EchoMesh – A Distributed VoIP Communication Platform

Submitted by:  
• Pratham Shah (B039)  
• Rishikesh Prabhu (B029)  
• Ishita Santosh (B001)

**NMIMS MPSTME**

Department of Computer Engineering

## 1. Introduction

Voice over **Internet Protocol (VoIP)** technology has transformed traditional communication by enabling **real-time audio communication** over the internet. **EchoMesh** is a lightweight, **browser-based VoIP platform** that supports **peer-to-peer connections**, allowing users to join a shared room using a simple access code. This approach initiates direct voice sessions without requiring centralized servers. The platform leverages principles of **distributed computing** to offer enhanced **security**, **low latency**, and **minimal infrastructure dependencies**.

## 2. Problem Statement

Most existing **VoIP solutions** depend on **centralized servers**, which introduces concerns around **scalability**, **latency**, and **privacy**. EchoMesh addresses these issues by providing a **peer-to-peer**, **decentralized communication** experience that minimizes dependency on central infrastructure. The system empowers users with **secure**, **efficient**, and **browser-based voice calling** capabilities, eliminating the need for account setups or application installations.

## 3. Literature Review

The foundation of EchoMesh is grounded in research and widely adopted standards. **RFC 7478** outlines the potential and use cases for **WebRTC** technology, while early **peer-to-peer networking research** (Schollmeier, 2001) demonstrates the advantages of decentralization. Performance evaluations of **WebRTC** (Jansen et al., 2018) confirm its ability to support **low-latency audio transmission**. Additionally, **Socket.io** documentation offers insights into building **real-time event-driven** applications that are essential in **distributed systems**. These studies and standards collectively influenced the core architecture of EchoMesh.

## 4.1 Related Work

The development of EchoMesh draws heavily upon concepts introduced in various **coursework modules**, especially those focused on **Distributed Computing**, **Computer Networks**, and **Web Technologies**. Central to the system is the implementation of a **peer-to-peer (P2P) architecture**, a concept introduced during coursework on distributed systems, which emphasized the importance of **decentralization** and **resource autonomy**. By leveraging **WebRTC**, a browser-native API for real-time communication, EchoMesh enables direct media exchange between clients, embodying the idea of **resource sharing** without intermediary servers—a key feature in distributed environments.  
  
The **signaling mechanism**, implemented using **Socket.io**, was inspired by modules on **event-driven programming** and **real-time communication models**. **Socket.io** handles signaling events that coordinate the **WebRTC handshake**, echoing coursework projects that dealt with **message passing** and **socket-level programming**. Additionally, the design decision to build a **stateless signaling server** using **Node.js** and **Express.js** demonstrates practical application of **RESTful API design principles** and **lightweight server models**, both taught in web development subjects.  
  
Another cornerstone of the system is its emphasis on **scalability and fault tolerance**. These principles were explored in assignments and simulations involving **load distribution** and **failover mechanisms**, where decentralized models outperformed centralized systems under stress. EchoMesh’s architecture, which allows horizontal scaling without increasing server complexity, is a direct extension of these classroom learnings.  
  
Furthermore, the choice of using browser-based audio processing—converting analog to digital and vice versa—draws from coursework in multimedia systems and digital signal processing, where the fundamentals of audio transmission and encoding were covered. EchoMesh thus becomes a living demonstration of integrating these varied academic foundations into a functional, real-world application.  
  
[Insert Code Screenshot: Socket.io signaling setup in server.js]  
[Insert Code Screenshot: WebRTC connection handling in client.js]

## 5. System Architecture

EchoMesh is composed of three main components. The **Client** is responsible for capturing audio and managing the **WebRTC session**. The **Signaling Server**, built using **Node.js** and **Socket.io**, coordinates peer connections by handling the signaling process. Finally, **WebRTC** establishes **encrypted peer-to-peer audio channels**, allowing direct and secure communication between browsers.

## 6. Technologies Used

**Node.js** is used for backend signaling and room management. **Socket.io** facilitates **real-time signaling** and event communication. **WebRTC** powers direct peer-to-peer media communication. The **frontend interface** is constructed using **HTML**, **CSS**, and **JavaScript**, while **Express.js** serves static content through a lightweight HTTP server.

## 7. Implementation Details

A user starts by entering a room code on the website. The **server** then checks for a peer using the same code. Upon a match, a **WebRTC handshake** is initiated through **Socket.io**, and **audio streams** are exchanged directly between browsers. The audio is **converted from analog to digital** at the sender’s end and reversed back to analog at the receiver’s end, ensuring seamless **voice communication** without a centralized relay.

## 8. Results and Testing

The platform achieved **low-latency communication** (under 200ms) and proved compatible across browsers including **Chrome**, **Firefox**, and **Edge**. It remained reliable under varying network conditions and supported **NAT traversal** successfully. The simple user experience required no account registration or application installation, demonstrating the practicality of browser-native **peer-to-peer VoIP**.

## 9. Conclusion

**EchoMesh** demonstrates the real-world viability of **distributed systems**. It showcases how **peer-to-peer VoIP** can be both **efficient** and **secure**, providing a blueprint for future **decentralized communication platforms** that prioritize user privacy and system scalability.

## 10. Future Enhancements

Proposed enhancements include adding **video support**, implementing **end-to-end (E2E) encryption**, supporting **group calls** using **mesh topology**, converting the platform into a **Progressive Web App (PWA)**, and integrating **presence indicators** to display active connected users in real time.

# 11. References

1. RFC 7478 – WebRTC Use Cases

2. https://webrtc.org

3. https://socket.io/docs/

4. https://nodejs.org/en/docs

5. Schollmeier, R. (2001). Peer-to-Peer Networking

6. Jansen, M. et al. (2018). WebRTC Benchmarks