Sapora - Complete Technical Documentation

LAN Video Conferencing Suite

PONNURU KARTHIK-CS23B1045
ALUGUBELLI LAXMI SUSHANTH REDDY-CS23B1046

Table of Contents

Executive Summary

Overview

Sapora is a **LAN-based video conferencing application** designed for secure, high-performance real-time collaboration within local area networks. It provides enterprise-grade features similar to Zoom but optimized for trusted LAN environments without requiring internet connectivity.

Key Features

Core Capabilities

- **Real-time Audio Conferencing** with automatic server-side mixing
- **Multi-participant Video Streaming** with dynamic grid layout
- **Text Chat** with broadcast and private messaging

- **File Transfer** with MD5 integrity verification
- **Screen Sharing** for presentations and collaboration
- **LAN Discovery** for automatic server detection
- **Meeting Scheduler** for organizing sessions
- **Multi-room Support** for parallel meetings
- **User Management** with real-time participant list
- **Connection Management** with heartbeat monitoring

Technology Stack

Backend: Python 3.10+, Socket Programming (TCP/UDP), PyQt6, OpenCV, PyAudio, mss
Protocols: TCP (control/files), UDP (audio/video), WebSocket (optional)
Key Libraries: numpy, opencv-python, pyaudio, PyQt6, flask-socketio

System Requirements

Minimum: Dual-core 2.0 GHz CPU, 4 GB RAM, 100 Mbps LAN
Recommended: Quad-core 2.5 GHz+ CPU, 8 GB RAM, Gigabit LAN (wired)

System Architecture

High-Level Architecture

CLIENT LAYER

├── PyQt6 Main UI (main_ui.py)

│ ├── Login Dialog with LAN Discovery

│ ├── Multi-tile Video Grid (Dynamic Layout)

│ ├── Chat Panel (Broadcast/Private)

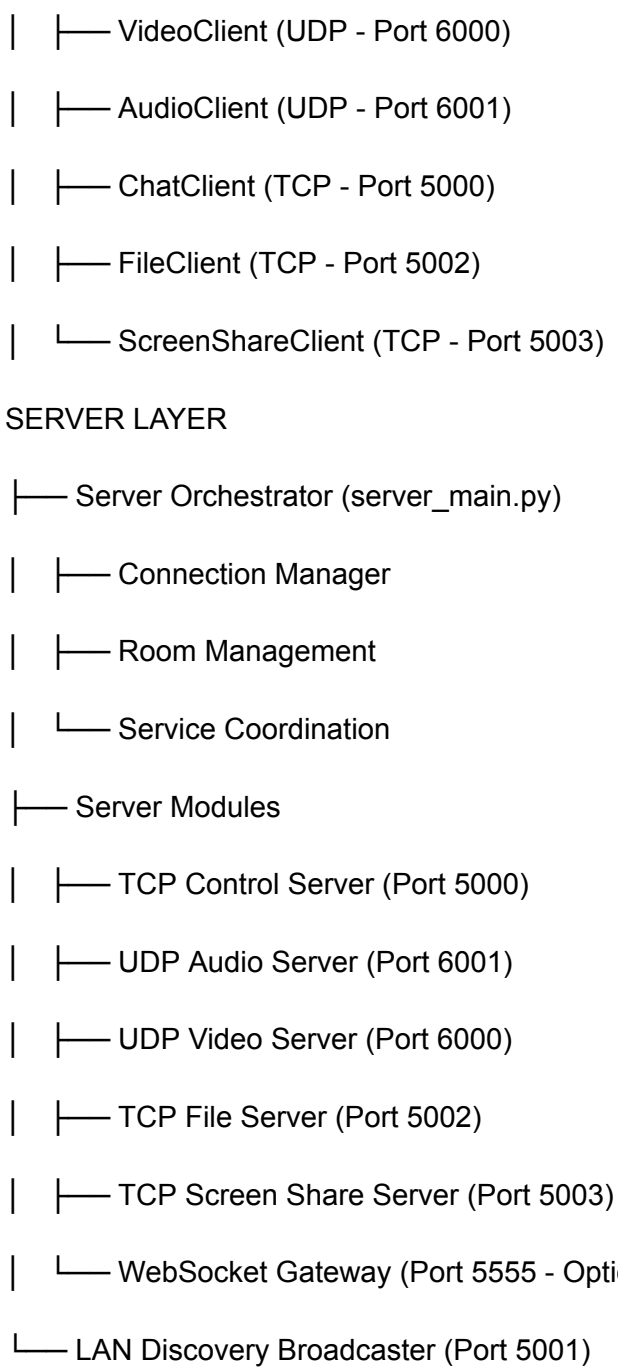
│ ├── Control Bar (Mute, Video, Screen, Chat, Files, End)

│ ├── Screen Share Viewer

│ ├── File Transfer Dialogs

│ └── Meeting Scheduler

├── Client Modules



Port Allocation

Port	Protocol	Service
5000	TCP	Control Server
5001	UDP	LAN Discovery
5002	TCP	File Transfer
5003	TCP	Screen Share

Port	Protocol	Service
5555	WebSocket	Electron Gateway
6000	UDP	Video Streaming
6001	UDP	Audio Streaming

Component Descriptions

Server Components

Connection Manager - Tracks all client connections, maintains username-to-socket mappings, manages room assignments, monitors heartbeats (3s interval), handles timeouts (15s idle)

TCP Control Server - Handles registration, routes chat messages (broadcast/unicast), implements case-insensitive username matching, broadcasts user list updates

UDP Audio Server - Receives audio chunks (PCM 44.1kHz), implements jitter buffers (max 200ms), mixes audio (averaging), broadcasts to room participants, excludes sender's audio

UDP Video Server - Receives JPEG frames, relays to room participants, excludes sender from broadcast

TCP File Server - Handles chunked transfers (32KB), verifies MD5 checksums, notifies recipients, prevents path traversal, enforces 100MB limit

TCP Screen Share Server - Manages presenter/viewer connections, relays JPEG frames, handles stop signals

Client Components

Main UI - PyQt6 interface with login dialog, multi-tile video grid, chat panel, control bar, screen share viewer, file dialogs, meeting scheduler

Video Client - OpenCV capture (640×480, 15 FPS), JPEG compression (quality 55), UDP streaming

Audio Client - PyAudio capture (44.1kHz, mono, 1024 samples), UDP streaming, playback of mixed audio

Chat Client - TCP connection, JSON messages, broadcast/private messaging, callbacks for UI

File Client - TCP connection, chunked transfer, progress tracking, MD5 verification

Screen Share Client - Presenter mode (mss capture, JPEG compression), Viewer mode (display frames)

Communication Protocols

Message Format

All messages use a standardized 10-byte header
Header:
Version (1 byte)
Message Type (1 byte)
Payload Length (4 bytes)
Sequence Number (2 bytes)
Payload Data (N bytes)

Struct Format: `!BBIHH` (Network byte order, big-endian)

Message Types

Control Messages (TCP - Port 5000)

- `0x01` CMD_REGISTER - Client registration
- `0x02` CMD_HEARTBEAT - Keep-alive (every 3s)
- `0x03` CMD_USER_LIST - User list broadcast
- `0x04` CMD_DISCONNECT - Disconnect notification

Chat Messages (TCP - Port 5000)

- `0x10` MSG_CHAT - Text message (JSON)

File Transfer (TCP - Port 5002)

- `0x20` FILE_METADATA - File info
- `0x21` FILE_CHUNK - File data (32KB)
- `0x22` FILE_REQUEST_UPLOAD - Upload request
- `0x23` FILE_REQUEST_DOWNLOAD - Download request
- `0x24` FILE_ACK_SUCCESS - Success ACK
- `0x25` FILE_ACK_FAILURE - Failure notification

Screen Share (TCP - Port 5003)

- `0x30` SCREEN_FRAME - Screen frame (JPEG)
- `0x31` SCREEN_START - Start sharing
- `0x32` SCREEN_STOP - Stop sharing

Streaming (UDP)

- `0x40` STREAM_VIDEO - Video frame (Port 6000)
- `0x41` STREAM_AUDIO - Audio chunk (Port 6001)

Protocol Examples

Registration: Client → Server: {"username": "John Doe", "meeting_id": "team_meeting"}

Server → All: {"users": [{"username": "John Doe", "ip": "192.168.1.10"}]}

Chat (Broadcast):{

"sender": "John Doe",

"target": "all",

"text": "Hello everyone!",

"meeting_id": "team_meeting",

"timestamp": 1699123456.789

}

Chat (Private):{

"sender": "John Doe",

"target": "Jane Smith",

"text": "Private message",

"meeting_id": "team_meeting",

"timestamp": 1699123456.789

}

Features Documentation

1. Audio Conferencing

Specifications: 44.1kHz, 16-bit PCM mono, 1024 samples/chunk (~23ms), ~705 kbps, 50-100ms latency

How It Works:

1. PyAudio captures microphone audio
2. Sent as 1024-sample chunks via UDP
3. Server buffers in jitter buffers (max 200ms)
4. Server mixes by averaging samples
5. Server broadcasts mixed audio (excluding sender's own)
6. Client plays through speakers

Features: Automatic mixing, jitter buffering, room isolation, low latency, mute control

Usage: Click microphone icon to toggle mute (green=unmuted, red=muted)

2. Video Conferencing

Specifications: 640×480, 15 FPS, JPEG quality 55, 10-30 KB/frame, 1.2-3.6 Mbps, 100-200ms latency

How It Works:

1. OpenCV captures webcam video
2. Frames resized to 640×480, compressed to JPEG
3. Sent via UDP at 15 FPS
4. Server relays to room participants
5. Clients display in multi-tile grid

Grid Layouts: 1×1 (1 user), 2×1 (2 users), 2×2 (3-4 users), 3×2 (5-6 users), 3×3 (7-9 users)

Features: Dynamic grid, participant identification, local indicator "(You)", automatic layout, room filtering

Usage: Click video icon to start/stop, tiles auto-appear, scroll for more participants

3. Text Chat

Specifications: TCP, JSON format, <50ms latency

How It Works:

1. User types message, selects recipient
2. Client sends MSG_CHAT via TCP
3. Server routes (broadcast to all or unicast to user)
4. Server sends delivery confirmation
5. Recipients display in chat panel

Features: Broadcast/private messaging, case-insensitive matching, delivery confirmation, message history, timestamps, auto-scroll

Usage:

- Broadcast: Select "All", type, press Enter
- Private: Select username, type, press Enter

4. File Transfer

Specifications: TCP, 32KB chunks, 100MB max, MD5 verification

How It Works:

- Upload: Select file → Calculate MD5 → Send metadata → Send chunks → Server validates → Notify recipients
- Download: Click download → Request file → Receive metadata → Receive chunks → Validate MD5 → Save

Features: Targeted sharing, progress tracking, integrity verification, notifications, path traversal protection

Usage:

- Upload: Click "Upload File", select file, choose recipient
- Download: Click download button in notification

5. Screen Sharing

Specifications: TCP, mss capture, JPEG quality 60, 10 FPS, 200-300ms latency

How It Works:

- Presenter: Capture screen → Compress JPEG → Send via TCP → Local preview
- Viewer: Receive frames → Display in screen share area

Features: Full screen capture, local preview, stop control, multi-viewer support

Usage:

- Start: Click screen share button
- Stop: Click button again or close presenter window

6. LAN Discovery

Specifications: UDP broadcast (port 5001), 5s interval, 15s TTL

How It Works:

1. Server broadcasts presence every 5s
2. Client listens for broadcasts
3. Displays discovered servers in login dialog
4. Removes servers not seen for 15s

Usage: Launch client, wait for servers in "Discovered Servers" list, click to auto-fill IP

7. Meeting Scheduler

Specifications: JSON storage, ISO 8601 datetime

Features: Schedule meetings, view list, auto-fill login, edit/delete, persistent storage

Usage:

- Schedule: Click "🗓️ Scheduler", enter ID/title/time, click Save
- Join: Click meeting in list, details auto-fill

8. Multi-Room Support

Specifications: Meeting ID-based isolation, unlimited rooms, dynamic creation

How It Works:

1. Client specifies meeting ID during registration
2. Server assigns to room
3. All communication filtered by room

Features: Complete isolation, unlimited rooms, automatic cleanup

Usage: Enter unique meeting ID in login (blank = "default" room)

9. User Management

Features: Real-time participant list, connection status, username display, IP tracking

Usage: View participant list in chat panel (auto-updates)

10. Connection Management

Specifications: 3s heartbeat interval, 15s idle timeout

Features: Heartbeat monitoring, automatic cleanup, graceful disconnect, error recovery

Usage: Automatic - no user action required

Installation & Setup

Prerequisites

Python 3.10+python --version

System Dependencies:

- Windows: Visual Studio Build Tools (for PyAudio)
- macOS: `brew install portaudio`
- Linux: `sudo apt-get install portaudio19-dev`

Installation Steps

1. Clone Repositorygit clone <repository-url>

cd sapora

2. Install Python Dependenciespip install -r requirements.txt

3. Verify Installationpython -c "import cv2, pyaudio, PyQt6; print('All dependencies OK')"

Running the Application

Start Server:cd server

python server_main.py

Expected output:==

Starting Sapora Server - LAN Collaboration Suite

==

Starting TCP Control Server...

Starting UDP Audio Server...

Starting UDP Video Server...

Starting File Transfer Server...

Starting Screen Share Server...

All Sapora Server services started successfully!

Start Client:cd client

python main_ui.py

Find Server IP:

- Windows: `ipconfig` → IPv4 Address
- macOS/Linux: `ifconfig` or `ip addr`

User Guide

Joining a Meeting

1. Launch client application
2. Enter server IP (or select from discovered servers)
3. Enter your name
4. Enter meeting ID (or leave blank for "default")
5. Click "Join Meeting"

Meeting Controls

Bottom Control Bar:

- 🎤 **Microphone:** Toggle mute/unmute
- 📹 **Video:** Start/stop video

- 🖥️ **Screen Share:** Start/stop screen sharing
- 💬 **Chat:** Open/close chat panel
- 📁 **Files:** Open file transfer dialog
- 📞 **End Call:** Leave meeting

Chat Features

Send Message:

1. Type message in input field
2. Select recipient (All or username)
3. Press Enter or click Send

Private Messaging:

- Select username from dropdown
- Case-insensitive matching

File Sharing:

1. Click "Upload File"
2. Select file and target
3. Recipients see notification with download button

Screen Sharing

Start: Click screen share button → Your screen broadcasts → Local preview shown
Stop: Click button again or close presenter window
Viewing: Shared screen appears in screen share area

File Transfer

Upload: Click "Upload File" → Select file → Choose recipient → Wait for confirmation
Download: Click download in notification → Choose save location → Wait for completion

---

Network Specifications

Bandwidth Requirements

Per Participant:

- Audio Upload: ~705 kbps
- Audio Download: ~705 kbps

- Video Upload: 1.2-3.6 Mbps
- Video Download: 1.2-3.6 Mbps × (N-1) participants

Example (4 participants, all video on):

- Upload: ~4.3 Mbps
- Download: ~11 Mbps

Recommended: 100 Mbps LAN for 8-10 participants, wired connections preferred

Latency Characteristics

- Audio: 50-100ms
- Video: 100-200ms
- Chat: <50ms
- Screen Share: 200-300ms

Firewall Configuration

Server: Allow incoming on ports 5000-5003, 6000-6001
Client: Allow outgoing to server IP on above ports

Windows Firewall:netsh advfirewall firewall add rule name="Sapora Server" dir=in action=allow program="C:\path\to\python.exe" enable=yes

---

API Reference

Server API

ConnectionManager Methods:

- `add_client(socket, address)` - Register TCP client
- `remove_client(socket)` - Remove client
- `register_stream(type, address)` - Register UDP endpoint
- `get_room_by_ip(ip)` - Get room for IP
- `get_audio_listeners()` - Get audio endpoints
- `get_video_listeners(room)` - Get video endpoints for room

Message Packing:pack_message(msg_type, payload) -> bytes

unpack_message(data) -> (version, msg_type, length, seq, payload)

Client API

VideoClient: VideoClient(server_ip, server_port, username, frame_callback)

start_streaming() # Start capture and send

stop_streaming()   # Stop capture

start_receiving() # Start receive thread

AudioClient: AudioClient(server_ip, username)

start_streaming() # Start capture and send

stop_streaming()   # Stop capture

start_receiving() # Start playback

ChatClient: ChatClient(server_ip, server_port, username, meeting_id)

set_callbacks(user_list_cb, message_cb)

send_message(text, target="all")

connect()

disconnect()

FileClient: FileTransferClient(server_ip, status_callback)

upload_file(filepath, target="all") -> bool

download_file(filename, save_path) -> bool

Configuration

Network Settings (constants.py)

CONTROL_PORT = 5000

FILE_TRANSFER_PORT = 5002

SCREEN_SHARE_PORT = 5003

VIDEO_PORT = 6000

AUDIO_PORT = 6001

Media Settings

VIDEO_WIDTH = 640

VIDEO_HEIGHT = 480

VIDEO_FPS = 15

VIDEO_QUALITY = 55 # JPEG quality (0-100)

AUDIO_RATE = 44100

AUDIO_CHANNELS = 1

AUDIO_CHUNK = 1024

Performance Tuning

UDP_STREAM_BUFFER = 65536

FILE_CHUNK_SIZE = 32768

MAX_FILE_SIZE = 104857600 # 100 MB

CONNECTION_TIMEOUT = 5.0

HEARTBEAT_INTERVAL = 3.0

CLIENT_IDLE_TIMEOUT = 15.0

Project Structure

sapora/

├── server/

│   ├── server_main.py   # Main orchestrator

│   ├── connection_manager.py   # Client state management

│   ├── tcp_handler.py   # Control server

```
│   ├── udp_audio_server.py     # Audio mixing
│   ├── udp_video_server.py     # Video relay
│   ├── file_server.py          # File transfers
│   ├── screen_share_server.py  # Screen sharing
│   └── utils.py                # Helper functions
├── client/
│   ├── main_ui.py              # PyQt6 GUI
│   ├── audio_client.py         # Audio capture/playback
│   ├── video_client.py         # Video capture/display
│   ├── chat_client.py          # Chat communication
│   ├── file_client.py          # File transfers
│   ├── screen_share_client.py  # Screen capture/view
│   └── utils.py                # Helper functions
├── shared/
│   ├── constants.py            # Configuration
│   ├── protocol.py             # Message types
│   ├── helpers.py              # Serialization
│   └── lan_discovery.py        # Server discovery
├── requirements.txt            # Python dependencies
└── README.md                   # Quick start guide
```
