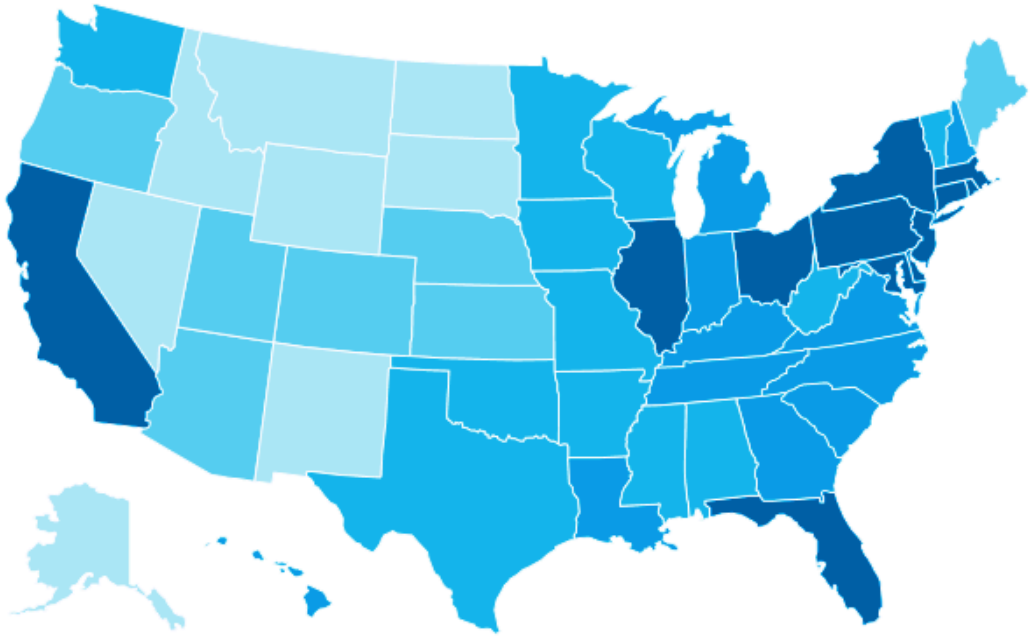

Computer Network Project Report

ChatBlink – Real-Time Messaging & Video Calling

Rayyan-ur-Rehman 23k-0634
Muhammad Abdullah khan 23k-0607
Abdul Ahad Munaf 23k-0590

Executive Summary

The **ChatBlink** project is a compact yet capable real-time communication system designed to facilitate instant text messaging and peer-to-peer video .Built on a Node.js and Express backend with Socket.IO serving as the communication layer, the application employs WebRTC to enable seamless video interactions between users. The overall technical approach emphasizes simplicity, modularity, and ease of deployment while demonstrating the core concepts behind real-time web applications.

Introduction

The primary motivation behind **ChatBlink** appears to be the development of a lightweight, accessible platform that allows users to communicate instantly without needing extensive setup or external services. Many existing chat solutions are either overly complex, resource-intensive, or tightly bound to commercial ecosystems.

The application supports anonymous usage scenarios such as random pairing by integrating video calling, the project expands beyond a typical text-only chat and enters the increasingly important realm of real-time multimedia communication. Through this combination, it solves the problem of enabling quick, secure, and flexible communication between users in a frictionless environment.

Project Architecture and Design

The architecture of **ChatBlink** follows a clear client–server model. The server, implemented with Node.js and Express and located in the `src/` directory, is responsible for serving frontend files and managing WebSocket connections via Socket.IO. This server acts as the signaling backbone for both text messaging and WebRTC-based video communication.

On the client side, the `public/` directory contains the actual user-facing interface, which consists of simple HTML pages enhanced by Bootstrap styling and JavaScript logic. These pages manage everything from user identity input to message display and video stream handling.

The application is organized around modular responsibility:

- `server.js` initializes the application and ties together configuration, Express routing, and Socket.IO events.
- The `handlers/` folder separates logic for chat and video functionalities (`chatHandler.js` and `videoHandler.js` respectively).
- The `config/` directory contains a central configuration file (`constants.js`), enabling clarity and consistency regarding environment variables and default values.

This modular structure ensures that each part of the system interacts cleanly with the others while remaining independently maintainable. Chat messages, room management, and signaling events all pass through clear Socket.IO channels, while the browser handles rendering and WebRTC peer connections after the initial handshake.

Technical Implementation

Core Logic

At the heart of **ChatBlink** lies its event-driven communication model. When the server boots up through `server.js`, it establishes a Socket.IO instance that listens for incoming client connections. Once a client joins, they may choose to participate in random matching or select a specific room.

Text chat flows through well-defined events such as `chatMessage`, while room assignment and user pairing rely on additional event handlers within `chatHandler.js`. The video-calling mechanism follows the typical WebRTC pattern: browsers capture audio and video through `getUserMedia()`, then coordinate the creation of peer connections. The signaling process—transmitting offer, answer, and ICE candidate exchanges—occurs entirely via Socket.IO, handled in `videoHandler.js`. Once the handshake completes, media streams flow directly between peers without passing through the server.

Key Dependencies

The technology stack supporting these operations is intentionally lean. Node.js provides the runtime environment, while Express handles basic HTTP serving. Socket.IO is responsible for all real-time communication and signaling processes. WebRTC powers the direct video and audio exchange, leveraging built-in browser APIs. On the frontend, the project uses Bootstrap for responsive design and plain JavaScript for logic, avoiding heavy frameworks in favor of simplicity and transparency.

Code Quality

Despite its compact scope, the project is structured thoughtfully. The separation between chat logic and video logic is clear, and the use of configuration files demonstrates good engineering practices. The inclusion of an `ARCHITECTURE.md` file further reflects attention to documentation and maintainability. While the codebase uses plain JavaScript and static HTML

Conclusion

ChatBlink represents a well-executed exploration of real-time web communication using modern yet accessible technologies. Its implementation of both text and video communication in a clear, modular structure demonstrates a solid understanding of WebRTC, Socket.IO, and event-driven design. While the system remains lightweight, it forms a strong foundation that can be expanded into a more robust or production-grade communication platform. Its achievements lie in its simplicity, its clarity, and its ability to effectively showcase the core concepts of real-time interaction in the browser.