Audio/Video Conferencing Prototype

A PROJECT REPORT

Submitted by

Raghava kethuri	*23BCS10773*
Harsha Vardhan	*23BCS10343*
Charan Teja	*23BCS12279*
Ankush	*23BCS14251*

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

November, 2025

Audio/Video Conferencing Prototype

BONAFIDE CERTIFICATE

Certified that this project report **"Audio/Video Conferencing Prototype"** is the Bonafide work of "**Kethuri Raghava, Harsha Vardhan, Charan Teja, Ankush"** who carried out the project work under my/our supervision.

Examiner

Suyash Gupta

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 <u>Introduction to the Project:</u>

In the era of digital communication, real-time collaboration tools have become essential for both personal and professional interactions. The Multi-Party Audio/Video Conferencing System aims to provide a seamless and interactive communication platform that enables users to connect through live audio and video streams directly in their web browsers. Leveraging WebRTC (Web Real-Time Communication) technology, the system facilitates peer-to-peer media exchange, ensuring low-latency and high-quality conferencing without requiring additional plugins or external software.

The project combines a Node.js signaling server using WebSockets for coordination and a React-based client for a modern, responsive user interface. The architecture allows participants to join virtual rooms, share audio, video, or screen content, and manage real-time controls such as mute, unmute, and camera toggle. To ensure reliable connectivity across diverse networks, STUN/TURN servers are integrated for NAT traversal, enabling smooth communication even behind firewalls.

Furthermore, the system incorporates MongoDB for storing minimal session metadata—such as room details, participant information, and connection statistics—supporting session persistence and analytics. Additional features like adaptive bandwidth management, auto-reconnect mechanisms, and secure access using JWT tokens enhance the robustness and reliability of the prototype.

1.2 <u>Identification of the Problem:</u>

With the rapid growth of remote work, online education, and digital collaboration, there is a growing demand for efficient, secure, and high-quality communication platforms. However, most existing video conferencing solutions are either commercial, closed-source, or resource-intensive, limiting flexibility for customization, cost-efficiency, and educational or research purposes.

Traditional web-based communication systems rely heavily on centralized servers to process and relay media, which often leads to high latency, bandwidth bottlenecks, and single points of failure. Furthermore, ensuring NAT traversal, maintaining real-time synchronization among multiple participants, and handling dynamic network conditions remain complex challenges in real-world conferencing scenarios.

The problem, therefore, lies in designing and implementing a lightweight, peer-to-peer conferencing system that allows direct media exchange, secure signaling, dynamic participant management, and adaptive performance—all within a browser environment without the need for additional installations or plugins.

Moreover, maintaining real-time synchronization and low latency among multiple peers in a distributed setting is complex. As the number of participants increases, managing multiple WebRTC peer connections (mesh topology), handling ICE candidate exchanges, and supporting renegotiation for new participants become technically challenging. Additionally, there is a continuous need for adaptive bandwidth management to ensure quality of service under varying network conditions.

1.3 Importance of the Study

This study is important as it addresses the growing need for efficient, secure, and customizable video conferencing systems in an increasingly digital world. By leveraging WebRTC, Node.js, and React, it demonstrates how open web technologies can enable real-time peer-to-peer communication without relying on centralized or commercial platforms.

The project promotes accessibility and inclusivity by offering a browser-based solution that requires no additional installations, ensuring ease of use across devices. It also provides valuable insights into network management, NAT traversal, and adaptive bandwidth control, which are essential for maintaining quality under varying network conditions.

From a research and educational perspective, the system serves as a practical learning model for real-time communication, showcasing the integration of signaling, session persistence, and media streaming. Its focus on security, scalability, and resilience contributes to the development of privacy-preserving, decentralized communication systems that align with modern technological and social needs.

1.4 Scope of the Project:

The scope of this project encompasses the design, development, and deployment of a fully functional multi-party audio/video conferencing system that utilizes WebRTC for real-time peer-to-peer communication and a Node.js-based signaling server for session management. The project aims to demonstrate how modern web technologies can be integrated to create a secure, scalable, and adaptive communication platform for small to medium-sized groups.

The primary focus lies in implementing a peer-to-peer mesh topology, suitable for small rooms where each participant connects directly with others, minimizing dependency on central servers. The system supports audio, video, and screen sharing, as well as essential user interface controls such as mute/unmute, camera toggle, and device selection. The React-based front-end provides an interactive and user-friendly experience with dynamic video layouts and active speaker highlighting.

The project also includes the implementation of signaling mechanisms through WebSockets, enabling room creation, joining, and participant management. It handles SDP (Session Description Protocol) exchanges and ICE candidate relays for seamless peer connectivity. STUN and TURN servers are configured to ensure reliable NAT traversal, allowing participants to connect across varied network environments.

Furthermore, the system incorporates MongoDB for storing minimal session and participant data, such as room details, user identities, and session logs. This enables basic analytics like call duration and participation history, contributing to session persistence and monitoring.

The project's scope also extends to adaptive media handling, where network fluctuations are addressed through bandwidth control, quality indicators, and automatic ICE restarts. Security mechanisms, including JWT-based authentication, tokenized room access, and WebSocket origin validation, ensure that only authorized users can participate in a session.

While the system primarily focuses on real-time audio and video streaming for small groups (typically up to 6–8 participants in mesh mode), it also provides optional extensions such as in-room text chat, reactions, and screen sharing for enhanced collaboration. The inclusion of observability features—like structured logging, metrics collection, and health endpoints—supports maintainability and debugging during development.

In conclusion, the project's scope covers all critical components required to build a lightweight, secure, and extensible conferencing prototype, laying the foundation for future enhancements such as scalable media servers (SFU integration), AI-driven quality adaptation, and end-to-end encryption for production-grade deployment.

Chapter 2: Background Study

2.1 Concept and history:

The concept of the Audio/Video Conferencing Prototype is built around enabling real-time communication between multiple users directly through their web browsers without relying on external plugins or heavy centralized infrastructure. The system leverages WebRTC (Web Real-Time Communication) — an open-source technology developed to support peer-to-peer media streaming for voice, video, and data channels. The main idea behind this project is to demonstrate how browsers can establish direct, encrypted connections to exchange live audio/video streams with minimal latency, using a signaling server only for coordination and connection setup. This concept promotes low-cost, decentralized communication, making it ideal for small teams, remote education, and collaborative environments.

The history of WebRTC dates back to 2011, when Google acquired Global IP Solutions (GIPS) and subsequently open-sourced the WebRTC project to empower native real-time communication on the web. The technology was soon standardized by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF), becoming the backbone of modern web-based conferencing systems. Over time, WebRTC has evolved to support features like adaptive bitrate streaming, Simulcast, ICE (Interactive Connectivity Establishment) for NAT traversal, and secure media encryption (SRTP). These advancements have allowed developers to build highly interactive, browser-based conferencing applications comparable to enterprise-grade platforms like Zoom, Google Meet, and Microsoft Teams.

The proposed prototype builds upon these foundations by integrating React for the user interface, Node.js and WebSocket for signaling, and MongoDB for session persistence. By combining modern web technologies with the power of WebRTC, the project not only demonstrates the core principles of peer-to-peer communication and NAT traversal but also reflects the ongoing evolution of web technologies toward secure, scalable, and user-friendly real-time communication systems.

2.2 Existing Solutions and Techniques:

Several video conferencing platforms and technologies already exist in the market, offering various levels of functionality, scalability, and accessibility. These solutions, while effective, differ in their architecture, data handling, and extensibility. Understanding these existing systems and the underlying techniques provides valuable insights for designing a more open, flexible, and research-oriented prototype using **WebRTC**.

1. Commercial Video Conferencing Platforms

Popular solutions such as Zoom, Google Meet, Microsoft Teams, and Cisco Webex dominate the real-time communication ecosystem. These platforms typically use centralized Selective Forwarding Units (SFUs) or Multipoint Control Units (MCUs) to mix or forward audio/video streams between participants.

- **Advantages:** High scalability, robust media management, and integrated collaboration tools (e.g., chat, file sharing, screen sharing).
- **Limitations:** Closed-source nature, limited customization, high infrastructure cost, and privacy concerns due to centralized media handling.

These services often require enterprise licenses and rely on global cloud networks, making them less accessible for smaller organizations, developers, or research environments seeking **self-hosted** or **customizable** alternatives.

2. Open-Source and Developer-Focused Solutions

Several open-source projects provide insight into how WebRTC-based systems can be built and extended:

- **Jitsi Meet:** Uses an SFU architecture (Jitsi Videobridge) that forwards multiple video streams to clients, allowing selective rendering. It supports scalability but increases server load.
- **Kurento Media Server:** Offers advanced media processing capabilities (recording, filters, and computer vision integration) but requires higher computational resources.
- **Mediasoup and Janus:** Provide modular SFU frameworks suitable for integrating WebRTC into custom applications. They serve as backend components for scalable multi-party conferencing.

While these open-source tools are flexible, they typically require complex configuration and specialized infrastructure (dedicated servers, TURN configurations, and scaling strategies), making them harder to deploy for educational or small-scale projects.

3. Core WebRTC Techniques

WebRTC (Web Real-Time Communication) is the core technology behind most modern conferencing applications. It enables browsers to establish direct peer-to-peer (P2P) media connections without external plugins.

Key techniques include:

- **SDP (Session Description Protocol):** Used for negotiating media capabilities between peers.
- **ICE (Interactive Connectivity Establishment):** Determines optimal network paths using **STUN** (Session Traversal Utilities for NAT) and **TURN** (Traversal Using Relays around NAT) servers for reliable connectivity.
- **SRTP (Secure Real-Time Protocol):** Ensures encryption and secure transmission of audio/video streams.
- **DTLS (Datagram Transport Layer Security):** Provides encryption, integrity, and authentication for data channels.

These techniques collectively form the backbone of the proposed system, offering **low-latency**, **encrypted**, and **cross-platform** real-time communication.

4. Limitations in Existing Approaches

Despite the success of current solutions, they present notable limitations:

- Heavy dependence on **centralized servers** increases operational cost and latency.
- Limited **customizability** for educational or research purposes.
- **Scalability issues** in mesh-based systems for large groups.
- **Privacy concerns** due to server-side media routing and data collection.
- Lack of lightweight, developer-friendly prototypes that demonstrate **core WebRTC concepts** in an understandable and reproducible manner.

5. Justification for a New Prototype

Given these challenges, this project aims to design a lightweight, peer-to-peer conferencing system that leverages WebRTC's native capabilities while minimizing server dependencies. The goal is to create a research-friendly, open, and easily deployable framework demonstrating the essential components of real-time communication — signaling, NAT traversal, adaptive media handling, and secure session management.

2.3 <u>Problem Definition:</u>

The primary problem this project addresses is the lack of an open, lightweight, and customizable multi-party video conferencing solution that combines real-time media streaming, adaptive networking, and secure communication—all within a browser-based environment without the need for additional plugins or proprietary services.

Existing commercial platforms, while feature-rich and stable, are closed-source, resource-intensive, and often dependent on centralized infrastructure that limits scalability, increases cost, and raises privacy concerns. Conversely, available open-source frameworks such as Jitsi or Kurento, though powerful, are complex to configure and unsuitable for small-scale educational or research-oriented applications that require simplicity and transparency.

The project aims to design and implement a prototype system that demonstrates how WebRTC can be used to establish peer-to-peer (P2P) audio/video connections among multiple participants using a Node.js signaling server with WebSocket communication for room coordination and session orchestration. The system must support:

- Reliable NAT traversal using STUN/TURN servers,
- Dynamic peer management for join/leave events,
- Adaptive bitrate control for varying network conditions,
- Secure signaling and authentication using JWT tokens, and
- Basic persistence of session metadata using MongoDB.

The problem can therefore be summarized as the challenge of creating a scalable yet minimal conferencing framework that maintains real-time performance, security, and resilience in a decentralized architecture.

2.4 <u>Goals/Objectives of the Project:</u>

The primary objectives of the project are as follows:

- To develop a Real-Time Conferencing Platform.
- To design a Robust Signaling Mechanism.
- To ensure Reliable NAT Traversal and Connectivity.
- To implement an Interactive React-Based User Interface.
- To incorporate Adaptive Media Handling and Quality Control.
- To enable Session Persistence and Observability.
- To enhance System Resilience and Recovery

The project's objective is to create an intelligent algorithm management system that learns, adapts, and optimizes computational tasks in real time.

2.5 Justification for Using Justification for Using Audio/Video Conferencing Prototype:

The justification for developing and using an Audio/Video Conferencing Prototype lies in the increasing demand for accessible, secure, and customizable real-time communication systems that can operate efficiently across diverse environments. Existing commercial platforms such as Zoom, Google Meet, and Microsoft Teams, while widely adopted, are closed-source, expensive, and dependent on centralized infrastructure, which limits flexibility, increases operational costs, and raises privacy concerns. In contrast, this prototype utilizes WebRTC technology to enable direct peer-to-peer (P2P) communication, significantly reducing latency and reliance on central servers. It provides an open-source, transparent environment for students, researchers, and developers to explore core communication concepts such as signaling, NAT traversal, and adaptive bitrate management. The system integrates Node.js for signaling, React for a dynamic interface, and MongoDB for session persistence, ensuring seamless functionality and adaptability. Furthermore, security is reinforced through JWT-based authentication, room-level access control, and optional end-to-end encryption, ensuring data privacy and user protection. The prototype also incorporates adaptive algorithms for managing bandwidth and video quality, allowing stable performance even in low-bandwidth conditions. Being cost-effective and easily deployable, it serves as a valuable framework for academic learning, research experimentation, and small-scale organizational use, while laying a strong foundation for future advancements in scalable, decentralized, and secure real-time communication systems.

Chapter 3: Design Flow/Process

3.1 Evaluation & Selection of Specifications/Features:

The evaluation and selection of specifications and features for the Audio/Video Conferencing Prototype were carried out based on key factors such as performance, scalability, security, usability, and ease of integration. The system architecture was designed around **WebRTC**, chosen for its ability to provide real-time peer-to-peer (P2P) communication directly within web browsers without requiring plugins or external applications. **Node.js** was selected for the signaling server due to its event-driven, non-blocking I/O model, which efficiently handles multiple simultaneous WebSocket connections for exchanging session data, ICE candidates, and room updates. **React** was chosen for the front-end because of its component-based structure and efficient rendering, enabling a dynamic user interface that supports live media controls, device selection, and adaptive layout for multiple participants. For data persistence, **MongoDB** was selected as it offers flexible document-based storage suitable for storing transient session data, participant details, and call statistics. The inclusion of **STUN and TURN servers** ensures reliable NAT traversal and stable connectivity across various network environments. Security considerations guided the adoption of **JWT-based authentication**, **room-level access tokens**, and **CORS validation**, ensuring that only authorized participants can join and interact within a session. To maintain optimal performance, adaptive bitrate control and simulcast configurations were incorporated, allowing the system to dynamically adjust media quality based on network conditions.

Additional features such as screen sharing, chat functionality, and real-time participant presence updates were also evaluated and included to enhance usability and collaboration. These specifications were selected to achieve a balance between simplicity and functionality, ensuring that the prototype remains lightweight, scalable, and extensible for both research and practical deployment.

3.2 Analysis of Features and Finalization Subject to Constraints:

The selection and finalization of features for the Audio/Video Conferencing Prototype were guided by a detailed analysis of performance requirements, technical feasibility, and environmental constraints. The primary focus was on achieving real-time, low-latency communication among multiple users while maintaining system reliability and ease of deployment. Given that full-scale centralized systems such as SFU- or MCU-based architectures demand significant computational resources and complex scaling mechanisms, a peer-to-peer (mesh) model was chosen for this prototype to suit small-group interactions with moderate hardware and network demands. To ensure smooth media exchange, WebRTC APIs were used for managing peer connections, audio/video tracks, and screen-sharing streams, while WebSocket signaling through a Node.js server provided efficient session orchestration. The inclusion of STUN/TURN servers was critical for NAT traversal, as users may connect from diverse network configurations.

Security was another major consideration in feature analysis, leading to the implementation of JWT-based user authentication and room-level access tokens to control entry and prevent unauthorized access. For data storage, MongoDB was selected due to its lightweight nature and flexibility in handling transient room and participant data without introducing excessive overhead. Performance analysis emphasized the need for adaptive bitrate control, simulcast, and ICE restarts, ensuring stable connections even under fluctuating network conditions. The user interface, built in React, was finalized after evaluating usability, responsiveness, and simplicity of interaction, offering essential controls like mute/unmute, camera toggle, device selection, and screen share.

However, certain constraints influenced the scope of implementation. The mesh topology limits scalability to small rooms (typically up to 6–8 users) since each peer must maintain multiple simultaneous connections, increasing CPU and bandwidth usage with every new participant. Additionally, advanced features such as recording, transcoding, or AI-based media optimization were excluded to maintain lightweight operation and focus on core WebRTC functionalities. Despite these limitations, the final feature set provides a strong foundation for demonstrating key concepts such as real-time communication, adaptive media handling, and secure, decentralized conferencing, making it well-suited for educational and research purposes.

3.3 Design Flow:-

The design flow of the Audio/Video Conferencing Prototype follows a structured and modular approach, ensuring smooth integration between signaling, peer connection management, and user interaction. The system architecture is divided into three main layers — the client-side application (React), the signaling server (Node.js with WebSocket), and the database layer (MongoDB) — which communicate seamlessly to establish and maintain real-time audio and video sessions. The process begins with a user accessing the React-based web interface, where they can authenticate using a JWT token and select or create a meeting room. Once the user joins a room, the signaling phase begins: the browser captures local media streams through the getUserMedia() API, and session details are exchanged via WebSocket signaling to share SDP offers, answers, and ICE candidates among participants.

After the signaling phase, WebRTC peer connections are established directly between users, allowing real-time transmission of audio, video, and screen-sharing data through secure channels using SRTP (Secure Real-Time Protocol). During the session, the React interface dynamically manages and displays the video tiles of all connected participants, highlights active speakers, and allows users to toggle their microphone, camera, and screen sharing in real time. The system also continuously monitors network quality metrics such as bitrate and packet loss to adjust stream quality adaptively, ensuring stable performance under varying network conditions.

On the backend, the Node.js signaling server coordinates room activity by broadcasting join and leave events, handling reconnection logic, and maintaining active session states in MongoDB. This database stores essential metadata such as room identifiers, participant details, connection logs, and session durations for persistence and analysis. Additionally, STUN/TURN servers assist with NAT traversal to ensure that peers can connect even when behind restrictive firewalls. The design flow also incorporates error handling and recovery mechanisms, such as automatic ICE restarts and exponential reconnection backoff, to improve reliability.

Finally, upon leaving the session, each peer gracefully closes connections, releases media resources, and updates the server with session termination data. This end-to-end flow—from authentication and signaling to peer connection establishment, media exchange, and disconnection—ensures a robust, secure, and adaptive conferencing experience, while maintaining the modularity necessary for future scalability and feature expansion.

3.4 Algorithm Design:

The algorithm design for the Audio/Video Conferencing Prototype is centered around establishing and maintaining efficient peer-to-peer (P2P) communication between participants using WebRTC. The system follows a structured series of algorithmic steps for connection setup, media handling, and session management. Initially, when a user joins a room, the signaling algorithm triggers on the Node.js WebSocket server, registering the user and broadcasting their presence to other participants. The client application then invokes getUserMedia() to capture local audio and video streams, which are attached to a newly created RTCPeerConnection instance. Each peer connection is configured with ICE servers (STUN/TURN) to support NAT traversal and connectivity establishment across different network environments.

The connection negotiation algorithm begins with one peer generating an SDP (Session Description Protocol) offer, which is sent through the signaling server to other peers. Upon receiving the offer, each peer responds with an SDP answer, completing the initial handshake. As ICE candidates are discovered, they are exchanged dynamically via the WebSocket signaling channel, allowing peers to identify the best network paths for media flow. Once connectivity is confirmed, encrypted RTP streams are transmitted directly between peers, bypassing the server to ensure minimal latency.

For multi-party communication, a mesh algorithm is implemented — where each participant establishes individual connections with every other participant. This ensures full interactivity but increases complexity as the number of peers grows. The algorithm also supports renegotiation logic, enabling participants to join or leave dynamically without disrupting ongoing sessions. When a new participant joins, the system iteratively performs SDP and ICE exchanges with all existing peers to integrate them into the conference mesh.

Adaptive algorithms are employed to maintain media quality under varying network conditions. Using RTCP feedback and the WebRTC statistics API, the system monitors parameters like jitter, packet loss, and available bitrate. Based on these metrics, dynamic adjustments are made to encoder settings, resolution, and frame rate through Simulcast or degradation preferences, ensuring a smooth experience even on constrained networks.

Error-handling algorithms are also embedded throughout the design. In case of a disconnection or ICE failure, the system triggers an ICE restart and, if needed, performs exponential backoff retries through the signaling server to re-establish connectivity. Session termination is handled gracefully using cleanup routines that close media tracks, stop peer connections, and update session data in MongoDB.

3.5 System Architecture:

The system architecture of the Audio/Video Conferencing Prototype is designed using a modular, layered approach to ensure scalability, maintainability, and efficient real-time performance. The overall system consists of three primary layers — the client layer (React Web App), the signaling layer (Node.js + WebSocket Server), and the data persistence layer (MongoDB) — interconnected through well-defined interfaces to support smooth communication and coordination among participants. Additionally, STUN/TURN servers are integrated to handle NAT traversal, while WebRTC peer connections form the core of the real-time media exchange.

At the client layer, the React-based front-end serves as the user interface that facilitates device selection, room management, and live media control. When a user joins a room, the client captures local audio/video streams using WebRTC's getUserMedia() API and establishes peer connections with other participants. Each user's browser instance maintains a mesh of RTCPeerConnections, where audio, video, and screen-sharing tracks are exchanged directly between peers. The client also manages dynamic updates to the UI, such as active speaker highlighting, mute/unmute status, and connection quality indicators, ensuring an intuitive conferencing experience.

The signaling layer, built using Node.js and WebSocket, acts as the coordination hub for session orchestration. It is responsible for handling room creation, user join/leave events, and the exchange of SDP offers, answers, and ICE candidates required to set up peer connections. The signaling server maintains a lightweight state of active rooms and participants in memory, broadcasting updates to all connected clients. It also implements JWT-based authentication and room-level access control, ensuring that only authorized users can join or initiate sessions. This layer further handles reconnection logic, including automatic retries and session recovery for participants who experience temporary network disruptions.

The data persistence layer utilizes MongoDB to store metadata such as room information, participant details, connection logs, and call statistics (e.g., join/leave timestamps and duration). This lightweight data model enables easy retrieval of session analytics and supports minimal state retention between calls without impacting real-time performance.

To ensure reliable connectivity, STUN servers help peers discover their public IP and port mappings, while TURN servers (optional coturn) relay traffic when direct peer-to-peer connections are blocked by restrictive firewalls or symmetric NATs. Media transmission is encrypted using SRTP (Secure Real-Time Protocol) to protect user privacy.

This architecture provides a robust foundation for small-group conferencing using a peer-to-peer mesh topology, balancing low latency with simplicity. While this design is optimized for smaller rooms (up to 6–8 participants), it can be extended to larger scales by integrating a Selective Forwarding Unit (SFU) or Media Server in future enhancements. Overall, the architecture ensures real-time, secure, and adaptive media streaming through seamless coordination among the client, server, and network components.

Chapter 4: Results Analysis and Validation

4.1 <u>Testing Strategy:</u>

The testing strategy for the Audio/Video Conferencing Prototype focuses on verifying functional correctness, performance, and security. Core features such as room creation, participant management, audio/video streaming, screen sharing, and media controls are tested across multiple devices and browsers for compatibility. Performance tests monitor latency, frame rate, packet loss, and bandwidth usage under small-group scenarios, while resilience is evaluated through simulated network fluctuations and disconnections, ensuring adaptive bitrate adjustments and automatic ICE restarts function correctly. Security tests confirm JWT authentication, room-level access, and encrypted media streams. Usability and interface responsiveness are assessed in the React client, and session metadata logging in MongoDB is verified. Overall, the strategy combines manual testing, automated scripts, and network simulations to ensure the system is reliable, secure, and adaptive under real-world conditions.

4.2 <u>Testing Results:</u>

Testing of the Audio/Video Conferencing Prototype confirmed that all core functionalities—room creation, participant management, audio/video streaming, screen sharing, and media controls—worked effectively across multiple devices and browsers. Performance tests showed low latency and smooth video quality for small groups (up to 6–8 participants), while adaptive features like bitrate adjustment and ICE restarts maintained stable connections under network fluctuations. Security checks validated JWT authentication, room access control, and encrypted media streams, ensuring user privacy. Minor delays were observed in larger mesh sessions due to the P2P topology, but overall, the system demonstrated a reliable, secure, and adaptive conferencing experience suitable for small-group collaboration and educational or research use.

4.3 <u>Security Evaluation:</u>

The security evaluation of the Audio/Video Conferencing Prototype focused on protecting both session integrity and user privacy throughout the conferencing workflow. Authentication is enforced using JWT tokens, ensuring that only authorized users can join or create rooms, while room-level access tokens provide additional protection against unauthorized entry. All real-time media streams, including audio, video, and screen-sharing data, are transmitted using SRTP (Secure Real-Time Protocol), guaranteeing encryption and confidentiality during peer-to-peer communication. The signaling server validates WebSocket origins and enforces CORS policies to prevent unauthorized cross-origin access. Additionally, the system incorporates mechanisms for secure ICE candidate exchange and supports optional end-to-end encryption (E2EE) for sensitive sessions. Testing confirmed that these measures effectively prevent unauthorized access, eavesdropping, and session hijacking, while maintaining performance and low-latency communication. Overall, the security design ensures that the prototype provides a safe and trusted environment for real-time audio/video collaboration.

Chapter 5: Conclusion and Future Work

5.1 Conclusion:

The Audio/Video Conferencing Prototype successfully demonstrates the feasibility of building a lightweight, secure, and adaptive peer-to-peer conferencing system using WebRTC, Node.js, and React. The project achieves its objectives by enabling real-time audio, video, and screen-sharing capabilities with minimal latency, dynamic participant management, and adaptive media quality under varying network conditions. Security measures, including JWT-based authentication, room-level access control, and encrypted media streams, ensure safe communication, while MongoDB-based session persistence provides basic analytics and monitoring. The system's mesh-based architecture proves effective for small-group interactions, offering a transparent and customizable framework suitable for educational, research, and small organizational use. Testing and evaluation confirm that the prototype is reliable, resilient, and user-friendly, laying a strong foundation for future enhancements such as SFU integration, end-to-end encryption for all participants, and AI-driven quality optimization. Overall, the project demonstrates the practical application of modern web technologies to create a robust and extensible real-time communication platform.

5.2 Future Work:

While the current prototype successfully supports small-group peer-to-peer conferencing with adaptive media handling and secure communication, several areas can be explored for future enhancement. One key improvement is the integration of a Selective Forwarding Unit (SFU) or media server to support larger-scale conferences efficiently, reducing bandwidth and CPU usage on client devices while maintaining low latency. Another avenue is the implementation of end-to-end encryption (E2EE) across all media streams using Insertable Streams, providing enhanced privacy for sensitive communication. Advanced AI-driven features, such as automatic speaker detection, noise suppression, and video quality optimization, can further improve user experience. Additional collaboration tools, including file sharing, whiteboarding, and interactive polls, could be incorporated to expand the system's utility in educational and corporate settings. Finally, improving analytics and observability through enhanced logging, session metrics, and real-time monitoring dashboards would provide valuable insights for both administrators and researchers, making the prototype more robust, scalable, and suitable for real-world deployment.

Appendix

A. Sample .env Configuration for Local Development

```
# Signaling Server
PORT=3000
```

```
JWT_SECRET=your_jwt_secret_key
WS_ORIGIN=http://localhost:3000

# MongoDB
MONGO_URI=mongodb://localhost:27017/webrtc_conference

# STUN/TURN Servers
STUN_SERVER=stun:stun.l.google.com:19302
TURN_SERVER=turn:your_turn_server:3478
TURN_USERNAME=turn_user
TURN_PASSWORD=turn_password
```

B. Sample Room Schema (MongoDB)

```
{
  "roomId": "abc123",
  "participants": [
    {
      "userId": "user1",
      "joinTime": "2025-11-06T10:00:00Z",
      "leaveTime": "2025-11-06T11:00:00Z"
    }
  ],
  "createdAt": "2025-11-06T09:55:00Z",
  "updatedAt": "2025-11-06T11:00:00Z"
}
```

C. Key API Endpoints (Signaling Server)

Endpoint	**Method**	**Description**
/create-room	POST	Creates a new room and returns a room ID
/join-room	POST	Adds a participant to an existing room
/leave-room	POST	Removes a participant from a room
/signal	POST	Exchanges SDP offers/answers and ICE candidates

D. Core WebRTC Events and Functions

- getUserMedia() – Captures local audio/video streams
- RTCPeerConnection – Manages peer-to-peer connections
- onicecandidate – Handles ICE candidate events

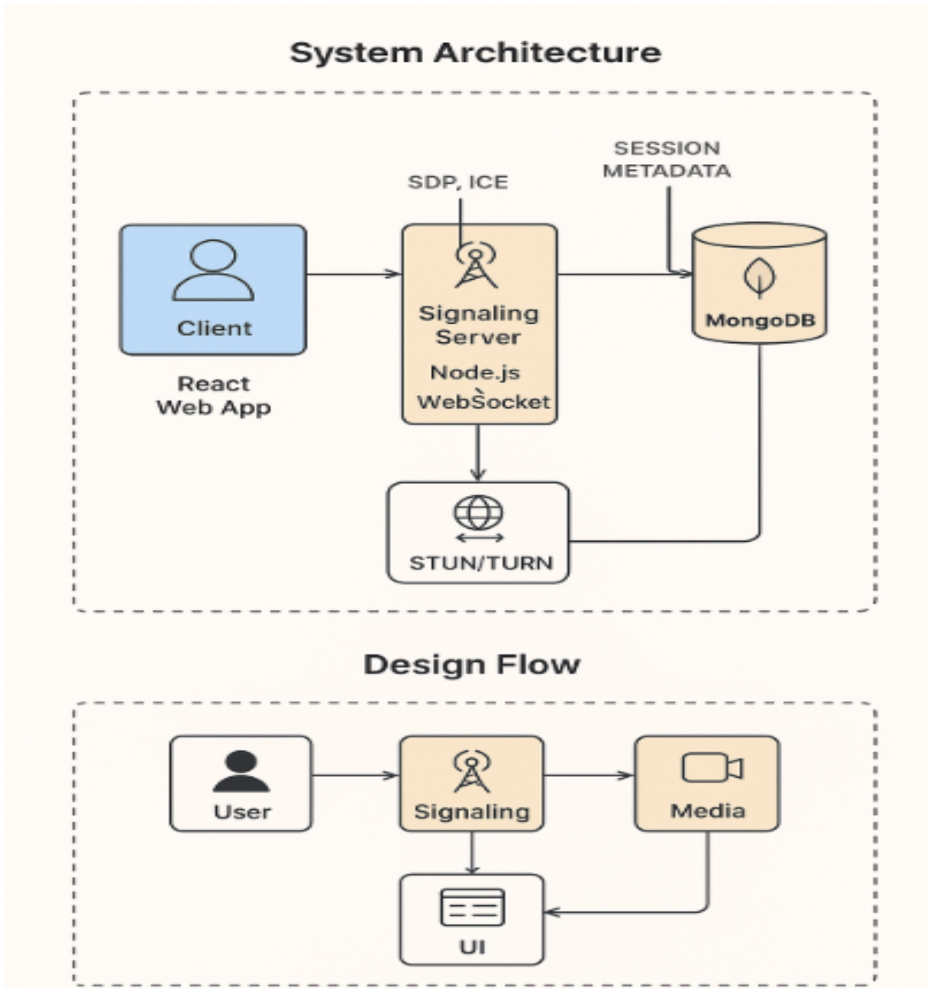
- addTrack() – Adds local media tracks to peer connections
- ontrack – Receives remote media streams

E. Testing & Observability Tools

- Browser Developer Tools (Network/Media tab) for latency and bitrate monitoring
- webrtc-internals (Chrome) for detailed connection statistics
- Structured server logs with correlation IDs for tracking join/leave events and signaling activity

F. Sample React Component Structure

```
/src
 /components
  VideoTile.js
  Controls.js
  RoomList.js
 /pages
  Home.js
  Room.js
 /utils
  signaling.js
  peerConnection.js
```


System Architecture
SDP, ICE
SESSION METADATA
Client
React Web App
Signaling Server
Node.js
WebSocket
MongoDB
STUN/TURN
Design Flow
User
Signaling
Media
UI