LAN MEET - INTEGRATED VIDEO CONFERENCING APPLICATION

CS23B2055 T. HARSHINI SRI SAVITHA

CS23B2048 K. MOKSHITHA

November 5, 2025

Contents

Chapter 1

Executive Summary

1.1 Project Overview

This project is a LAN-based, multi-user video conferencing system designed for seamless communication and collaboration over a local network. The system includes real-time audio, video, screen sharing, file transfer, and chat functionalities — all integrated under a single architecture. The project employs two main scripts:

- **unified_server.py:** Acts as the central hub managing all incoming and outgoing client connections.
- **main_app_updated.py:** The main client interface built with PyQt5 that handles UI and user interaction.

The overall architecture adopts a hybrid **client-server model**, efficiently combining both TCP and UDP protocols for different types of data transfer.

1.2 Core Features & Functionality

1.2.1 Real-Time Media Transmission

- **Webcam Streaming:** Users can enable or disable their webcams. The video feed is captured using OpenCV, compressed into JPEG format, and transmitted over UDP for real-time display.
- **Audio Streaming:** Microphone input is captured and streamed in real-time using PyAudio. Users can mute or unmute as needed.
- **Screen Sharing:** Enables users to broadcast their screen using the `mss` library. The captured frames replace webcam video, allowing live presentations.
- **Dynamic Presentation Mode:** The application automatically transitions from a grid layout to a presentation layout when a screen share begins. The sharer's feed becomes the main view, and other participants are shifted to a smaller side grid.

1.3 Technology Stack

- **Language**: Python 3.8+
- **GUI Framework**: PyQt5
- **Video Processing**: OpenCV (cv2)
- **Screen Capture**: mss library
- **Audio Processing**: PyAudio
- **Networking**: Python sockets (TCP/UDP)
- **Serialization**: Pickle, JSON, Base64

Chapter 2

System Architecture

2.1 Architecture Overview

LAN Meet employs a hybrid client-server architecture combining:

- **Centralized Coordination**: TCP server manages user authentication, chat, and file coordination
- **Distributed Media Streaming**: UDP broadcast servers relay video/audio streams
- **Multi-threaded Processing**: Separate threads for capture, transmission, and rendering

2.2 Component Architecture

2.2.1 Three-Tier System Design

The system is divided into three main layers:

1. **Presentation Layer (Client UI)**
    - PyQt5-based graphical interface
    - Video rendering and layout management
    - User interaction handling
    - Theme management system
2. **Application Layer (Network Manager)**
    - Connection management
    - Protocol handling (TCP/UDP)
    - Media capture and encoding
    - Multi-threaded I/O operations
3. **Infrastructure Layer (Servers)**
    - TCP chat/file server

- UDP video broadcast server
- UDP audio broadcast server
- Storage management

2.2.2 Module Breakdown

Client Application (client.py)

The main client application consists of:

- `VideoConferenceUI`: Main window and UI management
- Theme system with dual color palettes
- Video grid and presentation mode layouts
- Side panels (chat, participants, files)
- Control bar with feature toggles

Network Manager (networking_module.py)

Handles all network communication:

- `NetworkManager`: Core networking class
- Video sender/receiver threads
- Audio sender/receiver threads
- TCP chat listener thread
- File transfer management
- Screen capture integration

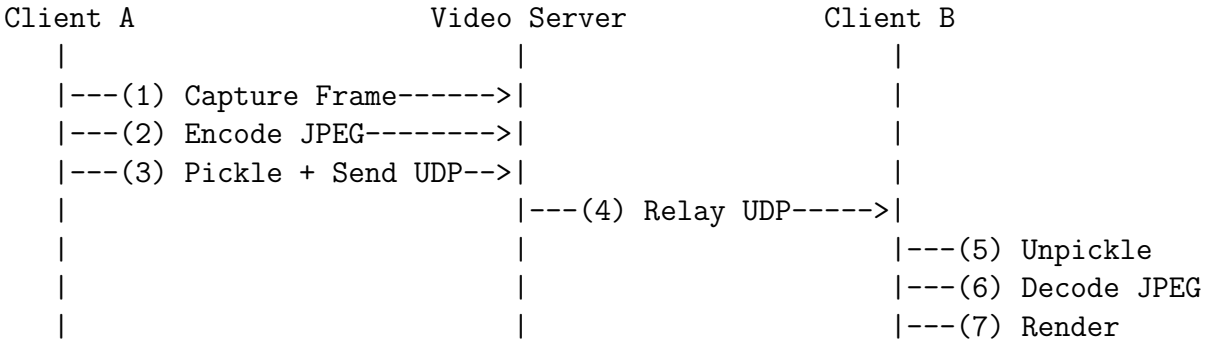
Unified Server (server.py)

Consolidated server implementation:

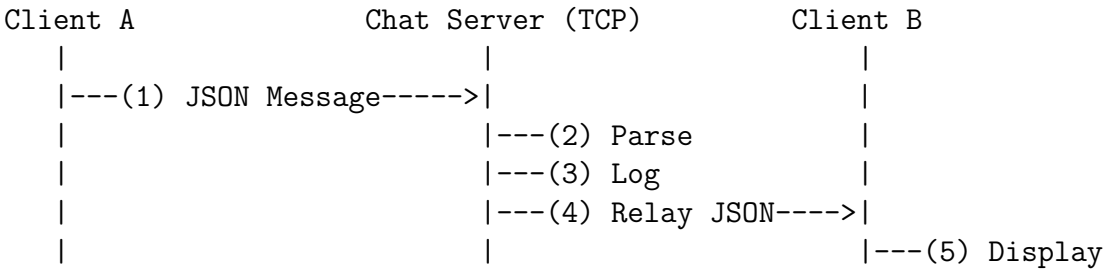
- `ChatServer`: TCP-based chat and file coordination
- `run_udp_broadcast_server`: Generic UDP relay function
- Separate threads for video/audio broadcasting
- File storage and retrieval system

2.3 Data Flow Diagrams

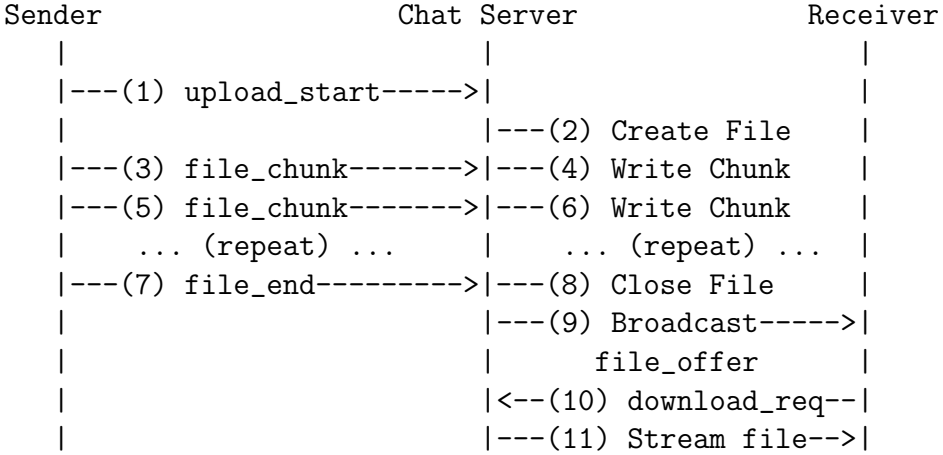
2.3.1 Video Streaming Flow



2.3.2 Chat Message Flow



2.3.3 File Transfer Flow



Chapter 3

Communication Protocols

3.1 Protocol Overview

LAN Meet uses three distinct communication channels for real-time collaboration:

Protocol	Port	Purpose	Transport
Chat/File	8004	Text, Files, Control	TCP
Video	5052	Video Frames	UDP
Audio	5051	Audio Samples	UDP

Table 3.1: Network Protocol Ports

3.1.1 Video Parameters

```
# Resolution
VIDEO_WIDTH = 640
VIDEO_HEIGHT = 360

# JPEG Compression Quality
JPEG_QUALITY = 75   # Webcam feed
JPEG_QUALITY = 85   # Screen share (higher quality)

# Network Parameters
UDP_BUFFER_SIZE = 65536  # Max UDP packet size
FRAME_RATE = 30          # Target ~30 FPS
```

3.1.2 Audio Parameters

```
# PyAudio Configuration
AUDIO_FORMAT = pyaudio.paInt16   # 16-bit PCM
AUDIO_CHANNELS = 1               # Mono
AUDIO_RATE = 44100               # 44.1 kHz sampling rate
AUDIO_CHUNK = 2048               # Samples per frame

# Network Parameters
UDP_BUFFER_SIZE = 65536
AUDIO_TIMEOUT = 200000  # Server timeout (seconds)
```

3.2 Client-Server Architecture

3.2.1 Client Initialization Sequence

1. Connect to TCP chat server (port 8004)
2. Authenticate with username
3. Create UDP video socket (port 5052)
4. Create UDP audio socket (port 5051)
5. Start background threads for:
    - TCP chat listener
    - UDP video sender (30 FPS)
    - UDP video receiver
    - UDP audio sender
    - UDP audio receiver
6. Request user list from server

3.2.2 Server Components

TCP Chat Server:

- Handles client authentication
- Relays chat messages (broadcast and private)
- Manages file transfers
- Maintains client registry
- Broadcasts user join/leave events

UDP Video Server:

- Receives video packets from clients
- Broadcasts packets to all other clients
- Tracks active clients with timeout mechanism
- No packet processing or transcoding

UDP Audio Server:

- Receives audio packets from clients
- Broadcasts packets to all other clients
- Tracks active clients with timeout mechanism
- No audio processing or mixing

Chapter 4

Implementation Details

4.1 Client Implementation

4.1.1 GUI Architecture

Main Window Structure

```
VideoConferenceUI (QMainWindow)

          central_widget (QWidget)
                root_layout (QVBoxLayout)
                    content_layout (QHBoxLayout)
                          video_area (QStackedWidget)
                                grid_view (QFrame)
                                      video_grid_layout (QGridLayout)
                                presentation_view (QWidget)
                                    presentation_label (main screen)
                                    participant_grid_widget
                          side_panel (QStackedWidget)
                              chat_widget
                              participants_widget
                              files_widget
                    control_bar (QFrame)
                    progress_bar (QProgressBar)

          pip_label (QLabel) [Floating self-view]
```

Theme System

The application supports dual themes with centralized color management:

```
THEMES = {
    'dark': {
        "root_bg": "#121212",
        "video_bg": "#181818",
        "panel_bg": "#1f1f1f",
        "text_primary": "#ffffff",
        "btn_accent": "#8ab4f8",
        # ... 20+ color definitions
    },
    'light': {
        "root_bg": "#ffffff",
```

```
        "video_bg": "#f0f0f0",
        # ... corresponding light colors
    }
}
```

Theme switching is instant and affects:

- Master stylesheet generation
- Dynamic button styles
- Chat history re-rendering
- Video placeholder regeneration

4.1.2 Video Layout Management

Grid View Mode

Default mode showing all participants in a grid:

- 3-column grid layout
- Each video: 320x240 pixels
- Automatic row calculation
- Self-view in floating PiP

Presentation Mode

Activated when any user shares screen:

- Main presentation area: Fills most of window
- Side participant strip: 240px wide, vertical layout
- Presenter's video shown large
- All others in small thumbnails

Layout Switching Logic

```
def rebuild_video_grid(self):
    # 1. Detect presenter
    for username, widget in self.video_widgets.items():
        if getattr(widget, 'frame_type') == 'screen':
            presenter_username = username
            break

    # 2. Detach all widgets
    for widget in self.video_widgets.values():
        widget.setParent(None)

    # 3. Build appropriate layout
    if presenter_username:
```

```
        # Presentation layout
        self.video_area.setCurrentWidget(
            self.presentation_view
        )
    else:
        # Grid layout
        self.video_area.setCurrentWidget(
            self.grid_view
        )
```

4.1.3 Network Manager Integration

Threading Model

```
Main Thread (PyQt GUI)

          network_thread (QThread)
              NetworkManager (QObject)
                  video_sender_thread
                  video_receiver_thread
                  audio_sender_thread
                  audio_receiver_thread
                  chat_listener_thread
                  file_sender_thread (dynamic)
```

Signal-Slot Communication

```
# Network Manager Signals
connected = pyqtSignal()
disconnected = pyqtSignal(str)
error = pyqtSignal(str)
local_frame_ready = pyqtSignal(object)
video_frame_received = pyqtSignal(str, object, str)
chat_message_received = pyqtSignal(str, str, bool)
user_list_updated = pyqtSignal(list)
file_offer_received = pyqtSignal(dict)
# ... etc

# Connected in GUI
self.network_manager.connected.connect(
    self.on_connected
)
self.network_manager.video_frame_received.connect(
    self.update_remote_feed
)
```

4.2 Server Implementation

4.2.1 Unified Server Architecture

Single server process manages three services:

```python
# Main thread
chat_server = ChatServer()
chat_server.start()  # Blocking TCP server

# Daemon threads
video_thread = Thread(
    target=run_udp_broadcast_server,
    args=(VIDEO_PORT, "Video", VIDEO_TIMEOUT)
)
audio_thread = Thread(
    target=run_udp_broadcast_server,
    args=(AUDIO_PORT, "Audio", AUDIO_TIMEOUT)
)
```

4.2.2 Chat Server Implementation

Client Management

```python
class ChatServer:
    def __init__(self):
        self.clients = {}   # {conn: username}
        self.usernames = {}  # {username: conn}
        self.receiving_files = {}  # Active uploads
        self.available_files = []  # File metadata
```

Message Handling Pipeline

```python
def handle_client(self, conn, addr):
    # 1. Authenticate
    username = self.authenticate_client(conn)

    # 2. Register client
    self.clients[conn] = username
    self.usernames[username] = conn

    # 3. Broadcast join
    self.broadcast_user_list()

    # 4. Message loop
    fileobj = conn.makefile("rb")
    while True:
        line = fileobj.readline()
        obj = json.loads(line)
        self.handle_json_message(obj, username, conn)
```

4.2.3 UDP Broadcast Server

Generic relay function for video/audio:

```python
def run_udp_broadcast_server(port, name, timeout):
    clients = {}  # {(ip, port): last_seen_time}
    socket = socket.socket(socket.AF_INET, SOCK_DGRAM)
    socket.bind((HOST, port))

```

```
    while True:
        data, addr = socket.recvfrom(UDP_BUFFER_SIZE)

        # Register/update client
        clients[addr] = time.time()

        # Broadcast to all except sender
        for client_addr in clients.keys():
            if client_addr != addr:
                socket.sendto(data, client_addr)

        # Remove timed-out clients
        remove_stale_clients(clients, timeout)
```

4.3 Screen Sharing Implementation

4.3.1 Capture Mechanism

Screen capture uses the `mss` library with Qt timer integration:

```
def set_screen_share_enabled(self, enabled):
    if enabled:
        # Start timer in main thread
        self.screen_capture_timer = QTimer()
        self.screen_capture_timer.timeout.connect(
            self._capture_screen
        )
        self.screen_capture_timer.start(33)  # 30 FPS
    else:
        self.screen_capture_timer.stop()

def _capture_screen(self):
    with mss.mss() as sct:
        monitor = sct.monitors[1]
        img_rgba = np.array(sct.grab(monitor))
        img_bgr = cv2.cvtColor(img_rgba, COLOR_BGRA2BGR)
        frame_resized = cv2.resize(
            img_bgr, (VIDEO_WIDTH, VIDEO_HEIGHT)
        )

        with self.screen_frame_lock:
            self.latest_screen_frame = frame_resized.copy()
```

4.3.2 Thread Safety

- Capture in main thread (Qt timer)
- Send from network thread
- Shared buffer protected by lock

Chapter 5

Setup and Installation

5.1 System Requirements

5.1.1 Hardware Requirements

- **Minimum**:
  - CPU: Dual-core 2.0 GHz
  - RAM: 2 GB
  - Network: 100 Mbps LAN
  - Webcam: 480p
  - Microphone and speakers
- **Recommended**:
  - CPU: Quad-core 2.5 GHz+
  - RAM: 4 GB+
  - Network: 1 Gbps LAN
  - Webcam: 720p or higher
  - Quality headset

5.1.2 Software Requirements

- Python 3.8 or higher
- Operating System: Windows 10/11, Linux, macOS
- Network: All devices on same LAN

5.2 Installation Steps

5.2.1 Python Dependencies

Install required packages:

```
pip install PyQt5
pip install opencv-python
pip install numpy
pip install pyaudio
pip install mss
```

Or use requirements file:

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

`requirements.txt`:

```
PyQt5>=5.15.0
opencv-python>=4.5.0
numpy>=1.19.0
pyaudio>=0.2.11
mss>=6.1.0
```

5.2.2 Network Configuration

Server Setup

1. Identify server IP address:

```
# Linux/Mac
ifconfig
# Windows
ipconfig
```

2. Edit server configuration in `server.py`:

```
HOST = '0.0.0.0'  # Listen on all interfaces
CHAT_PORT = 8006
VIDEO_PORT = 5052
AUDIO_PORT = 5053
```

3. Ensure firewall allows traffic on these ports

4. Start the server:

```
python server.py
```

Client Setup

1. Edit network configuration in `client.py`:

```
VIDEO_SERVER_IP = '192.168.1.100'  # Server IP
CHAT_SERVER_IP = '192.168.1.100'   # Server IP
VIDEO_SERVER_PORT = 5052
CHAT_SERVER_PORT = 8006
```

2. Launch the client:

```
python client.py
```

3. Enter username when prompted

Chapter 6

User Guide

6.1 Getting Started

6.1.1 Joining a Meeting

1. Launch the application
2. Enter your username in the dialog
3. Wait for connection confirmation
4. Your video feed appears in the bottom-left (PiP)
5. Other participants appear in the main grid

6.1.2 Main Interface Overview

Video Area

- Main grid shows all participants (3 columns)
- Self-view in floating Picture-in-Picture (bottom-left)
- Automatically switches to presentation mode during screen shares

Control Bar (Bottom)

- **Mute/Unmute**: Toggle microphone
- **Camera On/Off**: Toggle webcam
- **Share Screen**: Start/stop screen sharing
- **Chat**: Open chat panel
- **Participants**: View participant list
- **Files**: Browse shared files
- **Change Theme**: Switch light/dark mode
- **Leave Meeting**: Exit the call

6.2 Features Guide

6.2.1 Video and Audio

Camera Control

- Click "Camera On" to disable video
- When camera is off:
  - Others see a placeholder with your name
  - Saves bandwidth
  - Your self-view shows placeholder
- Click "Camera Off" to re-enable

Microphone Control

- Click "Mute" to mute microphone
- Button turns red when muted
- Click "Unmute" to speak again
- No audio is transmitted when muted

6.2.2 Screen Sharing

Starting Screen Share

1. Click "Share Screen" button
2. Camera automatically turns off
3. Your screen is captured and broadcast
4. Button shows " Stop Sharing" (red)
5. All participants enter presentation mode

Presentation Mode

When anyone shares their screen:

- Shared screen fills most of the window
- All participants (including yourself) appear in small thumbnails on the right
- Mode automatically ends when sharing stops

Stopping Screen Share

- Click " Stop Sharing" button
- Returns to normal grid view
- Camera remains off (toggle manually to re-enable)

6.2.3 Chat Features

Opening Chat

1. Click "Chat" button in control bar
2. Side panel slides in from right
3. Chat history is preserved
4. Close with X button in panel header

Sending Messages

- **Broadcast**: Type message and press Enter or click Send
- **Private**: Use format: `/w username message`
- Example: `/w Alice Hello, how are you?`

6.2.4 File Sharing

Uploading Files

1. Open chat panel
2. Click "Upload File" button
3. Select file from dialog
4. Progress bar shows upload status
5. File becomes available to all participants

Downloading Files

1. Click "Files" button
2. Panel shows all available files:
    - Select the required file.
    - Click the download button.
    - The selected file would be uploaded in the Downloads folder in the same directory.

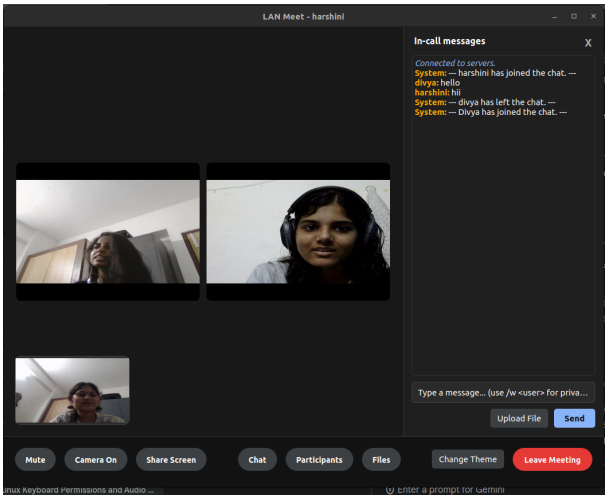


Figure 6.1: Enter Caption

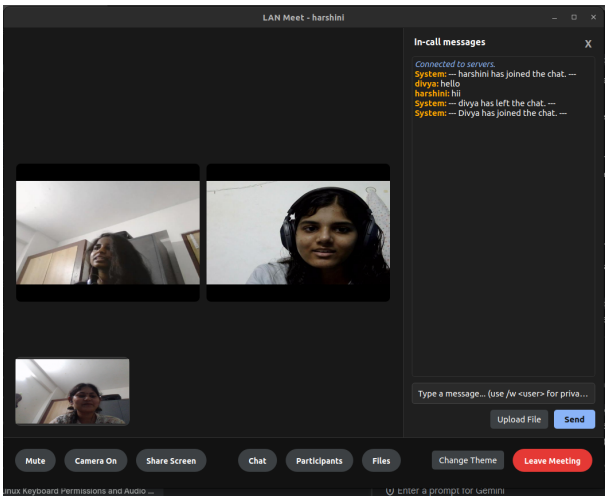


Figure 6.2: Enter Caption

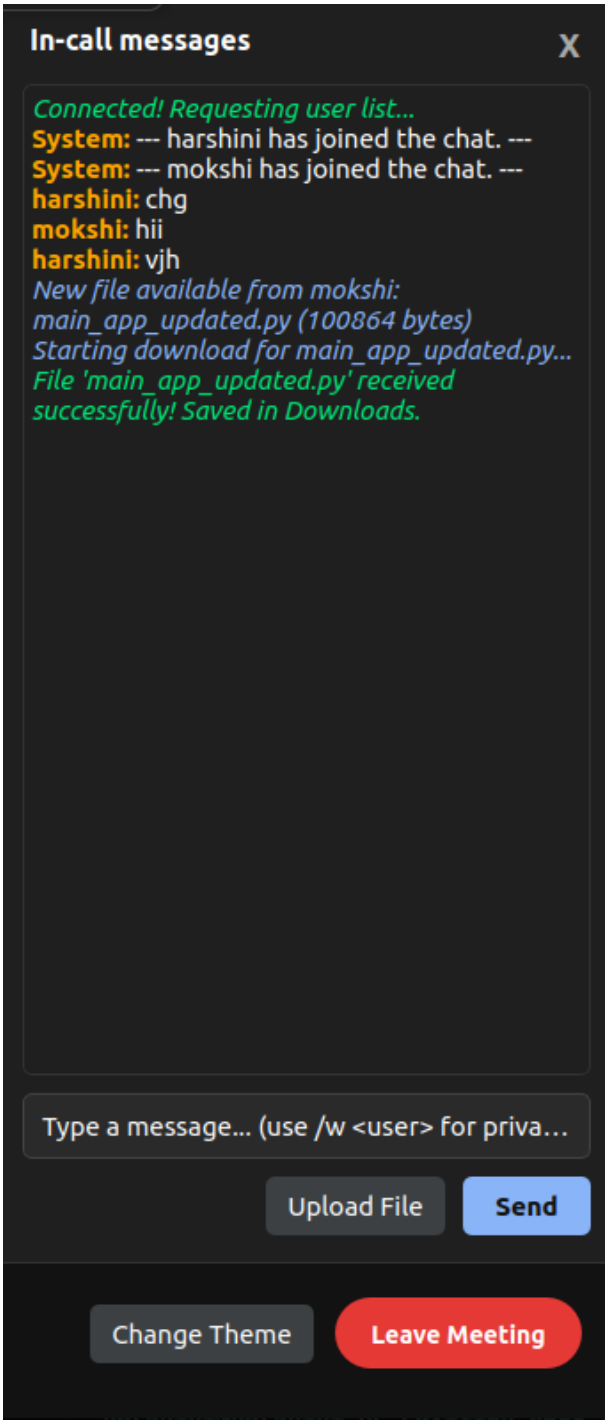


Figure 6.3: Enter Caption

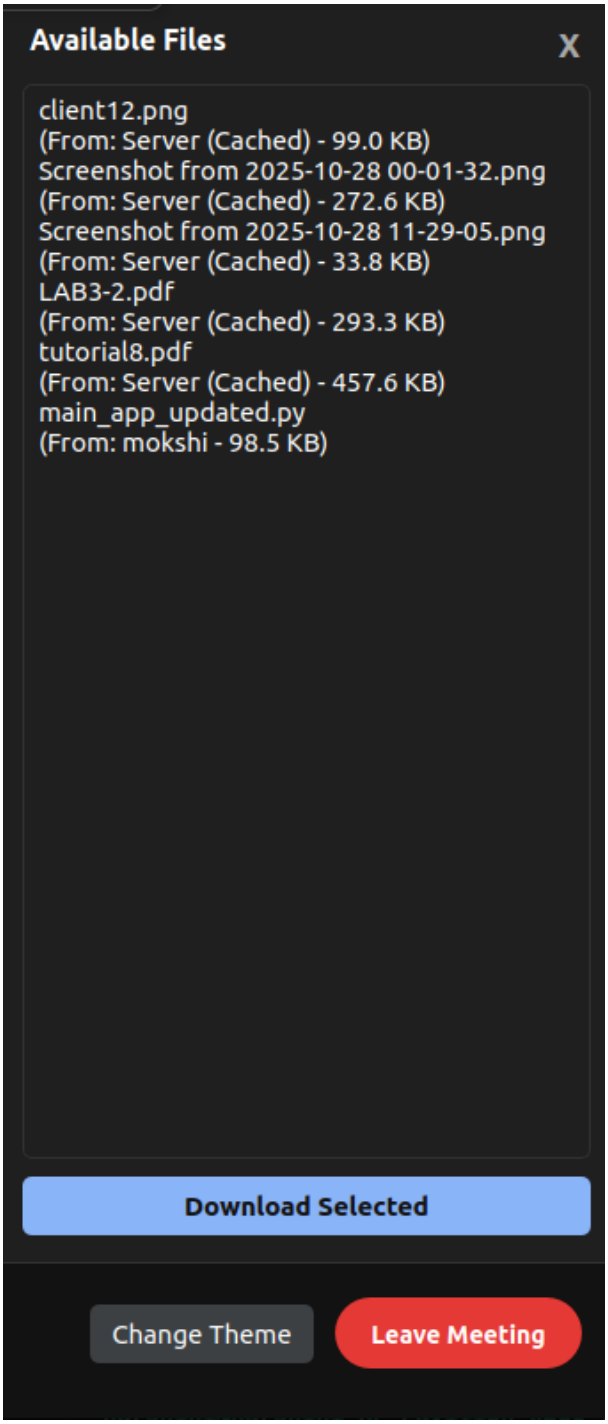


Figure 6.4: Enter Caption

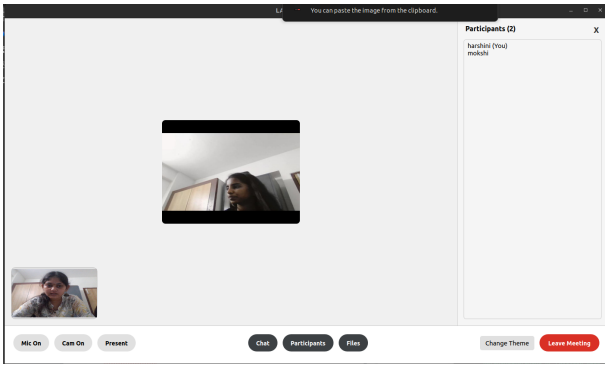


Figure 6.5: Enter Caption

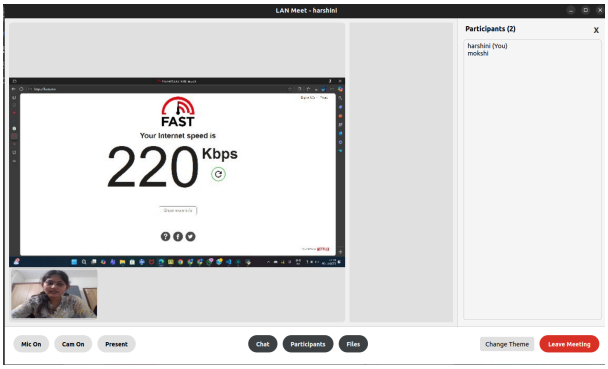


Figure 6.6: Enter Caption