LEGIoT: a Lightweight Edge Gateway for the Internet of Things

Roberto Morabito, Riccardo Petrolo, Valeria Loscrì, Nathalie Mitton

LEGIoT:
a Lightweight Edge Gateway for the Internet of Things

Roberto Morabito[a,*], Riccardo Petrolo[b], Valeria Loscri[c], Nathalie Mitton[c]

[a] *Ericsson Research, Jorvas, Finland.*
[b] *Rice University, Houston, Texas, USA.*
[c] *Inria Lille - Nord Europe, France.*

Abstract

The stringent latency together with the higher bandwidth requirements of current Internet of Things (IoT) applications, are leading to the definition of new network-infrastructures, such as Multi-access Edge Computing (MEC). This emerging paradigm encompasses the execution of many network tasks at the *edge* and in particular on constrained gateways that have also to deal with the plethora of disparate technologies available in the IoT landscape.

To cope with these issues, we introduce a Lightweight Edge Gateway for the Internet of Things (LEGIoT) architecture. It relies on the modular characteristic of microservices and the flexibility of lightweight virtualization technologies to guarantee an extensible and flexible solution. In particular, by combining the implementation of specific frameworks and the benefits of container-based virtualization, our proposal enhances the suitability of edge gateways towards a wide variety of IoT protocols/applications (for both downlink and uplink) enabling an optimized resource management and taking into account requirements such as energy efficiency, multi-tenancy, and interoperability.

LEGIoT is designed to be hardware agnostic and its implementation has been tested within a real sensor network. Achieved results demonstrate its scalability and suitability to host different applications meant to provide a wide range of IoT services.

Keywords: Internet of Things, Edge Computing, Gateway, Virtualization, Container, Sensor Network

*Principal corresponding
Email address: roberto.morabito@ericsson.com (Roberto Morabito)

1. Introduction

Thanks to its ability to connect all the *objects* around us to the Internet with minimum human intervention, the Internet of Things (IoT) is leading a revolution in different domains of our everyday life (e.g., healthcare, transportation, agriculture, vehicles, etc.) [1]. On the other hand, these tremendous potentialities push on the proliferation and evolution of different technologies, from ubiquitous and pervasive computing, to embedded devices, communication standards, sensor networks, Internet protocols, and applications [2]. As a result, the IoT landscape is currently fragmented, i.e., characterized by different devices and protocols, while their integration is marked as crucial for the development of newer, exhaustive, and accurate use-cases [3, 4, 5]. Beside heterogeneity, scalability is another limiting factor for current IoT deployments, with a number between 26 and 50 billion of devices expected to be connected to the Internet by 2020 [6].

To cope with these issues, industry players together with the research community have constantly sought new solutions for supporting efficient deployments. For example, over the last years, *Cloud* has played a crucial role to enhance and extend IoT networks capabilities. Computation offloading, service management, data storage, monitoring systems, and offline analysis of large amounts of data represent only a limited subset of all the operations that traditional sensor networks have *outsourced* to the cloud. However, IoT cloud-based architectures are currently facing several challenges on meeting the increasing demand of high performance. In particular, relying only on cloud infrastructures can become a bottleneck — in terms of both latency and bandwidth requirements — for applications requiring real-time operations and mission critical communications.

In this respect, the Multi-access Edge Computing paradigm (MEC) [7] defines an emerging network infrastructure with the objective to deliver low-latency, bandwidth-efficient, and resilient end-user services. It is worth clarifying that this approach does not intended to replace cloud-based infrastructures, but rather, it aims to increase computation, networking, and storage resources at the network edge, by means of intermediate layers placed between end-user devices and Cloud. As result, it is important to design the efficient entities placed at the network edge, which act as interface between cloud services and IoT devices. The use of *middleware* in this context has

already been widely demonstrated. In particular, Cloud-to-sensors bridging functionalities are normally executed by IoT gateways that, in most cases, are designed to provide only connectivity, routing-forwarding functionalities, and other minor features.

Research Challenges. We have afore-discussed how the requirements for emerging edge-oriented architectures are pushing network infrastructure designers towards directions where IoT edge gateways are required to embed more complex functionalities, encompassing the capacity to handle a variety of services. Clearly, the heterogeneity of the different instances and applications generates further challenges. From an architectural point of view, IoT edge gateways have to be designed to:

(i) *bridge different networking technologies* by interacting with multiple cloud-based services and heterogeneous sensor devices;
(ii) *ensure a high flexibility* while integrating newer applications and at the same time preserving services' isolation;
(iii) *exploit a common resource abstraction* that also guarantee to use the same gateway software on top of different hardware platforms;
(iv) *ensure a virtuous trade-off* between design requirements, specific performance targets, and applications manageability.

Contributions. With the objective to tackle the challenges introduced by emerging Edge-IoT scenarios, and in order to cope with the limitations of current IoT gateways implementations, in this paper we propose LEGIoT: a Lightweight Edge Gateway for the Internet of Things architecture.

LEGIoT is characterized by two different modules: the *Northbound* in charge of the communication with the Internet and the *Southbound* that manages the exchanges with sensors. All the components of LEGIoT, are virtualized by means of Docker[1] containers, which introduce fast building process, instantiation, high density of application/services, and isolation between the different instances. The high flexibility introduced by our design enables an easy integration and support towards the deployments of new services, which are continuously proposed in the IoT landscape. Moreover, thanks to its isolation features, LEGIoT is well suited for multi-tenant scenarios, such as smart buildings. Tenants can indeed deploy and run their own applications/services without interfering with the rest of the system.

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

LEGIoT aims to be the first IoT edge-gateway architecture capable to simultaneously achieve:

(i) *interoperability*, thanks to the implemented orchestrator module, different heterogeneous sensor networks can be abstracted and managed;
(ii) *high energy-efficiency* by implementing different mechanisms for the allocation/deallocation of service;
(iii) *flexibility in managing different services*, thanks to the implementation of a bridging interface that allows a straightforward communication between downlink and uplink protocols;
(iv) *fast allocation*, *service isolation*, *backup capabilities*, and *multi-tenancy*, by exploiting the benefits derived on the use of lightweight virtualization technologies.

LEGIoT is implemented on real hardware, using four of the most popular IoT Single-Board Computers (SBC). To study its performance while interacting with real sensor nodes, we use one of the FIT IoT-lab site[2]. Achieved results demonstrate an efficient level of adaptability on top of all the devices under test. Scalability has been largely proved, as well as the suitability to host different applications i.e., provide a wide range of services such as data processing, data aggregation, data compression etc.

To summarize, in this work we propose a microservice-based Lightweight Edge Gateway for the IoT. It relies on a layered architecture and on the versatility given by emerging virtualization technologies for tackling the research challenges previously mentioned. To the best of our knowledge, this is the very first contribution that brings at once: *(i)* a real implementation of a microservice-based gateway, *(ii)* a *cross-cutting* interoperability spanning for the entire IoT infrastructure, *(iii)* an exhaustive performance evaluation, carried out by means of a testbed implementation, in which we demonstrate lightweightness and feasibility of our deployment.

The remainder of the paper is organized as follows. In Section 2 we introduce the main characteristics of the LEGIoT architecture, the software, and the enabling technologies. In Section 3 we focus on the performance evaluation and in particular on the impact of virtualization technologies. Section 4 summarizes the main takeaways for both empirical and qualitative

[2]https://www.iot-lab.info

analysis under which LEGIoT is evaluated. Section 5 explores Related Works and it highlights the main advantages of our solution. Finally, Section 6 concludes the paper, giving possible future research directions.

2. The LEGIoT architecture

In current IoT deployments the presence of a gateway, as interface between sensor domain and backbone network, is essential [8]. However, in the solutions available in literature [3, 4, 5], gateways functionalities are often limited to traffic forwarding and protocol conversion.

In this work, the key idea is to exploit emerging software solutions and architectural principles, in order to enhance basic gateway features and to build a modular and flexible architecture. By merging the modularity characteristic of *microservices* and the flexibility given by container virtualization technologies, we build a highly customizable gateway, capable to fulfill the strict requirements of current IoT applications/scenarios [9].

Microservices enable the concept of modular independence among different services, which can work as standalone entities and/or interact with other components. This approach can be implemented via containers; such technologies indeed, introduce APIs that allow an efficient management of heterogeneous applications in a very flexible a versatile way.

Main drivers for the design and implementation of LEGIoT are: *(i)* interoperability; *(ii)* high energy-efficiency; *(iii)* fast allocation and flexibility in managing different services; *(iv)* isolation; *(v)* backup capabilities; *(vi)* multi-tenancy. Figure 1 shows in details the LEGIoT architecture. In order to be compliant with edge computing scenarios, LEGIoT can interact with a heterogeneous set of IoT devices, and with remote end-users acting for example, behind different cloud service providers. In the following subsections, we provide detailed information about the technological choices and the design of LEGIoT components.

2.1. Enabling Technologies

Hardware platforms. ARM architectures are constantly becoming more widespread thanks to their low-power characteristics and contained costs [10]. In the last few years, we have witnessed an increasing proliferation of different Single-Board Computers (SBC) as enabling hardware technology in a wide range of IoT use-cases [11, 12, 13, 14]. To characterize performance and suitability of our LEGIoT — on top of hardware with different features

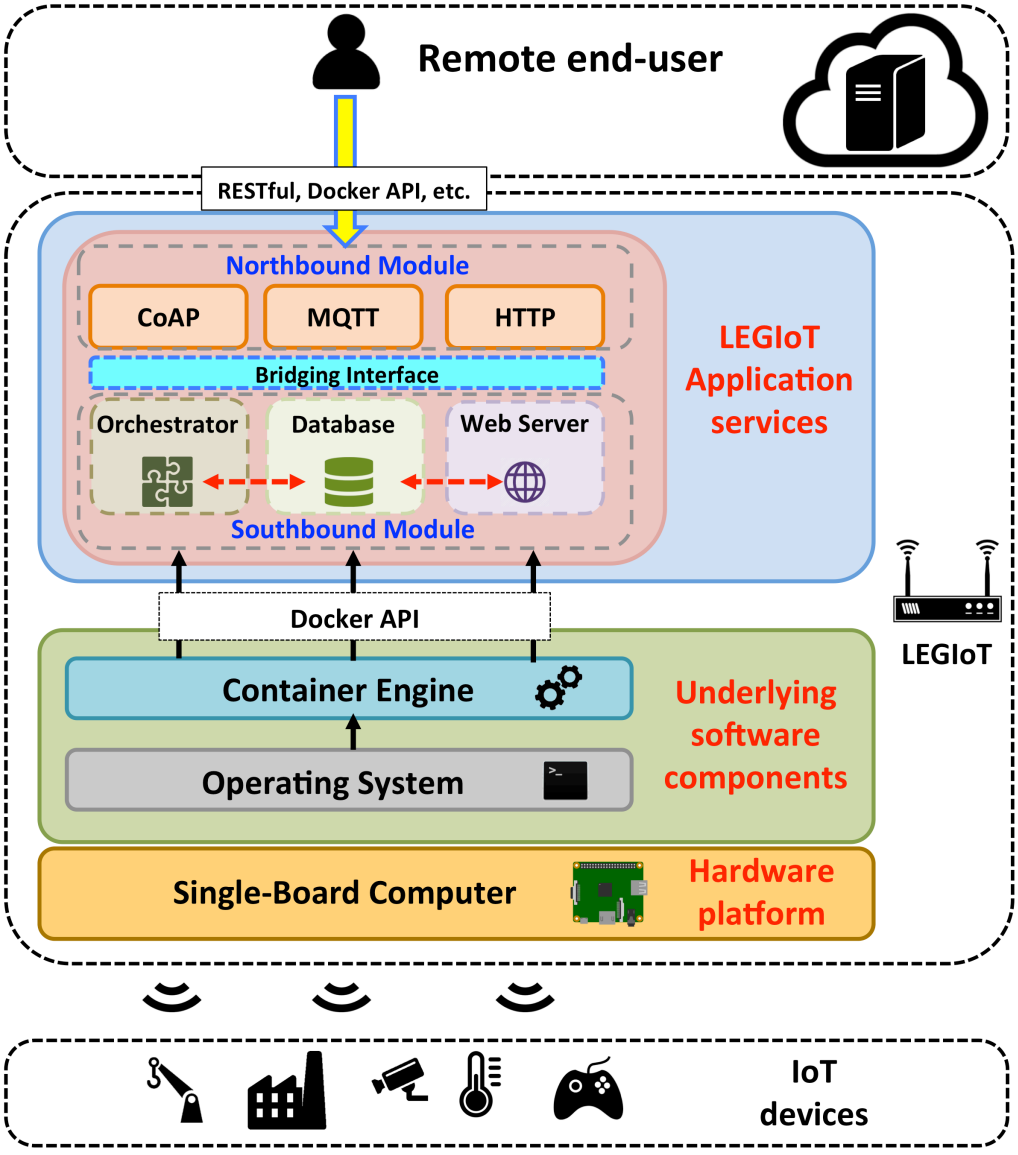


Figure 1: The LEGIoT architecture.

— in this work, we consider different SBCs. This will also help to identify possible weaknesses and performance upper-bounds of the evaluated devices. In particular, we include four devices belonging to two of the most common SBC families: Raspberry Pi and Odroid. To have a fair comparison, we choose two boards for each manufacturer: *Raspberry Pi 2* (RPi2), *Raspberry*

Pi 3 (RPi3), *Odroid C1+* (OC1+), and *Odroid C2* (OC2). The main features of these devices are summarized in Table 1. RPi2 and OC1+ on one side, and RPi3 and OC2 on the other, represent the subset to be compared as they have similar specifications.

Table 1: Single-Board Computer data sheet.

	RPi2	**RPi**3	**OC**1+	**OC**2
CPU	Quad Core @ 900 MHZ	Quad Core @ 1.2 GHz	Quad Core @ 1.5 GHz	Quad Core @ 2 GHz
Memory	1 GB DDR2	1 GB DDR2	1 GB DDR3	2 GB DDR3
Ethernet	10/100 Mb/s	10/100 Mb/s	10/100/1000 Mb/s	10/100/1000 Mb/s
USB	2.0	2.0	2.0	2.0

Underlying software components. As stated in [11], lightweight virtualization technologies enable a system that benefits of features such as: *(i)* fast building process, instantiation, and initialization of containers; *(ii)* high density of application/services due to the small container image; *(iii)* isolation between different instances. This is mainly due to the lightweight characteristics of container technologies if compared to alternative solutions such as hypervisor-based virtualization — the differences between these two approaches are widely discussed in [15]. In our implementation, we use Docker[3] (version 1.11.0) containers for executing the different instances. Docker introduces an underlying container engine, together with a functional API that allows to easily build, manage, and remove virtualized applications. With the Odroid platforms we use Ubuntu version 14.04 (for OC1+) and Ubuntu version 16.04 (for OC2) as Operating System (OS), while for the Raspberry Pi boards we use the image provided by *Hypriot*[4], which is characterized by a lightweight environment specifically optimized for an optimal use of Docker. In the choice of base boards OS, we specifically target stable releases. All the SBCs use 16GB *Transcend Premium 400x Class 10 UHS-I microSDHC* Memory Card as storage device. It is worth highlighting that unlike Raspberry Pi, the Odroid boards provide integrated support for *embedded MultiMediaCard*

[3] `https://www.docker.io/`

[4] `https://blog.hypriot.com/`

eMMC cards. This alternative storage solution offers superior performance in terms of read/write speed, in the order of hundreds of MB per seconds.

LEGIoT Application services. On top of the hardware and software runtime environment elements, we have a modular application layer that has the role to interface — by guaranteeing the requirement of interoperability — sensor network and end-user. In particular, as shown in Figure 2, our architecture defines two different modules, the *Northbound* (in charge of the communication with the Internet) and the *Southbound* (in charge of the communication with sensors).

All the components shown within Northbound and Southbound modules are virtualized by means of Docker containers ensuring the deployment of different services within an isolated environment. Moreover, due to the independence between software modules, virtualization brings other benefits e.g., software upgrading is simplified. Containers can be easily removed and/or replaced by updated protocols/applications versions, without impacting the overall gateway functionalities. Also, reliability can benefit from this architecture as it is easier to monitor all the running components, and identify in real-time the current status of the running services — e.g., potential software failure, software restart and update, etc. More generally, the high flexibility introduced by our design enables an easy integration of new services brought by the constant evolution of the IoT landscape.

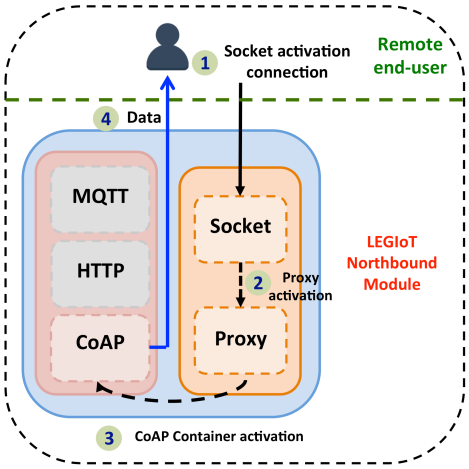
The *Southbound module* is responsible for the interaction with sensors and it is characterized by three main components: *(i)* a webserver that exposes services to the *Bridging Interface* — we use **WildFly**[5], an application server written in Java, which runs on multiple platforms; *(ii)* a search server in which all sensor data is stored — we use **Elasticsearch**[6], which provides a distributed search engine with an HTTP web interface and schema-free JSON documents; *(iii)* an orchestrator that ensures and manages communication paths between all containers and it guarantees interoperability between different sensor domains according to the VITAL[7] specifications [1]. The orchestrator is fully implemented in Python language.

The *Northbound module* provides all the necessary features for interfacing and serving the gateway with remote enduser requests. In particular, it

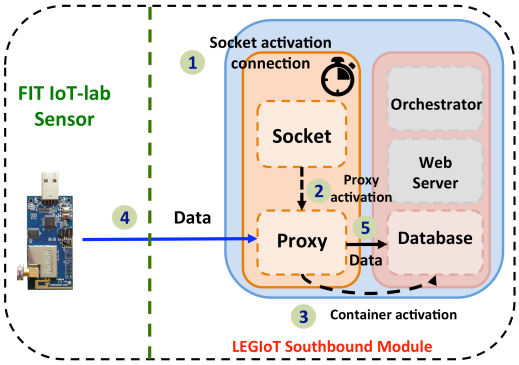
[5] `http://wildfly.org`
[6] `https://www.elastic.co`
[7] `http://vital-iot.eu`

makes available a set of protocols that can be used according to specific use-cases, needs and, requirements. CoAP, MQTT, and HTTP are example of widely used protocols to establish a communication between the gateway and remote end-users. It worths clarifying that some components, e.g., CoAP and MQTT, can be used to interact with both network sides, sensors and end-users. In our implementation, we use *libcoap*[8], *Mosquitto*[9], and *Apache HTTP server*[10] for instantiating CoAP, MQTT, and HTTP respectively. An easy integration of alternative and proprietary protocols is provided by the use of containers. The gateway can, indeed, download from remote registries the Docker images of a specific application without impacting the functionalities of the running applications.

The *Bridging interface* works as middleware between the functionalities of Southbound and Northbound modules. This interface forwards data aggregated — stored in the database container — from sensor nodes to a server by using one (or more) of the available protocols.



(a) Northbound module.



(b) Southbound module.

Figure 2: On-demand container activation for the LEGIoT.

In order to meet the *energy-efficiency* requirement, we implement a framework that allows the *on-demand activation* of Docker containers to be run.

[8] https://libcoap.net/
[9] https://mosquitto.org/
[10] https://httpd.apache.org/

In particular, we use *systemd-socket-proxyd* — a small TCP and Unix domain socket proxy that provides socket activation support for services that do not natively support socket activation. In the socket-activation framework, a socket is listening on a given port that is eventually served by the proxy/container combination. As soon as the socket receives the first connection, the *systemd* activates the proxy service, which starts the container. The proxy is also used to forward the traffic between the container and the network. Figure 2 shows a practical use of this component for both the Northbound and Southbound modules. Referring first to the Northbound module, a *Remote end-user* — for example, through a cloud service provider — ping the socket endpoint (1), which activates the proxy service (2); the latter will start the *CoAP server* container (3). After the activation, the container becomes accessible from the *Remote end-user*, who may retrieve the data from the CoAP server itself. The process afore-discussed is similar to the one of the Southbound module. The main difference lies in the fact that the socket activation is here automatically instantiated by a *timer*. Such property takes account of the periodic data transmission of IoT sensors. Therefore, it makes sense to activate the containers only when needed. Clearly, if the gateway needs to interact with event-based sensors, the socket-proxy mechanism can be activated through a *probe* signal sent by the sensor itself — similarly at the Northbound module. The example refers to the activation of two specific containers but it can be extended to all other applications stored in the gateway.

One of the main benefits introduced by this mechanism is the possibility to lower the power consumed by the gateway and, at the same time, to allow a better hardware resources usage. This might be needed in scenarios where, for example, sensors and/or remote end-users do not frequently interact with the gateway and remain *silent* over long periods. In these cases, it is possible to minimize the resources employed by the containers and dedicate them to other applications.

2.2. Use-case

The architecture proposed in this paper may be crucial for many Internet of Things applications. In this context indeed, having processing capabilities closer to the edge of the network (where *things* are deployed) results to be more efficient — in terms of latency and bandwidth — than sending all data to Clouds. Moreover, thanks to its isolation features, LEGIoT ends to well suit multi-tenant scenarios, such as smart buildings.

According to [16], intelligent buildings can be defined as systems that permit their intercommunication and which allow communication between the buildings themselves and the individual tenants. Transducers and sensors are available to measure most building related parameters and in any given situation, there may be particular needs driving their specific use.

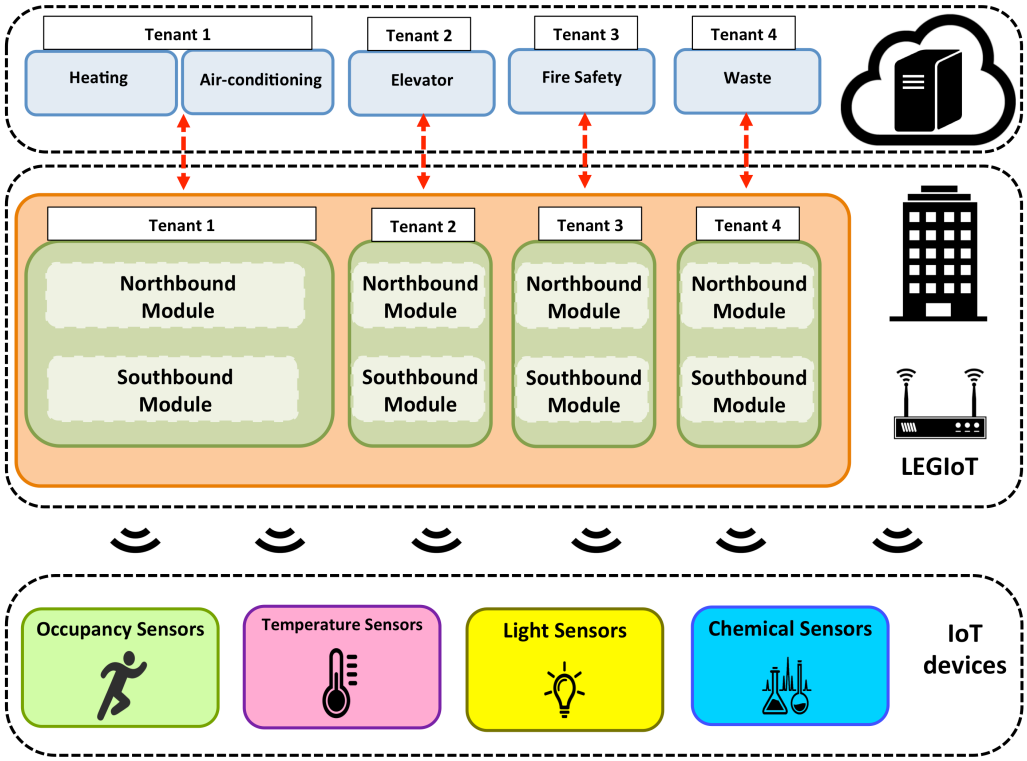


Figure 3: Smart Building architecture.

In Figure 3, we illustrate a possible smart building architecture. Sensors deployed to sense the physical environment may use different technologies (e.g., Wi-Fi, IEEE 802.15.4, Bluetooth, RFID, Visible Light Communication, etc.) to communicate with the LEGIoT. Once the communication is established, data observed by a sensor can be used by different tenants for their own application (e.g., heating, fire safety, waste management, etc.) without interfering with the rest of the system. For example, let us suppose that a company that provides water services wants to deploy its own application for collecting metering information; thanks to the isolation properties, this application will not alter the rest of the system.

3. Performance Evaluation

3.1. Experimental Setup

Sensor Network. In order to evaluate the performance of our Gateway in a real environment, we ran experimentations on the FIT IoT-lab testbed[11], a large scale infrastructure facility, spread across six different sites in France, which features almost 3000 wireless sensors nodes. The FIT IoT-lab site of Lille has been used for our performance evaluation; it is deployed over a 225 m^2 area, and it features hundreds of M3 open nodes — Table 2 summarizes the main features of these devices.

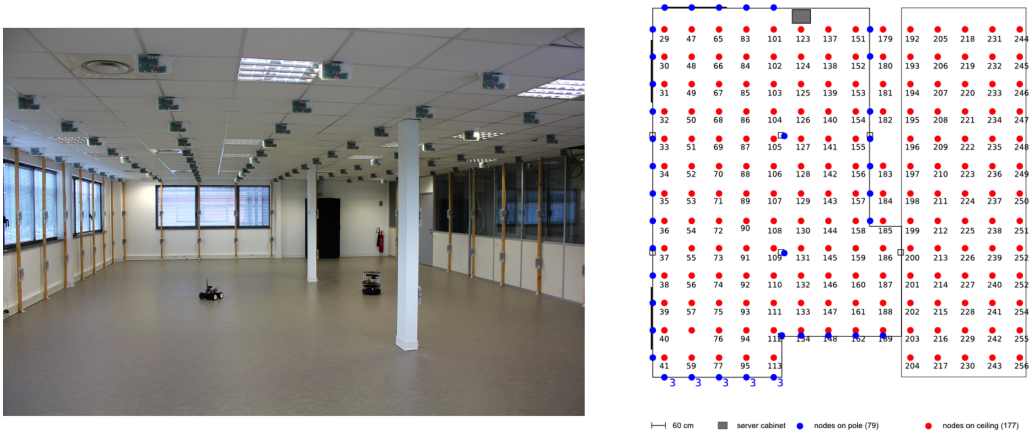


Figure 4: FIT-IoT-Lab.

The sensor nodes use Contiki-OS[12] as Operating System and in particular we flashed their firmware with *e*CACHACA, an extended version of the ranking mechanism CACHACA (Confident-based Adaptable Connected objects discovery to HArmonize smart City Applications) that we previously proposed in [17]. By running this algorithm, sensor nodes can evaluate and classify their neighborhood and the available services (e.g., temperature, light, humidity, etc.); each node indeed, advertises itself and the physical phenomena that it can observe. CACHACA is based on a rule-based fuzzy inference system and it uses parameters such as the Received Signal Strength Indication and the Timestamp — of the last frame received from a neighbor —

[11] `https://www.iot-lab.info`
[12] `http://www.contiki-os.org`

to rank nodes and services. The evaluation and all the functionalities of CACHACA are out of the scope for this work; we use the algorithm in order to establish the communication between sensor nodes and the gateway. Each sensor indeed, will send data packets to the gateway every 60 seconds — according to the ETSI specification [18] for Smart Cities.

Table 2: M3 data sheet.

Parameter	Specification
MCU	ARM Cortex M3, 32 bits, 72 Mhz, 64 kB RAM
Radio Communication	802.15.4 PHY standard, 2.4 Ghz
Power	3, 7 V LiPo battery, 650 mAh
Sensors	Light, Pressure and Temperature

LEGIoT. Our gateway architecture is implemented by using the most popular IoT SBC allowing a better assessment of the strengths and weaknesses of the different devices. In particular, we use four different SBCs (Figure 5): *Raspberry Pi 2*, *Raspberry Pi 3*, *Odroid C1+*, and *Odroid C2*.

(a) (b) (c) (d)

Figure 5: Single-Board Computers under test: (a) RPi2, (b) RPi3, (c) OC1+, (d) OC2.

Remote Server. The Northbound module evaluation is performed using a general-purpose laptop that features an Intel Core 2 Duo PC running Linux 3.13.0 with an Intel 82567LM Gigabit Ethernet card. The laptop is directly connected to the Network Interface Card of the SBC under test. In our scenario the laptop represents an end-user that can remotely access and manage the different LEGIoT functionalities.

Testbed configuration. Figure 6 shows the overall testbed setup for both the *Southbound interface* and *Northbound interface* performance evaluation. LEGIoT interacts with the FIT IoT-Lab sensors through IEEE 802.15.4 radio. On the Northbound interface, LEGIoT is connected (via Ethernet) with a general purpose server.

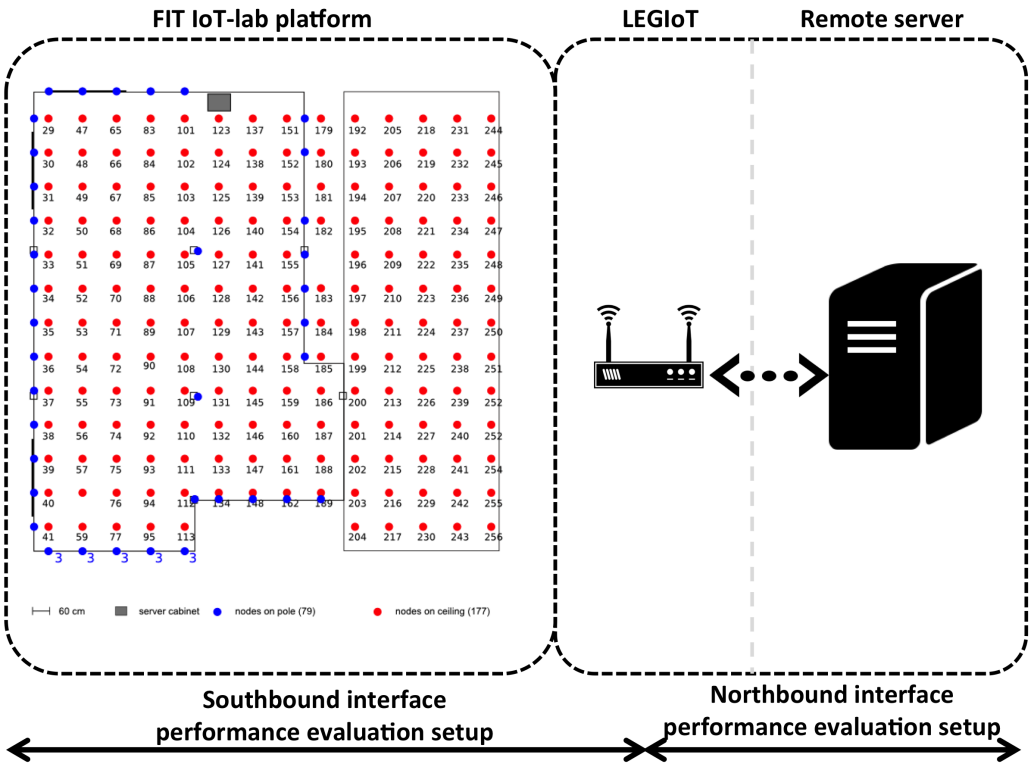


Figure 6: Testbed setup.

3.2. Experimentation Results

The validation of our proposal covers two different aspects. First, we evaluate the performance of the *Southbound interface* for each Single-Board Computer under test. In particular, we aim to understand how an increasing number of sensors affects the gateway performance. Then, we estimate the performance of the *Northbound interface*, by considering different uplink applications.

3.2.1. Southbound Interface performance

In this first evaluation, we are interested in the evaluation of the CPU, RAM, and Power Consumption of each SBC when they are stressed with different numbers of sensors. This analysis shows how hardware resources of LEGIoT are used in IoT scenario. The operations carried out by the Southbound module in this specific test involve all the virtualized services belonging to it — orchestrator, database, webserver. The sensors transmit

data to the sink connected via USB to the gateway. Once data is received, the Orchestrator ensures the storage in the Database and the availability on the webserver.

Figure 7a shows the average CPU usage when the number of connected sensor nodes increases. We can observe that an increasing number of connected sensors does not produce a linear increase in the CPU usage in all the SBCs under test. In the comparison between the 32-bit boards, RPi2 shows a higher outflow of resources. This result is somehow expected as the OC1+ features a CPU with a higher frequency clock compared to the RPi2. In any case, by considering the scenario in which 100 nodes are connected to the gateway, the CPU usage difference is slightly above the 2%. RPi3 and OC2+ — the 64-bit boards under evaluation — show similar performance. Overall, OC1+ shows the best scalability performance also compared to the more powerful OC2. The reason for this result is clarified by Figure 7b, which shows the CPU usage as a function of time — we only show the 75-node case, as it is consistent with the remaining cases. This analysis shows how the different devices treat, from the CPU usage perspective, the reception of data generated by sensors. We can observe that the trend is common to all devices; however, during the first interaction between the gateway and the sensors, OC2 generates a higher CPU usage compared with the remaining boards. These peaks result in a higher average of CPU usage, as mentioned before. Moreover, OC2 performs the Southbound module operations in a shorter time. This means that OC2 introduces a performance trade-off, as it generates higher CPU usage by shortening the speed of instances execution — due to the higher OC2 CPU clock speed. In addition, all cases investigated show the presence of peaks only during the first iterations between the gateway and the sensors. This can be explained by the fact that sensors running *e*CACHACA use a random time for scheduling transmission — on average, each node transmit over a period of 1 minute within an offset of 10 seconds. Such feature can also be observed by the remaining interactions, where a more time-spread CPU usage can be identified.

The RAM memory usage analysis defines if the gateway has to deal with memory intensive applications and how the increasing number of devices affects the memory performance. This estimation represents a relevant aspect because it suggests if the gateway could host further memory-intensive applications, intended to specific functionalities requested by a tenant and not covered by this work — e.g., data compression applications [19]. Figure 8a depicts the RAM SBCs memory usage in function of the number of sensor

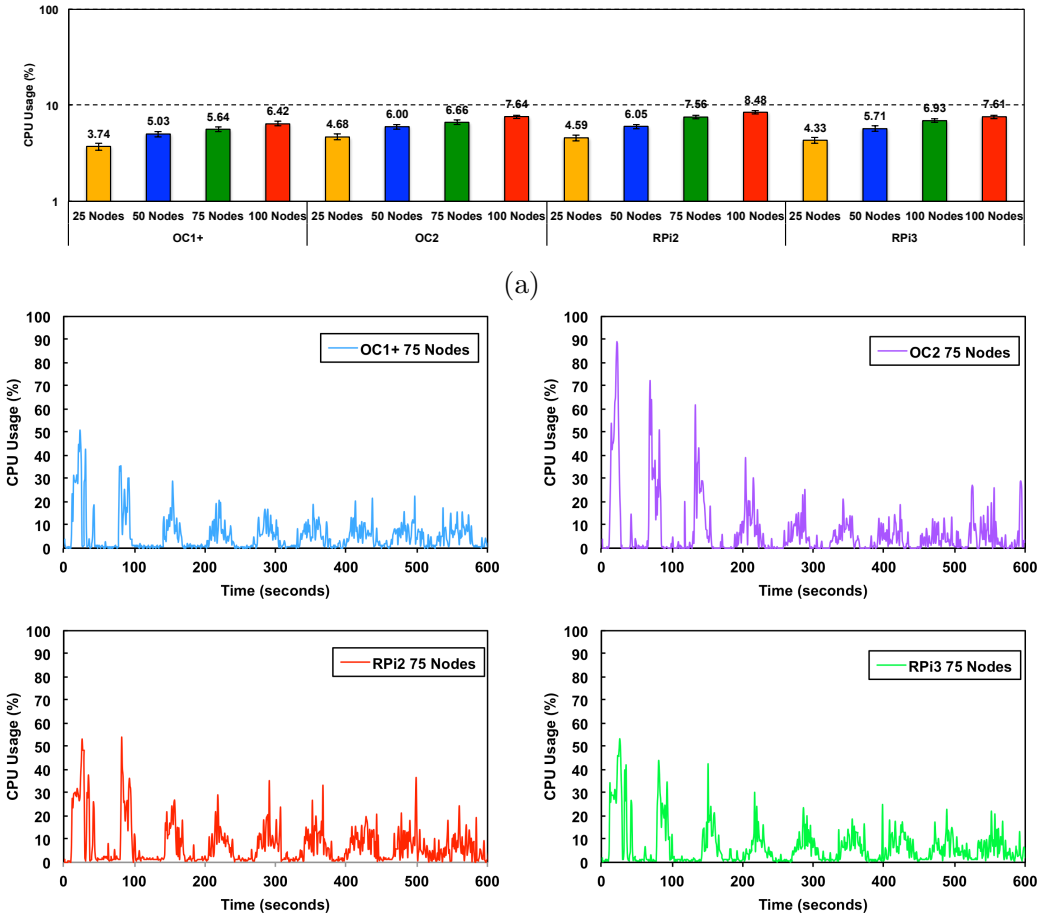


Figure 7: CPU resource utilization evaluation: (a) average CPU usage; (b) instantaneous CPU usage.

nodes — the result is normalized since the devices feature different RAM capacities. Two main insights can be drawn from this result. First, the overall usage of RAM barely overcomes the 30%, regardless the SBC considered and the number of sensors connected. Second, contrarily to what found during the CPU analysis, OC1+ introduces a slightly higher memory usage than other hardware under test.

Similarly to the previous analysis, we also want to identify usage peaks and other performance peculiarities for the RAM; then, we consider the lightest and heaviest workloads—25 and 100 nodes. Figure 8b shows a *flat* trend

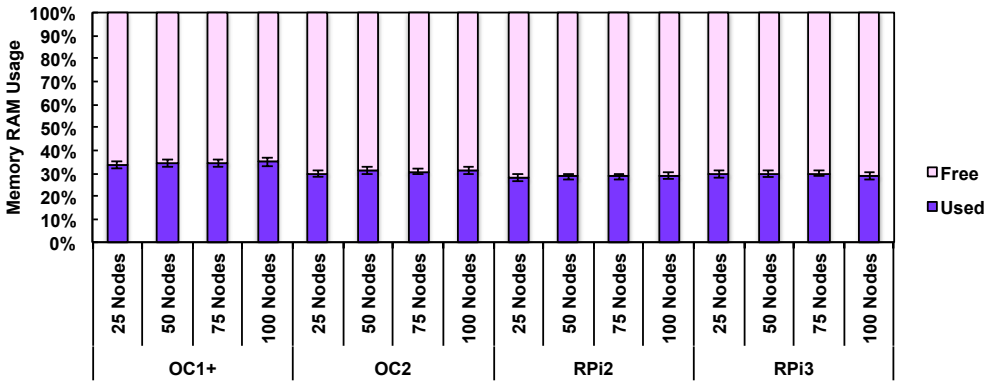
for the entire duration of the test, despite of the sensors connected to the gateway. This implies that the main contribution to the RAM usage is given by the application running in the Southbound module, and not by the communication with sensors. In fact, the figure shows an evident memory usage increase (respect to the RAM usage in idle state) after that all containers start receiving and processing data — approximately after 20 seconds from the beginning of the test. Finally, the plot also shows how the management of a higher number of sensors produces an almost negligible RAM variation.

To have a better overview on the overall performance of the system, we also evaluate the system load, a parameter that indicates the overall amount of computational work that a system performs, by including all the processes or threads waiting on I/O, networking, database, etc. [20]. It is usually expressed using three different values, averaging the past one, five, and fifteen minutes of system operation — in our experiments, we only consider the 1-minute system load. The upper-bound for the system load strictly depends on the number of CPU cores of the system under test. A system load equal to four represents the upper-bound in a system with four CPU cores — like in the RPi — after which the performance starts deteriorating.

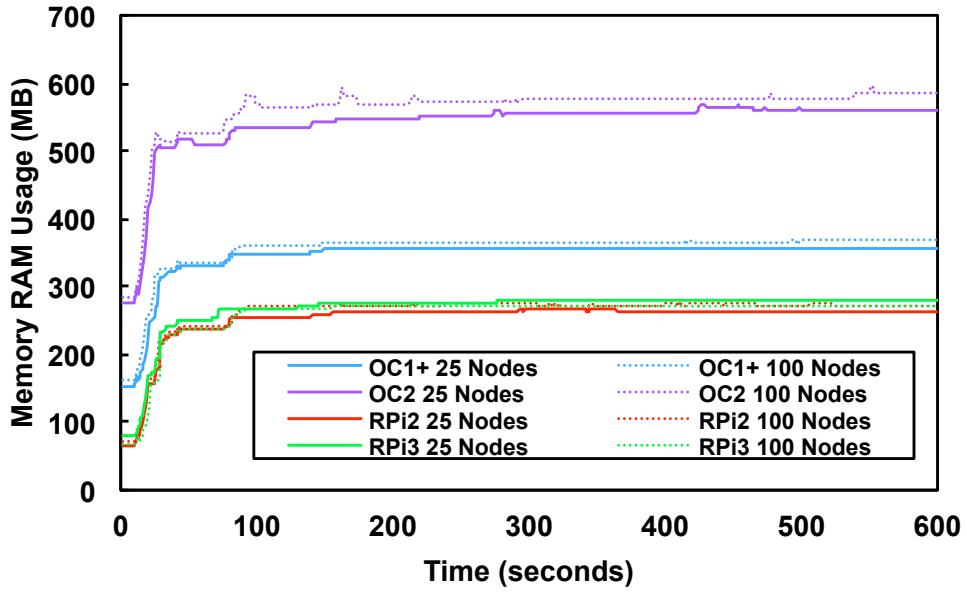
We refer to the case of 100 nodes and, in line with what was observed in the CPU and RAM performance, the average system load shows (Figure 9) a high scalability for LEGIoT, as it is never higher than one and far from the upper-bound of four.

Considering the performance of the Southbound module as a whole, we can state that our design has an efficient level of adaptability on top of all the devices under test. Scalability is largely proved, as well as the suitability to host different applications meant to provide a wider range of services such as data processing, data aggregation, data compression etc.

The last part of the Southbound module performance evaluation covers a further aspect related to the gateway requirements, i.e., the energy consumption. In particular, we want to understand how much power is consumed by the different boards when running the different southbound module tasks. In general, this evaluation can be useful in contexts in which several gateways are deployed in the same network. Network architects can efficiently design more complex networks, estimating the total amount of power consumed by a set of gateways, without neglecting the aspect of power consumption/performance trade-off. Furthermore, energy efficiency analysis can help the deployment of systems where several SBCs are combined to realize *low-power clusters* — often installed at the edge of the network — that aim



(a)



(b)

Figure 8: RAM Memory resource usage evaluation: (a) average RAM usage; (b) instantaneous RAM usage.

to replace ordinary server machines for a better energy efficiency/monetary cost trade-off as demonstrated in [21, 22].

We measure the power consumption of the SBCs by means of a voltage meter — USB-1608FS-Plus with 16 bits resolution. The power consumption

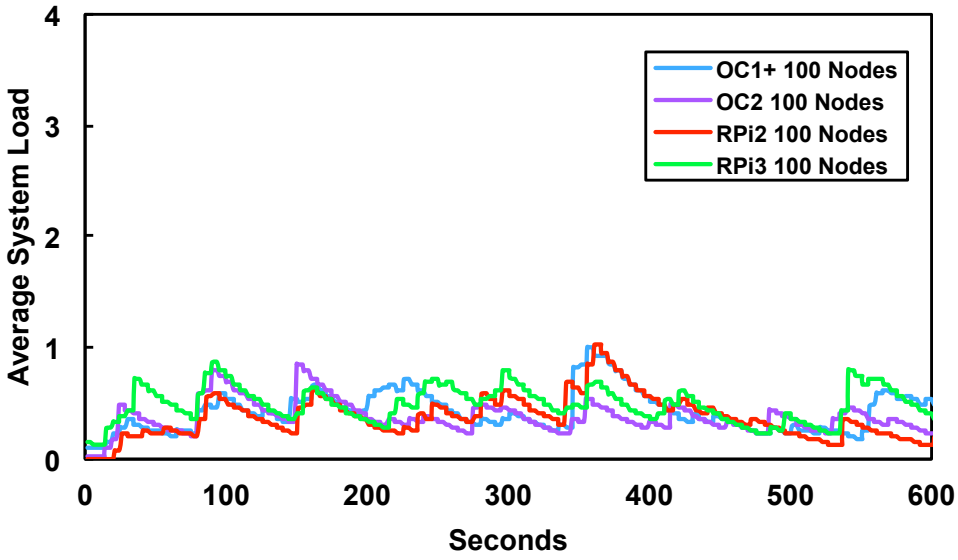


Figure 9: System Load.

is measured interrupting the power lines of the device and inserting a measurement shunt in the 5 V line. This allows, through indirect calculation, to quantify the power consumption with a marginal measurement error [23].

Figure 10a shows the average power consumption of LEGIoT in function of the number of connected sensors. Consistently to the discussion of previous results, it is not possible to find a general rule to describe the power consumption trend. We observe that a growing number of sensors connected does not produced a relevant power consumption increase for all the devices under analysis. This trend is clearly confirmed in Figure 10b that shows the maximum power dissipated by the different SBCs. The lack of a clear trend in the power consumption can be explained by the fact that sensors randomly interact with the gateway after the first transmission.

Figure 10c depicts the instantaneous power consumption of the boards that produces on average lowest (RPi2) and highest (RPi3) dissipation (for the cases of 25 and 100 nodes). This comparison clearly highlights the difference between the two devices as well as the higher power density in the comparison between 25 and 100 nodes.

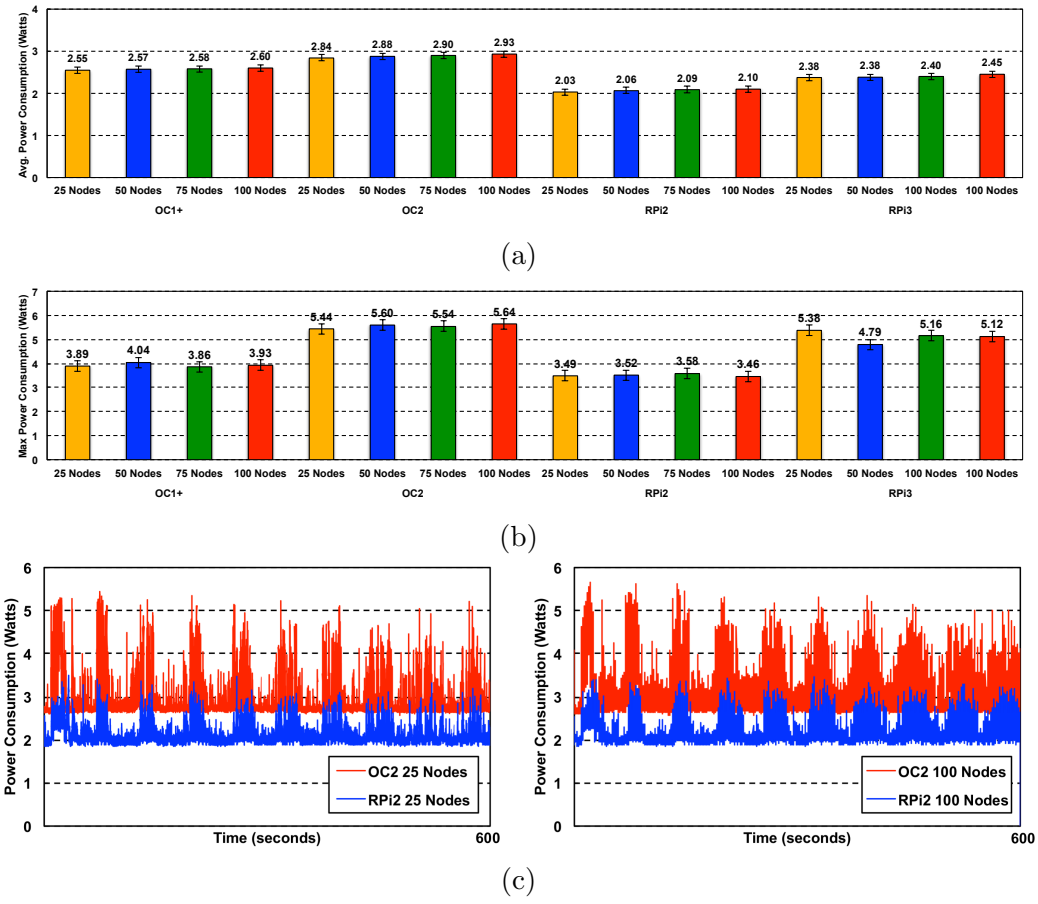


Figure 10: Power Consumption analysis: (a) average; (b) maximum; (c) instantaneous.

3.2.2. Northbound Interface performance

Once stored in the database container, data sensed needs to be available to remote end-users through the Internet. As already mentioned, several protocols can be used for establishing the connection between the gateway and remote end-users. While the system design can accommodate a wide variety of protocols, in the evaluation we make available three of the most popular application layer protocols: CoAP [24], HTTP, and MQTT [25]. The Northbound module performance evaluation aims to define how these protocols impact the gateway performance. Similarly to the Southbound analysis, we consider a growing number of remote users connected at the gateway. We fix the number of requests executed by each client to 40000.

As explained in our experimental setup, the role of end-user is played by a laptop directly connected to the different boards under test, through LAN cable. We are not interested in evaluating parameters like *Transactions per second* and *Request per second*; we focus at the performance of the gateway itself.

During this evaluation, the application layer protocols (i.e., MQTT, CoAP, HTTP) are executed one by one. For instance, when CoAP server is under analysis, its container is the only one to run in the Northbound module. This allows the specific characterization of the performance for each of the services in different hardware platforms.

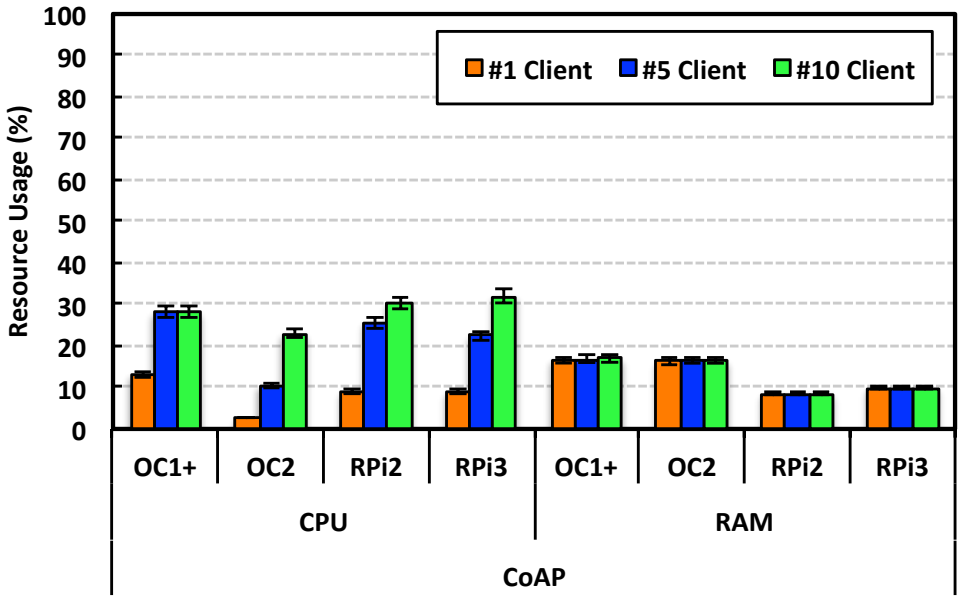


Figure 11: SBCs' resource usage during CoAP uplink transmission.

Figure 11 shows the result for the CoAP evaluation, from the CPU and memory RAM usage perspective. What stands out most is the lightweight impact of the protocol on the gateway performance. The CPU resource usage is approximately 30% in the case of 10 clients connected at the gateway. The only *outlier* is OC2 that shows a more efficient result when compared to the other boards — and for all the analyzed sub-cases. Another interesting aspect that can be observed is that the CPU usage shows a relevant increase in the step between one and five clients. However, a similar increase is not

observable when the number of connected clients further double — with the exception of OC2 that in any case shows better resource usage optimization. Also the RAM memory usage has a *flat* trend, similar to what observed for the Southbound module analysis. We also notice how the Odroid boards consume double memory resources compared to Raspberry Pi boards. This is mainly due to the quantity of memory that the devices employ when in idle state — the specific value varies from board to board and highly depends on the software environment.

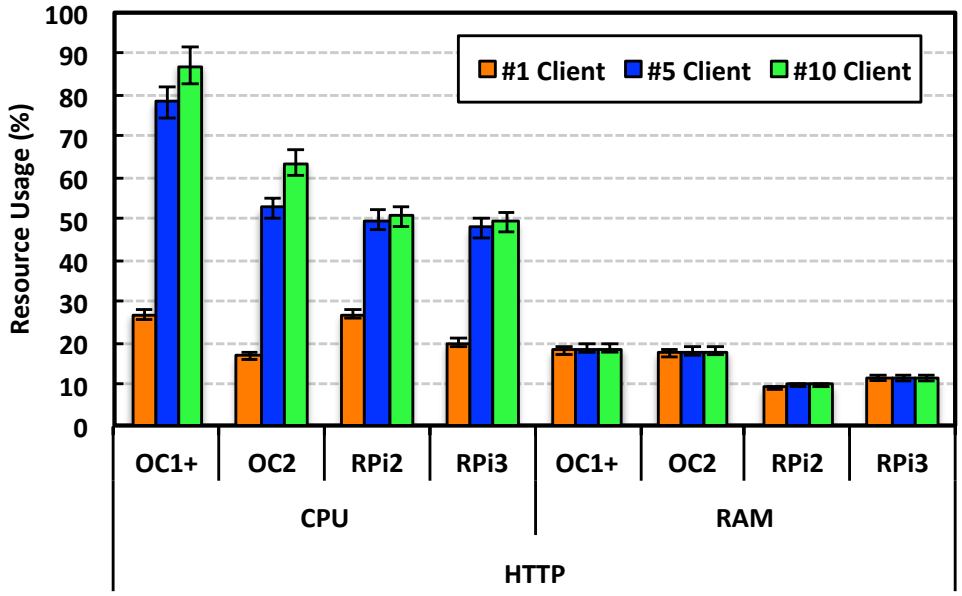


Figure 12: SBCs' resource usage during HTTP uplink transmission.

Figure 12 reports the HTTP test results. From the RAM memory usage perspective, we can notice a slight increase compared to CoAP. CPU usage shows instead different aspects although the performance trend, as a function of the users connected, roughly follows the CoAP case. It appears indeed how the management of HTTP requests has a bigger impact on the gateway performance. This result depends on all the intrinsic differences in the CoAP and HTTP protocol design. Raspberry Pi boards show a lower CPU usage compared to the Odroid, which is an unexpected result. Further investigations highlight that this behavior is due to the higher number of software interrupts (roughly 30%) that occurs on Odroid boards, showing

the sub-optimal capacity of these devices in the HTTP server utilization. This aspect will require however mored investigation, as well as the need to test alternative HTTP software solutions.

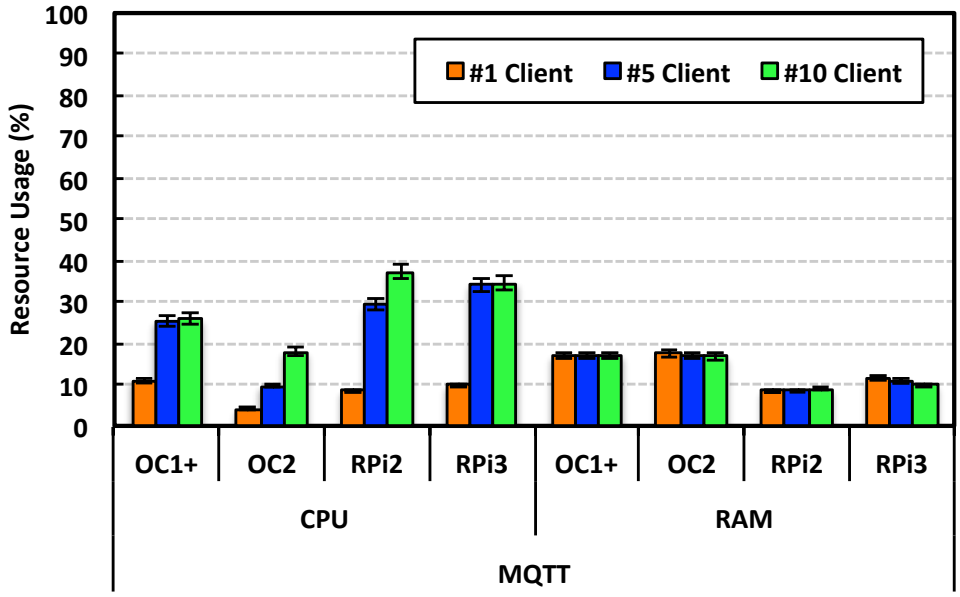


Figure 13: SBCs' resource usage during MQTT uplink transmission.

MQTT evaluation is shown in Figure 13; the similarities with the CoAP case are evident, showing therefore the high suitability and efficiency of this protocol on managing uplink communications.

3.2.3. Application performance characterization

In this subsection we want to characterize the impact that different gateway application components have on the performance. This assessment becomes crucial for understanding if our orchestrator implementation — which ensures *interoperability* among heterogeneous sensor networks and manage the communication paths in the Southbound module — owns the desired characteristic of *lightweightness*. *Database* and *webserver* rely on existing software, therefore the *orchestrator* behavior represents the sole criterion for assessing such requirement. Indeed, whatever is the performance impact of database and webserver, future and more optimized LEGIoT implementations can benefit from the flexibility given by the containers. The same

applies for the Northbound module, for which we do not provide a similar analysis. For the sake of completeness, the orchestrator is implemented in Python language, and it features approximately three hundred lines of codes.

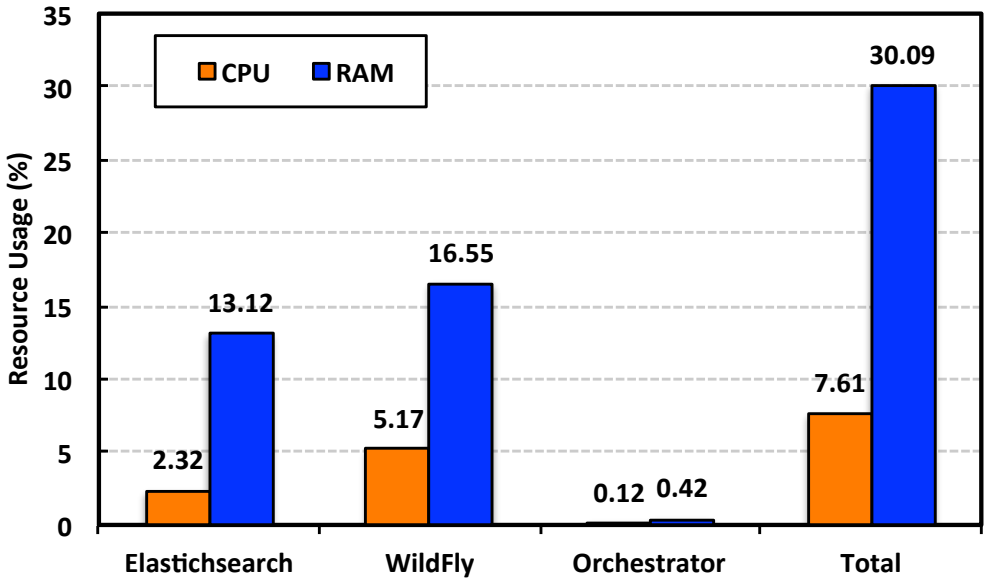


Figure 14: Southbound module applications performance characterization.

Figure 14 shows the outcome of the applications performance characterization. We only report the results of a specific case — RPi3 receiving data from one hundred sensors — as other results comply with this specific case. As regards to the CPU resource usage, it can be observed that the webserver is the component that produces the higher CPU utilization (approximately 5%). Coherently, from the RAM performance perspective, we can notice again how the webserver is the application more greedy in terms of memory resources. The *Orchestrator* is the container that requires less resources. This outcome shows an essential feature of the LEGIoT implementation, i.e., we are able to equip the gateway of a fundamental application with a minimal and almost negligible performance impact. For future and more optimized implementations, flexibility given by the use of containers will allow to easily replace database and webserver components, which show the higher impact in the Southbound module performance.

4. Empirical and Qualitative discussions

In this section we summarize the main takeaways for both empirical and qualitative analysis under which LEGIoT is evaluated.

4.1. Empirical analysis

Our proposal has been validated by means of a wide performance evaluation. The main goal this study is to show the performance impact of our implementation and its suitability on different hardware platforms. In particular, we evaluated LEGIoT on a set of well-known SBC. Furthermore, we took advantage of the FIT IoT-LAB testbed, in order to emulate a real and concrete use-case. We connected a large number of sensors to our gateway and executed a real-time performance evaluation.

To estimate the resources used by the physical node, we refer to the *Volume* metric introduced by Wood et al. in [26]. This metric considers that a physical node (in our case the gateway) can be loaded along one or more of three *dimensions* — CPU, network, and memory. The volume expresses how much the system is (over)-loaded along multiple dimensions in a combined way and it can be used to fairly estimate all resources used by each component. Equation 1 defines this metric; *cpu* stands for the normalized CPU usage, *mem* for memory, and *net* for network. The higher the utilization of a resource, the greater the volume. As a consequence, if multiple resources are heavily utilized, this will result in a higher volume.

$$Vol = \frac{1}{1 - cpu} \times \frac{1}{1 - mem} \times \frac{1}{1 - net} \tag{1}$$

However, one of the main requirements of our gateway implementation is the energy efficiency. Therefore, it makes sense to redefine the *Volume* metric by taking into account also the trade-off between resource used and power consumed while executing different tasks. This led us to define *Gateway Performance Efficiency Factor (GPEF)*, which is characterized by Equation 2:

$$GPEF = \frac{1}{1 - cpu} \times \frac{1}{1 - mem} \times \frac{1}{1 - net} \times PowerConsumption \tag{2}$$

Before discussing the results, it is worth mentioning that in our experiments, GPEF networking attribute is set to 1 for all cases, as the amount of

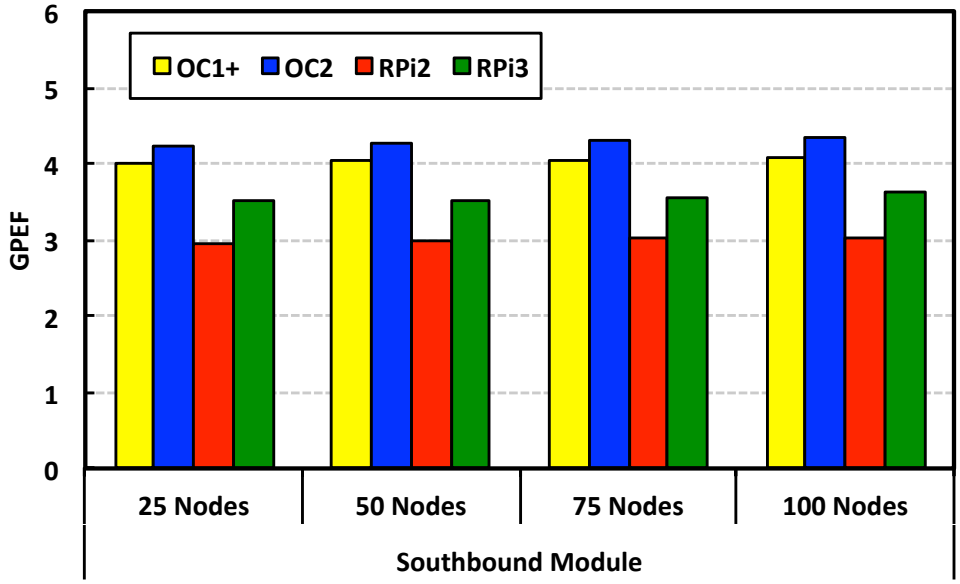


Figure 15: Southbound module GPEF characterization.

network traffic handled by LEGIoT is the same for each test, regardless of the SBC used.

Considering the underlying-hardware differences of each platform, in this last analysis we define the most suitable device to current LEGIoT implementation. By leveraging a *hardware independent* metric, we also detect potential weakness of the software implementation when operating on a given platform.

The results for the *Southbound module* (Figure 15) show that there is no *one size fits all* rule for the GPEF of the analyzed nodes. Raspberry Pi boards get the best over the Odroid boards, although within a limited range.

GPEF *Northbound module* evaluation shows a different trend that highly depends on the uplink application protocol (Figure 16). RPi2 is again the most efficient board, while OC2 outperforms RPi3. For the HTTP protocol, coherently with what has been found in Section 3, Odroid boards present a GPEF worse than Raspberry Pi.

In the GPEF evaluation — where Raspberry Pi boards perform better than Odroid ones — we are considering a relatively lightweight workload, i.e., sensing operations. As shown in [27], it exists a strong dependency

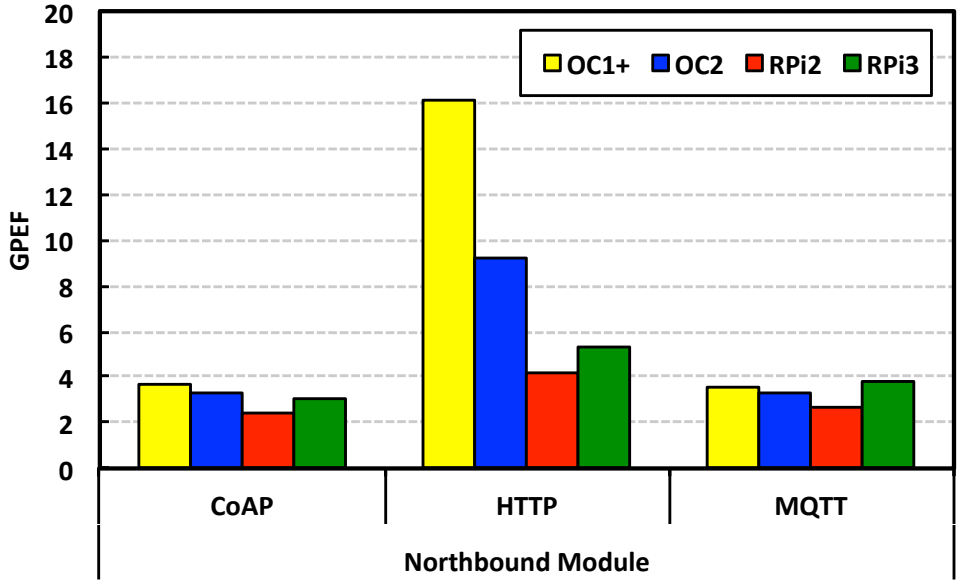


Figure 16: Northbound module GPEF characterization for the case of five clients connected to the gateway.

between edge-node performance and executed tasks. In fact, if we consider a more complex workload (e.g., video analytics) [28], GPEF evaluation may be completely different. Therefore, in heterogeneous scenarios where edge-computing devices can provide a wide range of services, Odroid boards could be more suitable.

4.2. Qualitative analysis

Interoperability. The IoT landscape is highly fragmented in terms of hardware capabilities / constraints, network protocols, and application requirements. To deal with this challenge, in our architecture we propose an orchestrator that guarantees interoperability between different sensor domains. Sensor data is indeed stored into LEGIoT according to the VITAL ontology [1], which relies on Linked Data standards (i.e., RDF, JSON-LD, and ontologies). In particular, VITAL combines several ontologies and it uses Semantic Sensor Network (SSN) [29] to describe sensors, including their accuracy and capabilities, observations, methods for sensing, concepts for operating and survival ranges, and deployments. Once stored, data can be accessed by all the authorized users to build their own applications, with-

out interfering with other tenants, and without the need to develop specific communication protocols.

High energy-efficiency. Our claim of energy-efficiency is based on the implementation, integration, and use of the Socket-Proxy (S/P) container activation introduced in Section 2.1. Although the performance evaluation presented in Section 3 has been performed without enabling such mechanism, here we present the result of a further empirical assessment. In this evaluation, sensors from the FIT-IoT lab platform were transmitting data, on average, every minute. This implies that between two consecutive sensing operations, all the active containers can be temporarily paused through the S/P framework and consequently lowering LEGIoT resource usage and power consumption. To prove this, we have set the S/P container activation to freeze running applications for 35 seconds between two sensing operations — we included a guard interval for avoiding the risk of overlapping with the upcoming sensing operation.

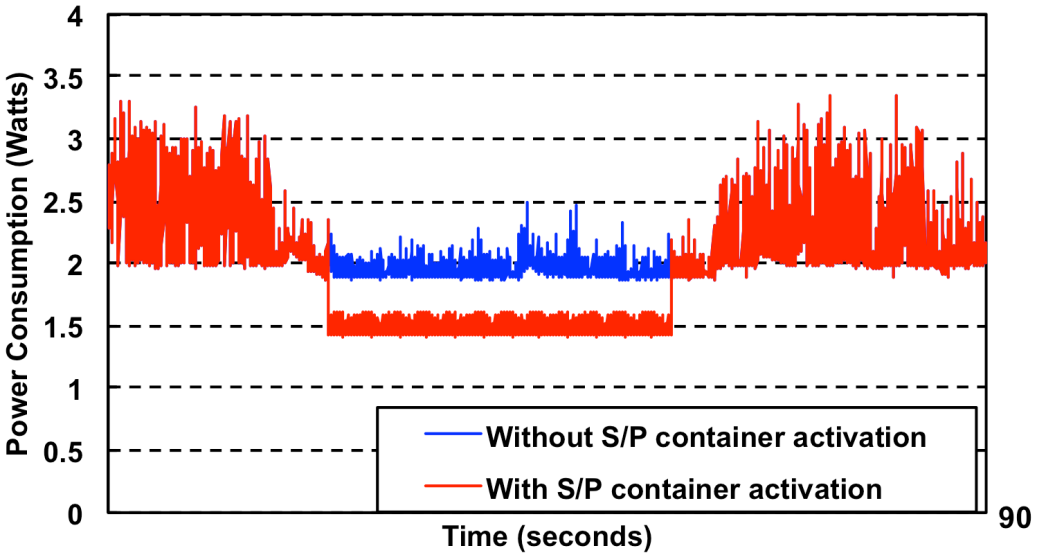


Figure 17: Energy Efficiency evaluation. In this evaluation, we consider the RPi2 when receiving data from 100 sensors.

Figure 17 shows how the LEGIoT power consumption is lowered of approximately 0.5 Watts during the 35 seconds time frame. Taking into consideration this energy efficiency gain, it is reasonable to expect that in environments where gateways operate twenty-four hours a day and seven days a

week, the use of such mechanism can have a remarkable impact.

Fast services allocation. The containers activation time of SBCs, when managing a workload that gradually becomes more complex, has been exhaustively evaluated in [28] by using the same set of boards presented in this paper. Container activation time remains below 2000 milliseconds in most of the analyzed cases. This represents a significant result if we consider the reduced hardware capabilities of SBCs when compared to more powerful devices, in which instead the containers activation time stands around the hundred of milliseconds. According to [30], activation time can be further reduced through alternative container-engine configurations.

Easy management of different services. As evidenced by this work, the main benefit that clearly emerges from employing container technologies for LEGIoT implementation is the possibility of avoiding the strict dependency on a given technology or use-case. Applications designed to manage and use extremely different technologies can be deployed on containers. In addition, equipping LEGIoT of newer services becomes easier as the only operation to be performed is configuration and instantiation. This allows to avoid all the complex re-programmability operations and updating processes needed in order to preserve the software life-cycle management.

Isolation and Multi-Tenancy. Level of security isolation guaranteed in applications developed within containers has led in the past to a lively debate especially in the open-source community[13]. Although first versions of Docker were lacking of a satisfactory level of security, the latest released versions include several security enhancements to cope with these issues[14]. A tangible proof of this is the release of the Docker Security Benchmark, which has been developed together with the Center for Internet Security. This tool allows to test the penetration of your application deployment environment towards a wide variety of known security issues[15].

A direct consequence of the *isolation*, ensured by Docker itself, is the possibility to make LEGIoT a *multi-tenant* platform. This means that the gateway can be shared between different users. This characteristic turns out to be particularly useful in contexts in which the gateway keeps the peculiarity of being a *vendor-independent* platform (as shown in Section 2.2),

[13]https://zeltser.com/security-risks-and-benefits-of-docker-application/
[14]https://blog.docker.com/2016/02/docker-engine-1-10-security/
[15]https://www.cisecurity.org/cis-benchmarks/

where different tenants can benefit of the same hardware platform, but still apply their own policies in the application management.

Moreover, it is important to remark that to store and distribute Docker images — containing all the applications code and dependencies — there are public/private registries. This latter feature can allow network designer to place private registries either at the gateway or in the cloud. For all the communications *registry to Docker client*, the Transport Layer Security (TLS) protocol can be enabled making secure communication over HTTPS. TLS ensures authenticity of the registry end-point, encryption of the traffic to/from registry, and a certificate-based client-server authentication. It is possible to install a Certificate Authority (CA) root certificate for the registry and setting the client TLS certificate for verification [31]. The combination of these features makes LEGIoT secure and then capable to strongly ensure isolation and multi-tenancy.

Backup capabilities and **Ease application updating**. A *Docker container* is a runnable instance of a *Docker image*. Docker images are stored in specific private/public registries that, in LEGIoT context, can be set up both locally and in a cloud-service. When a container is created/executed, the configuration setup of the *dockerized* application is specified together with any other dependency (e.g., libraries). While running a container from an image, Docker uses an overlay file-system (UnionFS) to add a read-write layer on top of the image. UnionFS allows Docker to store images as a series of layers; the different stored layers are cached during the build process, speeding up the building process and saving disk space.

When a new container is created, a new writable layer on top of the underlying layers is added. All changes made to the running container — such as writing new files, modifying existing one, and/or deleting — are written into a thin writable container layer. Therefore, the major difference between a container and an image is this top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged. Since each container has its own writable container layer and all changes are there stored, multiple containers can share access to the same underlying image and yet have their own data state.

In LEGIoT, the advantages brought by Docker in the efficient management of the file-system are multiple. First, LEGIoT can benefit of easier *backup capabilities*. For example, if one of the LEGIoTs tenants wants to

monitor its managed sensors at a given period time after a trigger event, it can activate a *database container* and start storing the sensed data specifically for that period of time. The execution of such container will not interfere with a parallel *database container*, which was already executing the same task. Similarly, if a system failure happens in the sensor network, the tenant has also the possibility to backup running *database container* and *flesh out the story* up to the failure moment.

The second advantage is given by the possibility to execute easy update operations on the stored images. For instance, referring specifically to the *CoAP container*, new *under test* protocol functionalities can be verified, without interfering with already *working* version. This can be easily done by running a CoAP container from the stored CoAP image, and integrating the new functionality in the running container. The implemented changes can be *committed* under a new image without contrasting with the one previously stored. The new image is built on top of the older one, optimizing also the disk space usage. This mechanism allows to easily integrate and test new protocol functionalities. Furthermore, the whole platform can benefit of an optimized software life-cycle management, since platforms upgrade turns out to be extremely easy.

5. Related Work

In this Section we survey the most related works available in literature. In particular, we summarize LEGIoT competitors into three different categories: IoT gateways, Virtualization at the Edge of the Network, and Microservices-based IoT gateways. For each of these solutions, we consider the following key features:

- *Real-Testbed implementation.* It provides information about the existence of real prototypes of the proposed architectures.
- *Scalability.* As consequence of a real-testbed implementation, we want to evaluate scalability and its trend in function of the number of sensors that can be simultaneously connected to the gateway.
- *Interoperability Downlink/Uplink.* It shows the ability of the gateway to interact with different technologies and to easily update/merge new applications.

- *Gateway Energy-efficiency optimization.* In nowadays IoT gateways, this may be considered as key feature. It is then important to see how related works deal with this challenge.

- *Isolation and Multi-tenancy.* These are strictly interconnected properties that are becoming essential in recent use-cases.

Table 3 shows the fulfillment of the above requirements.

IoT Gateways. Current literature presents several proposal of network architectures in which the presence of a gateway is required as interface between backbone and sensor network. A set of requirements and common features that an IoT gateway must include is listed in [32] by Chen et al. In particular, a gateway has to act as a proxy, which interconnects the sensor domain with the backbone network. The outlined features are: (i) *multiple interfaces*, needed to avoid possible mismatch between the technology employed by the sensors to connect with the IoT gateway, and the rest of the network with the IoT gateway as well; (ii) *protocol conversion*, in order to address the issue explained before; (iii) *manageability*, which refers to the needs of the gateway to be managed by external servers, and to the ability of it to control, configure and operate with the sensors. LEGIoT is capable to meet all of the aforementioned properties.

Authors in [33] introduce an IoT architecture in which a device called "Wireless Gateway" provides functionality of backbone between M2M (Machine-to-Machine) devices and remote peers (i.e., client) over the Internet. More in detail, the wireless gateway is characterized by two different interfaces; the *north interface* that provides discovery functionality to the mobile clients, and enables clients to detect M2M devices and to retrieve data from them. The *south interface* — which is a collection of REST (Representational State Transfer) web services — delivers management and storage functionality just for the M2M devices. This architecture represents a good starting point, however it results limited in the number of interfaces and manageability. Moreover, even though authors provide a comprehensive description of the architecture, details regarding the hardware used for the prototype are completely missing. The authors claim a device *"scalable to handle huge amount of traffic as it is based on RESTful paradigm and SenML"*. However, a performance evaluation that proves the statement is missing.

In [34], authors propose a smart IoT gateway that has three important benefits: it can communicate with different networks, it has flexible protocol

to translate different sensor data into a uniform format, and it has unified external interfaces. This architecture results similar to LEGIoT, however, thanks to lightweight virtualization techniques, in our proposal, we can achieve better performance in terms of isolation, multi-tenancy, energy-efficiency, and backup capabilities. Authors claims better scalability than other solutions. However, they do not include any empirical characterization that allows to fully demonstrate the requirement. The proposed gateway does not include any mechanisms for flexible applications' allocation, which would consequently bring benefits in terms of energy efficiency. Furthermore, implementation of security policies are, also in this case, left to future implementations.

In [35], authors propose a gateway and Semantic Web enabled IoT architecture to provide interoperability between systems, which utilizes established communication and data standards. The Semantic gateway as Service (SGS) allows translation between messaging protocols such as XMPP, CoAP and MQTT via a multi-protocol proxy architecture. The interoperability properties as well as the multi-protocol architecture are concepts similar to our proposal, however we offer properties that are becoming crucial in IoT scenarios, such as flexibility, isolation, etc. Our solution indeed, is not protocols dependent, different standards can run on top of it without effecting the rest of the system. Authors released software implementation but there are no instructions on the type of hardware suitable. Gateway scalability evaluation is not performed and authors do not include any mechanisms for flexible applications' allocation, which would consequently bring benefits in terms of energy efficiency.

Finally, in [36] authors propose a *Distributed Internet-like Architecture for Things (DIAT)*. This architecture satisfies most of the requirements met by LEGIoT, by exploiting a layered architecture that provides various levels of abstraction. Compared to our solution, DIAT features a complex architecture suitable more for IoT Platform-as-a-Service (PaaS) [37] and IoT middleware [38] solutions. That is, adaptability of DIAT on low-power edge nodes such as SBCs may be questionable. An empirical evaluation that fully test scalability is missing.

Virtualization at the Edge of the Network. Current literature presents several proposals of solutions in which virtualization technologies are employed at the network edge and/or on top of low-power nodes such as Single-Board Computers (SBCs).

Container technologies are also used in a Capillary Network scenario [8],

Table 3: Comparison with alternative proposals.

Gateway feature	**LEGIoT**	**[33]**	**[36]**	**[34]**	**[35]**	**[8]**
Scalability	Yes	No	Not evaluated	Not evaluated	No	Partially
Real-Testbed implementation	Yes	Yes	Yes	Yes	No	Yes
Interoperability Downlink — Uplink	Yes — Yes	Limited — Limited	Yes — Yes	Yes — Yes	Yes — Yes	Limited — Yes
Energy-efficiency	Yes	Not evaluated	Not evaluated	No	No	Partially
Isolation and Multi-Tenancy	Yes	No	Yes	No	No	Yes

where they serve for packaging, deployment, and execution of software both in cloud and in more constrained environments (i.e., local Capillary gateways). The dual purpose of this latter entity is to provide connectivity between short-range and cellular networks, and to make available different software components for local device management and instantiation of distributed cloud processes. Scalability test has not been performed in this case.

In [39], Imsail et al. evaluate Docker containers as enabling technology for the deployment of an Edge Computing platform. The conclusion is that — after evaluating: *(i)* deployment and termination of services; *(ii)* resources & services management; *(iii)* fault tolerance; *(iv)* caching capabilities — Docker represents a good solution to be employed in edge computing contexts.

Following these hints, in [11], we included lightweight virtualization technologies in the design of an IoT gateway. We employed virtualized software in order to provide a dense deployment of services at the gateway level. Particularly interesting is the analysis of the possible interactions between IoT sensors and the gateway. Such analysis suggests how the dynamic allocation of services, by means of containers, brings several benefits from the gateway performance perspective.

In [19], we proposed the design of an IoT gateway that can be efficiently employed also in edge computing architectures. In that study, we have shown how to efficiently and flexibly use Docker containers in order to customize an IoT platform, which offers several virtualized services that range from: *(i)* Device Management capabilities; *(ii)* Software Defined Networking (SDN) support; *(iii)* Orchestration and Data Management capabilities.

Microservices-based IoT gateways. The Agile project[16] proposes an architecture for gateways embracing the microservices-based concepts. Design guidelines[17] are the same that brought us to design and implement

[16] `http://agile-iot.eu/`

[17] `http://github.com/Agile-IoT/Architecture/blob/master/README.md`

LEGIoT. However, according to project documentation[18] and project publications[19], Agile gateway has not been fully prototyped yet — the project has started in the second half of 2016 and it will last three years. In these terms, LEGIoT can be considered as a *precursor implementation* of future AGILE releases. From the architectural point of view, Agile does not include any mechanism to enhance energy efficiency and, so far, its implementation has been designed in order to be tested only on top of Raspberry Pi boards. Agile project has also planned to test gateway performance through the use of FIT IoT-lab platform. A direct comparison with LEGIoT could be therefore feasible in the near future.

6. Conclusions

In this paper, we introduce LEGIoT: a Lightweight Edge Gateway for the Internet of Things. We have shown how, by means of container-virtualization technologies, is possible to satisfy important gateway requirements that are crucial aspects in the evolving IoT/Edge landscape. Our proposal shows flexibility — given by the use of emerging technologies — and newer architectural paradigms, allowing to deliver an efficient IoT services provisioning. Furthermore, we have demonstrated how our approach can be very efficient on keeping gateway's lifecycle easily upgradable, and reactive at the introduction of new IoT protocols and services.

Acknowledgments

This work is partially funded by the FP7 Marie Curie Initial Training Network (ITN) METRICS project (grant agreement No. 607728)

[18] `http://agile-iot.eu/resources/agile-wiki/`
[19] `http://agile-iot.eu/resources/#Scientific_Papers`

References

[1] R. Petrolo, V. Loscrì, N. Mitton, Towards a Smart City based on Cloud of Things, a survey on the smart city vision and paradigms, Transactions on Emerging Telecommunications Technologies 28 (1) (2015) 1–12.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Communications Surveys Tutorials 17 (4) (2015) 2347–2376.

[3] F. Berkers, M. Roelands, F. Bomhof, T. Bachet, M. van Rijn, W. Koers, Constructing a multi-sided business model for a smart horizontal iot service platform, in: Proc. of ICIN - International Conference on Intelligence in Next Generation Networks, 2013.

[4] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, M. Mohammadi, Toward better horizontal integration among iot services, IEEE Communications Magazine 53 (9) (2015) 72–79.

[5] I. Farris, R. Girau, L. Militano, M. Nitti, L. Atzori, A. Iera, G. Morabito, Social virtual objects in the edge cloud, IEEE Cloud Computing 2 (6) (2015) 20–28.

[6] D. Evans, The Internet of Things - How the Next Evolution of the Internet is Changing Everything, CISCO white paper (2011) 1–11.

[7] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, et al., Mobile-edge computing introductory technical white paper, Mobile-edge Computing (MEC) industry initiative.

[8] O. Novo, N. Beijar, M. Ocak, J. Kjllman, M. Komu, T. Kauppinen, Capillary networks - bridging the cellular and IoT worlds, in: Proc. of WF-IoT - IEEE World Forum on Internet of Things, 2015.

[9] O. Yilmaz, et al., 5G Radio Access for Ultra-Reliable and Low-Latency Communications, Ericsson Research Blog.

[10] B. Smith, ARM and Intel battle over the mobile chip's future, Computer 41 (5) (2008) 15–18.

[11] R. Petrolo, R. Morabito, V. Loscrì, N. Mitton, The design of the gateway for the cloud of things, Annals of Telecommunications (2016) 1–10.

[12] R. Morabito, R. Petrolo, V. Loscri, N. Mitton, G. Ruggeri, A. Molinaro, Lightweight Virtualization as Enabling Technology for Future Smart Cars.

[13] D. Guinard, V. Trifa, Building the Web of Things: With Examples in Node.Js and Raspberry Pi, 2016.

[14] X. Q. Li, X. Ding, Y. Zhang, Z. P. Sun, H. W. Zhao, IoT Family Robot Based on Raspberry Pi, in: Proc. of ISAI - International Conference on Information System and Artificial Intelligence, 2016.

[15] R. Morabito, J. Kjällman, M. Komu, Hypervisors vs. lightweight virtualization: a performance comparison, in: Proc. of IC2E - International IEEE Conference on Cloud Engineering, 2015.

[16] ITU, Intelligent sustainable buildings for smart sustainable cities, Available at: `https://www.itu.int/en/ITU-T/focusgroups/ssc/Documents/website/web-fg-ssc-0136-r6-smart-buildings.docx` (2015).

[17] R. Petrolo, V. Loscri, N. Mitton, Confident-based Adaptable Connected objects discovery to HArmonize smart City Applications, in: Proc. of WD - IFIP Wireless Days, 2016.

[18] ETSI, Technical report 103 055, Available at: `http://www.etsi.org/deliver/etsi_tr/103000_103099/103055/01.01.01_60/tr_103055v010101p.pdf` (2011).

[19] R. Morabito, N. Beijar, Enabling Data Processing at the Network Edge through Lightweight Virtualization Technologies, in: Proc. of SECON - International IEEE Conference on Sensing, Communication, and Networking workshop, 2016.

[20] R. Walker, Examining load average, Linux Journal 2006 (152) (2006) 5.

[21] C. Pahl, S. Helmer, L. Miori, J. Sanin, B. Lee, A Container-based Edge Cloud PaaS Architecture Based on Raspberry Pi clusters, in: Proc. of FiCloudW - Future Internet of Things and Cloud Workshops, 2016.

[22] F. P. Tso, D. R. White, S. Jouet, J. Singer, D. P. Pezaros, The Glasgow Raspberry Pi Cloud: A scale model for cloud computing infrastructures, in: Proc. of ICDCSW - International IEEE Conference on Distributed Computing Systems Workshops, 2013.

[23] F. Kaup, P. Gottschling, D. Hausheer, PowerPi: Measuring and modeling the power consumption of the Raspberry Pi, in: Proc. of LCN - International Conference on Local Computer Networks, 2014.

[24] C. Bormann, A. P. Castellani, Z. Shelby, CoAP: An Application Protocol for Billions of Tiny Internet Nodes, IEEE Internet Computing 16 (2) (2012) 62–67.

[25] U. Hunkeler, H. L. Truong, A. Stanford-Clark, MQTT-S: A publish/subscribe protocol for Wireless Sensor Networks, in: Proc. of COMSWARE - International Conference on Communication Systems Software and Middleware and Workshops, 2008.

[26] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Sandpiper: Black-box and Gray-box resource management for virtual machines, Computer Networks 53 (17) (2009) 2923–2938.

[27] R. Morabito, Inspecting the performance of low-power nodes during the execution of edge computing tasks, in: Proc. of CCNC - International IEEE Consumer Communications and Networking Conference, 2017.

[28] R. Morabito, Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation, IEEE Access 5 (2017) 8835–8850.

[29] M. Compton, et al., The SSN Ontology of the W3C Semantic Sensor Network Incubator Group, Web Semantics: Science, Services and Agents on the World Wide Web 17.

[30] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, Slacker: Fast Distribution with Lazy Docker Containers, in: FAST, Vol. 16, 2016, pp. 181–195.

[31] Docker, Secure Engine Docker security, Available at: `https://docs.docker.com/engine/security/security/`.

[32] H. Chen, X. Jia, H. Li, A brief introduction to iot gateway, in: Proc. of ICCTA - International Conference on Communication Technology and Application, 2011.

[33] S. K. Datta, C. Bonnet, N. Nikaein, An IoT gateway centric architecture to provide novel M-2-M services, in: Proc. of WF-IoT — IEEE World Forum on Internet of Things, 2014.

[34] S. Guoqiang, C. Yanming, Z. Chao, Z. Yanxu, Design and Implementation of a Smart IoT Gateway, in: Proc. of GreenCom and iThings/CPSCom, 2013.

[35] P. Desai, A. Sheth, P. Anantharam, Semantic Gateway as a Service Architecture for IoT Interoperability, in: Proc. of MS - International IEEE Conference on Mobile Services, 2015.

[36] C. Sarkar, A. U. N. SN, R. V. Prasad, A. Rahim, R. Neisse, G. Baldini, DIAT: A scalable distributed architecture for IoT, IEEE Internet of Things journal 2 (3) (2015) 230–239.

[37] F. Li, M. Vögler, M. Claeßens, S. Dustdar, Efficient and scalable IoT service delivery on Cloud, in: Proc. of CLOUD - International IEEE Conference on Cloud Computing, 2013.

[38] M. A. Chaqfeh, N. Mohamed, Challenges in middleware solutions for the Internet of Things, in: Proc. of CTS - International Conference on Collaboration Technologies and Systems, 2012.

[39] B. I. Ismail, E. M. Goortani, M. B. A. Karim, W. M. Tat, S. Setapa, J. Y. Luke, O. H. Hoe, Evaluation of Docker as Edge computing platform, in: Proc. of ICOS - International IEEE Conference on Open Systems, 2015.