

# ENABLE: An Engine-Native ASIO Bridge for Low-Latency Multichannel Audio Exchange in XR Deployments

Nicușor Amarie  
Transilvania University of Brașov  
Brașov, Romania  
nicusor.amarie@unitbv.ro

Daniel Calaver  
Transilvania University of Brașov  
Brașov, Romania  
daniel.calaver@unitbv.ro

Vlad Popescu  
Transilvania University of Brașov  
Brașov, Romania  
vlad.popescu@unitbv.ro

Mattia Mura  
Università di Cagliari  
Cagliari, Italy  
m.mura135@studenti.unica.it

Gianluca Fadda  
Università di Cagliari  
Cagliari, Italy  
gianluca.fadda@unica.it

Maurizio Murrone  
Università di Cagliari  
Cagliari, Italy  
maurizio.murrone@unica.it

**Abstract**—Real-time audio in XR engines normally relies on the engine’s built-in backend, which can limit access to professional multichannel devices; reaching an ASIO interface therefore often involves external routing or additional integration. We present ENABLE (Engine-Native ASIO Bridge for Low-latency Exchange), an architecture that surfaces a professional ASIO interface within the engine’s audio pipeline rather than through external routing. ENABLE couples the ASIO driver callback to the engine audio callback through a wait-free single-producer/single-consumer ring buffer, exposes multichannel input selection and input-to-output monitor mappings, and addresses the independent hardware and engine clock domains with an asynchronous sample-rate converter driven by a fill-level control loop, making local buffering explicitly configurable. The ASIO bridge core is isolated from the host engine by an adapter; reuse beyond the Unity reference implementation remains to be validated. ENABLE forms the hardware capture and monitoring stage of a larger networked spatial-audio pipeline for hybrid performance: it is integrated with a WebRTC LAN transport with software clock-offset estimation and is designed to integrate with a hardware-clocked Audio-over-IP layer (e.g., AES67/PTP) in production, providing the multichannel hardware input that a downstream live-encoding stage consumes. An initial virtual-ASIO run supports software-path feasibility; objective end-to-end latency and professional-hardware performance remain unmeasured.

**Index Terms**—ASIO, multichannel audio capture, engine-native audio plugin, real-time audio, wait-free ring buffer, asynchronous sample-rate conversion, clock synchronisation, Audio-over-IP, WebRTC

## I. INTRODUCTION

Extended reality (XR) systems increasingly combine spatial rendering, live performance, networked interaction, and real-time media capture inside a single engine runtime. Game engines provide mature graphics, animation, physics, and spatial audio abstractions, but their default audio backends are usually designed around consumer playback and operating-system-managed devices. This creates a practical mismatch when

an XR deployment must ingest many low-latency channels from a professional interface, route selected inputs to hardware outputs, and expose the resulting signals to the same runtime that controls avatars, interaction, and spatialisation.

In common production workflows, professional audio hardware is integrated indirectly. Signals may be routed through a digital audio workstation, a virtual audio cable, a loopback driver, or an external network-audio application before they reach the engine. These approaches are usable, but they can introduce hidden buffering, additional configuration steps, and configuration-dependent channel-count limitations that are difficult to measure from inside the engine. They also separate audio routing from the XR scene state, which is problematic when the capture path must be spatialised, analysed, or synchronised with networked visual behaviour.

ENABLE addresses this gap by exposing Audio Stream Input/Output (ASIO) hardware directly inside the engine audio pipeline. ASIO is a professional driver model commonly used for low-latency communication between applications and audio interfaces [1]. ENABLE uses the ASIO driver callback as the hardware-side producer and the engine audio callback as the consumer. The two callbacks are intentionally decoupled by a bounded single-producer/single-consumer (SPSC) ring buffer, while an asynchronous sample-rate conversion (ASRC) stage adapts the hardware clock domain to the engine mixer clock domain.

The engineering objectives are to expose device channels and monitor routes under runtime control, decouple the hardware and engine callbacks despite their independent clocks, and make buffer behaviour observable through reproducible diagnostics. Low latency is a design objective; the initial evaluation addresses functional feasibility rather than a demonstrated performance advantage over external routing.

The paper makes the following contributions:

- an engine-native architecture for direct ASIO input and

- output access inside an XR runtime;
- a wait-free SPSC capture buffer that avoids per-sample locking in ring-buffer transfers;
- an ASRC stage driven by a fill-level control loop to compensate drift between hardware and engine clocks;
- configurable multichannel input and output mappings, including direct ASIO input-to-output monitoring;
- a validation scenario in which the bridge supplies audio to a LAN WebRTC spatial-audio prototype, preparing the same capture stage for future hardware-clocked Audio-over-IP integration.

The remainder of the paper is organised as follows. Section II reviews the relevant background: engine audio backends, ASIO, clock-domain mismatch, and networked audio transport. Section III describes the ENABLE architecture and implementation. Section IV discusses latency, deployment implications, and the relationship between the ASIO bridge and the WebRTC validation pipeline. Section V summarises limitations and future work.

## II. BACKGROUND

### A. Audio in XR engines

XR engines expose high-level components for playback, spatialisation, mixing, and effects. These abstractions are appropriate for asset-based content, but they normally sit above a platform audio backend chosen by the engine or operating system. The developer is therefore not always able to select the professional driver model, channel count, hardware buffer size, or clocking behaviour required by live multichannel deployments.

Unity, for example, provides audio extension points such as audio source callbacks, mixer processing, and native audio plug-ins [2]. These APIs make it possible to generate or process audio within the engine, but they do not by themselves provide a professional multichannel ASIO hardware endpoint. ENABLE is positioned below the spatial and interaction logic of the engine: it supplies an engine-visible stream of hardware samples and receives engine-selected output routes without requiring an external routing application.

Existing solutions provide important building blocks: NAudio exposes ASIO recording and playback [3], Jack Audio For Unity connects engine channels to an external JACK server [4], and Virtual Audio Cable connects applications through virtual endpoints [5]. ENABLE itself uses NAudio, rather than introducing a new hardware driver. Its contribution is the combined engine integration of callback ownership, SPSC buffering, fill-controlled ASRC, monitor routing, and diagnostics. Other solutions could implement these mechanisms; the distinction is their explicit integration here, not exclusive access to ASIO or a measured latency advantage.

### B. ASIO callback model

ASIO exposes a low-latency driver interface between an application and a professional audio device [1]. In an ASIO application, the driver invokes a callback whenever a new hardware buffer is available. The application must finish its

work within the callback deadline, otherwise the device may underrun or drop samples. This places strict constraints on the callback path: it should avoid blocking operations, memory allocation, and locks that can be contended by another thread.

In an engine integration, the ASIO callback and the engine audio callback are not the same callback. They may run at different buffer sizes, on different threads, and under different clock domains. Directly performing engine operations inside the ASIO callback would violate engine-thread ownership rules and increase the risk of missed audio deadlines. ENABLE therefore treats ASIO as a real-time producer and the engine audio callback as a real-time consumer.

### C. Clock-domain mismatch

Even when two audio systems are configured for the same nominal sample rate, their clocks are not identical. A hardware ASIO interface may produce buffers according to its own oscillator, while the engine mixer consumes audio according to the platform output clock. If the producer is slightly faster than the consumer, a fixed-size buffer will gradually fill until it overflows. If the producer is slower, the buffer will drain until it underruns.

A large ring buffer can hide this mismatch temporarily, but it does not solve it. ENABLE instead makes the mismatch explicit. The ASIO stream is written into a bounded buffer, then read through an ASRC stage that adjusts its read ratio according to the buffer fill level. In this model, the buffering setpoint is explicit, although achieved latency and regulation still require measurement.

### D. Networked audio context

Hybrid XR performance often requires live audio to leave the local machine. Two families of network transport are relevant. WebRTC provides an accessible real-time media stack with congestion control, jitter buffering, and statistics exposed through the WebRTC stats model [6]. It is convenient for LAN prototypes and remote interaction experiments. However, WebRTC is not designed as a sample-accurate, hardware-clocked professional audio fabric.

Audio-over-IP standards such as AES67 define interoperable high-performance audio transport over IP networks [7]. In production systems, clock distribution is commonly tied to Precision Time Protocol (PTP), specified by IEEE 1588 [8]. ENABLE does not replace these network layers. Instead, it provides the engine-native hardware I/O stage that such layers require: multichannel capture, monitoring, and routing inside the XR runtime.

Recent work has addressed adjacent aspects of this problem space, including low-latency network music performance with edge-assisted synthesis [9], NMP over 5G networks [10]–[12], networked immersive-audio scenarios in the musical metaverse [13], and low-delay multimodal transmission for hyperscanning experiments [14]. ENABLE is complementary to these works: it focuses on the local engine-native professional hardware I/O stage that can feed such downstream networked audio systems.

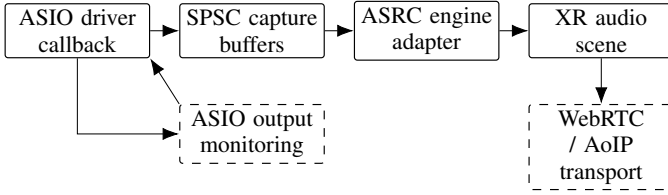


Fig. 1. ENABLE separates hardware capture, engine playback, hardware monitoring, and downstream transport. Monitoring uses the driver’s input block, not the capture ring; AoIP integration is future work.

The broader Musical Metaverse vision [15] also raises audio hardware, processing, and interoperability challenges, discussed by Boem, Tomasetti, and Turchet [16]. Buffa et al. investigate reusable Web Audio Modules for immersive collaboration [17], while Schlagowski et al. examine the experience of remote musicians in mixed reality [18]. ENABLE addresses a complementary prerequisite: making live instrument channels available to the engine that renders the shared musical scene.

### III. SYSTEM DESCRIPTION

#### A. Design overview

ENABLE is organised around three boundaries: the ASIO hardware boundary, the engine audio boundary, and the optional network transport boundary. The core bridge owns the ASIO device, receives input buffers from the driver, exposes output buffers for hardware monitoring, and maintains per-channel ring buffers. A thin engine adapter reads from the bridge inside the engine audio callback and presents the samples to ordinary engine audio objects. A separate network module may consume the same engine-visible streams for WebRTC or Audio-over-IP transport.

The architecture follows four design goals. First, the hardware callback path must remain real-time safe. Second, multi-channel access must be explicit rather than hidden behind the operating system’s default device. Third, clock drift between the audio interface and the engine must be compensated. Fourth, the bridge should not be tied to one XR engine; only the adapter layer should depend on engine-specific APIs.

#### B. ASIO bridge core

The bridge core is responsible for driver enumeration, device startup, sample-rate selection, channel discovery, input capture, output buffer submission, and status reporting. In the Unity reference implementation, a manager component exposes the selected ASIO driver, requested channel count, sample rate, ring-buffer duration, level meters, and runtime start/stop controls. The bridge itself is isolated from scene logic: the engine sees a component-level API for reading channels and setting monitor routes, rather than calling the driver directly.

For each ASIO input buffer, the bridge obtains interleaved samples from the driver callback. It then deinterleaves each channel into a corresponding SPSC ring buffer using block writes. Metering values are computed in a separate traversal,

and meter state is stored separately from the audio transfer path. Output monitoring is handled by configurable routes from selected input channels to selected output channels, with optional duplication to a second output channel for headphones or stereo monitoring. This path copies the driver’s input block directly to its output buffers; it is not a general Unity-mix-to-ASIO output backend.

#### C. Wait-free SPSC buffer

Each input channel is represented by a bounded SPSC ring buffer. The producer is uniquely the ASIO callback; the consumer is uniquely the engine audio callback or a script reading from the bridge API. Because there is only one producer and one consumer, the buffer does not require a mutex. ENABLE uses monotonic read and write cursors published through volatile memory operations. The producer writes sample blocks and publishes the new write cursor after the block has been copied. The consumer reads available samples and publishes the new read cursor after consumption.

The wait-free property refers to these ring transfers: they contain no mutex or retry loop and their work is bounded by block length. It does not describe the complete managed callback, which still includes a metering lock and scratch-buffer allocations at startup or size changes.

Let  $w$  be the producer cursor,  $r$  the consumer cursor, and  $N$  the ring capacity in samples. At each transfer, the consumer-visible occupancy is computed as the non-negative cursor distance  $w - r$ , saturated at  $N$ ; the writable free space is the remaining capacity. This keeps the shared state limited to the two monotonic cursors and avoids a shared mutable sample count.

If a producer block contains more samples than the available free space, the bridge drops the excess incoming samples and increments an overrun counter. If the consumer requests more samples than the current occupancy, the missing tail is zero-filled and an underrun counter is incremented. These counters count affected samples, not callback events. They are exposed for diagnostics and allow the deployment to distinguish stable operation from buffer slack masking a clock or scheduling problem.

#### D. Adaptive sample-rate conversion

The ASRC stage is located in the engine adapter. Its role is to present a stream at the engine output sample rate while reading from the ASIO source sample rate. The reference implementation uses a local FIFO on the engine audio thread, a fractional source read pointer, and linear interpolation between adjacent source samples. Let  $f_s$  be the ASIO source rate and  $f_e$  the engine output rate. The nominal read ratio is:

$$\rho_0 = \frac{f_s}{f_e}. \quad (1)$$

The controller adjusts this ratio by a correction term  $c$  expressed in parts per million:

$$\rho = \rho_0 \left( 1 + \frac{c}{10^6} \right). \quad (2)$$

The correction is derived from the local FIFO fill error. If  $B$  is the FIFO fill in source samples after refill and before consumption, and  $B_t$  is the target fill, the normalised error is:

$$e = \frac{B - B_t}{B_t}. \quad (3)$$

ENABLE uses a proportional-integral controller, updating the integral before computing the correction:

$$I \leftarrow \text{clip}(I + K_i e \Delta t, -C_{\max}, C_{\max}), \quad (4)$$

$$c = \text{clip}(K_p e + I, -C_{\max}, C_{\max}). \quad (5)$$

Here  $B_t$  is the target FIFO fill, i.e., the buffering setpoint around which the ASRC tries to stabilise the stream. The proportional gain  $K_p$  controls the immediate read-ratio response to the current fill error, while the integral term  $I$ , updated through  $K_i$ , compensates slower long-term clock drift between the ASIO device and the engine output clock. The correction is expressed in parts per million (ppm), and  $C_{\max}$  clamps the maximum allowed resampling adjustment. In the Unity reference implementation used for the measurement in Table I, these runtime parameters were set to  $B_t = 3840$  samples (80 ms at 48 kHz),  $K_p = 1200$  ppm,  $K_i = 100$  ppm/s, and  $C_{\max} = 3000$  ppm; the latter corresponds to a maximum read-ratio change of approximately 0.3%. Here  $\Delta t$  is the output callback duration in seconds and clip denotes saturation to the stated interval.

These prototype settings balance buffering reserve against responsiveness: the 80 ms target leaves nominal margin relative to the 10.67 ms ASIO block, while a 10% fill error gives 120 ppm of immediate correction and accumulates a further 10 ppm per second. The correction limit bounds ratio excursions, not end-to-end latency. Startup fill is separately set to 40 ms, local FIFO capacity to 500 ms, and maximum bridge read block to 20 ms; capacity is not queued delay. The settings have not been optimised through a sensitivity or perceptual study.

When the FIFO fill rises above the target, the read ratio increases slightly and the ASRC consumes source samples faster. When the fill drops below the target, the read ratio decreases. This is intended to regulate buffering around the target; successful regulation depends on source availability, callback timing, and the correction range.

#### E. Multichannel input and output mapping

ENABLE exposes channel selection independently from the engine's default audio device. A component can read one ASIO channel, mix a set of ASIO channels to mono, or route selected input channels directly back to ASIO outputs. The monitoring path is useful when a performer must hear a hardware input with minimal additional processing, while the engine still observes the same input for metering, spatialisation, recording, or network transmission.

The mapping model is intentionally simple. Input channels are identified by zero-based indices corresponding to the ASIO

device's exposed channel order. Output channels are also selected by index. A route therefore consists of an input index, one or two output indices, a gain value, and enable/disable state. This representation is sufficient for common XR performance setups while remaining compatible with devices that expose different channel counts.

#### F. Unity reference implementation

The reference implementation targets Unity on Windows. An `ASIOManager` component owns the bridge instance and exposes driver selection, capture start/stop, sample rate, channel count, ring-buffer size, diagnostics, and VU meters. An `ASIOChannelPlayer` component attaches the capture stream to a `Unity AudioSource`. The player can run in direct mode or ASRC mode; ASRC is the default path for stable engine playback when the hardware and engine clocks diverge.

Additional helper components provide device dropdowns, channel dropdowns, status labels, level meters, and editor setup menus. These helpers are not part of the core architecture, but they make the bridge usable in executable builds and repeatable in experimental deployments. The core bridge remains separated from scene code so that another engine can replace the Unity adapter without changing the ASIO callback, ring buffer, or routing model.

Specifically, the bridge has no Unity API dependencies, but relies on .NET, NAudio, and Windows ASIO. The manager, streaming callback, ASRC, and UI currently reside in Unity components. Another engine would require a compatible bridge host or interop layer, an audio/lifecycle adapter, and extraction or porting of the ASRC, followed by validation. Engine independence therefore describes the separation of responsibilities, not demonstrated portability.

#### G. WebRTC LAN validation pipeline

The WebRTC component is not the definition of ENABLE; it is a validation and integration scenario. In the prototype, ENABLE supplies the local hardware capture and monitoring stage. The engine can then send selected audio streams over a LAN WebRTC connection while logging transport statistics such as connection state, selected candidate pair, round-trip time, jitter, packet counts, packet loss, and byte rates. A software clock-offset estimator is used to relate remote transport timing to local playback timing.

This integration pipeline illustrates how the engine-native ASIO stage can feed a networked spatial-audio system without requiring virtual cables or an external DAW. It also separates two latency domains: local hardware-to-engine latency, affected by the ASIO buffer, SPSC buffer, and ASRC fill; and network latency, affected by WebRTC packetisation, jitter buffering, network scheduling, and endpoint clocks. The local measurement below does not benchmark WebRTC; software clock-offset estimation neither synchronises hardware sample clocks nor measures one-way audio latency.

## H. Measurement instrumentation

ENABLE exposes diagnostics at three layers. At the ASIO layer, it reports active driver name, active sample rate, channel count, available samples per channel, dropped samples, underruns, and per-channel peak/RMS levels. At the ASRC layer, it reports FIFO fill, correction ppm, read ratio, and ASRC underrun count. In the WebRTC validation pipeline, it logs transport metrics available from the WebRTC statistics model, including packet loss, jitter, round-trip time, bitrate, and candidate information [6].

The purpose of the instrumentation is twofold. First, it supports deployment tuning: buffer sizes and ASRC targets can be adjusted while observing underruns and correction values. Second, it supports research reproducibility: local audio behaviour and network transport behaviour can be logged separately rather than collapsed into a single subjective latency observation.

## IV. DISCUSSION

### A. Local latency decomposition

ENABLE makes local latency easier to reason about because each stage is explicit. The hardware contributes the ASIO device buffer and driver scheduling delay. The bridge contributes the current SPSC fill level. The ASRC contributes residence time in its local FIFO, which is not necessarily equal to the configured target fill. The engine contributes its audio callback buffer and output backend latency. A simplified local capture-to-playback latency estimate can therefore be written as:

$$L_{\text{local}} \approx L_{\text{ASIO}} + L_{\text{ring}} + L_{\text{ASRC}} + L_{\text{engine}}. \quad (6)$$

This expression is not a substitute for hardware measurement, but it provides a practical model for tuning. A lower ASRC target is intended to reduce buffering at the expense of safety margin for scheduling jitter and clock drift; a higher target makes the opposite trade-off. Actual queue occupancy and endpoint delays must still be measured. The important distinction is that the settings are now exposed inside the engine rather than hidden in an external routing chain.

### B. Why ASIO remains separate from WebRTC

The WebRTC validation layer and the ASIO bridge solve different problems. ASIO provides low-latency access to a local professional audio interface. WebRTC transports media over a packet network and introduces its own jitter buffering, congestion control, and endpoint timing. Combining the two into a single undifferentiated block would make it difficult to identify whether latency or instability is caused by hardware capture, engine buffering, network jitter, or receiver buffering.

ENABLE therefore treats WebRTC as a downstream consumer of engine-visible audio. This design is important for research. It allows the same ASIO capture stage to be tested locally, with direct hardware monitoring, and then reused in a networked experiment. It also allows a future WebRTC transport to be replaced by AES67/PTP or another Audio-over-IP layer without rewriting the local capture path.

### C. Use in XR performance

In hybrid performance, the performer may require direct monitoring, while the XR audience requires spatialised audio embedded in the scene. ENABLE supports both paths. The same hardware input can be routed directly to a headphone output for monitoring and simultaneously read by the engine for spatialisation or network transmission. Because the engine receives the signal as an ordinary audio stream, it can attach the sound to a moving avatar, apply spatial attenuation, or log audio-derived features alongside scene state.

This is especially relevant when the visual and acoustic representation of a performer must remain coupled. External routing can deliver audio into the engine, but it obscures the relationship between audio buffers and engine timing. ENABLE exposes that relationship and provides metrics that can be correlated with frame timing, network statistics, and user movement.

### D. Research value

The proposed bridge is not only a production convenience. It also provides a controlled platform for studying distributed musical interaction in XR. By separating local capture latency, engine latency, and network latency, experiments can compare different transport choices and buffering policies while keeping the hardware I/O path constant. This is useful for evaluating whether a given XR network condition is suitable for remote rehearsal, call-and-response performance, audience monitoring, or tightly synchronised ensemble playing.

For the Musical Metaverse, the practical contribution is to preserve individual instrument channels as scene sources while retaining a separate performer-monitoring path. This supports microphone-to-avatar assignments and spatial production within the runtime, addressing the local integration challenges discussed in [16]. Improved co-presence, perceived quality, or ensemble synchrony would require musician studies beyond the present software validation.

### E. Initial virtual-ASIO measurement

As an initial functional measurement, ENABLE was evaluated on a laptop using Voicemeeter Virtual ASIO as the ASIO endpoint and the built-in microphone as the source signal. The goal of this run was not to characterise the round-trip latency of a professional hardware interface, but to verify that the engine-native capture path, SPSC buffer, ASRC stage, Unity audio callback, and CSV instrumentation operated continuously once the ASIO driver had started.

The CSV contains 772 snapshots over the recorded interval, with input channel 1 selected for playback and a nominal 100 ms logging interval. Counter values in Table I are increases during that interval, not totals since startup; the result does not evaluate simultaneous multichannel scaling.

The results in Table I show that the Unity audio clock advanced consistently with wall-clock time and that no additional SPSC drops, SPSC underrun samples, or ASRC underrun samples were counted during the 80 s run. The non-zero input and output peak/RMS values confirm that the microphone

TABLE I  
INITIAL VIRTUAL-ASIO MICROPHONE STABILITY RUN

| Metric                          | Measured value           |
|---------------------------------|--------------------------|
| Test duration                   | 80.041 s                 |
| Unity DSP time advance          | 80.032 s                 |
| ASIO driver                     | Voicemeeter Virtual ASIO |
| ASIO sample rate                | 48 kHz                   |
| ASIO callback buffer            | 512 samples / 10.67 ms   |
| Unity DSP buffer                | 256 samples $\times$ 4   |
| ASRC target fill                | 80 ms                    |
| SPSC dropped samples, increase  | 0                        |
| SPSC underrun samples, increase | 0                        |
| ASRC underrun samples, increase | 0                        |
| Maximum input peak / RMS        | 0.239 / 0.103            |
| Maximum output peak / RMS       | 0.194 / 0.085            |
| ASRC fill, mean                 | 14.12 ms                 |
| ASRC fill, min–max              | 0.79–85.42 ms            |

signal reached the ASIO bridge and was emitted by the Unity audio path. The ASRC controller remained active during the run. Its underrun counter was already 506 samples at the first snapshot and remained 506 at the last, hence the zero increase reported here. Because the test used a virtual ASIO device, these values should be interpreted as a software-path stability check rather than a final hardware latency benchmark.

The mean ASRC fill of 14.12 ms is below the 80 ms target and does not establish target convergence. The controller reads fill before consumption, while the published value is also updated after rendering; the main-thread logger is not synchronised to either phase. The field also excludes upstream ring residence. Phase-defined telemetry and longer controlled-drift tests are needed to explain this difference; the reported fill is not an end-to-end latency measurement.

## V. LIMITATIONS AND FUTURE WORK

The current implementation has several limitations. First, the reference implementation targets Windows ASIO devices. This is appropriate for many professional audio interfaces, but it does not directly cover macOS Core Audio or Linux JACK/ALSA deployments. The architecture is engine-agnostic at the bridge boundary, but additional native backends would be required for cross-platform professional I/O.

Second, the ASRC implementation uses linear interpolation. This is sufficient for drift compensation and prototype evaluation, but future versions should evaluate higher-quality interpolation or polyphase resampling, especially when the correction range is large or when the output is used for critical listening.

Third, the WebRTC LAN transport is a validation prototype rather than a production-grade hardware-clocked audio network. WebRTC is useful because it provides accessible real-time media transport and rich diagnostics, but its jitter buffer and congestion-control behaviour are not equivalent to an AES67/PTP system. Future work should connect ENABLE to a hardware-clocked Audio-over-IP layer and compare WebRTC timing against PTP-disciplined transport.

Fourth, the present paper reports the architecture and instrumentation rather than a complete cross-device benchmark

matrix. Objective end-to-end latency, professional-interface performance, and a quantitative comparison with virtual cables or external routing have not been measured. The virtual-device run itself depends on Voicemeeter and cannot demonstrate their elimination or a latency advantage. Future evaluation should include loopback latency measurement, long-duration drift tests, underrun/overrun stress tests, channel-count scaling, CPU usage, and controlled LAN/5G transport measurements. These experiments should distinguish local hardware-to-engine latency from network send-to-receive latency and from spatial-audio rendering latency.

Finally, the current implementation assumes a single producer and a single consumer per capture channel. This matches the real-time audio path, but systems that require multiple independent consumers should introduce fan-out stages after the engine adapter rather than reading the same SPSC buffer from multiple threads.

In summary, ENABLE contributes a reusable architectural pattern for integrating professional multichannel audio into real-time XR engines, combining explicit callback ownership, clock adaptation, monitoring, and diagnostics. The Unity implementation demonstrates local feasibility; measured latency, sustained regulation, and validation in other engines remain the next steps.

## ACKNOWLEDGMENT

The research activities described in this paper were carried out within the “HEAT – Hybrid Extended reAliTy” Project GA 101135637 funded by the EU Horizon Europe Framework Programme (HORIZON) and supported by a grant of the Romanian Ministry of Research, Innovation and Digitization, CNCS/CCCDJ - UEFISCDI, project number PN-IV-P8-8.1-PRE-HE-ORG-2025-0331, within PNCDI IV.

## REFERENCES

- [1] Steinberg Media Technologies GmbH, “3rd-Party Developers Support & SDKs,” accessed Jul. 2026. [Online]. Available: <https://www.steinberg.net/developers/>
- [2] Unity Technologies, “Native audio plug-in SDK,” Unity 6 Manual, accessed Sep. 2026. [Online]. Available: <https://docs.unity3d.com/6000.0/Documentation/Manual/AudioMixerNativeAudioPlugin.html>
- [3] NAudio contributors, “Recording with ASIO,” NAudio documentation, release/2.x, accessed Sep. 2026. [Online]. Available: <https://github.com/naudio/NAudio/blob/release/2.x/Docs/AsioRecording.md>
- [4] Jack Audio For Unity contributors, “Jack Audio For Unity,” software repository and documentation, accessed Sep. 2026. [Online]. Available: <https://github.com/rodrigodzf/Jack-Audio-For-Unity>
- [5] E. Muzychenko, “Virtual Audio Cable user manual,” accessed Sep. 2026. [Online]. Available: <https://vac.muzychenko.net/en/manual/>
- [6] World Wide Web Consortium, “Identifiers for WebRTC’s Statistics API,” W3C specification, accessed Jul. 2026. [Online]. Available: <https://www.w3.org/TR/webrtc-stats/>
- [7] Audio Engineering Society, “AES67-2023: AES standard for audio applications of networks – High-performance streaming audio-over-IP interoperability,” accessed Jul. 2026. [Online]. Available: <https://aes.org/publications/standards-store/>
- [8] IEEE Standards Association, “IEEE 1588-2019: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,” accessed Jul. 2026. [Online]. Available: <https://standards.ieee.org/standard/1588-2019.html>

- [9] L. Borgianni, C. Bua, D. Adami, and S. Giordano, "Reconstructing the Sound at the Edge: A Low-Latency Framework for NMP with Autotuned Local Synthesis," in *Proc. 2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*, 2025, pp. 1–8, doi: 10.1109/IS264627.2025.11284618.
- [10] K. Tsioutas and G. Xylomenos, "Network Music Performance in 5G Networks," in *Proc. 2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*, 2025, pp. 1–6, doi: 10.1109/IS264627.2025.11284565.
- [11] A. Carôt, M. Dohler, S. Saunders, F. Sardis, R. Cornock, and N. Uniyal, "The world's first interactive 5G music concert: Professional quality networked music over a commodity network infrastructure," in *Proc. 17th Sound and Music Computing Conference*, 2020, pp. 407–412, doi: 10.5281/zenodo.3898918.
- [12] 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, doi: 10.1109/OJCOMS.2024.3407708.
- [13] C. Rinaldi and C. Centofanti, "The Musical Metaverse: Advancements and Applications in Networked Immersive Audio," in *Proc. 2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, pp. 1–7, doi: 10.1109/IS262782.2024.10704154.
- [14] G. Grimm, M. Daeglau, V. Hohmann, and S. Debener, "EEG Hyperscanning in the Internet of Sounds: Low-Delay Real-Time Multi-Modal Transmission Using the OVBOX," in *Proc. 2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, pp. 1–8, doi: 10.1109/IS262782.2024.10704205.
- [15] L. Turchet, "Musical Metaverse: Vision, opportunities, and challenges," *Personal and Ubiquitous Computing*, vol. 27, pp. 1811–1827, 2023, doi: 10.1007/s00779-023-01708-1.
- [16] A. Boem, M. Tomasetti, and L. Turchet, "Issues and challenges of audio technologies for the musical metaverse," *J. Audio Eng. Soc.*, vol. 73, no. 3, pp. 94–114, 2025, doi: 10.17743/jaes.2022.0193.
- [17] M. Buffa, D. Girard, and A. Hofr, "Using Web Audio Modules for immersive audio collaboration in the musical metaverse," in *Proc. 2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, pp. 1–10, doi: 10.1109/IS262782.2024.10704108.
- [18] R. Schlagowski, D. Nazarenko, Y. Can, K. Gupta, S. Mertes, M. Billingham, and E. André, "Wish you were here: Mental and physiological effects of remote music collaboration in mixed reality," in *Proc. CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–16, doi: 10.1145/3544548.3581162.