Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

VoIP COMMUNICATION SYSTEM

Course Title: Computer Networking Lab
Course Code: CSE 312
Section: 222 D5

Students Details

Name	ID
Tasdid Rahman Khan	222002029
Md Syful Islam	222002111

Submission Date: 24 December,2024
Course Teacher's Name: Muhaimen Khan

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

Voice over Internet Protocol (VoIP) enables the transmission of voice and multimedia over the internet. It has revolutionized communication by making it faster and more cost-effective than traditional telephony. In this project, we propose the development of a basic VoIP system using Java socket programming. This system will allow two or more users to communicate with each other over a network using their microphones and speakers.

1.2 Motivation

Traditional voice communication systems often rely on expensive telephony infrastructure. VoIP, on the other hand, leverages the internet to transmit voice data packets, offering a cost-effective and flexible alternative. This project aims to develop a VoIP communication system using Java and socket programming to explore the fundamental principles of VoIP technology. [**?**].

1.3 Objectives

The primary objective of this project is to develop a working VoIP communication system that can:

- Capture audio from the microphone of the client.
- Transmit the captured audio to a remote server over a network using UDP sockets.
- Receive audio on the server and play it in real-time using speakers.
- Ensure low latency and sufficient quality for real-time voice communication.
- Support basic P2P communication between two or more clients.

1.4 Problem Definition

1.4.1 Problem Statement

Traditional communication methods are costly and lack flexibility. This project addresses these issues by creating a VoIP system that uses the internet for low-cost and real-time communication.

1.4.2 Complex Engineering Problem

The following table must be completed according to your above discussion in detail. The column on the right side should be filled only on the attributes you have chosen to be touched by your own project.

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
P1: Depth of knowledge required	Understanding Java networking, audio handling, and threads.
P2: Range of conflicting requirements	Balancing good audio quality with low delays.
P3: Depth of analysis required	Ensuring safe data handling and smooth transmission.
P4: Familiarity of issues	Handling problems like data loss and delays.
P5: Extent of applicable codes	Following rules for socket communication.
P6: Extent of stakeholder involvement and conflicting requirements	Mainly involves developers and users.
P7: Interdependence	Server and clients depend on network reliability.

1.5 Design Goals

- Ensure smooth and real-time communication.
- Use resources efficiently for audio handling.
- Provide a simple and user-friendly interface.
- Enable both text messaging and audio recording options.

1.6 Application

This project can be used in:

- Online communication tools for work and personal use.
- Educational platforms for real-time discussions.
- Low-cost phone system alternatives in remote areas.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The project builds a VoIP system using Java to demonstrate key ideas in real-time voice transmission, socket programming, and handling multiple users. [**?**] [**?**] [**?**].

2.2 Project Details

The server application, written in Java, listens on port 8080 for incoming connections. It can manage multiple users at the same time using threads. The server also has a graphical interface to show its status.

2.2.1 Client Application

The client application allows users to connect to the server and talk. It includes a graphical interface for sending messages, voice data, and interacting with other users. Users can also record and send pre-recorded audio messages.

2.3 Implementation

2.3.1 Workflow

The server starts and waits for clients to connect. Once clients connect, communication channels are established between the server and the clients. Voice data is recorded on the client side, transmitted to the server, and played back in real-time on other clients. Additionally, users can send text messages and recorded audio messages, which are displayed or played back as needed.

2.3.2 Tools and Libraries

- **Java Socket API:** For communication over the network.
- **javax.sound.sampled:** For recording and playing sound.
- **Swing:** For creating the graphical interface.

2.3.3 Implementation details (with screenshots and programming codes)

Server Code Highlights:

1. Manages connected clients safely using synchronized lists.
2. Sends messages, live audio, and recorded audio data to all connected clients

Pseudo code of server:

1. Initialize server socket on port 8080.
2. Create a synchronized list to store client handlers.
3. Create a method ‘startServer‘ to accept client connections.
   - Display a message indicating the server is running.
   - Loop to accept new client connections:
   - Accept client socket.
   - Create a new ‘ClientHandler‘ for the client.
   - Add the ‘ClientHandler‘ to the list of client handlers.
   - Start a new thread to handle communication with the client.
4. Create a ‘ClientHandler‘ class to manage each client’s communication:
   - Initialize input/output streams for the client.
   - Listen for messages and audio data:
   - If a string message is received, broadcast the message to all clients.
   - If audio data is received, broadcast the audio to all clients.
   - Handle client disconnection by removing the client handler from the list and closing the socket.
5. Broadcast method:
   - Iterate through the list of client handlers and send the message/audio to all clients except the sender.
6. Custom painting for a gradient background in the server’s GUI.
7. Start the server when the "Start Server" button is pressed.

Client Code Highlights:

1. Records sound using TargetDataLine.
2. Plays sound using SourceDataLine.
3. Allows sending text messages and audio messages.
4. Provides options for recording and playing back audio files.

Pseudo code of client:

1. Ask the user for a username (if not provided, use "Anonymous").
2. Connect to the server on localhost at port 8080.
3. Create input/output streams to communicate with the server.
4. Send the client's username to the server.
5. Create a GUI for the client with a text area for messages and buttons for various actions (send message, send audio, listen to audio, start voice call).
6. In a new thread, listen for incoming messages or audio from the server:
   - If a string message is received, append it to the chat area.
   - If audio data is received, save it as a file and prompt the user to play it.
7. For sending messages: - Capture the message from the text field and send it to the server.
   - Display the sent message in the chat area.
8. For sending audio:
   - Capture audio from the microphone, save it as a byte array, and send it to the server.
9. For listening to audio:
   - Play the received audio file.
10. For starting a voice call:
    - Set the call status to true and create a microphone thread to capture audio.
    - Create a speaker thread to play audio received from the pipe.
    - Connect both threads via piped streams to simulate real-time voice communication.
11. Provide feedback and prompts using 'JOptionPane' for actions such as audio recording, playback, and connection status.
12. Handle client disconnection gracefully.

2.4 Algorithms

Audio and Message Transmission Steps

- Record sound from a microphone or prepare a text message.
- Convert the data (sound or text) into byte data.
- Send the byte data over the network.
- Receive the data on the other side.
- Play the sound or display the message in the chat interface.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The system was tested locally with:

- A server running on localhost.
- Multiple client instances connecting to the server.

3.2 Results Analysis/Testing

3.2.1 Latency

The system performed well with minimal delay during communication.

3.2.2 Audio Quality

The audio quality was acceptable for basic communication. While sufficient for general use, some compression techniques could be applied to improve quality further, especially in high-noise environments. Testing revealed occasional distortion when multiple users transmitted audio simultaneously, suggesting the need for improved buffer management. Overall, the system maintained a good balance between quality and performance for real-time voice communication.

3.3 Results Overall Discussion

The project successfully demonstrated a working VoIP system. Some challenges like maintaining quality during network delays were observed.

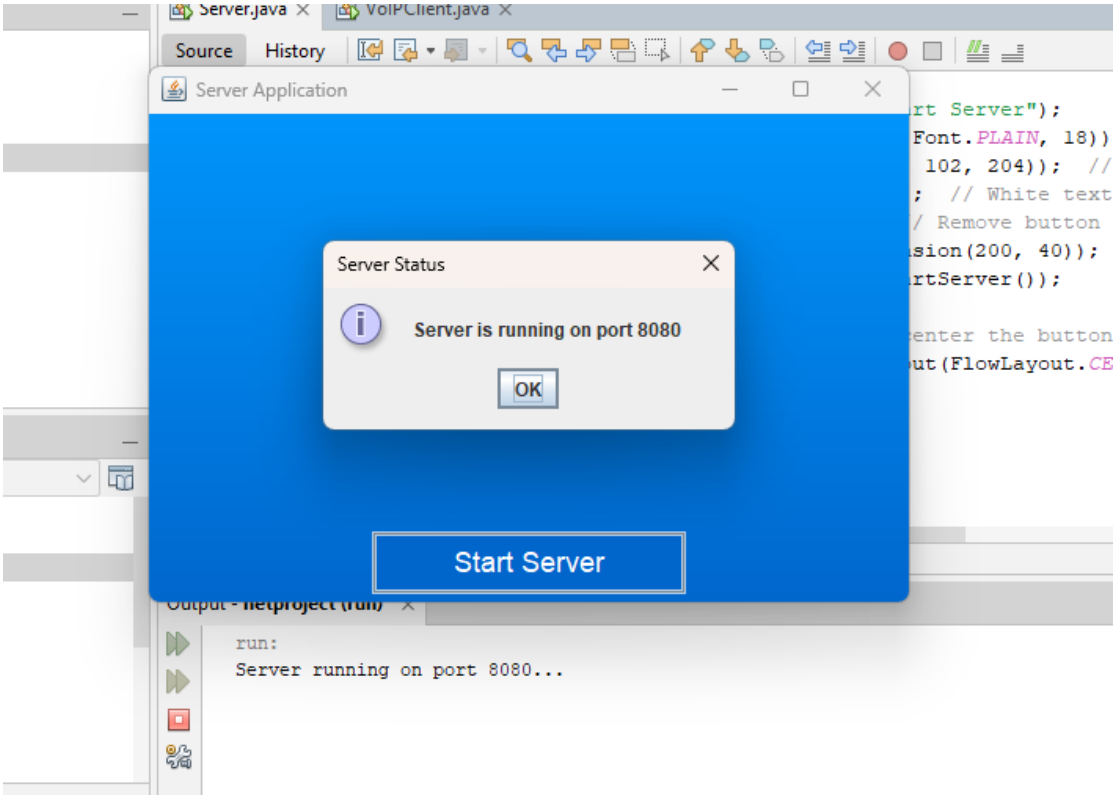

Figure 3.1: Sample output of server

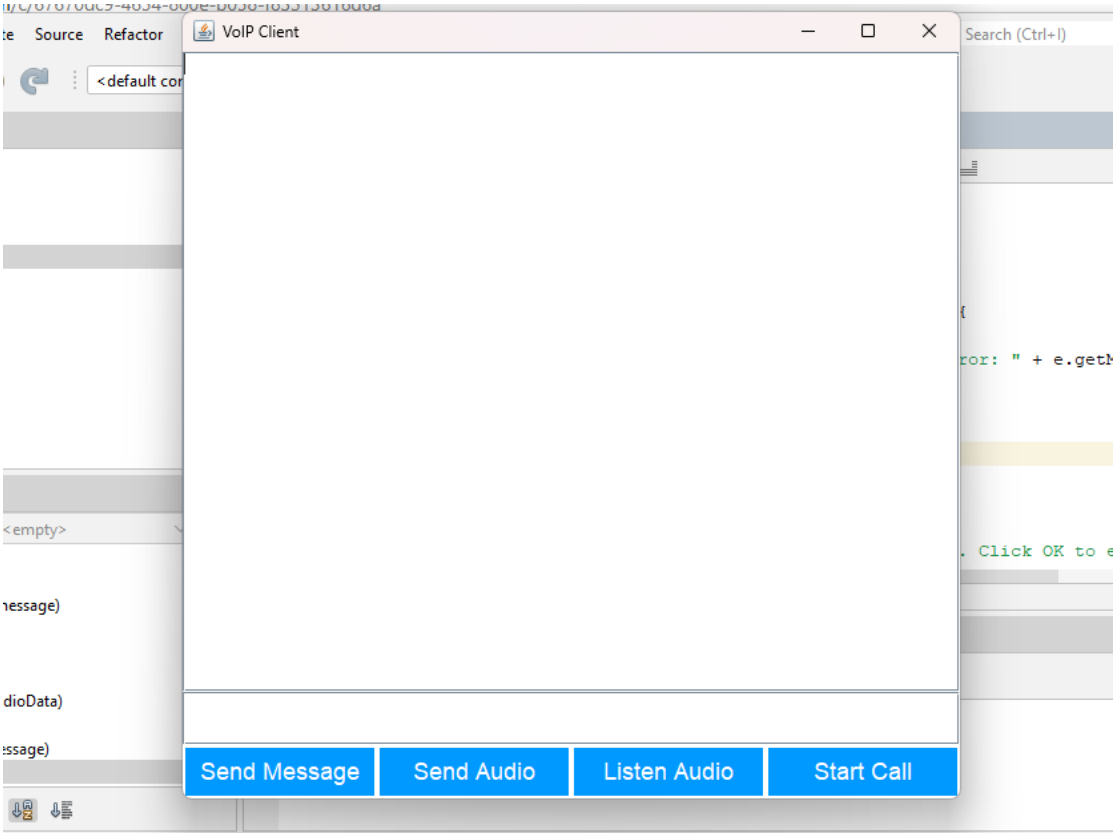

Figure 3.2: Interface of the project

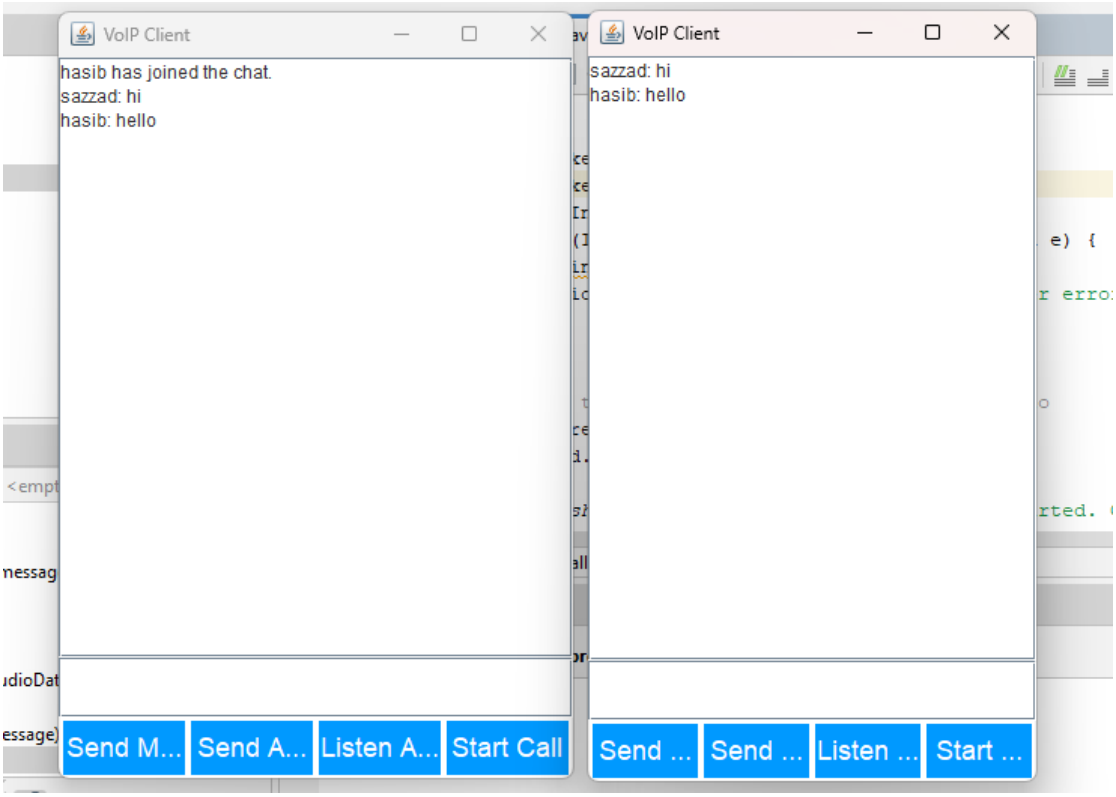

Figure 3.3: Send Message

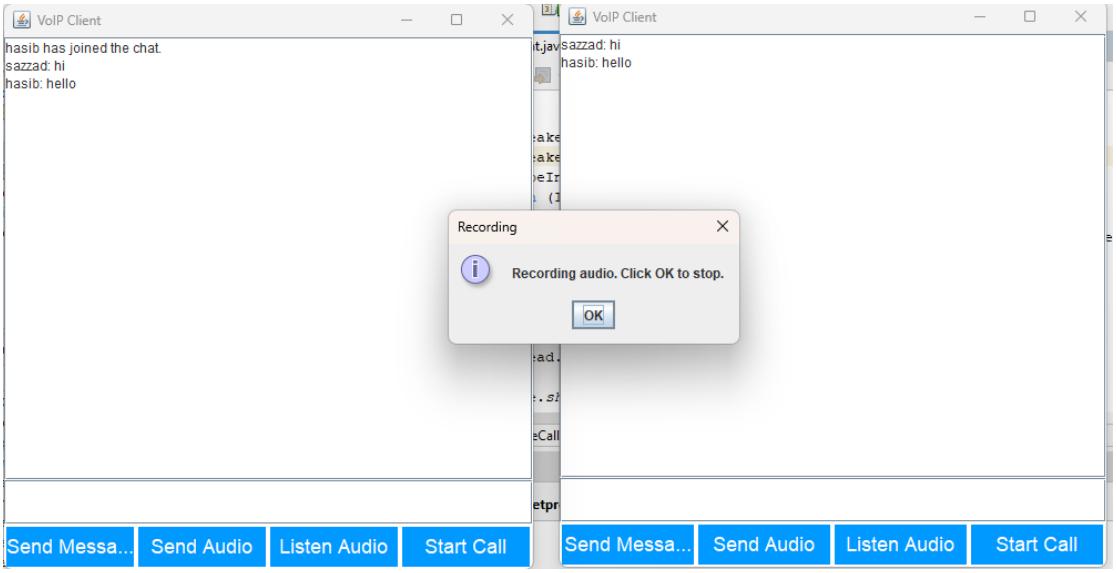

Figure 3.4: Recording Audio

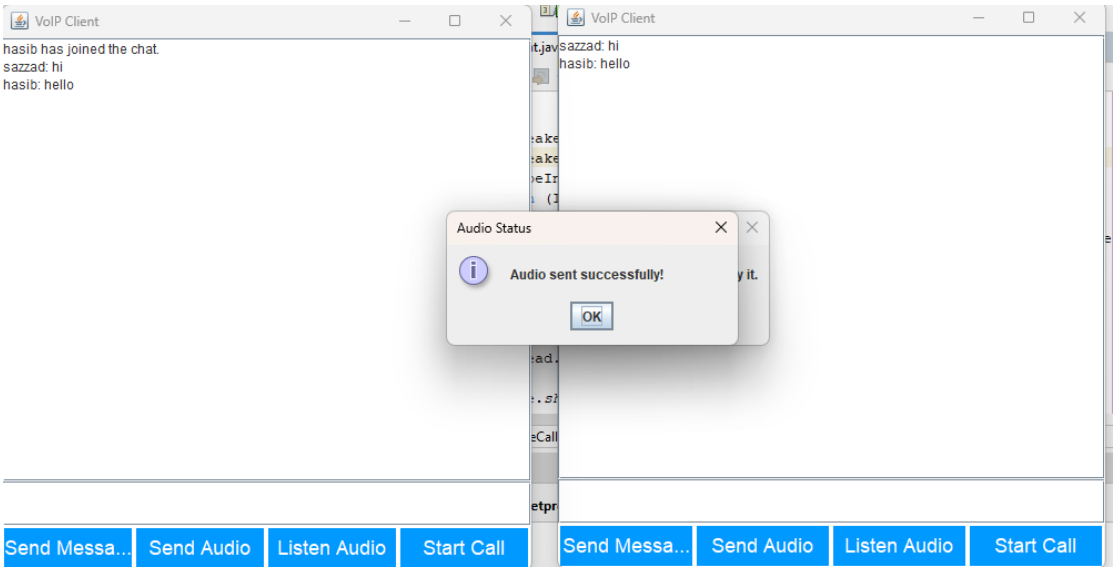

Figure 3.5: Send Audio

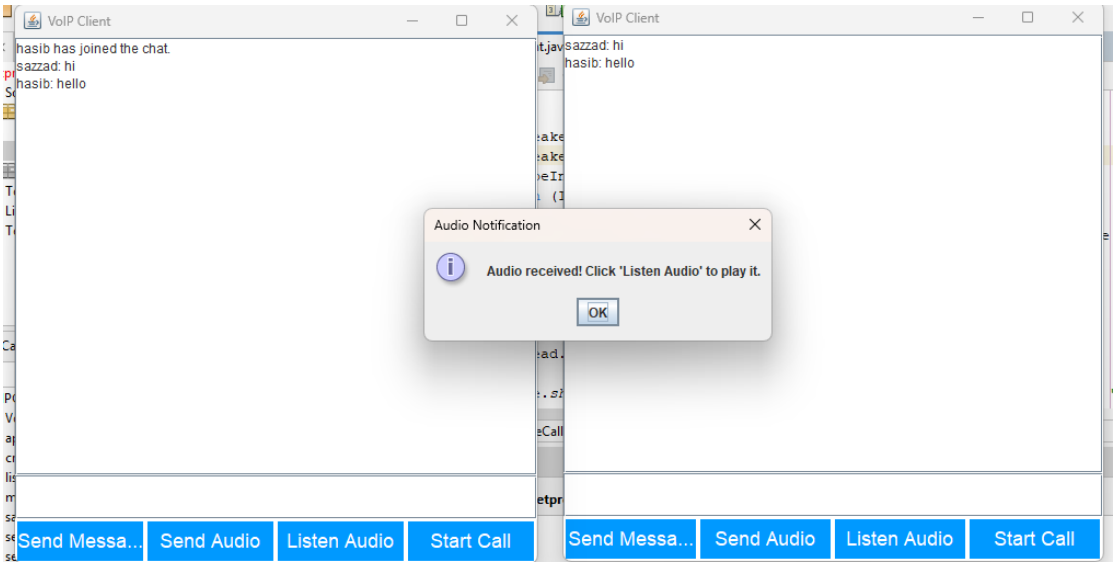

Figure 3.6: Listen Audio

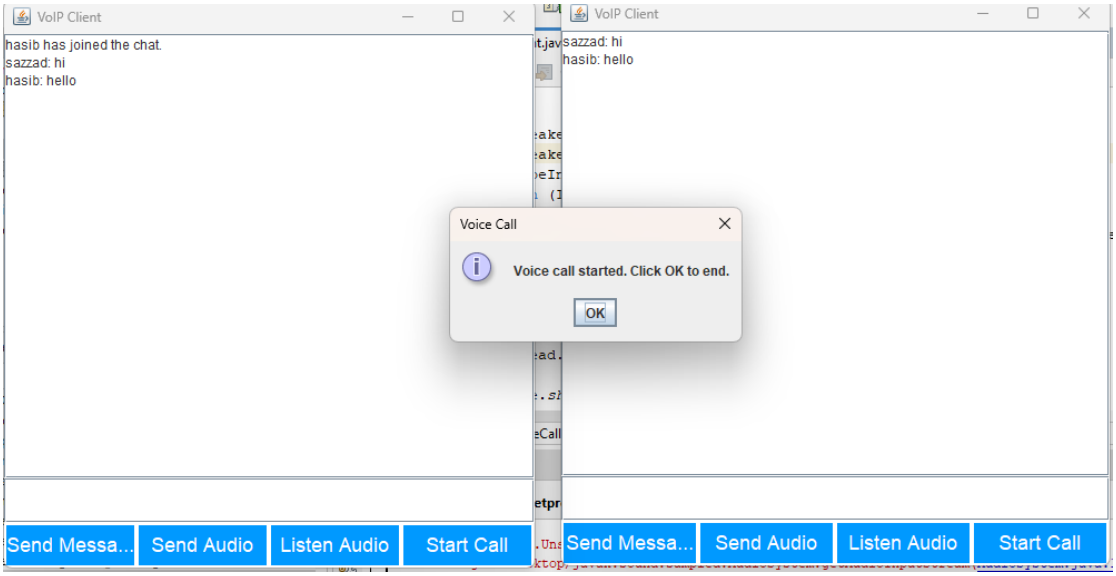

Figure 3.7: voice Call

Chapter 4

Conclusion

4.1 Discussion

This project created a simple VoIP system using Java. It successfully achieved real-time communication over a network and demonstrated how Java can handle audio processing and networking tasks efficiently. The project combined essential networking concepts with real-time audio streaming, making it an excellent example of practical Java application development. Added features like text messaging and audio recording provide more versatility, enhancing the user experience by offering multiple modes of interaction. Additionally, the system's modular design ensures easier scalability and integration of advanced features in future iterations.

4.2 Limitations

- Not suitable for large numbers of users.
- Requires a stable network for good performance.
- Basic audio quality, not suitable for high-fidelity use cases.

4.3 Scope of Future Work

- Use better audio compression to improve quality.
- Add encryption for secure communication.
- Scale the system to support more users.
- Enhance user experience with additional features like file sharing or video support.

References

1. Khasnabish, B. (2003). "Implementing Voice over IP" A practical guide to building VoIP systems, including design, protocols, and implementation challenges. Publisher: Wiley-IEEE Press. ISBN: 978-0471460527.

2. Douskalis, B. (1999). "IP Telephony: The Integration of Robust VoIP Services" This book provides an overview of VoIP technologies and their integration into networks. Publisher: Addison-Wesley. ISBN: 978-0201615891.