

Nexus Muse: a Technical Framework for Interactive Networked Music Systems using Universal MIDI Packets

Gregorio Andrea Giudici, Filippo Daniotti, Alberto Boem, and Luca Turchet

Department of Information Engineering and Computer Science

University of Trento

gregorio.giudici@unitn.it, filippo.daniotti@unitn.it, alberto.boem@unitn.it, luca.turchet@unitn.it

Abstract—MIDI 1.0 has been the main industry standard for digital musical instruments since the 1980s, but its limitations have led practitioners to often adopt ad-hoc protocols for networked music systems. This practice undermines interoperability and communication across systems, resulting in severe fragmentation. To address this issue, we present Nexus Muse, a technical framework for developing co-located interactive networked music systems. Built upon Universal MIDI Packets (UMP) and the Network MIDI 2.0 transport extension, the framework ensures seamless compatibility with MIDI 1.0 while fully supporting the advanced capabilities of MIDI 2.0. The framework comprises three components: the Muse Cube embedded platform, a Unity plugin for developing cross-platform interactive networked applications, and a JUCE template for extending hard real-time audio applications with network capabilities. We evaluate latency and jitter across different configurations, and demonstrate the framework’s versatility through five representative use cases spanning XR, AI-assisted performance, and mobile interaction. The results provide evidence that UMP and Network MIDI 2.0 constitute a practical communication foundation for interoperable co-located interactive networked music systems, illustrating how open standards can support diverse musical applications without resorting to bespoke protocols.

Index Terms—Internet of Musical Things, Embedded Computing, Network MIDI 2.0, Musical XR

I. INTRODUCTION

Recent advances in networking and music technologies have given rise to several paradigms for musical interaction over networks, from real-time remote performance to co-located ensembles to location-independent systems [1]–[4]. These paradigms can be unified under the *Internet of Musical Things (IoMusT)*, which broadly refers to the network of interconnected musical devices and systems that communicate and share data over networks, enabling multi-directional musical interactions between musicians, audiences, and interactive devices across co-located or remote networks [5]. By enabling interconnections between Musical Things, these approaches facilitate radically new performance ecosystems encompassing musical stakeholders and machines. Consequently, a range of systems and protocols have emerged to interconnect heterogeneous musical devices over co-located or remote networks (e.g., [6]–[11]).

However, in this context, researchers have mainly focused on addressing issues such as *latency* and *reliability*. Latency

is the critical adversarial factor that affects musicians’ quality of experience in networked performance [12], [13]. End-to-end delay is the most significant parameter preventing synchronization between distributed performers, with sensitivity thresholds varying based on musical context [14]–[16]. Many low-latency solutions employ transmission based on User Datagram Protocol (UDP) to minimize delay compared to Transmission Control Protocol (TCP), but this design choice inherently sacrifices reliability, since packets are delivered without guarantees of arrival, ordering, or recovery. To address these issues, a wide range of mechanisms has been proposed, including packet loss concealment algorithms at the audio level [17]–[19] and error-resilience algorithms for symbolic streams, such as recovery journals, redundancy and selective retransmissions [20]–[22].

While Musical Instrument Digital Interface (MIDI) [23] has served as the de facto industry standard for musical technologies and Digital Musical Instruments (DMI) since the 1980s, its historical limitations (such as reduced transmission bandwidth and low resolution) have led many practitioners to supplement or replace it with more general-purpose protocols [24], [25]. Consequently, the majority of networked music systems available today have mostly implemented ad-hoc solutions, often based on Open Sound Control (OSC) [26] due to its simplicity and extensibility [27]–[30].

However, the resulting systems are often difficult to extend or interconnect with other platforms and consumer devices. This situation has led to a fragmentation of the technological ecosystem and undermines the IoMusT principles of *ubiquity*, *interoperability*, and *standardization* [5]. Overcoming fragmentation, therefore, calls not for yet another bespoke protocol, but for convergence on a shared standard that is interoperable by design and compatible with existing musical hardware.

In 2023, the MIDI Association and the Association of Musical Electronics Industry proposed Universal MIDI Packet (UMP) as the new transport format [31], ensuring seamless integration and backward compatibility with existing MIDI 1.0, while enabling next-generation features of MIDI 2.0 to address the well-known limitations of MIDI 1.0. UMP represents an industry standard that DMI manufacturers and music

technology companies are progressively adopting, with major manufacturers gradually updating their products to comply with it. This industry-wide adoption is crucial for enabling seamless interoperability across systems and establishing a unified communication standard for both manufacturers and artists. Despite the growing adoption of UMP, its potential as the communication backbone for interactive networked music systems has received little attention.

Motivated by these developments and the need to address fragmentation, this paper presents **Nexus Muse**, a technical framework for developing interactive networked music systems based on UMP. The framework, primarily intended for co-located interactions, leverages Network MIDI 2.0 as the transport protocol for network communication.

This paper makes the following contributions:

- 1) The design and implementation of the Nexus Muse framework, with all hardware, software, and documentation available as open-source and freely accessible online;
- 2) Comprehensive evaluation of latency and jitter with different configurations;
- 3) Five illustrative use cases showing the framework’s versatility;
- 4) Empirical insights into the potential of standardized UMP-based network protocol to enhance interoperability and reduce fragmentation in IoMusT ecosystems.

II. BACKGROUND

A. Networked Systems for Sound and Music

Interactive networked music systems can be approached from different perspectives, depending on their design goals and use cases. Audio-based systems transmit raw audio data, requiring high bandwidth and imposing strict latency constraints [32]–[34]. In contrast, symbolic-based systems transmit abstract musical representations such as MIDI messages or control parameters, which require significantly lower bandwidth while maintaining precise timing [35].

The network infrastructure also distinguishes different system categories. Local Area Network (LAN) systems operate over wired Ethernet connections, providing deterministic latency and high reliability with minimal packet loss [36]. Wireless LAN (WLAN) systems, on the other hand, offer mobility and flexibility through WiFi networks but introduce more variable jitter and are more susceptible to packet loss due to interferences [37].

Another important distinction is between remote and co-located performance. With remote networked performance, we refer to musicians playing from different geographical locations connected over the internet. Here, the end-to-end latency depends on distance and network infrastructure [38]. The quality of experience is highly sensitive to delay, with musicians experiencing synchronization challenges above 20–30 ms [39]. Differently, co-located networked performance involves musicians present in the same physical space connected over a local network, with latency typically below 20 ms, enabling more natural ensemble interaction [40], [41].

Finally, architectures can be centralized or distributed. Centralized architectures employ a hub or server that routes all communication between devices, simplifying synchronization but introducing a single point of failure. Distributed (or peer-to-peer) architectures allow devices to communicate directly without a central node, providing scalability but requiring complex synchronization mechanisms.

B. Network Protocols for Symbolic-based Systems

The MIDI 1.0 protocol has long been the standard for connecting physical controllers to sound synthesis and processing engines. Its most notable network extensions are RTP-MIDI and ipMIDI. RTP-MIDI transmits MIDI over UDP using Real-Time Protocol (RTP) framing and recovers lost packets through an optional journaling mechanism, achieving high reliability [42]. The operating systems of Apple natively supports it as AppleMIDI, while Windows and Linux require third-party drivers. In addition, RTP-MIDI inherits the semantics—and the limitations—of MIDI 1.0, and it lacks the MIDI Capability Inquiry feature.

On the other hand, ipMIDI is a simpler implementation that forwards raw MIDI 1.0 messages directly over UDP without RTP framing and reliability mechanisms. It requires a simpler setup while achieving lower latency compared to RTP-MIDI, but sacrifices reliability, thus making it unsuitable for performances over unstable network infrastructures.

Despite MIDI being the industry standard, many practitioners have preferred OSC because of its simplicity and flexibility. OSC’s primary advantage lies in its flexible structure (i.e., a URL-style naming scheme that can accommodate arbitrary data structures). However, OSC lacks built-in timing mechanisms and time-stamping precision required for rhythmically synchronized networked performances. Many extensions of OSC have been proposed to address these limitations. One notable example is *O2* [10], which provides better routing, type safety, and structured message handling while maintaining similar latency compared to vanilla OSC.

Nevertheless, despite these extensions, OSC-based protocols lack standardized semantic meaning for musical parameters, requiring custom agreement between devices on message naming and data interpretation. Furthermore, a file format for storing and sharing OSC data—similar to the Standard MIDI File—has not yet been standardized. These gaps directly hinder interoperability in heterogeneous networked music systems. A complementary approach was proposed by *MUSEPA* [11], a semantic web architecture for discovery, interoperability, and automatic inference in IoMusT ecosystems; it targets discrete events such as device discovery and reconfiguration rather than continuous real-time interaction, and can thus complement—but not replace—protocols like MIDI and OSC.

Because many workflows use both MIDI and OSC, bridging tools such as libmapper [27] and TouchOSC have emerged to convert between the two formats, allowing MIDI controllers to interface with OSC-enabled applications. These conversion layers, however, add setup complexity and can introduce performance overhead.

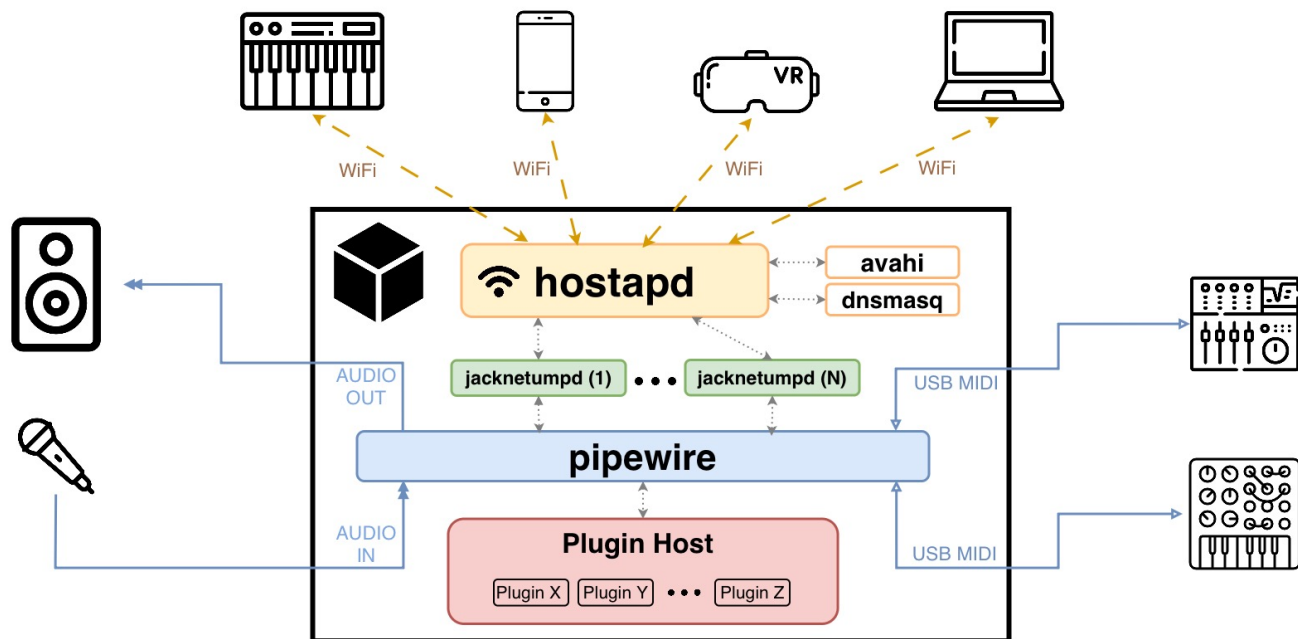


Fig. 1: Example diagram of a system developed using the Nexus Muse framework, with the Muse Cube embedded platform serving as central communication hub.

C. Universal MIDI Packet and Network MIDI 2.0

Although MIDI 1.0 has endured for over four decades, its design reflects the technological constraints of its 1983 origins: a 31.25 kBaud serial DIN transmission rate, 7-bit resolution (0–127 values, roughly 0.8% quantization), and 16 channels that limit routing, multi-expression polyphony, and device addressing.

To overcome these constraints, MIDI 2.0 introduces the first major overhaul of the MIDI standard [31]. Key improvements include 32-bit resolution (approximately $2 \times 10^{-10}\%$ quantization), 256 channels (16 groups times 16 channels), MIDI Capability Inquiry (MIDI-CI) for automated property exchange and profile negotiation, and standardized device profiles for interoperability across manufacturers.

Delivering these capabilities requires a new transport. UMP is the new high-speed transport format for MIDI 2.0, designed for USB, Ethernet, and WiFi rather than legacy 5-pin DIN. UMP features variable-length packets of 32, 64, 96, or 128 bits, dual protocol support where Group 0 carries MIDI 1.0 stream for legacy compatibility and Groups 1–15 carry MIDI 2.0 stream, and embedded timestamps for precise multi-note rendering and optional jittering reduction. Crucially, UMP ensures seamless integration and backward compatibility with existing MIDI 1.0 ecosystems.

While UMP defines the packet format, transporting it over a network is the role of Network MIDI 2.0, a UDP-based transport for low-latency, reliable communication in distributed musical systems [43]. It specifies session management—device discovery, capability exchange, and connection establishment—where discovery, achieved via mDNS/DNS-

SD [44], [45], enables plug-and-play interoperability without manual configuration.

To remain robust over UDP, Network MIDI 2.0 includes Forward Error Correction (FEC) to recover lost packets without retransmission, selective retransmission for critical messages such as Note On, Note Off, and Program Change events, and sequencing and time-stamping to preserve event ordering and timing accuracy across the network.

III. PROPOSED TECHNICAL FRAMEWORK

The Nexus Muse framework provides a flexible, modular technical backbone for building co-located networked music systems, possibly centered on an embedded platform with integrated access-point capabilities and audio/MIDI I/O (see Fig. 1). It comprises three components: (1) the *Muse Cube*, a reference hardware/software platform built on a consumer embedded computer running open-source software, which serves as the central hub; (2) a *Unity plugin* for developing cross-platform interactive networked software; and (3) a *JUCE template* for extending hard real-time audio applications with network capabilities.

Although Nexus Muse is presented as a cohesive framework, its three main components are designed to operate independently. This modularity allows developers and artists to adopt only the elements that best fit their use case: for example, the Muse Cube can function as a standalone Network MIDI 2.0 hub that interoperates with third-party devices and software not developed within this framework, while the Unity plugin and JUCE template can be used with any network infrastructure, whether wired or wireless.

A key aspect of the proposed framework is the dual role of the Muse Cube platform, which acts as both the network provider and the central processing hub. First, as a wireless access point, the Muse Cube enables the rapid deployment of a dedicated and self-contained network. On activation, it automatically creates a WLAN to which other devices (such as head-mounted displays or smartphones) connect directly, with no external router or internet connection required. This streamlines setup, reduces configuration overhead, and makes the system well suited to portable, ad-hoc scenarios such as rehearsals, live performances, and installations. Second, as the central processing hub, the Muse Cube hosts audio processing tasks, manages MIDI routing, and interfaces with external instruments. Consolidating networking and computation in a single embedded device reduces system complexity and enables fully self-contained deployments, where both communication and processing happen locally.

This self-contained design also decouples the system from shared or managed infrastructure, allowing developers to define and reproduce consistent networking conditions while avoiding traffic contention, firewall restrictions, and protocol incompatibilities. Therefore, this approach provides a controlled, reproducible operational context that is particularly valuable for real-time musical interaction.

A. Muse Cube: Hardware

The Muse Cube is built on a Raspberry Pi 5, which provides sufficient processing power for both networking and audio/MIDI handling. Two Hardware Attached on Tops (HATs) extend its capabilities. The HiFiBerry DAC2 ADC Pro HAT delivers high-quality audio I/O at up to 192 kHz/24-bit resolution, while the PCIe HAT hosts a WiFi 6E module based on the Intel AX210 chip, with two omnidirectional antennas, adding 802.11ax connectivity on the 6 GHz band for higher throughput than the Raspberry Pi's built-in WiFi 5. Two USB 3.0 and two USB 2.0 interfaces connect external USB-MIDI devices such as synthesizers and sequencers, and a USB-C adapter supplies power. A custom 3D-printed case encloses the assembly (see Fig. 2).

We designed the Cube with a minimalist, unobtrusive form factor so that it integrates easily into performance spaces, studios, or educational settings without occupying excessive space or requiring complex setup. The audio and WiFi modules were selected to provide a self-contained baseline configuration, avoiding the need for additional external devices for audio I/O and wireless connectivity. In particular, this reduces the cabling, power supplies, placement requirements, and configuration steps otherwise associated with using an external audio interface or a router connected through Ethernet. The resulting version is therefore a compact default solution that prioritizes portability and ease of deployment, especially in portable or ad-hoc performance contexts.

At the same time, the Cube remains modular and extensible. Users may connect an external audio interface through USB, a dedicated router or access point through Gigabit Ethernet, and additional MIDI devices through the available USB ports.

Thus, the integrated configuration is not intended to preclude alternative audio or network topologies, but rather to offer a readily deployable starting point that can be adapted to the requirements of a specific performance, installation, or educational setting.

The assembly uses commercial, readily available components rather than custom PCBs or prototypes, allowing anyone to build their own Muse Cube and, more importantly, to customize it (e.g., extending the number of inputs/outputs or the network capabilities) to suit their specific needs.

B. Muse Cube: Software

The software stack running on the Muse Cube is built on open-source components that work together:

- **linux**: The embedded operating system powering the Muse Cube platform, providing real-time audio and MIDI I/O, and network stack support;
- **RaspAP**¹ and **hostapd**²: Create the Muse Cube's wireless access point, enabling other devices to directly connect to its network without relying on external infrastructure;
- **dnsmasq**³: Lightweight DNS and DHCP server assigning network addresses and enabling device discovery on the Cube network;
- **avahi**⁴: Implements Bonjour/mDNS service discovery, allowing other applications to automatically find the Network MIDI 2.0 endpoints without manual configuration;
- **pipewire**⁵: Modern audio/MIDI server, managing low-latency audio processing and routing between plugins and network MIDI interfaces;
- **jacknetumpd**⁶: Network MIDI 2.0 daemon translating UMP over UDP for local music processing;
- **Host with various audio plugins**: The central audio processing node, receiving MIDI commands, and producing routed audio output.

We prepared a GitHub repository⁷ to streamline the set up process of a new Muse Cube. The repository contains: detailed instructions on how to set up the required software components; references and assets to replicate the hardware configuration; five examples of system-specific configurations for the Muse Cube, mirroring the Use Cases we present in Section VI.

C. Unity Plugin

Unity⁸ is a popular cross-platform game engine and real-time 3D development environment used mainly for the development of mobile and XR applications. For this reason,

¹<https://raspap.com>

²<https://w1.fi>

³<https://thekelleys.org.uk/dnsmasq/doc.html>

⁴<https://avahi.org>

⁵<https://pipewire.org>

⁶<https://github.com/CIMIL/jacknetumpd>

⁷<https://github.com/CIMIL/nexus-cube-platform>

⁸<https://unity.com/>

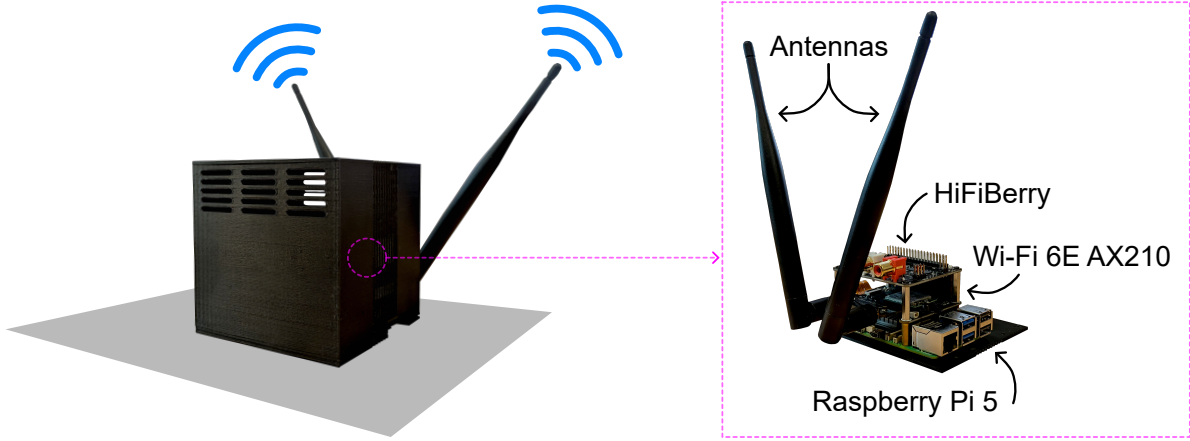


Fig. 2: Assembled Muse Cube embedded platform.

we developed a cross-platform Unity plugin⁹ that enables developers to integrate Network MIDI 2.0 features into Unity applications and develop software for different platforms, such as Windows, and Android-based devices.

The plugin is implemented as a C# wrapper around the open-source cross-platform *NetUMP* C++ library¹⁰, which provides the actual network communication and session handling. This architecture ensures that performance-critical operations, such as UDP transport, FEC encoding/decoding, and session management, are handled in optimized native code, while providing a convenient C# API for Unity developers.

The plugin is designed for ease of use, with comprehensive documentation and example scenes demonstrating common use cases. Developers can simply import the plugin into their Unity project and use the provided components without needing to understand the underlying network protocol. The plugin is used in the scenarios presented in Section VI, particularly for XR environments and smartphone or tablet-based control surfaces.

D. JUCE Template

We developed a JUCE template¹¹ that can be used to easily integrate Network MIDI 2.0 into software developed with the JUCE framework. JUCE¹² is the current standard framework for hard real-time audio software because it provides cross-platform Standalone and plugin support (e.g., VST3 and AU), along with a comprehensive set of optimized audio DSP classes, and exhaustive documentation and tutorials.

The template is not just a documentation form, but a complete working codebase that includes pre-implemented classes that use the *NetUMP* library, and streamline enhancing already stable audio plugin with network functionalities.

We used JUCE for the networked MIDI keyboard and the AI-improviser system presented in the use scenarios (see Section VI).

IV. TECHNICAL VALIDATION

This work aims to assess the versatility and interoperability of the proposed framework, rather than the features of the underlying UMP and MIDI 2.0 protocols or the networked interactive systems built with it. Therefore, instead of evaluating the usability or user experience of the resulting applications, this paper assesses the proposed framework from three complementary perspectives.

First, we experimentally characterize the Muse Cube's performance through latency and jitter measurements under different networking configurations. Second, we demonstrate the framework's architectural flexibility by implementing five representative proof-of-concept use cases spanning distributed musical control, AI-assisted performance, and XR interaction. Third, we analyze the resulting architecture with respect to interoperability, modularity, and deployability, discussing its strengths, limitations, and applicability to future IoMusT systems.

V. LATENCY AND JITTER EVALUATION

To evaluate the hardware and software architectural decisions of the Cube platform, we performed tests on latency and jitter with several network topologies and configurations. Tests were conducted along two main comparison axes:

- (1) AX210 vs. Broadcom (Raspberry Pi 5 built-in chipset);
- (2) Software access point vs. external WiFi router;

Ethernet measurements were also included as a lower bound, representing the best achievable performance. The experiments were conducted in a controlled office environment at the University of Trento. For the external WiFi router condition, we used *unim-x*, the university's enterprise-grade internal network infrastructure.

It should be noted that the external WiFi condition does not represent a dedicated, non-shared router connected directly to

⁹https://github.com/CIMIL/NetUMP_Unity

¹⁰<https://github.com/CIMIL/NetUMP>

¹¹https://github.com/CIMIL/Networked_JUCE_Template

¹²<https://juce.com>

the Cube’s Ethernet port. Rather, it reflects the performance of the available enterprise network infrastructure in the tested environment, which may introduce additional routing, contention, and management overhead. Therefore, the comparison is intended to evaluate the proposed self-contained software access point configuration against an existing shared infrastructure, rather than to establish that it universally outperforms all possible Ethernet-connected router deployments.

Latency was assessed by measuring the Round-Trip Time (RTT) using the `ping` utility. Specifically, 1000 Internet Control Message Protocol (ICMP) packets were transmitted with a 10 ms interval. On the other hand, we evaluated UDP jitter using the `iperf3` utility. In these tests, 160-byte UDP datagrams were sent at a target bitrate of 64 Kbps for 600 seconds, with jitter recorded at 1 s intervals.

Descriptive statistics are reported in Table I and Table II: for each test configuration, the best result along comparison axis (1) is indicated in **bold**, whereas the best result along comparison axis (2) is underlined.

TABLE I: RTT statistics (in milliseconds) measured using `ping` across different network configurations.

TEST Configuration	mean	std	min	max
Client ↔ Cube via Ethernet	0.457	0.076	0.289	0.724
Client ↔ Cube via Cube (AX210)	2.113	0.286	1.676	5.489
Client ↔ Cube via Cube (built-in)	27.431	40.475	1.617	191.625
Client ↔ Client via Cube (AX210)	19.134	33.181	3.770	206.864
Client ↔ Client via Cube (built-in)	30.484	38.015	3.811	138.308
Client ↔ Client via unitn-x	<u>11.431</u>	<u>20.460</u>	4.467	<u>138.546</u>

TABLE II: UDP jitter statistics (in milliseconds) measured using `iperf3` across different network configurations.

TEST Configuration	mean	std	median	min	max
Client ↔ Cube via Ethernet	0.046	0.008	0.044	0.033	0.065
Client ↔ Cube via Cube (AX210)	0.095	0.074	0.071	0.024	0.639
Client ↔ Cube via Cube (built-in)	3.031	3.695	2.110	0.585	30.064
Client ↔ Client via Cube (AX210)	0.519	2.125	0.275	0.104	24.944
Client ↔ Client via Cube (built-in)	4.038	4.371	2.542	0.472	29.162
Client ↔ Client via unitn-x	2.755	4.526	0.763	0.201	<u>24.084</u>

The results highlight a consistent performance advantage of the AX210 interface over the built-in Broadcom chipset across all evaluated scenarios. In the Client ↔ Cube via Cube configurations, the AX210 achieves an RTT mean approximately one order of magnitude lower than the built-in interface, along with substantially better stability (standard deviation below 1 ms compared to roughly 40 ms). A similar—though less pronounced—trend is observed in the Client ↔ Client via Cube scenario, where performance differences remain noticeable but more comparable. This behavior is also reflected in the UDP jitter measurements, providing strong evidence that the AX210 offers a more stable and predictable connection—an essential requirement for real-time operation, as is the case in our platform.

The comparison between the software access point and the external WiFi infrastructure yields more nuanced results.

During direct communication in Client ↔ Cube scenarios, the software AP outperforms the external network by approximately one order of magnitude in both mean and standard deviation of RTT, as well as in UDP jitter metrics. This outcome is expected, as communication involves a single hop. However, in the Client ↔ Client scenario, the external WiFi network exhibits lower RTT values, within the same order of magnitude as the software AP. Despite this, the software AP reports mean and standard deviation of UDP jitter by an order of magnitude. The slightly lower RTT observed for ICMP packets may suggest that the enterprise router provides a lower baseline latency than the software-based AP, although the max value indicates the presence of outliers, thus making the distribution skewed. However, when network infrastructure overhead is introduced, as in the UDP jitter test, the software AP exhibits lower jitter, thus greater stability, likely due to its role as a dedicated network.

VI. USE CASES

The following use cases are intended as illustrative demonstrations of the proposed framework rather than comprehensive evaluations of the resulting applications. Their purpose is to show how the same UMP-based infrastructure can support heterogeneous interactive musical systems with minimal application-specific modifications.

A. Centralized Networked Performance Hub

This use case presents the Muse Cube platform as a centralized performance hub that aggregates conventional MIDI hardware and software instruments into a single networked setup. A performer can connect legacy MIDI 1.0 devices (such as synthesizers, drum machines, and sequencers) via USB-MIDI to the Cube, which also hosts the internal plugin host for virtual instruments and effects. At the same time, smartphones, laptops, and tablets on the local wireless network can act as controllers or visual clients, so musicians can drive the instruments attached to the Cube without being physically cabled to them or forced to remain near the hardware.

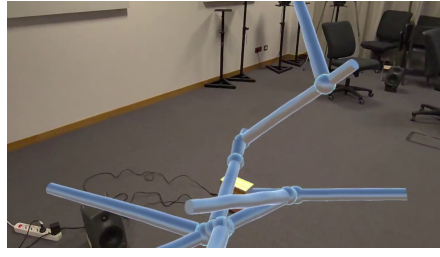
At the protocol level, the Cube exposes a series of Network MIDI 2.0 endpoints and routes both MIDI 1.0 and MIDI 2.0 messages. MIDI 1.0 streams from “older” instruments are encapsulated into UMP and exchanged alongside native MIDI 2.0 data, without requiring custom translation layers or ad-hoc glue protocols. This scenario therefore illustrates one of the framework’s core goals: enabling backward-compatible interoperability while providing a path toward higher-resolution and more expressive MIDI 2.0 workflows within a unified, co-located performance environment.

B. Distributed Control Surfaces

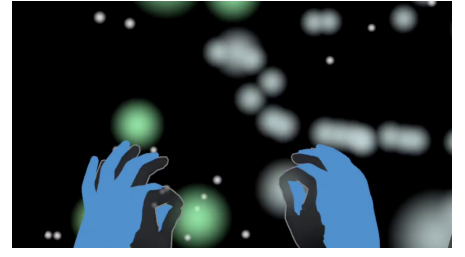
This use case shows the framework supporting a networked instrument whose control surface is distributed across several devices rather than confined to a single one. A musician plays a network-enhanced MIDI keyboard, while a companion application running on a consumer Android smartphone or tablet acts as an auxiliary controller (built with Unity 6.3 and



(a) **Distributed Control Surfaces:** an Android-based smartphone application and a networked MIDI keyboard both interact with a shared sound engine.



(b) **Immersive Visualization:** a real-time 3D visualization created by MIDI data.



(c) **XR MIDI Controller:** a 3D particle system is used as a MIDI controller for a synthesizer running inside the Cube.

Fig. 3: Three use cases developed using the Nexus Muse.

the NetUMP Unity plugin). Both components are wirelessly connected to the Cube platform and drive the same shared sound engine (see Fig. 3a).

The motivation is twofold: to expand the set of expressive parameters available to a performer beyond the physical controls of one instrument, and to enable collaborative play in which one participant plays notes while another shapes the timbre all without additional hardware beyond a device already present on the local network.

The networked MIDI keyboard is built on a Studiologic SL73 mk2 MIDI keyboard connected to a headless Raspberry Pi 5, which hosts a JUCE-based application¹³ developed from the provided template, and serves as the primary performance interface to the sound engine hosted in the Cube.

In contrast, the smartphone application is built with Unity and exercises the output direction of the library, complementing the receive-only visualizations described later in this section. Its touchscreen interface presents a set of faders and knobs; dragging each maps a normalized position to an assigned Control Change number, such as CC 4 (Expression). Each update is packed into a MIDI 1.0 Channel Voice message carried in a UMP word and transmitted to the Cube, which routes it to the plugin host for audio rendering.

In practice, a performer can mount the tablet beside the keyboard and ride expression with a free hand, or pass it to a second musician who modulates the sound engine in real-time while the first plays, realizing a collaborative interconnected instrument from commodity devices.

C. Musician-AI Partnership System

This use case demonstrates that the framework can support musical interaction between human performers and AI agents distributed across the same local network. A musician performs on the networked MIDI controller, while an AI-based improvisation system running on the Muse Cube listens to the same symbolic stream and generates responses in real-time. The Cube routes both human and machine-generated events to a shared sound engine, so the two participate in a single musical interaction loop.

In our prototype, the AI agent is implemented as a networked improviser based on a Markov-chain model described in [46], which consumes incoming Note On/Off messages and emits new sequences over the same transport. The agent is built using the JUCE template to extend an open source application¹⁴. Although the present implementation uses a relatively simple symbolic model, the architecture generalizes to more advanced machine learning systems, including neural sequence models or hybrid symbolic-audio agents, which can be integrated as additional endpoints without changing the underlying protocol. This underlines a central premise of the framework: because every participant, human or algorithmic, communicates through the same standardized UMP transport, new agents can be added to a performance as first-class endpoints rather than as one-off integrations, directly counteracting fragmentation.

D. Immersive Visualization

This use case demonstrates the framework as the backbone of a real-time visualization of MIDI data using an HMD, that can be experienced by audience members of a co-located performance or as a Mixed Reality (MR) installation. The result is a virtual sculpture that assembles in real-time in front of the audience, where sustained low notes become long descending limbs and bright staccato notes become short upward shoots (see Fig. 3b). A musician plays a networked MIDI instrument, like the one presented in Section VI-B. The MIDI data generated are streamed to an head-mounted display (Meta Quest 3) running an application developed with Unity 6.3 and the NetUMP Unity plugin. The application uses MIDI data to generate a growing 3D organic sculpture. Each Note On sprouts a tubular branch whose length, direction, and duration are mapped from velocity, pitch, and hold time respectively. Note Off then freezes and caps the branch. Control Change messages additionally drive global appearance parameters such as hue and emission, so a second performer or the player continuously modulates the visual. The rendering algorithm used in this example is inspired by the *Growth Model* of Yoichiro Kawaguchi [47], [48]. Rather than a transient an-

¹³https://github.com/CIMIL/Networked_SoundFont_Player

¹⁴https://github.com/CIMIL/Networked_AI_Improviser

imation, the structure accretes note by note into a persistent form, functioning as a kind of visual score of the performance.

Because the visualization is purely a consumer of the MIDI stream, it requires no awareness of the underlying network beyond subscribing to the same Network MIDI 2.0 endpoint: the Cube’s access-point and routing roles make the keyboard, the sound engine, and the headset a single co-located system with the low and stable latency reported in Section V, and the visualization can be added or removed from a performance without touching the audio signal chain.

E. XR MIDI Controller

While the previous scenario illustrated the use of XR systems for the visualization of external MIDI data, this use case focuses on an interactive and bidirectional approach. In this example system, performers wearing a HMD can generate MIDI notes with their hands while immersed in a cosmos of living particles (see Fig. 3c). Using the hand tracking of an HMD (Meta Quest 3), a performer can grab a single particle which is converted into a MIDI Note On (hand height and pinch are mapped to notes and velocity, lateral motion is mapped to a Control Change) and transmitted back to the Cube. The Cube then renders the corresponding sound from its plugin host. Each grabbed particle spawns an attractor whose strength, radius, and hue are mapped from velocity and note number, while lateral hand motion (sent as Control Change) modulates both the sound engine and the field’s ambient parameters at once, so the same expressive control is felt across both modalities. Visitors see a dark volumetric space of drifting points; grabbing or pinching condenses a swirling colored nebula around the hand, and multiple hands build a polyphonic, multi-colored cosmos that dissolves back into ambient drift on release.

The application is built with Unity 6.3 and the NetUMP Unity plugin, and produces a particle field of ≈ 600 particles running at 90 Hz on a standalone Meta Quest 3.

This scenario points toward technology-mediated audience participation, where several visitors collectively shape the musical state of an instrument hosted inside the Cube, and toward collaborative configurations in which one participant plays a keyboard while others sculpt the sound and the field through control messages—with every participant, gestural or physical, addressed through the same standardized transport.

VII. DISCUSSION

The design choices underlying Nexus Muse, together with the use cases presented in Section VI, highlight both the strengths and the current boundaries of the framework.

One key strength lies in the loose coupling between the components of the framework. Rather than requiring musicians and developers to adopt the full stack each layer can be used independently, provided it complies with the Network MIDI 2.0 and UMP standards. At the protocol level, this means that clients not built with the provided library and template can still connect to the Cube out of the box. For instance, recent versions of macOS expose native Network

MIDI 2.0 session support in Audio MIDI Setup¹⁵, allowing any compatible device to join a Cube-hosted session without any framework-specific software. This is a direct consequence of building on an open, manufacturer-backed standard rather than a bespoke transport, and it is precisely the property that ad-hoc OSC-based systems typically lack.

This openness extends to the Cube’s hardware and software layers as well. The internal architecture of the Cube is not a fixed appliance but a customizable embedded platform: additional audio inputs, MIDI DIN connectors, or other peripherals can be integrated as a given performance or installation requires, without altering the underlying routing logic. Similarly, the Unity plugin and the JUCE template do not presuppose a Cube-centric topology. Both components can be deployed in a conventional home or studio network using an off-the-shelf router. In this sense, the Cube is a convenient, pre-configured instantiation of the framework rather than a mandatory hub, allowing adoption to scale from lightweight, ad-hoc setups up to installation-grade deployments such as the one described in Section VI-E.

Taken together, these properties show that Nexus Muse achieves interoperability by design through standards compliance and modularity rather than proprietary interfaces. While this work does not claim to resolve the broader fragmentation of the IoMusT ecosystem, it shows that UMP and Network MIDI 2.0 provide a practical and technically viable foundation for building interoperable and co-located musical systems.

It is worth noting that the five use cases presented in Section VI do not exercise the full breadth of MIDI 2.0 capabilities, such as per-note expressive control or the complete MIDI-CI profile negotiation workflow. This reflects a deliberate scoping decision rather than a limitation of the framework: our goal was to demonstrate the versatility of Nexus Muse across different representative applications, rather than to exhaustively showcase every feature of the underlying protocols. Because the framework exposes standard-compliant Network MIDI 2.0 endpoints, these advanced capabilities remain fully available to practitioners who wish to build on top of it, independently of the specific use cases presented here.

Moreover, the latency and jitter results reported in Section V are consistent with the requirements of the use cases discussed in Section VI. All configurations using the AX210 interface remained within, or close to, the sub-20 ms range, supporting the responsiveness required by examples such as the “Distributed Control Surfaces” and “XR MIDI Controller”. Here, any perceptible delay between a physical or gestural input and its sonic and visual response would break the perception of a single, unified instrument.

At the same time, these results were obtained in a single controlled environment. Performances in denser or more electromagnetically noisy venues (e.g., festival stages, crowded exhibition spaces) may differ and has not yet been character-

¹⁵<https://support.apple.com/guide/audio-midi-setup/share-midi-information-over-a-network-ams1012/mac>

ized. Nevertheless, evaluating such scenarios aligns naturally with the modular design of the Cube. Because the system is built from swappable, off-the-shelf components rather than a fixed appliance, adapting it to more demanding environments (e.g., by upgrading antennas, adjusting channel selection, or adding shielding) does not require redesigning the underlying architecture. The documentation accompanying the open-source release is intended to support exactly this kind of adaptation.

The present evaluation involved at most two clients connected simultaneously to the Cube. Evaluating the system with a larger number of concurrent endpoints (e.g., an ensemble of six or more performers, or a busy installation as in Section VI-E) may reveal different behavior. Performance can also be constrained by other aspects such as WiFi channel contention or the Cube’s processing budget when audio hosting and MIDI routing execute concurrently. We expect that scaling to such scenarios will require extending rather than redesigning the architecture of a system: networking and processing roles could be split across multiple Cubes, or the wireless front-end upgraded, without changing the routing logic or Network MIDI 2.0 compliance. Systematically characterizing this behavior is left for future work.

Finally, the scope of this work is deliberately bounded to co-located scenarios, in contrast to the remote-performance focus that characterizes much Networked Music Performance (NMP) literature. This distinction is fundamental: the low and stable latency reported in Section V is achievable because all endpoints share a local wireless network under the Cube’s control. Extending the framework to remote, internet-scale scenarios would reintroduce many of the challenges addressed by prior NMP research, including inter-performer synchronization under variable and often asymmetric latency, jitter well beyond what local wireless links exhibit, and the need for explicit synchronization mechanisms absent from the current design. Investigating how these challenges can be addressed while preserving a UMP-based transport, rather than relying on ad hoc solutions, we believe, is a natural direction for future work, although it will likely require the development of dedicated timing, recovery, and adaptation mechanisms.

VIII. CONCLUSIONS

This paper presented Nexus Muse, a technical framework for co-located interactive networked music systems built on Universal MIDI Packets and Network MIDI 2.0. Combining the Muse Cube embedded platform with a Unity plugin and a JUCE template, the framework connects heterogeneous musical devices and applications through a shared symbolic transport, reducing fragmentation without introducing another bespoke protocol.

The framework’s modular, standards-based design lets components be adopted independently, with any Network MIDI 2.0/UMP-compliant endpoint joining regardless of toolchain. This grounds interoperability in open specifications rather than proprietary middleware, letting legacy MIDI 1.0 devices, commodity smartphones and headsets, and MIDI 2.0

instruments coexist in one performance environment. Capabilities such as per-note expressive control and MIDI-CI profile negotiation are exposed to artists, ready for future work beyond the present use cases.

The present work evaluates the framework built on top of Network MIDI 2.0 and UMP, rather than the protocol’s internal robustness mechanisms (e.g., forward error correction) themselves; and potentially improving these mechanisms, particularly under adverse or high-interference network conditions, is a task better suited to dedicated protocol-level studies and is left as an important direction for future work. On the framework side, our own priorities include systematically characterizing scalability to larger numbers of concurrent endpoints and to more demanding environments, both of which we expect to be addressable through the platform’s existing modularity rather than architectural redesign. Extending the framework beyond co-located settings toward hybrid or fully remote performance represents a further direction we consider both natural and promising, though it will require dedicated timing, recovery, and adaptation mechanisms.

Taken together, the results suggest that building networked music systems on open, manufacturer-backed transport-layer standards is a practical route toward reducing fragmentation, while leaving room for new instruments, interaction modalities, and performance ecosystems.

ACKNOWLEDGMENT

The authors thank Francesco Manenti for designing and printing the custom case that encloses the whole hardware assembly. The authors also acknowledge Benoit Bouchez, whose original open-source Network MIDI 2.0 software formed the foundation of our work. G. A. Giudici’s contributions are funded by the European Union (EU) under NextGenerationEU (M.D. 118/2023). This work has been supported by the MUSMET project funded by the European Innovation Council (grant n. 101184379). Views and opinions expressed are those of the authors only and do not necessarily reflect those of the EU or the European Research Executive Agency. Neither the EU nor the granting authority can be held responsible for them.

REFERENCES

- [1] Á. Barbosa, “Displaced soundscapes: A survey of network systems for music and sonic art creation,” *Leonardo Music Journal*, vol. 13, pp. 53–59, 2003.
- [2] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, “An overview on networked music performance technologies,” *IEEE Access*, vol. 4, pp. 8823–8843, 2016.
- [3] G. Weinberg, “Interconnected musical networks: Toward a theoretical framework,” *Computer Music Journal*, vol. 29, no. 2, pp. 23–39, 2005.
- [4] D. Keller, V. Lazzarini, and M. Pimenta, *Ubiquitous music*. Springer, 2014.
- [5] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, “Internet of Musical Things: Vision and Challenges,” *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018.
- [6] L. Gabrielli and S. Squartini, *Wireless Networked Music Performance*. Springer, 2015.
- [7] A. Fraietta, O. Bown, S. Ferguson, S. Gillespie, and L. Bray, “Rapid composition for networked devices: Happybrackets,” *Computer Music Journal*, vol. 43, no. 2-3, pp. 89–108, 2019.

- [8] B. Matuszewski, "A web-based framework for distributed music system research and creation," *AES-Journal of the Audio Engineering Society Audio-Acoustics-Application*, 2020.
- [9] M. Romani, G. A. Giudici, D. Stefani, D. Zanoni, A. Boem, and L. Turchet, "BCHJam: a Brain-Computer Music Interface for Live Music Performance in Shared Mixed Reality Environments," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2024, pp. 1–9.
- [10] R. B. Dannenberg, "O2: A network protocol for music systems," *Wireless Communications and Mobile Computing*, vol. 2019, no. 1, p. 8424381, 2019.
- [11] L. Turchet and F. Antoniazzi, "Semantic web of musical things: Achieving interoperability in the internet of musical things," *Journal of Web Semantics*, vol. 75, p. 100758, 2023.
- [12] N. Schuett, "The effects of latency on ensemble performance," *Bachelor Thesis, CCRMA Department of Music, Stanford University*, 2002.
- [13] K. Tsioutas, I. Doumanis, and G. Xylomenos, "A framework for understanding and defining quality of musicians' experience in network music performance environments," in *Audio Engineering Society Convention*, vol. 146, 2019.
- [14] C. Bartlette, D. Headlam, M. Bocko, and G. Velickic, "Effect of network latency on interactive musical performance," *Music Perception*, vol. 24, no. 1, pp. 49–62, 2006.
- [15] R. Hupke, D. Jan, N. Werner, and J. Peissig, "Latency and quality-of-experience analysis of a networked music performance framework for realistic interaction," in *Audio Engineering Society Convention 152*. Audio Engineering Society, 2022, pp. 1–10.
- [16] I. Doumanis, K. Tsioutas, and G. Xylomenos, "An ai-driven multi-feature approach for synchronisation and qoe assessment in network music performance," *Applied Sciences*, 2026.
- [17] P. Verma, A. Mezza, C. Chafe, and C. Rottondi, "A deep learning approach for low-latency packet loss concealment of audio signals in networked music performance applications," in *2020 27th Conference of Open Innovations Association (FRUCT)*. IEEE, 2020, pp. 268–275.
- [18] M. Sacchetto, C. Rottondi, and A. Bianco, "Implementation and optimization of burg's method for real-time packet loss concealment in networked music performance applications," *Personal and Ubiquitous Computing*, pp. 1–17, 2024.
- [19] F. Daniotti, L. Vignati, and L. Turchet, "Tilt loss: a perceptual loss function to improve music packet loss concealment," *Journal on Audio, Speech, and Music Processing*, 2026.
- [20] J. Lazzaro and J. Wawrzyniek, "A case for network musical performance," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, 2001, pp. 157–166.
- [21] N. Falquet and D. Fober, "Rtp midi: Recovery journal evaluation and alternative proposal," Ph.D. dissertation, GRAME, 2005.
- [22] L. Severi, A. Cuccarese, A. Bianco, C. Rottondi *et al.*, "Assessment of recovery journal-based packet loss concealment techniques for low-latency midi streaming," in *AES*, vol. 71, no. 12. Audio Engineering Society, 2023, pp. 873–886.
- [23] MIDI Association, "MIDI 1.0 Detailed Specification," feb 1996.
- [24] G. Loy, "Musicians make a standard: The midi phenomenon," *Computer Music Journal*, vol. 9, no. 4, pp. 8–26, 1985.
- [25] J. Rothstein, *MIDI: A comprehensive introduction*. AR Editions, Inc., 1995, vol. 7.
- [26] M. Wright, A. Freed, and A. Momeni, "Opensound control: State of the art 2003," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2003, pp. 153–159.
- [27] J. Malloch, S. Sinclair, and M. Wanderley, "Libmapper: (a library for connecting things)," in *Extended Abstracts on Human Factors in Computing Systems*. ACM, 2013, pp. 3087–3090.
- [28] R. B. Dannenberg, "Communication for real-time music systems: An overview of o2," *Computer Music Journal*, vol. 45, no. 4, pp. 7–19, 2021.
- [29] D. Johnson, D. Damian, and G. Tzanetakis, "OSC-XR: A toolkit for extended reality immersive music interfaces," in *Proceedings of Sound and Music Computing Conference*, 2019, pp. 202–209.
- [30] G. A. Giudici, D. Stefani, A. Boem, and L. Turchet, "Connecting midi interfaces to the musical metaverse," in *2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2025, pp. 1–9.
- [31] MIDI Association and Association of Musical Electronics Industry (AMEI), "Universal MIDI Packet (UMP) and MIDI 2.0 Protocol Specification," jun 2023.
- [32] C. Drili, C. Allocchio, and N. Buso, "Networked performances and natural interaction via lola: Low latency high quality a/v streaming system," in *International conference on information technologies for performing arts, media access, and entertainment*. Springer, 2013, pp. 240–250.
- [33] C. Werner and R. Kraneis, "UNISON: A Novel System for Ultra-Low Latency Audio Streaming Over the Internet," in *2021 IEEE 18th Annual Consumer Communications Networking Conference*, 2021, pp. 1–4.
- [34] J. Cáceres and C. Chafe, "Jacktrip: Under the hood of an engine for network audio," *Journal of New Music Research*, vol. 39, no. 3, pp. 183–187, 2010.
- [35] S. Ariyani, A. I. Wuryandari, and Y. Priyana, "Design and implementation of beatme server for networked musical performance," in *2012 International Conference on System Engineering and Technology (ICSET)*. IEEE, 2012, pp. 1–5.
- [36] H. von Coler, N. Tonnätt, V. Kather, and C. Chafe, "Sprawl: A network system for enhanced interaction in musical ensembles," in *Proceedings of the Linux Audio Conference*, 2020.
- [37] O. Kenny, L. C. Gomez, S. Sathe, and H. Von Coler, "Bikes: A moving network instrument for music and sound art," in *2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2025, pp. 1–7.
- [38] A. Carôt, C. Hoene, H. Busse, and C. Kuhr, "Results of the fast-music project—five contributions to the domain of distributed music," *IEEE Access*, vol. 8, pp. 47 925–47 951, 2020.
- [39] A. Carôt and C. Werner, "Fundamentals and principles of musical telepresence," *Journal of Science and Technology of the Arts*, vol. 1, no. 1, pp. 26–37, 2009.
- [40] C. Chafe, J.-P. Cáceres, and M. Gurevich, "Effect of temporal separation on synchronization in rhythmic performance," *Perception*, vol. 39, no. 7, pp. 982–992, 2010.
- [41] N. Silva, A. Boem, and L. Turchet, "Interactive iomust-based concerts: Real-time pattern recognition and audience experience," in *2025 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*. IEEE, 2025, pp. 1–11.
- [42] J. Lazzaro and J. Wawrzyniek, "Rfc 6295: Rtp payload format for midi," Internet Engineering Task Force, Tech. Rep., 2011.
- [43] MIDI Association and Association of Musical Electronics Industry (AMEI), "Network MIDI 2.0: User Datagram Protocol for Universal MIDI Packets," nov 2024.
- [44] S. Cheshire and M. Kroemer, "DNS-Based Service Discovery," IETF, Tech. Rep. RFC 6763, Feb. 2013.
- [45] —, "Multicast DNS," IETF, Tech. Rep. RFC 6762, Feb. 2013.
- [46] M. J. Yee-King, *Build AI-Enhanced Audio Plugins with C++*. Focal Press, 2024.
- [47] Y. Kawaguchi, "A morphological study of the form of nature," *SIG-GRAPH Comput. Graph.*, vol. 16, no. 3, p. 223–232, Jul. 1982.
- [48] —, "The Art of Gemotion in Space," in *Tenth International Conference on Information Visualisation (IV'06)*, 2006, pp. 658–663.