IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON TOWARDS SERVICE-CENTRIC INTERNET OF THINGS (IOT): FROM MODELING TO PRACTICE

Received October 1, 2018, accepted October 18, 2018, date of publication October 23, 2018, date of current version December 3, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2877523

Smart Audio Sensors in the Internet of Things Edge for Anomaly Detection

MATTIA ANTONINI[1,2], MASSIMO VECCHIO[1], FABIO ANTONELLI[1], PIETRO DUCANGE[3], AND CHARITH PERERA[4], (Member, IEEE)

[1]OpenIoT Research Unit, FBK CREATE-NET, 38123 Povo (TN), Italy
[2]Department of Information Engineering and Computer Science, University of Trento, 38123 Povo (TN), Italy
[3]SMARTEST Research Centre, eCampus University, 22060 Novedrate (CO), Italy
[4]School of Computer Science and Informatics, Cardiff University, Cardiff CF24 3AA, U.K.

Corresponding author: Charith Perera (charith.perera@ieee.org)

This work was supported by the AGILE Project through the Horizon 2020 Programme of the European Union under Grant 688088.

ABSTRACT Everyday objects are becoming smart enough to directly connect to other nearby and remote objects and systems. These objects increasingly interact with machine learning applications that perform feature extraction and model inference in the cloud. However, this approach poses several challenges due to latency, privacy, and dependency on network connectivity between data producers and consumers. To alleviate these limitations, computation should be moved as much as possible towards the IoT edge, that is on gateways, if not directly on data producers. In this paper, we propose a design framework for smart audio sensors able to record and pre-process raw audio streams, before wirelessly transmitting the computed audio features to a modular IoT gateway. In this paper, an anomaly detection algorithm executed as a micro-service is capable of analyzing the received features, hence detecting audio anomalies in real-time. First, to assess the effectiveness of the proposed solution, we deployed a real smart environment showcase. More in detail, we adopted two different anomaly detection algorithms, namely *Elliptic Envelope* and *Isolation Forest*, that were purposely trained and deployed on an affordable IoT gateway to detect anomalous sound events happening in an office environment. Then, we numerically compared both the deployments, in terms of end-to-end latency and gateway CPU load, also deriving some ideal capacity bounds.

INDEX TERMS Anomaly detection, digital signal processing, edge computing, embedded devices, Internet of Things, IoT gateway, machine learning, novelty detection, open-source platforms, outlier detection.

I. INTRODUCTION

The term ''Internet of Things'' (IoT), originally coined in 1999 by Ashton [1], is still attracting the critical attention of a multitude of researchers and industrial players. As we write, practically all industry sectors are going to be supported by smaller devices (*i.e.*, the ''things'') able to collect data and to push them to the Internet. However, around 20 years after the IoT coinage, it would be reductive to translate such vision into the routine process of physically connecting everyday objects to the Internet. Indeed, the original promise of the IoT was about connecting such everyday objects not for the sake of having them connected, rather with the goal of making them autonomous or, in other words, *smart*. Loosely speaking, such smartening process relies on the need of ''closing the loop'', which consists of sensing, communicating, reasoning, taking decisions, and actuating back into the physical side [2]. As already noticed by Poncela *et al.* [3], this can be seen as the specification of the more general Monitor-Analyze-Plan-Execute plus Knowledge (MAPE-K) reference model for Autonomic Computing, as initially proposed by P. Horn in 2001, and later formalized in [4].

So far, several research and development efforts on IoT have been conducted on the design and implementation of software solutions able to manage devices, resources and data, abstracting from their intrinsic heterogeneity, at hardware and software levels. The result of this race was the plethora of IoT software platforms released in the last few years, either as open source solutions or as commercial products [5]. All of them (at different levels of maturity) are able to support the exploding number of connected devices and associated data, by exposing common services, such as device virtualization, provisioning, and control, data production/consumption/storage, etc. [6].

Less has been done to truly realize what is usually referred to as ''embedded intelligence'' [7], that is one of the pillars of the Smart World vision. More in detail, the latter can be seen as the composition of various elements (*i.e.*, the ''smart things'') at different scales, layers, and granularity, ranging from smart sensors, smart devices, and objects to smart homes, smart infrastructures, smart factories, smart cities, etc. As observed Ma *et al.* [8], ubiquitous sensors, actuators, and embedded devices are currently paving the way towards the realization of such a paradigm (which is still in its infancy, from a research perspective) where computational intelligence is distributed throughout the physical environment (*i.e.*, the reason why it is called ''embedded'') to offer more reliable, more efficient, more relevant, and safer services to people. Indeed, nowadays, only quite expensive devices (most likely complex systems already composed of multiple sensors and actuators, ranging from mobile phones to connected vehicles) are really able to intelligently interact with humans and other digital systems, while affordable, yet reliable, *smart things* are quite less common.

In this paper, we make a step forward in this direction, by introducing a cost-effective, smart IoT device able to autonomously perform anomaly detection on-site. Generally speaking, anomaly detection is a branch of artificial intelligence (AI) that deals with the identification of patterns in a set or in a stream of digital data that do not conform to a pre-defined notion of normal behavior [9]. Usually, in order to detect anomalies, first, a model describing the patterns of a normal behavior is defined. Then, anomalous patterns (*i.e.*, patterns which are sufficiently distant from the ''normal'' model) may be detected. In IoT contexts, detecting anomalies may represent a very critical task. For this reason, often, current end-to-end solutions limit the role the IoT nodes (physically located near the phenomenon under observation, hence on the one end the technology stack) to simple data producers, while offloading the whole intelligence of the application to centralized, cloud-based elements (physically located on the other end of the technology stack), hence acting as the unique data consumer, in a classical publish/subscribe design pattern [10]. However, recently, new architectural paradigms are being introduced, which are usually referred to as the Fog and Edge Computing paradigms [11]–[13]. These two paradigms, though being quite different from each other, share several architectural choices. Briefly, the key-concept they share is the attempt to distribute computation, communication, control and storage closer to the end users, along the so-called *cloud-to-thing continuum* [14]. At least on paper and if correctly implemented, such a concept should finally overcome some of the major challenges that are currently hindering the IoT global dissemination and adoption. Some of these challenges are stringent latency requirements, network bandwidth constraints, resource-constrained devices, security and privacy requirements, uninterrupted services despite intermittent connectivity with the cloud endpoint [15], [16].

The main contributions of this paper, whose main goal is to prove the effectiveness of distributing the intelligence of a real IoT application along a continuum comprising several technological entities (*e.g.*, sensors, embedded devices, gateways, software platforms, etc.), may be summarized as:

- *Design Framework for Smart Audio Sensors*: in this paper, we analyze the bandwidth constraints imposed by a Bluetooth-UART interface. Since one of the main features of a Smart Audio Sensor (SAS) device is the wireless connectivity, we introduce a theoretical framework to tune its parameters, taking into consideration both latency and bandwidth constraints. Thus, based on the specific requirements of the application scenario at hand, the SAS can be dynamically re-configured to meet these requirements, without breaking the technology constraints.
- *Exploitation of the Edge Computing paradigm*: SASs autonomously sample the surrounding audio environment using cheap onboard microphones. Then, instead of transferring the digitized version of the acquired signal as is (as a ''dumb'' data producer would do), they first pre-process the signal in real-time by transforming it into the frequency domain and by computing some state-of-the-art features in this domain. The processing output is a more compact, though unrecoverable, representation of the original signal in the feature domain. This approach has several advantages with respect to the approach of sending the raw audio stream over the network: first of all, our feature-domain representation of the signal is compressed to a factor of 8 with respect to the original time-domain representation. Among the other advantages, at network level this means reduced network traffic, reduced radio utilization, less complex access policies to the wireless channel, reduced number of re-transmissions, etc. [17]. At application level, since the signal transformation is partially irreversible (*e.g.*, once transformed into the feature domain, the original signal cannot be completely reconstructed), reasonable levels of privacy are guaranteed by design.
- *Full integration with the AGILE ecosystem*: the SAS and the whole design framework have been fully integrated with the AGILE IoT gateway ecosystem, in order to validate their applicability. In particular, we propose a proof-of-concept scenario that describes how SASs can interact with the AGILE gateway and which modules we developed. Such modules allow developers and users to simply connect, configure and use the SASs, without much integration effort.
- *Modular, maintainable and scalable software architecture*: obeying the AGILE's software design and architectural principles, the developed modules follow the micro-service philosophy, thus they were independently implemented. Given that every module is independent of each other, the system allows simple updates. Then, if a module fails or crashes for some reason, the system is resilient and remains up, eventually with a limited set of capabilities. Moreover, the system is scalable: if several devices have to be managed, then the system can

dynamically instantiate resources and modules, in order to cope with the increased load, and vice-versa. Last, but not least, we are distributing the computational load among different devices. In fact, the SASs perform feature extraction, while the IoT gateway only runs the trained model.

- *Detection of audio anomalies in an office environment*: as a proof of concept, we validate our approach in a real use case, where we deployed a SAS in an office environment, in order to detect anomalies using a cheap microphone. The SAS is connected via Bluetooth to an AGILE gateway that infers, using an Anomaly Detection algorithm, over the features stream. Detection models have been trained using a pre-recorded audio stream coming from the same SAS. We evaluated two different aspects: firstly, the detection delay from the first recorded sample to the algorithm output, and then the CPU load of our module with respect to the AGILE CPU load.

The remaining of this paper is organized as following: Section II reviews the most recent works related to our scientific methodology, while Section III thoroughly describes the main technologies we adopted to develop our solution, by focusing on two important architectural elements supporting our technological framework, namely the Teensy-based device and the AGILE IoT gateway. Then, Section IV describes the design and the implementation of the core component of our solution, namely the Smart Audio Sensor (SAS), which is then comprehensively tested in a concrete anomaly detection use case, detailed in Section V. Finally, Section VI concludes this paper, also highlighting the most promising future research directions in this context.

II. RELATED WORKS

The goal of an anomaly detection process is to discover unusual events in a specific framework. It has been successfully applied in several domains, such as network intrusion detection [18], acoustic surveillance [19], natural disaster prediction [20], remote health-care [21], etc. Several surveys have been proposed in the specialized literature, in which the huge amount of different anomaly detection methodologies have been discussed. The most effective models for anomaly detection are mainly based on statistical analysis, information and graph theory, machine learning and data mining [9], [22], [23].

A. ANOMALY DETECTION ALGORITHMS

In general, models for anomaly detection can be derived using unsupervised, semi-supervised and supervised training algorithms [24]. When dealing with unsupervised training algorithms, no labeled training datasets are adopted for building the models. Usually, only patterns describing normal behavior are available, while additional information regarding anomalies are not available. In these situations, algorithms based on clustering, data density and proximity, and one-class detection models may be adopted [24]–[26]. Among these algorithms, the one-class support vector machine (1-SVM) algorithm still continues being one of the most adopted for unsupervised anomaly detection [27], [28]: a kernel model, namely a decision function, is derived by using normal data patterns, while new patterns projecting too far from the model are marked as anomalies. Moreover, also frequent pattern and association rule mining algorithms have been adopted for unsupervised anomaly detection: normal data vectors can be considered as transactions and frequent patterns and association rules can be mined. New transactions (*i.e.*, new measured data) that cannot be projected into the frequent patterns or into the mined rules are marked as anomalies [29], [30].

When a sufficient number of labeled training patterns is available, including both normal and anomalous situations, supervised learning algorithms can be employed. In particular, multi-class classification models can be trained using a fully labeled training set. Classification models, such as decision tree [31], multi-class SVM [19] and Bayesian classifier [32], have been successfully employed for different kinds of anomaly detection. A recent survey comparing a number of classification algorithms for a specific anomaly detection framework, namely intrusion detection in networks, is available in [33].

Finally, there exist specific situations in which only a low number of anomalous labeled patterns are available. In such cases, a semi-supervised training algorithm can be employed for learning the models for detecting anomalies. These kinds of algorithms usually are based on the hybridization of unsupervised and supervised algorithms [34]. As recently discussed by the authors of [35], a classification model (specifically, a neural network) is trained using the labeled patterns. Then, the unlabeled patterns, appropriately pre-elaborated using a fuzziness function, are classified and exploited for reinforcing the structure of the models. The work in [36] discusses the use of the so-called deep auto-encoder (DAE), a kind of deep belief network followed by an ensemble of K-NN classifiers. In particular, unlabeled data are used for training the DAE in order to reduce the dimensionality of the data [37]. The subset of labeled data, transformed by using the DAE, is used for training the ensemble of K-NN classifiers. New patterns are filtered by the DAE and then classified by means of the ensemble.

B. ANOMALY DETECTION IN IoT-BASED ARCHITECTURES

In the following, we summarize some recent contributions in the framework of anomaly detection carried out considering an IoT-based architecture. Islam *et al.* [20] discuss the application of a novel association rule-based approach for handling uncertain, vague and noisy data in anomaly detection. In particular, authors envision that data from sensors feed a web-based expert system able to predict flooding using rainfall and temperature measurements. In particular, the expert system employs a database of belief association rules that allows to identify anomalies in the

level of rainfall and temperature and to predict a flooding event.

Trilles *et al.* [38] present a classical cloud-based methodology for handing, in a real-time fashion, heterogeneous sources of data streams. As a proof-of-concept, they discuss the application of their methodology for environmental anomaly detection using meteorological data. The proposed methodology consider a logical architecture which comprises three layers, namely content, services and application layers. The content layer is composed of different sets of heterogeneous sensors which are deployed in a specific scenario and send streams of information to the service layer. The services layer includes *i)* connectors for handling the streams coming from different data sources, *ii)* a brokering system for allowing the access to data coming from sources that use different communication protocols and message encodings, and *iii)* algorithms that elaborate data. The application layer includes all users' applications. The different layers communicate by means of real-time massage services. In the discussed proof-of-concept, simple CUSUM (cumulative sum) statistical algorithm has been adopted, which is a nonparametric and univariate method for anomaly detection.

Lyu *et al.* [39] discuss a fog computing architecture for anomaly detection. Raw data are collected by the end nodes and sent to the fog nodes which are in charge of building the model for detecting the anomalies, performing both sensor layer and fog layer clustering. The results of these clustering steps are sent to the cloud in order to be merged. The cloud layer sends the merged clusters to the fog layers which carry out the anomaly detection steps. The identification of the clusters is based on a hyper-ellipsoidal clustering algorithm which adopts a scoring mechanism for distinguishing normal and anomalous events [40]. The proposed architecture is compared with both a centralized architecture and a distributed architecture in the same fog computing framework. Similar architectures have been previously introduced by Rajasegarar *et al.* [40] considering a multi-level hierarchical topology of Wireless Sensor Networks (WSNs). In the centralized architecture, sensor nodes just sent data to the cloud server which carry out all the data elaboration. In the distributed architecture, end nodes conduct a clustering step at the sensor layer, send the results of the clustering to the fog and cloud layers, and receive the results of fog and cloud layers clustering. Fog and cloud layers receive the clustering results from the lower layers, merge the clusters and send back to the end nodes the results of the merging. The fog layers are also in charge of the anomaly detection task. As an application scenario, the authors present a smart traffic light system, where the traffic light acts as fog node and receives signals from different devices such as sensors mounted on cars and pedestrian, flashing lights of the ambulances. The smart traffic light process all the data coming from the street in order to maintain a smooth and safe traffic flow.

A fog computing architecture for anomaly detection in smart cities has been recently presented in [24]. Pereira dos Santos *et al.* [24] discuss the application of their approach for monitoring the air quality in the city of Antwerp, Belgium, considering Low Power Wide Area Network (LPWAN) technologies for the communications. Also in this case, end nodes send raw data to the fog resources which perform the anomaly detection in a distributed fashion. Fog layers may send alerts to both end nodes and cloud servers, whenever anomalies have been detected. The cloud layer combines the results of the anomaly detection in order to update in real-time the status of the entire network. Moreover, the cloud layer can also perform global anomaly detection operations and show the results to the central dashboard of a control room of the smart city. The anomaly detection procedure has been carried out using an unsupervised approach based on clustering algorithms.

Rathore *et al.* [41] discuss a real-time geo-visualization framework. The framework is fed by micro-climate data sensed and transmitted by low-cost multi-sensors. These sensors send raw data to the gateway using the Zig-Bee transmission protocol. Then, by means of a 3G/4G wireless modem, data are transferred to a cloud server for the persistent storage. The anomaly detection process and the interactive geo-visualization have been implemented as a web application. The application gets data by means of SQL queries towards the data cloud server. Hyper-ellipsoidal clustering algorithms have been adopted for the anomaly detection.

C. ACOUSTIC ANOMALY DETECTION IN IoT CONTEXTS

We also identified some recent contributions regarding acoustic anomaly detection (AAD) based on IoT architecture. The very recent contribution in [19] introduces a pervasive IoT-based indoor acoustic surveillance architecture that detects and localizes anomalous sounds associated with abnormal events. The anomaly detection process is carried out in a distributed fashion: the proposed architecture includes both smart sensors, devoted to monitoring the environment, and delegate sensors, which are in charge of assisting the management of the sensors, the identification and the localization of anomalous events. Smart sensors are equipped also with resources capable to carry out local abnormality detection. Both SVM and LDA models have been adopted for the classification stage of the sound events.

An edge computing-based architecture for audio event detection has been discussed in [21]. Wireless audio sensors are deployed in indoor environments and raw data are sent to data concentrator devices, which are equipped with graphics processing units (GPUs). Such devices are in charge of data pre-processing, including feature extraction and anomaly detection tasks. The detection model is based on both clustering and classification algorithms. The data concentrators send the results of their elaboration to a remote server that takes care of the needs of people living in the considered scenario.

Some of the results achieve in the context of a EU project called DYNAMAP [42], aimed at developing low-cost sensor networks for real-time noise mapping, have been reviewed in [43]. In particular, the authors show the results achieved by their anomalous noise events detector algorithm for

TABLE 1. Summary of the main features of the recent methods for anomaly detection in IoT architectures.

Reference	Year	Technological approach	Anomaly Detection Model	Type of Training	Anomaly Detection Level	Data from sensors
[20]	2018	Web service	Association Rule	Unsupervised	Centralized	Raw
[38]	2017	WSN + Data brokering platform	CUSUM	Unsupervised	Centralized	Raw
[39]	2017	Fog	Clustering	Unsupervised	Distributed	Raw
[40]	2014	hierarchical WSN	Clustering	Unsupervised	Distributed	Pre-elaborated
[24]	2018	Fog	Clustering	Unsupervised	Distributed	Raw
[41]	2018	Cloud	Clustering	Unsupervised	Centralized	Raw
[19]	2018	Collaborative WSN	SMV and LDA	Supervised	Distributed	Pre-elaborated
[21]	2017	WSN + GPU-powered Edge gateway	Clustering and Classification	Semi-Supervised	Distributed	Raw
[43]	2017	hierarchical WSN	Classification	Supervised	Distributed	Pre-elaborated

mapping the traffic noise, considering both urban and sub-urban areas. The algorithm is based on a two-class anomaly detection model which discriminates between normal road traffic noise and anomalous noise events. A set of smart and low-cost acoustic sensors have been deployed along the roads to be monitored. These sensors perform simple signal pre-processing and also the event classification exploiting acoustic models based on machine learning algorithms. All the classification labels are sent to a centralized server which is in charge of updating the noise maps.

Finally, even though the work is not properly focused on IoT architecture, it is worth discussing the issues regarding AAD arisen in [44]. This work deals with important challenges in AAD, namely intra-class variations, such as the different duration for the same sound type, and spectral-temporal properties across classes, which include impulse-like sounds, tonal events, and noise-like events. Among the latter types of sounds/events, we can found, respectively, door slams, phone rings and printer sounds. In particular, the authors propose the use of both contextual information and prior knowledge of the event category and of the event boundary. Random forests are employed as models for anomaly detection.

D. GAP ANALYSIS

Table 1 summarizes the main features of the aforementioned recent approaches to anomaly detection in the IoT context. Observing this table, we notice that the majority of the recent related literature sticks to classical technological approaches (*e.g.*, web/cloud-based frameworks [20], [38], [41], and distributed WSN deployments [19], [40], [43]). Indeed, only a few works attempt to rely on the Fog/Edge Computing paradigm to distribute the execution of the different tasks of an environmental sensing IoT application along the cloud-to-thing continuum [21], [24], [39].

However, it is worth noticing that the fog node used in [39] is a full-fledged server PC equipped with an Intel i7-4790 quad core processor and 16 GB of RAM while, in [21], even though the authors propose a Mobile Edge Computing (MEC) [45] approach, they rely on a high performance General Purpose Graphics Processing Unit (GE-GPU) directly installed on a Jetson TK1 development board [46], that is yet a quite powerful (and expensive) edge gateway device. On the other hand, as described in Section III-B, the full gateway framework adopted in our experimentation is deployed on a very cheap Raspberry PI 3 [47] single-board computer.

Moreover, the sensor nodes of [39] simply collect environmental data and wirelessly transmit the digitized values to the fog node, which is responsible of the full processing and analysis of all raw data flows coming from (possibly various) sensor nodes. The same approach is also followed in [24] to detect anomalies in slowly changing environmental phenomena (*e.g.*, 3 particle matter indicators enriched with GPS locations). Conversely, as described in Section IV, in our framework the sensor nodes are able to locally pre-process high-definition raw audio streams in real time, hence only transmitting the extracted features to the near gateway. In this case, the intelligence of the application is truly diluted along the technology chain (*e.g.*, from tiny sensing terminals, through cheap IoT gateways, until web/cloud endpoints), as also the most constrained devices of the chain can bear non negligible computational overheads.

III. TECHNOLOGICAL BACKGROUND

This section introduces and describes the main Edge technologies adopted to develop and assess the proposed framework.

A. THE TEENSY-BASED SMART AUDIO SENSOR

A Smart Audio Sensor (SAS) was originally defined in [48] as an embedded device equipped with one or more microphones that is able to record and perform some computations *directly* on the audio stream, without losing real-time capabilities. In this way, it is possible to have devices that are able to monitor the surrounding environment, even in some edge situations. For instance, a SAS deployed in a street may detect gunfire events, even in absence of light or illumination. Moreover, by using an array of microphones, it may be also possible to identify the direction of the event source.

However, SASs should be developed adopting powerful computing units able to execute operations directly on the audio stream (*e.g.*, ARM Cortex-M4). In the prototyping landscape, one of the most promising boards for embedded intensive computations is the Teensy board [49]. Teensy, developed by PJRC, is a versatile and powerful development platform for embedded projects compatible with the Arduino Environment, thanks to the Teensyduino libraries. Most of the Arduino sketches run on Teensy boards. Teensy is available in six different flavors, based on the requirements of the project.

The most powerful board is the Teensy v3.6, that is equipped with a 32 bit ARM Cortex-M4F processor, with the DSP instruction set, working at 180MHz with a floating point unit (FPU), 256KB of RAM, 1MB of Flash memory, 58 Digital I/O, 6 UART interfaces, 4 I^2C buses, 3 SPI interfaces, 2 I^2S Digital Audio buses, 1 micro-SD card slot and the possibility to connect one Ethernet shield at 100Mbps.

Teensy boards have been designed by following the Arduino philosophy *"Easy to mount, cheap to produce"* in order to extend the board capabilities. Shields can be easily developed and plugged, thanks to the high availability of I/O pins. Even Arduino shields can be plugged to a Teensy board using the Teensy Arduino Shield Adapter.

A powerful extension shield for Teensy boards is the Audio Shield. This board, created by PJRC, is able to add I/O audio capabilities to Teensy. It is powered by the powerful SGTL5000 Low Power Stereo Codec [50] and allows 16 bits high-quality audio recording at 44.1 KHz (CD quality), using either the on-board mono-channel microphone or the stereo line-level input. Moreover, it supports stereo line-level output and stereo headphones through the 3.5mm jack soldered on the board. A Teensy board can be physically connected to an Audio shield using the 14x2 extension header and the audio stream is transferred from one device to the other one, using the I^2S Digital Audio bus, that is a special communication bus designed for audio streams that supports up to 2 different audio channels. The audio data transfer is managed by the Teensy Audio library, a software library that is also able to execute various operations on audio streams. Furthermore, if the library is executed on Teensy 3.x boards, it can run real-time, computationally-intensive operations, like FFTs, using the DSP instruction set provided by the ARM Cortex-M4 processor. In addition, another library, OpenAudio for Teensy [51], has been developed on top of the Teensy Audio library and provides additional features and operations for real-time audio processing. This enables developers to create sound-reactive projects with reduced costs.

B. THE AGILE-BASED IoT GATEWAY FRAMEWORK

In general, SASs represent a versatile kind of embedded devices that can be adopted within almost every IoT scenario, spanning from industrial plants to smart city contexts, simply by connecting them to a gateway, either through a wired or wireless radio communication technology. In particular, gateways are able to bridge different networking stacks, *e.g.*, IP and non-IP worlds, and execute computational intensive operations closer to the deployed devices. In the context of gateway devices suitable for the IoT, one opportunity is offered by the Adaptive Gateway for dIverse muLtiple Environments (AGILE) [52]. AGILE is an open source modular software for IoT gateways which supports a wide range of components. Hence, it enables developers, users and companies to develop their own solutions and products on top of its stack. The AGILE architecture has been designed by following the micro-service paradigm, which was originally proposed for distributed systems. Nowadays, this paradigm is well-recognized and adopted in several different fields (*e.g.*, cloud computing), as it enables strong modularity, maintainability, scalability, reliability and resiliency against failures of systems [53]. Indeed, if a service fails, the whole system remains alive, with reduced capabilities in the the worst case. The AGILE project consortium adopted these concepts and ideas to design a robust framework for IoT gateways. This framework consists of different modules, where each module is implemented as an independent service within a Docker container,[1] which creates a sandbox where the component executes. Within a container, it is possible to use very specific technologies and programming languages, without suffering any compatibility issue with other modules. Interactions among modules exploit a well-defined set of Application Programming Interfaces (APIs) defined in the framework. APIs are available in two different manners: using a common Inter-Process Communication (IPC) bus (*e.g.*, DBus[2]) or using RESTful interfaces.

The AGILE gateway framework runs on many different hardware architectures, including x86 and ARM. In particular, AGILE has been successfully tested and deployed on affordable Raspberry Pi 3 (RPi3) computers [47], which is a single-board computer. Despite its reduced cost, a RPi computer can rely on several analog and digital input/output lines to extend its basic capabilities (*e.g.*, connecting sensors and actuators). For instance, the Libelium company (which is partner of the AGILE consortium) has developed an extension shield for the RPi, called the Maker's Shield [54], [55], equipped with sensors, buttons, LEDs, and two XBee sockets. This board can help makers and developers to create complete IoT solutions in a easy and straightforward way, by just plugging the shield on top of the RPi GPIO header. Moreover, considering the presence of two XBee-compliant sockets, it is possible to extend the networking capabilities of the AGILE gateway by adding new families of supported devices and radio technologies.

Finally, the AGILE framework has been designed to be suitable in many scenarios and applications. To this aim, the consortium identified five different pilot tests: open field and cattle monitoring, enhanced retail services, port area monitoring for public safety, air quality and pollution monitoring, and self tracking. All these pilots are implemented by exploiting the modularity and the fine-granularity of the whole framework. In Section V, we will introduce a new showcase for the AGILE IoT gateway, that is the rare sound event detector in an office environment, based on anomaly detection algorithms.

IV. THE PROPOSED WIRELESS SMART AUDIO SENSOR

In this paper, we propose a tiny and affordable wireless smart audio sensor (SAS) for indoor environments. Our wireless SAS has been developed using commercial boards and components. As depicted in Figure 1, the device comprises

[1] https://www.docker.com/

[2] https://www.freedesktop.org/wiki/Software/dbus/

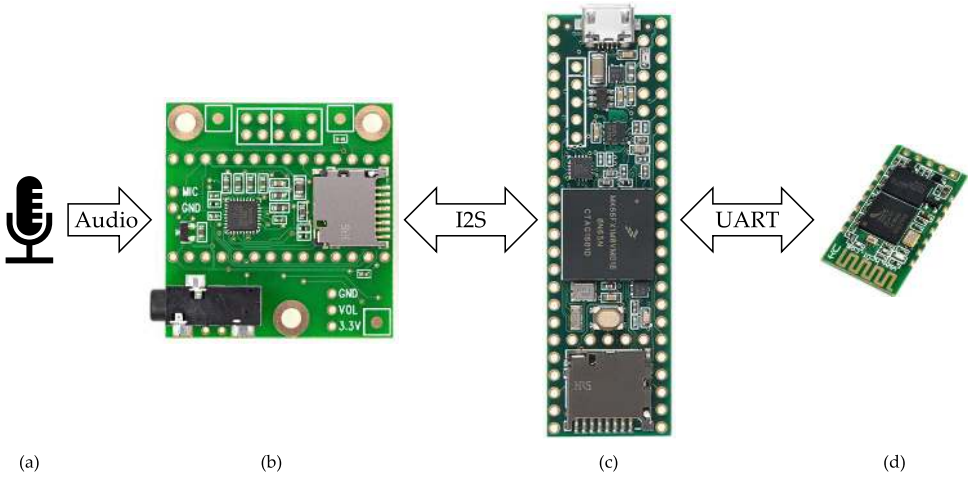

FIGURE 1. Hardware schema of the proposed Wireless Smart Audio Sensor. A Teensy 3.6 (c) is connected, through I^2S bus, to a Teensy Audio Shield (b) that records the audio signal coming from a mono-channel microphone (a). Then, the Teensy MCU (c) computes the FFT and Mel coefficients of a recorded audio frame. Finally, the Teensy board (c) sends all the Mel coefficients to an UART endpoint using a UART interface provided by a HC-05 Bluetooth 2.0+EDR module (d).

four different modules: a mono-channel electret microphone capsule (Figure 1a), a Teensy Audio Shield (Figure 1b) for audio recording at 44.1 KHz with 16-bit resolution, a Teensy 3.6 board (Figure 1c) for local audio processing, and a HC-05 Bluetooth module (Figure 1d) for serial data transfer.

During the initial phase of the design of our SAS (Figure 2a), the device was conceived to record an audio stream with sample-rate of 44.1 KHz and 16-bit resolution and to transmit the signal over an UART interface using the Bluetooth module. Another device, e.g. gateway, was in charge to perform some addition computations on the sampled signal. The radio module is able to establish a Bluetooth 2.0 + EDR connection (up to 3 Mbps) and supports UART baud-rates up to 460800 baud (from specifications). We conducted some tests and figured out that the module guarantees good performances up to 230400 baud using the 8N1 mode. Such baud-rate is not enough to transmit the audio stream since it requires (in this case 1 bps = 1 baud)

$$\begin{aligned} Sample\_Rate \cdot Bit\_Resolution &= 44100 \cdot 16 \\ &= 705600\ bps, \end{aligned} \quad (1)$$

where each sample is represented with 16 bits. A possible solution was to reduce the sample-rate to a value that was small enough to realize a bit rate that fits in the available bandwidth. Starting from the maximum baud-rate, the maximum achievable sample rate is calculated by using (2).

$$\begin{aligned} \frac{Maximum\_baudrate}{Bit\_Resolution} &= \frac{230400}{16} \\ &= 14400\ Hz. \end{aligned} \quad (2)$$

Since 14400 Hz is not a recommended sample-rate (is not an integer division of 44100), the first smaller acceptable rate is 11025 Hz. The Nyquist-Shannon sampling theorem tells that the maximum available signal bandwidth is one half (5512.5 Hz) of the sampling-rate, thus the computed sample-rate is too small to create a Smart Audio Sensor because high-frequency components are required in many applications.

The above bandwidth bottleneck pushed a re-design of the Teensy board behavior. At the beginning, the DSP pre-processing was not performed on the SAS but on the other side of the Bluetooth link, e.g., gateway. SASs are commonly adopted in use-cases in which the raw audio stream is transformed to other more meaningful quantities such as Mel-Frequency Cepstral Coefficients (MFCC) or Mel coefficients. The former are mainly used as features for speech-based applications, the latter are frequently adopted as features in non-speech scenarios. Both of them require the preliminary computation of the frequency spectrum from the raw audio stream. The Teensy's libraries provide efficient DSP operations exploiting the DSP capabilities offered by the ARM M4F MCU. Using such libraries, we have implemented the whole software flow to compute the Mel coefficients within the Teensy firmware. Figure 2b shows the sequence of operations required to calculate such coefficients. After the sampling performed by the Audio Shield at $f_{sr} = 44100Hz$, the Teensy MCU applies the Hanning window to the audio stream, in order to reduce discontinuities in the signal, and calculates the Fast Fourier Transform (FFT) at N_{FFT} points using a temporal window large N_{FFT} audio samples overlapped by $\delta_{overlap}$ with the previous window. Overlapping is used to maintain a high correlation between following windows. The FFT implementation returns N_{FFT} complex samples made of two *float32* numbers. Then, we compute the square magnitude of the FFT output in order to have an instantaneous estimation of the power distribution over frequency. Consequently, we apply the Mel-Filtering using N_{mel} bins. This filtering is performed using a filter-bank, composed

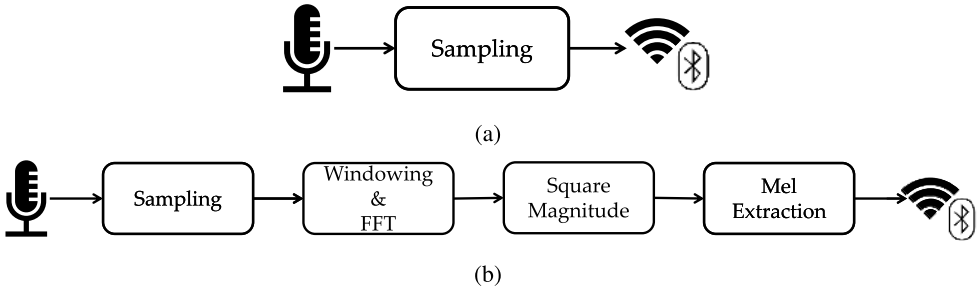

FIGURE 2. Software flowcharts of the implemented SAS. (a) is the early design of the SAS with only recording capabilities. (b) is the re-designed version of the SAS with embedded feature extraction. (a) Early design. (b) After re-design.

of N_{mel} filters, that implements the mel-scale, the non-linear perception scale of the human ear [56]. Every Mel-filter has triangular shape, has response 1 at the central frequency and it linearly decreases down to 0 when it reaches the central frequencies of neighbor filters. The frequency response of the i-th filter is described in (3).

$$H_{mel_i}(k) = \begin{cases} 0 & k < f(i-1) \\ \dfrac{k - f(i-1)}{f(i) - f(i-1)} & f(i-1) \leq k < f(i) \\ 1 & k = f(i) \\ \dfrac{f(i+1) - k}{f(i+1) - f(i)} & f(i) < k \leq f(i+1) \\ 0 & k > f(i+1) \end{cases} \tag{3}$$

$f(i)$ is the central frequency of the i-th filter. To calculate these frequencies, two different techniques have been proposed in literature: the Slaney's formulation [57] and the HTK's formulation [58]. In this paper, we adopt the former formulation that splits the frequency domain in to two different regions: a linear region for frequencies within the range 0-1000 Hz and a log region for frequencies grater than 1000Hz. Equations (4) and (5) describe how to pass from hertz to mel and vice-versa, respectively.

$$m = \begin{cases} \dfrac{f}{200/3} & f \leq 1000Hz \\ \dfrac{1000}{200/3} + \dfrac{log(6.4)}{27} \cdot log\left(\dfrac{f}{1000}\right) & f > 1000Hz \end{cases} \tag{4}$$

$$f = \begin{cases} \dfrac{200}{3} \cdot m & m \leq \dfrac{1000}{200/3} \\ 1000 \cdot exp\left(\dfrac{log(6.4)}{27} \cdot \left(m - \dfrac{1000}{200/3}\right)\right) & m > \dfrac{1000}{200/3} \end{cases} \tag{5}$$

In order to compute the central frequencies, we have firstly to calculate the mel representations of the lower and higher frequencies of the audio bandwidth, *e.g.*, 0 Hz and 22050 Hz, using (4). Then, we compute N_{mel} linearly spaced values in the range of the previously computed lower and higher mel representations. Now, applying (5) to every value, we obtain the central frequencies $f(i)$ of our filters.

After the mel-filtering, the smart audio sensor transmits the N_{mel} coefficients, represented as *float32*, over a Bluetooth channel using an 8N1 UART interface working at br baud.

A. DESIGN FRAMEWORK

In this section, we present and develop our design framework for Smart Audio Sensors that perform the computations presented above. Before starting, we have to define four parameters that will be tuned in order to correctly implement the SAS. Since the SAS operates on frequency representations of the audio stream, the first two parameters that we present are the length of the temporal window on which we compute the FFT (N_{FFT}) and the overlapping fraction ($\delta_{overlap}$) between two following audio windows. These parameters influence the maximum length of the pre-processing window, the length of the temporal window and the granularity of the audio transformation. The third parameter is the number of Mel coefficients (N_{mel}) computed over the frequency spectrum. It affects the transmission delay and the granularity of filters over the spectrum. The last parameter is the baud-rate (br) of the Bluetooth module. It affects only the transmission delay.

A Smart Audio Sensor should be designed by minimizing the latency introduced by recording (t_{rec}), pre-processing (t_{pre}) and data-transmission (t_{tx}) phases. We assume that the maximum latency for a real-time response is 150ms. This particular value is the maximum Mouth-to-Ear latency for VoIP systems, as defined in the G.114 ITU Recommendation. (6) defines the *Real-Time condition*.

$$t_{rec} + t_{pre} + t_{tx} < 0.15[s]. \tag{6}$$

Moreover, during the design phase of a SAS, we have to set the processing and the transmission parameters in order to satisfy another condition that we call *Buffering-Processing condition* (7).

$$t_{pre} + t_{tx} < t_{buffering}. \tag{7}$$

Since the FFT has to buffer N_{FFT} samples before it is able to compute the frequency spectrogram and the window overlap fraction is $\delta_{overlap}$, a full pre-processing buffer is available

every

$$t_{buffering} = \left(1 - \delta_{overlap}\right) \frac{N_{FFT}}{f_{sr}} [s], \quad (8)$$

where f_{sr} is the sampling rate. The SAS has to complete all the pre-processing operations and the transmission phase within the buffering window $t_{buffering}$ in order to not overlap with other adjacent ones.

On the left-hand side of (7), we have two terms: t_{pre} and t_{tx}. The former can be rewritten as

$$t_{pre} = t_{FFT} + t_{mag} + t_{mel}, \quad (9)$$

where t_{FFT}, t_{mag} and t_{mel} are the temporal duration of FFT, square magnitude and mel-coefficients extraction operations, respectively. More in details, time required to compute an FFT t_{FFT} operation depends on the number of points N_{FFT} and it has shape

$$t_{FFT} = 4.3 \cdot 10^{-8} \cdot N_{FFT} \cdot log_2(N_{FFT}) \quad [s], \quad (10)$$

where the coefficient $4.3 \cdot 10^{-8}$ has been empirically found by interpolating time durations required to compute FFTs with different value of N_{FFT}.

The second term (t_{mag}) of (9) describes the time required to compute the square magnitude of the FFT spectrogram. Even this term depends on the number of FFT coefficients N_{FFT} and we have empirically found that it has shape

$$t_{mag} = 2.46 \cdot 10^{-8} \cdot N_{FFT} + 2.5 \cdot 10^{-6} \quad [s] \quad (11)$$

The last term (t_{mel}) of (9) represents the time required by the Teensy MCU to compute the mel-coefficients starting from the squared magnitude of the FFT coefficients. Equation (12) shows the experimental formula to compute t_{mel}.

$$t_{mel} = 5.1 \cdot 10^{-7} \cdot N_{mel} + 3 \cdot 10^{-7} \cdot N_{FFT} - 1.9 \cdot 10^{-5} \quad [s] \quad (12)$$

All empirical coefficients have been found by running 5000 times each single operation in the pre-processing loop with different values of parameters $N_{FFT} \in \{256, 512, 1024, 2048, 4096\}$ and $N_{mel} \in \{40, 64, 128\}$. Moreover, we used the ARM functions available in the Teensy *arm_math.h* library.

The left-hand side of the *Buffering-Processing condition* (Equation (7)) contains a second term, t_{tx}, that keeps track of the time spent to transmit the N_{mel} mel coefficients through an UART interface over the Bluetooth channel. This term (Formula (13)) depends on three different parameters: the number of coefficients N_{mel}, the effective bit-rate br_{eff} and the number of bits (N_{bits}) used to represent each coefficient, *e.g.*, $N_{bits} = 32$ if we use *float32*.

$$t_{tx} = \frac{N_{mel} \cdot N_{bits}}{br_{eff}} \quad [s]. \quad (13)$$

It is important to note that the bit-rate br_{eff} used in (13) is not equivalent the baud-rate br configured on the serial connection, *e.g.*, 230400 baud. We have to scale the configured baud-rate by 0.8, so $br_{eff} = br \cdot 0.8$. This scale factor comes from the serial mode configured. Since we are using the 8N1 mode, we transmit 1 start bit, 8 data bits, 0 parity bits and 1 stop bit. Resuming, we are actually transmitting 10 bits every 8 information bits. Effective bit-rates are shown in Table 2.

TABLE 2. Configured baud-rates Vs Effective bit-rates with 8N1 mode.

Configured baud-rate br	Effective bit-rate br_{eff}
57600	46080
115200	92160
230400	184320

B. A POSSIBLE PARAMETERS TUNING

As described above, a SAS designer has to tune only four parameters ($\delta_{overlap}$, N_{FFT}, N_{mel}, br) that are free variables. Typically, the overlap fraction $\delta_{overlap}$ is set to 0.5 in order to keep a good correlation between consequent windows. A possible set of parameters is the following:

$$\delta_{overlap} = 0.5, \quad N_{FFT} = 4096$$
$$N_{mel} = 128, \quad br = 230400.$$

Now, we prove that this set of parameters verifies both the design conditions. First, using (8), we compute the inter-arrival time between two consequent windows and we get

$$t_{buffering} = 46.44 \cdot 10^{-3} \quad [s].$$

Hence, we compute the time required for pre-processing operations using Formulas 9, 10, 11, 12, thus we obtain

$$t_{pre} = (2.11 + 0.1 + 1.3) \cdot 10^{-3} = 3.51 \cdot 10^{-3} \quad [s].$$

Then, using (13), we compute the time required to transmit N_{mel} coefficients as *float32* and we get

$$t_{tx} = 22.2 \cdot 10^{-3} \quad [s].$$

Immediately, we can see that the *Buffering-Processing condition* (Equation (7)) holds, since

$$\begin{aligned} t_{pre} + t_{tx} &= (3.51 + 22.2) \cdot 10^{-3} \\ &= 25.71 \cdot 10^{-3} \\ &< 46.44 \cdot 10^{-3} = t_{buffering}. \end{aligned}$$

Now, we have to verify if the *Real-Time condition* (Equation (6)) holds. The recording delay t_{rec} is simply the time required to record N_{FFT} samples at 44100 Hz and in this case it has value

$$t_{rec} = 92.9 \cdot 10^{-3} \quad [s].$$

Using this result we can compute the overall delay introduced by recording, pre-processing and transmission phases and we obtain

$$t_{rec} + t_{pre} + t_{tx} = 118.61 \cdot 10^{-3} < 150 \cdot 10^{-3} \quad [s].$$

The *Real-Time condition* holds and our set of parameters can be used to build a SAS. Other possible parameter combinations are shown in Table 3.

TABLE 3. Other possible parameter combinations assuming $\delta_{overlap} = 0.5$.

N_{FFT}	N_{mel}	br
4096	128, 64, 40	230400
4096	64, 40	115200
4096	40	57600
2048	64, 40	230400
2048	40	115200
1024	40	230400

C. COTA: CONFIGURATION OVER THE AIR

In the previous section, we have presented a framework that requires the tuning of four parameters to design a Smart Audio Sensor. Since SASs may be deployed in dangerous or unaccessible locations, device's parameters should be remotely set or changed from the other side of the Bluetooth link. These devices can be connected to an AGILE gateway and the gateway framework offers this functionality to remotely configure parameters. It is called Configuration Over The Air (COTA) and is implemented as a small micro-service that pushes parameters over the serial link used to communicate with a SAS. Parameters are sent to the remote device as a text string with the following format

$$\#\delta_{overlap}; N_{fft}; N_{mel}; br\$.$$

An example of a possible string is

$$\#0.5; 4096; 128; 230400\$.$$

The SAS replies with an acknowledgement message, which is *OK*, for a valid configuration or with *ERROR* if it rejects the configuration. After the acknowledgement message, the remote device reboots with the new configuration.

Moreover, the COTA module exposes an UI, accessible through the AGILE web-base UI, that allows users to configure parameters. The interface is really user-friendly and ready to use. The title shows the device family and ID, the the body contains four different fields to set the value of parameters. The UI controller is able to verify if the proposed configuration verifies our two design conditions. If one condition is not met, the interface prompts an error. Figure 3 shows an example of the COTA UI interface.

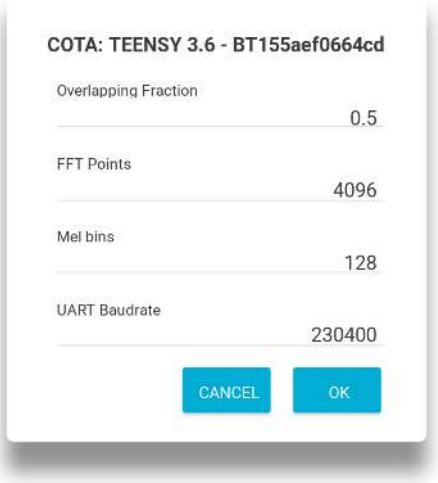

FIGURE 3. COTA UI.

V. PROOF OF CONCEPT

In order to demonstrate the applicability of the Smart Audio Sensors introduced in Section IV, we have designed and deployed an IoT Smart Office environment that uses a SAS and an AGILE gateway to detect audio anomalies. In this context, anomalies may be screams, door slams or every event that is not a normal keyboard typing, a call or a talking. Figure 4 provides a block diagram depicting how the different technological entities and comprising components interact with each other.

The SAS implements the pre-processing technique previously described and it is configured as follow:

$$\delta_{overlap} = 0.5, \quad N_{FFT} = 4096$$
$$N_{mel} = 128, \quad br = 230400.$$

The device has been deployed in a top corner of a rectangular room, which has a surface of 25 square-meters. The SAS records the audio-stream from an electret microphone, then pre-processes it by extracting mel coefficients and, finally, sends computed features to an AGILE gateway using the Bluetooth interface. On the other side of the radio link, the gateway runs a python-based micro-service that collects mel coefficients, through the serial interface, and feeds an Anomaly Detection algorithm to detect if an anomaly occurred or not in the current time frame. If yes, the micro-service sends a notification to the user showing an error in the logging console.

Our objective is twofold: we want to evaluate the impact over the CPU of Anomaly Detection algorithms running on the AGILE gateway and the delay from the anomalous event to the event detection. In this way, we have considered two different anomaly-detection algorithms among all the algorithms offered by the Scikit-Learn[3] toolbox: Elliptic Envelope (EE) [59] and Isolation Forest (IF) [60], [61].

The EE algorithm assumes that the training set has a Gaussian distribution, thus it models a robust covariance estimator over data. Giving the estimation of the inlier location and covariance, the algorithm uses the Mahalanobis distance to measure outlyingness of unseen data points.

The Isolation Forest algorithm exploits random forests to isolate outliers from inliers. IF randomly chooses a feature and then splits in a point between the maximum and the minimum values of the selected feature. The number of splitting operations that are required to isolate an instance is equal to the path from the root node of a tree to the leaf node. This is due to the recursive nature of splitting that can be modeled by a tree structure. We obtain a detection rule and a measure of normality by averaging path lengths over a forest of trees. This is due to an intrinsic property of IF that generates short paths for anomalies since they are isolated form inlier values.

[3]http://scikit-learn.org

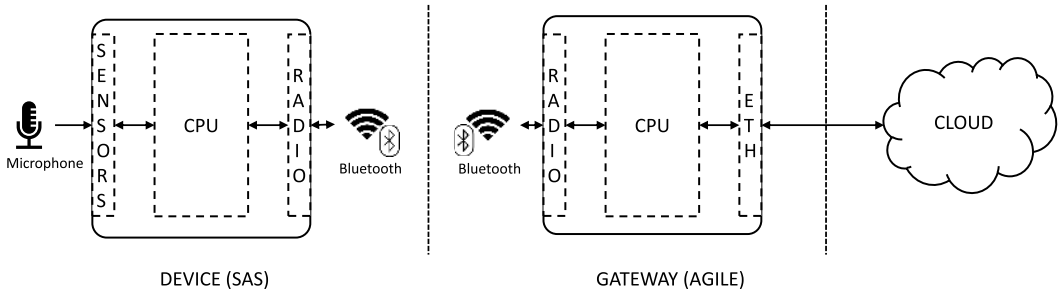

FIGURE 4. Block diagram of our technological framework, that is the deployed things-to-cloud continuum and the corresponding interactions with the IoT edges.

A. TRAINING

Since Anomaly Detection techniques belong to the Unsupervised Learning family of machine learning algorithms, a labeled dataset is not required to train our models. However, such algorithms require an unlabeled dataset that describes the normality condition of the office environment. In order to realize a reliable and real dataset, we conducted an audio recording campaign using a Teensy Audio Shield and a Teensy board that was programmed only as a recorder and stream forwarder over a UART interface. As stated in Section IV, the bandwidth available over a Bluetooth interface is not sufficient to transmit the audio stream recorded at 44.1 KHz with 16-bit resolution (Equation (1)), thus we configured the UART interface over USB that is able to reach higher baud-rate, up to 4.608 Mbaud. We recorded the office audio stream for 4 hours saving a WAV file every 60 seconds. The obtained dataset was split in two different subsets: 2 hours reserved for training set and 2 hours assigned to the test set. In particular, the training phase comprises two steps: feature extraction and the training of a model. Feature extraction calculates the mel coefficients starting from audio files using APIs offered by LibROSA [62], a python package for audio analysis. The sequence of operations is the same one explained in Figure 2. Anomaly Detection algorithms were implemented in Python3 using the Scikit-Learn framework, a powerful machine learning python toolbox. As we stated above, we selected two different algorithms among all the tens of available algorithms: Elliptic Envelope (EE) and Isolation Forest (IF). These algorithms require few hyper-parameters to configure the training process. Since we want to study the CPU load of the model inference, we used the default value of parameters for both of the algorithms: we used 0.1 as contamination fraction and 100 as the number of adopted trees in IF. Firstly, we have trained our algorithms on just the first half of the set. Then, we retrained them on the entire set.

B. EVALUATION

As we stated above, the evaluation of the proposed solution is conducted by considering the CPU load due to an Anomaly Detection algorithm running on an AGILE gateway and the *event-detection latency*, the delay between the anomalous event and the event detection.

Since, the AGILE gateway has been designed to run over a Raspberry Pi 3 computer, we conducted the load evaluation by running the AGILE gateway modules and the AD algorithm on the same device in two different fashions: embedded within a Docker container and executed as a native Python script. We recorded the user CPU load, since the module and AGILE run in the userspace, using the `top`[4] utility. Each experiment comprises three different phases: in the first 15 seconds, the system runs all the AGILE gateway modules and AD module without inferencing the feature stream, since it is not attached to the detection module. Then, we connect the feature stream to the AD algorithm for 30 seconds. Finally, we detach the data stream form the module and we continue to record the CPU load for other 15 seconds. We profiled the CPU load for both the AD algorithms. Moreover, in order to provide a baseline, we also repeated the same experiments over a x86 machine that run the AD algorithm and the AGILE gateway framework.

Figure 5 shows the CPU load due to the execution of the Isolation Forest algorithm. As we can see, the average CPU load on a Raspberry Pi increases from 5.2% to 30.6% when the module is embedded within a Docker container and from 6.8% to 30.4% when the module is *natively* executed. We can see that when the algorithm is running the Docker overhead is 0.2%. For IF, the Docker overhead on a x86 machine is the same (0.2%) since the CPU load is 26.1% (*dockerized*) and 25.9% (native), respectively. Figure 6 depicts the computing load due to the Elliptic Envelope algorithm. The graph shows that the Raspberry Pi CPU load passes from 5.6% to 7.9% when the algorithm is embedded within a container and from 5.6% to 7.3% when the script is natively executed. The load over a x86 machine is 1.18% when the module is containerized and 80.3% when the Python script is run natively. This behavior might be due to how the Python interpreter distributes the load over the CPU.

In general, except for the EE run as native script over a x86 machine, the CPU load of IF is higher then the load required to compute EE. This is due to the nature of the

[4]http://man7.org/linux/man-pages/man1/top.1.html

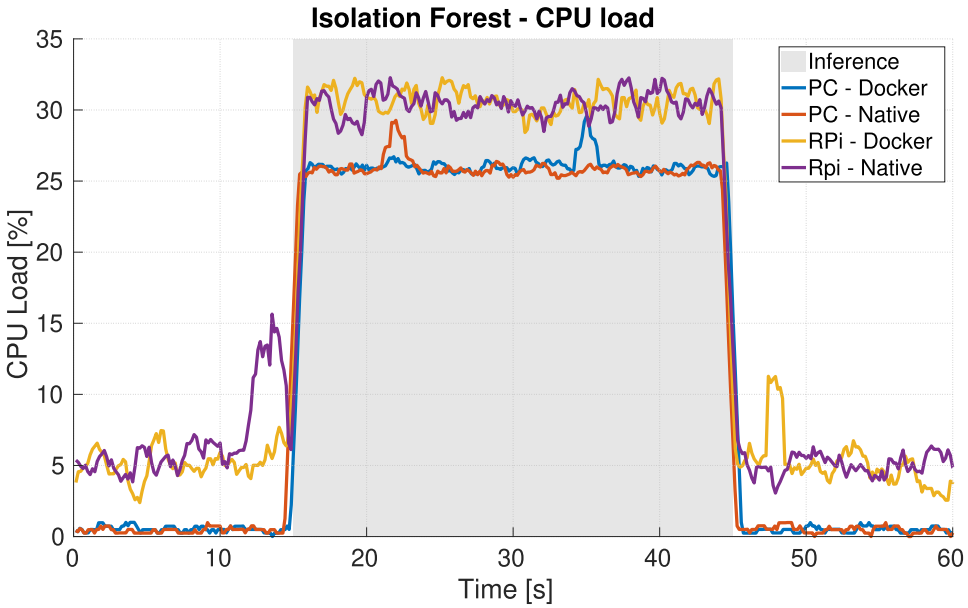

FIGURE 5. CPU load of Isolation Forest.

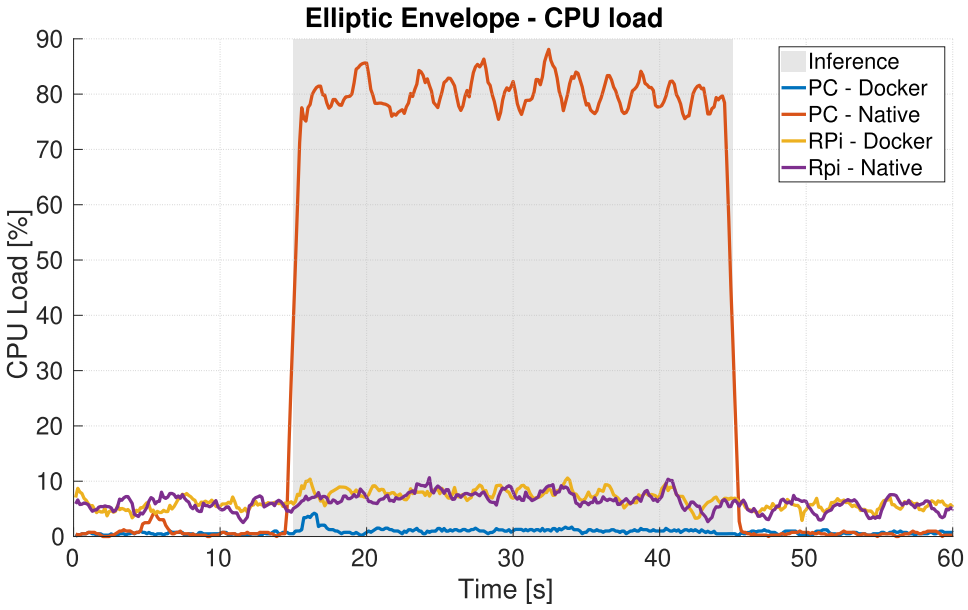

FIGURE 6. CPU load of Elliptic Envelope.

running algorithm. EE has to check if the input sample is inside or not the region defined by the learned decision function. If the sample is outside, the algorithm notifies an anomaly. The IF algorithm has to compute the output of all the trees present in the ensemble (*e.g.*100) and then it takes a decision. It is also possible to evince this behavior from the delay introduced by the algorithm to compute the output.

Table 4 shows that the IF is much slower than EE since IF has to compute the output of 100 trees.

Using the inference delays (t_i), we can compute the average overall latency ($t_{detection}$) from the event to the detection. (14), which describes this latency, is the sum of 4 contributions: t_{rec}, the time required to record an audio frame, t_{pre}, the delay introduced by Teensy to pre-process the audio frame, t_{tx}, the transmission delay, and t_i, the time required to infer the data in the AD model.

$$t_{detection} = t_{rec} + t_{pre} + t_{tx} + t_i. \tag{14}$$

Applying (14), we get $t_{detection} = 120.3[ms]$ when we infer with an EE model and $t_{detection} = 341.5[ms]$ when we adopt IF as AD algorithm. Since EE is much faster and also lighter than IF, we decide to adopt Elliptic Envelope as AD algorithm on our gateway.

Moreover, we conducted an evaluation campaign in order to profile the performance of the adopted algorithm in terms of F1 score [63] and Error Rate (ER) [64]. F1 is defined as

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

TABLE 4. Inference delays t_i.

Algorithm	RPi - Docker [ms]	RPi - Native [ms]	x86 - Docker [ms]	x86 - native [ms]
Elliptic Envelope	1.7	1.7	0.18	0.18
Isolation Forest	222.9	234.4	19.3	15.7

where P is the precision ($P = \frac{TP}{TP+FP}$, $TP =$ True Positive and $FP =$ False Positive) and R is the recall ($R = \frac{TP}{TP+FN}$, $TP =$ True Positive and $FN =$ False Negative). Error Rate (ER) is defined as

$$ER = \frac{S + D + I}{N}$$

where S is the number of substitution (correct time instant but wrong class), D is the number of deletions (event not detected but present in the ground truth), I is the number of insertions (event detected but not present in the ground truth), and N is the number of events in the ground truth. ER can have value grater that 1.

We tested the adopted algorithm using a self-made dataset generated using the mixture generation engine [65] developed for the DCASE2017 Challenge. We created a test dataset made by 501 examples: 167 with a gunshot, 167 with a glass-brake and 167 with a baby-cry. Each sample is 30 second long and was obtained adding the rare event (*e.g.*, a gunshot, a baby-cry or a glass-break) with probability one over the recorded office audio stream. We applied this dataset to our system and we computed the evaluation metrics using the *sed_eval* toolbox [66] using a time-collar of 500 ms. The EE algorithm trained with the 60 minutes dataset obtained a F1-score of 50.55% and an ER of 0.71. The same algorithm trained with the 120 minutes dataset got a F1-score of 50.42% and an ER of 0.71. We evaluated also the IF algorithm on the same datasets and we obtained a F1-score of 56.07% and an ER of 0.63 training with the 60 minutes dataset and a F1-score of 53.49% and an ER of 0.65 using the 120 minutes training dataset. Even if IF performs better than EE, we use EE since it is faster and lighter.

C. DEPLOYMENT ON AGILE

In the previous section, we discovered that the Elliptic Envelope is the best algorithm for our proof of concept, even if it has lower performance than the Isolation Forest algorithm. The Elliptic Envelope algorithm is 2 orders of magnitude faster and much lighter than IF, thus the system may be able to respond earlier and waste less computational resources. We deployed the EE algorithm and the model trained with 60 minutes of office audio recordings in the AGILE Anomaly Detection micro-service. Moreover, we binded a serial port to the module and we connected the Teensy board programmed as a SAS with the parameters presented before.

The system is able to receive the features computed by the embedded board and it is able to detect Audio Anomalies (*e.g.*, hand claps). Once an anomaly happens, the micro-service notified the user by logging detection messages in the terminal output. However, it is important to remember the nature of the incoming signal and the source of events. Since we are working with a transformed audio stream, we have many feature frames per second (21.5 frames per second) and the algorithm can erroneously detect (False Positive) an anomaly. In order to reduce isolated False Positives, possible solution was to apply a median filter, with window length N (N has to be odd), to the algorithm output. This filter considers the $N-1$ past samples and the current one, then it sorts all the predictions. If at position $\frac{N+1}{2}$ there is an anomaly, the filter declares that an anomaly occurred. We have to choose a small value of N (*e.g.* 5, 7,...) such that it does not introduce delays or hide anomalies. A good value of N is 5. Moreover, another aspect that affects the accuracy of the deployed system is the quality of the dataset. The training dataset should contain all the audio events that define the condition of *normality* in the considered environment. The anomaly model was trained using a real audio stream recorded in a office environment during the working time, thus the audio stream contained normal office sounds, *e.g.*, talking, phone ringing, typing and clicking.

This kind of scenarios suffers weaknesses related to the physical deployment of the SAS. Since the audio sensor embeds an electret microphone, which has a sort of directivity even if it is omni-directional, should be positioned in the direction of the audio source in order to have a better transduction from the audio signal to the electric signal with reduced reverberation. Moreover, the raw audio signal exiting from the microphone is really small and it requires a pre-amplification before the elaboration. The pre-amplification gain should be fine tuned in order to guarantee a good quality of the signal with an high Signal-to-Noise Ratio (SNR). Noise may be introduce by the microphone cable (depends on the cable quality and length) and by the amplifier itself. Another weakness is related to the power consumption since the SAS is always active and continuously streams mel features over the Bluetooth channel. It should be deployed closer to a power source like an electric plug or connected to a high-capacity battery. Future works will characterize the power consumption of this device.

In the considered PoC, we deployed just one SAS, which streams the extracted features. The proposed system is able to deal with multiple streams coming from different SASs. This is possible since the Anomaly Detection module supports more than one serial port binding. Using a multiplexing technique, it is possible to choose one of the stream and then infer on it. Another possibility, is to deploy one AD micro-service per device, but this approach is much more hungry of CPU power.

VI. CONCLUDING REMARKS

This section concludes our study: first, it summarizes the rationale behind the whole approach and the most interesting results obtained; then, it traces the most promising, future research directions.

A. CONCLUSIONS

In this paper, we have presented a class of smart objects called Smart Audio Sensors (SASs), that are audio devices able to autonomously record audio streams, locally perform computations on the recorded streams and send the results of these computations over a wireless link. At the beginning, we designed a device that was a simple audio recorder, with Bluetooth connectivity to transmit the raw audio stream. However, we had to deal with a Bluetooth link that does not have enough bandwidth to carry an audio stream sampled at 44.1KHz with 16-bit resolution. Therefore, we decided to migrate most of the computations directly into the audio device firmware. In particular, we implemented the entire software flow (Figure 2) to extract the mel-coefficients from the raw audio stream. We defined a mathematical framework to design parameters of the mel extraction software flow. Such framework is based on two conditions, namely the *Real-Time condition* (Equation (6)) and the *Buffering-Processing condition* (Equation (7)), that have to be satisfied by the flow parameters.

After that, we have integrated the SAS into the AGILE gateway ecosystem and we have developed an ad-hoc module, called Configuration Over The Air (COTA), to remotely configure and push the mel-flow parameters without direct intervention on the device firmware. This module has an user interface (UI) available within the AGILE UI, and it is also able to detect if the inserted configuration is valid or not.

In order to quantitatively assess the effectiveness of the proposed solution, we have deployed the proposed SAS in a real smart office scenario. This device is responsible of collecting and locally computing the mel-coefficients, while transmitting the computed features to an AGILE gateway instance. The latter simply receives the mel-coefficients from its wireless radio interface and run a micro-service that executes a purposely trained anomaly detection algorithm, in charge of detecting anomalous events in the received feature-transformed stream. With respect to the specific anomaly detection algorithms, we compared two different options, namely Elliptic Envelope and Isolation Forest, in terms of average computing latency and user CPU load at gateway level. We observed that, on the AGILE gateway instance, the best model is Elliptic Envelope, being it two orders of magnitude faster and one order of magnitude lighter than the Isolation Forest counterpart.

B. FUTURE WORKS

The proposed framework can be easily accommodated in different verticals, especially within the Industrial IoT (IIoT) domain. In this case, the sensing devices may be directly deployed on-machine, so as to locally perform their computations before sending data to the gateway, which, in this way, has only to execute the machine learning algorithm. This enables new vertical use-cases focused on diagnostics, prognostics and predictive maintenance, which reduce expenses and optimize the machine life-time.

However, industrial scenarios often require a higher number of sensing devices in order to effectively monitor different point of a plant. In these situations, our framework can easily scale up, providing a gateway that is able to manage large numbers of devices and related data streams. Since we keep deploying an AGILE gateway instance on a cheap Raspberry Pi 3 computer, we prefer not exceeding 80% of CPU load. Given this constraint and extrapolating the CPU load of the Elliptic Envelope container from Figure 6, we assert that more than 30 parallel Elliptic Envelope containers can be concurrently hosted on a single gateway. Then, given the harsher conditions of a typical industrial environment in terms of electromagnetic interferences, for our preliminary campaign we have changed the radio interface from Bluetooth to Wi-Fi as the latter, besides offering higher resistance against electromagnetic interferences, allows for higher data-rates. We conducted a wide performance test over the Wi-Fi channel using IPerf,[5] obtaining stable data-rates of 4Mbps, at UDP level. Considering that our gateway is able to support up to 31 devices, this means that every device can stream data up to a maximum data-rate of 129 Kbps.

Currently, we are applying this framework in a real industrial plant available at the Micro-Nano Facility (MNF) of Fondazione Bruno Kessler (FBK, Trento, Italy).[6] This facility allows researchers to study, develop and build micro devices by processing raw silica wafers in an extremely clean and fine-controlled environment, also known as *Clean Room*. The Clean Room is composed of different modules that keep the environment suitable for silica processing. One of the most important modules is the air treatment system that controls both the injection and expulsion of the air from the room, while keeping constant the air pressure and the relative humidity. These systems are extremely critical since a malfunction could have disastrous effects, for processes and equipments. Each system has two electric engines, while each engine is connected with a belt to a shaft that rotates a fan to push or extract the air from the Clean Room. Since these engines do not have on-board sensors, we are developing retrofitting kits to sense vibrations (using MEMS accelerometers) and temperature sensors. These kits sample the physical dimension, perform a time-frequency feature extraction and then stream data using a Wi-Fi radio chip to the AGILE gateway. More in detail, the kit computes 32 features, expressed as 32-bits float numbers, starting from a 4 seconds time window sampled at frequency of 1 KHz. It follows that each kit has to transmit 1024 bits every 4 seconds. Given that the maximum data-rate for each kit is as large as 129 Kbps,

[5] https://iperf.fr/
[6] https://mnf.fbk.eu/

then the effective bandwidth allocation for each kit is only 0.2%. Such bandwidth requirement allows for radio chips to stay in a low power consumption mode (sleep mode) for more than the 99% of the time, which increases the life-time of battery-powered kits.

At the time of writing, we have installed 2 kits for each machine (one on top of the engine chassis and one directly on the shaft bearing) for a total of 4 machines (thus, 8 kits in total). The gateway is able to manage all the data streams coming from these kits and, contextually, it is able to infer anomalies per stream. The inference output (*i.e.*, 0 if the anomaly is absent; 1 otherwise) is used to tag data streams that are sent to a remote cloud data broker. The remote system stores data and allows for historical analysis through a Grafana-based dashboard.[7] Last but not least, since the gateway supports the Configuration Over The Air (COTA), we are able to dynamically reconfigure devices based on the actual needs: if an anomaly occurs, we can easily reconfigure the device in order to have a finer analysis of the system. A the same time, we are also developing and testing a novel approach to automatically train and deploy anomaly detection models on the gateway. In this way, we can update models and reduce misdetections.

Another interesting research direction consists in developing a complete Data-Science framework running directly an the AGILE gateway. This would surely allow developers and data-scientists to directly create their own IoT applications on the edge, with a smoother learning curve.

ACKNOWLEDGMENT

The co-authors of this paper would like to thank Gabriele Maurina for his valuable support during the preliminary phase of this research.

REFERENCES

[1] K. Ashton, "That 'Internet of Things' thing," *RFID J.*, vol. 22, no. 7, pp. 97–114, Jun. 2009.
[2] C. G. Cassandras, "Smart cities as cyber-physical social systems," *Engineering*, vol. 2, no. 2, pp. 218–219, Jun. 2016.
[3] J. Poncela *et al.*, "Smart cities via data aggregation," *Wireless Pers. Commun.*, vol. 76, no. 2, pp. 149–168, May 2014.
[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.
[5] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Generat. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016.
[6] A. Bröring *et al.*, "Enabling IoT ecosystems through platform interoperability," *IEEE Softw.*, vol. 34, no. 1, pp. 54–61, Jan./Feb. 2017.
[7] B. Guo, D. Zhang, Z. Yu, Y. Liang, Z. Wang, and X. Zhou, "From the Internet of Things to embedded intelligence," *World Wide Web*, vol. 16, no. 4, pp. 399–420, 2013.
[8] J. Ma, C. Alippi, L. T. Yang, H. Ning, and K.-I. Wang, "Introduction to the IEEE CIS TC on smart world (SWTC) [society briefs]," *IEEE Comput. Intell. Mag.*, vol. 13, no. 1, pp. 7–9, Feb. 2018.
[9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.
[10] A. Marsico, A. Broglio, M. Vecchio, and F. M. Facca, "Learn by examples how to link the Internet of Things and the cloud computing paradigms: A fully working proof of concept," in *Proc. 3rd Int. Conf. Future Internet Things Cloud*, Aug. 2015, pp. 806–810.

[7] https://grafana.com/

[11] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 1st Quart., 2018.
[12] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
[13] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–43, Oct. 2017.
[14] M. Chiang, S. Ha, C.-L. I, F. Risso, and T. Zhang, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 18–20, Apr. 2017.
[15] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.
[16] B. Alturki, S. Reiff-Marganiec, and C. Perera, "A hybrid approach for data analytics for Internet of Things," in *Proc. 7th Int. Conf. Internet Things*, 2017, pp. 1–8.
[17] M. Vecchio, R. Giaffreda, and F. Marcelloni, "Adaptive lossless entropy compressors for tiny IoT devices," *IEEE Trans. Wireless Commun.*, vol. 13, no. 2, pp. 1088–1100, Feb. 2014.
[18] G. R. Kumar, M. Nimmala, N. Gugulothu, and S. R. Gali, "CLAPP: A self constructing feature clustering approach for anomaly detection," *Future Generat. Comput. Syst.*, vol. 74, pp. 417–429, Sep. 2017.
[19] A. R. Hilal, A. Sayedelahl, A. Tabibiazar, M. S. Kamel, and O. A. Basir, "A distributed sensor management for large-scale IoT indoor acoustic surveillance," *Future Generat. Comput. Syst.*, vol. 86, pp. 1170–1184, Sep. 2018.
[20] R. U. Islam, M. S. Hossain, and K. Andersson, "A novel anomaly detection algorithm for sensor data under uncertainty," *Soft Comput.*, vol. 22, no. 5, pp. 1623–1639, 2018.
[21] R. M. Alsina-Pagès, J. Navarro, F. Alías, and M. Hervás, "HomeSound: Real-time audio event detection based on high performance computing for behaviour and surveillance remote monitoring," *Sensors*, vol. 17, no. 4, p. 854, 2017.
[22] S. Agrawal and J. Agrawal, "Survey on anomaly detection using data mining techniques," *Procedia Comput. Sci.*, vol. 60, pp. 708–713, 2015.
[23] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data Mining Knowl. Discovery*, vol. 29, no. 3, pp. 626–688, 2015.
[24] J. P. P. dos Santos, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck, "Anomaly detection for smart city applications over 5G low power wide area networks," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, May 2018, pp. 1–9.
[25] D. Wang, D. S. Yeung, and E. C. C. Tsang, "Structured one-class classification," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 6, pp. 1283–1295, Dec. 2006.
[26] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLoS ONE*, vol. 11, no. 4, p. e0152173, 2016.
[27] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, Oct. 2016.
[28] X. Miao, Y. Liu, H. Zhao, and C. Li, "Distributed online one-class support vector machine for anomaly detection over networks," *IEEE Trans. Cybern.*, to be published.
[29] Z. He, X. Xu, J. Z. Huang, and S. Deng, "A frequent pattern discovery method for outlier detection," in *Proc. Int. Conf. Web-Age Inf. Manage.*, 2004, pp. 726–732.
[30] M. D. Ruiz, M. J. Martin-Bautista, D. Sánchez, M.-A. Vila, and M. Delgado, "Anomaly detection using fuzzy association rules," *Int. J. Electron. Secur. Digit. Forensics*, vol. 6, no. 1, pp. 25–37, 2014.
[31] J. Kevric, S. Jukic, and A. Subasi, "An effective combining classifier approach using tree algorithms for network intrusion detection," *Neural Comput. Appl.*, vol. 28, no. 1, pp. 1051–1058, 2017.
[32] L. Koc, T. A. Mazzuchi, and S. Sarkani, "A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier," *Expert Syst. Appl.*, vol. 39, no. 18, pp. 13492–13500, 2012.
[33] A. S. A. Aziz, S. E.-L. Hanafi, and A. E. Hassanien, "Comparison of classification techniques applied for network intrusion detection and classification," *J. Appl. Logic*, vol. 24, pp. 109–118, Nov. 2017.
[34] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 3, no. 1, pp. 1–130, 2009.

[35] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, "Fuzziness based semi-supervised learning approach for intrusion detection system," *Inf. Sci.*, vol. 378, pp. 484–497, Feb. 2017.

[36] H. Song, Z. Jiang, A. Men, and B. Yang, "A hybrid semi-supervised anomaly detection model for high-dimensional data," *Comput. Intell. Neurosci.*, vol. 2017, Nov. 2017, Art. no. 8501683.

[37] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[38] S. Trilles, Ò. Belmonte, S. Schade, and J. Huerta, "A domain-independent methodology to analyze IoT data streams in real-time. A proof of concept implementation for anomaly detection from environmental data," *Int. J. Digit. Earth*, vol. 10, no. 1, pp. 103–120, 2017.

[39] L. Lyu, J. Jin, S. Rajasegarar, X. He, and M. Palaniswami, "Fog-empowered anomaly detection in IoT using hyperellipsoidal clustering," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1174–1184, Oct. 2017.

[40] S. Rajasegarar *et al.*, "Ellipsoidal neighbourhood outlier factor for distributed anomaly detection in resource constrained networks," *Pattern Recognit.*, vol. 47, no. 9, pp. 2867–2879, 2014.

[41] P. Rathore, A. S. Rao, S. Rajasegarar, E. Vanz, J. Gubbi, and M. Palaniswami, "Real-time urban microclimate analysis using Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 500–511, Apr. 2018.

[42] *The Dynamap Project Web Site*. Accessed: Sep. 30, 2018. [Online]. Available: http://www.life-dynamap.eu/

[43] J. C. Socoró, F. Alías, and R. M. Alsina-Pagès, "An anomalous noise events detector for dynamic road traffic noise mapping in real-life urban and suburban environments," *Sensors*, vol. 17, no. 10, p. 2323, 2017.

[44] X. Xia, R. Togneri, F. Sohel, and D. Huang, "Random forest classification based acoustic event detection utilizing contextual-information and bottleneck features," *Pattern Recognit.*, vol. 81, pp. 1–13, Sep. 2018.

[45] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[46] *The Official Wiki for Embedded Tegra & Jetson TK1 Board*. Accessed: Sep. 30, 2018. [Online]. Available: https://elinux.org/Jetson_TK1

[47] *Raspberry Pi Web Site*. Accessed: Sep. 30, 2018. [Online]. Available: https://www.raspberrypi.org

[48] J. Ye, T. Kobayashi, and T. Higuchi, "Smart audio sensor on anomaly respiration detection using FLAC features," in *Proc. IEEE Sensors Appl. Symp.*, Feb. 2012, pp. 1–5.

[49] PJRC. *Teensy USB Development Board*. Accessed: Sep. 30, 2018. [Online]. Available: https://www.pjrc.com/teensy/

[50] NXP-Freescale. *SGTL5000 Datasheet*. Accessed: Sep. 30, 2018. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/SGTL5000.pdf

[51] *Openaudio Library for Teensy*. Accessed: Sep. 30, 2018. [Online]. Available: https://github.com/chipaudette/OpenAudio_ArduinoLibrary

[52] *The Agile Project Web Site*. Accessed: Sep. 30, 2018. [Online]. Available: http://agile-iot.eu/

[53] J. Thönes, "Microservices," *IEEE Softw.*, vol. 32, no. 1, p. 116, Jan./Feb. 2015.

[54] *Maker's Shield—Hardware Definition*. Accessed: Sep. 30, 2018. [Online]. Available: https://github.com/Agile-IoT/agile-makers-shield-hardware

[55] *Maker's Shield—Software Repository*. Accessed: Sep. 30, 2018. [Online]. Available: https://github.com/Agile-IoT/agile-makers-shield-software

[56] S. S. Stevens, J. Volkmann, and E. B. Newman, "A Scale for the measurement of the psychological magnitude pitch," *J. Acoust. Soc. Amer.*, vol. 8, no. 3, pp. 185–190, Jan. 1937.

[57] M. Slaney, "Auditory toolbox: A MATLAB toolbox for auditory modelling work," Interval Res. Corp., Palo Alto, CA, USA, Tech. Rep. 1998-10, 1998.

[58] S. Young *et al.*, *The HTK Book*, 3rd ed. Cambridge, U.K.: Cambridge Univ. Press, 2015.

[59] P. J. Rousseeuw and K. Van Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999.

[60] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.

[61] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discovery Data*, vol. 6, no. 1, pp. 1–39, Mar. 2012.

[62] *Librosa*. Accessed: Sep. 30, 2018. [Online]. Available: https://librosa.github.io/librosa/

[63] N. Chinchor, "MUC-4 evaluation metrics," in *Proc. 4th Conf. Message Understand. (MUC4)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 22–29.

[64] G. E. Poliner and D. P. W. Ellis, "A discriminative model for polyphonic piano transcription," *EURASIP J. Adv. Signal Process.*, vol. 2007, no. 1, 2006, Art. no. 048317.

[65] A. Mesaros *et al.*, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Proc. Detect. Classification Acoust. Scenes Events Workshop*, 2017, pp. 85–92.

[66] A. Mesaros, T. Heittola, and T. Virtanen, "Metrics for polyphonic sound event detection," *Appl. Sci.*, vol. 6, no. 6, p. 162, 2016.

MATTIA ANTONINI received the B.Sc. degree *(summa cum laude)* in computer, electronics and telecommunication engineering, and the M.Sc. degree *(summa cum laude)* in communication engineering from the University of Parma, Parma, Italy, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree in computer science with the University of Trento, Trento, Italy. He is a member of the OpenIoT Research Unit (Open Platforms and Enabling Technologies for the Internet of Things), FBK CREATE-NET, Trento.

His research interests include the design and development of open-source platforms for the Internet of Things, with the focus on machine learning-based applications in the fog and edge computing.

MASSIMO VECCHIO received the M.Sc. degree in information engineering *(magna cum laude)* from the University of Pisa, Pisa, Italy, and the Ph.D. degree in computer science and engineering (with Doctor Europaeus mention) from the IMT Institute for Advanced Studies, Lucca, Italy, in 2005 and 2009, respectively. Since 2015, he has been an Associate Professor with eCampus University. In 2017, he joined FBK CREATE-NET, Trento, Italy, to coordinate the research activities of the OpenIoT Research Unit. He is the Project Coordinator of AGILE (www.agile-iot.eu), a project co-founded by the Horizon 2020 programme of the European Union. His current research interests include computational intelligence and soft computing techniques, the Internet of Things Paradigm and effective engineering design and solutions for constrained and embedded devices. Regarding his most recent editorial activity, he is a member of the editorial board of the *Applied Soft Computing* Journal and of the newborn *IEEE Internet of Things Magazine*, besides being the managing editor of the IEEE IoT Newsletters.

FABIO ANTONELLI is currently the Head of the OpenIoT Research Unit (Open Platforms and Enabling Technologies for the Internet of Things), FBK CREATE-NET, Trento, Italy. He received the master's degree in electronics engineering from the Politecnico di Milano, Milan, Italy. He worked in the Telco Sector (within Alcatel and Telecom Italia Groups) for over 15 years, gaining extensive knowledge in experimental research, design, software development and management of ICT projects. More recently, in Fondazione Bruno Kessler, his interests have shifted on applied research in multimedia networking, architectures and platforms for the Internet of Things, where he has contributed and coordinated applied research activities in different European research projects in the Future Internet, multimedia, and Internet of Things domains.

PIETRO DUCANGE received the M.Sc. degree in computer engineering and the Ph.D. degree in information engineering from the University of Pisa, Pisa, Italy, in 2005 and 2009, respectively. He is currently an Associate Professor at eCampus University, Novedrate, Italy, where he is the Director of the SMART Engineering Solutions & Technologies Research Centre. His main research interests include evolutionary fuzzy systems, big data mining, social sensing, and sentiment analysis. Furthermore, he has been involved in a number of R&D projects in which data mining and computation intelligence algorithms have been successfully employed. He has co-authored over 40 papers in international journals and conference proceedings. He is a member of the Editorial Boards for *Soft Computing* and the *International Journal of Swarm Intelligence and Evolutionary Computation*.

CHARITH PERERA (M'14) is currently a Lecturer (Assistant Professor) at Cardiff University, U.K. He received the B.Sc. degree (Hons.) in computer science from Staffordshire University, U.K., and the MBA degree in business administration from the University of Wales, Cardiff, U.K., and the Ph.D. degree in computer science at The Australian National University, Canberra, Australia. He was with the Information Engineering Laboratory, ICT Centre, CSIRO. His research interests are Internet of Things, Sensing as a Service, Privacy, Middleware Platforms, and Sensing Infrastructure. He is a member of the ACM.