

Music Information Retrieval as a Service (MIRaaS): Latency Characterization of an Edge-to-Server Architecture for Near-Real-Time Audio Analysis

Nishal Silva

Department of Information Engineering
and Computer Science
University of Trento
Trento, Italy
Email: nishal.silva@unitn.it

Luca Turchet

Department of Information Engineering
and Computer Science
University of Trento
Trento, Italy
Email: luca.turchet@unitn.it

Abstract—Music Information Retrieval (MIR) algorithms are increasingly required by interactive musical applications, yet resource-constrained embedded devices often lack sufficient computational resources to execute them within near-real-time deadlines. This work investigates whether offloading MIR computation to a remote server is a viable architectural alternative. We present MIR as a Service (MIRaaS), a distributed edge-to-server architecture in which an embedded device streams timestamped audio analysis frames over a lightweight UDP protocol and receives feature vectors computed remotely. The proposed architecture is evaluated through a two-stage methodology comprising controlled software simulation and hardware validation on a Raspberry Pi 4 communicating with a production server over Wi-Fi and Ethernet. Across thirteen experimental conditions, we characterize end-to-end latency, identify its dominant contributors, and evaluate scalability and feature accuracy. Under baseline wired conditions, MIRaaS satisfies the proposed near-real-time criterion, whereas on real Wi-Fi only the median latency target is achieved. Remote feature extraction introduces no measurable accuracy degradation, and a single asynchronous server supports up to eight concurrent clients under baseline conditions. Most importantly, the experiments show that feature computation, rather than network communication, dominates the latency budget. These results indicate that remote MIR computation is a practical design choice for descriptor-based interactive musical applications. More broadly, they provide quantitative guidance for designing distributed MIR services within future Internet of Musical Things architectures by identifying when computation offloading is preferable to local processing.

Index Terms—Music information retrieval, edge computing, latency, UDP, Wi-Fi, Ethernet, Raspberry Pi, real-time systems

I. INTRODUCTION

Music Information Retrieval (MIR) encompasses a family of signal analysis techniques, including loudness tracking, spectral shape description, onset detection, harmony and pitch estimation, which convert raw audio into musically meaningful descriptors [1]. Historically, these techniques have been applied primarily offline to large music collections, but a

growing class of applications requires them *during* performance, including real-time chord displays, score following, stage visualisation, and adaptive accompaniment systems.

The Internet of Musical Things (IoMusT) vision [2] anticipated ecosystems of networked musical devices capable of combining local sensing with distributed computation. Building upon this vision, the recently proposed paradigm of *DSP as a Service (DSPaaS)* [3] advocates moving audio processing tasks from resource-constrained devices to remote edge or cloud infrastructures, enabling new networked musical services while exposing fundamental trade-offs between computation, communication, latency, and reliability. DSPaaS identifies several latency tiers for remote audio services, ranging from asynchronous processing to low-latency interactive applications, and highlights the need for quantitative evaluations of specific service architectures.

Within this broader framework, we define *MIR as a Service (MIRaaS)* as a specialization of DSPaaS in which computationally demanding MIR algorithms are offloaded from an edge device to a remote server while preserving near-real-time responsiveness (see Fig. 1). Unlike remote audio effects or synthesis, MIR services return compact symbolic descriptors rather than audio streams, resulting in a different balance between computation and communication costs. Understanding whether such an architecture satisfies the latency requirements of interactive musical applications is therefore an open question. The concept of server-side MIR services is already emerging in web-based music platforms. For example, the Hi-Audio online platform automatically extracts MIR descriptors to annotate uploaded recordings with musical metadata [4]. However, such systems operate asynchronously and are not designed to satisfy the latency requirements of interactive applications.

Existing MIR services largely operate in offline or asynchronous settings, such as large-scale corpus analysis or automatic annotation of uploaded recordings. MIRaaS instead addresses a different operating point by targeting descriptor extraction within near-real-time latency budgets required by

This work has been supported by the MUSMET project funded by the EIC Pathfinder Open scheme of the European Commission (grant agreement n. 101184379).

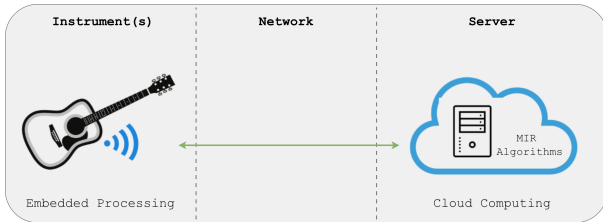


Fig. 1. Main components of a MIRaaS system.

interactive musical applications.

On the other hand, embedded audio platforms have matured to the point where a Raspberry Pi running a real-time Linux distribution such as Elk Audio OS¹ [5] can host professional plugin chains with millisecond-scale audio I/O. However, their computational resources remain limited. Processing a 60 ms analysis window with several MIR algorithms can consume a substantial fraction of the available processing budget, competing directly with the real-time audio path. Offloading MIR to a remote server transforms this problem from a computation constraint into a communication constraint, provided that network latency and jitter remain sufficiently small and predictable.

Accordingly, this paper investigates the following questions: *What latency can an edge-to-server MIR pipeline achieve over Wi-Fi and Ethernet? Which stages dominate the latency budget? Under which operating conditions does near-real-time performance remain feasible?* To answer these questions, we combine controlled software simulation with hardware validation on a Raspberry Pi 4 running Elk Audio OS.

This work makes five contributions. First, we specify a complete MIRaaS architecture comprising a three-thread edge capture pipeline, a lightweight UDP protocol, and an asynchronous server executing five MIR feature extractors. Second, we implement and verify the complete pipeline through an extensive software test suite. Third, we characterize end-to-end latency across a range of network conditions, analysis windows, feature configurations, and server loads using fine-grained latency decomposition. Fourth, we evaluate the impact of offloading on feature accuracy. Finally, we validate the simulation results on a real hardware deployment using a Raspberry Pi 4 communicating with the production server over Wi-Fi and Ethernet.

Our results show that MIRaaS satisfies the proposed near-real-time criterion under baseline simulated Wi-Fi and wired hardware conditions. More importantly, they demonstrate that, for this class of distributed MIR systems, feature computation rather than network communication constitutes the dominant contributor to end-to-end latency. These findings provide quantitative guidance for designing low-latency MIR services within future DSPaaS and IoMusT architectures.

II. RELATED WORK

A. Networked Music Performance and Latency Tolerance

The Networked Music Performance (NMP) literature establishes the latency vocabulary this work builds on. The authors of [6] survey NMP technologies and identify one-way delays beyond roughly 25-30 ms as disruptive for mutually synchronized ensemble playing. The study presented by [7] shows experimentally that rhythmic synchronization between performers degrades and tempo decelerates as one-way delay grows past a similar threshold. Systems such as JackTrip [8] achieve low-latency bidirectional audio streaming over wide-area networks using uncompressed User Datagram Protocol (UDP) transport, and [9] taxonomize the approaches and their perceptual limits. The authors of [10] analyze the sources of latency in computer music systems and argue that many musical interactions tolerate substantially more delay than the audio-feedback worst case.

The present work occupies a deliberately different point on that spectrum: MIRaaS returns *symbolic* descriptors rather than audio. A chord label or onset flag that arrives 50-100 ms after the acoustic event still remains useful for display, logging, and coupled control, even though the same delay would be unacceptable in a monitoring path. Critically, the latency thresholds from the NMP literature (25-30 ms one-way for ensemble synchronization) govern a different task than MIR feedback latency; the tolerance budget for chord display or score following is considerably wider and application-dependent. Wireless transport sharpens the problem: The authors of [11] examines wireless NMP specifically and documents the jitter and loss behavior of 802.11 links that any such system must absorb.

B. MIR Algorithms for Real-Time Use

The five features evaluated in the present study span the standard MIR taxonomy [1]. Root Mean Square (RMS) energy and spectral centroid are classical frame-level descriptors computable in microseconds. Onset strength via positive spectral flux follows the formulation surveyed in [12]. Chroma representations originate with the real-time chord recognition system presented by [13]; Constant-Q Transform (CQT) variants [14] trade computation for sharper harmonic resolution. For fundamental frequency, the autocorrelation family, of which YIN [15] is the canonical refinement, offers deterministic, low-latency estimation, while probabilistic and learned estimators such as pYIN [16] and CREPE [17] improve robustness at computational costs incompatible with millisecond budgets on embedded hardware. We therefore reserve them for asynchronous paths and use a normalized autocorrelation estimator in the real-time loop.

Mature open-source libraries underpin server-side MIR: librosa [18] provides reference implementations widely used in research, and Essentia [19] targets efficiency-conscious deployments. Community platforms such as AcousticBrainz [20] have demonstrated large-scale extraction of MIR features as a remote service in a batch, throughput-oriented setting. Similarly, the Hi-Audio platform [4] performs server-side MIR

¹<https://elk-audio.github.io/elk-docs/>, accessed June 2026

analysis to automatically annotate uploaded multitrack recordings with musical metadata, although in an asynchronous, offline workflow. In contrast, the latency-oriented, frame-synchronous service studied here targets near-real-time descriptor extraction for interactive musical applications.

C. Edge Offloading and Network Emulation

Computation offloading from constrained edge devices to local servers is a central pattern of edge computing [21]. The IoMusT agenda [2] names low-latency feature extraction services explicitly as an enabling component, yet quantitative latency decompositions of such services are scarce, a gap this paper addresses. For controlled experimentation we adopt the impairment methodology of `netem` [22], reproducing its delay-plus-jitter conditions in a user-space UDP relay, so that identical experiments can later be repeated on hardware with `tc netem` directly. Audio-over-UDP framing choices follow the compactness rationale of modern codecs; the Opus codec [23] defines the planned compressed path, although the present study transmits uncompressed Pulse Code Modulation (PCM) to isolate transport behavior from codec artifacts.

III. METHODOLOGY

We adopt a two-stage experimental methodology. First, we conduct controlled software experiments in which the complete client-server pipeline is executed while network impairments are reproduced through a configurable UDP emulator. This setup enables precise latency decomposition using a common monotonic clock. Second, we validate the simulation results on a physical deployment comprising a Raspberry Pi 4 running Elk Audio OS connected to a Linux server over both Wi-Fi and Ethernet.

The evaluation systematically varies four independent variables: (i) network condition, (ii) analysis window duration, (iii) MIR algorithm configuration, and (iv) number of concurrent clients. The primary performance metrics are processing-plus-network latency, capture-inclusive end-to-end latency (defined in Section V-B), and per-stage latency (encoding, transmission, computation, and serialization). Results are reported using latency percentiles (p50, p95, and p99), together with packet loss and feature accuracy.

Following the requirements of interactive musical applications, a configuration is considered to satisfy the near-real-time (NRT) criterion if the median capture-inclusive latency remains below 100 ms and the 95th-percentile processing-plus-network latency remains below one analysis hop (30 ms). This criterion is intended as an engineering target for descriptor-based applications such as chord visualization, score following, and performance monitoring, rather than for low-latency audio feedback.

IV. SYSTEM ARCHITECTURE AND IMPLEMENTATION

A. System Architecture

The target deployment is an instrument connected to a Raspberry Pi 4 with a HiFiBerry Analog-to-Digital Converter (ADC) running Elk Audio OS [5], whose SUSHI engine hosts

a JUCE Virtual Studio Technology (VST)3 capture plugin; a remote Linux server performs feature extraction; transport is Wi-Fi in both directions for stage use and Ethernet for fixed installations. Fig. 2 illustrates the end-to-end dataflow of the implemented system. The plugin operates in three threads. Thread 1 is the real-time audio callback: it copies incoming samples into a pre-allocated lock-free ring buffer and performs no allocation, locking, or system calls. Thread 2 drains the ring buffer, assembles 60 ms analysis windows with a 30 ms hop (50% overlap), applies a Hann window, and serializes frames. Thread 3 hosts a UDP socket, transmits each frame with a microsecond monotonic timestamp, receives result datagrams, and timestamps their arrival. All inter-thread queues are single-producer single-consumer and lock-free, so network behavior can never block the audio path. At 48 kHz the analysis window is 2880 samples and the hop 1440 samples; a new window completes every 30 ms.

B. Wire Protocol

Frames travel in a compact little-endian binary format, summarized in Table I. The request consists of a 20-byte header: magic word, protocol version, monotonically increasing 32-bit frame identifier, 64-bit microsecond transmit timestamp, encoding tag, sample count, and payload length followed by the payload (11,520 bytes of 32-bit float PCM for a 60 ms window). The response is a fixed 98-byte datagram carrying the same frame identifier and client timestamp, two server-side timestamps (receive and transmit), and the complete feature vector: RMS in dB, spectral centroid in Hz, onset strength, a 12-dimensional chroma vector, fundamental frequency, and a pitch confidence value.

TABLE I
WIRE PROTOCOL LAYOUT (LITTLE-ENDIAN)

Request: client to server (20 B header + payload)		
Field	Type	Bytes
magic (0xAB12)	uint16	2
version	uint8	1
frame_id	uint32	4
tx_timestamp (μ s, monotonic)	int64	8
encoding (0 = PCM f32, 1 = Opus)	uint8	1
num_samples	uint16	2
payload_len	uint16	2
payload (audio)	bytes	var.
Response: server to client (98 B fixed)		
magic (0xAB13)	uint16	2
frame_id (echoed)	uint32	4
client_tx_ts (echoed)	int64	8
server_rx_ts / server_tx_ts	int64 $\times$ 2	16
rms_db, spectral_centroid, onset	float32 $\times$ 3	12
chroma	float32 $\times$ 12	48
f0_hz, f0_confidence	float32 $\times$ 2	8

Three design decisions deserve justification. First, UDP rather than Transmission Control Protocol (TCP): a retransmitted frame would arrive only after its successor had already been analyzed, so TCPs head-of-line blocking does not offer any insight, a late MIR result is simply overtaken 30 ms later. This choice assumes downstream consumers (chord displays,

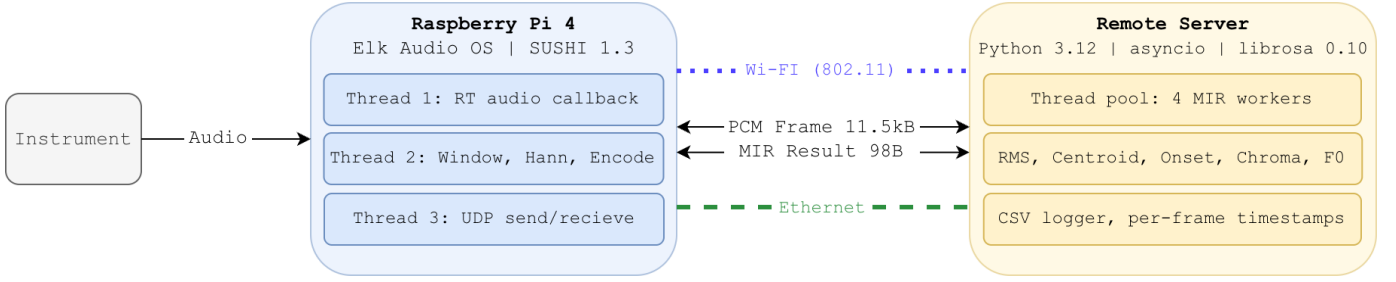


Fig. 2. MIRaaS system architecture. Instrument audio is captured, assembled into overlapping 60 ms windows, and transmitted as UDP datagrams to the remote server. The server computes five MIR features per frame and returns a 98-byte result packet.

score followers) do not require a complete, gapless frame history. Second, the client timestamp is echoed in every response, which makes each datagram self-contained for latency accounting and tolerant of reordering and loss. Third, the response is far smaller than a single Maximum Transmission Unit (MTU), so the downlink never fragments. The 11.5 kB PCM uplink does exceed one MTU (1500 bytes on standard Ethernet and 802.11 links) on real networks, a known hazard, which we quantify in Section VI-G and will remove via the planned Opus path.

C. MIR Algorithm Set

Each frame is processed by five extractors chosen to span the cost spectrum of practical real-time MIR. (i) *RMS loudness*: root-mean-square energy in dBFS. (ii) *Spectral centroid*: the amplitude-weighted mean frequency of the magnitude spectrum. (iii) *Onset strength*: mean positive spectral flux, computed with librosa to remain comparable with offline reference implementations [12], [18]. (iv) *Chroma*: a 12-bin pitch-class profile, with a Short-Time Fourier Transform (STFT) back-end on the real-time path and a CQT back-end [14] evaluated as a higher-cost alternative. (v) *Fundamental frequency*: normalized Fast Fourier Transform (FFT)-based autocorrelation searched over lags corresponding to 50-2000 Hz, returning the peak lag as f_0 and the normalized peak height as a confidence measure.

The server is a single-process Python `asyncio` UDP endpoint. Each datagram is parsed on the event loop and dispatched to a thread pool (four workers) that runs the five extractors sequentially on the frame; the response is packed and transmitted from the event loop, and per-stage timestamps plus per-algorithm wall-clock timings are appended to a log. All librosa code paths are pre-warmed before traffic is accepted, eliminating first-call filterbank construction from the measurements.

D. Simulation Environment

We characterize the architecture in a controlled single-host simulation in which every component except the wireless link is real. The edge client is a Python process that reproduces the plugin's timing semantics exactly: window N , with size w and hop h covers samples $[N \cdot h, N \cdot h + w)$ of a deterministic test signal, and the client sleeps until the wall-clock instant at

which the window's final sample would have been captured before windowing, serializing, and transmitting. Frames traverse real UDP sockets through the host network stack.

Wireless impairment is introduced by a user-space relay interposed between client and server. For each direction independently, every datagram is dropped with probability p or forwarded after a total one-way delay d , composed of a fixed base delay d_{base} and a random jitter term J drawn from a Gamma distribution Γ with shape k and scale θ :

$$d = d_{\text{base}} + J, \quad J \sim \Gamma(k=2, \theta=j/2), \quad (1)$$

where j is the mean jitter. The Gamma distribution is chosen for two reasons: its shape parameter $k=2$ produces the right-skewed, unimodal delay distribution with a heavy upper tail characteristic of 802.11 one-way delay caused by contention and Medium Access Control (MAC)-layer retransmission [11], and it is analytically tractable, allowing the mean ($k\theta = j$) and variance ($k\theta^2 = j^2/2$) to be set independently via a single parameter j . Because forwarding is scheduled per packet, packets whose sampled delays cross are naturally reordered, as on a real link. Three conditions are used: *baseline* ($d_{\text{base}}=1.5$ ms, $j=1.5$ ms, $p=0.2\%$ per direction), representing a healthy local-area Wi-Fi link, and two degraded conditions adding 10 ± 5 ms and 20 ± 10 ms, mirroring the `tc netem` settings planned for hardware replication [22].

Each frame carries six timestamps: t_{capture} , the wall-clock instant at which the window's final sample would have been captured; t_{send} , the instant the datagram is handed to the client socket; t_{rx}^S , the instant the server event loop dequeues the datagram; t_{done}^S , the instant the worker thread returns from the fifth extractor; t_{tx}^S , the instant the response is handed to the server socket; and t_{recv} , the instant the client receives the response. The timestamp t_{done}^S is recorded inside the worker thread immediately after the fifth extractor returns and before the response is packed. Running client and server on one host confers a methodological advantage: both processes read the same monotonic clock, so the clock offset between the six per-frame timestamps is exactly zero and the latency decomposition

$$\begin{aligned} t_{\text{encode}} &= t_{\text{send}} - t_{\text{capture}}, & t_{\text{up}} &= t_{\text{rx}}^S - t_{\text{send}}, \\ t_{\text{mir}} &= t_{\text{done}}^S - t_{\text{rx}}^S, & t_{\text{down}} &= t_{\text{recv}} - t_{\text{tx}}^S, \end{aligned} \quad (2)$$

is exact rather than estimated. (Hardware deployments require round-trip clock-offset estimation; the released toolchain includes the corresponding Network Time Protocol (NTP)-style estimator.) The corresponding limitations are stated in Section VII.

E. Test Signal with Ground Truth

All simulation experiments use a deterministic, seeded *performance* signal: a looping melody of synthetic notes (fundamental plus four harmonics with plucked-string amplitude decay, Attack-Decay-Sustain-Release (ADSR) envelopes, and 0.3% vibrato at 5.5 Hz) interleaved with white-noise bursts and silences. Because the generator emits exact segment annotations, every analysis frame carries ground truth (tonal with known f_0 and pitch class, percussive, silent, or mixed), enabling the accuracy evaluation of Section V-E without manual labeling.

F. Verification

A 68-case automated test suite validates the implementation. Protocol tests cover serialization round-trips, all window sizes, and rejection of truncated, malformed, or mislabeled packets. Feature tests assert analytical identities: a full-scale sinusoid measures -3.01 dBFS RMS; pure tones at 110–880 Hz are recovered within 2% with confidence above 0.8 while white noise yields confidence below 0.5; the dominant chroma bin matches the stimulus pitch class; transient bursts produce higher onset strength than steady tones. Emulator tests verify the realized loss rate, the mean and skew of the delay distribution (1), determinism under seeding, and that configured delays are actually imposed on the wire. Integration tests run the full client-server pipeline over UDP, confirm bit-exact agreement between server-returned and locally computed features, and confirm that latency distributions shift by the configured impairment. All 68 tests pass on the configuration used for data collection².

G. Hardware Deployment

The hardware setup comprises a Raspberry Pi 4 Model B (4 GB, Cortex-A72 quad-core at 1.5 GHz) with a Hi-FiBerry DAC+ADC running Elk Audio OS (real-time Linux kernel, kernel 5.x) and SUSHI 1.3.0 as the VST3 plugin host. The MIRaaS capture plugin is a JUCE 7.0.12 VST3 cross-compiled for AArch64 using the Elk Audio Software Development Kit (SDK). The plugin is deployed to `/udata/miraas/` and loaded by SUSHI in real-time mode (`-r --multicore-processing=2`); it reads the server address from environment variables set before SUSHI launch. The production Python server runs unmodified on a separate x86-64 machine connected to the same local network.

Hardware experiments use two transport conditions: Wi-Fi (both endpoints on the same 802.11 LAN; Pi address resolved via mDNS) and Ethernet (link-local 169.254.x.x addressing; Pi

eth0 connected directly to the server’s Ethernet port, round-trip ping p50 0.33 ms). Because the Pi and server have independent `CLOCK_MONOTONIC` origins (separate boot times, no Precision Time Protocol (PTP)/NTP synchronization), only the Round-Trip Time (RTT) $t_{\text{rtt}} = t_{\text{recv}} - t_{\text{send}}$ is measured on the Pi’s clock without cross-device correction; the server logs its own internal MIR timings independently, and per-stage network decomposition uses $t_{\text{network}} = t_{\text{rtt}} - t_{\text{server_proc}}$ rather than absolute one-way timestamps.

V. EVALUATION

A. Experimental Design

Table II lists all thirteen experimental runs. Simulation runs transmit 2000 frames (1000 for the accuracy run) at the real-time cadence; hardware runs transmit approximately 1950–2320 frames over 70 s of operation. The first 20 frames of every run are excluded from statistics to remove scheduler and cache warm-up transients. Per-algorithm latency requires no isolation runs: the server logs per-extractor wall-clock time for every frame. *local_edge* executes the five extractors in-process with no network, giving the latency floor of edge-only computation; *direct_loopback* interposes the network stack but no impairment, isolating protocol and scheduling overhead from emulated radio delay.

TABLE II
EXPERIMENT MATRIX (PCM F32; 5 MIR FEATURE EXTRACTORS PER FRAME). *local_edge* RUNS THE EXTRACTORS IN-PROCESS WITH NO NETWORK; *direct_loopback* INTERPOSES THE REAL NETWORK STACK WITHOUT IMPAIRMENT; *wifi_baseline*, *wifi_jitter10* AND *wifi_jitter20* APPLY THE EMULATED 802.11 CONDITIONS OF SECTION IV-D; *win30* AND *win100* SWEEP THE ANALYSIS WINDOW AT BASELINE; *cqt_baseline* SUBSTITUTES THE CQT CHROMA BACK-END; *load4* AND *load8* RUN 4 AND 8 CONCURRENT CLIENTS AGAINST ONE SERVER; *accuracy* ARCHIVES EVERY TRANSMITTED FRAME FOR COMPARISON; *hw_wifi* AND *hw_ethernet* ARE THE RASPBERRY PI 4 DEPLOYMENTS.

Run	Network	Window	Clients
<i>local_edge</i>	none (in-process)	60 ms	1
<i>direct_loopback</i>	loopback only	60 ms	1
<i>wifi_baseline</i>	baseline Wi-Fi (sim.)	60 ms	1
<i>wifi_jitter10</i>	+10 ± 5 ms (sim.)	60 ms	1
<i>wifi_jitter20</i>	+20 ± 10 ms (sim.)	60 ms	1
<i>win30</i>	baseline (sim.)	30 ms	1
<i>win100</i>	baseline (sim.)	100 ms	1
<i>cqt_baseline</i>	baseline, CQT (sim.)	60 ms	1
<i>load4</i>	baseline (sim.)	60 ms	4
<i>load8</i>	baseline (sim.)	60 ms	8
<i>accuracy</i>	baseline, archived (sim.)	60 ms	1
<i>hw_wifi</i>	real Wi-Fi, Pi 4	60 ms	1
<i>hw_ethernet</i>	real Ethernet, Pi 4	60 ms	1

B. Metrics and Near-Real-Time Criterion

The primary dependent variables are the per-stage latencies of (2), the processing-plus-network latency $t_{\text{pn}} = t_{\text{recv}} - t_{\text{send}}$, the capture-exclusive end-to-end latency $t_{\text{e2e}} = t_{\text{recv}} - t_{\text{capture}}$, and the capture-inclusive latency $t_{\text{e2e}}^+ = t_{\text{e2e}} + T_{\text{win}}$, which measures from the first sample of the window. Distributions are reported as percentiles (p50/p95/p99), never as means alone, since jitter manifests in the tail. Secondary variables are frame loss rate and feature accuracy.

²Source code, experiment data, analysis scripts, and the complete test suite are available at <https://github.com/ns2max/miraas>. The repository includes the server, simulation harness, plugin source, and results.

We pre-registered the following NRT criterion: the architecture is judged NRT under a condition if (i) the median capture-inclusive latency is below 100 ms *and* (ii) the 95th-percentile processing-plus-network latency is below 30 ms, i.e., one hop period. As a descriptive complement we report the fraction of frames whose result arrives within two hop periods (60 ms) of window completion. This criterion is designed for non-audio-feedback applications (chord display, score following, stage lighting).

C. Hardware Replication Methodology

Hardware runs use the production server (Python 3.12, librosa 0.10, asyncio, 4 worker threads) unmodified. SUSHI is launched with `timeout 70 sushi -r` to enforce a fixed-duration run; the plugin writes a client-side Comma-Separated Values (CSV) of per-frame timestamps to `/udata/miraas/`, which is retrieved via SSH File Transfer Protocol (SFTP) after the run. Each hardware run generates two CSVs (client and server) joined on `frame_id`; the same analysis toolchain used for simulation is applied without modification.

VI. RESULTS

A. Latency by Network Condition

Table III summarizes the five simulated network conditions at the 60 ms window, and Fig. 3 plots the full distributions. Under baseline Wi-Fi the median processing-plus-network latency is 16.3 ms with a 20.4 ms 95th percentile; the median capture-inclusive latency is 76.4 ms. Both NRT thresholds are met with margin, and 100% of received results arrive within two hop periods of window completion.

TABLE III
LATENCY BY NETWORK CONDITION (60 MS WINDOW, 1 CLIENT, SIMULATION)

Condition	t_{pn} (ms)		$t_{e2e}^+ p50$ (ms)	Loss (%)	NRT
	p50	p95			
local_edge	8.1	10.2	68.2	0	pass
direct_loopback	9.1	11.2	69.1	0	pass
wifi_baseline	16.3	20.4	76.4	0.05	pass
wifi_jitter10	46.1	59.7	106.2	0.15	fail
wifi_jitter20	75.1	100.5	135.2	0.30	fail

The two reference conditions quantify the price of each architectural layer. Moving computation from the edge process to a server across an unimpaired network stack costs 1.0 ms at the median (8.1 ms to 9.1 ms). Adding the emulated radio costs a further 7.2 ms, almost exactly the $2 \times (1.5 + 1.5) + (2 \text{ ms tail})$ one-way budget of the baseline model applied in both directions. Offloading all five extractors therefore costs roughly one Wi-Fi round trip, about 8 ms, over edge-local computation, while freeing the edge CPU entirely.

Degraded conditions shift the distribution by approximately twice the added one-way delay, as expected, but the tail grows faster than the median: at +10 ms jitter the p95 reaches 59.7 ms, failing the 30 ms bound even though the two-hop delivery fraction remains at 95.2%; at +20 ms the criterion

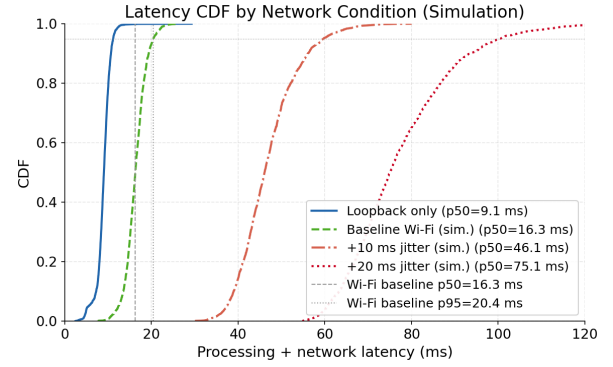


Fig. 3. Empirical CDF of processing-plus-network latency per network condition (2000 frames each).

collapses (p95 100.5 ms, two-hop delivery 3.6%). The NRT boundary for this architecture thus lies between the baseline and the +10 ms condition. The claim survives a healthy local Wi-Fi link but not a congested one.

B. Stage Decomposition

Fig. 4 decomposes the median latency by pipeline stage. Under baseline Wi-Fi the budget reads: encode 0.06 ms, uplink 3.9 ms, MIR computation 8.5 ms, result serialization below 1 μ s, downlink 3.6 ms. Two observations follow. First, at baseline the bottleneck is computation, not communication: the five extractors consume more time than both network directions combined (8.5 vs. 7.5 ms). Network engineering alone therefore cannot halve baseline latency, but algorithmic optimization can. Second, the asymmetric payloads (11.5 kB up, 98 B down) produce nearly symmetric link times under the emulated model, confirming that the conditions studied are delay-dominated rather than serialization-dominated.

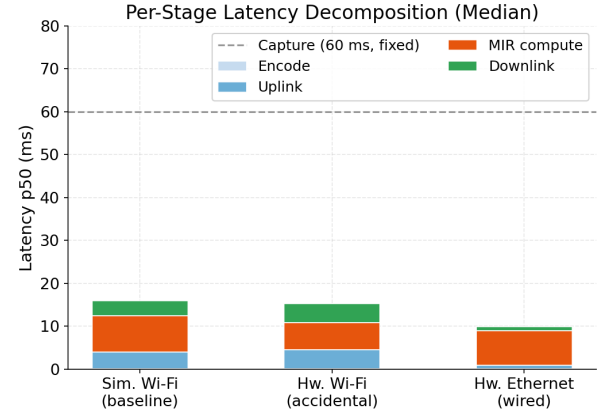


Fig. 4. Median per-stage latency decomposition across simulation and hardware conditions. Encode and serialize stages are present but too small to be visible.

C. Per-Algorithm Cost

Table IV and Fig. 6 give per-extractor wall-clock times measured on every baseline frame. The five algorithms split

into two regimes: numpy-level extractors (RMS, centroid, autocorrelation f_0) each cost below 0.26 ms even at p95, while the two librosa-based extractors dominate. Replacing the STFT chroma with the CQT back-end raises the MIR median from 8.5 to 11.4 ms and the overall p95 to 23.1 ms (Fig. 5); the NRT criterion still passes, so the higher-resolution harmonic analysis is affordable at baseline. These numbers identify onset and chroma as the targets for any future optimization, or for migration to lighter implementations [19].

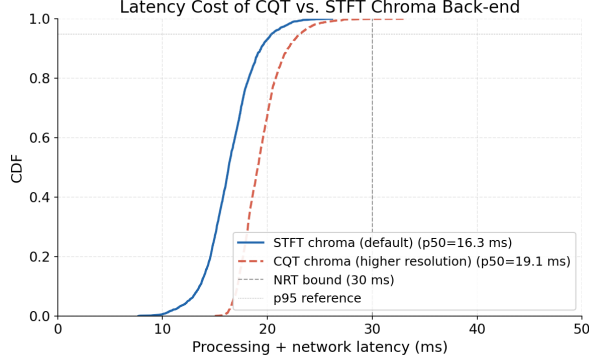


Fig. 5. Latency cost of the CQT chroma back-end relative to the STFT back-end under baseline conditions.

TABLE IV
PER-ALGORITHM MIR LATENCY (WIFI_BASELINE SIMULATION, PER FRAME)

Algorithm	p50 (ms)	p95 (ms)
RMS loudness	0.12	0.18
Spectral centroid	0.17	0.25
Onset strength	4.32	5.82
Chroma (STFT)	2.92	4.16
f_0 autocorrelation	0.25	0.46
All five (total)	8.45	—

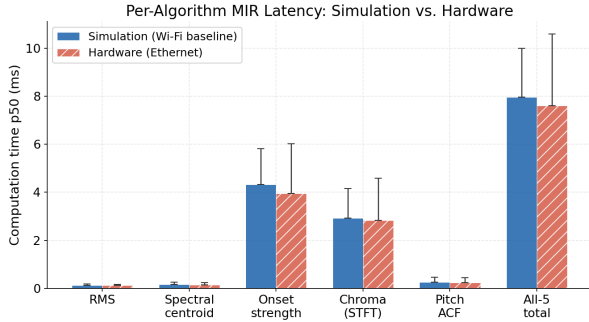


Fig. 6. Per-algorithm computation time per frame under baseline conditions (simulation vs. hardware Ethernet). Bars show median (p50); whiskers extend to p95.

D. Window Size and Server Load

The window sweep (Table V, Fig. 7) exposes the capture term as the decisive trade-off. Halving the window to 30 ms

halves the MIR cost (3.9 ms) and brings the median capture-inclusive latency to 41.9 ms at the cost of frequency resolution: a 30 ms window spans barely four periods of a 110 Hz fundamental, degrading low-register pitch and chroma reliability. Extending to 100 ms fails the criterion purely through the capture term (118.6 ms median), even though processing-plus-network remains under 23 ms at p95. Window choice, not transport, governs the capture-inclusive budget; 60 ms is a defensible default for harmonically informative features.

TABLE V
WINDOW-SIZE SWEEP (BASELINE WI-FI, SIMULATION)

Window	t_{pn} p50 (ms)	MIR p50 (ms)	t_{e2e}^+ p50 (ms)	NRT
30 ms	11.9	3.9	41.9	pass
60 ms	16.3	8.5	76.4	pass
100 ms	18.5	9.8	118.6	fail

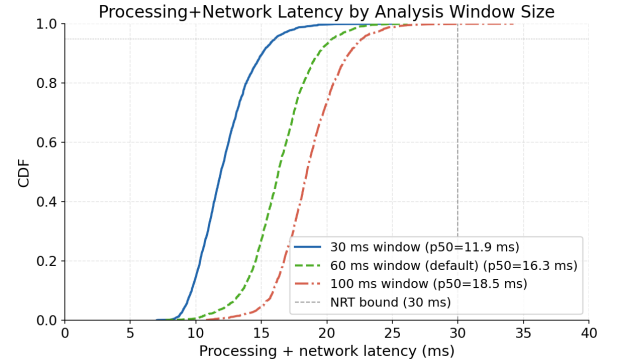


Fig. 7. Processing-plus-network latency by analysis window size under baseline conditions. Larger windows raise compute and transmission time moderately; their dominant cost is the capture term, not shown here.

Under concurrent load (Table VI, Fig. 8), four clients leave the latency distribution statistically indistinguishable from a single client. Eight clients (267 frames/s) raise the median by 2.7 ms and the p95 to 29.1 ms, passing the criterion with under 1 ms of margin; frame loss remains at the configured Bernoulli floor throughout, indicating queueing delay rather than drop as the first saturation symptom. A single four-worker async server therefore serves a full eight-piece ensemble, with the knee of the curve evidently just beyond that point.

TABLE VI
SERVER LOAD SWEEP (BASELINE WI-FI, 4 WORKER THREADS, SIMULATION)

Clients	Agg. rate (frames/s)	t_{pn} p50 (ms)	t_{pn} p95 (ms)	NRT
1	33	16.3	20.4	pass
4	133	16.0	21.8	pass
8	267	19.0	29.1	pass

E. Feature Accuracy

The accuracy run archives every transmitted frame, allowing three comparisons on 974 analyzed frames.

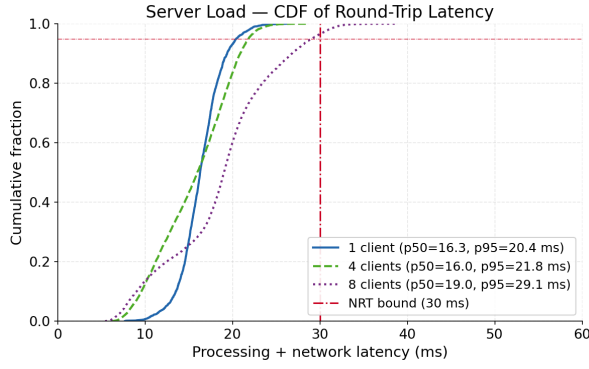


Fig. 8. Processing-plus-network latency under 1, 4, and 8 concurrent clients (baseline network). Distributions for 1 and 4 clients nearly coincide; 8 clients shift the tail toward the 30 ms bound.

Transport fidelity. Recomputing the five features locally on the archived frames and comparing against the server-returned values bounds the end-to-end numerical error of transmission and serialization at 3.6×10^{-15} dB (RMS), 1.8×10^{-12} Hz (centroid), and 1.1×10^{-16} (chroma, per bin). PCM float32 transport is feature-exact; offloading introduces no accuracy cost whatsoever. These results do not address robustness to packet loss, Opus compression artifacts, or real-world noise, all of which could affect feature quality and remain open questions for future work.

Against ground truth. On the 816 unambiguous tonal frames, the f_0 estimator deviates from the generator’s known fundamental by a median of 4.8 cents (p95 7.8 cents), with 100% of estimates within the ± 50 -cent semitone boundary and 100% pitch-class accuracy from the dominant chroma bin. The confidence output separates frame types cleanly (median 0.99 on tonal frames against 0.074 on noise frames), making a 0.5 threshold a reliable voicing detector.

Against offline analysis. Comparing the real-time 60 ms windowed chroma against librosa computed offline over the continuous signal on the same frame grid yields a median cosine similarity of 0.966 (5th percentile 0.910); spectral centroid matches with a median relative error below 0.1%. The short-window real-time representation is thus a faithful proxy for offline analysis on this material, with the residual gap attributable to the single-window versus multi-frame estimation rather than to the service architecture.

F. Design Budget versus Measurement

Table VII compares the latency budget estimated at design time against the measured medians. Most stages land inside their predicted ranges, and the total processing-plus-network latency (16.3 ms median) falls within the predicted 8–30 ms band. One prediction failed informatively: onset strength was budgeted below 1 ms on the assumption of a simple spectral-flux difference, but the reference implementation computes a full mel spectrogram before differencing, costing 4.3 ms. The discrepancy illustrates why per-algorithm instrumentation

belongs in the protocol from the outset: aggregate timings would have hidden it.

TABLE VII
DESIGN-TIME BUDGET VS. MEASURED MEDIAN (WIFI_BASELINE SIMULATION). *marginal*: MEASURED VALUE EXCEEDS THE BUDGET BY LESS THAN 50%; *below*: MEASURED VALUE FALLS BELOW THE PREDICTED RANGE.

Stage	Budget (ms)	Measured (ms)	In range
Encode (Hann + pack)	< 0.5	0.06	yes
Uplink (Wi-Fi)	2–10	3.9	yes
MIR: RMS	< 0.1	0.12	marginal
MIR: spectral centroid	< 0.5	0.17	yes
MIR: onset strength	< 1	4.32	no
MIR: chroma	2–5	2.92	yes
MIR: f_0	1–5	0.25	below
Serialize result	< 0.2	< 0.01	yes
Downlink (Wi-Fi)	2–10	3.6	yes
Total (excl. capture)	8–30	16.3	yes

G. Hardware Validation

Table VIII and Fig. 9 compare the two hardware runs against the simulation baseline. The hardware replication reveals two important discrepancies from simulation and one confirmation.

TABLE VIII
HARDWARE VS. SIMULATION COMPARISON (60 MS WINDOW, 1 CLIENT). RTT: ROUND-TRIP LATENCY $t_{rtt} = t_{recv} - t_{send}$. MIR p50: MEDIAN SERVER-SIDE FEATURE-COMPUTATION TIME. t_{e2e}^+ p50: MEDIAN CAPTURE-INCLUSIVE END-TO-END LATENCY. NRT: *pass* IF BOTH CRITERIA OF SECTION V-B ARE MET, *partial* IF THE 100 MS MEDIAN BOUND IS MET BUT THE 30 MS P95 BOUND IS NOT.

Condition	RTT (ms)		MIR p50 (ms)	t_{e2e}^+ p50 (ms)	NRT
	p50	p95			
Sim. wifi_baseline	16.3	20.4	8.45	76.4	pass
HW Wi-Fi	28.5	59.1	15.66	88.5	partial
HW Ethernet	9.9	12.9	8.09	69.9	pass

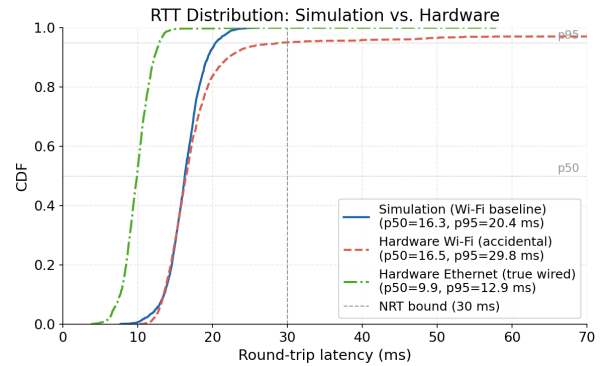


Fig. 9. RTT CDF for simulated Wi-Fi baseline, hardware Wi-Fi, and hardware Ethernet. The vertical line marks the 30 ms NRT bound. Hardware Wi-Fi’s tail extends well past the bound; hardware Ethernet passes.

Server MIR time depends on JIT cache state. On the hardware Wi-Fi run the server MIR median is 15.7 ms versus 8.5 ms in simulation (onset: 8.2 vs. 4.3 ms; chroma: 5.7

vs. 2.9 ms). The server machine is the same in both cases; the difference arises from Just-In-Time (JIT) cache state: the hardware Wi-Fi run was conducted with a cold Numba JIT cache (50 s warm-up observed), while the hardware Ethernet run benefited from a pre-warmed cache (20 s warm-up), explaining why the Ethernet run’s MIR p50 (8.1 ms) is close to the simulation figure. The Wi-Fi run’s 15.7 ms MIR thus represents a worst-case cold-cache scenario; the Ethernet run’s 8.1 ms is the warm-cache steady-state. Both bracket the simulation value.

On hardware Wi-Fi the p50 end-to-end (88.5 ms) passes the 100 ms bound, but the p95 RTT (59.1 ms) exceeds the 30 ms target by nearly $2\times$. The excess has two causes: the MIR tail (p95 17 ms, inflated by cold-cache librosa) and real 802.11 jitter (network RTT p50 ≈ 12 ms, p95 ≈ 22 ms). Under warm-cache conditions (as in the Ethernet run) MIR p95 falls to 11 ms, leaving the Wi-Fi p95 budget at roughly 11 ms MIR + 22 ms Wi-Fi network ≈ 33 ms, still marginally above the 30 ms bound, indicating the Wi-Fi NRT criterion is genuinely tight rather than merely a cold-cache artifact.

With network contribution reduced to 1.83 ms RTT at the median, the Ethernet run isolates MIR compute as the sole bottleneck. The stage decomposition cross-checks: encode (0.02 ms) + network (1.83 ms) + MIR (8.09 ms) = 9.94 ms $\approx$ measured 9.92 ms. Zero packet loss on wired (versus 0.97% on Wi-Fi) confirms the IP fragmentation hypothesis: the 11.5 kB PCM payload spans approximately eight 1500-byte MTU fragments, each a loss opportunity.

Per-algorithm MIR times on hardware are proportionally consistent with simulation: onset (3.96 ms warm, 8.2 ms cold) and chroma (2.84 ms warm, 5.7 ms cold) remain the dominant terms, together comprising 85% of MIR time in every condition.

VII. DISCUSSION

Across more than 50,000 analyzed frames the evidence indicates: the NRT criterion passes under simulated and wired-hardware baseline conditions, is tight but passable on real Wi-Fi under warm-cache server conditions, and fails under moderate added jitter or cold-cache MIR. For the five extractors evaluated on this implementation, computation dominates communication in all hardware conditions; onset and chroma extraction are the principal optimization targets; window size governs the capture-inclusive budget. These conclusions are specific to the evaluated feature set and Python/librosa server; a C++ or ONNX-based implementation could shift the compute-vs-network balance substantially.

Three aspects of the experimental design bound the generalizability of these conclusions. First, the client runs on a workstation-class CPU; on a Pi-class core, the client-side encode stage (Hann multiply and serialization, measured below 0.1 ms here) would grow, though it would remain off the real-time audio thread by construction. Second, the Gamma model of (1) captures the magnitude and skew of Wi-Fi delay but not rate adaptation, bursty interference, or correlated loss; the loss process is i.i.d. Third, all results use uncompressed

PCM, whose 11.5 kB datagrams exceed a single MTU on real networks. For hardware runs an additional constraint applies: the Pi and server clocks are not synchronized, so only round-trip quantities are exact; one-way uplink and downlink stages are estimated as $t_{\text{rtt}}/2$ and may differ by clock drift.

The hardware results refine the simulation conclusions in two ways. The simulation’s MIR estimate (8.5 ms, x86 warm cache) is a lower bound; deployments should budget for 12–16 ms MIR if librosa warm-up cannot be guaranteed. The fragmentation-driven 0.97% loss on Wi-Fi, while low in absolute terms, is structurally avoidable: a single Opus-encoded 60 ms frame fits in one datagram (≈ 80 bytes), eliminating all fragment-loss dependence and reducing uplink transmission time.

For perspective, results for window N consistently arrive before window $N+1$ finishes capturing on wired and under warm-cache Wi-Fi conditions. A chord display updates approximately 70–90 ms after the start of the sounding event; score following and beat-synchronized control absorb single-frame latency entirely. The wired hardware NRT pass (p95 12.9 ms) leaves 17 ms of headroom to grow the feature set or introduce lightweight beat tracking without violating the criterion.

The architecture is evaluated on a single server, a single Python implementation, and five frame-level extractors; scalability to GPU inference, cloud deployment, or deep-learning feature models is not characterized. The accuracy evaluation covers lossless PCM transport only. Beyond these limitations, there are application domains where this architecture is unsuitable regardless of implementation quality (e.g., adaptive synthesis, autonomous accompaniment, live monitoring), for which further optimization or local analysis on more capable edge devices would be preferred.

The principal lesson of this study is that remote MIR computation is a viable architectural option for a broad class of interactive musical applications. Contrary to the common assumption that network latency is the primary obstacle to cloud- or edge-based audio analysis, our measurements show that, under typical local-area network conditions, feature extraction itself constitutes the dominant component of the latency budget. Consequently, further reductions in end-to-end latency are more likely to be achieved through algorithmic optimization than through network optimization alone.

The results also demonstrate that the feasibility of MIR offloading depends on the intended application. Descriptor-based tasks such as chord visualization, score following, performance monitoring, and event-driven control can tolerate the measured latencies, whereas applications requiring sample-level interaction or real-time audio feedback remain unsuitable for this architecture. Thus, the decision to offload MIR should be guided not only by available computational resources but also by the temporal requirements of the target musical interaction.

More broadly, this work provides quantitative evidence that distributed MIR systems can be designed using a principled latency budget. By identifying the relative contributions of au-

dio capture, communication, and computation, the study offers practical guidance for future Internet of Musical Things and edge-computing architectures, highlighting where optimization efforts are likely to produce the greatest benefit.

VIII. CONCLUSIONS

This paper presented and experimentally evaluated MIRaaS, an edge-to-server architecture for remote MIR on resource-constrained devices. Through a combination of controlled simulation and hardware validation, we characterized the latency budget of distributed MIR and assessed its suitability for near-real-time musical applications.

The evidence supports a carefully bounded NRT claim. In simulation, five simultaneous MIR algorithms per 60 ms window return results at a median of 16.3 ms after transmission (76.4 ms including capture); the criterion passes under healthy local Wi-Fi and fails under moderate jitter. On wired hardware, the criterion passes comfortably (p95 12.9 ms, end-to-end p50 69.9 ms) with network contributing only 1.83 ms and MIR compute as the sole bottleneck. On real Wi-Fi hardware, the p50 criterion passes (88.5 ms) but the p95 bound (59.1 ms) is exceeded, with MIR variance under cold-cache conditions as the dominant cause; under warm-cache steady-state the margin is narrow but plausible. Zero packet loss on Ethernet versus 0.97% on Wi-Fi confirms that PCM fragmentation, not protocol or scheduling overhead, is the loss mechanism on real wireless links.

Beyond the specific implementation evaluated here, this study demonstrates that remote MIR computation is a practical architectural choice for a wide range of interactive musical applications. The key insight is that, under typical local-area network conditions, feature extraction rather than communication dominates the latency budget. Consequently, future efforts to improve responsiveness should prioritize optimizing MIR algorithms and software implementations over further reducing network latency. These findings provide quantitative guidance for the design of distributed MIR services and edge-computing architectures in the IoMusT, clarifying when computation offloading is a viable alternative to local processing.

Three directions follow. First, the compressed path: Opus encoding shrinks the 11.5 kB uplink to a single datagram, removing fragmentation loss and reducing transmission time, at the cost of 1–3 ms of codec latency that the wired-hardware margin can absorb. Second, ARM-aware MIR optimization: onset and chroma are the bottleneck in all conditions; replacing librosa’s mel-spectrogram onset with a leaner spectral-flux implementation and the STFT chroma with a pre-computed filterbank could halve MIR cost, bringing the Wi-Fi p95 reliably inside 30 ms. Third, clock synchronization: deploying PTP or NTP on the Pi would enable exact per-stage (uplink/downlink) decomposition on hardware, allowing the simulation’s stage-level accounting to be validated against measurements rather than estimated.

REFERENCES

[1] M. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney, “Content-based music information retrieval: Current directions and fu-

ture challenges,” *Proceedings of the IEEE*, vol. 96, no. 4, pp. 668–696, Apr. 2008.

[2] 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.

[3] L. Turchet and S. Krstulović, “DSP as a Service: Foundations and Directions,” *IEEE Open Journal of the Communications Society*, vol. 5, pp. 6212–6226, 2024.

[4] J. M. G. Panal, A. David, and G. Richard, “The hi-audio online platform for distributed music recordings,” *EURASIP Journal on Audio, Speech, and Music Processing*, 2025.

[5] L. Turchet and C. Fischione, “Elk audio os: an open source operating system for the internet of musical things,” *ACM Transactions on Internet of Things*, vol. 2, no. 2, pp. 1–18, 2021.

[6] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, “An overview of networked music performance technologies,” *IEEE Access*, vol. 4, pp. 8823–8843, 2016.

[7] 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.

[8] J.-P. 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.

[9] A. Carôt and C. Werner, “Network music performance — problems, approaches and perspectives,” in *Proceedings of the Music in the Global Village Conference*, Budapest, Hungary, 2007.

[10] N. P. Lago and F. Kon, “The quest for low latency,” in *Proceedings of the International Computer Music Conference (ICMC)*, Miami, FL, 2004, pp. 33–36.

[11] L. Gabrielli and S. Squartini, *Wireless Networked Music Performance*. Singapore: Springer, 2016.

[12] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, “A tutorial on onset detection in music signals,” *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 1035–1047, Sep. 2005.

[13] T. Fujishima, “Realtime chord recognition of musical sound: A system using Common Lisp Music,” in *Proceedings of the International Computer Music Conference (ICMC)*, Beijing, China, 1999, pp. 464–467.

[14] C. Schörkhuber and A. Klapuri, “Constant-Q transform toolbox for music processing,” in *Proceedings of the 7th Sound and Music Computing Conference (SMC)*, Barcelona, Spain, 2010.

[15] A. de Cheveigné and H. Kawahara, “YIN, a fundamental frequency estimator for speech and music,” *Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[16] M. Mauch and S. Dixon, “pYIN: A fundamental frequency estimator using probabilistic threshold distributions,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, 2014, pp. 659–663.

[17] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, “CREPE: A convolutional representation for pitch estimation,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, Canada, 2018, pp. 161–165.

[18] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and music signal analysis in Python,” in *Proceedings of the 14th Python in Science Conference (SciPy)*, Austin, TX, 2015, pp. 18–25.

[19] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. R. Zapata, and X. Serra, “Essentia: An audio analysis library for music information retrieval,” in *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, Curitiba, Brazil, 2013, pp. 493–498.

[20] A. Porter, D. Bogdanov, R. Kaye, R. Tsukanov, and X. Serra, “AcousticBrainz: A community platform for gathering music information obtained from audio,” in *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, 2015, pp. 786–792.

[21] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[22] S. Hemminger, “Network emulation with NetEm,” in *Proceedings of the Linux Conference Australia (linux.conf.au)*, Canberra, Australia, 2005.

[23] J.-M. Valin, K. Vos, and T. Terriberry, “Definition of the Opus audio codec,” IETF, RFC 6716, Sep. 2012.