5\xff\x02\x00\xf9\x00\xf7\x01\xcb\x02[\x03\xa3\x03\xb7\x03\xaf\x03\x9f\x03\x
9c\x03\xb0\x03\xd6\x03\xf2\x03\xd9\x03p\x03\xca\x02\x16\x02\x8a\x01@\x01#\x0
1\x07\x01\xd3\x00\x85\x00*\x00\xce\xffo\xff\x07\xff\x94\xfe \xfe\xb8\xfdh\xf
d2\xfd\x06\xfd\xd2\xfc\x7f\xfc\x05\xfc[\xfb\x11\xfb\xf1\xfa&\xfb\x90\xfb\xf8
\xfbB\xfcx\xfc\xbd\xfc+\xfd\xb5\xfdJ\xfe\xde\xfe~\xff5\x00\xf0\x00\x81\x01\x
be\x01\xac\x01{\x01h\x01\x8a\x01\xb8\x01\xb1\x01H\x01\x8c\x00\xc0\xff"\xff\x
bf\xfek\xfe\xef\xfd:\xfdn\xfc\xc8\xfbk\xfbJ\xfb@\xfb6\xfb9\xfb[\xfb\xa1\xfb\
xec\xfb%\xfcW\xfc\xae\xfcS\xfdE\xfeT\xff6\x00\xc6\x00\x11\x01S\x01\xc1\x01o\
x029\x03\xdc\x03#\x04\t\x04\xc1\x03\x92\x03\xa5\x03\xe9\x03#\x04\x13\x04\xb2
\x034\x03\xe4\x02\xf3\x02O\x03\xae\x03\xc2\x03p\x03\xe5\x02o\x02B\x02O\x02P\
x02\xf8\x017\x01L\x00\xab\xff\xac\xffF\x00\n\x01j\x01-\x01\xad\x00\xa7\x00\x
b2\x01\xc4\x03%\x06\xf1\x07\xac\x08\x85\x08"\x08\x13\x08r\x08\xfd\x08X\t\\\t
!\t\xd7\x08\x96\x08S\x08\xf9\x07\x81\x07\xfe\x06\x91\x06U\x06\x87\x06\x
d4\x06\x1e\x00\xb2\x00\x1b\x01V\x01\\\x01$\x01\xaa\x00\x07\x00a\xff\xd2\xfe_
\xfe\xf0\xfdq\xfd\xe5\xfcn\xfc4\xfcM\xfc\xb1\xfcG\xfd\xf8\xfd\xba\xfez\xff!\
x00\x8b\x00\xa6\x00z\x00-\x00\xeb\xff\xc3\xff\xa1\xffS\xff\xb9\xfe\xe0\xfd\x
ff\xfcU\xfc\xfe\xfb\xe8\xfb\xe4\xfb\xc4\xfbv\xfb\xf2\xfa:\xfaV\xf9V\xf8g\xf7
\xbe\xf6\x85\xf6\xbb\xf6>\xf7\xe0\xf7t\xf8\xe9\xf8F\xf9\x9d\xf9\t\xfa\x99\xf
aQ\xfb"\xfc\xeb\xfc\x84\xfd\xcc\xfd\xba\xfdc\xfd\xf2\xfc\x98\xfcr\xfc\x84\xfc
\xb1\xfc\xd1\xfc\xc8\xfc\x8f\xfc?\xfc\xf7\xfb\xce\xfb\xc8\xfb\xe4\xfb%\xfc\
x93\xfc/\xfd\xea\xfd\x9d\xfe0\xff\xa1\xff\xfd\xffR\x00\x93\x00\xb0\x00\xa6\x
00\x90\x00\x90\x00\xb9\x00\xf6\x00$\x01/\x01\x1e\x01\x05\x01\xea\x00\xc2\x00
\x8a\x00R\x00<\x00c\x00\xc2\x006\x01\x91\x01\xbe\x01\xc6\x01\xd0\x01\x06\x02
\x8c\x02j\x03\x83\x04\x9c\x05u\x06\xe4\x06\xef\x06\xc2\x06\x93\x06\x83\x06\x
95\x06\xb8\x06\xd6\x06\xdd\x06\xbb\x06c\x06\xce\x05\x11\x05P\x04\xbc\x03v\x0
```

```
xe3\xfe\x1e\xff\xe7\xff\xfb\x00\t\x02\xd8\x02U\x03\x8a\x03\x85\x03O\x03\xee\
x02u\x02\x04\x02\xbc\x01\xa5\x01\xa7\x01\x8d\x01!\x01L\x00&\xff\xea\xfd\xe2\
xfcA\xfc\x15\xfcN\xfc\xc4\xfcL\xfd\xbf\xfd\x05\xfe\x1f\xfe\x1c\xfe\x1c\xfeB\
xfe\xa2\xfe8\xff\xeb\xff\x9f\x00C\x01\xce\x01A\x02\x93\x02\xb8\x02\xb3\x02\x
97\x02\x82\x02\x8b\x02\xa9\x02\xb8\x02\x8a\x02\x0e\x02[\x01\xac\x009\x00\x0e
\x00\t\x00\xea\xff\x85\xff\xe6\xfeU\xfe)\xfe\x8b\xfeV\xff$\x00\x96\x00\x8f\x
00F\x00 \x00Z\x00\xdd\x00M\x01H\x01\xae\x00\xc1\xff\xf0\xfe\x9c\xfe\xcb\xfe5
\xffd\xff\x07\xff \xfe\xf8\xfc\xf3\xfb_\xfb[\xfb\xcd\xfb\x81\xfc2\xfd\xa2\xf
d\xba\xfd\x8d\xfdN\xfd%\xfd&\xfdN\xfd\xa8\xfdV\xfe}\xff\x06\x01\x8d\x02\x8c\
x03\xa8\x03\xfe\x02\x1b\x02\xa4\x01\x00\x02\x17\x03v\x04\x93\x05\x10\x06\xe4
\x05@\x05v\x04\xbf\x03/\x03\xb8\x02J\x02\xed\x01\xbe\x01\xce\x01\x02\x02\x10
\x02\xb1\x01\xda\x00\xcd\xff\xfb\xfe\xbe\xfeC\xff\x06\x00\x07\x01\xe3\x01i\x
02\x89\x02O\x02\xf5\x01\xc3\x01\xf3\x01\x83\x027\x03\xc7\x03\x0f\x04!\x04\x1
a\x04\x02\x04\xcf\x03\x82\x036\x03\x0f\x03\x00\x03\xdb\x02r\x02\xd0\x011\x01
\xcd\x00\xa0\x00q\x00\x16\x00\xa3\xffa\xff\x83\xff\xf2\xffR\x00[\x00\x12\x00
\xc9\xff\xe4\xff\x7f\x00W\x01\xfa\x01\t\x02v\x01\x83\x00\x93\xff\xed\xfe\x9a
\xfey\xfeW\xfe\x1c\xfe\xca\xfde\xfd\xf4\xfcs\xfc\xec\xfb\x82\xfbg\xfb\xba\xf
br\xfcY\xfd.\xfe\xbf\xfe\x07\xff\x1c\xff \xff+\xffT\xff\xac\xffA\x00\x04\x01
\xc0\x01/\x02,\x02\xc5\x01;\x01\xd2\x00\x9e\x00\x80\x00:\x00\xaa\xff\xdc\xfe
\x01\xfeL\xfd\xd2\xfc\x87\xfcP\xfc!\xfc\xff\xfb\xfb\xfb\x1c\xfcV\xfc\x9e\xfc
\xfb\xfc\x83\xfdL\xfeS\xffo\x00b\x01\xfb\x01{\x02\xfd\x01\xaf\x01t\x01x\x01\
xc9\x01R\x02\xdd\x02/\x03\x1a\x03\x95\x02\xb9\x01\xb9\x00\xd0\xff&\xff\xd4\x
fe\xdb\xfe%\xff\x81\xff\xad\xffb\xffz\xfe\x11\xfd\x8f\xfb|\xfaE\xfa\xf7\xfa\
x94.'
Sending b'2066' bytes
```

Received message



Advantages

- Real-Time Communication:
    - The system enables real-time audio communication between multiple clients.
- Scalability:
    - The server handles multiple client connections concurrently, allowing for scalability.
- Flexibility:
    - The use of threading allows for simultaneous communication with multiple clients without blocking the main execution.

Future Scope

- Enhanced Security:
    - Implement encryption for secure communication between clients and the server.
- User Authentication:
    - Integrate a user authentication system to ensure authorized access to the server.
- Improved Audio Quality:
    - Explore options for improving audio quality and reducing latency in real-time communication.
- Graphical User Interface (GUI):

- Develop a GUI for a more user-friendly experience, allowing users to easily manage connections and settings.

Conclusion

The provided client-server code establishes a TCP-based communication system for real-time audio streaming. Threading enables concurrent handling of multiple client connections on the server. The use of pickling facilitates the serialization of audio data for transmission. While the project demonstrates a basic framework, further refinement and additional features could enhance its usability and robustness.

In conclusion, this project provides a foundation for real-time audio communication and can be extended and enhanced for various applications, including online conferencing, gaming, or remote collaboration. The incorporation of security measures and additional features would contribute to the robustness and versatility of the system.