**Department of Computer Science & Engineering**

**CSE 3162 Computer Networks Lab**

**Mini Project – Progress Report**

**Title of Project: Video Calling app**

**Team details:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl .No. | Registration No. | Student Name | email | Contact no |
| **1** | 210905191 | Shreya Sadhasivam | **shreya.sadhasivam@learner.manipal.edu** | 9150009778 |
| **2** | 210905416 | Syed Murtaza Ali | **syed.ali@learner.manipal.edu** | 9450989006 |
| **3** | 210905019 | Viraj Chetan Desai | **viraj.desai@learner.manipal.edu** | 7972296948 |
| **4** | 210905180 | Himanshu Banerji | **himanshu.banerji@learner.manipal.edu** | 8175856609 |

**Project Summary:**

In this project, a simple video and audio communication system was developed using Python. The two primary modules, the server and the client, are responsible for sending and receiving audio and video data. The implementation leverages the power of the socket library for communication, cv2 (from OpenCV) for video capture and display, pickle for serialization, and pyaudio for audio streaming.

**Key Features:**

1. **Bi-directional Audio Communication**: Both the server and client can send and receive audio data, allowing for a two-way conversation.
2. **Video Streaming:** The server captures video from its webcam, sends it to the client, and the client displays the incoming video stream.
3. **Chatting Feature**:

This functionality is an integral part of the system, enabling textual communication alongside audio and video. Here's how it works:

**Integrated Text Box**: Both the server and the client have an integrated text input field, allowing users to type and send messages in real-time.

**Instant Messaging**: As soon as a message is sent from one end, it is instantly displayed on the other end. This ensures real-time communication, vital for conversations.

1. **Threading:** This ensures that multiple processes, like sending and receiving audio, happen simultaneously without blocking each other.

**Progress Report:**

**Implementation Details**:

1. **Socket Creation and Binding**: The server creates a socket, binds to a specific IP and port, and starts listening for incoming connections. The client creates a socket and connects to the server using the given IP and port.
2. **Audio Configuration**: Both the server and client utilize the **pyaudio** library to handle audio streaming. The parameters, such as audio format, channels, rate, and chunk size, are defined for consistency.
3. **Threading**: Threading is used to handle simultaneous audio send and receive operations. This ensures that the audio streaming is smooth and independent of other operations, such as video streaming.
4. **Video Streaming**: Video frames are captured using the **cv2** library from a webcam, resized, serialized using **pickle**, and then sent over the socket. The client deserializes the received data to obtain the video frame.

**Improvements**:

1. **Code Structure**: The code has been structured for better readability and modularity. Functions have been defined to reduce redundancy, and constants are defined at the beginning of the code for both client and server.
2. **Exception Handling**: Proper exception handling have been implemented .

**Plans for Further Enhancement**:

1. **Compression**: Video compression techniques must be used to reduce the size of the transmitted video data since the current size of the data seems to cause a frame rate drop..
2. **User Interface**: Implement a user-friendly interface for better user experience.
3. **Optimization**: Attempt implementation of asynchronous buffers.
4. **Chat functionality** : Implement chat functionality between the two users for better call experience

**Current Code:**

**server.py**

import socket

import cv2

import pickle

import struct

import threading

import pyaudio

import imutils

def sendAudio(client\_socket, audio\_format, channels, rate, chunk):

    stream\_send = p.open(format=audio\_format, channels=channels, rate=rate, input=True, frames\_per\_buffer=chunk)

    while True:

        try:

            data = stream\_send.read(chunk)

            client\_socket.sendall(data)

        except:

            break

def receiveAudio(client\_socket, audio\_format, channels, rate, chunk):

    stream\_receive = p.open(format=audio\_format, channels=channels, rate=rate, output=True, frames\_per\_buffer=chunk)

    while True:

        try:

            data = client\_socket.recv(chunk)

            if not data:

                break

            stream\_receive.write(data, chunk)

        except:

            break

host\_ip = '192.168.29.122'

port = 9999

socket\_address = (host\_ip, port)

audio\_format = pyaudio.paInt16

channels = 2

rate = 44100

chunk = 1024

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(socket\_address)

server\_socket.listen(5)

print("[\*] LISTENING AT:", socket\_address)

p = pyaudio.PyAudio()

try:

    while True:

        client\_socket, addr = server\_socket.accept()

        print('[+] CONNECTED FROM:', addr)

        if client\_socket:

            vid = cv2.VideoCapture(0)

            t\_audio\_send = threading.Thread(target=sendAudio, args=(client\_socket, audio\_format, channels, rate, chunk))

            t\_audio\_send.start()

            t\_audio\_receive = threading.Thread(target=receiveAudio, args=(client\_socket, audio\_format, channels, rate, chunk))

            t\_audio\_receive.start()

            while vid.isOpened():

                try:

                    ret, frame = vid.read()

                    frame = imutils.resize(frame, width=320)

                    serialized\_frame = pickle.dumps(frame)

                    message = struct.pack("Q", len(serialized\_frame)) + serialized\_frame

                    client\_socket.sendall(message)

                    cv2.imshow("SENDER'S VIDEO", frame)

                    key = cv2.waitKey(1) & 0xFF

                    if key == ord('q'):

                        client\_socket.close()

                        vid.release()

                        cv2.destroyAllWindows()

                        break

                except:

                    break

except KeyboardInterrupt:

    server\_socket.close()

    p.terminate()

**client.py**

import socket

import cv2

import pickle

import struct

import threading

import pyaudio

def sendAudio(audio\_format, channels, rate, chunk):

    stream\_send = p.open(format=audio\_format, channels=channels, rate=rate, input=True, frames\_per\_buffer=chunk)

    while True:

        try:

            data = stream\_send.read(chunk)

            client\_socket.sendall(data)

        except:

            break

def receiveAudio(audio\_format, channels, rate, chunk):

    stream\_receive = p.open(format=audio\_format, channels=channels, rate=rate, output=True, frames\_per\_buffer=chunk)

    while True:

        try:

            data = client\_socket.recv(chunk)

            if not data:

                break

            stream\_receive.write(data, chunk)

        except:

            break

host\_ip = '192.168.29.122'

port = 9999

audio\_format = pyaudio.paInt16

channels = 2

rate = 44100

chunk = 1024

payload\_size = struct.calcsize("Q")

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((host\_ip, port))

data\_buffer = b""

p = pyaudio.PyAudio()

t2 = threading.Thread(target=sendAudio, args=(audio\_format, channels, rate, chunk))

t2.start()

t3 = threading.Thread(target=receiveAudio, args=(audio\_format, channels, rate, chunk))

t3.start()

try:

    while True:

        while len(data\_buffer) < payload\_size:

            packet = client\_socket.recv(8 \* 1024)

            if not packet:

                break

            data\_buffer += packet

        packed\_msg\_size = data\_buffer[:payload\_size]

        data\_buffer = data\_buffer[payload\_size:]

        msg\_size = struct.unpack("Q", packed\_msg\_size)[0]

        while len(data\_buffer) < msg\_size:

            data\_buffer += client\_socket.recv(8 \* 1024)

        frame\_data = data\_buffer[:msg\_size]

        data\_buffer = data\_buffer[msg\_size:]

        frame = pickle.loads(frame\_data)

        cv2.imshow("RECEIVING VIDEO", frame)

        key = cv2.waitKey(1) & 0xFF

        if key == ord('q'):

            break

finally:

    client\_socket.close()

    p.terminate()