A. Franklin, R. Mitic, D. Hedin, R. Lindell, and H. Frisk,
"Development of a Web-Based Real-Time Distributed Reverberation Chamber for Live Concert Applications,"
*J. Audio Eng. Soc.*, vol. 73, no. 12, pp. 872–881 (2025 Dec.).
https://doi.org/10.17743/jaes.2022.0245.



# Development of a Web-Based Real-Time Distributed Reverberation Chamber for Live Concert Applications

**AUSTIN FRANKLIN,**[1, 2, *] **RICHARD MITIC,**[1] *AES Student Member*, **DANIEL HEDIN,**[1, 3]
(austin.franklin@mdu.se) (richard.mitic@mdu.se) (daniel.hedin@mdu.se)
**RIKARD LINDELL,**[1, 4] **AND HENRIK FRISK**[2]
(rli@du.se) (henrik.frisk@kmh.se)

[1]*School of Innovation, Design, and Technology, Mälardalen University, Västerås, Sweden*
[2]*Department of Composition, Conducting, and Music Theory, The Royal College of Music, Stockholm, Sweden*
[3]*Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden*
[4]*Dalarna Audiovisual Academy (DAVA), Dalarna University, Falun, Sweden*

This paper documents the challenges encountered during the development of a mobile web-based, real-time distributed reverberation chamber for a live concert setting. Initial efforts focused on leveraging existing open-source and commercial solutions, but these proved inadequate due to limitations in network compatibility, audio routing, and mobile device support. WebRTC emerged as the most viable option, offering peer-to-peer communication and firewall traversal capabilities. However, significant obstacles persisted, including network reliability, audio hardware/software ecosystem limitations, and debugging complexity. The project highlights the fragmented nature of real-time audio streaming technologies and the need for more transparent, reliable, and well-documented tools. Despite these challenges, the authors developed a functional web application prototype using WebRTC, Web Audio API, and custom Session Description Protocol modifications, offering insights into the complexities of deploying real-time audio systems over mobile networks. The paper concludes with reflections on speculative approaches, such as decentralized audio routing and native app development, and calls for improved documentation and standardized debugging tools to support future innovations.

## 0 INTRODUCTION

The authors began a project in February 2024 to create a simple proof-of-concept: distributing a live reverberation chamber as a stereo audio signal over a mobile network from a concert hall to up to three remote locations and back to the original site, where an audience would experience the distributed performance alongside a live music event. The end goal was to create a merged reality for the audience and to explore their experience of the site-specific aural qualities of the remote locations merged with those of the concert hall. The prototype, named Auxtrument, was officially used in concert in November 2024 following a successful demonstration at the 2024 International Conference on Quality of Multimedia Experience [1].

It was anticipated that this project would be a relatively straightforward endeavor, given the established use of network-based tools for collaborative performance. Available open-source and commercial solutions were initially explored, but each proved inadequate for various reasons. The authors' efforts extended to experimenting with various protocols and network stacks, leading them to settle on Web Real-Time Communications (WebRTC) as the most viable option. WebRTC is marketed as a reliable, standardized solution that appeared to address the core challenges faced: seamless real-time communication across devices, browsers, and networks [2]. However, even after adopting WebRTC, significant obstacles persisted, particularly in three key areas: (1) network compatibility and stability, (2) limitations within the audio hardware and software ecosystem, and (3) debugging complexity and software fragmentation. Many of these challenges remained unresolved throughout the project, contradicting WebRTC's reputation as a mature and reliable technology.

Boem et al. state that platforms based on WebRTC were not originally designed for networked music experiences but rather as "environments where a single data stream...

*To whom correspondence should be addressed, email: austin.franklin@mdu.se

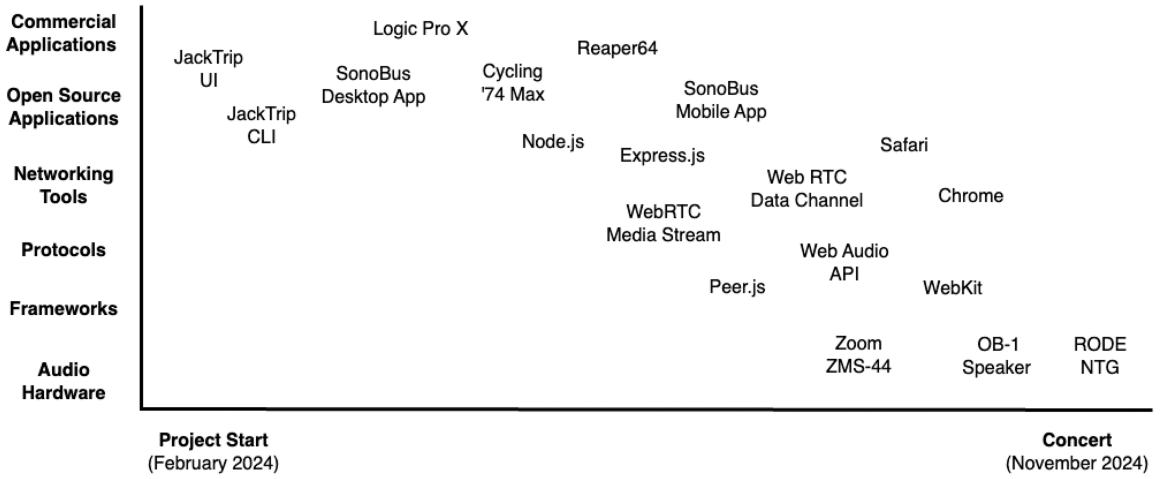



Fig. 1. The timeline of development for the Auxtrument.

is broadcast to all users simultaneously" [3]. Additionally, they suggest that overcoming these limitations using custom servers and protocols, while possible, remains challenging. These issues raise critical questions about the reliability, scalability, and practical limitations of these tools—where things like frequent version updates and deprecations can be difficult to track across multiple web stacks and may render software components, or even entire applications, unusable. While the Auxtrument application provided a workable solution, the development and testing process exposed significant gaps in documentation, unpredictable "black box" network behaviors, and difficulties in achieving consistent performance across different devices and environments.

Fig. 1 shows the development. Each item represents a tool, protocol, or software/hardware component laid out in the order in which it was experimented with. Given the aims and timeline of the project, a proper state-of-the-art investigation into the full gamut of mobile web audio streaming technologies was not conducted. Given the reputation of WebRTC as tried-and-tested software, and the saturation of open-source and commercially available audio streaming solutions, it was not reasonable to assume that many of the problems encountered were not already solved. Instead, the journey is documented, highlighting motivations and processes, aiming to shed light on the broader complexities of deploying real-time audio systems over mobile networks using exploratory and practice-based methods.

## 1 METHODOLOGY

Exploratory research, as described by Stebbins, emphasizes open-ended investigation into uncharted territories, allowing for documenting and reflecting on unexpected outcomes during the process [4]. Practice-based research methods, in which creative practice serves as both the process and outcome of research, are also applied, generating new insights through artistic exploration and critical reflection [5, 6]. To distinguish this work as practice-based, the methodology was structured to align with the criteria proposed by Candy [5]:

1. Define the objectives and research question(s).
2. Document the tools, setup, and actions used.
3. Analyze the results, noting the challenges.

This research began with an artistic question: "How can the real-time distribution of site-specific acoustic reverberation between a concert hall and multiple remote locations create a merged reality audience experience?" However, as development progressed, a series of challenges was encountered—including network reliability, limitations in existing audio ecosystems, and debugging complexities—and the focus necessarily shifted. In response, the research question evolved: "What are the practical and technical constraints of implementing a mobile, real-time, distributed reverberation system for live performance using existing audio streaming technologies?"

To investigate this question, three objectives were established for achieving a high-fidelity user experience: (1) the audio signals must be high-resolution to preserve the inherent quality of the transmitted sounds and accurately relay spatial properties of the acoustic environments, (2) the system should be easy to set up and use and work over different types of networks, and (3) to support use in any location, it must be compatible with mobile devices. This framing allowed for critically assessing available tools and infrastructures as active conditions that shape the artistic possibilities of the original research question. In writing this paper, the authors set out to retroactively analyze and better understand the causes behind several of these challenges. The findings are documented here in the belief that they contribute to a broader understanding of how current technologies shape the possibilities—and limits—of mobile networked music performance, while also illuminating the assumptions embedded in their design and use.

## 2 SURVEY OF TECHNOLOGIES

In evaluating potential software solutions, hands-on tests were conducted with a variety of audio applications, focusing primarily on their networking capabilities and suit-

ability for real-time audio transmission of multiple channels and audio routing capabilities across different platforms and clients. Given the goal of integrating mobile devices with the Auxtrument, applications with mobile support were also explored. However, mixer apps on mobile phones were not tested, as these typically require custom plugins for advanced routing and signal processing—something that mobile operating systems restrict. As the authors are primarily composers, sound artists, and researchers, they explored solutions well-known in their respective field(s), and the list of software is by no means exhaustive. These findings guided the decision to further explore alternative networking approaches for real-time audio streaming.

### 2.1 Existing Audio Applications

Although the aim was to have Auxtrument work on mobile networks, initial tests involved using JackTrip [7] and SonoBus [8] on MacBook laptops. Tests started with laptops and Wi-Fi connections, anticipating this would be a quick and practical first step in prototyping while allowing for evaluating the software. Although JackTrip's user interface (UI) was straightforward, it lacked the flexibility needed for audio routing between clients, sending the same audio stream to all participants without the ability to control volume or mute channels on the host side. It also did not have an implementation for mobile phones. These limitations made it unsuitable for the authors' purposes. SonoBus, on the other hand, addressed this issue by offering a mobile app and multichannel input/output capabilities, allowing users to specify channel destinations.

The authors attempted to connect using SonoBus over a mobile hotspot using the Swedish Tele2 service provider, with one laptop connected via eduroam. Eduroam is a global network roaming service for students and employees in academia [9]. Its authentication requirements are standardized across all networks, but individual providers can and often do add additional security features. The setup using SonoBus with Tele2 and eduroam proved unsuccessful. In contrast, connections between two laptops granted eduroam certificates worked, and even certified mobile devices connected to eduroam were able to stream audio via the SonoBus iOS app without issues.

These initial findings led the authors to suspect that the security features implemented at Mälardalen University and The Royal College of Music in Stockholm, where eduroam networks are accessible, were blocking peer-to-peer (P2P) connections from external networks. A major obstacle was SonoBus's lack of options to specify ports or other network settings, making it ineffective when dealing with firewall restrictions. This posed a significant obstacle, as eduroam was the primary testing network, and there was no reliable way to determine in advance whether an external network would permit a connection.

Digital audio workstations are powerful tools that feature a wide range of audio plugins. Reaper [10] and Logic [11] can host Apple's AUNetSend and AUNetReceive plugins, which are used to send audio between applications, hardware instruments, and more on a Mac [12]. However, these plugins use zero-configuration networking through Bonjour [13] and multicast Domain Name System service records to send audio over a local area network only. Other well-established live audio tools such as Cycling '74 Max have networking capabilities, limited primarily to control data via User Datagram Protocol and Open Sound Control protocols [14]. One workaround for audio streaming is to convert audio vectors to jitter matrices and send the video stream over the network. However, Max is not compatible with iOS.

Following the completion of the prototype, the search of existing audio streaming applications was broadened. The Virtual Rehearsal Room and a list of alternatives [15] were found. Virtual Rehearsal Room is based on Audio over Open Sound Control, the underlying protocol behind SonoBus, using Pure Data as a plug-in for the Mozilla browser [16]. These alternatives—including LoLa, NinJam, and Soundjack—reflect a range of design priorities and technical trade-offs. LoLa, for example, requires dedicated hardware and a stable 1-Gbps connection, typically only available on academic networks. While Internet service providers (ISPs) advertise such speeds, real-world performance varies. NinJam, bundled with Reaper, addresses latency by aligning performers to musical measures, sometimes increasing latency by one or more bars. Soundjack uses P2P networking but requires User Datagram Protocol port 50050 to be forwarded—an issue on networks like eduroam, where port forwarding is often blocked for security reasons.

What became clear through this review was that no single tool fit these needs perfectly. SonoBus remains the closest to meeting the criteria due to its cross-platform support and intuitive interface. Yet the inability to manually configure network ports or tunnel through more restrictive firewalls ultimately limited its utility in real-world settings. As a final thought, the process of evaluating these alternatives confirmed that while the existing ecosystem of remote audio tools is mature in some respects, it remains fragmented, with little attention paid to mobile-first, multi-endpoint collaboration under restrictive network conditions.

### 2.2 Existing Web Technologies

To create a realistic reverberation chamber effect, minimizing network latency was essential. Although a strict target was not set, the goal was to reduce the delay below the threshold of the precedence effect of approximately 40 ms for complex sounds such as speech and music [17]. Beyond this threshold, the delayed sound is perceived as a distinct echo rather than a single auditory event. Given this requirement, it was determined that P2P connections would be the most suitable approach.

Existing technologies and protocols were explored. For instance, an implementation was made that relies on WebSockets through Node.js by setting up a remote server and having clients connect to a central computer. The as-

sumption was that WebSockets, being a well-established technology, would provide a reliable foundation for audio transmission. However, a significant challenge was quickly encountered—WebSockets by themselves were unable to reliably connect two devices on disparate networks. Given these limitations, it was eventually decided that building a mobile web app using WebRTC would be the most feasible approach due to its native P2P and firewall traversal capabilities and broad browser support.

WebRTC relies on the clients negotiating a suitable way to connect to each other. The process of negotiation, known as signaling, takes place outside the WebRTC framework (e.g., using a custom-built signaling server based on WebSockets). The first step exchanges session descriptions in the form of a Session Description Protocol (SDP) offer that describes the type of connection (if it is video or audio), the codecs and codec parameters to be used, and more. The second step of signaling uses Interactive Connectivity Establishment (ICE) to establish a way for the peers to stream to each other.

ICE looks for the lowest-latency connection by trying to (1) establish a P2P connection using a Session Traversal Utilities for NAT (STUN) or Traversal Using Relays around NAT (TURN) server, (2) establish a direct Transmission Control Protocol (TCP) connection via the HTTP and HTTPS ports, and (3) fall back on an indirect connection via a TURN server. This means that even in cases where network address translation (NAT) prohibits a P2P connection using STUN, the TURN relay should act as a failsafe that permits the connection, albeit with increased overall latency, provided that it is explicitly configured to do so.

## 3 THE AUXTRUMENT AND THE NETWORK

From an implementation perspective, the final version of the Auxtrument is a web application based on WebRTC that is deployed using Express.js [18] running on top of Node.js [19]. The Peer.js library is used for establishing the connection [20]. Peer.js is a free library and service intended to provide easy-to-use WebRTC connections. On the client side, it provides a library that wraps WebRTC, allowing clients to establish connections using only an ID. This requires the use of a Peer.js signaling server that uses the IDs to automate the connection process. Peer.js provides its own signaling servers and the source code for the server to enable projects to host their own applications.

This stack helps streamline and simplify the number of actions a user must perform to establish a connection. Users simply navigate to the Uniform Resource Locator and select whether they are the central hub or location peers. The signaling and P2P initiation is automatically handled in the background, and the audio input and output devices are specified by each client in the UI prior to establishing a connection. The concert-ready version of the Auxtrument is limited to three remote clients and one central hub, but nothing prevents further expansion into more complex configurations.

### 3.1 Network Challenges

Due to the limited number of IPv4 addresses, most ISPs rely on some form of NAT or carrier-grade network address translators (CGNAT). This allows multiple clients to share the same external Internet Protocol (IP) address, thus conserving the address space. NAT and CGNAT typically prevent connections to clients to be initiated from the outside, which causes problems if two peers situated behind NAT or CGNAT want to establish P2P connections. Similar situations may occur when firewalls are used to protect local networks. The first priority of the ICE protocol is to find ways to establish P2P connections in the presence of NAT, CGNAT, or firewalls (e.g., using “hole punching”). Somewhat simplified, hole punching is based on indirectly manipulating the state of the firewall or NAT by initiating communication from the inside of a network in such a way that the initiator is exposed on the external network [21].

Experiments with the Auxtrument show the complexity of the network setup and identifying and mediating issues. Tests with Tele2 and Tre for example, some of Sweden’s largest mobile service providers, did not work with this WebRTC implementation whatsoever using STUN. Because of the fact that most of the testing occurred between eduroam networks, where the authors are based, and other ISPs, it was not immediately apparent that the mobile network itself, or combination of ISP and eduroam networks together, was the problem over firewall restrictions when used with laptops.

This led to tests with other mobile network providers in Sweden, primarily by visiting service provider stores, setting up some basic equipment, and running the Auxtrument using borrowed Subscriber Identity Modules. Based on these tests, it was discovered that Telia worked perfectly in all the situations where other providers previously had not. The frustration with mobile network providers is that there is no way to know exactly why Telia works and the others do not, in terms of how specifically each CGNAT is configured, and what penetration methods might be successful prior to experimentation. To complicate the matter further, the authors were able to get a connection working behind CGNAT, but via a fiber-based ISP.

The authors experimented with exploring the networks using traceroute, but the results were inconclusive. While there are other tools for mapping the architecture of a network, they were not pursued due to the complexity of the setup, in particular, the interaction between the IP layer and mobile network that, among other things, allows for roaming between carriers. As Kanaris and Pouwelse note, there are many different network configurations and NAT architectures that make penetration techniques difficult. However, they document an issue where both peers behind the same CGNAT cannot establish a connection using STUN since the server is outside of the intranet and only sees the “middle” network [22]. This explains, at least in part, some of the issues when testing with Tele2 mobile networks.

### 3.2 Webrtc Media Stream

WebRTC presented challenges with minimizing latency using its adaptive jitter buffer. The buffer automatically resizes when latency exceeds its capacity and does not reset unless the web page is refreshed. Setting the jitter buffer is "best effort," meaning that WebRTC may not honor the request and prioritize its own settings in cases where there are network interruptions. There are other not-so-well-known methods for minimizing latency, such as implementing a playout delay—a Real-time Transport Protocol header extension, which provides the sender's intent to the receiver on how quickly a frame needs to be rendered [23]. Community forums and bug reports show mixed reviews with playout delay implementation, with some suggesting the feature does not work with Chrome's Javascript API layer at all. This mirrors the authors' experience using this feature in Chrome.

### 3.3 WebRTC Data Channel

Using the WebRTC Data Channel was explored to overcome latency issues and have explicit control over the jitter buffer. By sending segments of audio data as bytes via this channel, the authors hoped to piggyback on its low-latency performance and then create an audio channel for playback on the other end. Initially, the latency was acceptable for this use case, between 10 and 40 ms; however, it would unexpectedly increase to around 20–30 s for brief moments of time. When this occurred, the audio was significantly affected, with clear indications of data loss. The authors were initially astonished: where do those packets live for such a long time? While the issue has been unable to be reproduced in a controlled manner, it seems to be easier to trigger on unreliable networks (i.e., mobile networks).

The authors hypothesize that it is a combination of network quality of service and WebRTCs use of Stream Control Transmission Protocol for reliable data channel messaging [24]. Specifically, when a transmitted message exceeds the maximum transmission unit of the network, it is segmented into smaller portions. Variable latency in each segment can be due to scheduling messages in the network stack, congestion, or differing processing times between network nodes, and this latency can easily compound. When packets arrive out of order, additional buffering, reordering mechanisms, or even retransmission may be needed.

Even though the channel can be configured to be "unordered" to reduce latency, lost packages still need to be retransmitted to preserve audio quality. It could be the case that high network congestion causes frequent retransmission, further exacerbating the congestion and data loss. It is assumed that quality of service plays a part in this process since WebRTCs media stream does not suffer from the same issue. Furthermore, there are unknowns about the ways in which mobile network topology contributes to latency, given that some tests with 5G encountered greater variance, while 4G seemed to provide more reliable results.

Moreover, when implementing the authors' own jitter buffer using the data channel, an unexpected issue was discovered where playback sounded like white noise rather than the intended audio stream, even when testing using a local area network. The issue occurred when the buffer size grew too large, with the exact threshold depending on the mobile device's browser running the node client. Strangely, setting the buffer past this threshold caused the client to misinterpret the type of data, specified as frames of 128 bytes, from an `ArrayBuffer` to an `UInt8Array`, causing the audio data to be in the range of 0–255 instead of the intended normalized range of –1 to 1. The WebRTC data channel effectively supports dynamic typing through Javascript, but it is not explicitly defined by the API itself. Instead, the data type is dynamically determined at runtime based on what is passed to `send()`, and the receiver must handle different types accordingly. The authors have no explanation as to why this process fails in their own implementation using the data channel.

## 4 THE AUXTRUMENT AND AUDIO

The Auxtrument uses the open-source and royalty free Opus codec that is preferred by WebRTC [25]. It operates with a sampling rate of 48 kHz to match that of most web browsers to prevent resampling at any stage in the transmission process, along with variable bitrate encoding. The recording for both the locations and central hub is handled using the web browsers' `getUserMedia()` javascript function of the MediaDevices API, which returns an audio stream from the selected audio input device.

At the same time, audio playback requires the Web Audio API. Once the audio has been received from all locations, the hub parses the channels from the streams of each location and merges them into a single multichannel stream. For instance, three locations with three interleaved stereo streams are merged into a single stream with six channels that is sent to a multichannel audio device and controlled via a mixer (Fig. 2). This process is performed using the splitter and merger web audio graph nodes [26]. The splitter transforms the interleaved stereo signals into separate mono signals, and then the merger node mixes all six mono signals for the left and right channels, respectively.

### 4.1 WebRTC Session Description Protocol

When setting up the aforementioned WebRTC connection, the protocol requires stream configuration information. SDP munging is required to configure a media stream to contain more than the default single channel [27]. The SDP is the standard that describes media communication sessions, and it contains the codec parameters, source address, and timing information of audio and video streams. For the Auxtrument, streaming stereo audio—to be diffused in the concert hall—was chosen as the minimum requirement for achieving an immersive experience, while remaining lightweight enough to minimize hardware demands and avoid potential bandwidth issues. Using WebRTC, streaming a stereo signal requires forcing the boolean `stereo = 1` flag to the description. However, the WebRTC documentation makes no mention of editing the SDP or specifying stereo transmission, and in hindsight, this makes

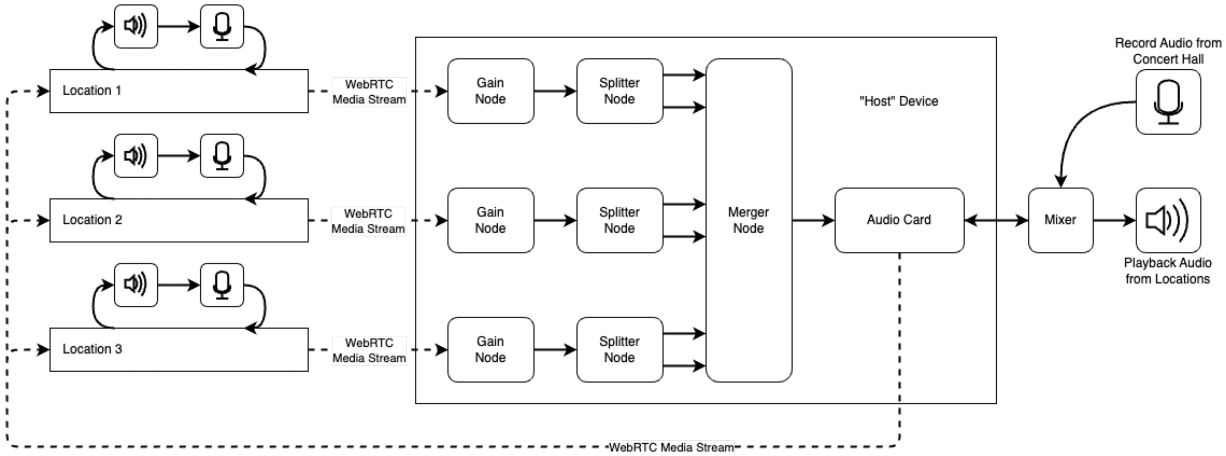



Fig. 2. WebRTC and Web Audio API between Auxtrument peers.

```
function modifySDPForStereo(sdp) {
  const sdpLines = sdp.split('\r\n');
    for (let i = 0; i < sdpLines.length; i++) {
      if (sdpLines[i].startsWith('a=fmtp:111')) {
        sdpLines[i] += 'channel_mapping=0,1,2,3,4,5';
        num_streams=4;coupled_streams=2;
        maxaveragebitrate=510000';
      }
      if (sdpLines[i].startsWith('a=rtpmap:111')) {
        sdpLines[i] = 'a=rtpmap:111 multiopus/48000/6';
      }
    }
  const modifiedSDP = sdpLines.join('\n');
  return modifiedSDP;
}
```

Listing 1. SDP munging function.

sense given that most WebRTC applications use mono audio streaming for video conferencing.

In addition to this, WebRTCs `getUserMedia()` function includes a `channelCount` constraint that allows users to specify the number of channels in the local stream. This does not, however, determine the number of channels one is expecting to send and be received by another peer over the network. That property must be munged in the SDP file negotiated between two peers during signaling. Without this, a stereo signal is taken from one peer and a mono signal is received by the other. For multichannel audio specifically, the required codec name is `multiopus` as opposed to `opus`, which also must be munged in the SDP offer along with the channel configurations. However, WebRTC does not mention the codec, and in both cases, the necessary solutions were found in community forums. Listing 1 shows a function that edits the SDP file for multichannel streaming.

### 4.2 Audio Ecosystem Limitations

To ensure ease of use and portability, mobile device support was prioritized—enabling connections from remote locations with minimal hardware while maintaining reliable connections. iOS development was focused on primarily because most of the team uses Apple devices. Apple's restrictive approach to external microphone compatibility is part of its broader sandboxing and security philosophy, which prioritizes system integrity over user flexibility. While this makes iOS more secure and stable, it also creates significant challenges for real-time audio applications—especially those relying on WebRTC and web-based tools rather than native apps [28]. Unlike macOS, where WebRTC freely accesses external audio devices via standard APIs, iOS restricts audio input at the system level. Even when a device is selected using WebRTCs input selection function, iOS does not always honor the selection. This was the case with a Zoom ZMS-44 two-channel audio interface, which promised iOS compatibility.

After testing other devices, the RØDE NTG mobile boom mic proved compatible with Google Chrome and WebRTC on iOS. However, iOS treated the external mic as an additional input rather than a full replacement. The internal mic did not turn off completely and continued picking up audio, albeit at a lower volume. The streams for both inputs were merged into a single channel and sent over the network, leading to an undesirable mix of sources. In the end, the investigation ended here, mainly due to time constraints. However, the RØDE NTG proved otherwise successful given the project goals. The RØDE NTG also features a 3.5 mm line output that was used with an OB-1 speaker by Teenage Engineering for playback.

## 5 DEBUGGING

While developing the Auxtrument in line with their objectives, the authors encountered numerous challenges and gained valuable insights. However, debugging proved particularly difficult due to the complex and fragmented nature of the technology stack, which involved multiple layers and components. Furthermore, the lack of comprehensive and accurate documentation often hindered progress, forcing reliance on community forums and other unofficial sources.

### 5.1 Debugging Complexity

Debugging presented a huge challenge throughout the project, particularly because of the "black box" nature of these technologies and the fact that hearing no sound from another location during testing could represent one of dozens of potential issues, many of which are extremely

time consuming, if not impossible to find. Problems could lie with audio hardware, operating system, browser type and version, WebRTC, Web Audio API, the authors' own implementation, a library, or mobile network(s). In most browsers, for example, WebRTC and Web Audio API work together seamlessly, whereas in the latest version of Safari at the time (17.5.1), multichannel through the Web Audio API became nonfunctional after updating.

Furthermore, a known regression in Safari 17.5.1 causes the `AudioContext()` to stop after the browser loses and then regains focus (e.g., minimizing the window) [29]. Even when interacting with UI elements designed to produce sound, the audio remains unresponsive for the rest of the session. This is indicative of broader challenges with WebKit, the engine underlying Safari, which has historically lagged in implementing certain web standards. For instance, the `AudioWorklet` interface, crucial for low-level audio processing, was not implemented in WebKit until 2021, hindering developers relying on this feature for enhanced audio capabilities [30]. These inconsistencies across the Apple ecosystem add complexity to debugging, as behaviors can vary significantly between WebKit and other browser engines and devices, sometimes necessitating browser-specific workarounds.

Another major challenge is logging and monitoring in a real-time WebRTC environment. Traditional in-browser debugging tools often fall short because WebRTC operates asynchronously and across multiple layers, meaning errors can occur at any stage—signaling, negotiation, network transport, or playback—without clear visibility into where the failure originates. While tools like `webrtc-internals` (Chrome) and `about:webrtc` (Firefox) can provide useful insights, they require detailed expertise to interpret, and they do not always reveal the full picture when dealing with custom audio routing or multichannel implementations. They also do not communicate any information about how, if at all, WebRTC and Web Audio API are collaborating.

The abstraction process used by Peer.js makes establishing connections across devices and networks easier, but more difficult to control and debug. For instance, there are bug reports that express difficulty connecting certain mobile providers. There are even reports with Peer.js and iOS version compatibility, suggesting it is not guaranteed to work across all variations reliably. Furthermore, Peer.js includes its own list of STUN and TURN servers to use as defaults. In many reports, mirrored by the authors' own experience, the Peer.js TURN relay tries repeatedly to negotiate between candidates before eventually failing to establish a connection. As a post concert test, the authors set up their own TURN relay through Metered, a WebRTC TURN relay service, and noticed a bug in the Auxtrument code that was not properly overwriting the default Peer.js servers with their own server specifications. After fixing the issue, a connection was able to be established using the TURN relay on both eduroam and Tele2 networks. This points to an issue with the TURN relays provided by Peer.js.

One reason Peer.js was not suspected sooner was because it does not log anything, and there was no reason to expect that the servers were not even being considered. When development began, using a TURN relay was explicitly decided against due to its added latency (an average roundtrip time of approximately 115 ms based on some of the tests). It is possible that considering this option sooner, rather than for trying to debug network issues after the fact, might have helped discover the problem with Peer.js earlier in the development process. In any case, this says nothing about the use of STUN with different network architectures and NATs, which still presents significant challenges.

### 5.2 Documentation

There were many moments throughout development where a property, method, or constraint was not documented through the WebRTCs documentation repository. Instead, these were found elsewhere on community forums such as Stack Overflow, CodePen examples, and Reddit threads. A recent Editor's Draft from World Wide Web Consortium (W3C) that was published on February 13, 2025, includes the most complete overview of the more esoteric capabilities of WebRTC that the authors have been able to find [31].

Although this is an incomplete draft as of the day of writing, many unknowns about WebRTC features are revealed, although the information is largely inaccurate. For example, the `channelCount` constraint is initially listed as a boolean, but another place on the same web page correctly lists it as unsigned long. This is the case with all constraints WebRTC describes in this draft. The latest published version from October 8, 2024, makes one mention of the `channelCount` constraint, but does not define it or reference how it should be used and with what data type.

## 6 OTHER APPROACHES

There are several speculative approaches and improvements that could have been considered to answer the research question and address the challenges presented in this paper. While WebRTC provided a workable solution, the unpredictability of its implementation and distribution, particularly for a live concert setting, highlights the need for more user-friendly and reliable solutions.

### 6.1 Decentralized Audio Routing

Instead of relying on a single WebRTC-based application to handle all connections, separate devices could have been used for each pair of connections. This approach would have simplified the network topology and reduced the complexity of debugging, as each connection would be independent. However, this would have required more devices. Although the authors chose to develop a web-based application for its cross-platform compatibility, they could have considered developing a native app for iOS and Android. A native app would have provided better access to device hardware and operating system features, custom handling of jitter buffers, and compensation for packet losses, thus potentially overcoming some of the limitations presented using web-based solutions.

### 6.2 Virtual Private Networks

Routing all connections over a virtual private network (VPN) could have simplified peer discovery since each device can be assigned a hostname or static virtual IP address. This option was decided against in practice simply because the WebRTC application was already developed. It was also posited that the underlying network routes of a VPN would effectively be the same as those negotiated by WebRTCs ICE protocol and, therefore, the VPN would serve no additional benefit. However, the VPN's data encryption may slightly increase overall transmission latency.

As a retrospective test, the authors attempted to recreate the Auxtrument between two SonoBus clients connected to a VPN—a laptop connected to a corporate WiFi network and an iPhone connected to the Tre mobile network. Using SonoBus' *connect to group* functionality, the laptop connected successfully. When the phone attempted to connect, an error message appeared, and no audio was transferred. Using SonoBus' experimental *direct connection* functionality using the IPs provided by the VPN, audio was able to be sent in only one direction—from phone to laptop. There was no further debugging information available from SonoBus, so it is unclear exactly where the error occurred.

### 6.3 Alternative Protocols

As a potential alternative to WebRTC, the authors experimented with creating a transmission protocol similar to Reliable (Unreliable) Streaming Protocol (RUSH) [32], which is based on QUIC [33] and allows fine-grain control over delivery guarantees and avoidance of congestion. However, it was determined that implementing an entire streaming protocol from scratch was not feasible in the time available. One of the marketed benefits of WebRTC was that the implementation already exists in browsers, and for a networked music performance, the authors were inclined to favor tried-and-tested software over an experimental option.

## 7 THE AUTHORS' VISION

The grand vision of the research project (Information Retrieval in Embedded Systems for Audiovisual Artistic Processes) is the ambition to have a musical performance in remote locations where the performers and audience are interconnected through a mobile interface. This interface will allow access not only to the sound of the performance but to the data involved in producing the performance, thereby blurring the performer-audience boundary. In addition, participants are also envisioned coming and going from the performance ecosystem in real time. Currently, Internet of Things technologies are the only viable option for such a system, placing great importance on established technologies such as WebRTC. This vision could, for example, be a merged reality concert, where the functionality of the Auxtrument allows any user to filter sound through the physical environment of another user at any time. For this—and other experimental network music performance applications—to succeed, the technologies discussed in this paper need to be reliable, robust, and scalable.

## 8 CONCLUSION

One of the primary challenges faced was the lack of comprehensive documentation for WebRTC, especially multichannel audio and SDP munging. In the future, the authors would expect more detailed and accessible documentation for developers. In addition, they hope for the development of standardized debugging tools that provide better visibility into the interaction between WebRTC and Web Audio API in the browser. WebRTC has been available since 2011 and stable since 2018. Nevertheless, it is suggested that web browser developers increase their efforts to make their implementations compliant with W3C standards. Finally, addressing the issues with mobile networks regarding transparency and compatibility are paramount, particularly with STUN server signaling, but also wishful thinking. Perhaps until the transition to IPv6 address space is complete, or until NAT vendors become more aware of the requirements of significant P2P applications, many of these mobile network challenges will remain unsolved.

The journey creating a web-based, real-time distributed reverberation chamber reveals both the potential and limitations of current technologies for real-time audio streaming over mobile networks. While WebRTC provided a foundational solution, its implementation exposed significant challenges, including network compatibility issues, and hardware/software ecosystem limitations, and debugging complexity. These challenges highlight the need for more robust, transparent, and well-documented tools to support real-time audio applications. Looking ahead, the development of standardized debugging tools and more comprehensive documentation for WebRTC and related technologies will be critical for advancing real-time audio streaming applications. This project serves as a case study in the complexities of web audio technologies and offers valuable insights for future research and development.

## 9 ACKNOWLEDGMENT

Information Retrieval in Embedded Systems for Audiovisual Artistic Processes (IRESAP) is supported by The Knowledge Foundation (KKS).

## 10 REFERENCES

[1] A. Franklin, D. Hedin, R. Lindell, and H. Frisk, "Merging Places: A Real-Time Distributed Live Reverberation Chamber," in *Proceedings 16th International Conference on Quality of Multimedia Experience (QoMEX)*, pp. 54–57 (Karlshamn, Sweden) (2024 Jun.).

[2] S. Loreto and S. Romano, *Real-Time Communication With WebRTC* (O'Reilly Media, Sebastopol, CA, 2014).

[3] 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*, pp. 206–214 (Utrecht, The Netherlands) (2024 Sep.). https://doi.org/10.5281/zenodo.13904834.

[4] R. A.Stebbins, *Exploratory Research in the Social Sciences*, Sage Research Methods, vol. 48 (SAGE Publications, Thousand Oaks, CA, 2001). https://doi.org/10.4135/9781412984249.

[5] L. Candy, "Practice Based Research: A Guide," Tech. Rep. 2006-V1.0 (2006 Nov.).

[6] S. Scrivener, "The Art Object Does Not Embody a Form of Knowledge," *Work. Pap. Art Design*, vol. 2 (2002 Jan.).

[7] JackTrip Labs, "JackTrip," https://www.jacktrip.com/ (accessed Feb. 27, 2024).

[8] Sonosaurus, "SonoBus," https://sonobus.net/ (accessed Feb. 10, 2024).

[9] K. Wierenga, S. Winter, and T. Wolniewicz, "RFC 7593: The eduroam Architecture for Network Roaming," Internet Engineering Task Force (2015 Sep.). https://doi.org/10.17487/RFC7593.

[10] Cockos, "REAPER: Digital Audio Workstation," https://www.reaper.fm/ (accessed Feb. 10, 2024).

[11] Apple, "Logic Pro," https://www.apple.com/logic-pro/ (accessed Feb. 12, 2024).

[12] Apple, "Audio Unit Properties," https://developer.apple.com/documentation/audiotoolbox/audio-unit-properties#InputOutput (accessed Feb. 12, 2024).

[13] Apple, "Bonjour," https://developer.apple.com/bonjour/.

[14] Y. Amo, G. Zissu, S. Eloul, et al., "A Max/MSP Approach for Incorporating Digital Music via Laptops in Live Performances of Music Bands," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 94–97 (London, UK) (2014 Jun.).

[15] Institut für Elektronische Musik und Akustik (IEM), "Alternatives to VRR," (2020 Mar.). https://vrr.iem.at/docs/alternatives/.

[16] W. Ritsch, "Towards a Message Based Audio System," *Proceedings of the 12th Linux Audio Conference*, pp. 23–30 (Karlsruhe, Germany) (2014 May).

[17] H. Wallach, E. B. Newman, and M. R. Rosenzweig, "The Precedence Effect in Sound Localization," *Am. J. Psycol.*, vol. 62, no. 3, pp. 315–336 (1949 Jul.). https://doi.org/10.2307/1418275.

[18] Open JS Foundation, "Express: Fast, Unopinionated, Minimalist Web Framework for Node.js," https://expressjs.com/ (accessed Feb. 18, 2024).

[19] Open JS Foundation, "Run JavaScript Everywhere," https://nodejs.org/en (accessed Feb. 18, 2024).

[20] PeerJS, "The PeerJS Library," https://peerjs.com/ (accessed Feb 18, 2024).

[21] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *Proceedings of the USENIX Annual Technical Conference*, pp. 179–192 (Anaheim, CA) (2005 Dec.).

[22] O. Kanaris and J. Pouwelse, "Mass Adoption of NATs: Survey and Experiments on Carrier-Grade NATs," arXiv preprint arXiv:2311.04658 (2023 Nov.).

[23] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RFC 3550: RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force (2003 Jul.). https://doi.org/10.17487/RFC3550.

[24] R. Stewart, M. Tüxen, and K. Nielsen, "RFC 9260: Stream Control Transmission Protocol," Internet Engineering Task Force (2022 Jun.).

[25] J.-M. Valin, K. Vos, and T. Terriberry, "RFC 6716: Definition of the Opus Audio Codec," Internet Engineering Task Force (2012 Sep.).

[26] P. Adenot and H. Choi, "Web Audio API 1.1," W3C First Public Working Draft (2024 Nov.). https://www.w3.org/TR/webaudio-1.1/.

[27] A. C. Begen, P. Kyzivat, C. Perkins, and M. J. Handley, "RFC 8866: SDP: Session Description Protocol," Internet Engineering Task Force (2021 Jan.). https://doi.org/10.17487/RFC8866.

[28] M. Blochberger, J. Rieck, C. Burkert, T. Mueller, and H. Federrath, "State of the Sandbox: Investigating macOS Application Security," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pp. 150–161 (London, UK) (2019 Nov.).

[29] WebKit Bugzilla, "Bug 276016: REGRESSION (iOS 17.4.1 - 17.5.1): Web Audio Sounds Stop Playing After Losing Focus of Safari in iOS 17.5.1," https://bugs.webkit.org/show_bug.cgi?id=276016 (accessed Feb. 26, 2025).

[30] Mozilla Developer Network, "AudioWorkletNode," (2024 Jul.). https://developer.mozilla.org/en-US/docs/Web/API/AudioWorkletNode.

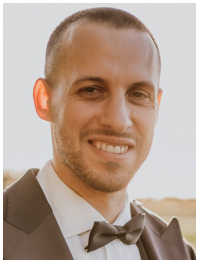
[31] C. Jennings, F. Castelli, H. Boström, and J.-I. Bruaroey, "WebRTC: Real-Time Communication in Browsers," W3C Editor's Draft (2025 Sep.). https://w3c.github.io/webrtc-pc/.

[32] K. Pugin, A. Frindell, J. Cenzano, and J. Weissman, "RUSH - Reliable (Unreliable) Streaming Protocol," Internet Engineering Task Force (2021 Jul.). https://datatracker.ietf.org/doc/draft-kpugin-rush-00.

[33] J. Iyengar and M. Thomson, "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force (2021 May). https://doi.org/10.17487/RFC9000.

## THE AUTHORS


Austin Franklin


Richard Mitic


Daniel Hedin


Rikard Lindell


Henrik Frisk

Austin Franklin is a composer and researcher at Mälardalen University and the Royal College of Music in Stockholm, working with information retrieval, embedded systems, and artistic processes. He maintains a real-time timbral analysis library for Cycling '74 Max. His music has been performed at Carnegie Hall and received awards including the American Prize and the Petrichor International Music Competition.

•

Richard Mitic is a Ph.D. candidate at Mälardalen University, researching spatial audio, digital signal processing, and large-scale music information retrieval. He holds a master's in Music and Electronics from the University of Glasgow and has extensive industry experience from working with media streaming protocols and MPEG standardization to software for music streaming.

•

Daniel Hedin is a senior lecturer in computer science at Mälardalen University and a guest researcher at Chalmers University of Technology, specializing in language-based security. His research covers secure information flow, data minimization, and formal verification, with a recent focus on securing untrusted code and web applications. He is the principal designer of several practical tools, including JavaScript sandboxes and security-enhanced interpreters.

•

Rikard Lindell is a composer and professor at Mälardalen University (computer science, interaction design) and Dalarna University (audiovisual studies). His artistic work includes interactive installations, electroacoustic and orchestral compositions, and phonogram releases. In research, he explores code as design material and contributes to the International Conference on New Interfaces for Musical Expression community. He has developed commercial audiovisual performance apps for iOS.

•

Henrik Frisk is a composer and performer of contemporary music. He is a professor of composition at the Royal College of Music in Stockholm, where he teaches electroacoustic music. His research focuses on improvisation, interactivity, spatialization, and experimental music, with current projects exploring spatial perception and multidisciplinary approaches to electronic music heritage.