

Wasted Bandwidth: The Case for Lossless Audio in Networked Music Performance

Leonardo Severi

Department of Electronics and Telecommunications

Politecnico di Torino

Torino, Italy

leonardo.severi@polito.it

Abstract—Networked Music Performance and emerging Musical Metaverse scenarios impose strict latency requirements that have traditionally been met through two transmission strategies: uncompressed Pulse Code Modulation audio, which preserves full fidelity at a high bandwidth cost, or low-latency lossy codecs such as Opus, which reduce bandwidth at the expense of signal integrity. Lossless compression, more commonly associated with storage and streaming, has so far received little attention in real-time interactive contexts.

We derive QLAC, a low-latency lossless audio codec, from the FLAC reference implementation, adapting it for unbuffered operation and pairing it with a compact payload format suited to network transmission. We evaluate it along three dimensions: encoding time, decoding time, and bandwidth reduction with respect to uncompressed transmission. We report a characterization aimed at assessing whether lossless transmission can offer a favorable trade-off between fidelity and bandwidth under real-time constraints.

Index Terms—Networked Music Performance, Lossless audio compression, Low-latency audio, Real-time audio streaming, Internet of Sounds, Musical metaverse, Audio codec.

I. INTRODUCTION

Networked Music Performance (NMP) denotes the practice of musicians located in geographically distinct places playing together in real time over a transmission network, with the goal of reproducing the co-presence and synchronization characteristic of performing within a physical space. The dominant constraint in this scenario is end-to-end latency: for an ensemble to keep a stable common tempo without a conductor, the one-way transmission delay must remain within a threshold of 20 – 30 ms [1], a figure that roughly corresponds to the maximum acoustic separation tolerable between performers in real space ($\approx 10m$ apart). This requirement dictates the entire chain of technical choices in the field (e.g., the use of UDP, privileging low latency over reliability), all subordinated to preserving an extremely tight time budget.

Increasingly, this line of research is also being explored in connection with the Musical Metaverse (MM), part of the emerging Internet of Sounds (IoS) paradigm born from the convergence of Sound and Music Computing with the Internet

of Things [2]. The MM describes a version of the metaverse dedicated to musical activities, attending concerts, making music together, learning and teaching, within virtual, immersive, shared environments in which users interact through their own avatars. In this context, NMP constitutes the foundational layer on the auditory plane. A growing body of work has addressed the integration of networking with Musical Extended Reality (Musical XR) [3], [4], spanning networking architectures for the MM, immersive and spatial-audio rendering, remote conduction, telematic performance systems, and packet loss concealment, yet across these efforts latency and its variability remain the acknowledged fundamental constraint on genuine co-performance, at both the auditory and visual level.

The present work sits within this layer, addressing a question that this literature has largely taken as settled rather than examined: the choice of audio representation on the wire under the latency budget. At the level of audio transmission, the field typically frames the choice as one between two approaches: the first sends uncompressed PCM, which keeps latency to a minimum and reproduces the signal with exact fidelity, but requires a fixed and substantial bandwidth (768 kbps for a single 48 kHz, 16-bit channel) and provides no built-in robustness against packet loss; the second approach depends on a low-latency lossy codec, most often Opus [5]. Opus operates well below the uncompressed rate and provides packet loss concealment natively [6]. Its cost is twofold: first, the coding is lossy; second, its algorithmic delay, though low relative to storage codecs, is not negligible against the NMP latency budget. That delay is the frame length plus CELT's 2.5 ms look-ahead, so Opus reaches its 5 ms floor only at its shortest 2.5 ms frame. Rate and latency are moreover coupled: shortening the frame toward the low-latency regime lowers coding quality at a fixed bitrate (equivalently, it raises the bitrate needed for a given quality), introducing an inherent trade-off between latency and coding efficiency. Between these poles lies lossless coding, which would cut bandwidth without discarding signal. Because FLAC-style forward-adaptive prediction derives its model from the current block alone, with no look-ahead beyond it, the algorithmic delay of such a codec is just the block length itself: unlike Opus, it adds no look-ahead term on top of the packetization delay, so a block of N samples costs exactly N/f_s of coding latency and nothing more. The only remaining question is the computational cost

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

of encoding and decoding a block, which is what we measure.

Lossless coding has been tried only sporadically in NMP [7], and to our knowledge has not been systematically characterized in the per-block, low-latency regime that decides real-time viability. This paper addresses that gap. We reformulate FLAC [8]’s lossless model into a pure, header-free, block-reentrant codec, and instrument it to expose, for each independently decodable block, across block sizes, the joint trade-off between compression and encode/decode latency. The contribution’s aim is to provide concrete quantification of this trade-off, measured in encoding/decoding time and coding gain. The rest of this manuscript is organized as follows: Sec.II presents a characterization of related audio codecs and the NMP usual practices, Sec.III introduces the basics of lossless codecs with a particular focus on FLAC [8], enabling the understanding of Sec.IV which describes the derivation of QLAC (Quick Lossless Audio Codec) from FLAC and the implementation choices. Sec.V assesses the viability of QLAC as a replacement for uncompressed PCM along three axes: the encode time, the decode time and the size of encoded blocks. Sec.VI discusses the limitations and the possible improvements to this work. Finally, Sec.VII draws the conclusions.

II. RELATED WORK

A widely cited survey of the field [1], though now some years old, frames a tension that has largely persisted: general-purpose high-fidelity codecs add algorithmic delay it considers hardly acceptable for the NMP delay budget, leading several frameworks to stream uncoded (Pulse Code Modulation, PCM) audio and sacrifice a codec’s bandwidth savings rather than add to that budget. Our contribution intervenes precisely here, so we organize the related work around three axes: low-delay lossy coding, low-delay lossless/hybrid coding, and the compression stance of deployed NMP tools.

A. Low-delay lossy coding

The dominant open codec for interactive audio is Opus [5], which fuses the SILK (speech) and CELT (music) layers and scales to algorithmic delays as low as 5 *ms*, below AAC-LD [9]’s 20 *ms*, which is what makes it usable for network music performance [6]. Opus is open and royalty-free, and is the de-facto choice when some compression is acceptable. On the perceptual/predictive side, Fraunhofer’s Ultra Low Delay (ULD) coder reaches roughly 5.3 *ms* algorithmic delay at 48 kHz [10], but it is licensed as source or object code through Fraunhofer IIS rather than openly available. A transmission and coding scheme tailored to NMP was later proposed to make it more resilient to lost and late packets [11], and the codec is offered as one of the selectable options, alongside CELT, in the SoundJack framework [12]. The AAC-LD/-ELD family ($\approx 15 - 20$ *ms*, licensed) and, in the consumer/industrial space, Qualcomm’s aptX Low Latency [13] (≈ 40 *ms*, proprietary, Bluetooth-only, lossy), which targets lip-sync, round out this category.

B. Low-delay lossless and hybrid coding

The most direct precedent is the integration of WavPack into an NMP system by Kurtisi and Wolf [7], who exploited WavPack’s purely backward-adaptive, low-latency operation to run it with 128-sample blocks and no additional buffering. Because WavPack’s block headers dominate the packet at such small block sizes, they introduced a transport-layer repacking scheme that strips redundant header fields (from 64 to 16 bytes per mono block), reconstructing format-compliant blocks at the receiver. Operating WavPack in its *lossy*, target-bitrate mode, they reported near-transparent quality at 4 bits/sample (PEAQ/ODG), significantly outperforming their earlier AD-PCM scheme. They rejected WavPack’s own low-latency variant because it reduces header overhead only by introducing inter-block dependencies, which is unacceptable for loss-prone NMP transport.

WavPack is open source (BSD-licensed) and provides both a lossless mode and a distinctive lossless/lossy *hybrid* mode, in which a lossy base stream is paired with a correction stream that together restore the original signal bit-for-bit [14]. Its decorrelation is backward-adaptive: the predictor weights adapt sample-by-sample from previously decoded data instead of being retransmitted, so each block carries only a compact set of initial decorrelation terms and weights rather than a full coefficient set [7]. This keeps per-block overhead low, which is what makes the codec efficient at small block sizes; the low latency itself follows from operating on small blocks in the time domain without look-ahead buffering [7]. A dedicated low-latency/streaming variant does exist but has remained peripheral to the mainline codec: it originated as an experimental super/sub-block branch [15] and was later reworked into a separate streaming library [16], neither of which is part of the standard WavPack releases. Notably, a feature request on the WavPack issue tracker observes that, Opus aside, there is essentially no open-source codec that is simultaneously low-latency and hybrid/lossless [17].

MPEG-4 ALS [18] is the standardized point of comparison for forward-adaptive lossless coding: like FLAC [8] it uses forward-adaptive *linear predictive coding* (LPC) with the quantized coefficients transmitted alongside an entropy coded residual, and it is directly relevant to us because it already confronts the block-independence problem through random-access frames, using progressive-order prediction to avoid needing samples from the previous frame at a random-access point. MPEG-4 ALS is, however, comparatively heavy: its multichannel and RLS-LMS [19] tools make it markedly more complex than other lossless coders and demand more processing power, and it retains its own container.

In the industrial/proprietary space, Qualcomm’s aptX Lossless delivers CD-quality lossless audio but requires Qualcomm chipsets at both endpoints and a link of roughly 1.2 *Mbps*, falling back to lossy modes otherwise; it is a closed Bluetooth technology, not usable for IP-based NMP.

FLAC itself deserves specific treatment, since our codec derives from it. FLAC is an open, patent-free lossless codec

using LPC/fixed prediction and Rice coding, whose frames are already independently decodable, each carrying a byte-aligned sync code plus header fields (sample rate, bit depth, channels, frame/sample number) and a per-frame Cyclic Redundancy Check (CRC) so that a decoder can resynchronize mid-stream; the format’s design priorities are archival and seekability rather than minimal per-block cost.

FLAC has in fact already been named as an NMP codec candidate: Gu *et al.* [20] selected stock FLAC for their real-time-rehearsal path, running it at the audio device’s minimum input-buffer size and reporting a client-side processing delay as low as 10.5 *ms* for their real-time codecs. That use, however, treats FLAC as an off-the-shelf black box at default settings (compression complexity 5): the per-block framing and header overhead is left untouched, and the reported distortion evaluation is in fact carried out on ADPCM and MP3 rather than on FLAC, so no block-level characterization of FLAC is given. To the best of our knowledge, no prior work reduces FLAC to a header-free, per-block reentrant codec for the sub-block-size latency regime.

C. Compression practice in deployed NMP tools

The tools fall into two clear families.

The uncompressed-PCM family sends linear samples to eliminate codec delay entirely: JackTrip [21], the reference system from CCRMA, is built on sending uncompressed audio to avoid the latency introduced by compression encode/decode; LOLA [22] transmits audio and video with no quality-compromising compression over fibre, keeping latency below the $\approx 30 - 35$ *ms* perceptual threshold, at the cost of being proprietary and hardware-expensive; and recent web-based work by Sacchetto *et al* [23] deliberately bypasses the WebRTC audio pipeline to carry uncompressed stereo PCM, reaching 10 – 15 *ms* native and ≈ 40 *ms* in-browser with quality superior to mainstream conferencing tools.

The lossy-codec family accepts a small coding delay for bandwidth: Jamulus [24] is built entirely on Opus, and SonoBus [25] lets the user choose per-peer between full uncompressed PCM (16/24/32-bit) and low-latency Opus at various bitrates, where the Opus path adds about 2.5 *ms* of delay at each end. Lossless compressed transport is notably rare in this landscape: the WavPack integration above is essentially the only worked example.

The common assumption is that the only options are raw PCM or lossy Opus, with nothing “low-latency” in between.

III. BACKGROUND

A. The two-stage lossless model

Lossless audio coders share a common two-stage architecture: a decorrelation stage that removes the short-term predictability of the waveform, followed by an entropy-coding stage that assigns near-optimal codewords to what remains. Because the transform is exactly invertible, and the entropy code is uniquely decodable, the reconstruction is bit-identical to the input; compression arises entirely from the fact that the decorrelated signal, the prediction residual, carries far lower

entropy than the raw samples, being concentrated around zero and approximately Laplacian in distribution. This backbone is shared, with variation only in the specific predictor and entropy coder, by essentially every open lossless codec of the last two decades, FLAC and WavPack, among them.

The design space is thus largely defined by two orthogonal choices: how the predictor is obtained, and how the residual is entropy-coded.

B. Prediction

The predictor estimates each sample $\hat{x}[n]$ from a combination of preceding samples, and the coder transmits the residual $e[n] = x[n] - \hat{x}[n]$. Two predictor families are relevant here, both used by FLAC.

- **Fixed** (polynomial) prediction. The simplest predictors are the fixed integer polynomial predictors of orders 0-4, which model the signal as a low-degree polynomial and predict each sample from a fixed pattern of its predecessors. Order 0 transmits the sample itself; order 1 predicts $x[n] \approx [n - 1]$ and therefore transmits the first difference; higher orders transmit successive differences, each stage flattening a progressively smoother signal. Their decisive property is that the coefficients are neither signal-dependent nor transmitted: encoder and decoder apply the same fixed rule, so a fixed predictor costs nothing beyond a small type field and a handful of warm-up samples. This makes them cheap and robust, and frequently optimal on short or quasi-stationary blocks, where fitting a bespoke filter would not repay the cost of transmitting it.
- **Linear predictive coding** (LPC). For more structured material the coder fits a bespoke FIR predictor to each block. In the forward-adaptive scheme used by FLAC and by MPEG-4 ALS, the block is first tapered by an apodization window to reduce edge effects, its short-term autocorrelation is computed, and the Levinson-Durbin recursion solves the normal equations for the optimal coefficients of a chosen order (up to 32 in FLAC). Since these coefficients are real-valued, they are quantized to a fixed precision with an accompanying shift. This is the defining cost of forward adaptation: the quantized coefficients, the precision and shift, and the order-many warm-up samples must all be transmitted alongside the residual, so that the decoder can reconstruct the identical filter.

The alternative, backward-adaptive prediction as used by WavPack, instead derives the filter weights on both sides from already-reconstructed past samples and transmits no coefficients at all, at the price of carrying adaptive state across samples. FLAC’s choice of forward adaptation means the predictor is re-derived independently for every block and fully described within it. This property guarantees complete independence between blocks, which makes the algorithm suitable for the unreliable transmission of the NMP scenario.

For each block the encoder estimates the coded cost of the competing predictors, the constant and verbatim fallbacks,

each fixed order, and each LPC order and window, and retains the cheapest, so predictor selection is itself a per-block, rate-driven decision. One further step precedes prediction: when every sample in a block shares a run of trailing zero bits, as happens when the source was quantized at a lower effective depth or scaled by a power of two, those *wasted bits* carry no information. FLAC detects this common right shift, removes it before coding, and records only the shift amount, restoring the bits after reconstruction; this is why our block header carries a wasted-bits flag and an optional shift count.

C. Entropy coding of the residual

The residual is entropy-coded with Rice coding, a computationally trivial special case of Golomb coding that is near-optimal for the roughly Laplacian, zero-centered residuals that prediction produces. Each residual is coded under a single parameter b that splits it into a unary-coded high part and a b -bit verbatim low part, so that small residuals cost close to $b + 1$ bits; b trades the length of the unary prefix against the width of the fixed suffix, and its optimal value, which tracks the mean residual magnitude, is selected from the residual statistics. Because a single anomalous run of large residuals can make Rice coding wasteful, an escape mechanism lets the code fall back to storing residuals as raw, fixed-width values.

D. The FLAC structure

A FLAC file is built as a set of nested containers, each one wrapping and adding a service to the level below. Moving inward, these levels are the stream, the frame, the subframe, and the partitioned residual. The outer levels exist to make the stream self-describing, seekable, and robust to transmission errors. The inner levels hold the linear-predictive model that performs the compression. This division is what the present work exploits, since it keeps the inner model intact and discards the outer machinery.

At the format level, the stream is opened by the `fLaC` sync word and a mandatory `STREAMINFO` block, followed by any optional metadata. `STREAMINFO` declares the sample rate, the channel count, the bit depth, the minimum and maximum block sizes, and an MD5 checksum of the decoded audio. Optional blocks may add seek tables, tags, and application data. All of this is written once, at the head of the stream. Its role is archival and broadcast oriented, letting any decoder identify the content, size its buffers, and reach an arbitrary position without external information.

The stream is then divided into frames, and each frame is the layered unit that carries the audio:

- **Frame:** A frame covers a fixed run of samples across all channels and is the unit of independent decoding and of mid-stream resynchronization. It opens with a header holding a sync code, a blocking-strategy bit, and coded fields for block size, sample rate, channel assignment, and bit depth. The header also carries the frame or sample number as a variable-length integer, and closes with a CRC-8. A CRC-16 then closes the whole frame. These fields let a decoder that joins late, or that hits a bit error,

find the next frame boundary and check its integrity. They are repeated for every frame.

- **Subframe:** Inside a frame, each channel becomes one subframe. Stereo material may first pass through inter-channel decorrelation, which stores a mid and side pair instead of left and right when that is cheaper. Each subframe then chooses one predictor among four types:
 - **Constant:** a single repeated value, for silence or held levels.
 - **Verbatim:** the raw samples, the incompressible fall-back for noisy signals.
 - **Fixed:** a polynomial predictor of order zero to four, with no transmitted coefficients.
 - **LPC:** a linear predictor whose quantized coefficients travel with the subframe.

The Constant and Verbatim types are self-contained. The Fixed and LPC types are the ones that model the signal, and each is followed by its coded residual.

- **Partitioned residual.** The residual of a Fixed or LPC subframe is Rice coded. Rather than use one parameter b for the whole block, FLAC may split the residual into 2^p partitions, each carrying its own b , under a partition-order field (p) and an entropy-method-type field. Partitioning lets the parameter track local variation in residual magnitude within the block, improving the fit on non-stationary signals at the cost of the extra per-partition parameters. The entropy-method-type field also encodes the escape case, in which a partition stores its residuals verbatim at a fixed width rather than Rice coding them.

The header, resynchronisation, and residual-layout fields are what make a FLAC stream self-describing and seekable, but they are fixed costs paid once per frame whatever the audio contains. Their relative weight grows as the block shrinks, which is exactly the regime that low-latency transport demands. It is this separation that the following section develops.

IV. METHODOLOGY

A. Derivation of QLAC

The derivation of QLAC from FLAC follows a single principle: keep FLAC’s intra-block prediction and residual-coding scheme exactly as described in Sec.III, and remove everything that exists only to serve a continuous, self-describing, multichannel stream.

The result is a pure per-block codec in which each call encodes or decodes one independent block and nothing else, with a single header byte represented in Fig. 1 and a payload layout depicted in Fig. 2.

The DSP kernels that implement prediction, windowing, entropy coding and bit-level serialization are inherited with few adaptations. This is a deliberate methodological control: because the predictor and residual model are unchanged, any measured difference in rate or latency between QLAC and stock FLAC is attributable to the structural changes described below rather than to a weaker or re-tuned coder.

0	Frame type (ft) 6 bits	Wb	Payload...
0x0: constant 0x1: verbatim [0x8, 0xC]: fixed with order (value - 8) [0x20, ...]: lpc with order (value - 32)		Wasted bits field	

Fig. 1: QLAC’s block header byte

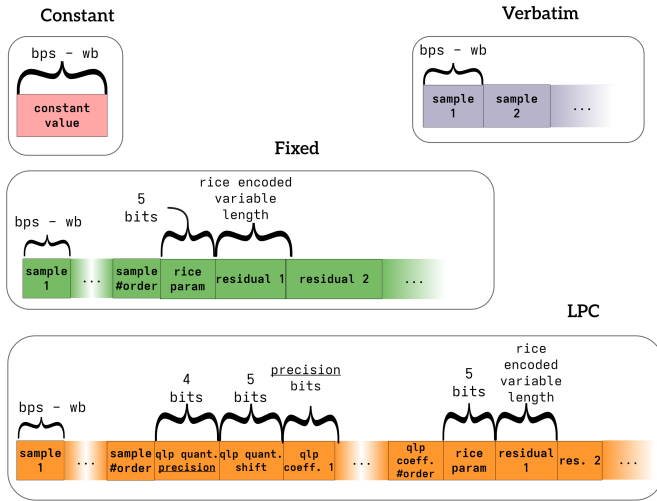


Fig. 2: The four payload layouts of QLAC. “bps” is the original bits-per-sample value (e.g., 16 for 16-bit PCM), and should be known a priori by the decoder. “wb” is the wasted bits count, a unary-coded value optionally preceding the payload if the Wb flag is set in the header. “qlp” stands for “quantized linear predictor”; the precision is the number of bits for each coefficient, and the shift is the exponent, equal for all the qlp coefficients. *Note: if the prediction error’s power is too high, the encoder does not use the Rice encoding but a fixed number of bits for the residual; this number would be encoded as an additional 5-bit field after “rice param”.*

Stock libFLAC exposes a push/pull streaming state machine: the caller feeds interleaved multichannel samples, the library buffers them, chooses block boundaries, performs inter-channel decorrelation, prepends stream and frame headers with sequence numbers and CRCs, and drives output through write callbacks; the decoder mirrors this with sync search, metadata parsing and error recovery. QLAC replaces this with a single stateless function on each side: encode and decode, each the exact inverse of the other. The encoder holds no input queue and no cross-block state. Every parameter that FLAC would negotiate on the wire, namely channel count (fixed at one, mono), sample rate, bit depth and block size,

is instead fixed out of band for the lifetime of an instance. These values are either provided by the application layer or negotiated between the two hosts before the stream begins or upon a configuration change. Analysis windows and all working buffers are sized once at initialization, so a coder instance is a pure function of its configuration, with no per-block allocation. Buffers are caller-owned and SIMD-aligned, permitting zero-copy operation on a single long-lived instance. The resulting block independence is the property that makes per-block latency measurement meaningful and the output suitable for real-time, independently recoverable transport. Because a block is now a single mono element, FLAC’s nested frame/subframe hierarchy, an array of one subframe, itself an entropy-coding-method wrapper around a residual, is collapsed into a single flat per-block descriptor. Each block records the predictor type and the presence (and number if present) of wasted bits, followed by only what that type requires: a single constant value, the verbatim samples, or, for fixed and linear prediction, the predictor order, the warm-up samples, and the residual together with its Rice parameter and, if applicable, the escape width. Linear prediction additionally carries the quantized coefficients with their precision and shift. Entropy coding is reduced to a single case as well, with a five-bit Rice parameter.

FLAC adapts the entropy code within a block by searching a partition order, splitting the residual into 2^p partitions each with its own Rice parameter. QLAC collapses this to a single partition: one Rice parameter and one possible escape width per block, estimated from the residual’s mean magnitude. This trades FLAC’s intra-block parameter adaptation, whose benefit shrinks as the block shrinks, for a smaller fixed descriptor, and is directly observable in the recorded parameter and escape statistics.

With the streaming machinery removed, a QLAC block carries none of the stream- and frame-level scaffolding enumerated in Sec.III. There is no frame header of any kind: the block begins directly with a one-byte subtype field: a reserved bit, six subtype bits and a wasted-bits flag, optionally (if wasted bits flag is set) followed by a unary wasted-bits value, then the predictor payload, and is finally zero-padded to a byte boundary. The frame sync code and the blocking/rate/channel/depth descriptors, the coded frame or sample number, the header and frame CRCs, the entropy-method-type (only FLAC’s “rice2”, since no backward compatibility is needed) and partition-order fields are thus all absent, as they are considered to be implemented by the transport/application protocol if needed.

The decoder is the exact inverse and correspondingly minimal.

B. Setup

In order to test the viability of QLAC as a replacement for uncompressed PCM in NMP applications, an assessment code has been developed that collects the three features to be evaluated:

- **Encoding time:** the time each block takes to be encoded, depending on the block size
- **Decoding time:** same as the encoding time, with lower measurement reliability due to the very small values
- **Compression ratio:** measured as the ratio between the compressed block and the original block. The original size was estimated as $bit_depth \times samples_count$.

Throughout, the block size is the natural latency axis: since a block of N samples must be filled before it can be coded and sent, it carries an intrinsic *packetization latency* of N/f_s , which for QLAC is also the codec’s entire algorithmic delay. The block sizes we test are therefore directly readable as delays at the sampling rates considered, and the timings reported below are the only additional cost on top of that packetization delay.

The program is a C++20 Linux-only code wrapping the C QLAC API. It reads wav files (all monaural in our case), loading them entirely in memory, and deinterlacing if they have more than one channel. It then detaches a thread, set to use the First-in First-Out (FIFO) scheduler on Linux, which belongs to the Real-Time (RT) class, to better represent the case of an RT audio application. The newly created thread iterates on each channel of the preloaded input audio file and each of the different possible block-sizes the program accepts as arguments. If the number of samples is not an integer multiple of the block-size, it is trimmed to the largest multiple of the block size. The encoder and decoder data structures are created only once, at the beginning of the process, for each block-size. The code then measures the time used to encode and decode each block. Optionally, the program can be compiled to verify that pre-encode and post-decode data are equal sample-wise. For each encoded block the statistics are written in an in-memory array, dumped to a CSV file after the end of the process, guaranteeing no Input/Output in the RT path.

The tests are executed on a Raspberry Pi 4B, which is a very common platform that simplifies reproducibility, with the SoC clocked at 1.8 *GHz* and 8 *GB* of memory, running a 64-bit Linux PREEMPT_RT kernel. The software was compiled to exploit the NEON [26] acceleration available on this ARM device: the most demanding LPC routines, namely the autocorrelation and residual computations, are explicitly written using NEON intrinsics and selected at runtime when available. The codec was set with no particular settings, but the maximum compression level (corresponding to the 8th FLAC compression level, with maximum LPC order equal to 12 and apodization window set to “*subdivide_tukey(3)*”).

C. Dataset

Two datasets were used for the tests:

- 1) **Medley-Solos-DB** [27]: provides a collection of 8 categories of solo musicians:
 - Clarinet
 - Distorted Electric Guitar
 - Female singer

- Flute
- Piano
- Tenor Sax
- Trumpet
- Violin

. The dataset is composed of a large collection of 3-second wav files (2^{17} samples, sampled @ 44.1 *kHz*), with 32-bit floating point samples. Due to its size, the dataset was subsampled by concatenating 232 randomly drawn files per category. Since the block size is always a power of 2 in our experiments, it is a divisor of the original size of each file, so a block never crosses the end or beginning of any original file in any experiment. Furthermore, since FLAC does not work with floating point samples, we generated signed 16-bit and signed 32-bit PCM versions of each input wav file using FFmpeg (`pcm_s16le` and `pcm_s32le`, respectively). No additional amplitude normalization or other audio processing was applied during this conversion; thus, the original floating-point amplitudes were retained and quantized to the corresponding integer sample format. Afterwards, we tested the codec on both.

- 2) **Bach Cello Suite Dataset** [28]: The dataset contains only cello music. It was designed with NMP in mind, having served as a benchmarking dataset for packet loss concealment techniques. As the original article states: “these are extremely raw, unedited recordings including instrument tuning, scales practice and occasional improvisations”. A subset of the original dataset of almost 9 hours was used. In this case files contain 16-bit integer samples, @ 48 *kHz*.

V. RESULTS

A. Compression ratio

The results are promising both in terms of compression ratio and encoding/decoding time. Table I and table II report the compression ratio measured as $B_{\text{compressed}}/B_{\text{original}}$ (with B being the size in bytes), varying the reference dataset on the columns and the block size on the rows. The upper limit on the average compression ratio was ≈ 0.7 , meaning that lossless coding saves at least $\approx 30\%$ of the bandwidth used by the application payload. The most interesting line is the one for block size 128 in I, which refers to 16-bit encoding; it represents the typical values of many NMP applications, corresponding to ≈ 2.9 *ms* of packetization latency at 44.1 *kHz* (see Sec.IV). This line shows a maximum average compression ratio of 0.5 for the singer’s data. On average, the algorithm saves 60% of the application payload.

B. Speed

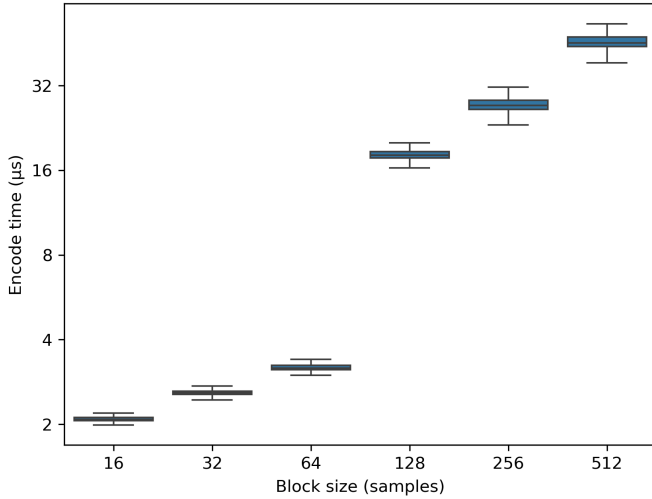
1) *Encoding speed:* Fig. 3a and Fig. 3b show the encoding time for 16-bit and 32-bit audio, respectively. Encoding time grows linearly with the block size, with a noticeable discontinuity between the block sizes of 64 and 128 in both cases. Outliers have been removed from the graph by simply hiding the boxplot fliers, but the maximum recorded values are

	BCSD	Clarinet	Distorted Electric Guitar	Female Singer	Flute	Piano	Tenor Sax	Trumpet	Violin
16	0.48	0.49	0.62	0.62	0.49	0.48	0.63	0.54	0.58
32	0.41	0.40	0.53	0.56	0.40	0.39	0.55	0.45	0.50
64	0.38	0.36	0.49	0.52	0.35	0.34	0.51	0.40	0.45
128	0.36	0.33	0.45	0.50	0.33	0.31	0.47	0.38	0.41
256	0.34	0.31	0.42	0.48	0.30	0.29	0.45	0.35	0.38
512	0.33	0.30	0.40	0.46	0.29	0.28	0.43	0.34	0.36

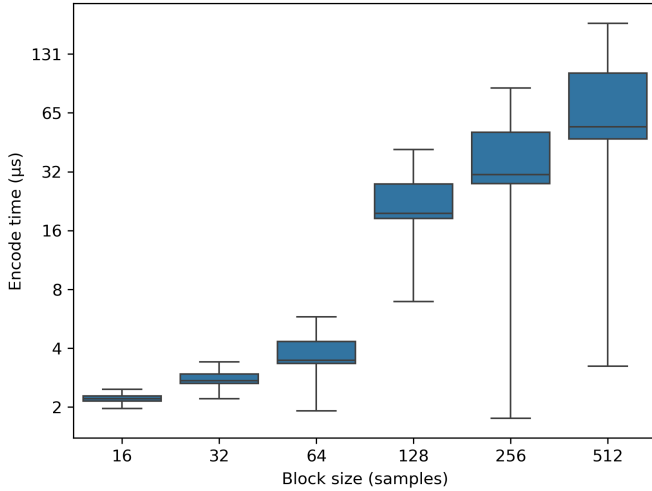
TABLE I: Average Ratio between compressed size and original size for each instrument and block size from 16 bits original wav. BCSD is Bach Cello Suite Dataset

	Clarinet	Distorted Electric Guitar	Female Singer	Flute	Piano	Tenor Sax	Trumpet	Violin
16	0.53	0.37	0.37	0.69	0.48	0.64	0.60	0.57
32	0.49	0.31	0.32	0.65	0.43	0.61	0.56	0.53
64	0.47	0.28	0.30	0.63	0.41	0.60	0.54	0.51
128	0.46	0.26	0.29	0.61	0.39	0.58	0.53	0.48
256	0.44	0.24	0.27	0.59	0.38	0.56	0.50	0.46
512	0.42	0.23	0.26	0.57	0.36	0.54	0.49	0.44

TABLE II: Average Ratio between compressed size and original size for each instrument and block size from 32 bits original wav



(a) Results for 16-bit audio



(b) Results for 32-bit audio

Fig. 3: Encode time distributions, varying the block size. Both axes use logarithmic scales.

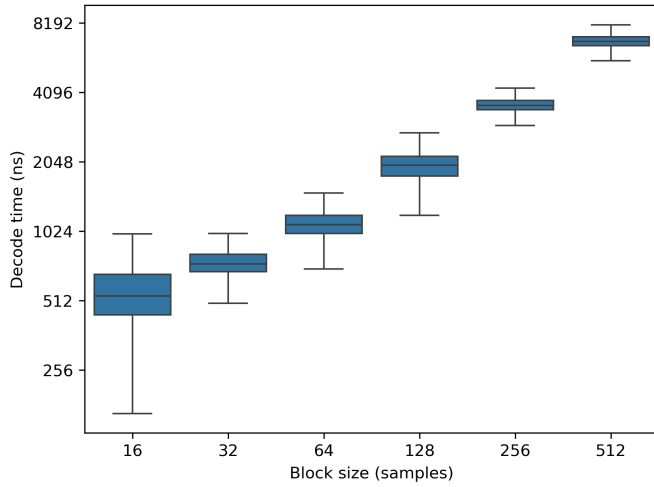
between $200 \mu s$ and $300 \mu s$, whereas the minima are always around $1.5 \mu s$ corresponding to constant blocks. The median value for block size 128 is $18.555 \mu s$, which corresponds to an $\approx 156\times$ speed w.r.t. the playback at $44.1 kHz$ and $\approx 144\times$ at $48 kHz$.

2) *Decoding speed:* The decoding time (Fig. 4) also grows linearly with the block size, and there is little difference between the 16-bit and the 32-bit case. The cost per sample is higher at lower block sizes, with a median around $32 ns$ per sample for the 16-sample block. Unlike the encoding phase, which stays constant regardless of the number of connected peers in the typical peer-to-peer broadcast scenario common in NMP, decoding scales linearly with the total number of received audio streams. The aforementioned figure implies a decoding speed of $1417\times$ at a $44.1 kHz$ sampling rate and $1302\times$ at $48 kHz$ (for a block size of 128 samples), keeping decoding well within reach in every realistic NMP setup.

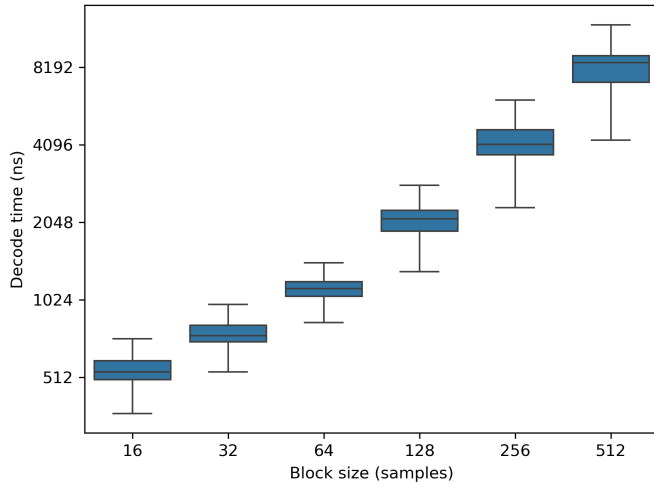
VI. LIMITATIONS AND FUTURE WORK

Although this work demonstrates that lossless compression comes at a negligible encoding cost in NMP applications, saving on average about half of the bandwidth, several limitations in the assessment deserve discussion. The encoder settings have not been widely explored: the most aggressive compression techniques available in FLAC have been retained in QLAC rather than pruned.

- The subdivide-Tukey apodization method with 3 partitions is costly, fitting up to 9 linear predictors per block, a computational cost that can probably be saved most of the time.
- An LPC of order 12 is probably rarely selected with small block sizes; limiting the order to lower values could further speed up the encoding process.
- Non-accelerated implementations have not been tested. Embedded devices have been quite popular in NMP (e.g., in [29]), and although most of them have DSP/SIMD capabilities, and therefore hardware floating-point operations, characterizing the timings of the integer-only



(a) Results for 16-bit audio



(b) Results for 32-bit audio

Fig. 4: Decode time distributions, varying the block size. Both axes use logarithmic scales.

version could provide useful insights into the capabilities of QLAC in embedded setups.

- The decoding-time procedure is debatable, since measuring sub-microsecond times introduces offsets and variability in the measurement process itself. Averaging over many decodes at a time could yield more precise results; nevertheless, decoding is fast enough to be safely considered a non-bottleneck, as it is still roughly $300\times$ real time at a sampling rate of 192 kHz .
- Time measurements may produce some outliers, as the Linux kernel still occasionally interrupts FIFO threads through the real-time throttling mechanism [30]. This can be disabled, or a small sleep call could be added after every n encode/decode executions to reset it. This most likely affected only a few outliers, as some cache misses probably did, and did not impact the reported results; still, a more robust evaluation could be performed.

Beyond a wider set of experiments varying more encoder features and target platforms, the main extension of this work would be a comparison with other encoders along all the considered objectives, the most prominent example being the WavPack implementation for NMP.

VII. CONCLUSION

This work re-examines a common assumption in the Networked Music Performance community: that in the low-latency regime the only viable transmission strategies are uncompressed PCM or lossy coding, almost invariably Opus, with nothing in between. By deriving QLAC from the FLAC reference implementation, retaining its intra-block predictor and residual model unchanged while stripping away every stream- and frame-level construct that serves continuous, self-describing, multichannel storage rather than minimal per-block cost, we obtain a pure per-block reentrant lossless codec whose blocks are independently decodable, a property well matched to the unreliable, delay-critical transport of NMP. These properties are especially pertinent to the Musical Metaverse, where NMP is the foundational auditory layer and the audio plane must scale to many concurrent participants within the same tight latency envelope. By halving the per-stream payload while keeping decode cost far below real time, QLAC relaxes the bandwidth pressure of many-to-many co-presence without compromising fidelity.

The characterization confirms that lossless transmission occupies a favorable point in this space. Across two datasets and a range of block sizes relevant to real-time operation, QLAC reduces the application payload by roughly 30% to 70% with respect to uncompressed PCM, with the largest savings on quasi-stationary material. This bandwidth reduction comes at a negligible cost: on a Raspberry Pi 4B, a common and reproducible embedded platform, encoding a 128-sample block runs $\approx 156\times$ faster than real time and decoding faster still, leaving substantial headroom within the latency budget. Lossless compression therefore emerges as a concrete third option, reducing bandwidth without sacrificing signal fidelity or exceeding the temporal constraints that define the field.

The source code of QLAC is available at

<https://github.com/leosev/qlac>.

REFERENCES

- [1] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, "An Overview on Networked Music Performance Technologies," *IEEE Access*, vol. 4, pp. 8823–8843, 2016.
- [2] L. Turchet, "Musical Metaverse: Vision, opportunities, and challenges," *Personal and Ubiquitous Computing*, vol. 27, no. 5, pp. 1811–1827, Oct. 2023.
- [3] A. Boem, M. Tomasetti, and L. Turchet, "Issues and Challenges of Audio Technologies for the Musical Metaverse," *Journal of the Audio Engineering Society*, vol. 73, no. 3, pp. 94–114, Apr. 2025.
- [4] C. Rinaldi and C. Centofanti, "The musical metaverse: Advancements and applications in networked immersive audio," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, pp. 1–7.
- [5] J.-M. Valin, K. Vos, and T. B. Terriberry, "Definition of the Opus Audio Codec," RFC 6716, 09 2012. [Online]. Available: <https://www.rfc-editor.org/info/rfc6716>
- [6] J.-M. Valin, G. Maxwell, T. B. Terriberry, and K. Vos, "High-Quality, Low-Delay Music Coding in the Opus Codec," 2016.
- [7] Z. Kurtisi and L. Wolf, "Using wavpack for real-time audio coding in interactive applications," in *2008 IEEE International Conference on Multimedia and Expo*. Hannover, Germany: IEEE, Jun. 2008, pp. 1381–1384.
- [8] M. van Beurden and A. Weaver, "Free Lossless Audio Codec (FLAC)," RFC 9639, 12 2024. [Online]. Available: <https://www.rfc-editor.org/info/rfc9639>
- [9] M. Schnell, R. Geiger, M. Schmidt, M. Jander, M. Multrus, G. Schuller, and J. Herre, "Enhanced mpeg-4 low delay aac - low bitrate high quality communication," in *Audio Engineering Society - 122nd Audio Engineering Society Convention 2007*, vol. 3, 05 2007.
- [10] U. Kramer, G. Schuller, S. Wabnik, J. Klier, and J. Hirschfeld, "Ultra low delay audio coding with constant bit rate," *117th Audio Engineering Society Convention*, 01 2004.
- [11] U. Krämer, J. Hirschfeld, A. Carôt, C. Werner, G. Schuller, and S. Wabnik, "Network music performance with ultra-low-delay audio coding under unreliable network conditions," 2007. [Online]. Available: <https://publica.fraunhofer.de/handle/publica/356793>
- [12] A. Carôt and C. Werner, "Distributed network music workshop with soundjack," *Proceedings of the 25th Tonmeistertagung, Leipzig, Germany*, p. 5, 2008.
- [13] "Qualcomm; aptX; Low Latency Synchronised Audio Technology — aptX.com," <https://www.aptx.com/aptx-low-latency>, [Accessed 24-09-2026].
- [14] D. Bryant, "WavPack audio compression," <https://www.wavpack.com/>, accessed: 2026-07-07.
- [15] —, "A low-latency version of WavPack," https://www.wavpack.com/low_latency.txt, experimental branch from WavPack 4.41. Accessed: 2026-07-07.
- [16] —, "wavpack-stream: A library for streaming audio compression, based on WavPack," accessed: 2026-07-07. [Online]. Available: <https://github.com/dbry/wavpack-stream>
- [17] M. Dąbrowski, "Integrate low latency mode into regular releases (issue #86)," 2020, wavPack issue tracker. Accessed: 2026-07-07. [Online]. Available: <https://github.com/dbry/WavPack/issues/86>
- [18] T. Liebchen, "Mpeg-4 als - the standard for lossless audio coding," *The Journal of the Acoustical Society of Korea*, vol. 28, 01 2009.
- [19] Haibin Huang, P. Franti, Dongyan Huang, and S. Rahardja, "Cascaded RLS-LMS Prediction in MPEG-4 Lossless Audio Coding," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 3, pp. 554–562, Mar. 2008.
- [20] Xiaoyuan Gu, M. Dick, Z. Kurtisi, U. Noyer, and L. Wolf, "Network-centric music performance: Practice and experiments," *IEEE Communications Magazine*, vol. 43, no. 6, pp. 86–93, 06 2005.
- [21] 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, Sep. 2010.
- [22] C. Drioli, C. Allocchio, and N. Buso, "Networked Performances and Natural Interaction via LOLA: Low Latency High Quality A/V Streaming System," in *Information Technologies for Performing Arts, Media Access, and Entertainment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 7990, pp. 240–250.
- [23] M. Sacchetto, P. Gastaldi, C. Chafe, C. Rottondi, and A. Servetti, "Web-Based Networked Music Performances via WebRTC: A Low-Latency PCM Audio Solution," *Journal of the Audio Engineering Society*, vol. 70, no. 11, pp. 926–937, Dec. 2022.
- [24] V. Fischer, "Case study: Performing band rehearsals on the internet with jamulus," 2015, accessed: 2026-07-06. [Online]. Available: <https://jamulus.io/PerformingBandRehearsalsoontheInternetWithJamulus.pdf>
- [25] J. Chappell and Sonosaurus LLC, "SonoBus: Source code for a real-time network audio streaming collaboration tool," 2024, accessed: 2026-07-06. [Online]. Available: <https://github.com/sonosaurus/sonobus>
- [26] Arm Limited, *Introducing Neon*, Arm Limited, 2024. [Online]. Available: <https://developer.arm.com/documentation/dht0002/a>
- [27] V. Lostanlen, C.-E. Cella, R. Bittner, and S. Essid, "Medley-solos-DB: A cross-collection dataset for musical instrument recognition," Sep. 2019.
- [28] P. Verma, A. I. 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)*. Trento, Italy: IEEE, Sep. 2020, pp. 268–275.
- [29] L. D. Murray and S. Fasciani, "Characterisation of Teensy 4.1 Ecosystem for Low-Latency Audio Network Transmission in Networked Music Performances," in *2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*. L'Aquila, Italy: IEEE, Oct. 2025, pp. 1–10.
- [30] Red Hat, Inc., "Real time throttling," 2026, accessed: 6 July 2026. [Online]. Available: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/real_time_throttling