Elk Audio OS: An Open Source Operating System for the Internet of Musical Things

LUCA TURCHET, Department of Information Engineering and Computer Science, University of Trento, Italy
CARLO FISCHIONE, Department of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden

As the Internet of Musical Things (IoMusT) emerges, audio-specific operating systems (OSs) are required on embedded hardware to ease development and portability of IoMusT applications. Despite the increasing importance of IoMusT applications, in this article, we show that there is no OS able to fulfill the diverse requirements of IoMusT systems. To address such a gap, we propose the Elk Audio OS as a novel and open source OS in this space. It is a Linux-based OS optimized for ultra-low-latency and high-performance audio and sensor processing on embedded hardware, as well as for handling wireless connectivity to local and remote networks. Elk Audio OS uses the Xenomai real-time kernel extension, which makes it suitable for the most demanding of low-latency audio tasks. We provide the first comprehensive overview of Elk Audio OS, describing its architecture and the key components of interest to potential developers and users. We explain operational aspects like the configuration of the architecture and the control mechanisms of the internal sound engine, as well as the tools that enable an easier and faster development of connected musical devices. Finally, we discuss the implications of Elk Audio OS, including the development of an open source community around it.

CCS Concepts: • **Computer systems organization** → **Real-time operating systems**; *Embedded systems*; • **Hardware** → Sound-based input / output; • **Networks** → Network services;

Additional Key Words and Phrases: Internet of Musical Things, embedded systems, smart musical instruments

ACM Reference format:
Luca Turchet and Carlo Fischione. 2021. Elk Audio OS: An Open Source Operating System for the Internet of Musical Things. *ACM Trans. Internet Things* 2, 2, Article 12 (March 2021), 18 pages.
https://doi.org/10.1145/3446393

1 INTRODUCTION

The Internet of Musical Things (IoMusT) is an emerging research field positioned at the intersection of the Internet of Things, music technology, human-computer interaction, and artificial intelligence [52]. The IoMusT relates to ecosystems of computing devices embedded in physical

L. Turchet and C. Fischione are also affiliated with Elk.
The authors acknowledge support from the European Institute of Innovation & Technology (Grant No. 749561) and the European Space Agency (Grant No. 4000132621/20/NL/AF).
Authors' addresses: L. Turchet (corresponding author), Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, Trento, IT, 38123, Italy; email: luca.turchet@unitn.it; C. Fischione, Department of Network and Systems Engineering, KTH Royal Institute of Technology, Malvinas Vg 6B, Stockholm, SE, 10044, Sweden; email: carlofi@kth.se.

2577-6207/2021/03-ART12
https://doi.org/10.1145/3446393

objects (Musical Things) dedicated to the production and/or reception of musical content. The IoMusT vision predicts that in the future, a new class of Musical Things will be connected to the Internet, which has the potential for a transformative effect on how stakeholders involved in musical activities (such as performers and audience members) conduct these activities and interact with musical objects (such as musical instruments or musical interfaces for the audience).

Various kinds of Musical Things are appearing as the IoMusT community grows, such as wireless gestural interfaces [21] or the so-called "musical haptic wearables" for performers [48] and audience members [55]. Nevertheless, the most prominent example of Musical Thing is represented by the class of "smart musical instruments" [46]. Such instruments encompass sensors, actuators, wireless connectivity to local and remote networks, as well as on-board processing. Smart musical instruments are capable of directly exchanging musically-relevant information with one another as well as communicate with a diverse network of external devices (such as smartphones, wearables, virtual reality headsets, or stage equipment). Instances of such kind of Musical Things are the Sensus Smart Guitar developed by Elk [50], other prototypes of smart guitars developed in academic contexts (see, e.g., References [49, 54]), or the Smart Cajón reported in Reference [53].

One of the obstacles to the rising of the IoMusT is the lack of appropriate and efficient development tools that allow music and audio hardware manufacturers to create connected devices. 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. Nowadays, the vast majority of professional audio devices targeting the hard requirements of real-time performance are produced using ad hoc real-time operating systems (OSs) or dedicated digital signal processors. These are complex to program, offer very limited support to interface to other hardware peripherals, and lack modern software libraries for networking and access to cloud services [10].

As the IoMusT emerges [18, 22, 28, 30], OSs specific to the musical domain are required on embedded hardware to ease development and portability of IoMusT applications. Like the IoT [5], the envisioned IoMusT is characterized by heterogeneous devices, which range from lightweight low-end devices to powerful 64-bit processors. However, traditional digital signal processors or real-time OSs currently running on musical devices, as well as typical OSs for sensor networks, are not capable of fulfilling all at once the diverse requirements of such a wide range of devices. To leverage the IoMusT, redundant development should be avoided and maintenance costs of IoMusT products should be reduced. Hence, a unifying, developer-friendly type of OS specific to IoMusT applications is desirable. The need for such a common OS for heterogenous Musical Things perfectly parallels the efforts within the IoT community that led to the creation and diffusion of the RIOT OS [1]. Nevertheless, OSs conceived for the IoT, such as RIOT, are not adequate to IoMusT scenarios, since the requirements of the IoMusT are different from those of the general field of IoT [52].

In this article, we revisit the requirements for an OS for the IoMusT, analyzing the generic requirements for software running on IoMusT devices. Second, we introduce Elk Audio OS developed by Elk,[1] an OSs that explicitly targets the IoMusT domain and eases development across a wide range of embedded hardware. We provide a comprehensive overview of Elk Audio OS, both from the OS point of view and from the development standpoint. We cover the key components of interest to potential developers and users, including the kernel, hardware abstraction, and software modularity. We explain operational aspects such as the configuration of the architecture and the control mechanisms of the internal sound engine, as well as the tools that enable an easier and faster development of connected musical devices. Finally, we discuss the implication of this novel technology, including the development of an open source community around it.

[1] www.elk.audio.

2 RELATED WORKS

This section surveys current technologies for building embedded and digital audio devices, as well as the tools available today for writing software for music and audio processing.

2.1 Embedded Systems for Musical Applications

Writing the software stack for an embedded hardware device is notoriously a difficult task, especially if the device has to implement non-trivial functionalities and connectivity [16]. In the past decade, the availability of development toolkits based on the Linux Kernel has decreased significantly the development cost and the time to market for realizing connected devices. The most notable example is Google's Android, which is running on billions of mobile devices around the world and has enabled many manufacturers to engineer products at a fraction of the price of previous solutions, opening at the same time the opportunity to third-party application developers to easily write software that is portable across all the devices supported by Android.

In the subfield of digital audio devices, however, the situation is radically different. The main reason is that the manipulation of audio signals with a processor is a task that requires very low processing latency (i.e., the time difference from when the input signal is acquired by a device and the output signal is produced by the same device), as well as very low jitter (i.e., the variation of latency). For musical applications the total latency must be kept constantly 10 ms with jitter below 1 ms, according to generally accepted guidelines indicated by designers of digital musical instruments [12, 14, 19, 32, 59]. This is due to the fact that asynchronies between tactile and auditory feedback (i.e., action-sound latency) when playing a musical instrument are disruptive to the musical performance. This hard requirement on real-time performance is the reason why today most audio devices are still built using dedicated digital signal processors and real-time OSs, which are complex to program and offer very limited support to interface to other hardware peripherals. Compared to Linux-based solutions, they also lack the availability of modern software libraries for networking and access to cloud services, which makes very difficult the realization of connected products.

Nevertheless, in the past few years various Linux-based platforms for creating self-contained musical instruments [2] have been developed, which mainly target the makers community [35] (a recent comparative study is reported in Reference [34]). 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. Other similar platforms targeting the makers community are Axoloti[2] and Prynth [13]. The state-of-the-art in this space (see Reference [34]) is represented by Bela [32, 33], a platform based on the BeagleBone Black single-board computer, which is extended with a custom expansion board featuring stereo audio, 8 channels each of 16-bit ADC and 16-bit DAC for sensors and actuators, a I2C port and 16 GPIO channels. To achieve latencies below 1 ms Bela uses the Xenomai real-time kernel extension, which according to the study reported in Reference [6] is the best-performing of the hard real-time Linux environments. Nevertheless, whereas the action-sound latency requirements are satisfied, the Bela real-time environment does not run on a wide spectrum of hardware, but only on the BeagleBone Black. Moreover, Bela is not conceived by design to support several connectivity options efficiently with concurrent audio and sensors processing. Furthermore, being a board that targets the makers community, it does not offer professional audio quality.

However, the performance levels offered by Bela are not met by OSs designed for low-end devices for the IoT, such as TinyOS [25], Contiki [9], or RIOT [1]. Such OSs do not target the stringent requirements on real-time performances and high audio quality, since they are conceived for

[2] http://www.axoloti.com/.

constrained devices that are dedicated to very different tasks (e.g., sensing the air temperature at interval of 5 s and transmitting it at intervals of 30 s), with minimal resources in terms of computation and memory, and where power consumption efficiency needs to be guaranteed due to a very tight energy budget. Conversely, devices for the IoMusT must necessarily have sufficient computational power to enable low-latency audio processing as well as wireless connectivity to local and remote networks, and this is particularly true for professional audio equipment. In addition, these constrained IoT devices generally do not interface with low-latency, highly reliable networks [20] such as those of the emerging Tactile Internet [26] that will form the upcoming generations of network infrastructures (e.g., 5G). Notably, OSs specific for the 5G are currently under discussion (see, e.g., References [27, 44]), but in this space little attention has been devoted by the research community to the musical domain. In a different vein, mobile phones and tablets have been widely used for musical applications, including networked music performances [15, 41]. However, they suffer from high audio processing latencies, and their architecture has not conceived specifically for musical applications.

2.2 Development Tools for Music and Audio Processing

Today, from the developers' perspective, the scenario for writing musical and audio applications on conventional computers (i.e., desktop and laptop PCs) is much better compared to the embedded systems. The wide availability of programming environments, Software Developer Kits (SDKs), specialized libraries, and frameworks has made possible the creation of a huge number of musical processing applications for such computers. However, conventional computers are generally considered non reliable for live usage by many musicians. The processing latency inside a general purpose OS such as Microsoft Windows or Apple's Mac OS X can hardly go below the 10 ± 1 ms guidelines for interactive musical applications [12, 14, 19, 32, 59]. Typically, conventional computers require additional hardware such as an audio interface and musical controllers.

Most musical and audio software can fall into three categories: digital audio workstations (DAWs) [24], audio plugins, and real-time audio programming environments. A DAW is the software equivalent of mixing consoles and multitrack recorders previously used in sound studios for music production. It is used for recording musicians on different tracks and produce the final piece of music. Audio plugins are the equivalent of audio effects, sound synthesizers and all other units that are used by musicians to modify or create sounds. There are several formats available for audio plugins. Some of them, like Apple's AudioUnits, Avid's RTAs, Propellerhead's Rack Extension (RE), are restricted to specific OSs or DAWs. Others instead offer different implementations on many platforms and different DAWs. By far, the de facto standard today is the Virtual Studio Technology (VST) developed by Steinberg Multimedia [45]. Several companies around the world focus on developing and selling audio plugins and thus there is a very large amount of code written using these standards for PCs. Therefore, porting all such code to embedded systems with minimal development effort and independently from the underlying hardware is a crucial step for the success of the IoMusT. However, limited efforts have been conducted thus far towards such direction within both industry and academy.

Digital signal processors offer development kits (such as Texas Instruments DSP dev kits[3] or Analog Devices dev kits[4]), which have a steep learning curve and do not provide an application framework, forcing developers to write significant amount of software in a limited development environment. In addition, those environments are not equipped with modern software libraries for networking and access to cloud services. However, application frameworks are available to code

[3] http://www.ti.com/processors/digital-signal-processors/overview.html.
[4] https://www.analog.com/en/products/processors-dsp.html.

Table 1. Comparison of Key Parameters of Existing Solutions for Creating Embedded Musical Devices, Namely, IoT OSs (such as TinyOS [25], Contiki [9], or RIOT [1]), Mobiles OSs (such as Android or iOS), DSP Processors (such as Texas Instruments' or Analog Devices' DSP dev Kits), and Open Source Projects for Makers (such as Bela [33], Axoloti, Prynth [13] or Satellite CCRMA [4])

	IoT OSs	Mobiles OSs	DSP processors	Platforms for makers	Elk-powered platforms
Low-latency audio processing	✗	✗	✓	✓	✓
High-performance audio processing	✗	✗	✓	✗	✓
Hardware independence	✓	✗	✗	✗	✓
Efficient audio development tools	✗	✓	✗	✓	✓
WiFi, Bluetooth, 4G/5G support	✓	✓	✗	✓	✓
VST/RE plugins support	✗	✗	✗	✗	✓
Linux-based	✓	✗	✗	✓	✓
Modern standard protocols-based	✓	✓	✗	✓	✓
Support for Machine Learning tools	✗	✓	✗	✗	✓

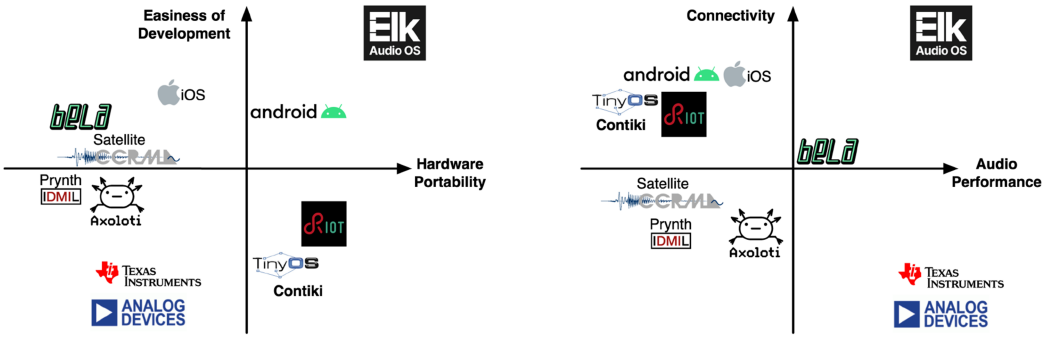

Fig. 1. Position of existing solutions for creating embedded musical applications along the orthogonal axes connectivity vs. audio performance (left) and easiness of development vs. hardware portability (right).

desktop audio plugins, one of the most popular today being JUCE.[5] However, JUCE has limited support for embedded systems and needs a system layer for embedded Linux to run. In a different vein, sound programming environments such as Pure Data [37], CSound [23], or Supercollider [31] are available on embedded systems such as Bela [32, 33] or Satellite CCRMA [3, 4]. Such tools, however, are typically not used for creating commercial products with professional audio quality, and are conceived for musicians rather than developers.

2.3 Comparison of Current Solutions

Table 1 compares the technologies surveyed in Sections 2.1 and 2.2 according to a set of criteria that are relevant to the development of Musical Things. The technologies have been grouped according to four categories, namely, IoT OSs (such as TinyOS [25], Contiki [9], or RIOT [1]), Mobiles OSs (such as Android or iOS), DSP processors (such as Texas Instruments' or Analog Devices' DSP dev kits), and open source projects for makers (such as Bela [33], Axoloti, Prynth [13] or Satellite CCRMA [4]). Along the same lines, Figure 1 illustrates a comparative positioning of such

[5] https://juce.com/.

technologies along the orthogonal axes connectivity vs. audio performance (left) and easiness of development vs. hardware portability (right).

From the conducted analysis, it emerges that different solutions for creating embedded musical applications exist. This space is, however, characterized by a fragmentation that implies the parallel use of several OSs, forcing the development and the maintenance of redundant application code running across the network of such devices. Therefore, an OS for the IoMusT is currently missing, which is capable of running on a variety of hardware with different resources as well as providing developer-friendly tools.

3 OPERATING SYSTEM REQUIREMENTS FOR IOMUST DEVICES

In this section, we propose a set of requirements for an OS running on IoMusT devices. In part such requirements are inspired to those of the RIOT OS [1]. A first category of requirements pertains to the system performance whereas a second category deals with platform independence. A third type of requirements concerns with network interoperability aspects while a fourth type of requirements focuses on the usability of the system from a developer's perspective. Finally, a fifth category concerns fundamental aspects related to security and privacy.

Computational performance. The key performance requirement is reactivity: the system must necessarily be hard real-time [7], capable of guaranteeing ultra-low-latency <1 ms and low jitter <1 ms for both audio and sensors processing. The system should support mono, stereo, or multichannel inputs and outputs.

Platform independence. The OS must support a variety of hardware platforms, ranging from constrained platforms (such as lower cost ARM CPUs) to powerful platforms. Therefore, the system must provide hardware abstraction so that most of the code is reusable on all IoMusT hardware supported by the OS itself. Moreover, the system should support a wide range of audio codecs and guarantee high audio quality. In addition, the system should provide support for multichannel digital audio interfaces (such as, for instance, the standards AES67 or AVB for Audio over Ethernet). Furthermore, at application and software library level, the system should provide standard interfaces to utilize third-party software modules.

Connectivity interoperability. IoMusT devices are expected to be connected to both local and remote networks. An OS for the IoMusT should thus offer support for heterogeneous wireless transceivers, including low-power wireless technologies such as IEEE 802.15.4 or Bluetooth low-energy, the 802.11 WiFi family (including millimeter waves [43]), as well as proprietary digital wireless transceiver for low-latency audio communication to external audio hardware. The OS should be able to support the wireless delivery and reception of both audio signals and musical messages such as Open Sound Control (OSC) [61] and Musical Instrument Digital Interface (MIDI). Moreover, the network stack should provide full-fledged TCP/IP implementations as well as the standard low-power IP stack should be supported (including UDP, CoAP, 6LoWPAN, and IPv6 [42]). Importantly, the OS should be capable of handling communication systems with latencies on the order of milliseconds and with the probability of packet loss on the order of at least 10^{-2}, while preserving a high audio quality (these stringent requirements are needed to guarantee credible musical interactions [40, 41] and to avoid perceivable deteriorations of the signal [11]). Furthermore, the system should be robust to network evolutions over time, for instance providing support to the next generation of cellular networks.

Programmability. The OS should be as developer-friendly as possible, offer APIs to ease software development and simplify the porting of existing software, as well as provide development tools allowing for easy porting of the software between various hardware (which minimizes development and maintenance efforts). Moreover, it should support standard high level programming languages, such as python and C++. In addition, the system should offer real-time

communication with external GUIs and controllers, allowing developers to choose freely among the many solutions they are already familiar with. Furthermore, the system should natively support all the functionalities of the Digital Audio Workstation such as hosting audio plugin, sequencing of musical notes, and multitrack recording.

Security and privacy requirements. Like for the IoT, the deployment of the IoMusT is expected to penetrate both private lives and industrial processes [52], which entails high standards for security and privacy of systems and applications. As a consequence, an IoMusT OS must be carefully developed and continuously reviewed to reduce possibilities of attack.

Support for AI/ML-based applications. Applications and services based on Artificial Intelligence (AI) and in particular Machine Learning (ML) are at the core of many Musical Things envisioned within the IoMusT paradigm [52]. In particular, the notion of intelligence of smart musical instruments described in Reference [46] is strongly rooted in AI and ML methods to run hard real-time music information retrieval [47] and semantic audio [47] applications and services. Therefore, the OS should be capable of supporting services based on Machine Learning (and Artificial Intelligence at large) as well as tools for developing them.

3.1 Elk Audio OS Goals

From the comparison of the requirements described in Section 3 and the features supported by the OSs reviewed in Section 2.1, it possible to derive that none of the existing OSs (neither a lightweight OS targeting wireless sensor networks nor a full-fledged OS) is able to fulfill requirements so diverse. The goals that motivated the design of Elk Audio OS are based on those requirements and are listed below:

(1) Simultaneous capture of both multichannel audio and sensor data;
(2) Hard real-time processing capabilities, with ultra-low latency round trip (<1 ms) and low jitter (<1 ms);
(3) Platform independence, the same OS runs on a variety of hardware platforms and supports a diverse range of audio codecs;
(4) Natively equipped with efficient development tools allowing developers to port the same code on different hardware;
(5) Natively equipped with a DAW supporting audio plugins and all functionalities of conventional DAWs;
(6) Support for a diverse range of connectivity options to local and remote networks and for the communication of both musical messages and audio signals.

Notably, part of such goals parallel the ones that the Bela board is conceived for Reference [33], while other goals are common to those that RIOT is based on Reference [1]. Bearing such goals in mind, the next section describes Elk Audio OS in detail.

4 ELK AUDIO OS ARCHITECTURE

Figure 2 illustrates a diagram of the main System-on-Chip (SOC) that runs Elk Audio OS as well as Elk Audio OS's most relevant software components and the interconnections between them (a description of each block in the diagram is reported in the subsequent sections).

Elk Audio OS is a Linux-based embedded OS highly optimized for low-latency and high-performance audio processing as well as for handling wireless connectivity. It aims at bridging the gap we observed between OSs specific for wireless sensor networks, conventional full-fledged OSs for desktop PCs supporting DAWs, and digital signal processors for interactive musical applications on embedded hardware. Elk Audio OS runs on a wide spectrum of hardware platforms, guarantees low round-trip latencies, supports music software plugins, provides wireless

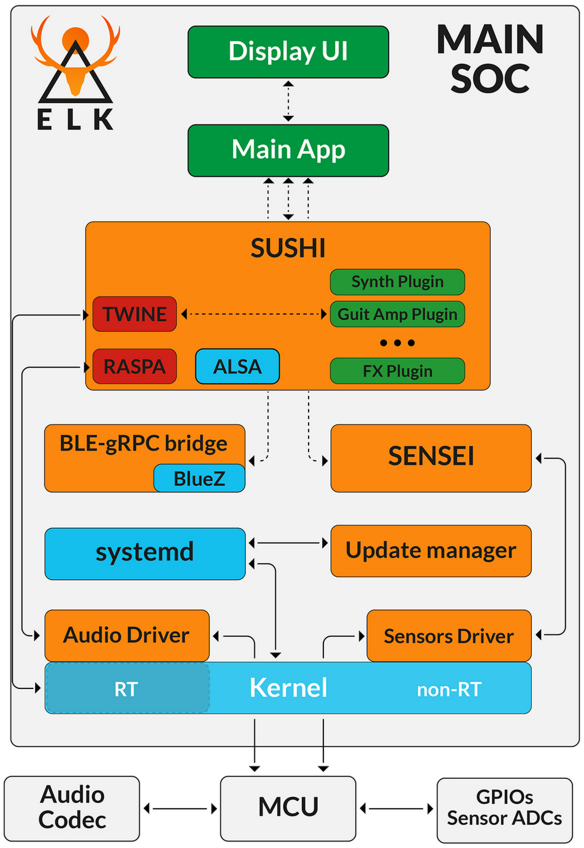

Fig. 2. A block diagram of the various software components of Elk Audio OS running on a System-on-Chip. Legend: MCU: the Microcontroller Unit; SENSEI: a daemon that handles several kinds of sensors and GPIOs; SUSHI: the core audio engine; RASPA: library for interfacing with the custom audio driver; TWINE: library handling multi-threading; BLE-gRPC bridge: software connecting Elk Audio OS blocks to each other and to external devices over WiFi or Ethernet; Update manager: module deploying software updates to the device.

connectivity via BLE, WiFi, 4G, and 5G, as well as offers efficient development tools. It is upgradable, future-proof, and is based on modern standard protocols. Elk Audio OS is controllable through an interface for a software that handles physical sensors and communicates with the outside world by means of a standard network protocol. In this way, multiple different graphical user interfaces can be implemented on the same device or on any external connected device (such as a mobile phone or tablet) to control the parameters of the interaction between the musician and the device.

Elk Audio OS is structured around a modular architecture where the software modules are aggregated at compile time. The only custom work required for each device is then just a matter of writing a small glue application that connects together Elk Audio OS's components for the particular use case. Such an approach allows one to build the complete OS in a modular manner, including only those components that are required by the use case.

4.1 Hardware Abstraction and Components

The management of hardware resources is a central aspect for an OS. In this regard, Elk Audio OS is portable to different hardware and provides a comprehensive hardware abstraction. Hardware targeted by Elk Audio OS include both ARM and Intel CPU architectures, as well as a number of

external components such as analog and digital sensors, actuators, MIDI controllers, and network interfaces.

Currently supported hardware are either one of the Elk's development boards or custom hardware designed for products powered by Elk Audio OS (such as the DV Mark Smart Multiamp). At present, Elk's development boards include Intel Joule 550x Compute Module, AAEON UpCore, Toradex Colibri iMX7, STM32MP157A-DK1, and Raspberry Pi 3B+ and 4. For each of these development boards, a custom expansion shield has been created to handle audio and sensors input/output. In the case of the recent Raspberry Pi 4, Elk supports the commercially available HiFi Berry shield. If the CPU and drivers are already supported, then porting Elk Audio OS to a new board is simply a matter of creating a few configuration files. Otherwise, the complexity of supporting a new CPU, or a new driver, depends on the actual hardware.

Hardware components of a powered by Elk device include the following.

Main SOC. This is the main integrated circuit on the board that contains the CPU and several other peripherals (e.g., USB and serial controllers, GPUs). Elk Audio OS boards use common SOCs that can run Linux and are also found in other embedded or mobile devices (e.g., the Intel Atom Cherry Trail X5-Z8350 or the NXP i.MX7).

Microcontroller Unit (MCU). This is custom for the various Elk Audio OS boards and can vary depending on requirements on I/O and price. Its purpose is to ease the communication with the Audio Codec(s) and to interface with General Purpose Input Output (GPIOs) pins or ADC used for sensors. For this purposes, on different boards we have used XMOS MCUs, Cortex-M4s or tiny CPLDs.

The OS is capable of abstracting also input/output peripherals related to sensors, audio, and the network, the handling of which is covered in Sections 4.3 and 5.

4.2 Standard Linux Components

Standard Linux components are open-source software pieces that are common to most embedded Linux devices. Elk Audio OS adopts the latest stable version of each one of them, usually without significant modifications to their code. Most of the work in bringing up these parts into Elk Audio OS is in their integration and configuration. For this purpose, tools from the Yocto Project are used, which are industry-standard tools to create custom embedded Linux distributions.

Kernel. This is a customized version of the Linux Kernel tuned for low-latency. In most of our boards, we make use of the Xenomai Cobalt Kernel, which uses a dedicated interrupt pipeline (I-Pipe) to achieve extremely low hard real-time performances. It is often described as a Dual Kernel architecture, because a small Real-Time kernel is running on the same memory space as the traditional Linux Kernel. However, the rest of Elk Audio OS does not depend strictly on Xenomai features, and we have successfully used the more common PREEMPT_RT patch [38] in other boards, using core isolation and other optimizations on top of it. In this second case, there is some latency and performance penalty, but the results are still good enough for some use cases.

systemd. As most modern Linux distros, Elk Audio OS adopts systemd for its init system and many other system components in userspace. The core Elk Audio OS processes have been designed to take advantage of systemd's parallelism to minimize device boot time.

Libraries / applications. We leveraged the power of many famous userspace libraries and applications for writing Elk Audio OS core services and for providing a good set of tools for developers writing product-specific components. Relevant examples are the ALSA project's userspace libraries (for MIDI) or the BlueZ stack (for Bluetooth connectivity).

4.3 Elk Audio OS Core Components

The components presented in this section include all the custom software written specifically for Elk Audio OS (they are indicated in orange/red in Figure 2). There are some variations between

Elk Audio OS boards for some parts (for example, each SOC family usually requires a dedicated driver for the specific peripherals used for audio IO), but the interface towards product-specific components will still be the same.

Audio and sensors drivers. The drivers for audio and sensors interface directly with hardware components (codecs, MCUs, etc.) included in the custom boards for Elk Audio OS. They are designed with the goal of achieving the lowest possible latency, so very often we are not using standard Linux infrastructures directly (like ALSA for the audio driver).

Update manager. This module takes care of deploying software updates to Elk Audio OS devices, making the process easy and robust for the final user. The system is guaranteed to always boot in a working condition even in presence of power failures during the update.

BLE-gRPC bridge. As explained in the next section, gRPC is our main choice for connecting Elk Audio OS blocks to each other and to external devices over WiFi or Ethernet. This component makes it possible to route the same protocols towards mobile devices connected over Bluetooth Low-Energy, which is not a standard TCP/IP connection.

SENSEI. This is a daemon that makes it very easy to handle several kinds of sensors and GPIOs (buttons, LEDs, etc.) on the physical device. The hardware connections can be declared flexibly using a JSON configuration file, without the need for custom software for each device. SENSEI then takes care of creating software events that can be accessed by the product-specific applications.

SUSHI. This is the core audio engine behind any device running Elk Audio OS. It is a multitrack and multichannel live plugin host, with advanced audio and MIDI routing capabilities, as well as Ableton Link[6] integration [17]. In other words, a small DAW is included in Elk Audio OS, which can be configured to run VST or RE plugins. As with all the other Elk Audio OS core components, SUSHI is a command line process that is fully configurable with its remote API (more on this on Section 5), so it does not ship with any GUI. SUSHI has been developed with the strict requirements of real-time programming in mind in all cases, especially when a Xenomai Cobalt Kernel is employed. The integration with the lower-level drivers and subsystems happens with dedicated libraries: **RASPA** for interfacing with the custom audio driver, and **TWINE** to offer a high-level API into common real-time functions such as managing multiple real-time threads for multicore processing and accurate timers (see Section 5.5). TWINE can also be used by third-party's plugins. Section 5 describes SUSHI in greater level of detail.

4.4 Gluing Components Together

To build a device using Elk Audio OS, one needs to develop software modules (indicated as the green-colored blocks in the diagram of Figure 2) able to glue together the different components listed above. Such software modules are specific to each device. Notably, writing these parts is very similar to writing an application for a normal desktop computer, thanks to all the abstractions provided by the underlying layers. Examples of these modules are the following.

Audio Plugins. As already stated, Elk Audio OS hosts normal VST and RE plugins, recompiled for the target architecture. It is usually trivial to get a VST plugin that already runs in Linux to run fine under Elk Audio OS, and we provide several tools to aid this process (see Section 7).

Display UI. This is a dedicated process to handle, e.g., a touchscreen display. Elk Audio OS does not provide a specific GUI framework, but developers are free to use any of the popular and well-maintained solutions for this purpose. For example, Qt is a popular choice in many embedded devices. All of Elk Audio OS's core processes (SUSHI, SENSEI, etc.) share these common features: (i) it is configurable via JSON files; (ii) it can be controlled at runtime using either Open

[6] https://www.ableton.com/en/link/.

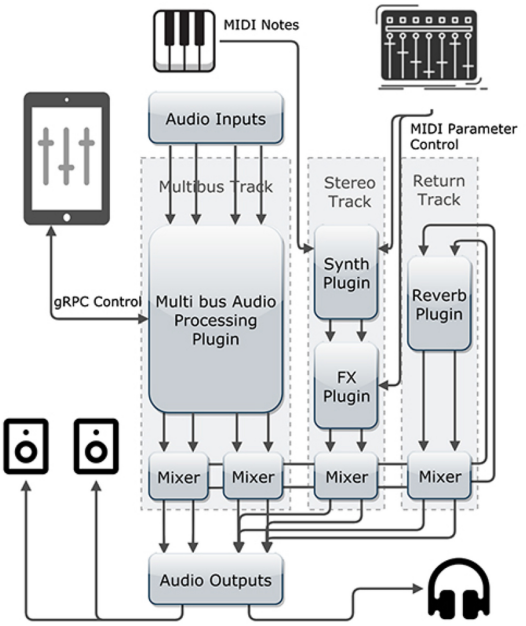

Fig. 3. The flowchart of SUSHI, the core audio engine of Elk Audio OS, which shares the same basic architecture with most common digital audio workstations.

Sound Control (especially for quick prototypes) or a Google's gRPC protocol; a logging system configurable and accessible from other applications.

Main application. The custom Main Application leverages the described modular architecture to connect all the components together for a specific device. For example, this will contain the logic for operations like "choose a different sound and change the display when the user press a button" or "use the LEDs on the front panel to display a VU Meter." Since gRPC is supported by a large number of languages, it is possible to write this component in Python, or Javascript, or Lua, and so on, in addition to the conventional choice of C/C++. An example workflow might be to quickly prototype the main application in Python and then port it to C++17 for production, without making any changes in the other parts.

5 SUSHI

SUSHI is a track-based, headless Digital Audio Workstation that is at the core of audio and MIDI processing in the Elk Audio OS. It works as a plugin host, supporting multiple plugin standards (such as VST 2.x and VST 3.x and Rack Extensions), features advanced audio and MIDI routing, simple scripting setup. It manages the recording and playback of audio and musical tracks with a sequencer. It is written to ensure high performance and stability under low latency conditions. It can be controlled through MIDI, OSC, or a gRPC interface.

5.1 Architecture

SUSHI shares the same basic architecture with most common DAWs, having an unlimited number of parallel channels (see Figure 3). Each track supports mono, stereo or up to 64 audio channels and as many plugins as the CPU can handle. Pure MIDI tracks are also supported, for instance outputting MIDI from an arpeggiator or step sequencer to an external device. There is also a multibus track mode with which a multichannel track can have multiple stereo outputs, each with their

own individual gain and panning controls, which in turn can be routed to any audio output. This is useful for multibus plugins. SUSHI also features aux sends and corresponding return tracks for effects processing.

5.2 Control

SUSHI supports a number of protocols and technologies for controlling and synching with external sources and devices: MIDI, OSC, Ableton Link, gRPC. MIDI input and output is supported through ALSA. This enables integration with any class compliant MIDI device, like USB MIDI keyboards or controllers. A flexible routing system allows one to route MIDI based on channels to any track. MIDI can be freely routed to tracks and MIDI Program Change and Control Change messages can be mapped to plugins and parameters, respectively. MIDI data can also be processed or generated by plugins like sequencers. SUSHI also supports OSC, and can both send OSC updates and receive notes and parameter changes through OSC. Another feature is tempo sync over Ableton Link, which enables one to seamlessly tempo sync SUSHI with other devices over WiFi.

As SUSHI is a headless host, intended for use in an embedded device, it does not feature a graphical user interface. In its place is a gRPC interface that can be used for controlling all aspects of SUSHI and hosted plugins. gRPC has bindings for most common programming languages, this gives total freedom to customize the behaviour and write a complete GUI for SUSHI in more or less any GUI framework of choice and account for multiple use cases. For some applications, user interactions will come from both front panel knobs and a handheld device such as a smartphone or a tablet. In that case, an app can use the gRPC interface to control SUSHI. Other Elk Audio OS devices could feature a built-in screen. In that case, the GUI will run on the same CPU as SUSHI, though in a different process. But as mentioned before, the GUI can be built in Python or any other programming language of choice. The dual kernel architecture of Elk Audio OS will guarantee that the graphics rendering will never interfere with the audio DSP.

5.3 Audio Frontends

SUSHI was built to work in perfect sync with RASPA, Elk's proprietary low-latency audio framework. Nevertheless, SUSHI also has built-in support for Jack,[7] as well as an offline mode where audio is read from and written to file. The latter can be used for testing in an environment that lacks audio codecs and for evaluating systems in a very early stage. It has also proven to be very useful in debugging.

The ability to run SUSHI with Jack, the most common audio framework on Linux, makes it possible to run on almost any Linux system. While it does not give the same ultra-low latency as running it with RASPA on an Elk Audio OS system, it does make it incredibly easy to test and develop plugins and setups for Elk Audio OS on a standard Linux machine. In fact, almost all of the development of SUSHI has been done on standard Linux machines.

When running with RASPA, SUSHI is limited to the number of inputs and outputs supported by the physical hardware. Conversely, when running with Jack, SUSHI exposes 8 input ports and 8 output ports that can then be freely routed to physical outputs or inputs, or other Jack software.

5.4 Configuration and Routing

Most of the initial setup of SUSHI is done through a JSON configuration file. In this file it is possible to specify the number of tracks to use, their channel setup (mono, stereo, multichannel), the plugins on the track, audio input and output routing, MIDI routing, which plugin parameters map to Control Change messages, and so on.

[7] http://jackaudio.org/.

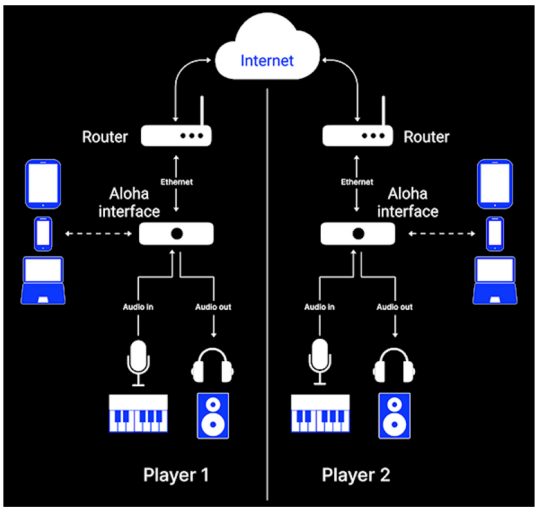

Fig. 4. A schematic diagram of the hardware components and data flow occurring during networked musical interactions between two players mediated by Aloha.

5.5 Threading

SUSHI can run its audio processing single threaded, but it also has built-in multithreading support to spread the audio processing over multiple cores, depending on the type of system it is running on. For developers that wish to utilize multithreading within a plugin, a small threading utility library called TWINE has been developed, which works with Elk Audio OS and SUSHI. This library also includes a few utility functions and wrappers for certain system calls like timers to abstract away some of the limitations of the dual kernel setup.

6 NETWORKING SUBSYSTEM

Elk Audio OS is capable of supporting the wired (via ethernet) and wireless communication of both audio signals and musical messages (such as OSC and MIDI). It offers wireless connectivity via BLE, WiFi, 4G, and even 5G. In particular Elk Audio OS presents a networking subsystem that is specifically conceived for remote musical interactions enabling musicians to play at a distance (of course compatibly with the constraints given by the physical laws and the available network infrastructure). To enable true real-time networked music performances [41] it is necessary that the OS is able to handle communication systems that avoid perceivable deteriorations of the signal. This translates in communications with latencies on the order of milliseconds, constant jitter, and with a probability of packet loss on the order of at least 10^{-2}, while at the same time preserving a high audio quality [20].

Elk's system for low-latency networked music performances is called Aloha.[8] Differently from other commercially available solutions, Aloha guarantees additions of just 1 ms latency to the audio signal when being sent, and 1 ms once received. This creates maximum space for the users' network and signal to deliver a true real-time experience. Aloha also encompasses a system for handling efficiently latency variations due to network fluctuations. While Aloha has been developed with normal cable Internet connections in mind, it is capable of interfacing with 5G communication systems. Figure 4 shows a schematic diagram of the hardware components and data flow occurring during networked musical interactions between two players mediated by Aloha.

[8] https://alohabyelk.com/.

Aloha is based on UDP. An Aloha-based interface produces a protocol data unit comprising 32 audio samples (each of 32 bits) for each audio channel, plus some overhead (about 256 bits including UDP headers). In total, the protocol data unit size is ≈290 bytes. Redundancy mechanisms help compensate for packet losses: after the transmitter sends a packet, the following packet includes 20% redundant information. Being the sampling frequency equal to 48 kHz and considering the redundancy, the packet transmission rate is approximately one packet every $32/(48 \cdot 10^3) \cdot (4/5) \approx 0.533$ ms. Mechanisms for packet error concealment [57] are also set in place.

At present Aloha has not been released yet on the market nor it is open source. The full detailed description of Aloha is out of the scope of this article.

7 ELK AUDIO OS DEVELOPMENT TOOLS

One of the core features of the Elk Audio OS is that it provides ultra-low-latency processing while still retaining the ease of development and flexibility of a conventional general-purpose Linux system. This section reviews the development tools that Elk Audio OS offers and that enable an easier and faster development of connected musical devices, with the possibility to reuse the code in future Elk Audio OS-powered products.

Elk Audio OS is able to support standard formats such as Steinberg's VST (both 2.x and 3.x) and Rack Extensions from Propellerheads. To load plugins in SUSHI, they need to be compiled for the system intended. Porting to Elk Audio OS code of plugins previously written for Windows, macOS, or even Linux is usually a rather straightforward process of recompiling the plugins using Elk's SDK.[9] Elk Audio OS provides several tools to aid this process, including a fork of the popular JUCE framework.[10]

In more detail, the SDK allows developers of musical and audio software to code their apps for different hardware devices without requiring modifications to their source code. This is possible thanks to the layer of the sophisticated architecture of Elk Audio OS, which totally abstracts the single applications from the underlying hardware. Such a layer runs also on a conventional PC (supporting Windows, Linux, and Mac OS X OSs), so the developers can write their software entirely without having to use the actual target device hardware, which speeds up the development process.

8 DISCUSSION

Elk Audio OS is a new software framework for ultra-low-latency audio and sensor data processing as well as for networked interactions. Not only it combines the resources of an embedded Linux system with the performance, audio quality, and timing guarantees typically reserved for dedicated DSP chips and microcontrollers but also comes with several tools that streamline the development of professional audio and music devices. Hence, for projects involving heterogeneous IoMusT hardware (such as smart instruments [46]), it is possible to build the whole software system upon Elk Audio OS and easily adopt existing libraries. Moreover, the availability of several networking protocols including the capability of interfacing with 5G make Elk Audio OS IoMusT-ready. Furthermore, thanks to Elk Audio OS, existing code of audio plugins developed for desktop PC can be reused for embedded devices.

Recently, the Elk Audio OS performances have been evaluated against a benchmark solution. Specifically, the study reported in Reference [58] compared the Elk Audio OS approach based on Xenomai with the most popular approach to real-time Linux, the PREEMPT RT patch [38]. Results revealed that Xenomai provides lower audio Round Trip Latency, lower scheduling latency, and

[9] https://github.com/elk-audio.
[10] https://juce.com/.

manages to exploit more CPU performance at a given latency setting while guaranteeing perfect audio quality.

A key component of this novel OS is its ability to interface with the new communication technologies that are envisioned in the Tactile Internet [26], such as 5G, which will enable the ultra-low latency and highly reliable exchange of musical content. To date, several demonstrations of these technologies used in combination with Elk Audio OS have been conducted in the form of remote music performances.[11] Specifically, thanks to Aloha, Elk Audio OS has been interfaced with Ericsson's cutting edge technologies for 5G. The result of such integration has been first showcased in September 2018 at Stockholm's Music Tech Fest and in February 2019 at Mobile World Congress in Barcelona with an actual remote music performance over 5G. Whereas in those initial demonstrations 5G testbeds were used, starting from October 2019 the real 5G infrastructure has been utilized in several demos across Europe and Asia. Notably, thanks to Elk Audio OS' networking subsystem Aloha it is possible to conceive novel communication architectures specific to the IoMusT, such as those based on the upcoming 5G as preliminary discussed in Reference [8].

All these features make it possible the development of the connected musical devices desired by contemporary musicians, along with cloud services responding to their needs of novel ways of musical expression and interactions with music and their audience [29, 39, 49, 54]. For all these reasons, it is plausible to expect that Elk Audio OS can become a game changer in the smart musical instruments landscape. Ultimately, the emergence of novel products and prototypes powered by Elk Audio OS as well as their use in actual performance contexts shows that it can fulfill the requirements listed in Section 3 (see, e.g., the Retrologue Hardware Synth described in Reference [56]).

Elk Audio OS was released open source in November 2019 and the announcement of such release was given at the 2019 Audio Developer Conferences in London. Since then a community has rapidly formed, aggregating an increasing number of code contributors from around the world: at the time of writing, the company's forum dedicated to the OS[12] gathers the contributions of more than 200 developers from industry, academia, and the maker/tinkerer communities. One of the success points for this rapid growth is certainly due to the technological advancement that the OS represents, which meets the interest of many practitioners. Other aspects that are critical to the adoption of Elk Audio OS are the relatively low barrier to entry as well as the openness to existing tools such as JUCE, Csound [23], FAUST [36], or Supercollider [60]. These factors are in line with the analysis performed by Morreale et al. about the key success criteria that enabled a community to form around the Bela platform [35]. The OS offers an extensive documentation, which includes several examples of code for rapid development of a connected audio device. Based on the collected users' feedback, users find value in the simplicity of getting started. However, large groups of people already know how to use the tools mentioned above to create audio plugins, or other interactive music software.

It is worth noticing that while Elk Audio OS was initially conceived for Internet of Musical Things applications, its use is not limited to the musical scenario. This technology indeed finds application in a large variety of systems and contexts where real-time auditory feedback is an important communication medium, especially in connected scenarios. The possible scenarios that can be envisioned include wireless acoustic sensor networks for ambient intelligence and smart cities, teleoperations in industry 4.0, smart cars, augmented and virtual reality, and speech technology. All such scenarios deeply require embedded systems that are dedicated to efficient audio computations and can interface with the network. The flexibility of ELK allows it to be run on a large

[11] https://elk.audio/5g/.
[12] https://forum.elk.audio/.

variety of platforms, from the smallest and low-cost, to the computationally most powerful and expensive ones. Therefore, Elk Audio OS can be positioned not only as an enabler of the IoMusT but also for the emerging paradigm recently termed "Internet of Audio Things" [51], where a mesh of connected devices dedicated to the production and consumption of auditory signals is created and used in effective, novel ways.

9 CONCLUSIONS

This article presented a comprehensive overview of Elk Audio OS developed by Elk, a Linux-based OS for embedded devices, conceived specifically for Internet of Musical Things applications. We analyzed the generic requirements for software running on IoMusT devices, and from this analysis, we derived that none of the existing OSs is able to fulfill the diverse requirements of IoMusT systems, which include constraints of heterogeneous hardware, hard real-time, high-quality audio, as well as various network stack (for local and remote communications). The article described the architecture and core implementations of Elk Audio OS, showing that its characteristics enable the creation of smart musical instruments and other Musical Things.

An increasing number of Musical Things are expected to be interconnected by wireless networks in the upcoming future, thus giving birth to the envisioned IoMusT. The ecosystems that will form around IoMusT technologies are expected to support novel forms of interactions between different stakeholders such as composers, performers, audience members, audio producers, live sound engineers, as well as music students and music teachers. Thanks to novel musical applications and services supporting these interactions there is a potential to revolutionize the way music is composed, performed, experienced, learned, and recorded. Whereas a lot remains to be done to accomplish the IoMusT vision, and Elk Audio OS represents one of the initial steps to make this potential more concrete.

Whether or not Elk Audio OS will be widely adopted and become a cornerstone for the IoMusT remains to be seen. Nevertheless, the availability of open source OSs such as Elk Audio OS is crucial to facilitate the rise of the IoMusT and avoid its fragmentation. With respect to this, the increasing community of developers that has recently aggregated around Elk Audio OS represents a promising perspective.

ACKNOWLEDGMENTS

The authors are grateful to all Elk team members, and in particular to Stefano Zambon, Gustav Andersson, Sharan Yagneswar, Nitin Kulkarni, Ilias bergström, Ruben Svensson, and Michele Benincaso.

REFERENCES

[1] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch. 2018. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet Things J.* 5, 6 (2018), 4428–4440.

[2] E. Berdahl. 2014. How to make embedded acoustic instruments. In *Proceedings of the Conference on New Interfaces for Musical Expression*. 140–143.

[3] E. Berdahl and W. Ju. 2011. Satellite CCRMA: A musical interaction and sound synthesis platform. In *Proceedings of the Conference on New Interfaces for Musical Expression*. 173–178.

[4] E. Berdahl, S. Salazar, and M. Borins. 2013. Embedded networking and hardware-accelerated graphics with satellite CCRMA. In *Proceedings of the Conference on New Interfaces for Musical Expression*. 325–330.

[5] E. Borgia. 2014. The Internet of Things vision: Key features, applications and open issues. *Comput. Commun.* 54 (2014), 1–31.

[6] J. H. Brown and B. Martin. 2010. How fast is fast enough? Choosing between Xenomai and Linux for real-time applications. In *Proceedings of the 12th Real-time Linux Workshop*. 1–17.

[7] G. C. Buttazzo. 2011. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*. Vol. 24. Springer Science & Business Media.
[8] M. Centenaro, P. Casari, and L. Turchet. 2020. Towards a 5G communication architecture for the Internet of Musical Things. In *Proceedings of the IEEE Conference of Open Innovations Association (FRUCT'20)*. IEEE, 38–45.
[9] A. Dunkels, B. Gronvall, and T. Voigt. 2004. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. IEEE, 455–462.
[10] J. Eyre. 2001. The digital signal processor derby. *IEEE Spectr.* 38, 6 (2001), 62–68.
[11] G. P. Fettweis. 2014. The Tactile Internet: Applications and challenges. *IEEE Vehic. Technol. Mag.* 9, 1 (2014), 64–70.
[12] S. A. Finney. 1997. Auditory feedback and musical keyboard performance. *Music Percept.: Interdisc. J.* 15, 2 (1997), 153–174.
[13] I. Franco and M. M. Wanderley. 2016. Prynth: A framework for self-contained digital music instruments. In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research*. Springer, 357–370.
[14] S. Fujii, M. Hirashima, K. Kudo, T. Ohtsuki, Y. Nakamura, and S. Oda. 2011. Synchronization error of drum kit playing with a metronome at different tempi by professional drummers. *Music Percept.: Interdisc. J.* 28, 5 (2011), 491–503.
[15] L. Gabrielli and S. Squartini. 2016. *Wireless Networked Music Performance*. Springer. DOI: https://doi.org/10.1007/978-981-10-0335-6_5
[16] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. 2009. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Science & Business Media.
[17] F. Goltz. 2018. Ableton Link–A technology to synchronize music software. In *Proceedings of the Linux Audio Conference*. 39–42.
[18] D. D. Haddad and J. A. Paradiso. 2019. The world wide web in an analog patchbay. In *Proceedings of the Conference on New Interfaces for Musical Expression*. 407–410.
[19] R. H. Jack, A. Mehrabi, T. Stockman, and A. McPherson. 2018. Action-sound latency and the perceived quality of digital musical instruments: Comparing professional percussionists and amateur musicians. *Music Percept.: Interdisc. J.* 36, 1 (2018), 109–128.
[20] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione. 2018. Low-latency networking: Where latency lurks and how to tame it. *Proc. IEEE* 107, 2 (2018), 280–306.
[21] D. Keller, C. Gomes, and L. Aliel. 2019. The handy metaphor: Bimanual, touchless interaction for the internet of musical things. *J. New Music Res.* 48, 4 (2019), 385–396.
[22] D. Keller and V. Lazzarini. 2017. Ecologically grounded creative practices in ubiquitous music. *Organised Sound* 22, 1 (2017), 61–72. DOI: https://doi.org/10.1017/S1355771816000340
[23] V. Lazzarini, S. Yi, J. Heintz, Ø. Brandtsegg, and I. McCurdy. 2016. *Csound: A Sound and Music Computing System*. Springer.
[24] C. N. Leider. 2004. *Digital Audio Workstation*. McGraw-Hill.
[25] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. 2005. TinyOS: An operating system for sensor networks. In *Ambient Intelligence*. Springer, 115–148.
[26] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van. 2016. The tactile internet: Vision, recent progress, and open challenges. *IEEE Commun. Mag.* 54, 5 (2016), 138–145.
[27] A. Manzalini and F. Marino. 2018. Operating systems for 5G services infrastructures: Convergence between IT and telecommunications industry structures. In *Proceedings of the IEEE 87th Vehicular Technology Conference*. IEEE, 1–5.
[28] A. T. Marasco and J. Allison. 2019. Connecting web audio to cyber-hacked instruments in performance. In *Proceedings of the Web Audio Conference*.
[29] J. Martinez-Avila, C. Greenhalgh, A. Hazzard, S. Benford, and A. Chamberlain. 2019. Encumbered interaction: A study of musicians preparing to perform. In *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 1–13.
[30] B. Matuszewski and F. Bevilacqua. 2018. Toward a web of audio things. In *Proceedings of the Sound and Music Computing Conference*.
[31] J. McCartney. 2002. Rethinking the computer music language: SuperCollider. *Comput. Music J.* 26, 4 (2002), 61–68.
[32] A. P. McPherson, R. H. Jack, and G. Moro. 2016. Action-sound latency: Are our tools fast enough? In *Proceedings of the Conference on New Interfaces for Musical Expression*.
[33] A. McPherson and V. Zappi. 2015. An environment for submillisecond-latency audio and sensor processing on BeagleBone black. In *Proceedings of the Audio Engineering Society Convention 138*. Audio Engineering Society. Retrieved from http://www.aes.org/e-lib/browse.cfm?elib=17755.
[34] E. Meneses, J. Wang, S. Freire, and M. M. Wanderley. 2019. A comparison of open-source Linux frameworks for an augmented musical instrument implementation. In *Proceedings of the Conference on New Interfaces for Musical Expression*. 222–227.
[35] F. Morreale, G. Moro, A. Chamberlain, S. Benford, and A. McPherson. 2017. Building a maker community around an open hardware platform. In *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 6948–6959.

[36] Y. Orlarey, D. Fober, and S. Letz. 2009. FAUST: An efficient functional approach to DSP programming. In *New Computational Paradigms for Computer Music*, G. Assayag and A. Gerzso (Eds.). Editions Delatour, Paris, France, 65–96.
[37] M. Puckette. 1997. Pure data. In *Proceedings of the International Computer Music Conference*.
[38] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. 2019. The real-time Linux kernel: A survey on PREEMPT_RT. *ACM Comput. Surveys* 52, 1 (2019), 1–36.
[39] C. Rossitto, A. Rostami, J. Tholander, D. McMillan, L. Barkhuus, C. Fischione, and L. Turchet. 2018. Musicians' initial encounters with a smart guitar. In *Proceedings of the 10th Nordic Conference on Human-Computer Interaction*. ACM, New York, NY., 13–24. DOI: https://doi.org/10.1145/3240167.3240223
[40] C. Rottondi, M. Buccoli, M. Zanoni, D. Garao, G. Verticale, and A. Sarti. 2015. Feature-based analysis of the effects of packet delay on networked musical interactions. *J. Audio Eng. Soc.* 63, 11 (2015), 864–875.
[41] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti. 2016. An overview on networked music performance technologies. *IEEE Access* 4 (2016), 8823–8843. DOI: https://doi.org/10.1109/ACCESS.2016.2628440
[42] Z. Sheng, S. Yang, Y. Yu, A. V. Vasilakos, J. A. McCann, and K. K. Leung. 2013. A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Commun.* 20, 6 (2013), 91–98.
[43] H. Shokri-Ghadikolaei, C. Fischione, P. Popovski, and M. Zorzi. 2016. Design aspects of short-range millimeter-wave networks: A MAC layer perspective. *IEEE Netw.* 30, 3 (2016), 88–96.
[44] D. Soldani and A. Manzalini. 2015. Horizon 2020 and beyond: On the 5G operating system for a true digital society. *IEEE Vehic. Technol. Mag.* 10, 1 (2015), 32–42.
[45] G. Tanev and A. Božinovski. 2014. Virtual studio technology inside music production. In *ICT Innovations 2013*, V. Trajkovik and M. Anastas (Eds.). Springer International Publishing, Heidelberg, 231–241. DOI: https://doi.org/10.1007/978-3-319-01466-1_22
[46] L. Turchet. 2019. Smart musical instruments: Vision, design principles, and future directions. *IEEE Access* 7 (2019), 8944–8963. DOI: https://doi.org/10.1109/ACCESS.2018.2876891
[47] L. Turchet, F. Antoniazzi, F. Viola, F. Giunchiglia, and G. Fazekas. 2020. The internet of musical things ontology. *J. Web Semant.* 60 (2020), 100548. DOI: https://doi.org/10.1016/j.websem.2020.100548
[48] L. Turchet and M. Barthet. 2019. Co-design of musical haptic wearables for electronic music performer's communication. *IEEE Trans. Hum.-Mach. Syst.* 49, 2 (2019), 183–193. DOI: https://doi.org/10.1109/THMS.2018.2885408
[49] L. Turchet and M. Barthet. 2019. An ubiquitous smart guitar system for collaborative musical practice. *J. New Music Res.* 48, 4 (2019), 352–365. DOI: https://doi.org/10.1080/09298215.2019.1637439
[50] L. Turchet, M. Benincaso, and C. Fischione. 2017. Examples of use cases with smart instruments. In *Proceedings of Audio Mostly Conference*. 47:1–47:5. DOI: https://doi.org/10.1145/3123514.3123553
[51] L. Turchet, G. Fazekas, M. Lagrange, H. Shokri Ghadikolaei, and C. Fischione. 2020. The Internet of Audio Things: State-of-the-art, vision, and challenges. *IEEE Internet Things J.* 7, 10 (2020), 10233–10249.
[52] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet. 2018. Internet of Musical Things: Vision and challenges. *IEEE Access* 6 (2018), 61994–62017. DOI: https://doi.org/10.1109/ACCESS.2018.2872625
[53] L. Turchet, A. McPherson, and M. Barthet. 2018. Real-time hit classification in a Smart Cajón. *Front. ICT* 5, 16 (2018). DOI: https://doi.org/10.3389/fict.2018.00016
[54] L. Turchet, J. Pauwels, C. Fischione, and G. Fazekas. 2020. Cloud-smart musical instrument interactions: Querying a large music collection with a smart guitar. *ACM Trans. Internet of Things* 1, 3 (2020), 1–29. DOI: https://doi.org/10.1145/3377881
[55] L. Turchet, T. West, and M. M. Wanderley. 2020. Touching the audience: Musical haptic wearables for augmented and participatory live music performances. *J. Person. Ubiq. Comput.* (2020), 1–21. DOI: https://doi.org/10.1007/s00779-020-01395-2
[56] L. Turchet, S. J. Willis, G. Andersson, A. Gianelli, and M. Benincaso. 2020. On making physical the control of audio plugins: The case of the retrologue hardware synthesizer. In *Proceedings of Audio Mostly Conference*. 146–151.
[57] P. Verma, A. I. Mezzay, C. Chafe, and C. Rottondi. 2020. A deep learning approach for low-latency packet loss concealment of audio signals in networked music performance applications. In *Proceedings of the 27th Conference of Open Innovations Association (FRUCT'20)*. IEEE, 268–275.
[58] L. Vignati, S. Zambon, and L. Turchet. 2020 (submitted). A comparison of real-time Linux-based architectures for embedded musical applications. *J. Audio Eng. Soc.* (2020).
[59] D. Wessel and M. Wright. 2002. Problems and prospects for intimate musical control of computers. *Comput. Music J.* 26, 3 (2002), 11–22. DOI: https://doi.org/10.1162/014892602320582945
[60] S. Wilson, D. Cottle, and N. Collins. 2011. *The SuperCollider Book*. The MIT Press.
[61] M. Wright. 2005. Open sound control: An enabling technology for musical networking. *Organised Sound* 10, 3 (2005), 193–200.

Received August 2020; accepted December 2020