

Chaipemeet: A Full-Stack Browser-Native Real-Time Web Video Conferencing Architecture Using Mesh WEBRTC Topologies and Socket.IO Signalling Systems

Shiv Kumar

Department Of Computer Science And Engineering

RTCIT, Ranchi

HARSH KUMAR GUPTA

Department Of Computer Science And Engineering

RTCIT, Ranchi

VISHAL KUMAR PANDEY

Department Of Computer Science And Engineering

RTCIT, Ranchi

ANKIT KUMAR MISHRA

Department Of Computer Science And Engineering

RTCIT, Ranchi

AKSHAY SINGH

Department Of Computer Science And Engineering

RTCIT, Ranchi

OHIT KUMAR SINGH

Department Of Computer Science And Engineering

RTCIT, Ranchi

Abstract

This study details the design, engineering, and performance evaluation of **ChaiPeMeet**, a self-contained, browser-native, real-time video conferencing application . The platform addresses commercial limitations, security fragmentation, and software installation overhead found in proprietary corporate solutions . Built entirely on open web standards, the proposed system combines a stateless **React.js** frontend with an asynchronous **Node.js/Express.js** backend. Persistent user management and room configurations are managed via a document-oriented **MongoDB** instance . Real-time full-duplex signaling and interactive text streams run over **Socket.IO**, while secure audio-video distribution uses mesh-based peer-to-peer **WebRTC** pipelines . Performance testing shows that the architecture achieves end-to-end media and chat transmission latency of under 200 milliseconds under normal operation (pp. 27, 43). This paper provides a complete architectural blueprint and implementation framework for open-source real-time communication systems .

Keywords—WebRTC, Socket.IO, real-time communication, peer-to-peer (P2P), mesh topology, Network Address Translation (NAT) traversal, Session Description Protocol (SDP), full-stack development, MERN stack, React.js, Node.js, Express.js, MongoDB, JSON Web Tokens (JWT), cryptography, bcrypt, getUserMedia API,getDisplayMedia API, multimedia streaming, low-latency applications.

1. Introduction

High-speed internet expansion and native browser optimizations have shifted real-time web services away from proprietary software towards open, standardized web protocols . Video conferencing has transitioned from a specialized enterprise resource into essential infrastructure for remote work, education, and digital collaboration .

However, widely used market tools like Zoom, Google Meet, and Microsoft Teams present specific engineering and operational challenges :

- They route metadata through closed, third-party infrastructure .
- They impose arbitrary time and group size limits on free usage tiers .
- They require regular desktop software installations or external plugin support .

ChaiPeMeet addresses these issues by delivering an ad-hoc, plugin-free, multi-user conferencing workspace that runs inside a single browser tab The core web communication technologies utilized include:

- **WebRTC**: Powers native, browser-to-browser media stream routing without third-party installations .
- **Socket.IO**: Establishes full-duplex communication channels for real-time state synchronization and signaling .
- **Signaling Servers**: Coordinates peer discovery and session parameter exchange before the peer-to-peer connection is opened .

The project proves that open-source stacks can sustain high-performance web communication safely, cost-effectively, and efficiently .

2. Literature Review

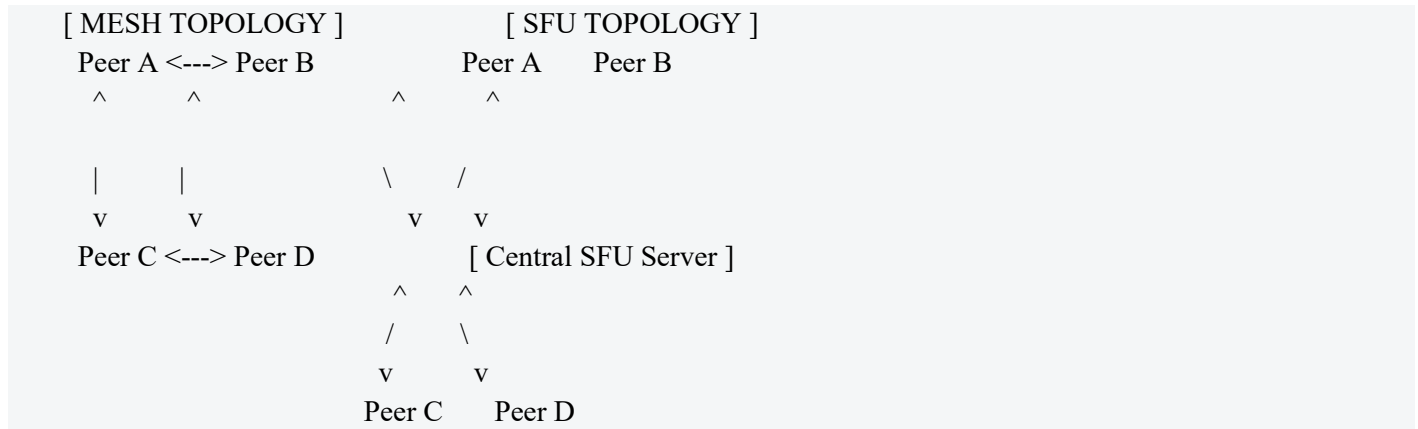
2.1 The Evolution of Web-Based Real-Time Communication

Early real-time web engines required third-party plugins like Adobe Flash, Silverlight, or custom Java applets . These setups created several problems, including high CPU utilization, memory leaks, security vulnerabilities, and fragmented user experiences . The launch of HTML5 introduced native media capture using the `getUserMedia` API . This laid the groundwork for the WebRTC standard, developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) .

2.2 Architectural Topologies for Multi-Party Real-Time Video

Modern video conferencing systems use one of three primary structural topologies:

1. **Mesh (Peer-to-Peer)**: Every participant connects directly to every other participant. This eliminates central media server costs but increases upstream bandwidth demands on the client ($O(N^2)$ scaling) .
2. **Selective Forwarding Unit (SFU)**: A central server receives media streams from each client and forwards them to all other participants without modification. This balances server costs and client resource requirements.
3. **Multipoint Control Unit (MCU)**: A central server collects, decodes, mixes, and re-encodes all incoming streams into a single combined output for each client. This reduces client processing requirements but significantly increases server-side computing costs.



ChaiPeMeet uses a direct Mesh topology, making it highly effective for small groups by routing media streams without media server overhead .

2.3 Network Address Translation (NAT) Traversal and Signaling

Because WebRTC routes media directly between browsers, it must handle firewalls and private network boundaries . This traversal relies on two main protocol systems :

- **STUN (Session Traversal Utilities for NAT)**: Allows clients to discover their public-facing IP addresses and port configurations.
- **TURN (Traversal Using Relays around NAT)**: Relays media streams through a fallback server when direct P2P connections fail due to strict, symmetric NAT configurations .

WebRTC manages media transfer but does not define an internal signaling protocol . Instead, applications must implement their own signaling channels—using technologies like WebSockets or Socket.IO—to exchange Session Description Protocol (SDP) configurations and ICE candidates .

3. Problem Statement & Research Gaps

While commercial video tools are widely available, they present several operational challenges for independent organizations, academic institutions, and privacy-focused developers :

- **Vendor Lock-In and High Licensing Costs**: Proprietary systems impose data retention policies and restrict programmatic access, limiting customization options .
- **Installation and Maintenance Friction**: Mandatory desktop clients create setup hurdles for end users and complicate configuration management for network administrators .
- **Data Sovereignty Vulnerabilities**: Routing meeting metadata through external, proprietary cloud setups poses compliance risks for sensitive or regulated organizational workflows .
- **Feature Restrictions**: Essential collaboration tools, such as presentation sharing and persistent in-session chat, are frequently restricted to paid subscription tiers .
- **Documentation Gaps in Academic Literature**: Existing research often covers isolated WebRTC concepts but lacks end-to-end implementation guides for securing, persisting, and scaling real-time web applications .

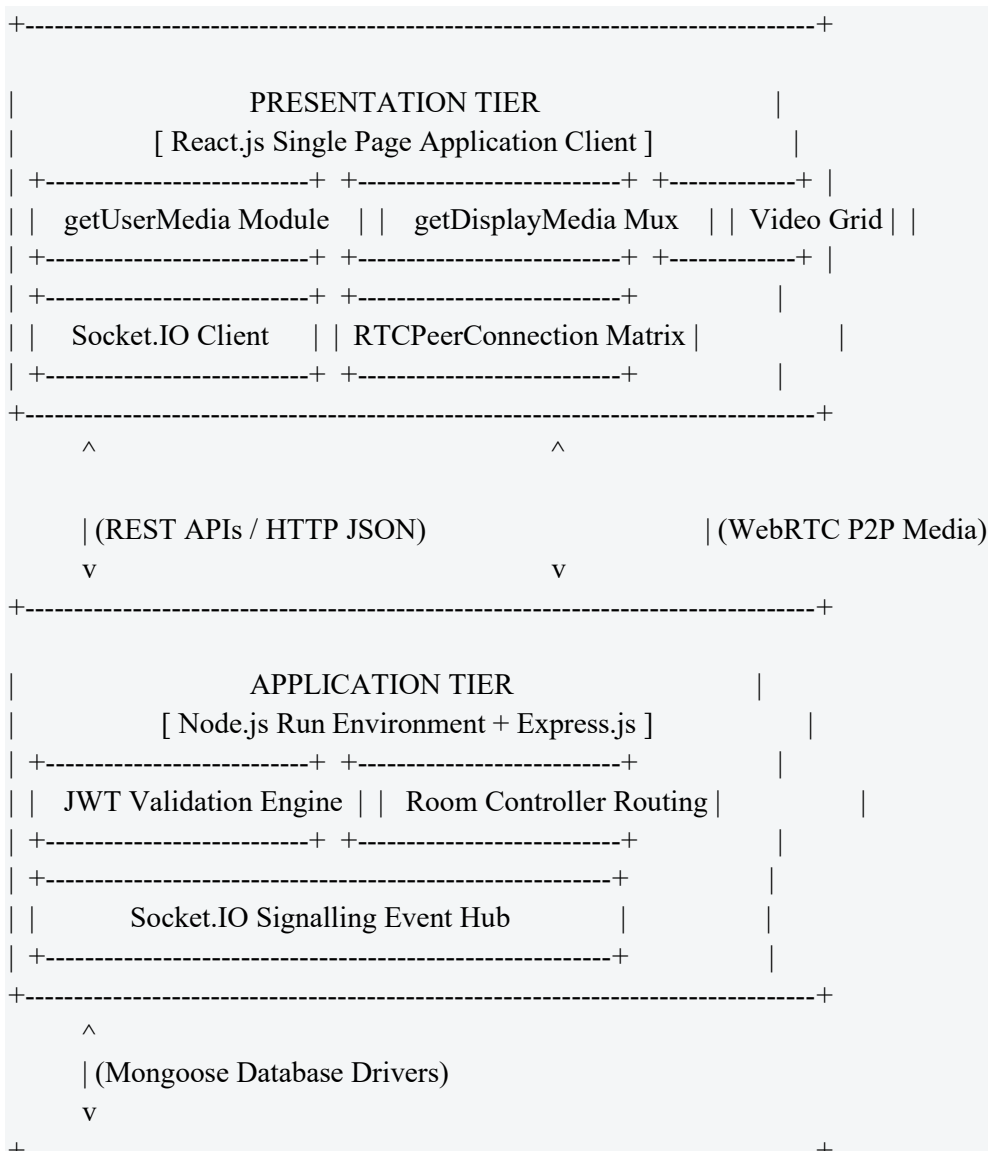
4. Objectives of the System

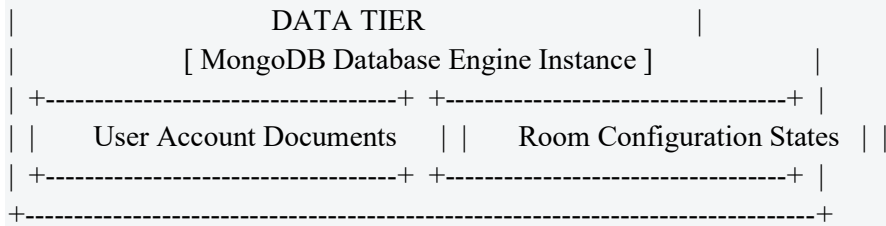
To address these gaps, the development of ChaiPeMeet focused on several core engineering goals :

- **Secure, Stateless Authentication:** Implement a registration and login flow using `bcrypt` for password hashing and JSON Web Tokens (JWT) for secure route verification .
- **Low-Latency Multi-Party Video:** Coordinate ad-hoc peer connections through a managed signaling server to stream synchronized audio and video tracks .
- **Native Screen Broadcasting:** Use the browser's native `getDisplayMedia` API to stream presentations smoothly without requiring external plugins or extensions .
- **Synchronized Chat Panel:** Build a low-latency chat interface using Socket.IO to share text, code snippets, and links during active calls .
- **Structured Data Tracking:** Implement schema definitions using an Object Data Modeling (ODM) library to track user profiles and room records cleanly .

5. System Architecture & Component Design

ChaiPeMeet uses a decoupled, three-tier architecture to separate presentation, application logic, and storage layers .





5.1 Presentation Tier

Built as a single-page application using React.js, this tier manages the user interface, handles browser media capture, and controls the WebRTC connection lifecycle . It features a responsive layout powered by CSS3 Grid and Flexbox that adjusts the video tile arrangement dynamically as users join or leave the session .

5.2 Application Tier

The application tier uses Node.js and Express.js to provide a RESTful API for user authentication and room lookup . It also runs a Socket.IO instance to handle real-time signaling events, room classification, and live chat message distribution.

5.3 Data Tier

A MongoDB instance serves as the storage layer, using Mongoose for schema enforcement and query validation . This layer tracks permanent user accounts and room data, while chat messages are held in server memory for the duration of the session to limit persistent database operations .

6. Database and Data Model Design

The data architecture relies on structural database collections modeled through Mongoose schemas .

6.1 User Document Schema

```
const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true, index: true },
  passwordHash: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

6.2 Room Document Schema

```
const RoomSchema = new mongoose.Schema({
  roomId: { type: String, required: true, unique: true },
  hostId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  createdAt: { type: Date, default: Date.now },
  isActive: { type: Boolean, default: true }
});
```

6.3 Transient Message Data Structure

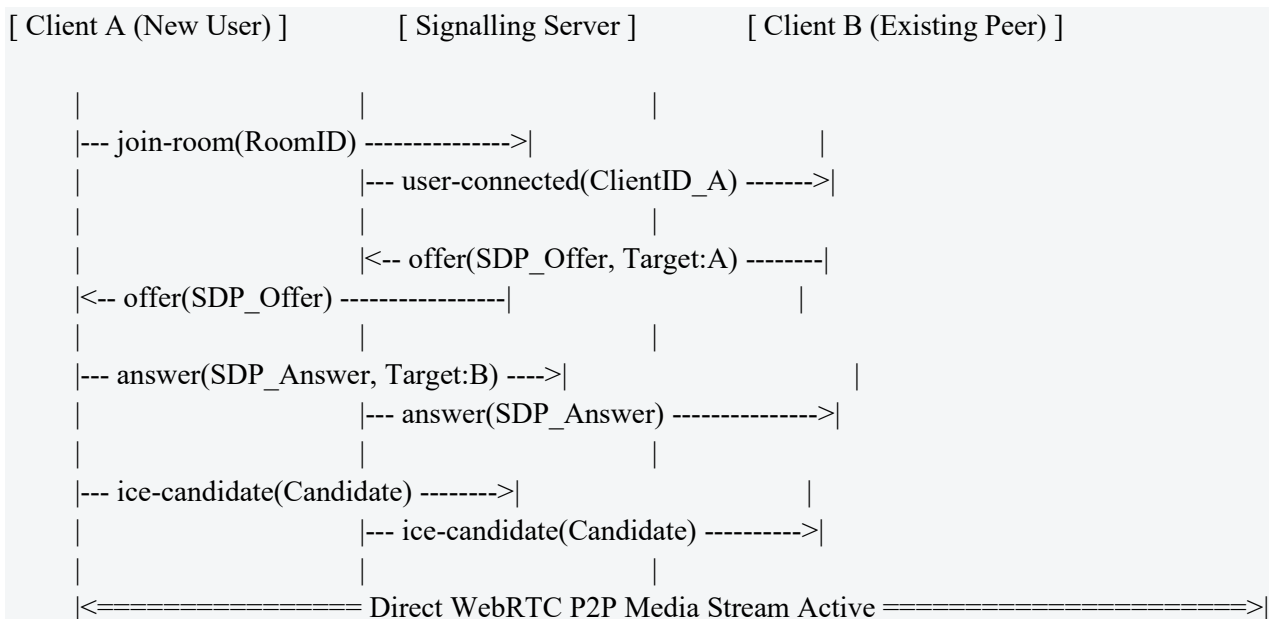
Chat records are held in memory during the conference session rather than written to permanent storage . They use the following data structure:

```
{
  "roomId": "string-uuid-identifier",
  "senderId": "object-id-reference",
  "senderName": "string-display-name",
  "text": "string-message-payload",
  "timestamp": "ISO-8601-date-string"
}
```

7. System Implementation & Flow Mechanics

7.1 Detailed WebRTC Signaling Procedure

Because WebRTC requires a signaling step to discover peers, clients coordinate through a series of Socket.IO events to exchange network and session metadata :



- Room Entry:** Client A emits `join-room` with the target room identifier .
- Peer Notification:** The server processes the request and sends a `user-connected` event to existing room members .
- SDP Exchange:** Client B creates an `RTCPeerConnection` for the new user, generates a Session Description Protocol (SDP) offer, and routes it through the server . Client A receives the offer, configures its remote settings, generates an SDP answer, and sends it back to complete the media negotiation .
- ICE Candidate Discovery:** Both clients gather and exchange Interactive Connectivity Establishment (ICE) candidates to discover the optimal network route .
- Connection Finalization:** Once the network path is verified, the `ontrack` event fires on each browser, mounting the incoming media streams onto the React video grid .

7.2 Core Socket.IO Event Management Matrix

The signaling backend monitors and routes several primary coordination events :

- `join-room` : Registers a user's socket connection to a specific logical room namespace .
- `offer` : Relays structural SDP media capability records from an initiating client to a target peer .
- `answer` : Forwards response media descriptions back to the initiating peer .
- `ice-candidate` : Shares network connectivity parameters between endpoints to assist NAT traversal .
- `send-message` : Distributes chat payloads immediately to all active sockets within the room .
- `screen-share-start` / `screen-share-stop` : Updates remote user interfaces when a participant starts or stops presenting .
- `disconnect` : Cleans up participant references on the server and notifies remaining room members .

7.3 Core Signaling Implementation Example

The server-side signaling hub manages room grouping and routes message payloads using a central event block:

```
// Node.js Signalling Server Block Setup
```

```
io.on('connection', (socket) => {  
  socket.on('join-room', (roomId, userId) => {  
    socket.join(roomId);  
    socket.to(roomId).emit('user-connected', userId);  
  
    socket.on('offer', (payload) => {  
      io.to(payload.target).emit('offer', { sdp: payload.sdp, sender: socket.id });  
    });  
  
    socket.on('answer', (payload) => {  
      io.to(payload.target).emit('answer', { sdp: payload.sdp, sender: socket.id });  
    });  
  
    socket.on('ice-candidate', (payload) => {  
      io.to(payload.target).emit('ice-candidate', { candidate: payload.candidate, sender: socket.id });  
    });  
  
    socket.on('disconnect', () => {  
      socket.to(roomId).emit('user-disconnected', userId);  
    });  
  });  
});
```

7.4 Screen Sharing Track Modification Mechanics

To share a presentation, the system uses the browser's `getDisplayMedia` API to capture the screen view . It then locates the active video sender within each established `RTCPeerConnection` and swaps the camera track for the display track :

```
// Screen Share Track Replacement Routine
```

```
async function initiateScreenShare(peersMap) {
  try {
    const displayStream = await navigator.mediaDevices.getDisplayMedia({ video: true });
    const screenTrack = displayStream.getVideoTracks()[0];

    peersMap.forEach((peerConnection) => {
      const senders = peerConnection.getSenders();
      const videoSender = senders.find(sender => sender.track.kind === 'video');
      if (videoSender) {
        videoSender.replaceTrack(screenTrack);
      }
    });
  } catch (error) {
    console.error("Screen capture initialization failed:", error);
  }
}
```

This track replacement mechanism updates the stream without renegotiating the core network connection or restarting the ICE gathering process .

8. Experimental Results, Testing, & Performance Discussion

8.1 Functional Verification Testing Matrix

We validated the system using standardized test scenarios to confirm correct behavior across modules :

Case ID	Targeted System Module	Provided Input Parameters	Expected System Behavior	Verification Status
TC-01	Account Registration	Valid string sets (Name, Unique Email)	Account generated; password securely hashed via bcrypt	Pass
TC-02	Account Registration	Previously registered email string	Decline request; return validation warning message	Pass
TC-03	User Authentication	Valid matching credentials	Generate and issue signed session JWT	Pass
TC-04	User Authentication	Incorrect password string parameter	Decline access; display credential error message	Pass
TC-05	Space Creation	Authenticated user button action	Create unique UUID room key; redirect to interface	Pass
TC-06	Room Connection Link	Valid active room UUID marker	Connect client socket; notify existing room peers	Pass
TC-07	Room Connection Link	Invalid room string identifier	Block connection; route user back to error view	Pass
TC-08	Multi-User Conference	Multiple devices using same ID	Connect direct bidirectional P2P video links	Pass

TC-11	Presentation Broadcast	Trigger browser viewport sharing	Replace local video track with screen capture	Pass
TC-13	In-Session Live Chat	Dynamic message payload submission	Deliver text messages instantly to all room members	Pass

8.2 Latency Evaluation & Topology Discussion

Performance testing shows that the signaling server and text chat panel consistently achieve delivery speeds under 200 milliseconds on a standard broadband connection .

[Observed System Latency Benchmarks]

Signalling Latency: < 200 ms [=====]

Text Chat Delivery: < 200 ms [=====]

Target Performance: < 200 ms [=====]

Because the application uses a direct Mesh network topology, media streams travel directly between client browsers without passing through a middle server tier . This design keeps server costs low, as the backend only processes lightweight signaling messages rather than resource-heavy video routing.

However, this mesh approach increases client-side resource usage as more participants join, since each browser must upload its media stream to every other peer individually . Testing indicates that this topology maintains stable connection quality for small sessions with 2 to 4 simultaneous active participants.



9. Conclusion & Future Scope

ChaiPeMeet demonstrates that low-latency, secure video conferencing applications can be built purely on open web standards . By combining WebRTC for peer-to-peer media transfer with Socket.IO for signaling, the platform eliminates the need for central media infrastructure, providing a self-hostable alternative to commercial services . The system handles user security through stateless JWT tokens and separates interface design from data persistence to ensure modularity .

Future Scope

To scale the application for larger groups and deployment environments, future updates will focus on three main areas :

1. **Transition to SFU Architecture:** Integrating a Selective Forwarding Unit (SFU) to reduce upstream bandwidth requirements from $O(N^2)$ to $O(N)$, allowing more users to join each room .
2. **AI-Powered Media Processing:** Introducing browser-native WebAssembly models to provide real-time noise suppression and background blur.
3. **Session Archiving:** Implementing server-side media recording pipelines to save and playback conference sessions safely .

10. References

1. Johnston, A. B., & Burnett, J. (2014). *WebRTC: Web Real-Time Communication Simplified*. Digital Bazaar Publishing.
2. Loreto, S., & Romano, S. P. (2012). Real-Time Communication for the Web. *IEEE Internet Computing*, 16(5), 87-91.
3. Rescorla, E. (2018). *WebRTC Security Architecture*. Internet Engineering Task Force (IETF), RFC 8827.
4. Foundation, O. S. (2023). *Socket.IO Client Real-Time Framework Specification Engine*. Documentation Registry, v4.x.
5. Extensible Web Standardization Group. *The W3C Screen Capture Interface Standard Library*. World Wide Web Consortium.
6. Central Project Documentation. (2026). *ChaiPeMeet: Real-Time Web Video Conferencing Application Major Report*. Department of Computer Science, RTC Institute of Technology (pp. 1-2).