L. Vignati, S. Zambon, and L. Turchet, "A Comparison of Real-Time Linux-Based Architectures for Embedded Musical Applications"
J. Audio Eng. Soc., vol. 70, no. 1/2, pp. 83–93, (2022 January/February).
DOI: https://doi.org/10.17743/jaes.2021.0052

A Comparison of Real-Time Linux-Based Architectures for Embedded Musical Applications

LUCA VIGNATI, STEFANO ZAMBON, AND LUCA TURCHET, *AES Associate Member*
(luca.vignati@unitn.it) (stefano@elk.audio) (luca.turchet@unitn.it)

Department of Information Engineering and Computer Science, University of Trento, Trento, Italy Modern Ancient Instruments Networked dba Elk, Stockholm

The Internet of Things (IoT) is fostering advancements in the embedded systems world, widening the range of available single-board computers and lowering their price. The Internet of Musical Things (IoMusT), the IoT musical counterpart, is thriving as well with more and more examples of embedded devices useful to build connected musical interfaces. For this purpose, real-time architectures based on the Linux operating system are increasingly used. In this paper, we compare two radically different approaches to real-time Linux audio: one system is based on the PREEMPT_RT patch and the ALSA framework and the other on the Xenomai patch and the Elk Audio OS. Our study aims at providing audio developers working on IoMusT devices and applications with a clear quantitative picture of how these two systems compare. Our results reveal that Xenomai provides lower audio round-trip latency, lower scheduling latency, and manages to exploit more CPU performance at a given latency setting while guaranteeing perfect audio quality. Nevertheless, PREEMPT_RT still delivers good performance, and it is widely supported resulting in a more accessible alternative. All the tests have been carried out on the Raspberry Pi 4B single-board computer combined with the HiFiBerry expansion HAT.

0 INTRODUCTION

The increasing availability of single-board computers and audio prototyping tools have fostered the development of embedded audio frameworks specifically targeting the design and construction of digital musical instruments [1]. In the past few years various Linux-based platforms have been proposed for this purpose, mainly targeting the makers community (for a recent comparative study see [2]). A prominent example is Satellite CCRMA [3, 4], which is based on Raspberry Pi or BeagleBoard xM single-board computers connected via a serial port to an Arduino microcontroller and uses open-source software for generating audio. Another similar platform is Prynth [5].

Currently, one of the most advanced platforms in this space (see [2]) is represented by Bela [6, 7], a platform based on the BeagleBone Black single-board computer, which is extended with a custom expansion board featuring stereo audio, eight channels each of 16-bit ADC and 16-bit DAC for sensors and actuators, an I2C port, and 16 GPIO channels. The CTAG face 2|4 [8] is a similar project also based on the BeagleBone that features a high number of analog inputs and outputs. To achieve latencies below 1 ms, they both use the Xenomai real-time kernel extension, which according to the study reported in [9] is the best-performing of the hard real-time Linux environments. Another approach based on Xenomai is Elk Audio OS [10], an operating system that explicitly targets low-latency and high-quality audio applications and eases development across a wide range of embedded hardware.

The frameworks reviewed here are relevant to the construction of a specific class of digital musical instruments, the so-called Musical Things, which belongs to the Internet of Musical Things (IoMusT) paradigm [11]. The IoMusT is an emerging research field positioned at the intersection of the Internet of Things (IoT) and music technology and relates to ecosystems of computing devices embedded in physical objects (Musical Things) dedicated to the production and/or reception of musical content. An example of Musical Things is represented by the class of "smart musical instruments" [12], a family of musical instruments based on embedded systems, which are envisioned to have advanced context-aware and proactive capabilities. Existing instances of such kinds of Musical Things are the Sensus Smart Guitar developed by Elk [13], other prototypes of smart guitars developed in academic contexts (see e.g., [14]), or the Smart Cajón reported in [15].

The development of a Musical Thing, such as a smart musical instrument, is rooted in the embedded hardware and software platform utilized for low-latency audio and sensor processing as well as wireless connectivity. As the Internet of Musical Things (IoMusT) emerges, audio-specific operating systems are required to ease the development and portability of IoMusT applications on embedded hardware. The increasing availability of low-cost System-on-Chip (SoCs) tailored for the IoT market makes the use of Linux more and more appealing for smart musical instruments development. This marks a transition from dedicated DSP systems (widely adopted in the music tech industry) to Linux-based real-time systems, which is made possible thanks to a few frameworks that modify or integrate with Linux to overcome its inherent non–real-time nature.

In this paper, we study the performance of two Linux-based architectures for real-time embedded audio, based respectively on the Xenomai framework and the PREEMPT_RT patch [16]. We aim at comparing their full stack of Xenomai and RASPA (Elk's audio API) against PREEMPT_RT, ALSA/Jack, and the POSIX synchronization primitives. We are interested in quantifying the difference among them in the context of real-time embedded audio, which is something new compared with previous studies that focused only on task-scheduling performance [9]. In particular we are interested in FX processing use cases, which require synchronous input/output operation. Moreover, the benchmark environment described in this paper is fully reproducible because it completely relies on open-source software and widely available hardware. Therefore, such an environment could be used to further validate our results as well as to compare other hardware/software architecture with the one we used in this work. We paid special attention to guaranteeing the fairness and reproducibility of this comparison by tuning both systems to the best of our effort and using the same version of common software when possible.

The remainder of this paper is structured as follows. In Sec. 1 we discuss Xenomai and PREEMPT_RT, and Sec. 2 presents a detailed description of the architectures of the two tested systems. Sec. 3 details test methodologies and results, and finally, we draw concluding remarks in Sec. 4.

1 SELECTED PLATFORMS

1.1 PREEMPT_RT

PREEMPT_RT is a set of patches for the Linux kernel developed by a group of kernel developers. The project was started by Ingo Molnar, and the first release was based on kernel version 2.6.11 (March 2005). The goal of this project is to trade the throughput of the system with predictability and low latency operation while maintaining the single-kernel approach to allow developers to write (user-space) real-time applications easily. In more detail, the PREEMPT_RT patch allows the real-time tasks to preempt the kernel everywhere, even in critical sections. However, some regions can still remain non-preemptible, like the top-half of interrupt handlers and the regions protected by raw spinlocks. During the development of the PREEMPT_RT patch, besides the new features introduced, several positive side effects have been observed. The systems in which this patch is applied are more sensitive to bugs due to latency constraints [17], allowing the kernel developers to use the patch as a stress-testing tool to discover more easily the bottlenecks of the kernel itself. Moreover, the patch introduced several scheduler improvements and analysis tools in the mainline kernel [16].

1.2 Xenomai

Released in 2002 by Philippe Gerum, Xenomai [18] is a layer that enables real-time in user-space. It currently supports more recent architectures than other frameworks such as RTAI: x86, x86_64, ARM, PowerPC, and ia64. Xenomai initially relied on the ADEOS layer [19], but in recent versions, only a simplified part of it has been maintained, in particular the interrupt-delivery subsystem called I-pipe (interrupt pipeline). The last version of Xenomai (version 3, released in 2015) allows the user to choose between the Cobalt (dual-kernel) and Mercury (single-kernel) setups. In the latter version, there is only one kernel carrying out both real-time and non–real-time tasks, so all the determinism improvement is delegated to the PREEMPT_RT patch. Because using a Mercury kernel on a PREEMPT_RT Linux provides virtually no gain in real-time performances compared to a PREEMPT_RT Linux, we will use the more performing Cobalt (dual-kernel) setup for Xenomai. Philippe Gerum is now working on an evolution of the Xenomai/I-pipe framework called EVL Project [20], which represents the future of Xenomai.

2 SYSTEM ARCHITECTURE

This section describes the architectures considered in this study, both hardware and software.

2.1 HARDWARE ARCHITECTURE

2.1.1 Raspberry Pi 4 Model B

The single-board computer (SBC) we chose for the study is the Raspberry Pi 4 Model B. Its SoC from Broadcom (BCM2711) features 4 ARM Cortex-A72 working at 1.5 GHz as well as many peripherals including the Inter-IC Sound (I^2S), Serial Peripheral Interface (SPI), and Inter-Integrated Circuit (I^2C). This model is available with three RAM sizes: 2, 4, and 8 GB. We did not have demanding requirements in terms of memory for this project so we picked the 2-GB model because it is likely to be the most widespread. The board provides additional handy features such as UART, Ethernet Wi-Fi, and Bluetooth connectivity, USB2, USB3, dual 4K video output, and the signature 40 pins Raspberry Pi GPIO connector.

2.1.2 HiFiBerry DAC+ ADC PRO

To extend the audio capabilities of the system, we used the DAC+ ADC PRO expansion board by HiFiBerry [21], which complies with the hardware-attached-on-top (HAT) specification, so it can be connected to the Raspberry Pi through its 40-pin connector. The DAC+ ADC PRO HAT board provides two output channels with RCA connectors and two input channels with one 3.5-mm stereo jack plug.

At the heart of the DAC+ ADC PRO are two integrated circuits: the TI-PCM1863 2-Channel ADC and the TI-PCM5122 Audio Stereo DAC. They both work with sampling frequencies up to 196 kHz and 24-bit Audio Data. The audio data transfer interface is I^2S, and the ICs can be configured via SPI or I^2C. They also feature selectable digital-filter latency, which will be discussed in Sec. 3.2.3.

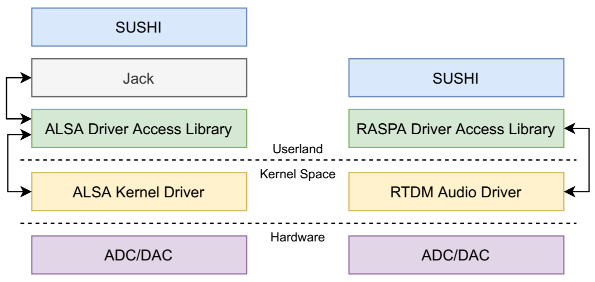


Fig. 1. Driver architecture of the two systems. ALSA is the Advanced Linux Sound Architecture, RASPA is the RTDM Audio-over-SP; API, RTDM is Xenomai's Real-Time Driver Model; DAC is the Digital to Analog Converter; and ADC is the Analog to Digital Converter.

2.2 SOFTWARE ARCHITECTURE

We chose to use 48 kHz as the sampling frequency of both systems because it is the most widespread among audio equipment. Both systems use a 32-bit floating-point precision for audio processing and use the same int to float conversion algorithm.

2.2.1 SUSHI

In both systems, the top-level component is SUSHI [22]. It is a real-time audio plugin host developed by Elk that supports multiple plugin standards. It also features advanced audio and MIDI routing, a simple scripting setup, and is written to ensure high performance and stability under low-latency conditions. It can be controlled through MIDI, OSC, or gRPC [23] protocols, and it is compatible out of the box with both tested systems. This is made possible by its multiple audio frontends, among which there are RASPA and Jack. Thanks to its specific capabilities, SUSHI enabled us to run the exact same code on both systems leveraging its compatibility with PREEMPT_RT through Jack, and with Xenomai through RASPA. Moreover, SUSHI uses a timer to provide accurate information on the ongoing DSP load, from which we fetched the measurements used in Sec. 3.2. For multithreaded work, SUSHI relies on TWINE [24], a library for parallel computing. TWINE uses a high-level API to provide a dual implementation for both Xenomai's and POSIX's synchronization primitives to synchronize parallel tasks.

2.2.2 PREEMPT_RT System

This architecture features the Linux kernel patched with the PREEMPT_RT patch, the Advanced Linux Sound Architecture (ALSA [25]) framework, and the Jack audio server [26]. This is the most widespread approach to real-time audio for Linux mainly due to its simplicity and ease of use, and recently the PREEMPT_RT patch made its way into the mainline kernel, affirming itself as the main real-time solution for the Linux kernel. We used the PREEMPT_RT patch version 4.19.127-rt55 to patch the Linux kernel version 4.16.127 in order to get the real-time–enabled kernel. One of the most significant advantages of this mainline kernel architecture is the possibility to leverage the ALSA framework connecting the codec driver with the machine and platform drivers, all already available. Although the driver may need a few adjustments, this is a much simpler task than writing it from scratch.

Our interest resides in synchronous operation for FX processing, so simultaneous input and output handling is required. ALSA, however, is asynchronous in its nature, meaning that the synchronization between input and output is left to the user-space application. For this reason, we needed to use Jack in order to connect ALSA to SUSHI. As it is shown in Fig. 1, Jack with the ALSA framework constitutes the synchronous audio mechanism required for SUSHI to run. When started, Jack creates a real-time thread with priority -11. The drivers of the HiFiBerry support buffer sizes as small as 64 samples, so lower buffer size settings can not be achieved by this system. Core isolation techniques were also adopted to force every process on the system to run on the first core leaving the other cores dedicated to the real-time tasks. Interrupt requests were isolated similarly, changing the core affinity of the DMA interrupts used by the audio drivers, and setting their priority to 85.

2.2.3 Xenomai System

For the Xenomai system, we used the Elk Audio OS 0.9.5, developed by Elk [10, 27]. At its core, Linux version 4.19.126 is patched with the Xenomai patches version v3.1-rc4, introducing Xenomai's Cobalt real-time co-kernel along with Linux's standard non–real-time kernel. It also features a custom RTDM audio driver and Raspa, a custom userspace library that manages audio formats and memory allocation for audio applications. The RTDM audio driver handles the interrupts triggered by the SPI peripherals, and Raspa provides the user-space with the virtual memory translation of the static address where the samples are placed by the DMA. The RTDM framework makes it possible to do this from the Cobalt real-time core. The buffer sizes, expressed in samples, that are supported by the custom RTDM audio driver are the following: 16, 32, 64, 128, 256, and 512. We also enforced the same core isolation rule we deployed on the PREEMPT_RT system even if it is not critical to this system because we wanted the two systems to experience the same kind and level of stress during the tests.

3 PERFORMANCE COMPARISON

There are three metrics used in this study: DSP load, scheduling latency, and round-trip latency (RTL). For each of these metrics, one or more tests have been carried out. In the following two subsections we will detail the measurement procedure and setup and finally the results of each of them.

3.1 MEASUREMENT PROCEDURE AND SETUP

3.1.1 DSP Load Test

This test aims to measure the maximum DSP load a system can reliably sustain without incurring in underruns (from now on called Xruns) with varying buffer sizes, core count, and plugin type. The variables at play in these tests are the following:

- **buffer size**: it indicates the size in audio frames of the buffer delivered at every interrupt, so for a given sampling rate (48 kHz in our case) it defines the window of available time between interrupts. For Xenomai we tested buffer sizes of 16, 32, 64, 128, 256, and 512, whereas for PREEMPT_RT only 64, 128, 256, and 512, because 16 and 32 are not supported by the ALSA drivers for HiFiBerry and buffer sizes higher than 512 samples result in a latency that is not interesting for our use case.
- **number of tracks**: multiple plugins can be instantiated in series (on the same audio track) and/or in parallel (on separate audio tracks). Because we are interested in the performance at varying core count, we used one track per processing core and we tested the systems varying the number of available tracks, with the same number of plugin instances per track. On PREEMPT_RT we tested from one track to three because one core was reserved for non–real-time tasks, whereas on Xenomai we could test up to four because of the dual-kernel approach.
- **maximum number of plugin instances per track**: A higher number of plugin instances results in a higher DSP load, so we mainly used the instance count to vary the DSP load conditions in the tests. Moreover, because we are interested in the maximum DSP load without incurring in Xruns, we are only interested in the maximum number of plugin instances per track that do not generate Xruns.
- **plugin type**: to test the system under different kinds of load we selected a plugin from the examples library of Faust [28] called "reverbDesigner" [29]. It exposes three parameters (N, NB, and BSO, more details can be found on the Faust GitHub page) that can be set at compile-time, directly affecting the DSP load of the resulting plugin. We compiled two versions of the plugin: a light version with N = 8, NB = 5, and BSO = 3 and a heavier version with N = 16, NB = 5, and BSO = 3. The entire test suite was performed using one version of the plugin and then repeated using the other.

The tests were conducted as follows: For each triplet of buffer size, number of tracks, and plugin type, we carried out multiple tests of 10 min to find the maximum number of plugin instances per track that does not generate Xruns. In real-world scenarios, the non–real-time part of the OS is typically involved in other activities such as networking and UI management, so we conducted each test while the system was undergoing a stress load generated by a stress generator tool called `stress-ng`. We used the following command:

```
$ stress-ng --sched other --cpu 1 --udp 1 --io 1 --netdev
  1 --aggressive --maximize
```

where `--sched other` runs every stressor as a non-real-time process, `--cpu`, `--io`, `--netdev`, and `--udp` instantiate one instance per stressor of, respectively, CPU, I/O, the network device, and UDP traffic. Finally, we validated the results obtained from the 10-min tests by rerunning all the tests for 2 h each, resulting in a total of 144 h of tests.

3.1.2 Scheduling Latency Test

The tool used to measure interrupt latency is `cyclictest`. It measures the amount of time that passes between a timer-driven interrupt and its handling by a userspace task. It does that by i) taking a time snapshot when starting to wait for a specific time interval (t1); ii) taking another time snapshot when the timer finishes (t2); iii) comparing the theoretical wakeup time with the actual wakeup time (t2 -(t1 + sleep_time)); this value is the latency for that timer wakeup.

`cyclictest` was chosen also because Xenomai provides it, along with its user-space libraries, already compiled to run in the cobalt kernel, so it was the obvious choice for comparing the two systems. Two identical tests lasting over 5 h each have been conducted on the two systems with the following command:

```
$ cyclictest -l100000000 -m -S -p 90 -i200 -h400
```

where `-l100000000` sets the total number of test iterations to 10^8, `-m` locks current and future memory allocations, `-S` enables SMP (Symmetric MultiProcessing) testing, `-p 90` sets the process priority to 90, `-i200` sets the time interval between interrupts to be 200 μs, and `-h400` dumps a latency histogram to stdout, tracking up to 400 μs.

Again, during this test the system was undergoing the load generated by the following command:

```
$ stress-ng --sched other --cpu 1 --udp 1 --io 1 --netdev
  1 --aggressive --maximize
```

3.1.3 RTL

To measure RTL, we used the `jack_iodelay` tool included in Jack2. This is a small command-line Jack app by Fons Adriaensen that measures the RTL of sound cards. The app consists of a DSP algorithm that does not measure instantaneous delay but rather leverages the Chinese Remainder Theorem on multiple sine waves at specific frequencies to measure the phase delay over a long period of time [30]. This technique allows for very high accuracy that is estimated by the author to be roughly 1/1000 of a sample. We performed the measurements for both the PREEMPT_RT and Xenomai systems using an external computer. The RTL of the external computer is measured first, then subtracted from the final results. This way the latency of the external system cancels out so it is not relevant for the final results.

3.2 RESULTS

This section presents the results obtained from the tests on DSP load, interrupt latency, and RTL, respectively. As far as the performance comparison between the two systems is concerned, we found these results to adhere to what we expected from previous studies [9] in terms of task-scheduling latency. In fact, Xenomai is better in every single test. However, we found more interesting differences between the two systems in terms of overall DSP performance when combined with a real-time audio driver.

3.2.1 DSP Load

The results of these tests are presented throughout the graphs in Fig. 2. They are organized in pairs, one per buffer size setting, for a total of six pairs. On each pair, the leftmost graph shows the maximum number of instances per track on the y-axis and the number of parallel tracks on the x-axis. Apart from the first two pairs (16 and 32 samples), each graph shows four series of data, respectively:

- **Xenomai Light**: these are the results of the tests performed on the Xenomai system using, instances of the less DSP intensive version of the reverbDesigner plugin.
- **PREEMPT_RT Light**: these are the results of the tests performed on the PREEMPT_RT system, using instances of the less DSP intensive version of the reverbDesigner plugin.
- **Xenomai Heavy**: these are the results of the tests performed on the Xenomai system using, instances of the more DSP intensive version of the reverbDesigner plugin.
- **PREEMPT_RT heavy**: these are the results of the tests performed on the PREEMPT_RT system, using instances of the more DSP intensive version of the reverbDesigner plugin.

At a first glance, the absence of PREEMPT_RT data in the first two pairs of graphs is evident. This already shows a significant advantage of the Xenomai system, as it can operate at buffer sizes as low as 16 samples, whereas the PREEMPT_RT system only supports buffer sizes of 64 samples and above. Moreover, looking at Fig. 2a, we can notice that the Xenomai system can run as many as five "heavy" and 11 "light" plugin instances in single-core operation, whereas it can run as many as 16 "heavy" and 32 "light" plugin instances when evenly spread across the four cores. As it is possible to notice from Fig. 2b, even with the very tight constraints of working with a buffer size of 16 samples, Xenomai manages to reliably sustain a DSP load of more than 85%.

To analyze results from PREEMPT_RT we have to look at the third pair of graphs, showing the test results in which 64 samples were used as the buffer size. Fig. 2e shows that in this setting PREEMPT_RT presents similar results to those that Xenomai provides when working at 16 samples of buffer size. In fact, Fig. 2f shows that the DSP load that PREEMPT_RT is capable of reliably carrying out is much lower, sitting below 75%. Additionally, it is possible to notice that PREEMPT_RT cannot run four parallel tracks because of its isolated non–real-time core that should not sustain any real-time load. Because of this, the maximum number of plugin instances across all tracks that the PREEMPT_RT system can sustain is smaller (27 for "light" and 12 for "heavy" plugin instances) because it has only three available real-time cores instead of four.

Increasing the number of parallel tracks leads to higher average DSP loads per instance, due to the cost of activation of additional threads at every interrupt callback and to the cache trashing resulting from the L2 cache being shared among all four cores. This can be seen in any rightmost graph of each pair, looking at the upward trend of the average DSP load per instance at the growing number of parallel tracks.

Increasing the buffer size leads to more plugin instances per track being sustained and higher average DSP loads being reliably tolerated by both systems, whereas the average DSP load per instance decreases as the time between audio interrupts widens. Increasing the buffer size also reduces the performance gap between the two systems, leading to the last pair of graphs where we can see that PREEMPT_RT almost matches the performance of Xenomai.

Also, Fig. 2l shows that when the two systems run the same number of plugin instances, the average DSP load of PREEMPT_RT is slightly higher than that of Xenomai. This is likely due to the much smaller average scheduling latency and the guarantee of never preempting real-time tasks provided by Xenomai leading to fewer context switches resulting in less overhead. Both SUSHI and the selected plugin are written in a real-time safe without any mode switch and we further confirmed using Xenomai's process monitor services during the tests. Mode switches happen in dual-kernel systems such as Xenomai whenever a task running in the real-time core performs a call that requires the system to switch to the non-real-time core. This results in a significant overhead and performance penalty.

As expected, Xenomai is better in different aspects. Nevertheless, we believe that the reported results are also quite usable in many scenarios for the PREEMPT_RT system because it is still able to handle reasonably high DSP loads

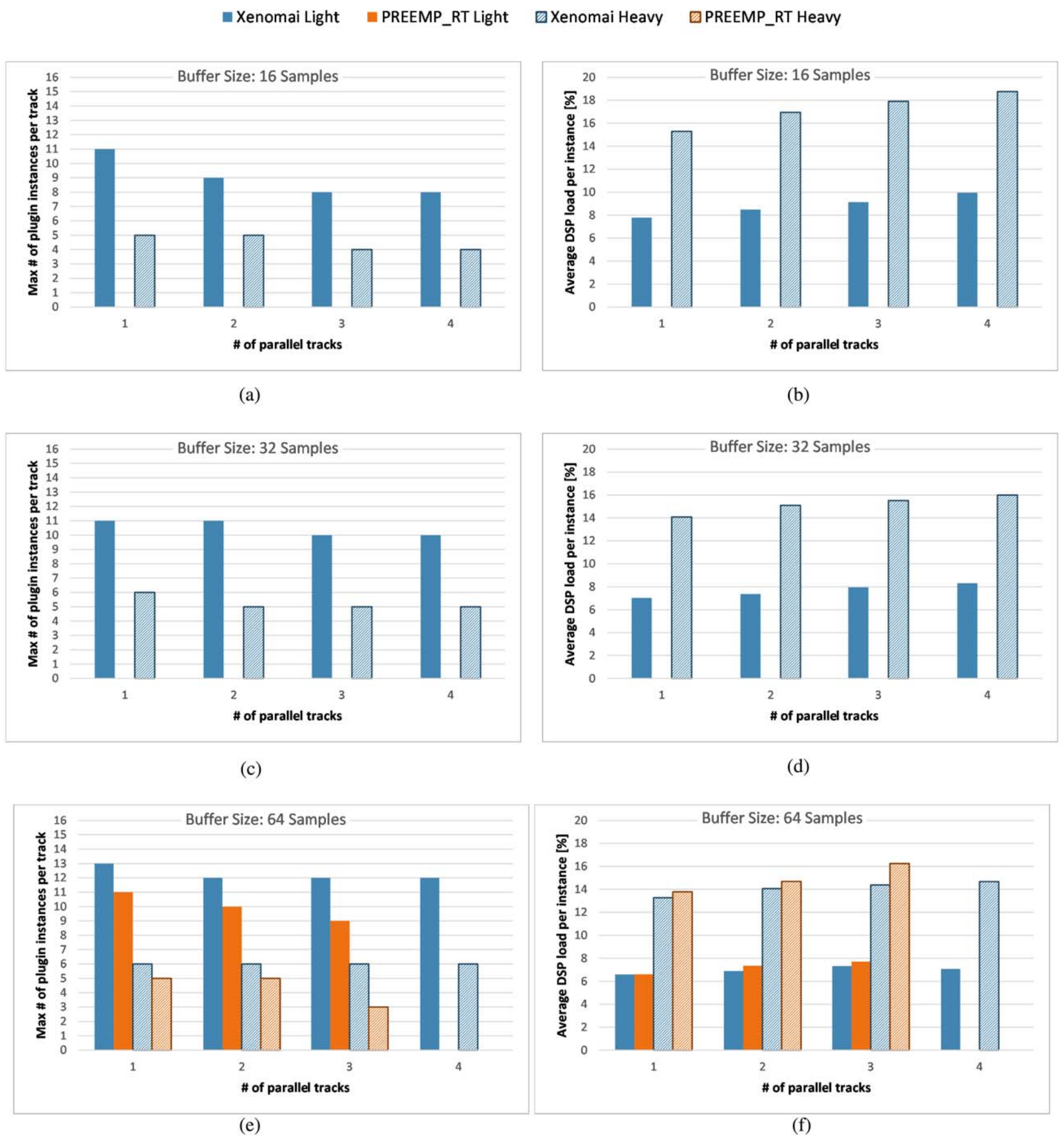


Fig. 2. DSP load results. (a) Xenomai manages to run up to 11 plugin instances in a single track and up to 32 over four parallel tracks. (b) The average DSP load per plugin instance grows with the number of parallel tracks due to the overhead of managing multiple threads and cache trashing. (c) With higher buffer sizes more plugin instances are sustained by the system. (d) Increasing the buffer size extends the time between audio interrupts, reducing the overhead associated with that. As a result, the average DSP load per plugin instance is lower. (e) At 64 samples, PREEMPT_RT is very affected by the overhead associated with the parallel tracks. In fact, we can see that the maximum number of plugin instances per track significantly reduces from 1 to 3 tracks. (f) As we can see, the PREEMPT_RT average DSP load per plugin instance is always slightly above that of Xenomai. This is because of its smaller scheduling latency and absolute prioritization of real-time tasks (g) As we can see, the behavior of the systems when running the “light” and the “heavy” versions of the plugin is very consistent. (h) The gap between Xenomai and PREEMPT_RT narrows down as the buffer size increases because the scheduling latency and other advantages of Xenomai become less and less relevant in a less interrupt-intensive scenario. (i) The PREEMPT_RT system is still heavily affected by the overhead of multiple parallel tracks, especially when running the “heavy” version of the plugin. (j) The difference in average DSP load per plugin instance between the PREEMPT_RT and Xenomai is almost negligible but is still appreciable. (k) PREEMPT_RT has almost closed the gap between itself and Xenomai, but still suffers some penalty when running three parallel tracks. (l) The two systems have almost identical behavior in terms of average DSP load per plugin instance. DSP stands for Digital Signal Processing.

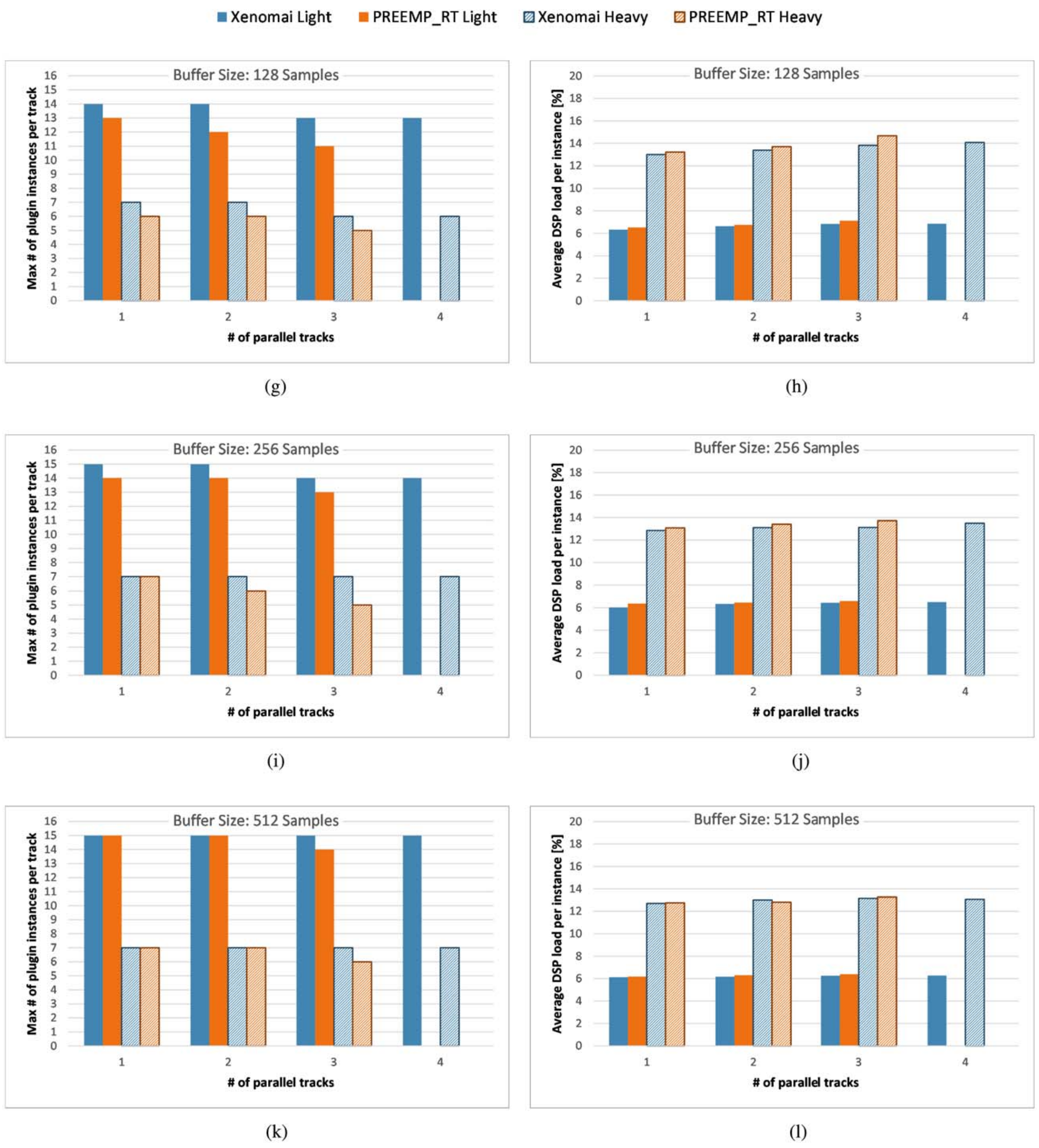


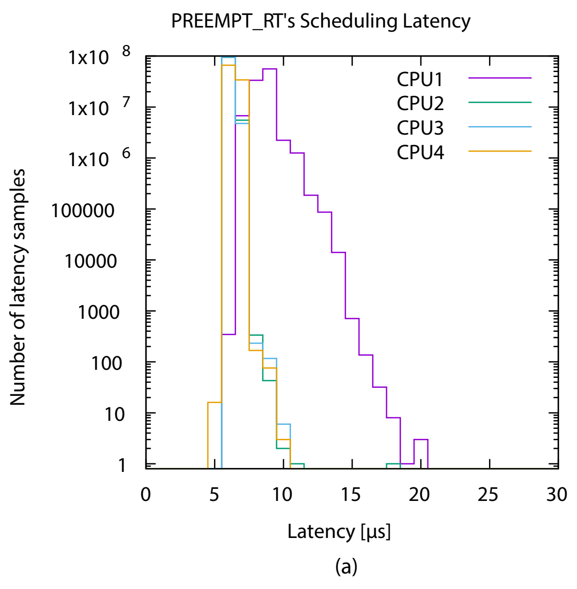
Fig. 2. (continued)

with the same level of reliability that Xenomai provides (0 Xruns).

3.2.2 Interrupt Latency

The results of the test described in Sec. 3.1.2 are shown in Fig. 3a and Fig. 3b. They show the overall superiority of Xenomai when it comes to scheduling latency, as well as the effectiveness of the core isolation on PREEMPT_RT. Fig. 3a shows the probability distribution of the interrupt latency of the PREEMPT_RT system. As it is possible to notice, there is a considerable difference between the behavior of the first core and the other three. This is because CPU1 is bearing all the load of the system, whereas the others have been isolated; hence, they are only dealing with the latency test and its interrupt.

Fig 3b shows the clear superiority of Xenomai both in the worst-case and the average scheduling latency. In fact, Xenomai manages to guarantee a worst-case latency of 12 µs, whereas PREEMPT_RT presents a worst-case latency of 20 µs, and the average scheduling latency for Xenomai



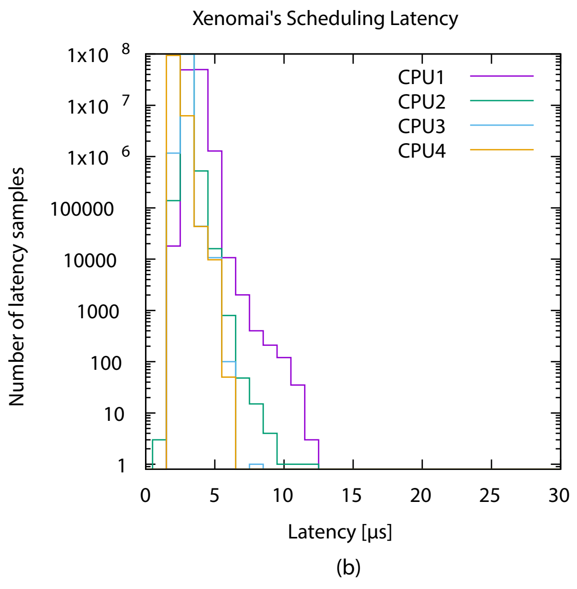


Fig. 3. Results of the scheduling latency tests. Scheduling latency on the Xenomai system. The worst-case latency is $12\mu s$ and the average is $2:5\mu s$.

and PREEMPT_RT, respectively, are 2.5 µs and 6.5 µs. The nonisolated CPU1 presents a slightly worse behavior due to the heavy load it is experiencing, but the dual-kernel nature of Xenomai alleviates the difference between the isolated and nonisolated cores. Overall, both systems present solid performance in this test, and the difference among them is smaller than expected.

3.2.3 RTL

The RTL is the latency resulting from the sum of all the delays of the system. In this case:

$$\text{RTL} = 2 * \text{buffer size}/f_s + \text{I}^2\text{S buffer latency} + \\ + \text{codec latency} \tag{1}$$

Table 1. RTL result comparison between the two systems.

(a) RTL, measured in milliseconds.

Buffer Size	PREEMPT_RT	Xenomai
16	-	1.2
32	-	1.86
64	3.34	3.2
128	6	5.86
256	11.34	11.2
512	22	21.86

(b) RTL, measured in samples.

Buffer Size	PREEMPT_RT	Xenomai
16	-	57
32	-	89
64	160	153
128	288	281
256	544	537
512	1,056	1,049

The measures of the RTL on both systems are detailed in Table 1. As it is possible to notice, the RTLs of the two systems are very close to each other, as expected because they are running with the same buffer sizes on the same hardware. The small difference between them is due to how each system configures the RX and TX FIFO buffers of the I^2S peripheral. The PREEMPT_RT system uses the stock I^2S driver, which needs those buffers to be higher to compensate for the higher interrupt latency of non–real-time systems (i.e., it needs to work on the standard Linux kernel as well). Elk Audio OS, on the other hand, can rely on the bounded interrupt latency that Xenomai guarantees and use smaller buffers on the I^2S peripheral.

The I^2S buffer latency on the PREEMPT_RT system is 20 samples, whereas on the Xenomai system, it is 13 samples, resulting in the 7-sample difference we can see in 2. Because the HiFiBerry Linux driver allows to set the latency of the filters on the DAC/ADC, we were able to match the configuration on both systems, resulting in a latency of two samples for the ADC and 10 samples for the DAC.

The total latency introduced by the I^2S buffer latency and the codec latency is as follows for the Xenomai system:

$$\frac{13}{f_s} + \frac{2+10}{f_s} = 0.52 \text{ ms} \tag{2}$$

and for the PREEMPT_RT system:

$$\frac{20}{f_s} + \frac{2+10}{f_s} = 0.66 \text{ ms} \tag{3}$$

with $f_s = 48$ kHz. Looking at Table 1, we can see that the Xenomai system can achieve latencies as low as 1.2 ms, whereas the PREEMPT_RT system can only go down to 3.34 ms.

4 CONCLUSION

In this paper, we compared the full stack of Xenomai, Raspa, and Xenomai's synchronization primitives against PREEMPT_RT, ALSA/Jack, and the POSIX synchronization primitive, aiming to quantify the difference among them in terms of DSP load, scheduling latency, and RTL Our results are in agreement with the ones from the study reported in [9], confirming the performance-wise superiority of Xenomai over the PREEMPT_RT patch. Xenomai, in fact, can provide lower RTL (1.2 ms), lower scheduling latency (worst-case 12 μs), and sustain very high DSP loads at small buffer sizes (over 85% average DSP load at 16 samples).

The ability of the Xenomai system to deliver such uncompromising performance is the reason why it is the backbone of high-performance systems such as Bela [6, 7], CTAG face 2|4 [8], or Elk Audio OS [27]. On the other end, PREEMPT_RT is very close behind still managing to provide very low RTL (3.34 ms) and very low scheduling latency (20 μs) and sustains still relatively high DSP loads at small buffer sizes (over 70% average DSP load at 64 samples). Although not suggested for safety-critical industrial applications [16], the performance it can deliver and its ease of use make it a good candidate for any application that does not really need the highest performance and lowest latency out of a system.

To conclude, both systems are viable building blocks of IoMusT-enabled devices because they can manage real-time audio as well as sensors and networking, but they cover different areas of the performance/complexity space.

5 ACKNOWLEDGMENT

The authors wish to thank the technical team of Elk for having provided technical support for this study.

6 REFERENCES

[1] E. Miranda and M. Wanderley, *New Digital Musical Instruments: Control and Interaction beyond the Keyboard*, Computer Music and Digital Audio Series, vol. 21 (A-R Editions, Inc., Middleton, Wisconsin, 2006).

[2] E. Meneses, J. Wang, S. Freire, and M. M. Wanderley, “A Comparison of Open-Source Linux Frameworks for an Augmented Musical Instrument Implementation,” in *Proceedings of the Conference on New Interfaces for Musical Expression*, pp. 222–227 (Porto Alegre, Brazil) (2019 Jun.). https://doi.org/10.5281/zenodo.3672934.

[3] E. Berdahl and W. Ju, “Satellite CCRMA: A Musical Interaction and Sound Synthesis Platform,” in A. Jensenius, M. Lyons (Eds.), *A NIME Reader*, Current Research in Systematic Musicology, vol. 3, pp. 373–389 (Springer, Cham, Switzerland, 2011). https://doi.org/10.1007/978-3-319-47214-0_24.

[4] E. Berdahl, S. Salazar, and M. Borins, “Embedded Networking and Hardware-Accelerated Graphics with Satellite CCRMA,” in *Proceedings of the Conference on New Interfaces for Musical Expression*, pp. 325–330 (Daejeon, Republic of Korea) (2013 Jun.). https://doi.org/10.5281/zenodo.1178476.

[5] I. Franco and M. Wanderley, “Prynth: A Framework for Self-Contained Digital Music Instruments,” in M. Aramaki, R. Kronland-Martinet, S. Ystad (Eds.), *Bridging People and Sound*, pp. 357–370 (Springer, Cham, Switzerland, 2016). http://dx.doi.org/10.1007/978-3-319-67738-5_22.

[6] A. McPherson and V. Zappi, “An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black,” presented at the *Audio Engineering Society Convention 138* (2015 May), conference paper 9331.

[7] A. McPherson, R. Jack, and G. Moro, “Action-Sound Latency: Are Our Tools Fast Enough?” in *Proceedings of the Conference on New Interfaces for Musical Expression*, pp. 20–25 (2016 Jul.). https://doi.org/10.5281/zenodo.3964611.

[8] H. Langer and R. Manzke, “Embedded Multichannel Linux Audiosystem for Musical Applications,” *J. Audio Eng. Soc.*, vol. 66, no. 4, pp. 286–291 (2018 Apr.). https://doi.org/10.17743/jaes.2018.0022.

[9] J. Brown and B. Martin, “How Fast is Fast Enough? Choosing between Xenomai and Linux for Real-Time Applications,” in *Proceedings of the 12th Real-Time Linux Workshop* (Nairobi, Kenya) (2010 Oct.).

[10] L. Turchet and C. Fischione, “Elk Audio OS: An Open Source Operating System for the Internet of Musical Things,” *ACM Trans. Internet Things*, vol. 2, no. 2, pp. 1–18 (2021 May). https://doi.org/10.1145/3446393.

[11] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, “Internet of Musical Things: Vision and Challenges,” *IEEE Access*, vol. 6, pp. 61994–62017 (2018 Sep.). https://doi.org/10.1109/ACCESS.2018.2872625.

[12] L. Turchet, “Smart Musical Instruments: vision, design principles, and future directions,” *IEEE Access*, vol. 7, pp. 8944–8963 (2018 Oct.). https://doi.org/10.1109/ACCESS.2018.2876891.

[13] L. Turchet, M. Benincaso, and C. Fischione, “Examples of Use Cases with Smart Instruments,” in *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences*, pp. 1–5 (London, United Kingdom) (2017 Aug.). https://doi.org/10.1145/3123514.3123553.

[14] L. Turchet, J. Pauwels, C. Fischione, and G. Fazekas, “Cloud-Smart Musical Instrument Interactions: Querying a Large Music Collection with a Smart Guitar,” *ACM Trans. Internet Things*, vol. 1, no. 3, pp. 1–29 (2020 Jul.). https://doi.org/10.1145/3377881.

[15] L. Turchet, A. McPherson, M. Barthet and, “Real-Time Hit Classification in a Smart Cajón,” *Front. ICT*, vol. 5, no. 16 (2018 Jul.). https://doi.org/10.3389/fict.2018.00016.

[16] F. Reghenzani, G. Massari, and W. Fornaciari, “The Real-Time Linux Kernel: A Survey on PREEMPT_RT,” *ACM Comput Surv*, vol. 52, no. 1, pp. 1–36 (2019 Feb.). https://doi.org/10.1145/3297714.

[17] L. C. R. Gonçalves and A. C. de Melo, “Application Testing under Realtime Linux,” presented at the *Linux Symposium* (Ottowa, Ontario, Canada) (2008).

[18] P. Gerum, “Xenomai-Implementing a RTOS Emulation Framework on GNU/Linux,” (2004).

[19] K. Yaghmour, “Adaptive Domain Environment for Operating Systems,” (2001).

[20] P. Gerum, “Dovetail Interface,” https://evlproject.org/dovetail/ (accessed Aug. 11, 2021).

[21] Modul 9 Gmbh, “HiFiBerry,” https://www.hifiberry.com/ (accessed Aug. 11, 2021).

[22] Modern Ancient Instruments Networked AB, “SUSHI Github Repository,” https://github.com/elk-audio/sushi (accessed Aug. 11, 2021).

[23] gRPC Authors, “gRPC: A High Performance, Open Source Universal RPC Framework,” https://grpc.io/ (accessed Aug. 11, 2021).

[24] Modern Ancient Instruments Networked AB, “elk-audio/twine: Thread and Worker Interface for Elk Audio OS,” https://github.com/elk-audio/twine (accessed Aug. 11, 2021).

[25] Linux Foundation, “Advanced Linux Sound Architecture (ALSA) Project Homepage,” https://www.alsa-project.org/wiki/Main_Page (accessed Aug. 11, 2021).

[26] S. Letz, Y. Orlarey, and D. Fober, “Jack Audio Server for Multi-processor Machines,” in *Proceedings of the International Computer Music Conference* (Barcelona, Spain) (2005).

[27] Modern Ancient Instruments Networked AB, “Elk,” www.elk.audio (accessed Aug. 11, 2021).

[28] Y. Orlarey, D. Fober, and S. Letz, “FAUST: An Efficient Functional Approach to DSP Programming,” in G. Assayag, A. Gerzso (Eds.), *New Computational Paradigms for Computer Music*, pp. 65–96 (Editions Delatour, Paris, France, 2009).

[29] Grame-CNCM, “Faust: Functional Programming Language for Real Time Signal Processing,” https://faust.grame.fr/ (accessed Aug. 11, 2021).

[30] Mikhail, “Automatic Estimation of Signal Round Trip Delay, Part 2b,” https://melp242.blogspot.com/2019/01/automatic-estimation-of-signal-round.html (accessed Aug. 11, 2021).

NOMENCLATURE

ADC =	Analog-to-Digital Converter
ALSA =	Advanced Linux Sound Architecture
DAC =	Digital to Analog Converter
DMA =	Direct Memory Access
DSP =	Digital Signal Processor
GPIO =	General Purpose Input/Output
gRPC =	Google Remote Procedure Call
I2C =	Inter-Integrated Circuit
MIDI =	Musical Instruments Digital Interface
OSC =	Open Sound Control
RASPA =	RTDM Audio-over-SPi API
RTDM =	Real-Time Driver Model
RTL =	Round-Trip Latency
SoC =	System-on-Chip
TWINE =	Thread and Worker INterface for Elk Audio OS
UART=	Universal Asynchronous Receiver/Transmitter
UDP =	User Datagram Protocol

THE AUTHORS

Luca Vignati

Stefano Zambon

Luca Turchet

Luca Vignati is a Ph.D. candidate at the Department of Information Engineering and Computer Science of University of Trento. He studied Computer and Electronic Engineering (B.Sc.) at the University of Pavia, Italy, and Computer Science and Engineering (M.Sc.) at Politecnico di Milano, Italy. He carried out his master thesis at Elk porting Elk Audio OS to a previously unsupported SoC. He has been awarded a post thesis scholarship from the University of Trento and a research scholarship from Fondazione C.M. Lerici.

•

Stefano Zambon is a cofounder and Chief Technical Officer of Elk. He holds a master's degree (summa cum laude) and a Ph.D. in Computer Science from Verona University. His experience combines academic-level expertise in signal processing with 11 years of industry experience working with large codebases, always for audio and musical applications.

•

Luca Turchet is an Assistant Professor at the Department of Information Engineering and Computer Science of University of Trento. He received master degrees (summa cum laude) in Computer Science from University of Verona, in classical guitar and composition from Music Conservatory of Verona, and in electronic music from the Royal College of Music of Stockholm. He received a Ph.D. in Media Technology from Aalborg University Copenhagen. His scientific, artistic, and entrepreneurial research has been supported by numerous grants from different funding agencies including the European Commission, the European Institute of Innovation and Technology, the European Space Agency, the Italian Minister of Foreign Affairs, and the Danish Research Council. He is cofounder of Elk, and is Associate Editor of IEEE Access and the Journal of the Audio Engineering Society. His main research interests are in music technology, Internet of Things, human–computer interaction, and multimodal perception.