Solving Interoperability within the Smart Building: a Real Test-Bed

D. Costantino*, G. Malagnini*, F. Carrera*, A. Rizzardi†, P. Boccadoro*, S. Sicari†, L.A. Grieco*
*Dip. di Ingegneria Elettrica e dell'Informazione (DEI), Politecnico di Bari, Italy,
Email: d.costantino2@studenti.poliba.it,g.malagnini@studenti.poliba.it, f.carrera@studenti.poliba.it,
pietro.boccadoro@poliba.it, alfredo.grieco@poliba.it
†Dip. di Scienze Teoriche e Applicate (DISTA), Universita' degli Studi dell'Insubria, Italy,
Email:alessandra.rizzardi@uninsubria.it,sabrina.sicari@uninsubria.it

***Abstract*—Internet of Things (IoT) is a key pillar in various smart-* domains, including smart home, smart building, intelligent transportation systems, industrial Internet, to name a few. Its potential in terms of value creation is extraordinary, according to economic estimates from many market sectors. At the same time, there are some technical challenges to face, which could refrain IoT adoption. In this study, the attention is focused on interoperability and data quality issues from a pragmatic standpoint. In particular, a real smart building testbed has been set-up by integrating two different IoT platforms: TLSensing and NetwOrked Smart Object (NOS). The former has been developed to enable the quick deployment of a IPv6 over the TSCH mode of IEEE 802.15.4e (6tisch) IoT network and to expose the data gathered from sensors through a web interface. The latter is a middleware that handles heterogeneous data sources and offers a unifying view to users. The resulting IoT system is able to: (i) gather data from multiple sources, including multimedia ones; (ii) expose network resources to user through a common interface; (iii) enforce security and data quality checks. The performance of this system has been experimentally analyzed to demonstrate a satisfactory responsiveness and ability to meet the aforementioned requirements.**

***Index Terms*—Internet of Things, 802.15.4, Image processing, middleware**

I. Introduction

The IoT has been conceived for supporting a large population of constrained devices in several operative scenarios. It resulted in a computing paradigm that embraces tracking devices, embedded chips, and multimedia communications [1], thus including voice, video, and scalar sensed data. Information acquired from the environment in which IoT devices are located, must be efficiently collected and safely analyzed to provide meaningful services to users. Data can either be scalar (e.g., values related to temperature, light, and so on) or multimedia (e.g., video-surveillance applications) ones [1]. As a consequence, they should be properly treated in order to ensure reliability. One of the most challenging application fields for IoT is Smart Building, which leverages modeling [2], monitoring, management, and resource optimization functionalities to grant effectiveness to a Building Management Systems (BMS) [3] together with video surveillance and access control [4]. Smart Building scenarios sit on top of a highly heterogeneous technological background, made of protocols, facilities, and advanced features, that span from wide range to Low-power Lossy Network (LLN) [5]. Accordingly, interoperability is a key issue to properly integrate heterogeneous devices, protocols, and standards in the Smart Building [6]. Moreover, data quality is an essential feature to provide meaningful information to users from sensed data. This work tackles interoperability and data quality issues from a pragmatic standpoint by designing, developing, and experimentally testing a Smart Building system that integrates two main components: TLSensing [7] and NOS [8]. TLSensing is a development environment for continuous monitoring of either critical and non-critical parameters. It relies on the 6tisch technology [9], which standardizes different mechanisms for allocating link-layer resources. It efficiently trades off among latency, bandwidth, and power consumption, while enabling industrial-grade network performances. The IoT platform, named NOS, instead, has been conceived as a cross-domain middleware able to gather, temporarily store, process, and share heterogeneous kinds of IoT data. The resulting IoT system is able to: (i) retrieve data from multiple sources, including multimedia ones; (ii) expose network resources to user through a common interface; (iii) enforce data quality checks. The performance of this system has been experimentally analyzed to demonstrate a satisfactory responsiveness and ability to enforce the required checks. The reminder of this work is organized as follows: Section II presents the technological background of IoT architecture, including Internet Engineering Task Force (IETF) protocol stack, operating systems, and hardware platforms. In Section III the main functional aspects of the envisioned solution are presented. Section IV describes the experimental campaign and discusses obtained results. Section V concludes the work and proposes future works.

II. Technological background

This Section describes the details of the defined system architecture, mainly composed of two elements: (i) the TLSensing platform; (ii) an IoT middleware layer, named NOS.

A. TLSensing platform

The main platform adopted in this work is TLSensing [7], an IoT system composed by: (i) IoT devices, continuously gathering environmental data (i.e., temperature, relative humidity,

light, and acceleration) thanks to both sensing and communication capabilities; (ii) an Operating System implementing 6tisch protocol stack, namely OpenWSN; and (iii) a web server application collecting the aforementioned data and elaborating them to provide their availability. TLSensing was originally developed to as an advanced health management system in future aerial vehicles [7]. Given the offered capabilities and the low energy consumption, it can be also used in common environments, such as houses and public places. The 6tisch protocol stack includes suitable technologies for low-power and short-range wireless communication. It is composed by:

- IETF Constrained Application Protocol (CoAP): application protocol which easily translates to HyperText Transfer Protocol (HTTP) for integration with the web. It natively supports multicast and offers very low overhead and simplicity for constrained environments [10];
- IETF Routing Protocol for Low-power and Lossy networks (RPL): gradient-based routing protocol that can ease formation and management of multi-hop topologies based on short-range low-power links. It supports multiple roots and is highly flexible, as it manages topology based on parametric optimization functions [6];
- IETF IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN): adaptation layer to let Internet Protocol version 6 (IPv6) datagram to fit the small payload size (up to 127 bytes in IEEE 802.15.4) by means of advanced header compression techniques [6];
- IETF 6TiSCH Operation Sublayer (6TOP): adaptation layer that enables the integration among higher-layers protocol and the novel IEEE 802.15.4e standard, through management and data interfaces; it also organizes the transmission of a IPv6 packet over a Time Slotted Channel Hopping (TSCH) protocol [11] [12];
- Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 Medium Access Control (MAC): powerful MAC layer based on the TSCH protocol, which ensures reliability and energy efficiency in challenging wireless Personal Area Network (PAN) [13];
- IEEE 802.15.4 Physical (PHY): low-power physical layer based on the Direct Sequence Spread Spectrum (DSSS) modulation scheme and operating at the 2.4 GHz of the Industrial, Scientific and Medical (ISM) band [13].

At the time of this writing, TLSensing platform supports OpenMote[1] and Telos Rev B[2], also known as TelosB, which are equipped with a first-generation Texas Instruments MSP430 MicroController Unit (MCU) and an IEEE802.15.4-compliant CC2420 radio transceiver. It also features temperature, humidity and light sensors. OpenMote, instead, is based on the TI CC2538 System on Chip (SoC). It uses an ARM Cortex-M3 MCU (32-bit, 32 MHz) with 512 kB of Flash Memory and 32 kB of RAM. Jointly adopting the OpenBase and the OpenMote, it is possible to sense temperature, humidity, light, and acceleration. TLSensing can easily extend its hardware support to other hardware platform, e.g., Zolertia Z1[3].

OpenWSN [14] is an open source operating system that implements the whole 6tisch protocol stack. It is mainly composed of two parts: (i) the firmware, running on top of each device; (ii) the software, executed by a dedicated device acting like a gateway while monitoring network activities. OpenWSN is used to both setup and monitor a Wireless Sensors Network (WSN) [15], exploiting the capability of a special-node, namely the PAN coordinator, acting as a border router between the IoT domain and the external Local Area Network (LAN). The coordinator directly interacts with the software part of OpenWSN to perform software monitoring operations and controlling data acquisition processes by adopting a client-server communication paradigm. In particular, the software acts as a client, periodically generating requests, sent towards the remote constrained devices. An application server runs on each mote (i.e., the constrained IoT devices) in order to: (i) process each requests; (ii) query on-board sensors; (iii) elaborate answers; (iv) deliver measurement data back to the coordinator. Messages are encapsulated into CoAP messages. It is worth noting that the PAN coordinator acts as a relay between the client and the server. Data retrieved by the monitoring software finally reach the server application that will make them available through a web-based/user-friendly interface.

B. NOS middleware

One of the main goals of the present work is to establish a communication between TLSensing and a Digicom IP Camera 400 HD[4]. The camera is able to capture videos and photos in any light condition; it provides an 10/100 Ethernet or Wi-Fi connection and supports a lot of protocols. In particular, Transmission Control Protocol/Internet Protocol (TCP/IP) and HTTP will be used in this work. It is evident that a direct communication among such entities is not possible, due to protocols incompatibilities; for this reason, NOS architecture has been introduced [8]. NOS is an IoT cross-domain middleware which interacts with multiple IoT-nodes over the HTTP protocol, analyzes and shares, in the form of services via Message Queue Telemetry Transport (MQTT), the coming data providing to the final user a clear, safe and filtered result. As it is shown in Figure 1 [8], NOS collects data provided by heterogeneous sources, which can be registered or not and temporarily stores them in the storage collection, named *Raw Data*. After this, NOS system filters the collected records according to the *Analysis* and *Data Annotation* layers, in order to obtain a uniform representation of the data themselves. During this phase, data records belonging to registered source are managed in a different way than the non-registered ones; in fact, they are characterized by different security communication schema, besides dedicated quality levels. As regards security [16], if the data source is a registered one, the

[1] http://www.openmote.com/

[2] https://telosbsensors.wordpress.com/

[3] http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf

[4] http://www.digicom.it/digisit/prodotti.nsf/itprodottiidx/IPCamera400HD

information access requires the authentication of the source and the decryption of the data; then, a score is assigned to the following parameters: authentication, confidentiality, integrity, and privacy. For non-registered sources, no information may be available, therefore they will have low value security and privacy scores. As regards data quality assessment, a score is assigned to timeliness, completeness, accuracy and precision levels [17][18]. In the end, final data (enriched with quality and security scores) are temporarily stored in the collection, named *Processed Data*, until they are shared with interested users by means of an authenticated publish&subscribe system, based on MQTT protocol [19]. NOS has been realized by means of Node.JS platform[5] and MongoDB[6]. The latter is a non-relational database (e.g., NoSQL), that is used for storage management. The modules reported in Figure 1 interact with each other using RESTful services. Finally, data are handled as document in JavaScript Object Notation (JSON), a lightweight data-interchange format. It has been chosen since it is both easy for humans to be read and wrote and for machines to be generated and/or parsed.

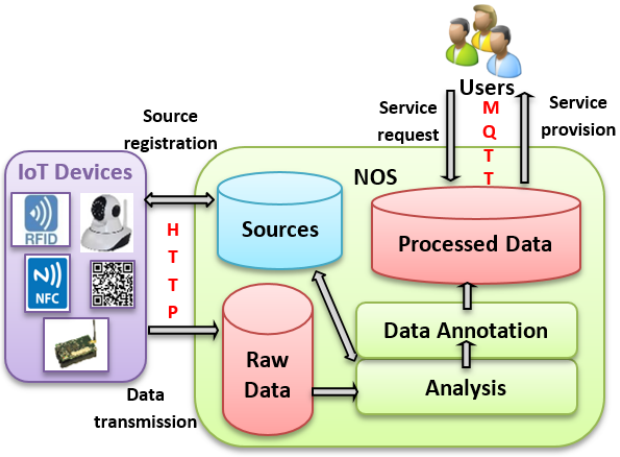

Fig. 1. NOS architecture

III. System Architecture and Interoperability

The overall architecture conceived in this work is depicted in Figure 2. The core entity is the TLSensing platform, which periodically gathers and processes data from the connected IoT devices. Then, NOS middleware has been connected to the TLSensing platform, in order to communicate with the IP Camera. In particular, the IP Camera communicates with NOS via Ethernet standard. NOS uses HTTP GET to require the images from the IP Camera (identified with "1" in Figure 2) and saves them in MongoDB following a specific format. After the analysis, the web app can require, through HTTP requests, the data analyzed (identified with "2" in Figure 2). In the next section, will be shown how images are managed by NOS.

[5]https://nodejs.org/it/
[6]https://www.mongodb.com/

A. Image Processing

To provide reliable and accurate results to TLSensing, NOS middleware handles snapshot from the IP Camera and enriches them with quality and security parameters, as introduced in Section II. Originally, NOS was conceived to manage scalar data. Nevertheless, to reach Smart Building control, complex data elaboration and processing must be proven. Therefore, image elaboration process has been modified to fit the application needs and properly manage multimedia data (e.g., images). The first action point was related to the creation of an HTTP GET request to the IP Camera for acquiring the latest snapshot captured. This action is repeated with a fixed pace that can be modified before starting the image acquisition process (as discussed in Section IV). The acquired image comes in the Joint Photographic Experts Group (JPEG) format. Once the HTTP response is received, the snapshot is extracted from the HTTP body and stored in a JSON variable with some additional information, such as timestamp, content-length, content-type, source identification, and a "processed" field equal to zero to denote that the image has not been analyzed yet; finally, the JSON instance is stored in the *Raw Data* collection.

Algorithm 1 simply explains how the image is analyzed and evaluated. At the beginning, a query is sent to MongoDB database: if there is a record with "processed" field equal to zero, then NOS gets it and saves it in a JSON variable. At this point, NOS verifies if the field denoting the identifier of the source matches with one of the records stored in the *Sources* collection (such a collection contains the information about registered sources): if it does, then the source is registered, so the analysis continues; if not, data come from a non-registered source. Registered data are further distinguished on the basis of the "content-type" field. In this way, both numerical data coming from environmental sensor and image data can be filtered with proper algorithms. Some basic security and quality image controls have been implemented, which could be extended and enhanced in future works. Security controls are named "integrity" [0,1] and "robustness" [0,1,2]; they verify, respectively, the correct resolution of the image and if the associated timestamp comply with the interval time said before. Quality controls were limited by the compressed nature of the JPEG format; "information" [0,1] verifies if the current image differs or not from the last acquired one; "detail" [0,1,2], instead, classifies images on the basis of its size. How such values are assigned to the "integrity", "robustness", "information", and "detail" fields will be clarified in Section IV. Finally, the system stores the final JSON record, including the scores just mentioned, in the final *Processed Data* collection. Such a process is repeated until there are unprocessed records in the *Raw Data* collection.

The results of NOS analysis are made available to the users on an HyperText Markup Language (HTML) page linked to TLSensing. Users can require the last image (and all its associated details) collected from the *Processed Data* collection or make a query to visualize filtered images according to the

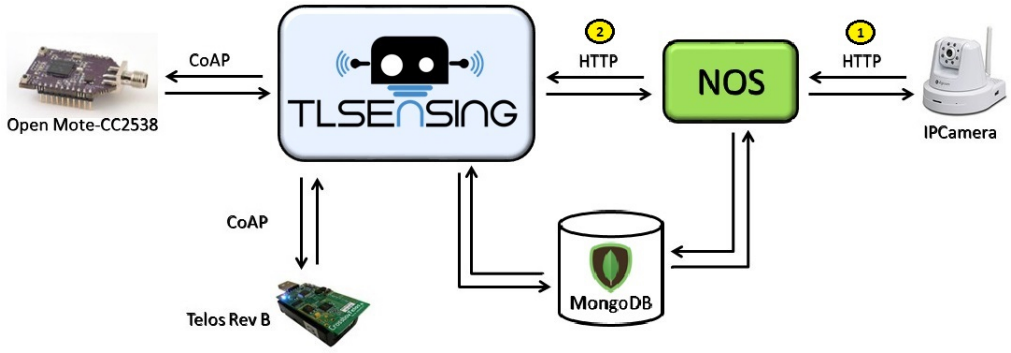

Fig. 2. Overview of system architecture and interactions between main components

```
1  if rawdata.find(processed:0) > 0 then
2      rawdata.GetRecord();
3      else if Record.type=='Image' then
4          else if SorcesAnalysis() Registered? then
5              ImageSecurityVerification();
6          else
7              ImageAnalysis();
8          end
9      else if SorcesAnalysis()=Registered? then
10         SensorDataSecurityVerification();
11     else
12         SensorDataAnalysis();
13     end
14     data.SaveRecord();
15 end
16 wait;
```

Algorithm 1: Pseudo-code associated to image processing.

scores assigned during the analysis. Every request is performed by means of a HTTP GET method to NOS server, which makes itself queries to MongoDB database.

IV. Experimental setup and performance evaluation

The envisioned experimental setup is hereby described. It is also clarified how the Smart Buildings requirements have been addressed, along with a detailed analysis of some key performance indices. TLSensing has been installed on the Pandaboard ES[7], a versatile low-power and low-cost single-board computer. It integrates a dual-core 1.2 GHz ARM Cortex-A9 MPCore Central Processing Unit (CPU), 384 MHz PowerVr SGX540 GPU, IVA3 multimedia hardware accelerator with a programmable DSP, 1 GB of DDR2 SDRAM, as well as an SDCard slot, 10/100 Ethernet, Wi-Fi, and Bluetooth interfaces, output video signal via DVI and HDMI, and two USB ports. Along with TLSensing and NOS platforms, two software tools have been installed, namely Node.JS (v5.0.0) and MongoDB (v2.4.9). Both the IP Camera and the Pandaboard have been connected in order to create a local network. The described IoT platform has been deployed in a research laboratory, periodically monitoring both access control information and environmental parameters, such as light and temperature conditions. To properly address the Smart Building characteristics, the available features have been stress-tested, with specific reference to image acquisition. This is supposed to be the heaviest among the tasks executed by the whole system. Therefore, the overall behavior of the envisioned solution has been tested in order to analyze the images acquired, with specific requests, during the observation period of time, equal to 30 minutes. The analysis has been performed with the NOS system acquiring frames with three different sampling periods: 15, 30, and 60 seconds. The NOS system performs HTTP requests to the IP Camera to get the images. Since the request-response mechanism guarantees the camera to synchronously answer, the system receives the frames and store them. Data are communicated in JSON format. In particular, the JSON variable, in correspondence to the field named *ImageData*, contains the sequence of bytes that describe the image received by the IP Camera, in JPEG format; the other fields contain the attributes associated with the sample image. The used structure is shown in Figure 3. The obtained output is shown through the HTML page depicted in Figure 4. As for the implemented

```
var doc = {
          "id_rawdata" : rawdata.getPK(),
          "sourceid" : rawdata.getSourceid(),
          "type" : rawdata.getType(),
          "timestamp" : rawdata.getTimestamp1(),
          "content-length" : rawdata.getContentLength(),
          "datatype" : rawdata.getDatatype(),
          "datacontent" : rawdata.getDatacontent(),
          "processed" : 0 ,
          "ImageData" : rawdata.getImageData()
                    };
```

Fig. 3. JSON variable describing a sample image

image controls, NOS operations have been evaluated during the setting of *detail*, *information*, *integrity*, and *robustness* fields. More in detail:

[7]https://www.cs.utexas.edu/ simon/378/resources/PandaBoardES.pdf

Fig. 4. TLSensing platform web page showing values of the analyzed images coming from our research laboratory.

- As for *detail* field, images have been captured in different brightness conditions. It has been seen how dark images, with a homogeneous pixel distribution and no details, are characterized by smaller dimensions, compared with a threshold *v1*. For this reason, the system classifies the images with a *detail* value equal to 0. Instead, blurred images, with few details, are classified with the *detail* value equal to 1, as they have sizes between *v1* and *v2*. In the case of very detailed images, with a dimension larger than *v2*, the *detail* field results to be equal to 2. Figure 5 reports the average dimension of 10 samples series, classified according to the three levels of detail. Thresholds variations, *v1* and *v2*, are related to the image resolution; in fact, with a resolution of 480x640 pixels, the thresholds are set to *v1*=8 kilobyte and *v2*=20 kilobyte;
- The *information* field depends on variations. In particular, when acquiring a series of consecutive samples (e.g., 10) of the same image, if each sample demonstrates a 1% variation compared to the previous one, then, the *information* field is set to 1 for the first image and 0 for the following ones;
- The *integrity* field has also been validated by replacing in the HTTP GET request the IP Camera address with the address of a generic image from the web. This image has a different resolution compared to the ones obtained by the IP Camera. During the analysis, the NOS system sets integrity to 0 (i.e., in this way, it has been simulated the presence of a non-registered source, since the IP Camera is considered a registered one); therefore, the images legitimately received by the known IP Camera have the *integrity* field set to 1;
- The *robustness* field is set on the basis of the comparison between the timestamps of the currently received image and the previous one. In detail, if the differences between the two timestamps is equal to the sampling time period, then *robustness* is set to 2. Instead, if such a difference is a positive value (e.g., due to a connection delay), then *robustness* is set to 1. Lastly, if such a difference is a negative value, then *robustness* is set to 0.

However, it was known that some error or violation attempts would affect the information quality control, because such an implemented feature does not go deeply within the image content, mainly in terms of completeness, accuracy and precision; anyway, this do not represent a main goal of the presented work, so it is left as a future extension. Afterwards, it has been evaluated how the system works setting different time intervals between two consecutive images' acquisition. Figure 6 shows how the processing time of a single image varies during three test sessions, each one of 30 minutes. The acquisition time for the three sessions equals 15, 30, and 60 seconds, respectively. As a consequence, the number of acquired samples is different for each test (for instance, 120, 60, and 30 samples, respectively).

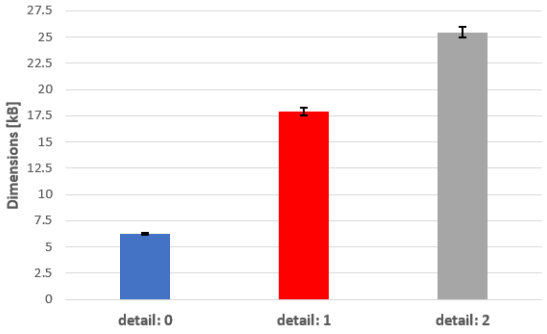

Fig. 5. Average dimension of samples captured on three images with different detail levels

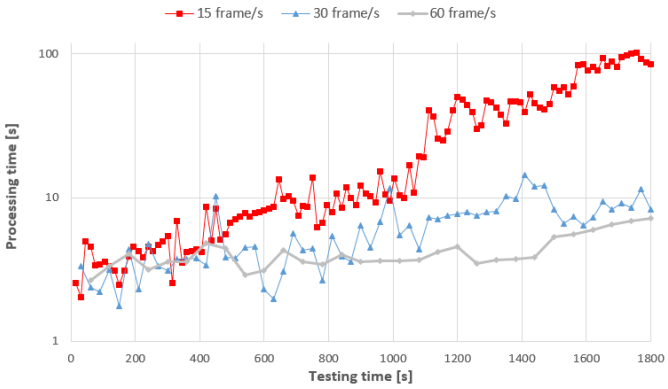

Fig. 6. Processing time of samples acquired during three test sessions, 30 minutes long each

The experimental campaign highlighted a general increase of the processing time in every test session. In particular, in the case of the 15 seconds test, after that period of time, the processing time interval exceeds the acquisition one; the more frequent the acquisition, the longer the processing queue. As a matter of fact, this results in a bottleneck that could be due to the limited computational capabilities of the PandaBoard, stressed by recurrent queries to MongoDB. Moreover it has been observed that the number of records in the database continuously increase and this may also affect the processing time with a higher acquisition period. When

comparing the experimental evidences, it clearly results that after 1000 seconds (about 16 minutes), the system slows down, which results in a longer processing time. In order to effectively test the presented system's suitability in a Smart Building context, the same experiments have been repeated with different time periods. By comparing system behaviors, it clearly emerges that the system no longer suffers of such a congestion phenomenon. The general system slowdown is caused by the higher data dimension, with respect to the original one. A possible solution could be the exploitation of a more powerful motherboard. Nevertheless, this would only be a shot-term answer to the questions, as it will only delay the increasing in processing time. To definitely solve the problem, a better approach could be a software modification, specifically aimed at improving interaction between NOS and the database (i.e., indices implementation in MongoDB, or efficient queries usage). Note that NOS platform has been conceived as hardware-independent, able to run on any gateway or node. Hence, specific application context and requirements could lead to the appropriate device/hardware combination. As a further consideration, performances have also been evaluated in terms of RAM occupancy. Compared with the total amount available on the board, values reached about 60%, 26%, and 24% for the three cases, respectively. Such results suggest that usability of this version of the presented solution in a hard real-time or near real-time context is particularly limited. Nonetheless, the system still remains an interesting candidate for BMSs support and access control.

V. Conclusions and future works

This work focused on communication among heterogeneous IoT technologies and platforms in a Smart Building context. In particular, several hardware and software IoT solutions have been glued together to provide a reliable monitoring system, handling both scalar and media data belonging to either Internet Protocol version 4 (IPv4) and IPv6 realms. To reach this goal, an experimental testbed allowing interoperability among different data sources and IoT platforms has been realized. The proposed solution has been deployed in one of our research laboratories to continuously monitor environmental conditions and access control. The testbed has been practically validated to be robust against abnormal behaviors. Despite the noticeable results, the system still shows some margins of enhancement. In particular, under certain conditions, the processing time increases over acceptable thresholds, preventing its usability in real-time contexts. The outcome is motivated by the involved technologies, which could be replaced in the next future with more powerful ones (i.e., Raspberry Pi 3). Another future work concerns the integration, and/or the comparison, of the proposed system with other existing IoT middlewares, as to reveal advantages and drawbacks. To prove scalability, it would also be of interest to deploy the presented solution in a large-scale environment. Finally, the presented solution is targeted to Smart Building applications. Nevertheless, without serious restyling activities of the algorithms and/or the processing tasks, the proposed solution can be easily exploited and extended to other domains that require the management of heterogeneous data provided by IoT-enabled technologies.

VI. Acknowledgment

This work has been partially funded by the research project E-SHELF (by the Apulia Region-Italy, code: OSW3NO1) and the FFABR fund (by the Italian MIUR).

References

[1] Q. Wang, Y. Zhao, W. Wang, D. Minoli, K. Sohraby, H. Zhu, and B. Occhiogrosso, "Multimedia iot systems and applications," in *2017 Global Internet of Things Summit (GIoTS)*, 2017, pp. 1–6.

[2] E. Patti, A. Acquaviva, F. Abate, A. Osello, A. Cocuccio, M. Jahn, M. Jentsch, and E. Macii, "Middleware services for network interoperability in smart energy efficient buildings," in *Design, Automation Test in Europe Conference Exhibition*, 2012, pp. 338–339.

[3] X. Qu, B. Duan, Y. Yan, Y. Zhong, and Q. Yin, "Information exchange interface between smart building and utility based on ontology mapping," in *IECON*, 2017, pp. 286–290.

[4] D. Minoli, K. Sohraby, and B. Occhiogrosso, "Iot considerations, requirements, and architectures for smart buildings. energy optimization and next-generation building management systems," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 269–283, 2017.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[6] O. Hersent, D. Boswarthick, and O. Elloumi, *The internet of things: Key applications and protocols*. John Wiley & Sons, 2011.

[7] S. Sciancalepore, G. Piro, F. Bruni, E. Nasca, G. Boggia, and L. A. Grieco, "An iot-based measurement system for aerial vehicles," in *IEEE Metrology for Aerospace*, 2015, pp. 245–250.

[8] S.Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini, "A secure and quality-aware prototypical architecture for the Internet of Things," *Information Systems*, vol. 58, pp. 43–55, 2016.

[9] M. R. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel, *6TiSCH Wireless Industrial Networks: Determinism Meets IPv6*. Springer, 2014, pp. 111–141.

[10] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[11] Q. Wang and X. Vilajosana, "6tisch operation sublayer (6top) interface," 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-6top-interface-04

[12] M. R. Palattella, T. Watteyne, Q. Wang, K. Muraoka, N. Accettura, D. Dujovne, L. A. Grieco, and T. Engel, "On-the-fly bandwidth reservation for 6tisch wireless industrial networks," *IEEE Sensors Journal*, vol. 16, no. 2, pp. 550–560, 2016.

[13] "Ieee standard for low-rate wireless networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, 2016.

[14] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "Openwsn: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[15] D. Vasiljevi and G. Gardaevi, "Performance evaluation of openwsn operating system on open mote platform for industrial iot applications," in *2016 International Symposium on Industrial Electronics (INDEL)*, 2016, pp. 1–6.

[16] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[17] C. Cappiello and F. A. Schreiber, "Quality- and energy-aware data compression by aggregation in wsn data streams," in *2009 IEEE International Conference on Pervasive Computing and Communications*, 2009, pp. 1–6.

[18] A. Klein and W. Lehner, "Representing data quality in sensor data streaming environments," *J. Data and Information Quality*, vol. 1, no. 2, pp. 10:1–10:28, 2009.

[19] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini, "Aups: An open source authenticated publish/subscribe system for the Internet of Things," *Information Systems*, vol. 62, pp. 29–41, 2016.