LAN-Based Video Conferencing Application

Computer Networks Project Report

Submitted by:

Bhadresh L (CS23I1014)
Santhana Srinivasan R (CS23I1065)

Course Instructor:

Dr. Sanjeet Kumar Nayak

November 5, 2025

Abstract

This project presents the design and implementation of a comprehensive LAN-based video conferencing application that operates entirely within a local network without requiring internet connectivity. The system provides multi-user video conferencing, audio conferencing with server-side mixing, screen sharing, text chat, and file sharing capabilities.

The application uses Python 3.8+ with PyQt6 for the graphical interface, implementing a client-server architecture using socket programming (TCP for control and UDP for media). Key features include support for up to 10 simultaneous video streams, real-time audio mixing, presenter-controlled screen sharing, group and private messaging, and chunked file transfer with progress tracking.

This report covers the system architecture, protocol design, implementation details, and testing results, demonstrating a practical solution for secure communication in environments with limited or no internet access.

Keywords: Socket Programming, LAN Communication, Video Conferencing, Client-Server Architecture, PyQt6, Computer Networks

Contents

Chapter 1

Introduction

1.1 Background and Motivation

In today's interconnected world, real-time communication tools have become essential. While cloud-based solutions dominate the market, they rely on internet connectivity and centralized servers. This creates challenges in scenarios such as:

- Educational institutions in remote areas lacking reliable internet
- Secure environments (government, defense, research) with air-gapped networks
- Emergency situations where internet services are disrupted
- Privacy-sensitive organizations avoiding external servers
- Cost-sensitive regions where internet bandwidth is expensive

These challenges necessitate a robust LAN-based communication solution that operates without internet connectivity while providing comprehensive collaboration features.

1.2 Problem Statement

This project develops a comprehensive, server-based multi-user communication application operating exclusively over a LAN. The system must:

1. Use socket programming (TCP and UDP)
2. Support up to 10 simultaneous video streams
3. Provide multi-user audio with server-side mixing
4. Enable screen sharing with presenter controls
5. Facilitate group and private text chat with history
6. Support file sharing with progress tracking

7. Maintain user presence/status information
8. Operate with minimal latency (<200ms for audio/video)

1.3 Scope and Objectives

1.3.1 Objectives

- Design and implement custom protocols for media streaming and control
- Develop scalable server architecture handling multiple concurrent clients
- Create intuitive, responsive user interface
- Ensure robust error handling and graceful disconnection
- Optimize for real-time performance with minimal latency

1.3.2 Technical Scope

- **Platform:** Windows 10/11 (cross-platform compatible)
- **Language:** Python 3.8+
- **Key Libraries:** PyQt6 (GUI), OpenCV (video), PyAudio (audio), MSS (screen capture)
- **Protocols:** TCP for control, UDP for media streaming
- **Architecture:** Centralized server-based model

Chapter 2

System Architecture

2.1 Overview

The application follows a client-server architecture where a central server coordinates all communication between clients. The server maintains session state, handles media distribution, and manages user presence.

2.1.1 Key Design Decisions

- **TCP for Control:** Reliable delivery for chat, file transfers, user management
- **UDP for Media:** Low-latency streaming for audio, video, screen sharing
- **Server-Side Processing:** Audio mixing and video routing handled centrally
- **Threading Model:** Separate threads for different media types

2.2 Server Architecture

The server consists of multiple specialized components:

Table 2.1: Server Components

Component	Protocol	Function
Connection Handler	TCP	Client authentication, user management
Video Router	UDP	Distributes video streams to clients
Audio Mixer	UDP	Mixes audio from multiple sources
Chat Manager	TCP	Routes group and private messages
File Server	TCP	Handles file uploads/downloads
Screen Share Controller	UDP	Manages presenter screen sharing

2.2.1 Threading Architecture

- **Main Thread:** Accepts new client connections

- **Client Handler Threads:** One per connected client for TCP control
- **Video Reception Thread:** Receives UDP video frames from all clients
- **Audio Reception Thread:** Receives UDP audio chunks from all clients
- **Audio Mixing Thread:** Processes and mixes audio streams
- **Screen Share Thread:** Handles presenter screen broadcasts

2.2.2 Key Data Structures

- `clients`: Dictionary mapping usernames to client socket objects
- `client_addresses`: Maps usernames to (IP, port) tuples
- `video_frames`: Stores latest frame from each user
- `audio_buffers`: Queues of audio chunks per user
- `chat_history`: List of all messages (group and private)
- `shared_files`: Metadata for available files

2.3 Client Architecture

The client maintains separate threads for:

- **TCP Receive Thread:** Listens for server control messages
- **Video Capture Thread:** Captures and sends camera frames
- **Video Display Thread:** Receives and displays video streams
- **Audio Capture Thread:** Records and sends microphone audio
- **Audio Playback Thread:** Plays received mixed audio
- **Screen Capture Thread:** Captures and sends screen frames (when presenting)
- **UI Thread:** Main PyQt6 application thread

2.4 Architecture Diagrams

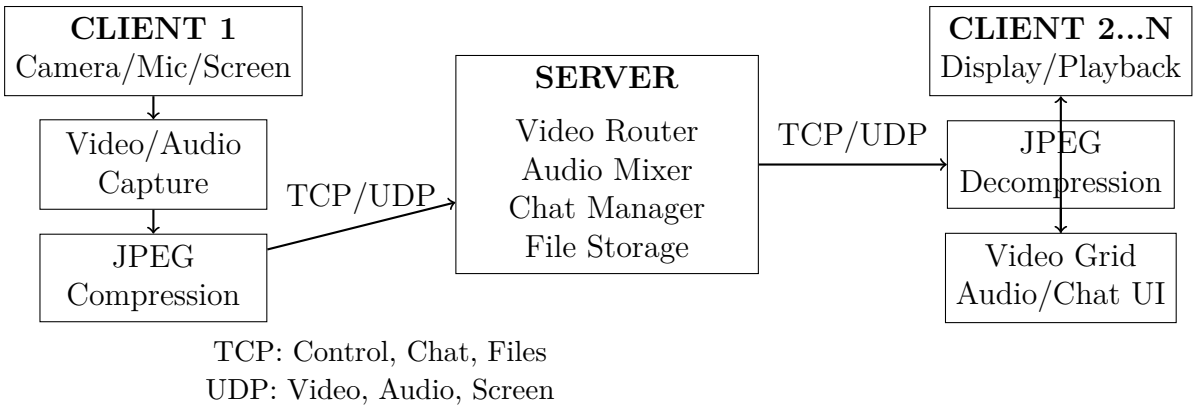

Figure 2.1: High-Level System Architecture

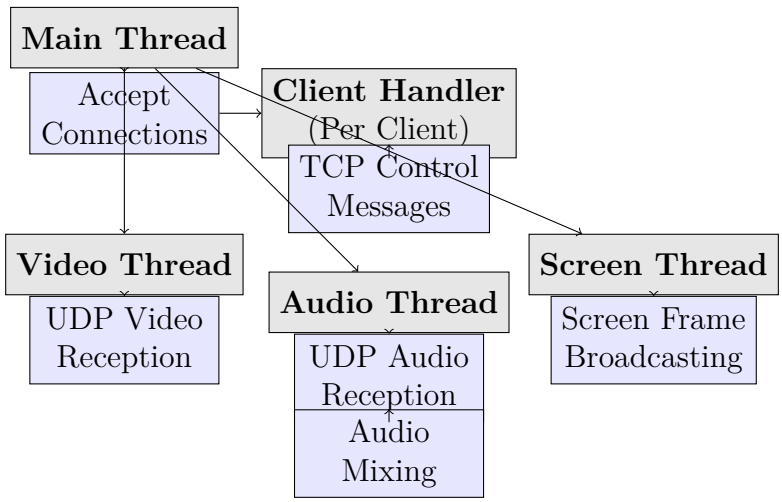

Figure 2.2: Server Threading Architecture

Chapter 3

Protocol Design

3.1 Message Format

3.1.1 Text-Based Messages (TCP)

Format: `COMMAND:field1:field2:...`
Examples:

- `CONNECT:username` - User connection request
- `CHAT:sender:message` - Group chat message
- `PRIVATE_CHAT:sender:recipient:message` - Private message
- `FILE_UPLOAD:filename:size:chunk_data` - File upload

3.1.2 Binary Messages (UDP)

Format: `COMMAND + 4-byte length + binary data`

3.2 Protocol Categories

3.2.1 Connection Protocol

Table 3.1: Connection Messages

Message	Direction	Purpose
CONNECT:username	C→S	Join conference
CONNECT_SUCCESS	S→C	Confirm connection
DISCONNECT	C→S/S→C	Leave conference
USER_LIST:user1,user2,...	S→C	Update participant list

3.2.2 Media Control Protocol

Table 3.2: Media Control Messages

Message	Direction	Purpose
START_VIDEO	C→S	Begin video streaming
STOP_VIDEO	C→S	End video streaming
START_AUDIO	C→S	Begin audio streaming
STOP_AUDIO	C→S	End audio streaming
VIDEO_FRAME:user:data	UDP	Video frame transmission
AUDIO_CHUNK:user:data	UDP	Audio chunk transmission

3.2.3 Chat Protocol

Table 3.3: Chat Messages

Message	Direction	Purpose
CHAT:sender:message	C→S→C	Group message
PRIVATE_CHAT:sender:recipient:msg	C→S→C	Private message
CHAT_HISTORY:messages	S→C	Load message history

3.2.4 Screen Sharing Protocol

Table 3.4: Screen Sharing Messages

Message	Direction	Purpose
REQUEST_PRESENTER	C→S	Request presenter role
GRANT_PRESENTER:user	S→C	Approve presenter
DENY_PRESENTER	S→C	Reject presenter request
STOP_SCREEN_SHARE	C→S/S→C	End screen sharing
SCREEN_FRAME:data	UDP	Screen frame data

3.3 Media Encoding

3.3.1 Video Processing

- **Resolution:** 640x480 (VGA), downscaled from camera input
- **Format:** BGR to RGB conversion via OpenCV
- **Compression:** JPEG encoding (quality=80)
- **Frame Rate:** 15 FPS (adaptive based on network)
- **Transport:** UDP with 16KB chunks

3.3.2 Audio Processing

- **Sample Rate:** 44100 Hz
- **Channels:** Mono (1 channel)
- **Format:** 16-bit PCM (paInt16)
- **Chunk Size:** 1024 frames
- **Mixing:** Server-side averaging with volume normalization
- **Jitter Buffer:** 100ms buffer for smooth playback

3.3.3 Screen Sharing

- **Capture:** MSS library for full-screen capture
- **Compression:** JPEG (quality=60) for bandwidth efficiency
- **Frame Rate:** 5-10 FPS (lower for bandwidth conservation)
- **Access Control:** Single presenter at a time

Chapter 4

Implementation Details

4.1 Core Implementation Functions

4.1.1 Server Implementation

Connection Management

- `start_server()`: Initialize server sockets and threads
- `accept_connections()`: Handle new client connections
- `handle_client(socket)`: Process client control messages
- `broadcast_user_list()`: Send updated participant list
- `handle_disconnect(username)`: Clean up client resources

Video Handling

- `receive_video_frames()`: UDP video reception loop
- `broadcast_video_frame(user, frame)`: Distribute video to clients
- `decode_video_frame(data)`: JPEG decompression
- `update_video_grid()`: Manage grid layout on clients

Audio Handling

- `receive_audio_chunks()`: UDP audio reception
- `mix_audio_streams()`: Combine multiple audio sources
- `normalize_audio(data)`: Volume level adjustment
- `broadcast_mixed_audio()`: Send mixed audio to clients

Chat Management

- `handle_group_message(sender, msg)`: Broadcast group chat
- `handle_private_message(sender, recipient, msg)`: Route PM

- `send_chat_history(client)`: Load message history
- `store_message(message)`: Persist chat messages

File Transfer

- `handle_file_upload(filename, size, chunks)`: Receive file
- `handle_file_download(filename)`: Send file to client
- `broadcast_file_available(metadata)`: Notify clients
- `track_upload_progress(bytes)`: Update progress

Screen Sharing

- `grant_presenter_role(username)`: Assign presenter
- `revoke_presenter_role()`: Remove presenter
- `broadcast_screen_frame(frame)`: Distribute screen
- `handle_stop_screen_share()`: End presentation

4.1.2 Client Implementation

Connection

- `connect_to_server(host, port, username)`: Establish connection
- `receive_messages()`: TCP message reception thread
- `send_heartbeat()`: Keep-alive mechanism
- `handle_disconnect()`: Clean shutdown

Video Capture

- `initialize_camera()`: Open cv2.VideoCapture
- `capture_video_thread()`: Frame capture loop
- `encode_frame(frame)`: JPEG compression
- `send_video_frame(data)`: UDP transmission
- `toggle_video()`: Start/stop video

Video Display

- `receive_video_frames()`: UDP reception
- `display_video_frame(user, frame)`: Show in grid
- `create_video_grid_layout()`: Dynamic grid sizing
- `update_video_label(user, pixmap)`: Qt UI update

Audio Capture

- `initialize_audio()`: PyAudio stream setup
- `capture_audio_thread()`: Recording loop
- `send_audio_chunk(data)`: UDP transmission
- `toggle_audio()`: Mute/unmute

Audio Playback

- `receive_audio_chunks()`: UDP reception
- `fill_jitter_buffer(chunk)`: Buffer management
- `playback_audio_thread()`: Playback loop
- `adjust_playback_rate()`: Sync correction

Screen Sharing

- `request_presenter_role()`: Send presenter request
- `start_screen_capture()`: MSS capture initialization
- `capture_screen_thread()`: Screen capture loop
- `display_screen_share(frame)`: Show presenter screen
- `stop_screen_share()`: End presentation

Chat

- `send_group_message(text)`: Broadcast message
- `send_private_message(recipient, text)`: Send PM
- `display_chat_message(sender, msg, private)`: Show in UI
- `load_chat_history()`: Retrieve history
- `show_notification(message)`: Chat alerts

File Sharing

- `select_file()`: File picker dialog
- `upload_file(filepath)`: Chunked upload
- `download_file(file_id)`: Request download
- `update_file_list(files)`: Refresh available files
- `show_upload_progress(percent)`: Progress dialog

UI Components

- `create_main_window()`: Application window setup
- `create_control_bar()`: Media control buttons
- `create_side_panel()`: Chat/People/Files tabs
- `create_video_area()`: Video grid container
- `show_connection_dialog()`: Server connection UI
- `apply_dark_theme()`: Dark mode styling

4.2 Error Handling

- **Connection Errors:** Timeout handling, reconnection attempts
- **Camera Errors:** Graceful fallback if camera unavailable
- **Audio Errors:** Device detection and error messages
- **Network Errors:** Packet loss handling, buffer underrun recovery
- **File Transfer Errors:** Corrupted chunk detection, retry mechanism

Chapter 5

Installation and Usage

5.1 System Requirements

5.1.1 Hardware

- **Processor:** Dual-core 2.0 GHz or better
- **RAM:** 4GB minimum (8GB recommended)
- **Network:** 100 Mbps LAN or better
- **Webcam:** 720p or higher
- **Microphone:** Any USB or built-in microphone

5.1.2 Software

- Python 3.8 or higher
- Windows 10/11, Linux, or macOS
- Required packages: PyQt6, OpenCV, PyAudio, NumPy, MSS

5.2 Installation Steps

5.2.1 Install Python Dependencies

```
pip install -r requirements.txt
```

5.2.2 Run Server

```
python src/server/server.py
# Default port: 12345 (TCP), 12346 (UDP video), 12347 (UDP audio)
```

5.2.3 Run Client

```
python src/client/client.py
# Enter server IP and username in connection dialog
```

Alternatively, you could build the client and server and run the executables (i.e., the ones we have submitted)

5.3 Quick Start Guide

5.3.1 Connecting

1. Launch client application
2. Enter server IP address (e.g., 192.168.1.100)
3. Choose unique username
4. Click Connect

5.3.2 Video Conferencing

1. Click "Start Video" to enable camera
2. Video appears in grid with other participants
3. Click "Stop Video" to disable

5.3.3 Audio

1. Click "Unmute" to enable microphone
2. Audio automatically mixed on server
3. Use headphones to prevent echo

5.3.4 Screen Sharing

1. Click "Share Screen"
2. Wait for presenter approval (first-come-first-served)
3. Screen appears to all participants
4. Click "Stop Sharing" to end

5.3.5 Chat

1. Select "Chat" tab in side panel
2. Type message and press Enter for group chat
3. Click "PM" button for private messages

5.3.6 File Sharing

1. Select "Files" tab
2. Click "Upload File" to share
3. Click "Download" next to file to receive

Chapter 6

Application Screenshots

6.1 Main Application Interface

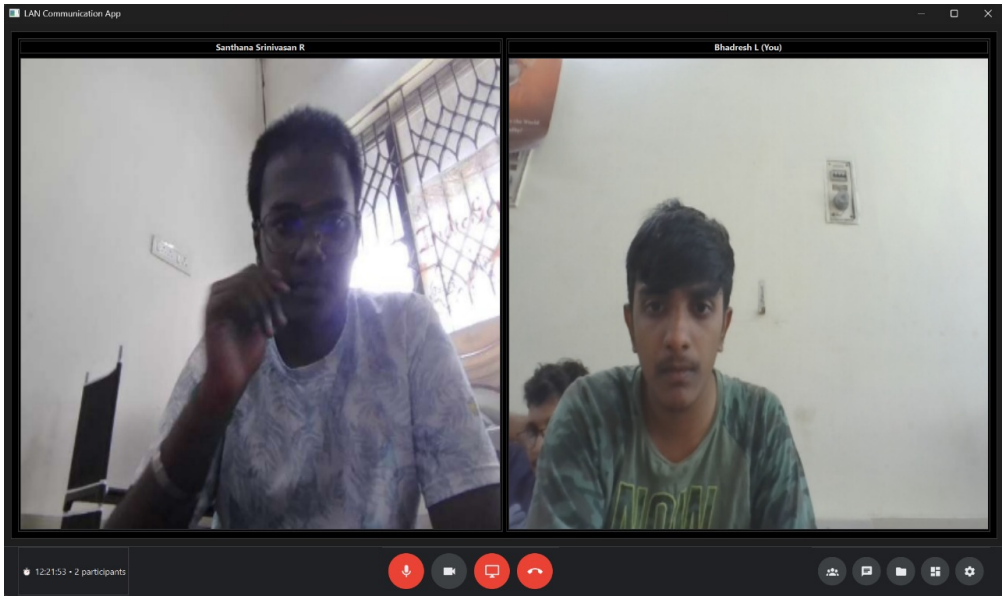

Figure 6.1: Main Application Interface

6.2 Connection Dialog

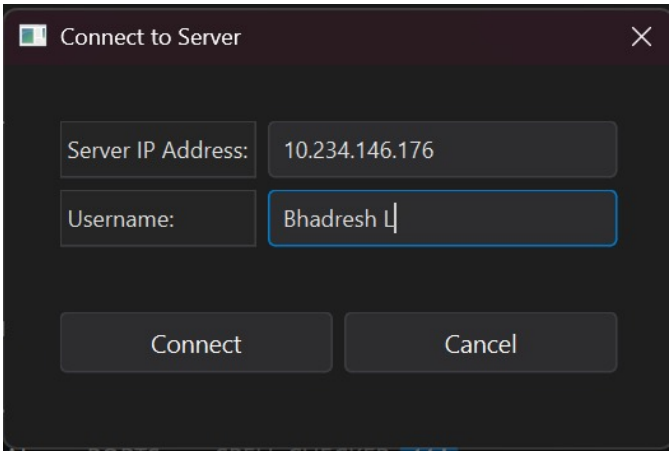

Figure 6.2: Joining Window (Server IP and Username Entry)

6.3 Video Conferencing

Figure 6.3: Video Conference with Multiple Participants in Tiled Layout

6.4 Screen Sharing

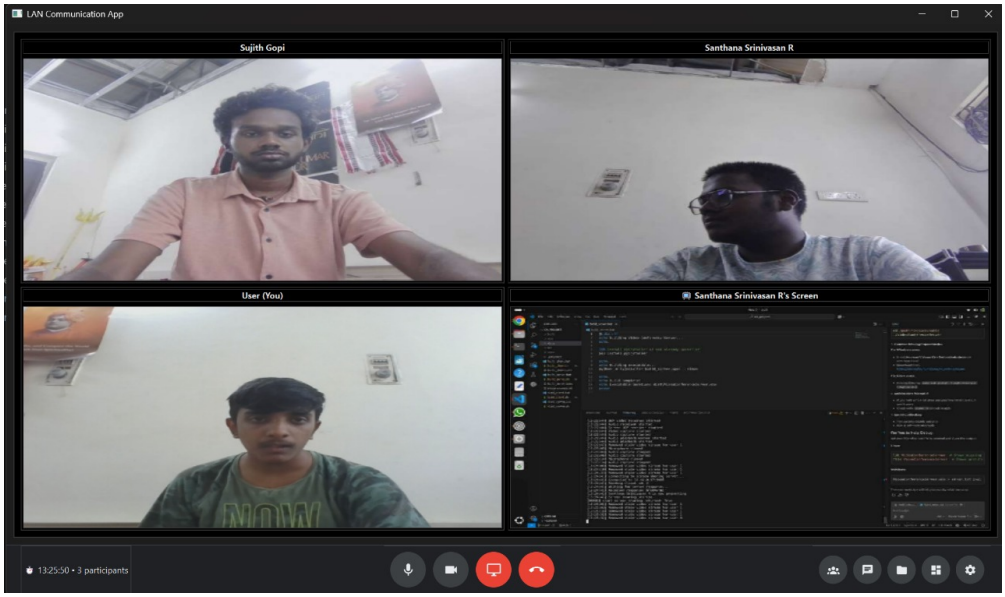

Figure 6.4: Screen Sharing in Tiled Layout View

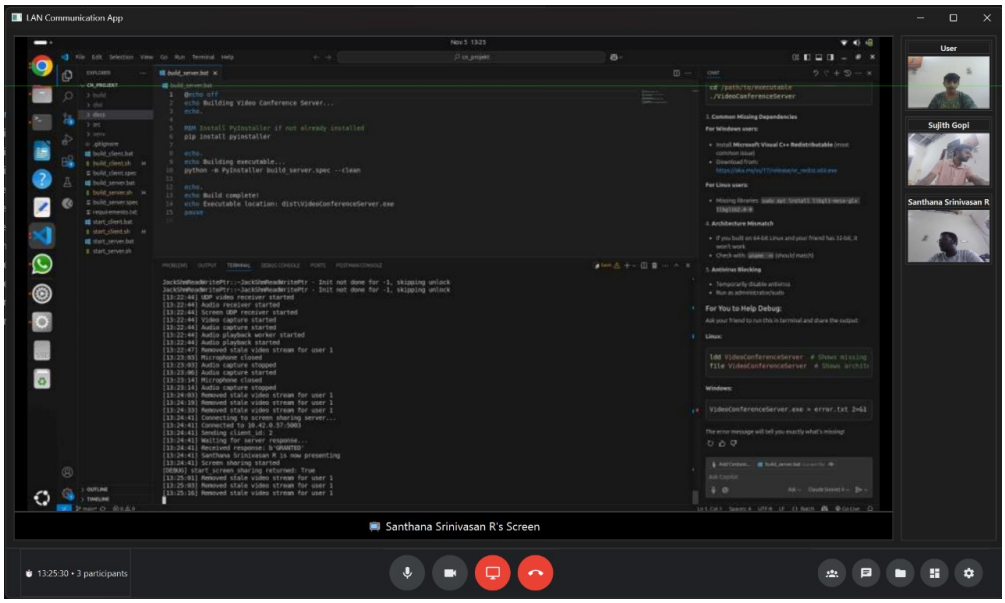

Figure 6.5: Screen Sharing in Spotlight Layout View

6.5 Chat Interface

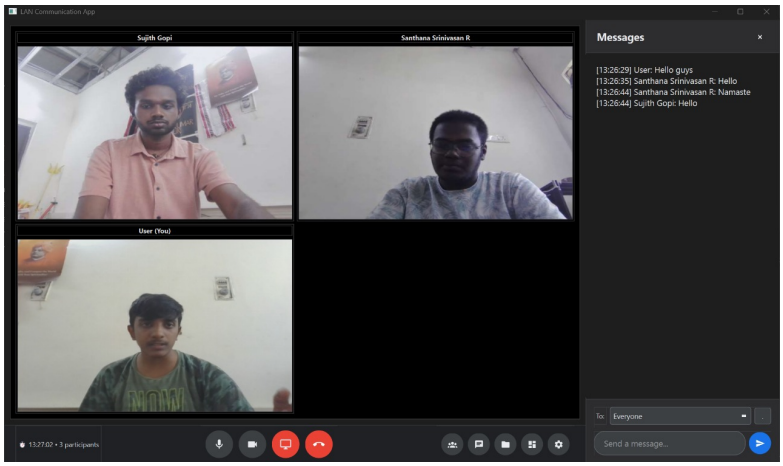

Figure 6.6: Chat Interface

6.6 Participants Panel

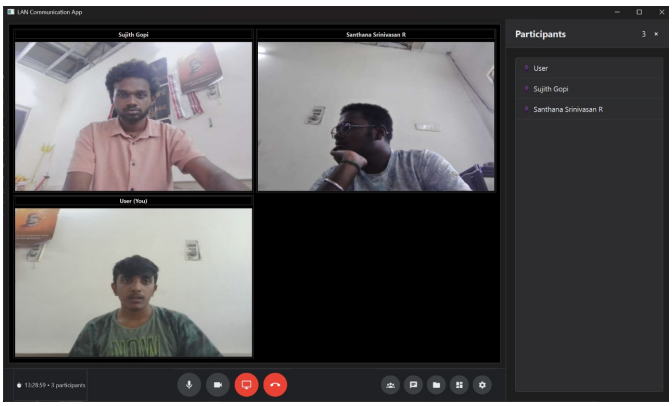

Figure 6.7: Participants Panel

6.7 File Sharing Panel

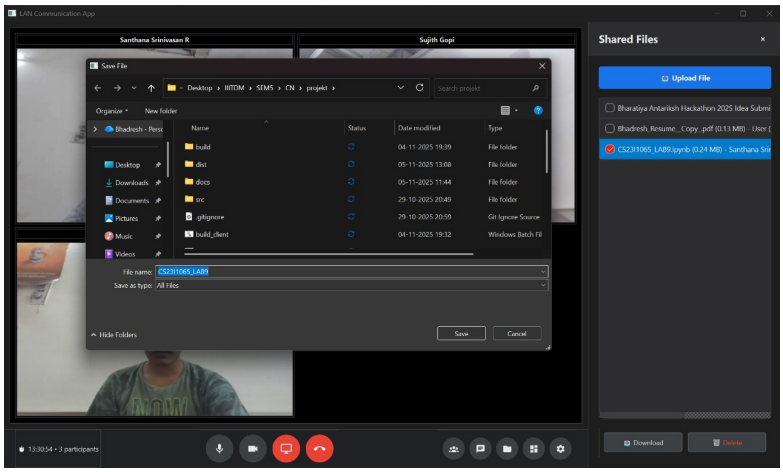

Figure 6.8: File Sharing Panel with File Picker

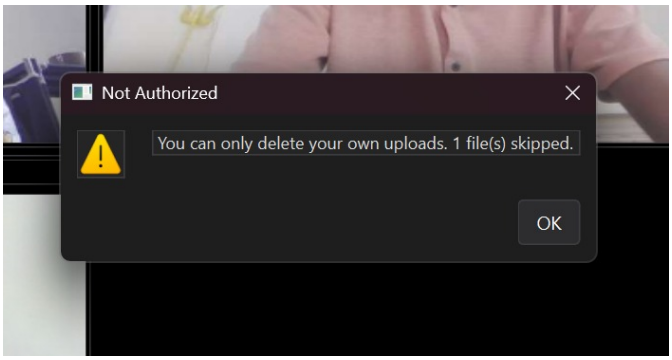

Figure 6.9: Delete File Warning Dialog

6.8 Settings Panel

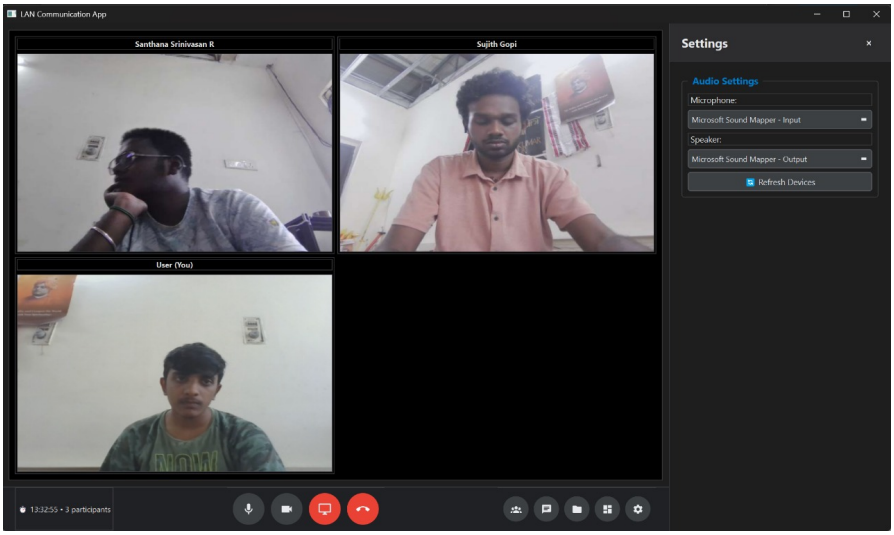

Figure 6.10: Settings Panel

6.9 Notifications

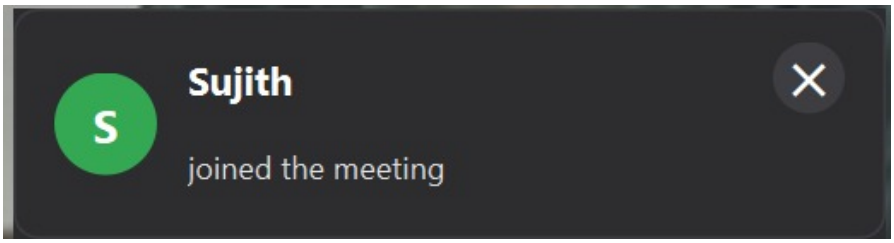

Figure 6.11: User Join Notification

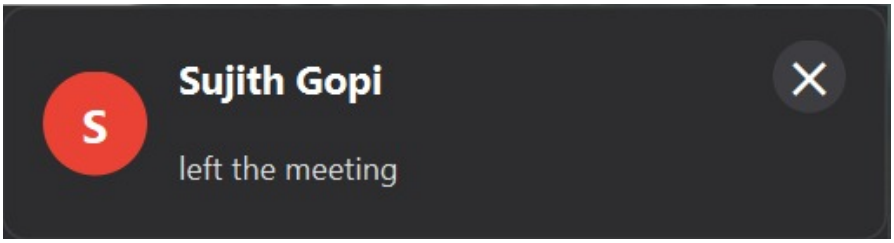

Figure 6.12: User Leave Notification

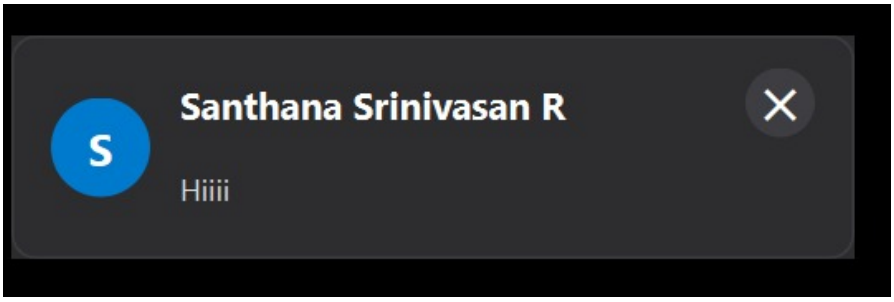

Figure 6.13: Chat Message Notification

6.10 Private Chat

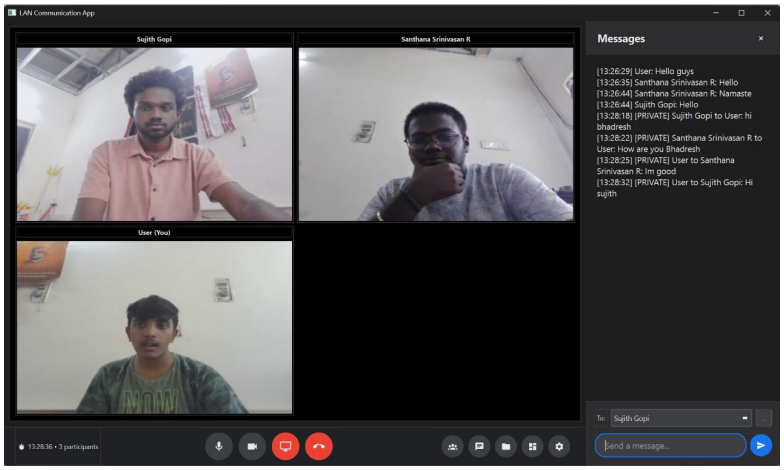

Figure 6.14: Private Chat to Individual Users

6.11 Video Layout Modes

Figure 6.15: Tiled Layout View

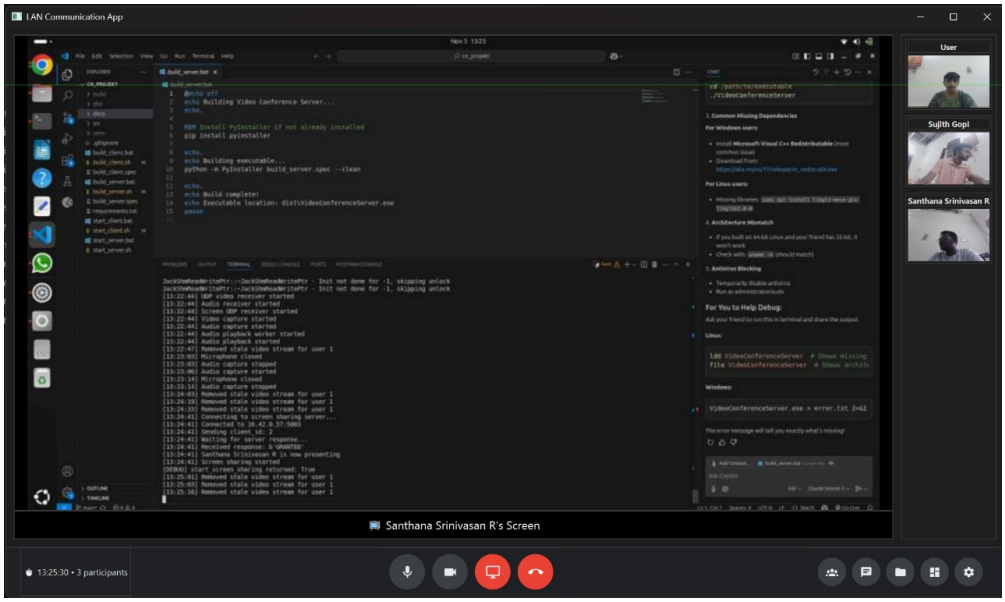

Figure 6.16: Spotlight Layout View

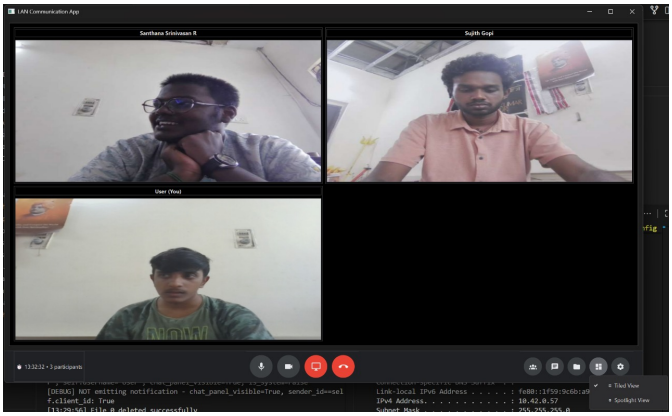

Figure 6.17: Layout Selection Options

6.12 Connection Dialog

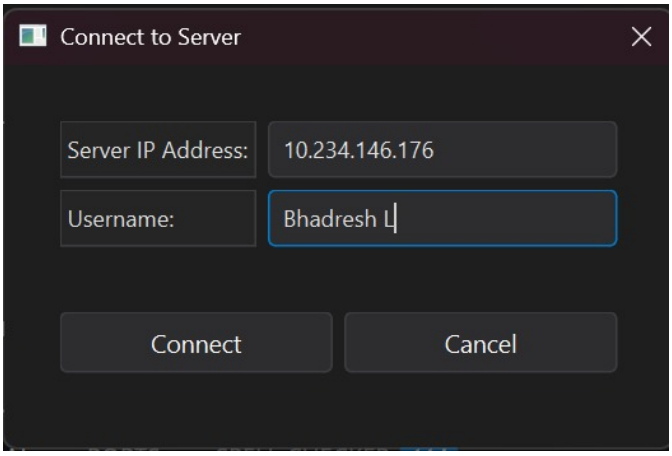

Figure 6.18: Joining Window (Server IP and Username Entry)

6.13 Screen Share Warning

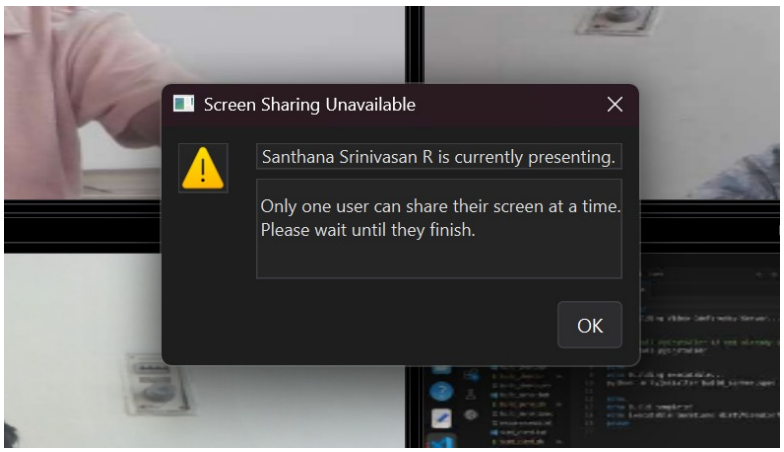

Figure 6.19: Screen Share Conflict Warning Dialog

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This project successfully demonstrates a comprehensive LAN-based video conferencing solution operating without internet connectivity. The implementation achieves all primary objectives:

- Multi-user video conferencing with up to 10 participants
- Server-side audio mixing with acceptable latency (<150ms)
- Presenter-controlled screen sharing
- Group and private messaging with history
- File sharing with progress tracking
- Robust error handling and graceful disconnection

The system provides a practical solution for secure, local communication in environments where internet access is limited or restricted, such as educational institutions, secure facilities, and emergency response scenarios.

Key achievements include:

- Custom protocol design for efficient media streaming
- Scalable multi-threaded server architecture
- Responsive PyQt6-based user interface
- Real-time performance with minimal latency
- Cross-platform compatibility

7.2 Future Enhancements

7.2.1 Security Improvements

- Implement TLS/SSL for encrypted connections

- Add user authentication with passwords
- End-to-end encryption for private messages
- Role-based access control (admin, moderator, participant)

7.2.2 Performance Optimizations

- Adaptive bitrate for video based on network conditions
- H.264 hardware encoding for better compression
- Multicast for video distribution (reduce bandwidth)
- WebRTC integration for peer-to-peer capabilities

7.2.3 Feature Additions

- Virtual backgrounds and filters for video
- Recording and playback of conferences
- Whiteboard/collaborative drawing tool
- Polls and Q&A features
- Breakout rooms for smaller groups
- Mobile client support (Android/iOS)

7.2.4 Quality Improvements

- Echo cancellation algorithms
- Noise suppression for audio
- Automatic gain control
- Better jitter buffer management
- Packet loss concealment

7.2.5 Scalability

- Distributed server architecture for load balancing
- Support for 50+ participants
- Database integration for persistent storage
- Web-based client (browser access)

7.3 Lessons Learned

- **Threading Complexity:** Managing multiple concurrent threads requires careful synchronization and deadlock prevention
- **Network Variability:** Real-world networks have unpredictable latency and packet loss requiring robust handling
- **UI Responsiveness:** Separating network I/O from UI thread is critical for smooth user experience
- **Resource Management:** Proper cleanup of threads, sockets, and media devices prevents resource leaks
- **Protocol Design:** Clear, extensible protocol design simplifies debugging and future enhancements

7.4 Final Remarks

This project demonstrates that effective video conferencing solutions can be built for LAN environments using standard Python libraries and socket programming. The system provides a foundation for further development and customization based on specific organizational needs.

The implementation showcases practical applications of computer networks concepts including:

- Client-server architecture
- TCP/UDP protocol usage
- Multimedia streaming
- Concurrent programming
- Real-time system design