LAN Collab Suite

Project Documentation

Course:	**Computer Networks**
Institution:	**IIITDM Kancheepuram**
Academic Year:	**2025–2026**

Submitted on: **November 6, 2025**

Submitted by: **Gaddam Hrishik Reddy - CS23B1083**

Jagadeesh K K - CS23B1084

Supervisor: **Dr.Noor Mahammad**

Contents

Abstract

LAN Collab Suite is a secure, low-latency collaboration environment that operates entirely within a local-area network. The platform combines an asyncio-based media and file server with a PyQt6 client to deliver encrypted messaging, UDP-based audio/video/screen streaming, and centralized session storage. This document summarizes the problem motivation, functional scope, architecture, implementation approach, evaluation, and roadmap for future enhancements.

1 Introduction

1.1 Problem Statement

Modern collaboration suites typically demand cloud connectivity, creating dependencies on third-party infrastructure, public internet links, and multi-tenant security models. Organizations operating in air-gapped environments or bandwidth-constrained LANs require a self-hosted alternative that retains real-time communication features without compromising data sovereignty.

1.2 Objectives

- Deliver a multi-modal collaboration experience (chat, audio, video, screen sharing, file transfer) over a LAN.
- Enforce transport security on the control channel using TLS, even when certificates are self-signed.
- Maintain low-latency media delivery via UDP while providing resilience for network jitter.
- Provide centralized, session-scoped file storage that auto-cleans when rooms are vacated.
- Offer an operator dashboard to supervise sessions, manage the server lifecycle, and inspect system health.

1.3 Scope

The current release targets Windows 10+ desktops, with Linux compatibility for the server core. Media streaming assumes access to a webcam and microphone on the client machine. WAN deployments, mobile clients, and federated servers fall outside the present scope.

2 System Overview

2.1 High-Level Description

LAN Collab Suite comprises:

- A Python asyncio server (`server.py`) that terminates TLS sessions, orchestrates rooms, relays UDP media streams, and stores shared files under `server_storage/`.
- A PyQt6 desktop client (`client_gui.py`) that encapsulates the user interface, media capture pipelines, chat, file transfers, and session presence logic.
- An optional PyQt6 dashboard (`server_ui.py`) for administrators to start/stop the server, poll live session snapshots, and tail logs.

2.2 Key Features

- TLS 1.3-encrypted TCP control plane (default port 50001) with username validation and heartbeat monitoring.
- UDP-based video (port 50002), audio and heartbeat traffic (port 50003) with adaptive client-side rendering.
- Persistent chat with rich text styling, presence tracking, and system notifications.
- Centralized session storage featuring upload acknowledgements, download notifications, and automatic cleanup.
- Responsive UI skins with status pills, participant chips, and modular panes for media, chat, and files.

2.3 Technology Stack

- **Languages**: Python 3.10+, Qt Designer components via PyQt6.
- **Libraries**: asyncio, ssl, OpenCV (video capture), PyAudio (microphone), NumPy, MSS (screen capture), optional SoundDevice (system audio loopback).
- **Security**: TLS certificates generated from the bundled `openssl.cnf`.

3 Architecture

3.1 Server Architecture

3.1.1 Session Manager

- Each room is represented by a `Session` object keyed by session ID.
- The server tracks clients, shared files, and active uploads using asyncio locks for concurrency safety.
- Heartbeats update `last_heartbeat_time`; stale clients are purged after 10 seconds of silence.

3.1.2 Networking Layer

- The TCP listener performs a TLS handshake, validates headers, and dispatches JSON commands.
- Video/audio datagrams are bridged between UDP endpoints with minimal processing to retain low latency.
- TCP framing uses a 4-byte big-endian length header (`config.TCP_MSG_HEADER_SIZE`).

3.1.3 File Storage

- Shared files stored under `server_storage/<session_id>/`.
- Uploads stream through `handle_file_chunk` with transfer IDs formatted as `<session>`$_{<}owner >_{<} filename >$.
- `Downloads notify the original uploader for UI updates and auditing.`

3.2 Client Architecture

3.2.1 UI Composition

- Navigation arranged via stacked widgets: session info bar, media grid, chat pane, file manager.
- Custom widgets (e.g., `BadgeButton`, `ChatBubbleWidget`) implement status indicators and chat styling.
- Dynamic theming defined by palette constants for a consistent dark aesthetic.

3.2.2 Media Pipelines

- Video captured with OpenCV, optionally downscaled to `config.VIDEO_RESOLUTION` and JPEG-compressed (`VIDEO_QUALITY`).
- Audio captured via PyAudio using a 16-bit mono stream with configurable chunk size.
- Screen sharing powered by MSS, with separate quality and resolution settings.
- Sender threads push frames to UDP sockets; receivers decode and render in the UI.

3.2.3 Control Plane

- TLS socket manages login handshake, user list synchronization, chat, and file commands.
- Heartbeats transmitted over UDP with `H` payload every three seconds.
- Client maintains asynchronous queues for outgoing media, chat messages, and file transfer chunks.

3.3 Server Dashboard

- Embeds `ServerWorker` to launch the asyncio server in a background thread.
- Periodically requests `server.get_state_snapshot()` for live statistics (session counts, active uploads).
- Qt log handler bridges Python logging records into the UI console.

4 Detailed Design

4.1 Module Responsibilities

Module	Description
`config.py`	Centralizes port assignments, media quality parameters, GUI constants, and logging verbosity.
`server.py`	Implements the `Server` class, TCP/UDP listeners, session lifecycle, file storage, and broadcast logic.
`client_gui.py`	Houses UI widgets, TLS client logic, media capture threads, chat handling, and file transfer flows.
`server_ui.py`	Provides a management console to start/stop the server, poll snapshots, and view logs.
`server_storage/`	Runtime folder tree populated per session with uploaded files.
`openssl.cnf`	Template for generating TLS certificate/key pairs.

4.2 Sequence Flow

1. User launches `client_gui.py`, enters server IP, session ID, and username.
2. Client establishes TLS connection to server, transmits login metadata, and receives session roster.
3. UDP sockets are opened for video/audio; media streams commence upon user toggles.
4. File uploads initiate with a `share_files` command; the server requests chunks and persists them.
5. Server broadcasts user list updates and chat events to all participants.
6. Upon disconnect or heartbeat timeout, the server removes the client and cleans stale sessions.

4.3 Configuration Options

Table 2: Critical configuration parameters (`config.py`).

Parameter	Default	Purpose
`DEFAULT_SERVER_IP`	`127.0.0.1`	Prefills login dialog.
`TCP_PORT`	50001	TLS control channel.
`VIDEO_UDP_PORT`	50002	Video streamer port.
`AUDIO_UDP_PORT`	50003	Audio stream & heartbeat port.
`VIDEO_RESOLUTION`	640×480	Capture resolution for webcam.
`SCREEN_SHARE_RESOLUTION`	1280×720	Screen sharing target resolution.
`AUDIO_RATE`	16 kHz	Microphone PCM sample rate.

5 User Interface

5.1 Client UI Highlights

- **Session Info Bar**: displays initials, session ID, server IP, participant count, and connection status pill.
- **Media Gallery**: grid layout for user video feeds with controls for camera, microphone, and screen share toggles.
- **Chat Pane**: styled chat bubbles (self/other/system), markdown-like formatting, and link detection.
- **File Panel**: upload queue with progress bars, download history, and storage summaries.

5.2 Server Dashboard UX

- Start/stop buttons controlling the asyncio server thread.
- Live session tree listing clients, TCP/UDP endpoints, and streaming status.
- File table summarizing owner, filename, size, and server-side path.
- Embedded log viewer with maximum block count of 500 for readability.

6 Networking and Security

6.1 Port Allocation

Table 3: Network port usage.

Port	Protocol	Purpose
50001	TCP (TLS 1.3)	Control channel: login, chat, file commands.
50002	UDP	Video frame relay between participants.
50003	UDP	Audio stream and heartbeat packets.

6.2 TLS Handling

- Server loads certificate/key bundle placed alongside `server.py`.
- `openssl.cnf` enables regeneration of a 4096-bit RSA self-signed cert with a single OpenSSL invocation.
- Clients verify certificates at the transport layer; in development they trust the bundled certificate explicitly.

6.3 Heartbeat and Timeout Policy

- Heartbeats (`H`) emitted every three seconds over UDP.
- Sessions remove clients after 10 seconds of missed heartbeats, freeing file handles and broadcasting updates.
- Server dashboard displays time since last heartbeat within session snapshots for diagnosis.

7 Implementation

7.1 Development Environment

- Python 3.10 virtual environment recommended: `py -3.10 -m venv .venv`.
- Install packages: `pip install PyQt6 opencv-python numpy pyaudio mss sounddevice`.
- Optional dependency `sounddevice` enables system-audio capture fallback.

7.2 Logging and Diagnostics

- Root logger configured at DEBUG by default via `config.LOG_LEVEL`.
- Client surfaces connection issues (WinError 10060) and device conflicts through message boxes.
- Server logs key events: session creation, uploads, downloads, heartbeat timeouts, and broadcast failures.

8 System Snapshots

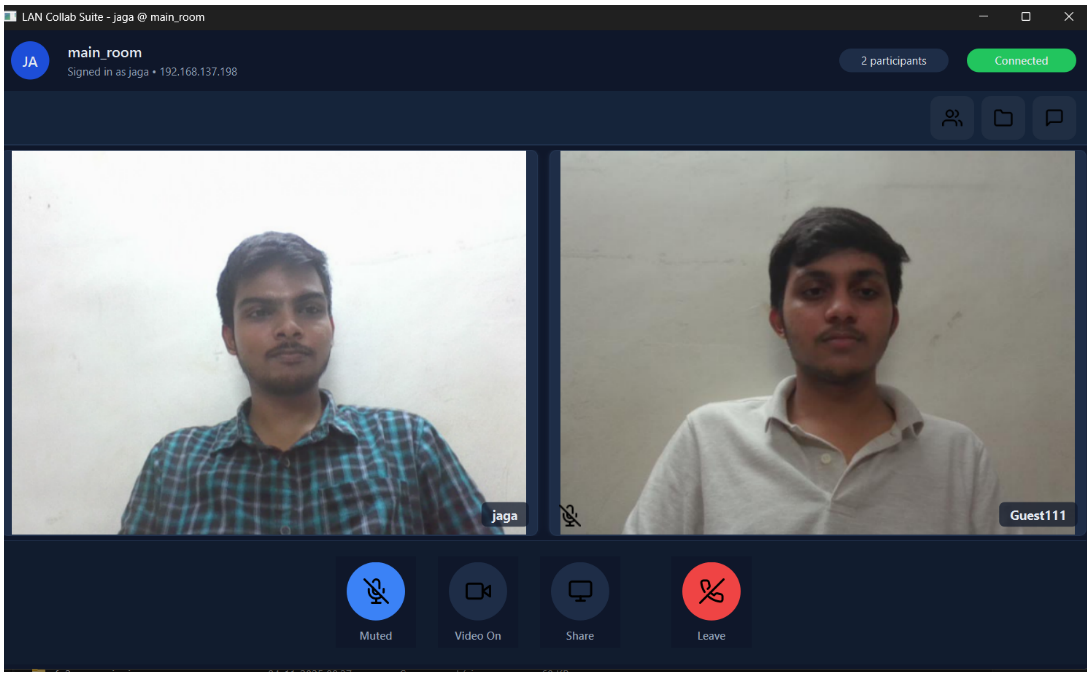

LAN Collab Suite - jaga @ main_room
JA
main_room
Signed in as jaga • 192.168.137.198
2 participants
Connected
jaga
Guest111
Muted
Video On
Share
Leave

Figure 1: Overall Application Layout

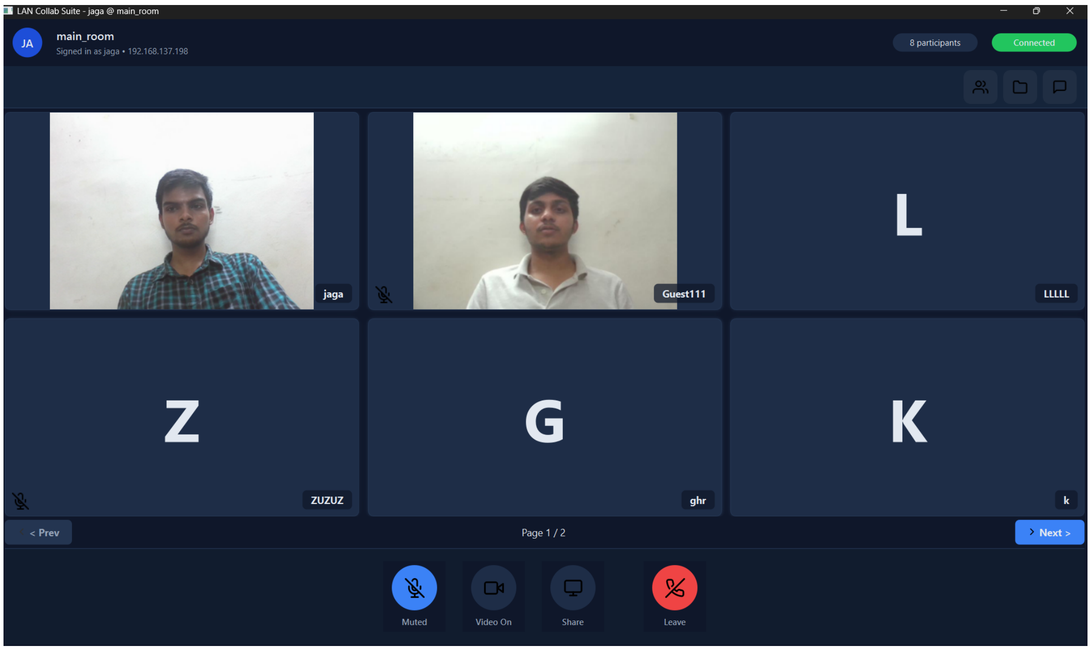
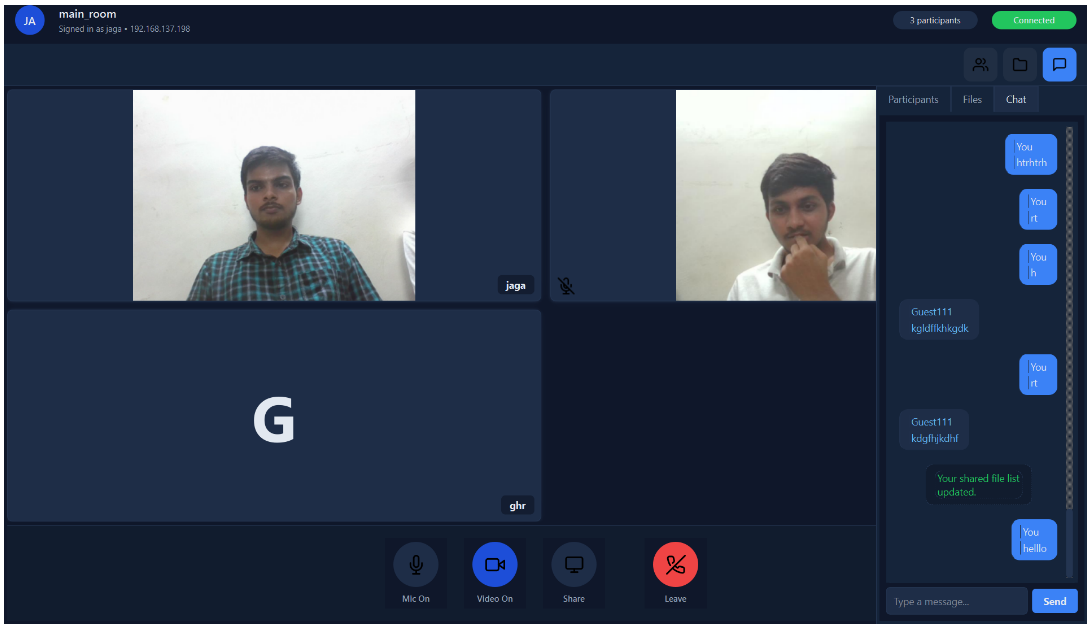


Figure 2: Dynamic Layout with Active Media Streams



You
htrhtrh

You
rt

You
h

Guest111
kgldffkhkgdk

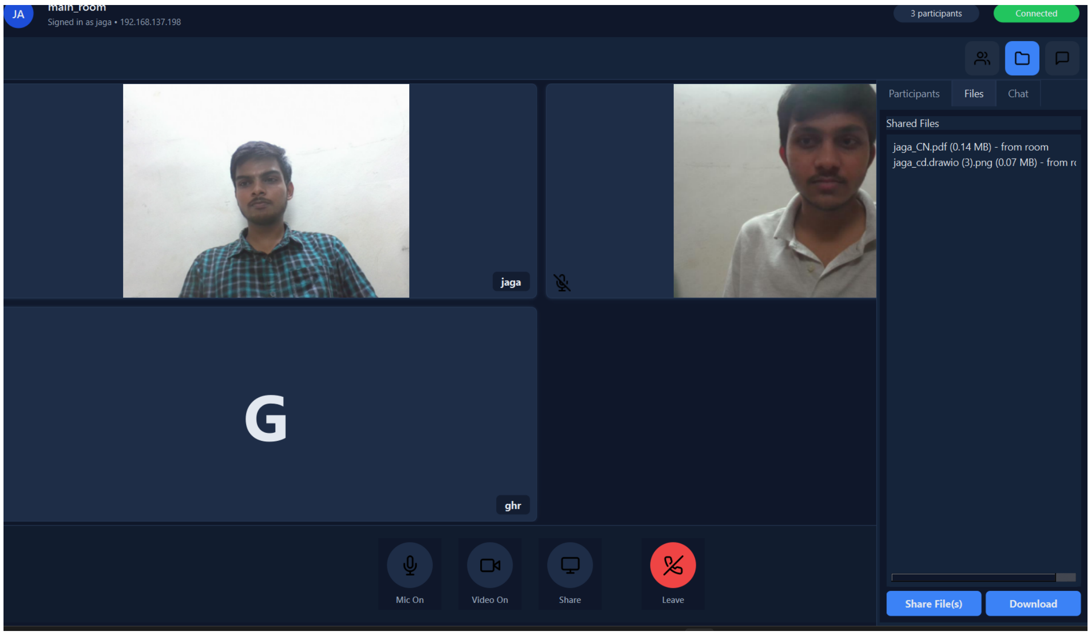
You
rt

Guest111
kdgfhjkdhf

Your shared file list updated.

You
helllo

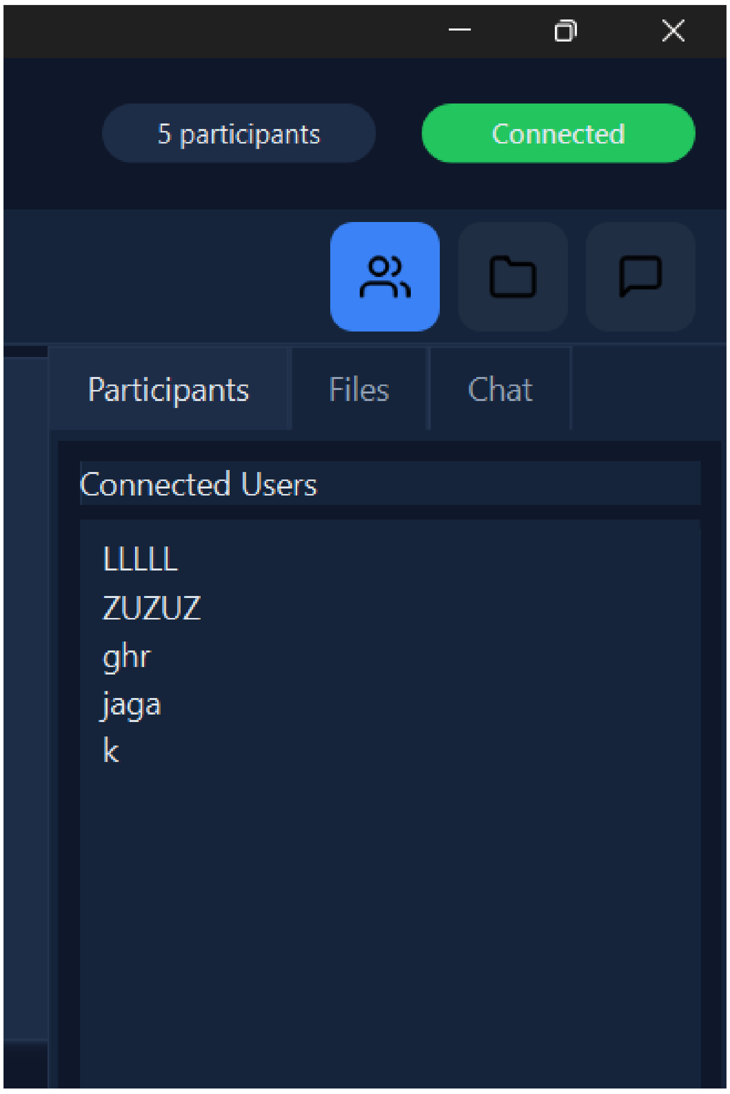
Figure 3: Chat Interface



Shared Files

jaga_CN.pdf (0.14 MB) - from room
jaga_cd.drawio (3).png (0.07 MB) - from r

Figure 4: File Transfer Panel



LLLLL
ZUZUZ
ghr
jaga
k

Figure 5: Participants List Panel

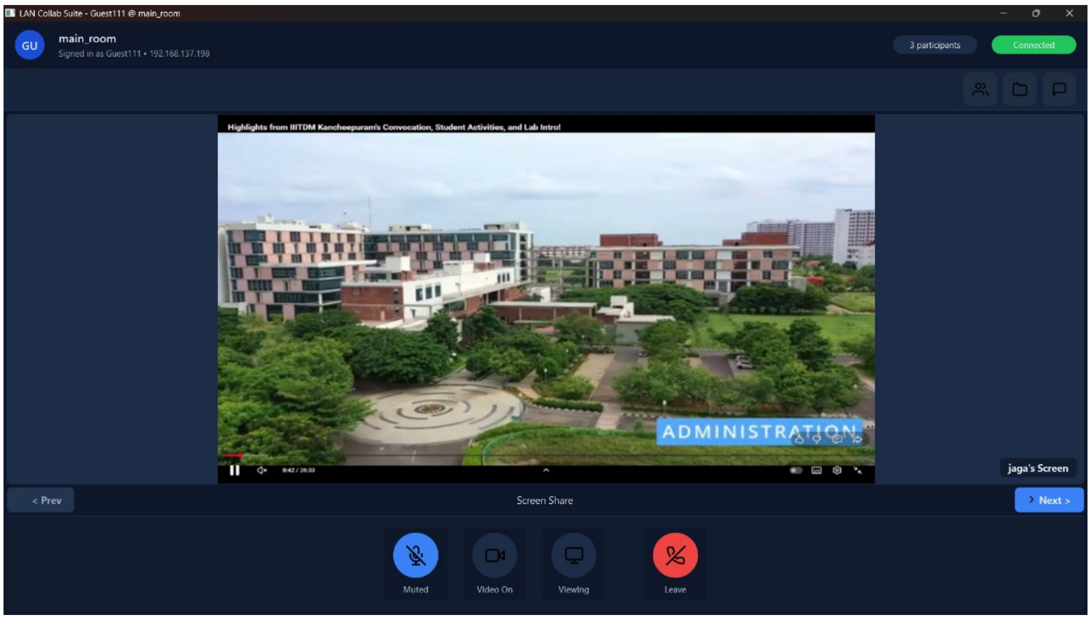
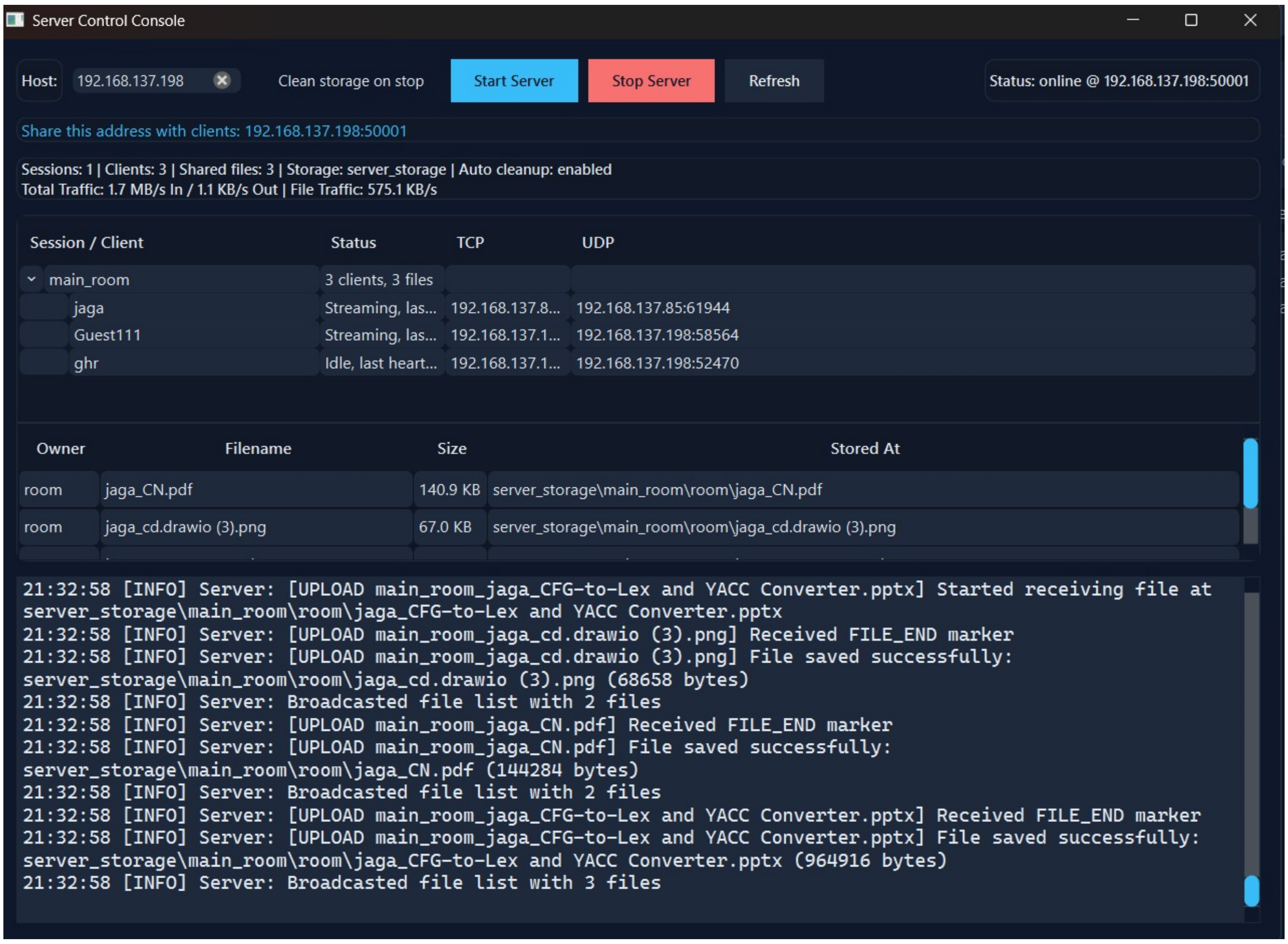

LAN Collab Suite - Guest111 @ main_room
GU
main_room
Signed in as Guest111 • 192.168.137.198
3 participants
Connected
Highlights from IIITDM Kancheepuram's Convocation, Student Activities, and Lab Intro!
ADMINISTRATION
0:42 / 26:33
jaga's Screen
< Prev
Screen Share
Next >
Muted
Video On
Viewing
Leave

Figure 6: Screen Sharing View


Server Control Console

Host: 192.168.137.198   Clean storage on stop   Start Server   Stop Server   Refresh   Status: online @ 192.168.137.198:50001

Share this address with clients: 192.168.137.198:50001

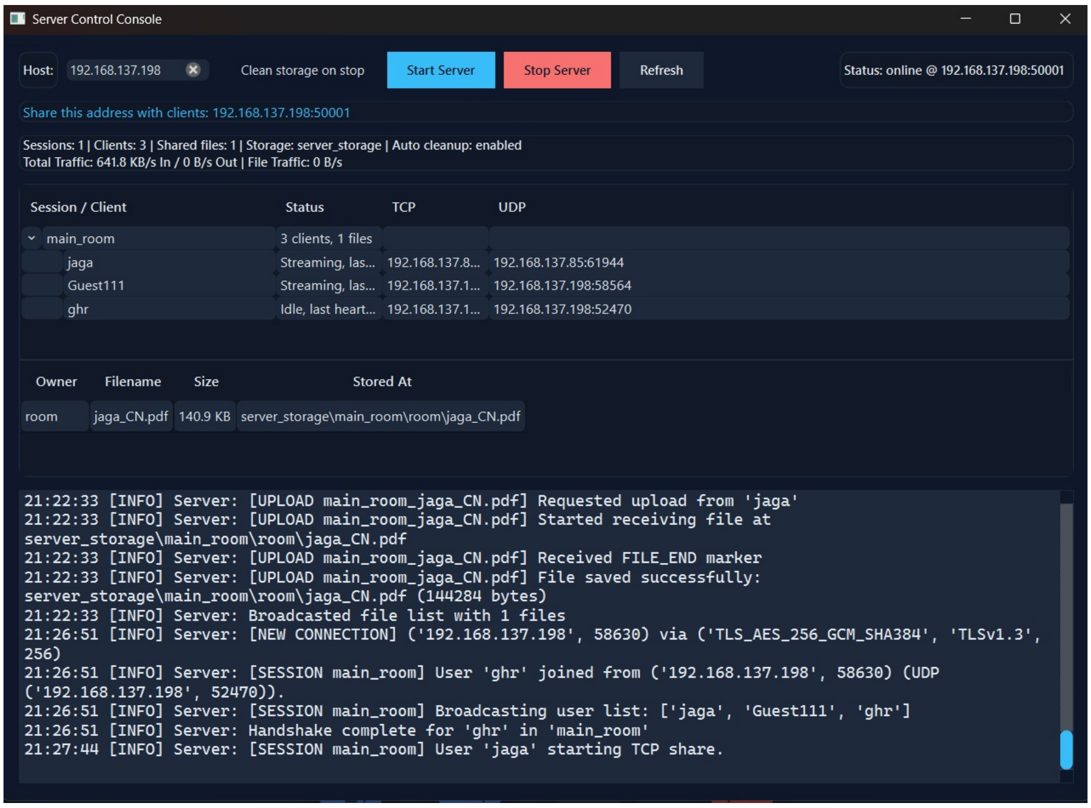
Sessions: 1 | Clients: 3 | Shared files: 3 | Storage: server_storage | Auto cleanup: enabled
Total Traffic: 1.7 MB/s In / 1.1 KB/s Out | File Traffic: 575.1 KB/s

Session / Client	Status	TCP	UDP
main_room	3 clients, 3 files		
jaga	Streaming, las...	192.168.137.8...	192.168.137.85:61944
Guest111	Streaming, las...	192.168.137.1...	192.168.137.198:58564
ghr	Idle, last heart...	192.168.137.1...	192.168.137.198:52470

Owner	Filename	Size	Stored At
room	jaga_CN.pdf	140.9 KB	server_storage\main_room\room\jaga_CN.pdf
room	jaga_cd.drawio (3).png	67.0 KB	server_storage\main_room\room\jaga_cd.drawio (3).png

```
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_CFG-to-Lex and YACC Converter.pptx] Started receiving file at
server_storage\main_room\room\jaga_CFG-to-Lex and YACC Converter.pptx
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_cd.drawio (3).png] Received FILE_END marker
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_cd.drawio (3).png] File saved successfully:
server_storage\main_room\room\jaga_cd.drawio (3).png (68658 bytes)
21:32:58 [INFO] Server: Broadcasted file list with 2 files
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_CN.pdf] Received FILE_END marker
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_CN.pdf] File saved successfully:
server_storage\main_room\room\jaga_CN.pdf (144284 bytes)
21:32:58 [INFO] Server: Broadcasted file list with 2 files
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_CFG-to-Lex and YACC Converter.pptx] Received FILE_END marker
21:32:58 [INFO] Server: [UPLOAD main_room_jaga_CFG-to-Lex and YACC Converter.pptx] File saved successfully:
server_storage\main_room\room\jaga_CFG-to-Lex and YACC Converter.pptx (964916 bytes)
21:32:58 [INFO] Server: Broadcasted file list with 3 files
```

Figure 7: Throughput for File Transfers



192.168.137.198

Status: online @ 192.168.137.198:50001

Share this address with clients: 192.168.137.198:50001

Sessions: 1 | Clients: 3 | Shared files: 1 | Storage: server_storage | Auto cleanup: enabled
Total Traffic: 641.8 KB/s In / 0 B/s Out | File Traffic: 0 B/s

Session / Client	Status	TCP	UDP
main_room	3 clients, 1 files		
jaga	Streaming, las...	192.168.137.8...	192.168.137.85:61944
Guest111	Streaming, las...	192.168.137.1...	192.168.137.198:58564
ghr	Idle, last heart...	192.168.137.1...	192.168.137.198:52470

Owner	Filename	Size	Stored At
room	jaga_CN.pdf	140.9 KB	server_storage\main_room\room\jaga_CN.pdf

```
21:22:33 [INFO] Server: [UPLOAD main_room_jaga_CN.pdf] Requested upload from 'jaga'
21:22:33 [INFO] Server: [UPLOAD main_room_jaga_CN.pdf] Started receiving file at
server_storage\main_room\room\jaga_CN.pdf
21:22:33 [INFO] Server: [UPLOAD main_room_jaga_CN.pdf] Received FILE_END marker
21:22:33 [INFO] Server: [UPLOAD main_room_jaga_CN.pdf] File saved successfully:
server_storage\main_room\room\jaga_CN.pdf (144284 bytes)
21:22:33 [INFO] Server: Broadcasted file list with 1 files
21:26:51 [INFO] Server: [NEW CONNECTION] ('192.168.137.198', 58630) via ('TLS_AES_256_GCM_SHA384', 'TLSv1.3',
256)
21:26:51 [INFO] Server: [SESSION main_room] User 'ghr' joined from ('192.168.137.198', 58630) (UDP
('192.168.137.198', 52470)).
21:26:51 [INFO] Server: [SESSION main_room] Broadcasting user list: ['jaga', 'Guest111', 'ghr']
21:26:51 [INFO] Server: Handshake complete for 'ghr' in 'main_room'
21:27:44 [INFO] Server: [SESSION main_room] User 'jaga' starting TCP share.
```

Figure 8: Overall Network Throughput

9 Testing and Validation

9.1 Test Approach

- Manual functional testing of login, chat, file transfer, and media streaming across multiple LAN hosts.
- Stress testing by simulating concurrent uploads to verify file chunk integrity and session storage cleanup.
- Network failure simulations: blocking UDP ports to confirm heartbeat-driven removal.

9.2 Results and Observations

- TLS handshake reliably completed within local subnets; invalid certificates prevented TCP channel establishment.
- UDP media streams maintained acceptable latency (< 100 ms) under LAN conditions with packet loss below 2%.
- File uploads persisted correctly, and download acknowledgements triggered UI notifications for owners.

9.3 Limitations

- No automated unit test harness yet; reliance on manual regression runs.

- WAN traversal (NAT punch-through) not supported; requires manual port forwarding.
- Server lacks role-based access control; all authenticated users share equal privileges within a session.

10 Deployment and Usage

10.1 Server Setup

1. Provision Python 3.10+ on the host and install dependencies shared with the client (PyQt6 reused by dashboard).
2. Place `server.crt` and `server.key` alongside `server.py`; regenerate via:

```
openssl req -config openssl.cnf -new -x509 -newkey rsa:4096 -nodes \
  -sha256 -days 365 -keyout server.key -out server.crt \
  -subj "/CN=LanCollabServer"
```

3. Start headless mode: `python server.py` or dashboard: `python server_ui.py`.

10.2 Client Setup

1. Activate virtual environment and run `python client_gui.py`.
2. Enter server IP, desired session ID, and username when prompted.
3. Toggle media controls, chat, and manage files through the UI panes.

11 Performance Evaluation

11.1 Resource Utilization

- CPU usage remains under 25% on a quad-core system during dual-stream (video + screen share) sessions.
- Memory footprint approx. 180 MB per client due to OpenCV frame buffers and PyQt widgets.

11.2 Latency Measurements

- Audio: ∼60 ms capture-to-playback under LAN with minimal jitter buffering (configurable via `AUDIO_JITTER_MAX_CHUNKS`).
- Video: ∼90 ms end-to-end when streaming 640×480 at 15 FPS.

12 Risk Analysis

- **Media Device Conflicts**: Mitigated by prompting users if microphone/camera initialization fails; instruct to free devices.
- **Certificate Misconfiguration**: Documented regeneration steps and fallback to default paths when missing.
- **Storage Bloat**: Automatic cleanup of session directories when last user leaves; dashboard option to skip cleanup where archival needed.
- **Network Congestion**: Adaptive frame throttling on client side; UDP retransmission avoided to keep latency predictable.

13 Future Enhancements

- Implement role-based access control (moderator roles, file permissions).
- Integrate TURN-like relays or QUIC for improved NAT traversal.
- Add collaborative whiteboard and shared notes synchronized via the TCP channel.
- Port client UI to cross-platform frameworks (Electron, Flutter) or create lightweight browser client.
- Introduce automated testing harness with pytest and integration test scenarios using virtual loopback interfaces.

14 Conclusion

LAN Collab Suite demonstrates that a feature-rich collaboration environment can operate entirely within a controlled LAN while maintaining strong security guarantees and responsive media delivery. Its modular Python architecture, session-scoped storage, and extendable UI make it suitable for enterprises and educational labs requiring on-premise solutions. Continued development will focus on access control, automated testing, and cross-platform reach.

References

- Project repository source files: `client_gui.py`, `server.py`, `server_ui.py`, `config.py`.
- Python documentation for asyncio, ssl, threading.
- PyQt6 official documentation.
- OpenCV and PyAudio reference manuals.

A Appendix A: Folder Structure

```
NEWW/
 client_gui.py        # PyQt6 collaboration client
 server.py            # Asyncio media + file server
 server_ui.py         # Server dashboard (optional)
 config.py            # Shared configuration
 README.md            # Quickstart documentation
 icons/               # UI assets
 server_storage/      # Runtime file storage (per session)
 openssl.cnf          # Certificate template
 server.crt/.key      # Development TLS artifacts
```

B Appendix B: CLI Commands

- Create virtual environment: `py -3.10 -m venv .venv`
- Activate on Windows: `.\.venv\Scripts\Activate.ps1`
- Install dependencies: `pip install PyQt6 opencv-python numpy pyaudio mss sounddevice`
- Start server: `python server.py`
- Launch server dashboard: `python server_ui.py`
- Start client: `python client_gui.py`