Real-time audio streaming, multi-channel, multi-user, walkie-talkie, DEVELOPMENT PROPOSAL

PREPARED FOR

Teton

PREPARED BY

Penda Svane

Overview

1. Implementation strategy

I chose to do research rather than an implementation. The reasons are as follows.

- I have been provided the initial scope of the problem, but not what is included in the edge cases for the solution. Nuances and details include:
    - The expected number of people using the system
    - The conditions in which the system will be placed, e.g. an intranet, global audio calling, network conditions, etc
    - Platform compatibility concerns
    - Whether it should be an in house solution or if using an existing platform or service is a viable option
    - Further discussion with the team on how the system interplays with other systems or if it is considered a standalone system.

Taking these variables into consideration, I concluded on an implementation strategy that I think was relative to the problem, the existing solutions at Teton (React Native - using web based technologies) and open source, free resources to cut down on costs for long-term subscriptions but may have high initial development cost to build and maintain.

2. System architecture

Approach

I chose not to further develop the interface, as I personally found investigating real-time data streaming, backend architecture and network configuration the most engaging and interesting. The user interface is also very dependent on the target audience, and issues such as accessibility also matter to me when developing an interface. I consider myself quite creative aesthetically, but applying relative limitations and constraints allows me to build attainable solutions.

The current solution uses a peer to peer protocol and mesh strategy, which allows for a bi-directional call between 2 clients.

tech stack

WebRTC - “is a free and open-source project providing web browsers and mobile applications with real-time communication via application programming interfaces.”

Socket.io - “It enables real-time, bi-directional communication between web clients and servers. It consists of two parts: a client-side library that runs in the browser, and a server-side library for Node.js.”

Express - “is a back end web application framework for building RESTful APIs with Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.”

Tailwindcss - “is an open source CSS framework. The main feature of this library is that, unlike other CSS frameworks like Bootstrap, it does not provide a series of predefined classes for elements such as buttons or tables.”

Additional info:

WebSockets - “WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection.”

Code

I opted for a simple and straightforward prototype. The user is greeted with a home page, where they can select a channel and are routed to the selected chatroom page.

Due to how simple I made the routing, when a client leaves the page by navigating back to the main page or by simply closing the tab in their browser, the message “bye” is broadcasted on the backend and closes the connection.

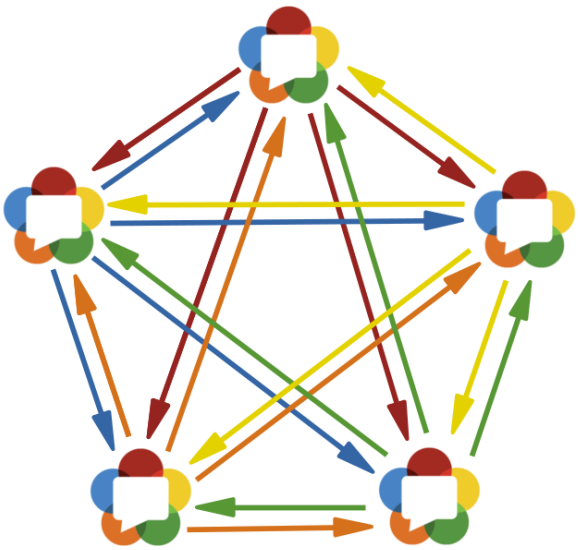
To handle multiple rooms and channels, for each client to client connection - I could have checked whether the current server is full, create or “spin up” a new socket io server when a new client joins an unoccupied room and let the next user join that. The state of the rooms and servers have to be in sync, to allow allocation of resources to be used dutifully. I wanted to do it, but perhaps next week.

Mesh topology

WebRTC Multiparty Topologies

Mesh Topology

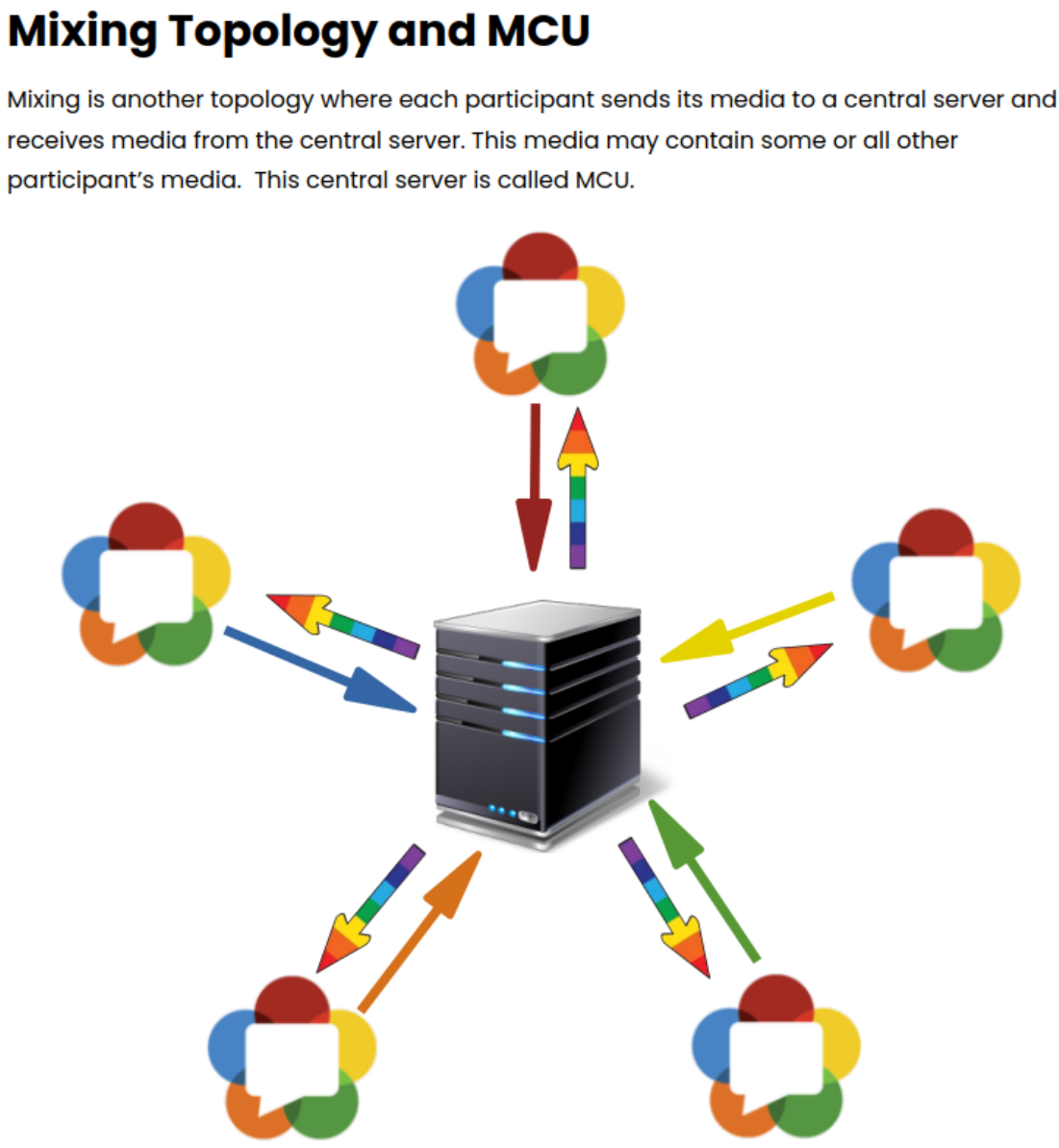
Mesh is the simplest topology for a multiparty application. In this topology, every participant sends and receives media from all other participants. We said it is the simplest because it is the most straightforward method. Moreover, there are no tricky works and a central unit such as a WebRTC server.



Mesh Topology in WebRTC

The current peer to peer solution is based on a mesh topology, which simply means that each peer or client is making a bi-directional connection to each individual client on the network / room. The issue in this, is as the number of clients join the room, the number of connections increases linearly, which means more strain on client side processing and a dependency on network reliance and client CPU speed. It also makes handling disconnects tedious and managing the room becomes less stable. Creating a multi-user room is possible but requires preparation.

Mixing topology



Mixing Topology and MCU in WebRTC

A preferred but more complex solution would be to have a mixing topology with an mcu. The mcu is essentially a centralised server, that combines the channels of each client together into potentially a single mixed audio stream, which can then be forwarded to each client - thus improving performance, as the MCU can be scaled by adding more servers rather than more connections. Although this solution improves network performance, the backend code on the mcu server becomes more complicated to manage and removes the

ability for direct peer to peer connection (unless you provide a function that checks if it is only 2 people in the limited private call, which could then be considered viable to have it a secure peer to peer channel instead).

3. Technical Obstacles

scope

Producing a multi-user, multi-server, real-time audio streaming service and application was not possible for me to complete given the time frame and limit in knowledge. Prior to knowing this outcome, I dedicated time to reading documentation and working on further understanding the technology behind getting the audio stream from one client to another.

I was able to get peer to peer client communication, but I wanted to have a solution that was a bit more robust and had the ability to combine my previous solution of channel/room creation and management. The scope was too big, so this document serves as a thought process into my considerations.

options

Media soup allows for server scalability horizontally.

https://mediasoup.org/documentation/overview/

4. Alternative solutions

Apart from creating a solution from scratch, I found a solution to build “WebRTC with NodeJS: Building a Video Chat App”, with the nice option of toggling to only audio streams to reduce costs and bandwidth loads.

I was tempted to start building this application, but due to the amount of time I spent researching - I concluded the best idea was not to, but to mention it.

https://www.metered.ca/docs/Video-Calls/JavaScript/Building-a-Group-Video%E2%80%93Calling-Application

Daily is another alternative, with solutions that work with React Native as well. Useful documentation and guides to get the application up and running.

https://docs.daily.co/guides/products/audio-only

A react-native solution using Daily APIs

https://github.com/daily-demos/party-line/tree/main/react-native

https://www.daily.co/blog/build-your-own-audio-only-clubhouse-clone-app-with-dailys-react-native-library/

8. Conclusion

I had many ideas of how to build and implement such a solution, but opted to present the simplest in my opinion and further discussion can be had on how the potential system can be built into a fullfledged solution with client options.