

From Mixing Console to the World – Enabling Remote Access to Spatial Audio

Richard Foss
Computer Science Department
Rhodes University
Makhanda, South Africa
r.foss@ru.ac.za

Abstract — A client-server immersive sound system is described that enables inputs to a mixing console to be spatialized by multiple remotely located users. Users can control the spatialization of mixing console audio sources via a client web browser user interface that displays a room with speakers. A tunnelling approach is used to transmit three-dimensional localization requests from clients to the server. Localization is implemented on the mixer by modifying the aux send levels on the mixer inputs, effectively enabling a matrix mixer. The aux send level values are determined by a rendering algorithm, running on the server. A user can select from a range of rendering algorithms. The aux buses carry the spatialized audio, and they are directed, via USB to the server. The server uses the positions of the listener's head and speakers to determine appropriate binaural intensity, delay and filter values for the incoming audio sources. Using these values, the server creates a binaural, spatialised audio stream, which it transmits to the clients via a Web Real Time Communication (WebRTC) Selective Forwarding Unit (SFU). A few approaches to remote streaming are described, together with their latencies.

Keywords—*immersive sound, matrix mixing, binaural spatialization, remote control, remote listening*

I. INTRODUCTION

This paper describes solutions to three requirements encountered in the context of an electro-acoustic music festival. The festival receives submissions from artists in various parts of the world. These electro-acoustic submissions are played back via a mixing console to multiple speakers, enabling an immersive experience for the audience. In some pieces, there will be live playing, at times using traditional African musical instruments with appropriate microphones.

There were three needs identified for the festival:

1. Composers should be able to, before the event, test the playback of their pieces from their remote locations. In particular, they should be able to adjust the spatialisation of the audio channels within their pieces, and hear the result of these adjustments as binaural playback over headphones.
2. People at remote locations should be able to hear the various music pieces, appropriately spatialised, over headphones.
3. Access should be simple, via a browser URL.

There are not the severe timing constraints required by networked music performance (NMP) [1] [2] [3], but for a composer spatialising sound, there should be close to immediate feedback.

The immersive sound system to be used for the festival has been described in [4]. It is a client-server system where the

server is a node.js [5] server and users can control spatialisation of sound sources from web browser clients. Rendering algorithms are run on the server, which is connected to the Front Of House (FOH) mixing console. These rendering algorithms determine aux bus send levels for the mixer inputs. The mixing console thus plays the role of matrix mixer, a role that is typically assigned to an external processor box in high end immersive sound systems such as L-Acoustics L-ISA [6].

In order to provide the binaural sound for remote listeners, the rendered speakers outputs, on the aux buses, are directed to the server. Here they are processed to provide a binaural replica of the loudspeaker sound and transmitted to remote listeners.

A more detailed description of this binaural processing will be given in section III. Regarding the transmission of binaural sound to remote listeners, and control data to and from remote listeners, two broad approaches were implemented and tested. These approaches included:

1. Tunnelling, where both control and audio are conveyed via a tunnel server, such as provided by ngrok [7], LocalXpose [8], and Cloudflare [9].
2. WebRTC (Web Real-Time Communication) [21], which allows for real-time peer to peer communication, and in particular a WebRTC Selective Forwarding Unit (SFU) [34]. This is a server based architecture that enables the routing of an audio stream to multiple remote clients. There have been two implementations that use the SFU concept – a Mediasoup implementation [10], and a Cloudflare SFU [11] implementation.

These two broad approaches, their various implementations, and the latencies they exhibited, will be discussed in section IV.

II. THE LOUDSPEAKER BASED IMMERSIVE SOUND SYSTEM

This section will give an overview of the loudspeaker-based immersive sound system used for live playback in the festival concert hall. A more detailed description can be found in [4].

As mentioned in the introduction, the immersive sound system is a client-server-based system, where the server is a node.js based server, and where there can be one or more clients running within web browsers on mobile or desktop devices.

Fig. 1 below is a diagram of the immersive sound system that uses speakers for the realization of spatialized sound.

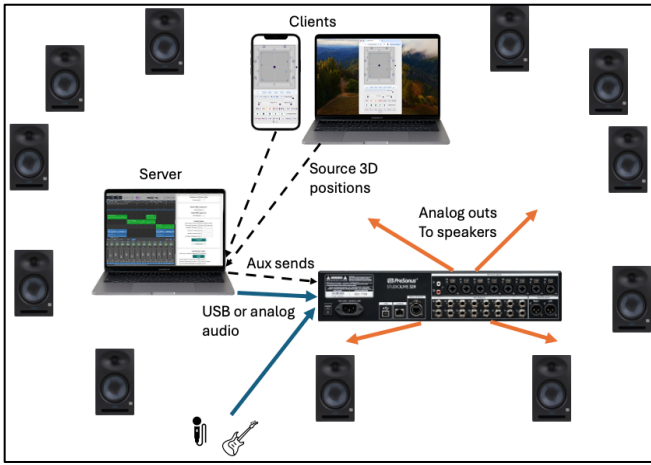


Fig. 1- Loudspeaker-based Immersive Sound System

The mixing console shown in Fig. 1 is used to mix sound sources transmitted to the mixing console, from the mix inputs to the aux buses via aux sends. The sources of sound can be USB from a DAW or multichannel wav player, or they can be analogue inputs from instruments or microphones. The aux outs on the mixing console are connected to speakers.

There can be one or more users controlling the spatialisation of audio sources from mobile or desktop devices via a web browser graphical user interface. The graphical user interface is shown in Fig. 2 below.



Fig. 2 - Client User Interface

When a client starts up, speaker positions are sent to it from a workstation server. Audio source and three dimensional coordinate data are sent from the users' devices to the workstation server. Each source can be associated with one of a number of rendering algorithms. The workstation

server will use a source's rendering algorithm to determine appropriate aux send values.

Currently the optional rendering algorithms are Distance Based Amplitude Panning [28], Vector Based Amplitude Panning [29], and Layer Based Amplitude Panning [30]. The mixing console reverb buses can be spatialised individually, or a reverb bus can be associated with a sound source and have the same localisation applied to it. This is contrasted with the Orchestra Toolkit approach, where reverberation is implemented using the Web Audio [13] and would be located on the client side.

Transport controls enable the control of a DAW running on the server, or an in-built multi-channel wav player. Source localisations can be recorded and played back, where the recording uses time code from the DAW or multi-channel wav player.

III. FROM SPATIALISED SPEAKER SOUND TO BINAURAL

This section will provide an overview of the process required to generate a binaural representation of spatialised sound transmitted to loudspeakers. A more detailed description can be found in [32].

In order to enable binaural spatialisation, the aux bus outputs containing the spatialised outputs for each speaker, are routed to the USB outputs of the mixer device. This will occur when the user clicks the "Binaural" button shown in the bottom left of Fig. 2. The USB channels are input to the computer, and from these inputs binaural samples are generated and sent to the web-based clients. The USB inputs to the computer are effectively the 'virtual speakers' from which the binaural audio is generated, as described in [31].

At server initialization time, a C++ binaural signal processing program (BinauralSpace) is started up. Using its knowledge of the speaker coordinates, the server calculates the spherical coordinates of the speakers relative to the head position (a static forward-looking position at the centre of the speakers) and these are passed to the BinauralSpace program.

Blocks of samples are received by the BinauralSpace program for each USB input channel, where each channel represents an output to a virtual speaker. The spherical coordinate values are used to calculate delays, gains, as well as high and low pass filter values for each ear. Within each channel, each sample will have a pre-computed left ear and right ear gain and filtering applied to it. Each input channel has two circular delay buffers allocated to it. The left and right ear delay values for each channel are used to determine the sample positions of the next output samples. For a particular sample interval, the left ear samples for all the channels are summed, and the right ear samples for all the channels are summed and transmitted to the server. The server, in turn, passes the binaural sample stream to the clients.

Each client, on receiving the samples places them in a circular buffer and starts up a Web Audio worklet node [33]. The worklet runs in the background, reading from the shared circular buffer and outputting to headphones. A more detailed description of these calculations can be found in [32].

A diagram of the process is shown in Fig. 3 below.

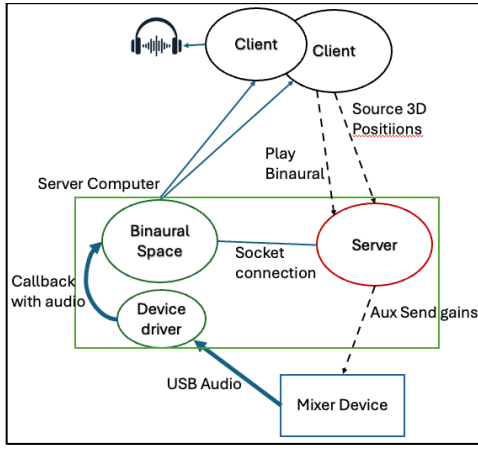


Fig. 3 - Binaural Playback Design

IV. BINAURAL SOUND AND CONTROL FOR REMOTE CLIENTS

In this section, there is a description of a number of approaches to enable the requirements laid out in the introduction. Note that in all the implementations, a remotely located user will be able to access the spatialisation capabilities and hear binaural spatialised audio by entering a URL in a web browser. In all implementations, stereo signals are being transmitted between endpoints, which as indicated in [14] is implemented by almost all existing streaming platforms. Since the current implementation enables spatialised audio to be sent from a server to client web browsers on the same local network, the logical next step was to “tunnel” the audio to remote clients.

A. Tunnelling

A number of tunnelling solutions exist, and three of them were tried – ngrok [7], LocalXpose [8], and Cloudflare [9]. In essence, a tunnelling solution exposes the local server to the public internet, bypassing firewalls and the need for Network Address Translation (NAT). An “Agent” is spawned, which is an intermediary between a local http server and a tunnel server. The agent provides a URL, which can be given to remote users. When users enter the URL in their web browsers they will be able to interact with the immersive sound system, spatialising sources and hearing the binaural result. This is shown in Fig. 4 below.

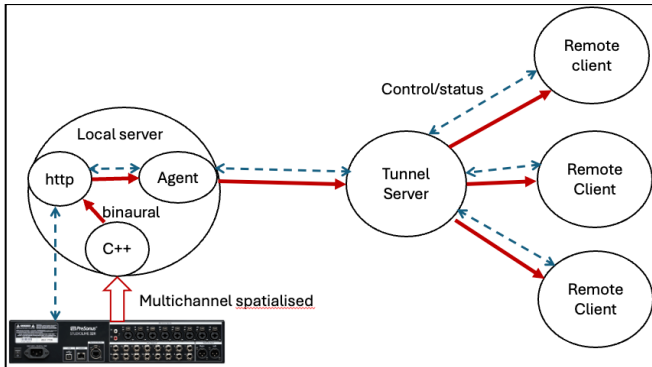


Fig. 4 - The "tunneling" configuration

Three systems were implemented, each using one of the tunnel solutions. In all three systems the local server and remote client described in sections II and III was used. There were two updates made to enable interaction via remote clients:

1. The interaction between http server and agent - for ngrok and LocalXpose the path from agent to http server is a local to local configuration. For Cloudflare, the path from agent to the http server is a Cloudflare to local configuration. The agent queries the tunnel server to find out which local port it is supposed to talk to.
2. Audio compression – in the local server, an Opus audio encoder is used to compress the audio [17], and a 2-byte sequence number and a 4-byte timestamp are added to the payload. Decoding and sequencing takes place on the client using the AudioDecoder interface of the WebCodecs API [18].

Latency tests were performed on the three systems. To perform a test, a 1kHz sine wave was played continuously. On a client, the master volume was set to zero. Then a “trigger” button was pressed, which caused a master volume change message to be sent to the local server, which was then forwarded to the mixer. On the client, the incoming samples were scanned for a “threshold” level crossing. The latency tests were intended to determine the complete cycle from user interaction to receipt of transformed audio. The latency was:

$$\text{Latency (msecs)} = \text{threshold crossing time} - \text{trigger time}$$

In these tests, the local server was on a residential fibre optic network, while the client was on a 5G network, via a cell phone “hotspot”. Throughout all the tests, there were no discernible audio breakages. Fig. 5 below shows the results of the tests.

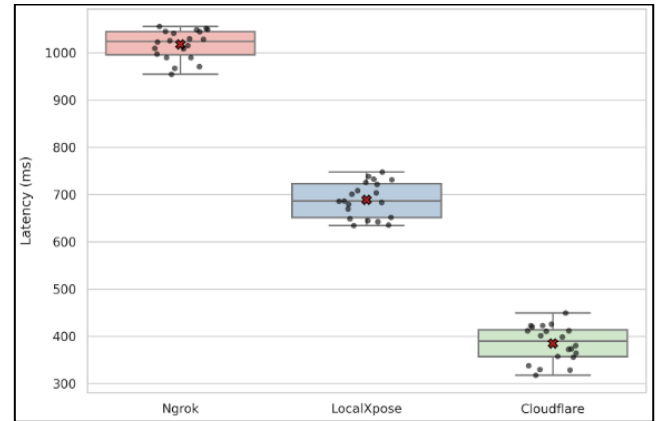


Fig. 5 - Latency Distribution across Tunnel Implementations

The Cloudflare tunnel with its relatively low latency and simple implementation provides a solution to the first and third of the stated needs, namely:

1. “a composer should be able to adjust the spatialisation of the audio channels within their pieces, and hear the result of these adjustments as binaural playback over headphones”, and
2. “access should be simple, via a browser URL”.

However, it does not adequately fulfil the second need, namely to enable multiple remote listeners. The Opus encoder removes bandwidth constraints. However during playback the server loops through every connected remote client and sends

a copy of the binaural audio packet to each one. So the server has to copy and push the packet to each remote client. With 48,000 samples per second and 960 samples per packet, 50 packets are sent per second. For 10 users, there are 500 packets per second being transmitted to the agent. The server is single threaded, and if one client experiences a momentary wireless fade, TCP pauses the entire transmission queue for that client. A 100 millisecond network break delays 5 sequential audio packets with a consequent audible click. Remote audio transmission using TCP is described in [16], and 3D control information is provided via head tracking. However, it is not clear how many remote users there are.

Given this limitation, a WebRTC (Web Real Time Communication) [21] solution was explored. The use of WebRTC is described in [24] where the authors describe the construction of a distributed reverberation chamber for concert applications. This does have some similarities to the given requirements, in that audio is distributed from a concert to up to three remote destinations, the reverberation chambers. To increase the number of destinations and enable a large number (>100) remote listeners, a WebRTC Selective Forwarding Unit (SFU) architecture was deployed [34]. With this architecture, the local server transmits a single binaural audio stream as encoded RTP (Real-time Transport Protocol) packets to the SFU, which resides on a remote server. The SFU does not perform packet copies for each client, but creates headers for each client. The Network Interface Card (NIC) uses Direct Memory Access (DMA) to access the header and the common payload, wraps them in a UDP compliant frame for each client, and transmits them.

B. Virtual Private Server and Selective Forwarding Unit

The first SFU-based implementation uses the mediasoup [10] node.js module in a Hetzner Virtual Private Server (VPS) [19] and the mediasoup library on the client (browser) side. To enable the transmission of RTP packets from the local server, and subsequent “fan-out” to multiple clients, the following steps are performed:

1. When started, the VPS node.js application creates a mediasoup worker and router.
2. The local server establishes a connection to the VPS and asks the VPS to register a producer.
3. The mediasoup router opens a UDP port and informs the local server of the port details.
4. The local server creates an RTP sender, and informs the VPS that it is ready to “produce” audio. The local server receives binaural spatialised audio from the C++ “BinauralSpace” program described in section III, but now passes the encoded audio to the RTP sender.
5. The mediasoup worker in the VPS creates a Producer object which listens for incoming UDP packets

To enable the streaming of audio to a remote client:

1. The client asks the VPS server to provide the parameters necessary to enable Web Real Time Communication (WebRTC) [21]. Mediasoup enables the creation of a UDP port and sends port and security information to the client browser

2. On receiving this information, the mediasoup client library creates a WebRTC connection, using the port and security data. It then sends a “consume” message to the VPS, thereby indicating its desire to receive audio.
3. The VPS responds with parameters, in particular a producer ID.
4. Following this a necessary security protocol is followed. The client gets browser Datagram Transport Layer Security (dtls) parameters from the mediasoup library, and sends them to the VPS. Once the VPS responds positively, there is a cryptographic handshake between client and VPS and the browser’s WebRTC application starts receiving audio.

Tunnelling, as described in sub-section A is used for the transmission of control data between clients and local server. An overview of the configuration and the interactions between local server, VPS, and remote clients is shown in Fig. 6 below.

Latency is determined by creating a latency processor audio worklet node, that is inserted between the media stream source and destination. A 1 kHz sine wave file is played and the master volume is pulled right down. When a trigger button on the client is clicked, the time is taken, and the aux bus volume on which the sine wave is being transmitted is raised to 0dB. The latency processor notes the time at which a threshold level is passed and uses this, together with the trigger time, to determine the latency. As for the tunnel latency tests, the local server was on a fibre residential network and the client was on a 5G network, via a cell phone “hotspot”. Twenty latency measurements were taken and found to be higher than expected, with a mean of 619 milliseconds and a standard deviation of 29.

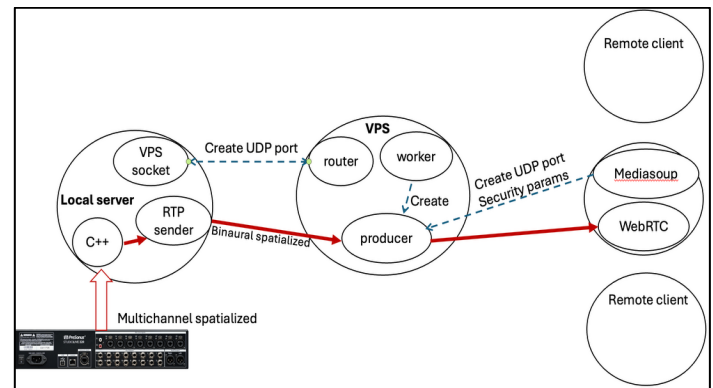


Fig. 6 - The VPS and SFU Configuration

To reduce the latency, and still have the SFU capability for multiple remote listeners, a configuration using a more distributed structure, rather than a single, remote VPS server was implemented. Known as a global edge network, the worker code that the local server needs to interact with runs on a physically close node. The global edge network used was the Cloudflare network [20].

C. Global Edge Network and Selective Forwarding Unit

A first attempt was made to send the audio data over a WebRTC Data Channel [23], using the node.js compatible node-datachannel module [12]. The node-datachannel module can create audio media tracks, but doesn’t enable the complexity of WebRTC audio track transmission as required by the cloudflare SFU. However, the local node.js server runs

an Electron framework [22], which enables a web browser-based user interface capability, known as the Renderer. The Renderer has a WebRTC stack that can be used to stream to the cloudflare SFU using WebRTC. To enable this streaming, the following steps are performed:

1. The Renderer loads audio “worklet” code that runs in a separate thread. The worklet reads audio from a circular buffer and streams it via WebRTC to the SFU, once the connection to the SFU has been created. The circular buffer is filled with binaural spatialised audio from the C++ “BinauralSpace” program described in section III.
2. To create the connection, the Renderer passes information about the streaming session that it wants to create and a track identifier to the worker, and asks the worker to create such a session. The worker requests a streaming session between the Renderer and the SFU, passing the Renderer’s session description and track data.
3. The SFU responds with its own session and track data, which the Renderer receives, and which it passes on to the local server.

To enable streaming from the SFU to remote clients, the following steps are performed:

1. A new client will access the local server via a cloudflare tunnel. When the local server receives the new client request it will pass the client the SFU session id and track name. When a user clicks the “binaural” button on the client UI to start streaming, the local server will pass the session id and track name to the worker.
2. The worker requests the SFU to provide the network and security related parameters necessary to stream to the client. The worker passes these on to the client. Once received, the client creates a new WebRTC connection with the SFU and the browser’s WebRTC application starts receiving audio.

Fig. 7 below shows the Cloudflare SFU configuration, where the worklet within the renderer transmits audio to the Cloudflare SFU, which in turn transmits to the remote client browsers.

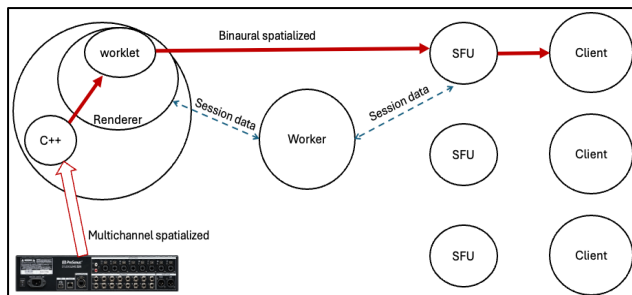


Fig. 7 - Cloudflare SFU Configuration

As with the mediasoup SFU implementation, latency is determined by creating a latency processor audio worklet node, that is inserted between the media stream source and destination. As for the tunnel and mediasoup latency tests, the local server was on a fibre residential network and the client was on a 5G network, via a cell phone “hotspot”. Fig. 8 below shows the results of the SFU implementation’s latency tests.

For the Mediasoup SFU:

Mean latency = 619.9 ms

Standard Deviation = 29.62 ms

For the Cloudflare SFU:

Mean latency = 294.1 ms

Standard Deviation = 25.93 ms

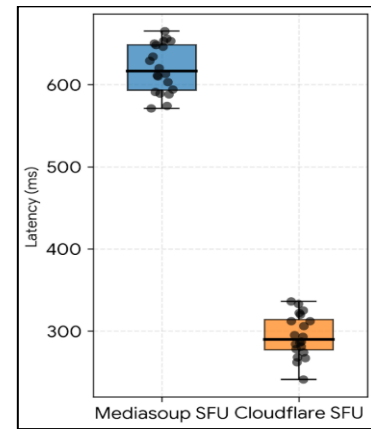


Fig. 8 - Mediasoup and Cloudflare SFU latencies

There are a number of factors contributing to the latency:

1. Transmission of trigger request from client to server
2. Buffering of mixer level request within the mixer
3. Local server USB input buffer
4. Spatialisation computation by “BinauralSpace”
5. WebRTC structuring for Renderer Transmission
6. Local server to SFU and SFU to client transmission
7. WebRTC jitter buffer within the client browser

In the Mediasoup configuration, all media packets are forced to route through a single physical data center regardless of user proximity, adding significant public internet transmission overhead. This adds substantially to the latency of point 6 above. Conversely, the Cloudflare architecture intercepts the media stream from the local server at the closest physical edge node, routing it across a private global fibre backbone and thereby enabling minimal latency. Regarding point 2 above, the mixing console used in the tests was a Presonus StudioLive 64s [27] and as described in [4], messages to the mixing console are queued for 50 milliseconds before being released for processing. Furthermore, a large contributor to the latency is point 7. Because of the variable packet arrival times, the client browser needs to hold 6-9 packets in memory, where each packet is 480 stereo samples (960 total samples) in length (10ms). This results in a buffer delay of 60ms to 90ms. Given the transmission variance on a cellular network, the buffer will be pushed towards a larger packet size.

Regarding the number of clients, the Cloudflare SFU-based system was tested in lab conditions, where the server resided on a residential network, and 17 clients were created within a university lab, each continuously streaming binaural audio from the server. Unlike traditional SFUs (e.g. Mediasoup) running on a single virtual machine, Cloudflare routes media tracks across their global Anycast network. Scaling from 17 to 100 subscribers does not increase load on a single server node; instead, fan-out happens near each

client's local POP (Point of Presence) [37]. This is corroborated by Cloudflare Support, who indicate

V. CONCLUSION

An immersive sound system has been described that, as the title suggests, enables the transmission of spatialised binaural audio to a large number of remote client workstations, as well as control over the spatialisation of sound sources entering a mixing console. This was made possible by the client-server architecture of the system, where the server incorporates an Electron Renderer. The Renderer allows for a WebRTC connection to a WebRTC SFU, while a tunnel server enables spatialisation control messaging. This mix of control and audio feedback enables remote spatialisation control by more than one user, and thus cooperative creativity.

Since the system was designed with a web-based user interface in mind, remote user access to remote spatialisation is via a URL that is generated by the server. This is contrasted with the well-known SPAT Revolution program, that does provide a remote capability [36], but which requires third party application installation for both remote control and remote streaming.

While there are not the low latency levels described in [15], there is also not such a strict networking requirement for the intended use case. Commercial applications such as JackTrip [25] and SonoBus [26] provide low latency peer to peer audio streaming, but they don't allow for spatialisation control and streaming for a large number of remote listeners. The use of WebRTC has been particularly advantageous. As mentioned in [35], WebRTC was designed for use cases such as the concert streaming application described in the introduction, rather than "truly networked music experiences".

The goals for the festival system, laid out in the introduction to the paper have been achieved - Composers should be able to control the spatialisation of their pieces from their remote locations and the resulting binaural audio should be accessible to multiple listeners, also at remote locations, while access should be simple, via a browser URL. The novelty of the system lies in this capability, spatialization control is enabled over any inputs to a mixing console, via a web browser located anywhere in the world, and the spatialized audio can be received from multiple locations simultaneously. While the system is yet to be tried in a festival setting, it has been tested in a laboratory setting, where Music Technology students, on laboratory workstations, were required to control and listen to spatialized sound derived from a server on a remote residential network. The latency was found to be acceptable. Technical correspondence with Cloudflare Support confirms that their Realtime SFU architecture is capable of supporting large-scale events with thousands of participants [38].

Moving forward, ways to reduce the latency associated with the enumerated seven factors will be investigated.

REFERENCES

- [1] M. Bosi, A. Servetti, C. Chafe, and C. Rottondi, "Experiencing Remote Classical Music Performance Over Long Distance: A JackTrip Concert Between Two Continents During the Pandemic," *J. Audio Eng. Soc.*, vol. 69, no. 12, pp. 934–945, December 2021.
- [2] M. Sacchetto, P. Gastaldi, C. Chafe, C. Rottondi, and A. Servetti, "Web-Based Networked Music Performances via WebRTC: A Low Latency PCM Audio Solution" *J. Audio Eng. Soc.*, Vol. 70, No. 11, pp. 926–937, November 2022.
- [3] P. Cairns, "Immersive Audio and Network Music Performance: Practical Applications and User Testing," Ph.D. dissertation, School of Arts and Creative Technologies, University of York, 2025.
- [4] R. Foss, "Employing the Processing Capability of Mixing Consoles and Audio Interfaces to Spatialize Sound" presented at the AES 2023 International Conference on Spatial and Immersive Audio, Huddersfield, UK, August 2023.
- [5] OpenJS Foundation, "Node.js (v26.1.0)" [Online]. Available: <https://nodejs.org/>
- [6] L-Acoustics, "L-ISA Processor II", [Online]. Available: www.l-acoustics.com/documentation
- [7] "ngrok tunneling server" [Online]. Available: <https://ngrok.com/>
- [8] "LocalXpose tunneling server" [Online]. Available: <https://localxpose.io/>
- [9] "Cloudflare tunnel" [Online]. Available: <https://developers.cloudflare.com/tunnel/>
- [10] Mediasoup, "'Communication Between Client and Server'" [Online]. Available: <https://mediasoup.org/documentation/v3/communication-between-client-and-server/#signaling-and-peers>
- [11] Cloudflare, "Realtime SFU" [Online]. Available: <https://developers.cloudflare.com/realtime/sfu/>
- [12] "node-datachannel" [Online]. Available: <https://github.com/murat-dogan/node-datachannel>
- [13] D. Dziwis, H. V. Coler and C. Porschmann, "Orchestra: A Toolbox for Live Music Performances in a Web-Based Metaverse" *J. Audio Eng. Soc.*, vol. 71, no. 11, pp. 802–812, November 2023.
- [14] J. Kares and V. Larcher, "Streaming Immersive Audio Content," presented at the AES Conference on Audio for Virtual and Augmented Reality, Los Angeles, CA, USA, September 2016.
- [15] S. Giacomelli, C. Centofanti, J. Santos, M. Galbiati, T. Salvi, F. Graziosi, and C. Rinaldi, "Remote Immersive Audio Production: State of the Art Implementation, Challenges, and Improvements," in *Proc. 2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, pp. 1–10.
- [16] R. Diaz and T. Koch, "Live Panorama and 3D Audio Streaming to Mobile VR," presented at the AES Conference on Headphone Technology, Aalborg, Denmark, August, 2016.
- [17] "prism-media v1.3" [Online]. Available: <https://npm.io/package/prism-media>
- [18] Mozilla, "WebCodecs AudioDecoder API" [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/AudioDecoder>
- [19] Hetzner, "Hetzner Virtual Private Server" [Online]. Available: <https://www.hetzner.com/cloud>
- [20] Cloudflare, "Cloudflare Global Network" [Online]. Available: <https://www.cloudflare.com/network/>
- [21] WebRTC, "Real-Time Communication for the Web" [Online]. Available: <https://webrtc.org/>
- [22] Electron, "The Electron Framework" [Online]. Available: <https://www.electronjs.org/>
- [23] Mozilla, "Using WebRTC Data Channels", [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Using_data_channels
- [24] A. Franklin, R. Mitic, D. Hedin, R. Lindell, H. Frisk, "Development of a Web-Based Real-Time Distributed Reverberation Chamber for Live Concert Applications", *J. Audio Eng. Soc.*, vol. 73, no. 12, December 2025.
- [25] JackTrip Labs, "JackTrip" [Online]. Available: <https://www.jacktrip.com/>
- [26] SonoBus, "SonoBus - High Quality Network Audio Streaming" [Online]. Available: <https://www.sonobus.net/>
- [27] Presonus, "Digital Console Mixers, StudioLive Series III", [Online]. Available: <https://www.presonus.com/products/Mixing-Systems>
- [28] T. Lossius, P. Baltazar, and T. de la Hogue, "DBAP - Distance-Based Amplitude Panning," in *International Computer Music Conference (ICMC)*. Montreal, Canada, 2009.

- [29] V. Pulkki, "Virtual Sound Source Positioning Using Vector Base Amplitude Panning," *J. Audio Eng. Soc.*, vol. 45, no. 6, pp. 456–466, June 1997.
- [30] I. Bukvic, "3D Time-Based Aural Data Representation Using D⁴ Library's Layer Based Amplitude Panning Algorithm", The 22nd International Conference on Auditory Display, Canberra, Australia, July 2016.
- [31] C.Pike, F.Melchior, A.Tew, "Descriptive Analysis of Binaural Rendering with Virtual Loudspeakers Using a Rate-All-That-Apply Approach", AES Conference on Headphone Technology, Aalborg, Denmark, August 2016
- [32] R. Foss, "A Binaural Capability to Mirror a Loudspeaker Configuration", Published in: 2025 Immersive and 3D Audio: from Architecture to Automotive (I3DA), Bologna, Italy, September 2025.
- [33] Mozilla, "Web Audio API" [Online] https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
- [34] J. Bordane, "WebRTC SFU: the complete guide" [Online]. Available: <https://www.metered.ca/blog/webrtc-sfu-the-complete-guide/>
- [35] A. Boem, M. Tomasetti, and L. Turchet, "Harmonizing the Musical Metaverse: Unveiling Needs, Tools, and Challenges from Experts' Point of View," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2024)*, Utrecht, The Netherlands, 2024, pp. 206–214.
- [36] Flux, "SPAT Revolution Remote" [Online]. Available: <https://www.flux.audio/project/spat-revolution-remote/>
- [37] Cloudflare, "Introduction - Cloudflare Realtime SFU architecture" [Online]. Available: <https://developers.cloudflare.com/realtime/sfu/introduction/>
- [38] Cloudflare Support, personal communication, Email Subject: "02330975 - Cloudflare Realtime SFU listener limit per stream," Sept. 21, 2026.