

Browser-Native Real-Time Multi-directional Streaming of Third-Order Ambisonics

Maurizio Ferrari
*Dept. of Information Engineering
and Computer Science
University of Trento
Trento, Italy
maurizio.ferrari-1@studenti.unitn.it*

Antonella Bevilacqua
*Dept. of Cultures and Societies
University of Palermo
Parma, Italy
antonella.bevilacqua@unipa.it*

Claudio Vittori
*R&D Dept.
Flower of Sound
Milan, Italy
claudio@flowerofsound.com*

Paul Adenot
*Mozilla Corporation
Paris, France
padenot@mozilla.com*

Michel Buffa
*SPARKS research group
I3S Laboratory
University Côte d'Azur
Sophia-Antipolis, France
michel.buffa@univ-cotedazur.fr*

Luca Turchet
*Dept. of Information Engineering
and Computer Science
University of Trento
Trento, Italy
luca.turchet@unitn.it*

Abstract—As the demand for spatial audio continues to grow for remote participation in professional conferences, virtual events, and immersive cultural experiences, there is an increasing need for audio delivery solutions that combine high perceptual quality with the accessibility and interoperability afforded by web technologies. While the Web has emerged as a ubiquitous platform for real-time communication and immersive experiences, the real-time transmission of high-fidelity spatial audio through web browsers remains challenging, particularly when audio is captured using dedicated spatial microphone arrays. This paper presents an open-source Web-based system for real-time multi-directional streaming of 16-channel third-order Ambisonics captured with a spherical microphone array (Zylia ZM-1). Using Mozilla Firefox, WebCodecs, and WebRTC DataChannels, the system transmits the Ambisonics signals among participants and renders each received soundfield locally as binaural headphone audio through Omnitone, with optional head-tracked rotation. The proposed configuration was implemented on Linux, Microsoft Windows 11 and macOS operating systems.

Index Terms—Ambisonics, spatial audio, real-time audio rendering, WebRTC, Web Audio.

I. INTRODUCTION

Networked immersive audio systems have become increasingly important for a broad range of applications, including remote participation in professional conferences, virtual events, cultural heritage experiences, telepresence, and collaborative artistic practices. Recent advances in immersive media have highlighted the importance of spatial audio for fostering presence, realism, and engagement in distributed virtual environments [1]. In particular, Higher-Order Ambisonics

(HOA) has emerged as one of the most versatile approaches for the representation and transmission of spatial sound scenes due to its playback-independent nature and compatibility with a wide variety of rendering systems, ranging from binaural headphones to multichannel loudspeaker arrays [2].

Among the various HOA configurations, third-order Ambisonics represents a practical trade-off between spatial accuracy and transmission complexity. Compared to first-order Ambisonics, it provides substantially improved spatial resolution and source localization while requiring only 16 audio channels, making real-time transmission feasible over contemporary communication networks and web technologies. Higher Ambisonics orders can further improve spatial fidelity but incur rapidly increasing bandwidth and computational requirements.

The increasing relevance of networked immersive audio has also been emphasized in the context of the Metaverse, where users are expected to interact through shared auditory environments distributed over communication networks [1], [3], [4]. In such scenarios, preserving the spatial characteristics of sound is essential to achieve convincing auditory experiences and support effective communication among geographically distributed users. At the same time, accessibility and interoperability have become key requirements for immersive media systems [5], [6]. Browser-native deployment eliminates installation barriers, facilitates interoperability, simplifies updates, and enables immediate participation in immersive audio sessions through a URL. Traditionally, immersive audio applications have relied on dedicated native software, proprietary platforms, or specialized hardware infrastructures. While these solutions can provide excellent performance, they often require software installation, platform-specific maintenance, and limited interoperability across devices. In contrast, web

This work was supported by the MUSMET project funded by the EIC Pathfinder Open scheme of the European Union (grant agreement n. 101184379). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Innovation Council. Neither the European Union nor the European Innovation Council can be held responsible for them.

technologies provide an attractive alternative by enabling cross-platform deployment through standard web browsers, thus significantly reducing barriers to adoption and facilitating access from heterogeneous devices and operating systems.

Technologies such as WebRTC, WebAssembly, WebGPU, and the WebXR Device API have enabled increasingly sophisticated browser-based applications. In particular, the Web Audio API provides a standardized framework for real-time audio synthesis, processing, routing, and spatialization directly within web browsers [7], [8]. The Web Audio API has become the foundation of numerous browser-based audio systems (e.g., the web audio modules [9]) and constitutes a key enabling technology for interactive spatial audio applications.

Recent research has explored the use of web technologies for immersive audio and WebXR applications [10]. Studies have investigated the capabilities of browser-based spatial audio frameworks, comparing different rendering approaches and evaluating their perceptual performance in immersive environments [11], [12]. These works demonstrate that modern browsers can support high-quality spatial audio rendering and immersive interaction. However, most existing efforts focus on sound synthesis, playback, and virtual sound source rendering rather than on the real-time acquisition and transmission of spatial audio signals captured from physical environments.

The transmission of HOA audio over communication networks has attracted growing attention in recent years. Several works have investigated immersive networked music performance systems [13], [14], adaptive Ambisonics transmission strategies [15], remote immersive audio production workflows [16], and HOA compression schemes for communication networks [17]. Furthermore, recent studies have proposed architectures for immersive networked music performances exploiting 5G and Multi-access Edge Computing technologies [18], [19]. Nevertheless, these approaches generally rely on dedicated software infrastructures and do not address browser-native deployment.

At the time of writing, spherical microphone arrays producing hardware-dependent A-format output, including the Zylia ZM-1 used in this work, cannot generally provide third-order Ambisonics directly to a browser. A Digital Audio Workstation (DAW) is therefore required to acquire and convert the microphone output into a 16-channel third-order AmbiX signal, which is then routed to the browser through a virtual audio device or an intermediate bridge, depending on the operating system. Given this necessary DAW-mediated capture and conversion stage, the integration of spherical microphone acquisition with browser-based HOA encoding, transmission, and reproduction remains largely unexplored. In particular, there are currently no readily available open-source solutions that allow users to: (i) convert the output of a multichannel spherical microphone array to HOA in a DAW and route the resulting signal to a web browser; (ii) encode and transmit HOA channels in real-time through standard browser technologies; and (iii) decode and reproduce through the browser the received spatial audio stream on heterogeneous operating systems without requiring dedicated native applications.

The main research question here addressed is: *Can current open Web standards support end-to-end, real-time streaming of 16-channel third-order Ambisonics — including capture, network transmission, mixing, and binaural rendering — directly through a web browser across different operating systems?*

To answer this question and address the gaps identified above, this paper makes three contributions:

- An open-source browser-native framework for end-to-end real-time multi-directional streaming of third-order Ambisonics, integrating DAW-mediated Ambisonics capture, transmission, and receiver-side binaural rendering within standard web technologies;
- A cross-platform architecture that enables the acquisition of multichannel Ambisonics audio from a spherical microphone array and its real-time transmission over WebRTC on Windows, Linux, and macOS without requiring dedicated native application on the receiver side;
- An evaluation demonstrating the feasibility of browser-based HOA streaming using existing Web standards, including WebRTC, WebCodecs, and the Web Audio API.

II. RELATED WORK

A. Networked Immersive Audio and HOA Transmission

Networked immersive audio has received increasing attention due to its relevance for telepresence, virtual events, and collaborative musical performances. Immersive Networked Music Performance systems extend traditional Networked Music Performance frameworks [20] by incorporating spatial audio techniques capable of recreating shared acoustic environments among geographically distributed musicians [13], [14]. Previous studies demonstrated the positive impact of immersive audio rendering on perceived presence and musical interaction quality, while also highlighting the stringent latency requirements imposed by real-time collaborative scenarios [13], [18].

Several approaches have addressed the transmission of HOA signals over communication networks. Ostan et al. proposed a dynamic HOA order adaptation strategy that adjusts the Ambisonics order according to network conditions to balance immersion and transmission reliability [15]. In the domain of audio coding, Hold et al. presented HOA coding extensions for the 3GPP Immersive Voice and Audio Services (IVAS) codec, enabling efficient transmission of HOA content through advanced parametric coding techniques [17]. Other works investigated remote immersive audio production systems and architectures for immersive audio delivery over distributed networks [16]. While these studies address transmission efficiency, adaptation, and network performance, they generally assume dedicated software infrastructures rather than browser-native implementations.

B. Web-Based Spatial Audio

The emergence of the Web Audio API has enabled browser-based spatial audio rendering and interactive audio applications. Tomasetti et al. provided one of the first systematic

analyses of spatial audio tools compatible with WebXR applications, highlighting the strengths and limitations of existing browser-based frameworks [11]. Their study showed that while several libraries support binaural rendering and basic spatialization, support for Ambisonics and advanced immersive audio functionalities remains limited.

More recently, Boem et al. conducted a perceptual evaluation of different browser-based spatial audio technologies in WebXR environments, concluding that spatial audio implementation choices significantly affect localization accuracy and user experience [12]. Their findings confirm the feasibility of high-quality spatial audio rendering in modern browsers, but focus exclusively on playback and virtual source positioning rather than spatial audio acquisition and transmission.

The role of web technologies in immersive audio has also been discussed in the context of the Internet of Sounds and the Musical Metaverse, where browser-based applications are envisioned as key enablers of accessible and interoperable immersive experiences [3].

C. 5G and Edge-Based Spatial Audio Architectures

Another research direction concerns the use of advanced networking infrastructures to support immersive audio services. Rinaldi et al. proposed the exploitation of Multi-access Edge Computing for real-time binaural rendering in cultural heritage applications, demonstrating how edge computing can reduce latency and computational requirements on client devices [19]. Subsequently, Turchet et al. introduced 5G-enabled architectures for immersive networked musical practices, combining spatial audio rendering with network slicing, software-defined networking, and edge computing resources [18]. Although these architectures address important challenges related to latency and scalability, they rely on dedicated processing infrastructures and do not exploit the browser as a universal execution environment. Consequently, they do not address the problem of browser-native HOA acquisition and streaming, which is the focus of the present work.

III. SYSTEM ARCHITECTURE

The proposed framework separates the local audio hardware from the browser's network thread to ensure predictable processing times and maintain precise synchronization between audio channels. Rather than relying on a full mesh topology between peers, each client maintains a dedicated WebRTC connection to a central Multipoint Control Unit (MCU). The system is structured into four functional layers: (i) platform-specific audio capture, (ii) WebCodecs encoding and DataChannel transport, (iii) server-side soundfield mixing, and (iv) receiver-side decoding and binaural rendering. A WebSocket endpoint on the MCU handles session setup and Interactive Connectivity Establishment (ICE) negotiation for each client connection. Such layers are described hereinafter. Fig. 1 illustrates the end-to-end signal flow. On the transmit path, a Zylia ZM-1 microphone feeds REAPER, which produces a 16-channel O3A signal in AmbiX format (ACN, SN3D). The signal reaches the browser via platform-specific

ingestion paths (Section III-A), where each channel is encoded independently and transmitted to the MCU. The MCU computes a mix-minus stream for each receiver and re-encodes it for delivery. Each receiver reassembles the 16-channel frame, which is then rendered to binaural.

We adopted Mozilla Firefox as the reference browser because, at the time of development, it was the only browser simultaneously satisfying the two critical platform requirements imposed by this architecture. The first requirement concerns *multichannel audio capture*. On macOS, the ingestion path captures 16 discrete audio channels from a virtual loopback device (BlackHole 16ch) via the `getUserMedia` API, requesting `channelCount: 16` with echo cancellation, noise suppression, and automatic gain control disabled to preserve the Ambisonics signal integrity. Chromium-based browsers (Chrome, Edge, Opera) impose a hard limit of two input channels due to an architectural constraint in their audio capture pipeline, originally designed for voice communication scenarios. This limitation has been tracked in the Chromium issue tracker since 2015 and remains unresolved at the time of writing. Firefox, whose audio backend negotiates the channel layout directly with the operating system's audio server (CoreAudio on macOS, PulseAudio/PipeWire on Linux), exposes the full channel count reported by the hardware or virtual device, thereby enabling direct 16-channel capture without an intermediate bridge process.

The second requirement concerns *codec support*. The proposed transport architecture relies on the WebCodecs API, specifically the `AudioEncoder` and `AudioDecoder` interfaces, to independently encode and decode each of the 16 O3A channels as mono Opus streams at the application level. This bypasses WebRTC's built-in `MediaStreamTrack`-based audio pipeline, which is limited to stereo and applies communication-oriented signal processing (echo cancellation, automatic gain control) that would corrupt the Ambisonics soundfield. Firefox 130 and later versions provide full WebCodecs support including Opus encoding and decoding, matching the capability available in Chrome since version 94. However, only Firefox combines this WebCodecs functionality with multichannel `getUserMedia` capture, making it the sole browser capable of executing the complete end-to-end pipeline on macOS without requiring an external bridge application. On Windows and Linux, where a Python bridge feeds the 16-channel audio via a local WebSocket, the multichannel capture constraint is not exercised and any browser implementing WebCodecs with Opus support could in principle serve as the client.

Table I summarizes the W3C standardization status of the web platform APIs employed by the proposed system. The WebRTC API, including the `RTCDataChannel` used for transport, and the Web Audio API, which provides the `AudioWorklet` processing nodes and the downstream binaural rendering pipeline, are both full W3C Recommendations. The Media Capture and Streams specification, which defines `getUserMedia`, is a W3C Candidate Recommendation — a mature stage indicating that the API surface is stable and

TABLE I
W3C STANDARDIZATION STATUS OF THE WEB APIS USED

API	W3C Status	Role in System
WebRTC 1.0	Recommendation (2025)	DataChannel transport
Web Audio API 1.0	Recommendation (2021)	AudioWorklet, rendering
getUserMedia	Candidate Recommendation (2025)	Multichannel capture
WebCodecs	Working Draft	Opus encode/decode

undergoing interoperability testing rather than active redesign. The WebCodecs API is the least mature of the four, currently classified as a W3C Working Draft maintained by the W3C Media Working Group. Despite this formal classification, the API has been shipped by default in Chrome since version 94 (September 2021), Firefox since version 130, and Safari since version 26, with no backward-incompatible changes to the audio encoding interfaces since initial release. All major browser vendors have demonstrated commitment to the specification by integrating it into their stable release channels, and real-world compatibility data reports over 96% support for Opus encoding and decoding across deployed browser populations.

We acknowledge that building on a Working Draft specification carries inherent risk; however, the practical stability, multi-vendor implementation consensus, and the fact that the Opus codec configuration surface used in this work represents the most basic and widely exercised subset of the API justify its adoption for research purposes. The architecture is furthermore designed to be resilient to potential specification evolution: all WebCodecs usage is encapsulated in a single class (`AmbisonicsNode`) that can be adapted to API changes without modifying the transport, signaling, or rendering subsystems.

A. Cross-Platform Low-Latency Audio Ingestion

Three platform-specific ingestion paths feed 16-channel O3A audio into the browser-side encoding pipeline: one for Windows, one for macOS, and one for Linux.

On Windows, the REAPER DAW captures a raw, hardware-dependent A-Format audio stream from a spherical microphone array. Specifically, in our system we used the Zylia ZM-1, but any similar microphone can potentially be utilized. The stream is transformed into an O3A signal through an Ambisonics Converter plugin¹, which applies the AmbiX [21] format utilizing Ambisonic Channel Numbering (ACN) ordering and Schmidt Semi-Normalization (SN3D) scaling. The resulting 16-channel discrete signal is routed out of the DAW via the virtual ReaRoute ASIO driver architecture, an extension of the Steinberg ASIO specification. A local Python bridge script binds to the ReaRoute ASIO interface through the `sounddevice`² library configured with explicit ASIO channel selectors. The bridge maintains an accumulation buffer that

absorbs different ASIO block sizes and always emits frame-aligned payloads. Each completed frame is prepended with an 8-byte timestamp and broadcast over a local WebSocket loopback to the browser client.

On macOS, the same REAPER and Ambisonics Converter pipeline is used to produce the 16-channel O3A signal. The routing difference is at the DAW output stage: instead of ReaRoute, the BlackHole³ 16-channel virtual audio driver is configured as the output device. The web client then captures this virtual loopback directly via the `getUserMedia` API, requesting 16 discrete channels with echo cancellation, noise suppression, and automatic gain control disabled to preserve the Ambisonics signal integrity. This removes the need for an intermediate bridge process, since the browser can open the BlackHole device as a native media input.

On Linux, the same REAPER pipeline produces the 16-channel O3A signal. Both REAPER and the Python bridge script are launched under the PipeWire `pw-jack` wrapper, which registers them as JACK clients inside the PipeWire graph. An accumulation buffer in the script absorbs any period mismatch and produces frame-aligned payloads. Each frame is prepended with an 8-byte timestamp and broadcast over the local WebSocket loopback to the browser client.

B. WebRTC Transport and Codec Architecture

Each browser client establishes a dedicated WebRTC connection with the MCU. A single DataChannel per client, configured as unordered with zero retransmissions, carries the full-duplex audio stream. The MCU acts as the SDP offerer, embedding frame size N_f in its offer. Each of the 16 O3A channels is encoded independently by a dedicated WebCodecs⁴ `AudioEncoder` instance configured for mono Opus⁵ at 64 kbps constant bit rate with `latencyMode` set to `realtime`, which instructs the Opus encoder to minimize algorithmic look-ahead at the cost of some quality.

The Opus frame duration is derived from the server-authoritative frame size $N_f \in \{120, 240, 480, 960, 1920, 2880\}$ samples, corresponding to durations of 2.5 ms through 60 ms at 48 kHz sample rate. The default is set to 960 samples (20 ms), which balances algorithmic latency, packetization overhead, and loss concealment granularity. The frame size is uniform across all clients and fixed for the lifetime of the DataChannel. Each outgoing packet prepends a 9-byte binary header to the raw Opus payload, as described in Table II.

The recording sync flag and the channel index are packed into a single byte using bit masking. The frame timestamp is an integer sample counter incremented by N_f per frame, providing a reliable key for receive-side reassembly. The per-channel sequence number enables reorder-tolerant packet loss estimation on the receiver. The recording sync flag allows the sender and receiver to export a temporally aligned recording segment without additional synchronization overhead.

¹<https://www.zylia.co/zylia-ambisonics-converter.html>

²<https://python-sounddevice.readthedocs.io>

³<https://existential.audio/blackhole>

⁴<https://www.w3.org/TR/webcodecs/>

⁵<https://opus-codec.org/>

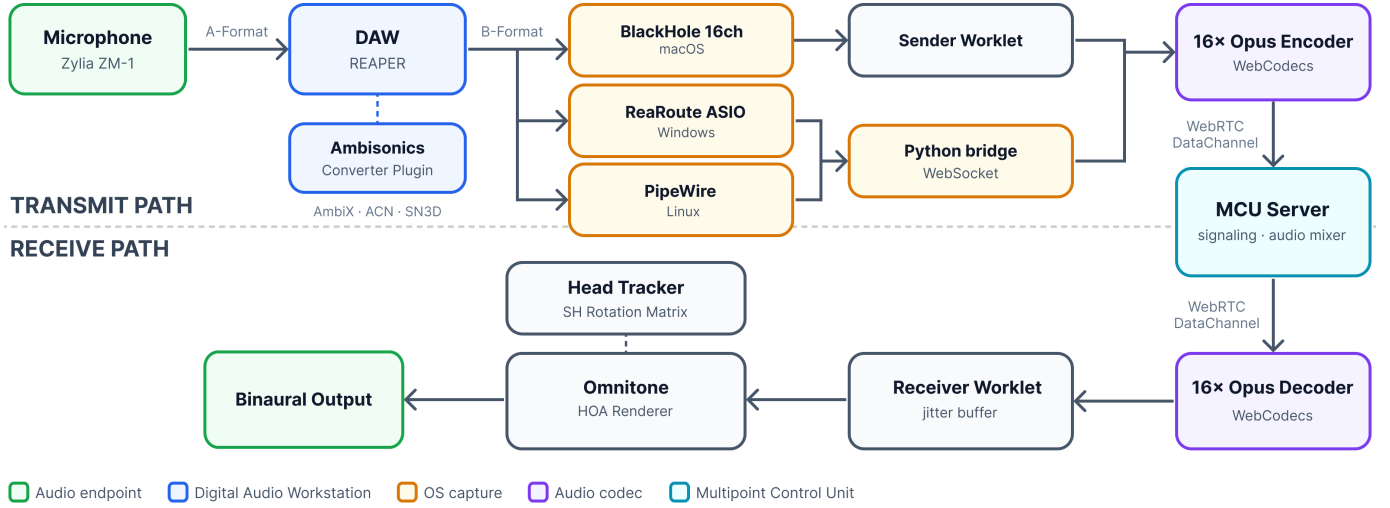


Fig. 1. End-to-end architecture of the proposed browser-native HOA streaming framework. On the transmit path, a 16-channel third-order AmbiX signal is captured from the Zylia ZM-1 via platform-specific OS paths, Opus-encoded per channel, and sent to the MCU over a WebRTC DataChannel. The MCU returns a mix-minus Ambisonics stream. On the receiver path, each receiver decodes the 16 channels, absorbs network jitter, and renders the stream to binaural audio through Omnitone. Colors denote functional roles.

TABLE II
DATACHANNEL PACKET HEADER

Bytes	Type	Field
0	uint8	Bit 7: Recording sync flag Bits 4–6: Reserved (not used) Bits 0–3: Channel index
1–4	uint32 LE	Per-channel sequence number
5–8	uint32 LE	Frame timestamp (samples)
9+	—	Opus payload

C. Server-Side Soundfield Mixing

The MCU is implemented in Go using the Pion WebRTC⁶ library. Each connecting client owns a PeerConnection, 16 Opus decoders and a jitter buffer. Clients are grouped into rooms, where a single mix-clock goroutine performs the following operations. First, it reads one decoded frame from every client’s jitter buffer as 16-channel pulse-code modulation (PCM) at `int16` precision. Second, it accumulates and sums them channel-wise in a global soundfield at `int32` precision, preventing overflow. Third, it delivers to each client the mix-minus frame: the global sum minus that client’s own contribution. Each mix-minus frame is then re-encoded by 16 Opus encoders and delivered to the client’s downlink DataChannel.

D. Decoding, Frame Assembly, and Binaural Output

On the receive path, 16 independent `AudioDecoder` instances reconstruct the PCM streams. Because the WebCodecs decoder may shift output timestamps by a codec-internal pre-skip offset, the original network frame timestamp is preserved in a per-channel first-in-first-out (FIFO) queue and retrieved

when the decoded `AudioData` callback fires, preventing timing misalignment in the assembled frame.

Decoded mono frames are assembled into complete 16-channel Ambisonics frames by a pooled slot structure keyed by frame timestamp; the assembly timeout policy that handles late or missing channels is detailed in Sec. IV-B. Assembled frames are then dispatched to a playback `AudioWorklet`, which implements the adaptive jitter buffer described in Sec. IV-A.

The worklet’s 16-channel output is routed into the Omnitone⁷ third-order HOA renderer, which applies Head-Related Transfer Function (HRTF) convolution to produce a binaural stereo output delivered to the system audio output. The renderer also accepts a rotation matrix that rotates the soundfield, so that the binaural image remains spatially anchored when the listener turns their head. Orientation is provided by a MetaMotionRL BLE inertial sensor worn by the listener. The sensor streams quaternions at 100 Hz to a local WebSocket. The browser applies Normalized Linear Interpolation between successive orientations, converts each quaternion to a rotation matrix and applies it to the Omnitone renderer.

IV. IMPLEMENTATION DETAILS

This section covers the design decisions and algorithmic mechanisms that underlie the architecture described in Section III. In particular, it addresses the choices that most directly affect perceived audio quality and real-time stability.

A. Adaptive Jitter Buffer

The jitter buffer is a timestamp-anchored circular ring buffer (default capacity 2 s) implemented in the receiver `AudioWorklet`. Each sample is written at its absolute timestamp position within the ring, guaranteeing correct phase alignment regardless of main-thread dispatch order. Slots are zeroed after

⁶<https://github.com/pion/webrtc>

⁷<https://github.com/googlechrome/omnitone>

each 128-sample read quantum to prevent stale data from resurfacing on underrun recovery. If the write pointer laps the read pointer – e.g., after a browser suspend starves the audio callback – an overflow guard snaps the read pointer forward to the newest valid region, bounding latency to the ring capacity without ever reading overwritten samples. To mitigate the transition introduced by this pointer adjustment a short crossfade is applied during active playback.

Playback begins once the buffered span exceeds a prebuffer threshold D_{pre} , which is adjusted at runtime by an adaptation loop on the main thread. The two adjustment directions are asymmetric:

- **Growth.** If the underrun count exceeds a threshold (default 2), the prebuffer increases by one frame duration N_f , up to a ceiling of $6N_f$ samples. The worklet is re-anchored first: the ring is flushed, playback pauses, and audio resumes only after the buffer refills to the new, deeper threshold. This introduces a short interruption in playback.
- **Decay.** If no underruns occur and the prebuffer exceeds the floor of $2N_f$ samples, it shrinks by $\lceil N_f/2 \rceil$ samples. No re-anchor is issued; the buffer drains naturally to the shallower threshold without interrupting playback.

B. Frame Assembly and Timeout Strategy

Decoded mono channels are assembled into complete 16-channel Ambisonics frames using a timestamp-indexed ring buffer. When all channels associated with a frame timestamp have been received, the assembled frame is forwarded to the playback worklet. Missing channels are padded with silence after a configurable assembly timeout T_{asm} , which starts when the first channel of a frame arrives. The timeout is computed as

$$T_{\text{asm}} = \min\left(\frac{N_f}{f_s} \cdot 1000 + 15, 0.75 \cdot \frac{D_{\text{pre}}}{f_s} \cdot 1000\right) \text{ ms},$$

where $f_s = 48\,000$ Hz is the sampling frequency and D_{pre} is the active prebuffer depth.

The first term dominates for large N_f (e.g., $N_f = 2880$), where the 15 ms margin absorbs scheduling and network jitter; the second term becomes active for small N_f (e.g., $N_f = 120$), capping the timeout at three quarters of D_{pre} so a single late frame cannot exhaust the jitter buffer and trigger cascading underruns.

The final value is clamped to a 4 ms lower bound to prevent scheduling unreliability induced by native browser `setTimeout` resolution limits.

C. Bundled Packet Format and Channel Redundancy

There are two transmit wire formats: legacy and bundled. The legacy format produces one packet for each channel every frame. The bundled format groups multiple channels into a single `DataChannel` message, reducing packet rate and header overhead at the cost of a higher packet loss impact. Each packet carries a 7-byte shared header containing the frame timestamp and a 16-bit channel bitmask, followed by the Opus

payloads in ascending channel order. The first-order channels (ACN 0–3) are included in two separate packets per frame, so that a single packet loss leaves the base soundfield intact.

D. Encoder Configuration and Opus Complexity

The Opus complexity parameter controls the trade-off between encoding quality and CPU usage. At small frame sizes (2.5–5 ms) the encoder must process frames at a very high rate, and higher complexity values would risk CPU starvation. The system therefore scales complexity with the frame duration: from 1 at 2.5 ms, to 7 at 60 ms and above, with intermediate values of 2 at 5 ms, 3 at 10 ms, and 5 at 20 ms.

E. NAT Traversal and Connectivity

Connectivity is established through the Interactive Connectivity Establishment (ICE) framework, which discovers and validates candidate transport paths between each browser client and the MCU. A public STUN server is used to obtain server-reflexive addresses whenever direct peer connectivity is possible. When restrictive or symmetric NATs prevent direct communication, traffic is relayed through a TURN server.

The current implementation employs Cloudflare’s TURN service. Temporary credentials are obtained by the signaling server at session startup and injected into the `RTCPeerConnection` configuration before ICE gathering begins. ICE candidates are exchanged using trickle ICE, allowing connectivity checks to start immediately and reducing connection establishment time.

F. Synchronized Bidirectional Recording

Each client supports a synchronized recording mode that captures the full 16-channel Ambisonics signal over a configurable duration (default 10 s). When a client initiates a recording, it sets the `isRec` bit in byte 0 of every outgoing packet header and captures pre-encode PCM frames locally. The MCU propagates the flag in its mix output, so all connected receivers begin accumulating the corresponding decoded frames. The audio captures are exported as 16-channel waveform audio file format (WAV) and enable a direct pre-encode versus post-decode comparison.

V. PERFORMANCE ANALYSIS

A. End-to-end Latency Budget

The one-way latency is the sum of a sequence of additive stages along the transmit and receive paths, where T_f is the frame duration in seconds. Each term is described below, in signal-flow order.

- **ADC conversion** (T_{adc}). First, the audio interface must digitize the analog microphone output. Modern converters typically introduce a delay of ≈ 0.5 ms.
- **Hardware capture buffer** (T_{cap}). The ASIO, CoreAudio or JACK driver delivers audio in fixed-size blocks. Typical values range from 1 to 5 ms depending on driver configuration.
- **Frame accumulation** (T_f). The sender worklet on macOS, or the Python bridge on Windows and Linux, must

collect exactly one complete frame of N_f samples before encoding can begin. This stage contributes exactly one frame duration. On the Windows and Linux path, the additional latency for WebSocket transport and frame de-interleaving is negligible.

- **Opus encoding** (t_{enc}). With `latencyMode` set to `realtime`, the per-frame compute cost is below 1 ms on any modern CPU and is effectively negligible.
- **Network propagation** ($2 \times \frac{1}{2} \text{RTT}$). One-way propagation delay from the client to the MCU and from MCU to the client; TURN relay adds a routing detour on top.
- **MCU processing** (T_{mcu}). On the server, each incoming stream is held in a per-client jitter buffer. Readout begins only after an initial prefill of $2 N_f$ decoded samples. The mix clock computes the mix-minus sum every T_f and re-encodes the result for dispatch. This stage contributes a delay of $3 T_f$.
- **Client-side decoding and assembly** (t_{dec}). Opus decoding plus frame-slot assembly completes in under 1 ms when all 16 channels arrive within the same scheduler tick. The assembly timeout T_{asm} does not contribute additional latency to the end-to-end budget: assembled frames are forwarded to the playback worklet, where they sit in the jitter buffer until D_{pre} is reached, so any wait introduced by T_{asm} is masked by the concurrent prebuffer fill.
- **Jitter buffer prebuffer** (T_{pre}). This is the dominant configurable latency term. The adaptive loop (Section IV-A) keeps D_{pre} between $2 N_f$ and $6 N_f$ samples, corresponding to $2 T_f$ – $6 T_f$ of buffered audio.
- **Render quantum** (T_{rq}). The Web Audio API inherently processes the entire audio graph in synchronous, fixed 128-sample blocks ($T_{\text{rq}} \approx 2.67 \text{ ms}$ at 48 kHz). The algorithmic latency of the native audio graph evaluation, encompassing both the AudioWorklet node and the downstream Omnitone HRTF convolution in a single rendering pass, accounts for exactly one render quantum.
- **Output buffer** (T_{out}). The operating system’s audio subsystem buffers samples before delivering them to the output device. This term is platform-dependent, typically 2–10 ms.
- **Browser internal latency** (T_{br}). A small, implementation-dependent residual from inter-thread hand-offs, event-loop scheduling, and codec queue dispatch not captured by the other stages.
- **DAC conversion** (T_{dac}). Symmetric to the ADC, the digital samples are reconstructed to analog for the output device, with a delay of $\approx 0.5 \text{ ms}$.

The one-way latency can be described by this formula:

$$L = T_{\text{adc}} + T_{\text{cap}} + T_f + t_{\text{enc}} + \frac{1}{2} \text{RTT} + T_{\text{mcu}} + \frac{1}{2} \text{RTT} + t_{\text{dec}} + T_{\text{pre}} + T_{\text{rq}} + T_{\text{out}} + T_{\text{br}} + T_{\text{dac}} \quad (1)$$

To isolate the effect of the frame size, Table III reports the latency based on the sum $T_f + T_{\text{mcu}} + T_{\text{pre}} + T_{\text{rq}}$, for three representative frame sizes, at both the prebuffer floor

and ceiling. The environment-dependent terms (T_{cap} , $\text{RTT}/2$, T_{out}) are excluded, as they are common to all frame sizes.

TABLE III
LATENCY VS. FRAME SIZE (AT $f_s = 48 \text{ kHz}$)

N_f	T_f	T_{mcu}	Min: $D_{\text{pre}} = 2 N_f$	Max: $D_{\text{pre}} = 6 N_f$
120	2.5	7.5	17.7 ms	27.7 ms
480	10	30	62.7 ms	102.7 ms
960	20	60	122.7 ms	202.7 ms

The minimum column equals $6 T_f + T_{\text{rq}}$ (accumulation frame, three frames for server processing, two prebuffer frames, one render quantum), and the maximum equals $10 T_f + T_{\text{rq}}$. The prebuffer depth is therefore the dominant variable: at the default frame size of 20 ms, the difference between a stable connection (prebuffer at floor) and an unstable one (prebuffer at ceiling) is 80 ms – larger than the network round-trip contribution on most links.

B. Bandwidth

The aggregate Opus payload bitrate is fixed at $16 \times 64 = 1024 \text{ kbps}$ regardless of frame size. In the bundled format, the 16 channels are grouped into only a few DataChannel messages per frame following the layout of Section IV-C, and the four first-order channels are duplicated so that the base soundfield survives a single packet loss; this redundancy adds a fixed $4 \times 64 = 256 \text{ kbps}$. Each packet carries a 7-byte shared header plus per-channel length variants and incurs roughly 100 bytes of transport encapsulation.

Table IV reports the wire-level bitrate in one direction for three representative frame sizes.

TABLE IV
ESTIMATED WIRE BITRATE (ONE DIRECTION)

	20 ms	10 ms	2.5 ms
Opus payload	1,024 kbps	1,024 kbps	1,024 kbps
First-order redundancy	256 kbps	256 kbps	256 kbps
Bundle header overhead	24 kbps	26 kbps	102 kbps
Transport overhead	160 kbps	160 kbps	640 kbps
Total	~1.5 Mbps	~1.5 Mbps	~2.0 Mbps
Payload fraction	70%	70%	51%

For a bidirectional session, these figures double: approximately 3.0 Mbps at 20 ms and 4.0 Mbps at 2.5 ms. Because the MCU returns a single mix-minus stream to each participant, this per-client bandwidth is independent of the number of connected clients; only the server’s aggregate throughput grows, as the sum of one uplink per transmitting client (talker) and one downlink per connected client.

C. MCU Client Capacity

While the per-client bandwidth of the MCU is independent of the number of connected clients, the computational cost

TABLE V
MCU CLIENT CAPACITY VS. FRAME SIZE (64 KBPS/CH, 2 vCPU)

T_f (ms)	N_f (samples)	Encodes/client/s	Capacity	Range
2.5	120	6,400	3	2–3
5	240	3,200	7	5–9
10	480	1,600	8	6–11
20	960	800	14	12–15

of the mix-minus architecture grows with each additional participant. Every mix tick, the server must re-encode each client’s 16-channel mix-minus output, so the aggregate Opus encode load scales as $16 \times N$ encodes per frame period, where N is the number of active clients. To quantify this, the MCU was stress-tested with synthetic clients added in steps of one every 10 seconds on the same 2-vCPU virtual machine used in Section VI. The capacity limit was defined as the client count at which the maximum mix-clock tick jitter first exceeded one frame period.

Table V presents the median break point across multiple benchmark iterations per frame size. At the default 20 ms frame, the MCU sustained approximately 14 simultaneous active talkers. The results show that the MCU’s capacity is CPU-bound rather than limited by memory or network bandwidth. Each client added approximately 10 MB of resident memory, at the measured ceiling of 14 clients the aggregate memory footprint remained under 150 MB. Similarly, network bandwidth did not constrain the result: at 14 active talkers the MCU’s aggregate egress of approximately 21 Mbps (14×1.5 Mbps per-client mix-minus, per Table IV) represents a small fraction of typical datacenter link capacity.

VI. EVALUATION

To assess the behavior of the framework under realistic operating conditions, the system was evaluated using geographically distributed clients and conversational speech. The evaluation considers four aspects: Round-trip latency, resilience of the adaptive jitter buffer to packet delay variation and loss, network statistics, and signal and spatial coding fidelity. All measurements were collected at the application level.

A. Experimental Testbed

The testbed reproduced a multi-directional spatial audio session over the public Internet, with the central MCU hosted in a Frankfurt datacenter on a virtual machine running Ubuntu 24.04 LTS (x64) with two Intel vCPUs and 8 GB of RAM. Three browser clients connected to the MCU from distinct geographic locations and operating systems. Two of them acted as transmitting clients (talkers) equipped with a spherical Ambisonics microphone array, one located in London on a Windows device and one in Milan on macOS. The third was configured in receiver-only mode for validation and was located in Trento on macOS. All three clients established direct ICE connections to the MCU using a public STUN

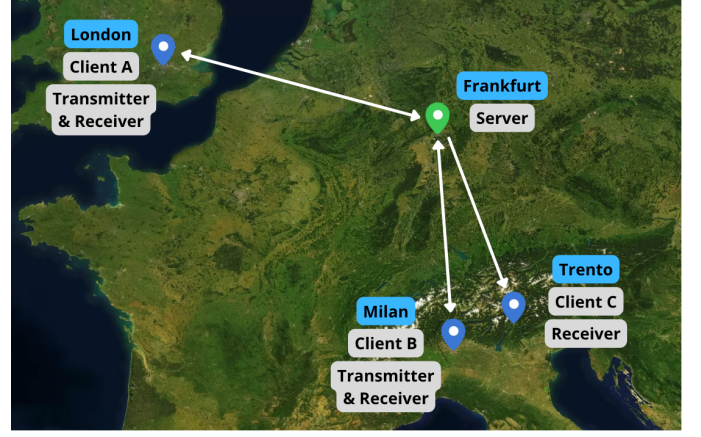


Fig. 2. Geographic distribution of the evaluation testbed.

server only; no TURN relay was interposed on the media path. The session used a uniform frame size of $N_f = 960$ samples (20 ms).

B. Evaluation Methodology

The system was tested exclusively with natural speech. Three sustained talk patterns were evaluated in sequence:

- 1) Client A transmitted while Clients B and C listened;
- 2) Client B transmitted while Clients A and C listened;
- 3) Clients A and B transmitted simultaneously while Client C listened.

Throughout each pattern, the client-side application logged round-trip time (RTT) from periodic DataChannel probes, packet delay variation (PDV), the active jitter buffer depth and the packet loss rate. Concurrently, the MCU’s system load average and network statistics were sampled with standard monitoring tools. The aggregate statistics reported below were extracted from the session reports.

C. Network Metrics

Table VI summarizes the per-client network metrics. The mean packet delay variation remained low at every client (1.6 ms, 5.0 ms, and 6.9 ms for Clients A, B, and C), and the resulting playback latency settled between 36.6 ms and 63.0 ms, within the predicted $[1.5 N_f, 6 N_f]$ envelope. Residual packet loss did not exceed 0.07 % at all clients despite the zero-retransmission DataChannel transport. Summing the terms for client B (Milan) – round-trip 90.5 ms, MCU processing 60 ms, playback prebuffer 63.0 ms, frame accumulation 20 ms, capture and output buffers ≈ 10 ms – yields a mouth-to-ear latency of approximately 250 ms.

D. Objective Spatial and Coding Fidelity

To quantify the impact of independent per-channel Opus encoding across the full tandem coding path, 16-channel test captures were evaluated using the synchronized recording pipeline (Sec. IV-F). Time-domain Signal-to-noise ratio (SNR) was measured per spherical harmonic component after delay

TABLE VI
MEASURED ROUND-TRIP, ONE-WAY LATENCY, JITTER BUFFER LATENCY
AND PACKET LOSS

Client	RTT	One-way	Buffer	Packet Loss
A - London	17.2 ms	8.6 ms	47.8 ms	0.07%
B - Milan	90.5 ms	45.2 ms	63.0 ms	0.06%
C - Trento	32.7 ms	16.3 ms	36.6 ms	0.02%

alignment and gain matching, and its mean decreases monotonically with Ambisonics order: 7.1 dB for Order 0 (W), 6.9 dB for Order 1 (5.1–8.2 dB), 4.9 dB for Order 2 (1.3–6.5 dB), and 4.3 dB for Order 3 (0.8–6.0 dB). This trend reflects the disproportionate effect of per-channel psychoacoustic quantization on low-energy, higher-order coefficients.

E. Informal Perceptual Assessment

The proposed framework was informally evaluated through a series of listening sessions involving the authors. The objective of these tests was not to perform a formal perceptual evaluation, but rather to verify the practical usability of the system during real-time communication.

Participants engaged in bidirectional conversations while listening to the received binaural rendering of the transmitted third-order Ambisonics sound field. Across repeated sessions conducted under the network conditions described in Section VI-A, speech remained intelligible and the perceived spatial characteristics of the sound scene were consistently preserved. In particular, the relative positions of the speakers were perceived as stable throughout the conversations, while no perceptually disturbing artifacts beyond the occasional effects expected from isolated packet losses were observed.

VII. DISCUSSION

A. Accessibility

A central motivation of this work is lowering the barrier to participation in immersive audio experiences, echoing the accessibility goals highlighted in the broader literature on immersive media [5], [6]. Current spatial audio streaming systems typically require dedicated native applications, proprietary codecs, or specialized hardware rendering infrastructure on the receiver side [15]. By contrast, the proposed framework requires only a web browser and a pair of headphones to receive and interact with a full third-order Ambisonics soundfield. On macOS, even the sender side requires no additional software beyond a DAW and a free virtual audio driver.

This zero-install deployment model has practical implications for several application scenarios identified in the literature. In remote participation for professional conferences and cultural events [5], attendees can join an immersive audio session by simply opening a URL, without needing to install platform-specific software or create user accounts. In educational settings [6], instructors can share spatial audio environments with students across heterogeneous devices. In networked music settings that do not require ultra-low latency,

the low barrier to entry can facilitate remote collaboration as well as support performance for a connected audience, which would otherwise require coordinating proprietary software installations across all participants.

The system also benefits from the browser’s built-in security and permissions model: microphone access is explicitly gated by a user consent prompt, encrypted transport is enforced by WebRTC’s mandatory DTLS/SRTP, and the sandboxed execution environment protects against the security risks inherent in installing native audio middleware.

B. Interoperability

The framework achieves cross-platform interoperability at two levels. At the application level, the same client-side JavaScript codebase runs unmodified on macOS, Windows, and Linux, with platform differences limited to the audio capture path (BlackHole, ReaRoute, or PipeWire). The signaling server, audio worklets, codec pipeline, and rendering subsystem are entirely platform-independent. At the signal level, the system adopts the AmbiX convention [21] (ACN ordering, SN3D normalization), which is the de facto standard for Ambisonics interchange. The 16-channel O3A signal is transmitted without proprietary spatial metadata or scene-description layers, ensuring that the captured soundfield can be rendered, recorded, and post-processed by any AmbiX-compatible toolchain. Furthermore, the transport architecture is built entirely on open W3C and IETF standards — WebRTC, DataChannels, WebCodecs, the Web Audio API [7], [8], and the Opus codec — rather than proprietary protocols. The system’s source code is publicly available under an open-source license⁸, which promotes reproducibility, community contributions, and independent verification of the results reported in this paper.

This combination of standard signal formats, open protocols, and cross-platform browser deployment stands in contrast with the dedicated software infrastructures assumed by previous networked immersive audio systems [15], [16], [18], and aligns with the vision of accessible, interoperable immersive audio ecosystems advocated by different scholars [1], [3].

C. Limitations

Various limitations of the current system should be acknowledged. First, as discussed in Section III, the requirement for multichannel `getUserMedia` capture restricts the fully bridge-free macOS path to Mozilla Firefox due to Chromium’s two-channel audio input limitation. While the Windows and Linux paths (which use an external Python bridge) are browser-agnostic, a single-browser requirement on one platform reduces the practical universality of the solution. This dependency will be resolved when Chromium and other browsers implement multichannel audio capture in the future.

Second, all 16 encoded Opus streams share a single DataChannel. Under severe network congestion, packet loss affects all channels uniformly. A potential optimization would

⁸<https://github.com/CIMIL/WebRTC-Ambisonics-Streaming>

assign separate DataChannels or priority levels to different Ambisonics orders, since lower-order components (particularly the omnidirectional W channel) carry more perceptual weight than higher-order components and could benefit from preferential treatment.

Third, each of the 16 Opus streams is masked independently, and a second tandem generation of independent coding occurs at the MCU's re-encode stage. The resulting quantization noise is uncorrelated across spherical harmonic components — unlike joint HOA codecs such as IVAS [17] — and disproportionately degrades the lower-energy higher-order components after binaural matrixing, consistent with Section VI-D's SNR trend.

Fourth, the spatial rotation of the soundfield is currently controlled through manual sliders of the browser User Interface or via a MetaMotionRL BLE sensor that streams orientation quaternions to the browser. The system does not integrate head tracking through the W3C Device Orientation API or WebXR Device API, limiting its applicability to fully immersive VR scenarios. Additionally, the MCU sums soundfields channel-wise without per-client rotation, so co-located talkers collapse to the same perceived direction. Per-source rotation or virtual seating at the MCU is left for future work.

Finally, the perceptual quality of the rendered binaural output has not been formally evaluated in this study. While the system preserves the full O3A signal chain from capture to rendering, a rigorous perceptual assessment would require controlled listening tests with spatial audio quality metrics, which we leave to future work.

VIII. CONCLUSION

This work demonstrates that modern Web standards have reached a level of maturity enabling browser-native immersive audio systems previously achievable only with native software. The proposed system showed that browser-native third-order Ambisonics streaming is technically feasible with latency compatible with conversational interaction using current web platform capabilities. By combining WebCodecs, WebRTC DataChannels, and the Web Audio API, the proposed system achieves real-time 16-channel spatial audio transport and binaural rendering without requiring any software installation, plugin, or proprietary infrastructure on the receiver side. The framework is open-source, cross-platform, and built entirely on open standards, contributing to the broader goal of making immersive audio experiences accessible to any user with a web browser and a pair of headphones.

A formal perceptual evaluation comparing the proposed system with native Ambisonics streaming pipelines, as well as with lower-order representations (first-order, stereo), would quantify the trade-offs between spatial fidelity, computational cost, and network bandwidth, following methodologies adopted in the literature (e.g., [12], [13]).

REFERENCES

- [1] J.-M. Jot, R. Audfray, M. Hertensteiner, and B. Schmidt, "Rendering spatial sound for interoperable experiences in the audio metaverse," in *2021 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*. IEEE, 2021, pp. 1–15.
- [2] F. Zotter and M. Frank, *Ambisonics: A practical 3D audio theory for recording, studio production, sound reinforcement, and virtual reality*. Springer, 2019.
- [3] A. Boem, M. Tomasetti, and L. Turchet, "Issues and challenges in audio technologies for the musical metaverse," *Journal of the Audio Engineering Society*, vol. 73, pp. 94–114, 2025.
- [4] C. Rinaldi, C. Centofanti, L. Comanducci, M. Pezzoli, A. Bernardini, A. Marotta, and F. Antonacci, "Immersive acoustics via next-generation networks: Achieving high-fidelity audio in the metaverse," *IEEE Open Journal of the Communications Society*, 2025.
- [5] M. Montagud, P. Orero, and A. Matamala, "Culture 4 all: accessibility-enabled cultural experiences through immersive vr360 content," *Personal and Ubiquitous Computing*, vol. 24, no. 6, pp. 887–905, 2020.
- [6] F. C. Rodríguez, M. Dal Peraro, and L. A. Abriata, "Democratizing interactive, immersive experiences for science education with webxr," *Nature Computational Science*, vol. 1, no. 10, pp. 631–632, 2021.
- [7] B. Smus, *Web Audio API: Advanced Sound for Games and Interactive Apps*. O'Reilly Media, Inc., 2013.
- [8] P. Adenot, C. Wilson, and C. Rogers, "Web Audio API," *W3C*, October, vol. 10, 2013.
- [9] M. Buffa, S. Ren, O. Campbell, T. Burns, S. Yi, J. Kleimola, and O. Larkin, "Web Audio Modules 2.0: An Open Web Audio Plugin Standard," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 364–369.
- [10] M. Buffa, D. Girard, and A. Hofr, "Using web audio modules for immersive audio collaboration in the musical metaverse," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2024, pp. 1–10.
- [11] M. Tomasetti, A. Boem, and L. Turchet, "How to Spatial Audio with the WebXR API: a comparison of the tools and techniques for creating immersive sonic experiences on the browser," in *Proceedings of Immersive and 3D Audio: from Architecture to Automotive (I3DA)*. IEEE, 2023, pp. 1–9.
- [12] A. Boem, S. Mazzei, and L. Turchet, "Spatial audio for webxr: Perceptual evaluation of sound localization technologies on the browser," in *2025 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*. IEEE, 2025, pp. 1–9.
- [13] F. Miotello, P. Ostan, G. Elli, C. Centofanti, M. Pezzoli, A. Bernardini, C. Rinaldi, and F. Antonacci, "Latency and quality of experience in immersive networked music performances using higher-order ambisonics spatial audio," in *2025 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*. IEEE, 2025, pp. 1–9.
- [14] P. Cairns, H. Daffern, and G. Kearney, "Immersive network music performance: Evaluation of the effect of auralized reverberation time," *Journal of the Audio Engineering Society*, 2025.
- [15] P. Ostan, C. Centofanti, M. Pezzoli, A. Bernardini, C. Rinaldi, and F. Antonacci, "Dynamic real-time ambisonics order adaptation for immersive networked music performances," in *33rd European Signal Processing Conference*. IEEE, 2025, pp. 1258–1262.
- [16] 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 *IEEE 5th International Symposium on the Internet of Sounds*. IEEE, 2024, pp. 1–10.
- [17] C. Hold, D. Weckbecker, G. Fuchs, M. Multrus, R. Tyagi, S. Brown, J. Torres, and S. Bruhn, "Coding Higher Order Ambisonics in 3GPP IVAS—Scaling Parametric Audio Coding to Higher Bitrates," in *33rd European Signal Processing Conference*. IEEE, 2025, pp. 196–200.
- [18] L. Turchet, C. Rinaldi, C. Centofanti, L. Vignati, and C. Rottondi, "5G-Enabled Internet of Musical Things Architectures for Remote Immersive Musical Practices," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 4691–4709, 2024.
- [19] C. Rinaldi, F. Franchi, A. Marotta, F. Graziosi, and C. Centofanti, "On the exploitation of 5G multi-access edge computing for spatial audio in cultural heritage applications," *IEEE Access*, vol. 9, pp. 155 197–155 206, 2021.
- [20] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, "An overview on networked music performance technologies," *IEEE Access*, vol. 4, pp. 8823–8843, 2016.
- [21] C. Nachbar, F. Zotter, E. Deleflie, and A. Sontacchi, "Ambix-a suggested ambisonics format," in *Ambisonics Symposium*, vol. 2011, 2011.

[1] J.-M. Jot, R. Audfray, M. Hertensteiner, and B. Schmidt, "Rendering spatial sound for interoperable experiences in the audio metaverse," in