

The design of the gateway for the Cloud of Things

Riccardo Petrolo, Roberto Morabito, Valeria Loscrì, Nathalie Mitton

▶ **To cite this version:**

Riccardo Petrolo, Roberto Morabito, Valeria Loscrì, Nathalie Mitton. The design of the gateway for the Cloud of Things. Annals of Telecommunications - annales des télécommunications, 2016, 10.1007/s12243-016-0521-z . hal-01321666v2

HAL Id: hal-01321666

https://inria.hal.science/hal-01321666v2

Submitted on 20 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

The design of the gateway for the Cloud of Things

Riccardo Petrolo · Roberto Morabito · Valeria Loscrì · Nathalie Mitton

Received: 30 September 2015 / Accepted: date

Abstract The increasing momentum of the Internet of Things (IoT) leaded to the development of a huge number of applications in different domains. Those applications are based on different standards and protocols, making therefore the IoT landscape widely fragmented. In this context, the evolution of Web semantic technologies together with the popularity of Cloud computing represents a solution to enable the horizontal integration of various IoT applications and platforms. This is what the Cloud of Things (CoT) aims to achieve. In this paper, we propose the design of a gateway for the Cloud of Things. The proposed gateway is able to manage semantic-like things and at the same time to act as an end-point for the presentation of data to users. Moreover, thanks to the use of virtualized software - which introduces a negligible impact in terms of performance - the gateway enables a lightweight and dense deployment of services. The paper describes the above technologies and how to combine them in order to design the gateway. Furthermore, we provide information about use cases, hardware, performance evaluation, and future hints to enhance the gateway.

Keywords Cloud of Things · Distributed Cloud Computing · Sensor Networks · Internet of Things

1 Introduction

In the last decades, Sensor Networks (SNs) have played a primary role for the research community. However, the progress focused more on communication

Riccardo Petrolo, Valeria Loscrì, Nathalie Mitton
Inria Lille - Nord Europe, Villeneuve d'Ascq, FRANCE
E-mail: firstname.lastname@inria.fr

Rorberto Morabito
Ericsson Research NomadicLab, Jorvas, FINLAND
E-mail: roberto.morabito@ericsson.com

This is the author's pre-print version. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising, promotion or for creating new collective works for resale or for redistribution to thirds must be obtained from the camera-ready copyright owner.

and networking aspects between such devices rather than on the interpretation of the produced data. Very recently, there has been an effort in order to understand data which comes out and goes into devices; those enhancements together with the Cloud computing are at the basis of the Cloud of Things (CoT) that we previously proposed in [24]. The CoT aims to better use distributed resources, putting them together and enabling therefore the horizontal integration of various Internet of Things (IoT) platforms. Similar concepts to CoT, such as Capillary Networks [27], and Fog Networks [2] include in their topology, the presence of intelligent nodes that, even with limited computation resources, can easily manage the assigned applications. Adopting this approach, in this paper, we design the architecture of a gateway for the Cloud of Things. The gateway is able to manage semantic-like things and to act as an end-point for the dynamic presentation of real world data to consumer applications and users. Specifically, in our proposal, we design a gateway through the use of virtualized software and, in particular, by exploiting all the benefits introduced by emerging lightweight virtualization technologies. As exhaustively explained in [20], those technologies introduce an almost negligible overhead and they are modeled in such a way to guarantee a lightweight and dense deployment of services. Lightweight technologies are indeed, not designed to be exclusively used in large data-centers or in specific Cloud environments, they are equally efficient, even when operating on gateway and/or embedded systems. Container-based solutions implement processes isolation by avoiding the emulation of virtual hardware and therefore reducing the overhead that characterizes other virtualization technologies such as hypervisors. The advantage of achieving higher density of virtualized processes - which run within each container - is, then, a direct consequence. With specific relation to our scenario, the versatility of container technologies allows a dynamic and optimized usage of the gateway, so that services can be allocated only when needed and according to the functionality of the SN. Results demonstrate a negligible impact in terms of performance when using container technologies. Anyway it is considered acceptable if compared to the benefits introduced, i.e., fast and efficient building process, high density of lightweight virtualized instances, and application isolation.

In conclusion, we show that the combination of all these features can represent a valuable way to convey several advantages - both sensors and gateway sides - and noticeable improvements in terms of resource allocation, service management, and energy efficiency; we also discuss different ways of future investigations.

The rest of the paper is organized as follows: Related Work is widely presented in Section 2. Section 3 surveys the Internet of Things evolution towards the Cloud of Things. In Section 4, we highlight the concept of distributed Cloud, introducing lightweight virtualization technologies. The technical architecture of the gateway, use cases, performance evaluation, and future hints are discussed in Section 5. Section 6 concludes the paper.

2 Related Work

Current literature presents several proposals of network architecture, in which the convergence between Cloud and IoT represents the key factor to enable large-scale IoT systems. Many of those solutions require the presence of a gateway able to provide specific interface functionality between backbone network - Cloud or in general the Internet - and the Sensor Network. In [13], Gubbi et al. present a Cloud-centric vision for IoT networks; the authors consider an Internet centric environment, in which all the services are in support of data produced by the sensors. In the proposed architecture, the presence of several Internet gateways is expected. The function of such devices is exclusively to provide communication between the sink node and the outside world through the Internet. Another function performed by the gateway is to enable an efficient addressing scheme. Indeed, one important issue that can be faced in IoT networks is due to the different addressing scheme used by the sensors to respect other network entities. To address this problem, the association of a Uniform Resource Name (URN) to a gateway is proposed.

In [6], authors 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. Even in this case, the gateway is designed to erase the existing heterogeneity between sensing domain and network. 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.

In [3], Chen et al. list a set of requirements and common features that an IoT gateway must include. In particular, it has to act as a proxy, which interconnect 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.

The design of the gateway proposed in [12] is characterized by the use of the Host Identify Protocol (HIP) that allows global addressing of all the objects connected in the sensing domain, while maintaining the use of IPv4 addresses. The mechanism is structured in such a way to allocate a single public IP-address to a large group of sensors devices, which are in turn under control of a single HIT gateway. Orchestration mechanisms to favorite the interaction between different HIT gateways and other functionality are proposed.

The paper [8] presents a Cloud Computing based platform for a specific use case: the management of mobile and wearable health-care sensors. In this particular scenario, the role of the gateway is to collect all the inputs coming

from the sensors and to forward them to the Internet. The authors make use of a mobile phone to perform all the gateway operations. Moreover, the gateway uses a set of REST web services ot transmit the sensed data to the Cloud. Similar solutions can be found in [11] and in [5].

Merlino et al. in [19], propose a Cloud-oriented environment in order to integrate IoT paradigms and resource ecosystems in a Smart City scenario; authors use OpenStack [1] as Cloud solution infrastructure.

In [14], 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.

Gyrard et al. in [15], introduce a method to integrate a semantic engine in IoT contexts by taking into account standardizations efforts in M2M environments. Semantic rules are integrated in the main actors who characterize the network architecture such as cloud, end-point devices, and M2M gateways.

Fog computing is another network paradigm, which aims to extend the traditional Cloud computing operations to the underlying network, with a special direction for IoT network. In [2], we can find a set of guidelines to enable Fog computing networks. The main objective is provide several services such as computation, storange, and networking on IoT nodes. Contrarily to the Cloud paradigm, which is usually strictly centralized, Fog computing provides all the services in a distributed way. Strictly related to the Fog Computing concepts, Farris et al. present an hybrid approach, where Fog computing features are used to support a dynamic cloud cooperation of mobile IoT devices [9]. By introducing the Mobile-IoT-Federation-as-a-Service (MI- FaaS) paradigm, the authors define a topology in which edge nodes operate as orchestrators to manage federations among public/private IoT clouds, in where IoT applications are integrated.

Recently, the European Telecommunications Standards Institute (ETSI) has introduced a new network architecture model named as Mobile-Edge Computing. According to [21], Mobile Edge Computing enables cloud-computing capabilities at the edge of the cellular network and close to the endpoint device itself. Similarly to the concepts afore mentioned, the idea behind this new paradigm is that moving part of the processing tasks closer to the device can bring several benefits by enhancing the performance with ultra-low latency and high bandwidth.

To summarize, although the presence of a gateway as an interface between sensor domains and backbone networks is provided in several of the analyzed architectures, its functionalities are often limited to traffic forwarding, and protocol conversion. The architecture proposed in this work includes further features that take advantage of the lightweight and versatile container-based virtualization.

[1] `https://www.openstack.org`

3 The Internet of Things evolution

In 1999 Kevin Ashton introduced for the first time the term "Internet of Things" (IoT) [1]. Some years later, the International Telecommunication Union (ITU) formally defined the IoT [26] as a technology capable to *connect anyone from anyplace and anytime to anythings* (Figure 1). Since then, the IoT starts to attract the attention of both academia and industry, thanks to its capability to create a world where all the *objects* around us are connected together and to the Internet with minimum human intervention, making possible the development of a huge number of applications in different domains [24] - e.g., home automation, smart cities, logistic, smart agriculture, eHealth, and so on -.

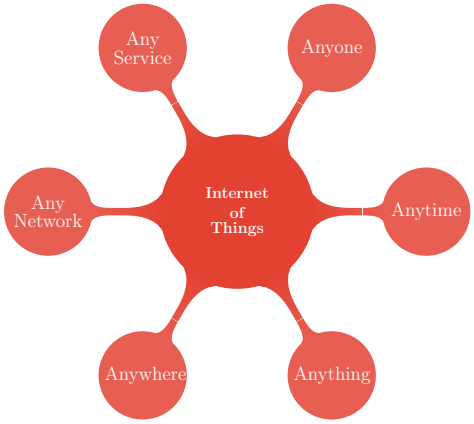


Fig. 1: The Internet of Things.

In the IoT, *objects* plays a key role. Data is indeed collected using sensors; then it is processed and decisions are made and performed by actuators. It is therefore important to highlight, as also stated in [22], that the IoT cannot exist without Sensor Network (SN); SN indeed provides the majority of hardware infrastructure, through providing access to sensors and actuators.

The concept of SN was existing a long time before the IoT was introduced, however, Sensor Networks were used in limited domains to achieve specific purposes (e.g., environment monitoring, event detection, and so on). In the last years, researchers have focused a lot on the field, making significant progress in order to address challenges such as resource constraints, dynamic topology, scalability, security, etc. However, the development focused more on communication and networking aspects between such devices (e.g., 6LoWPAN [17], RPL [10], etc.) rather than on the interpretation of the produced data. Just recently, with standards and common frameworks such as Sensor Web Enable-

ment (SWE) and W3C's sensor ontology [4], there is an effort to understand data, which comes out, and goes into sensor networks.

As stated by Diaz et al. in [7], semantic technologies, such as ontologies and linked data, can be used to solve the problem of scalability - many different devices can be added to the system at a fast pace, by formatting the information -. Interoperability is made easier as well: the data format provided by ontologies makes services equally available; regardless of the devices they are being provided.

In order to manage those sensors and to semantically annotate them and their data, different IoT platforms have been introduced in literature. Below, we describe two initiatives without pretending to be exhaustive:

- **GSN** [2] (Global Sensor Network) is a java middleware that integrates sensor data and distributed query processing. Anyway it does not consider the use of Cloud and semantic technologies.
- **OpenIoT** [3] is a Cloud-based middleware infrastructure capable to deliver on-demand access to IoT services issued from multiple platforms. It is not designed to run on resource constrained devices and therefore it is not suitable for our context.

OpenIoT, and many other solutions in literature are Cloud-based. In the last decade, indeed, Cloud technology has become very popular thanks to its capability to transform *everything* in service. Also the IoT landscape comes under this "revolution"; in [24] we revisited the concept of IoT, defining the **Cloud of Things** (CoT) as an evolution in which sensors and their observations can be seen as a service (Sensing as a Service [23]); we also presented the VITAL platform as a CoT architecture. VITAL [4] project is a FP7 European project, which aims to integrate Internet-Connected Objects (ICOs) among multiple IoT platforms and ecosystems. To this end, mechanisms to abstract, virtualize, and manage things are developed.

In the next section, we highlight the evolution of the Cloud technology towards distributed Cloud, introducing the emerging lightweight virtualization technologies and the "containerization" concept.

4 Container-based Virtualization for a distributed Cloud

In the context of the Cloud of Things, data-centers can play an important role regarding off-line analysis of large amount of data. To accomplish real-time operations - specially on small time-scale - different actions are required. From this point of view, a distributed Cloud can meet all the requirements of such scenarios, and address numerous issues. In the distributed Cloud [18], the services are not only able to be run in a data-center, but also close to the device itself, for example in the IoT gateways or in the radio base stations. The

[2] `https://github.com/LSIR/gsn`

[3] `http://www.openiot.eu`

[4] `http://vital-iot.eu`

amount of data produced by sensor networks can reach several gigabytes; this data, is affected by a discrete level of redundancy, which has to be reduced by means of operations like filtering, compressing, and other real-time processing operations. Additionally, based on data analysis, there might need to perform actions on the devices. As stated in [18], those operations *"should be done as close to the data source as possible, preferably already on the capillary gateway before data is sent over the cellular uplink. Each application requires its own way of performing compression, filtering and aggregation."*

Another important aspect, which encourages the adoption of a distributed Cloud, is due to the fact that a bad network performance can become the bottleneck for the whole system. For example, particular IoT services, such as high-bandwidth sensors (e.g., cameras), might have very strict latency requirements. A centralized Cloud would make these IoT services much more dependent on latency and delay issues without ensuring optimal performances. On the other hand, moving part of the computation closer to the sensors and enabling ubiquitous computation, would bring several benefits. In most of the cases, all the operations described before have to be instantiated only temporarily, within a short period of time, and in an efficient way.

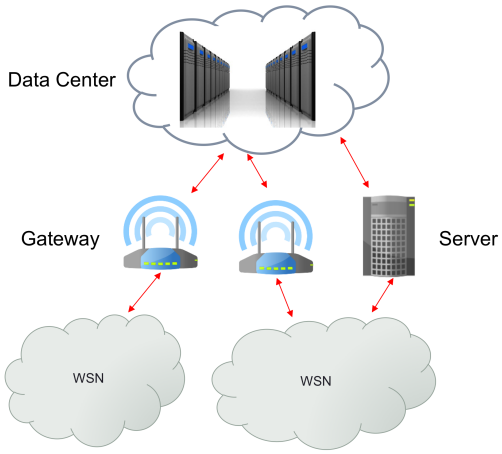


Fig. 2: Distributed Cloud topology.

The entities that characterize a distributed Cloud topology (Figure 2) may have different requirements based on hardware capabilities. For example, a gateway is less powerful than a server in terms of processing power. Therefore, emerging lightweight virtualization technologies - such as containers - can introduce several benefits, which match with the following requisites: *fast initialization, low overhead, and good energy efficiency.* Traditional hypervisor-based virtual machines may not be the most appropriate means to provide processes virtualization in low-power consumption devices. Indeed, hypervisor-based virtualization operates at hardware level and a full operating system is installed

on a virtual machine. Moreover, the emulation of virtual hardware introduces more overhead, producing an hypervisor-based image substantially larger and an increasing use of system resources.

Container-based virtualization can be considered a lightweight alternative to hypervisor-based virtualization. Containers implement isolation of processes at the operating system level, avoiding then the overhead due to virtualized hardware and virtual device drivers. The overhead it is therefore smaller compared to the one introduced by other virtualization alternatives.

This overhead allows the running of multiple containers, even in resource constrained devices such as the one used in the IoT landscape. Despite they share the same operating system, each running container operates with independent characteristics: independent virtual network interfaces, independent process spaces, and a separate file system. Container-based virtualization architecture is showed in Figure 3.

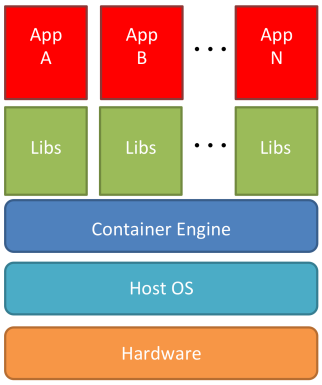


Fig. 3: Container-based virtualization architecture.

The concept of "containerization" is not new in virtualization, but it has achieved much more relevance and practical use recently with the advent of Docker [5]. In a Docker container, one or more processes/applications can run simultaneously. Moreover, an application can be designed to work in multiple containers, and to interact with others.

Three main components characterize the Docker architecture [6]:

- **Docker images** defined as a read-only templates. These templates represent the basic entity from which Docker containers are created. Building new images or updating existing ones is very simple by means of the Docker APIs.
- **Docker registries** are public or private repositories where the images are stored. From this registries, it is possible to upload or download images.

[5] `https://www.docker.com/`

[6] `http://docs.docker.com/introduction/understanding-docker/`

– **Docker containers** are the run components of Docker. Containers are created from Docker images; several operations can be performed on Docker containers: running, starting, stopping, moving, and deleting.

A container uses a basic image stored in a specific local or remote registry. When a container is executed, the configuration information about the application running is specified together with any other dependency (e.g., libraries). While running a container from an image, Docker uses the UnionFS (Union File System) 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.

This layered approach introduces several advantages:

1. Smaller disk image.
2. Higher density of virtualized instances.
3. Fast and efficient building process.

Such features make possible the integration of the functionality of containers in a wide-range of contexts: smart devices, Cloud, embedded systems.

5 Design of the gateway

As shown in Figure 4, the gateway is in charge to gather and manage data produced by sensors, and at the same time, it is enhanced to act as an endpoint for the communication with Cloud data-centers or with other local devices.

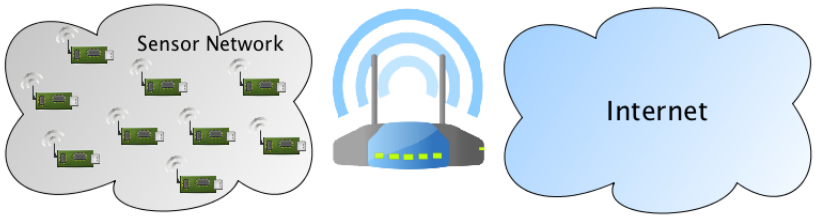


Fig. 4: Network topology.

In order to communicates with the Wireless Sensors Network, our gateway uses the IEEE 802.15.4 protocol [16]. We assume that each sensor uses a standardized format (e.g., IPSO [28]) for describing its services (i.e., temperature, light, humidity, etc.). At the same time, we also suppose that a mechanism for the neighbor discovery is running; then, with a frequency f, each sensor advertises its services. Thanks to this operation (detailed in [25]), the gateway will be able to build its Neighbor Table, in which it will store information such as service name, timestamp of the last frame received from that neighbor, and connectivity statistics (i.e., RSSI). Those information are stored into

a database and semantically annotated according to an extended version of the W3C SSN ontology [4]. At the same time, the gateway exposes a RESTful endpoint, which allows sensors data to be queried.

The interactions of the gateway with sensors and/or other entities (Data Centers, local servers, etc.) are multiple and may occur depending on the particular use case. Along with the traditional routing and forwarding functionality, our gateway is able to add further services by supporting more applications. Within the IoT scenarios, the services have to be instantiated only temporarily and in an efficient way, for example because of the strict latency requirements of some IoT application.

Therefore, introducing containers in such environment allows to have a system that benefits from all the features explained in the previous section - fast instantiation and initialization and high density of application/services due to the small container images size -. Furthermore, from the users perspective, containers can be the mean for customizing a specific platform - the gateway in our case - in line with the specific use cases.

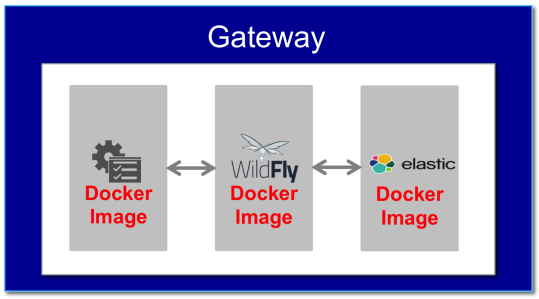


Fig. 5: Docker images on the gateway.

Before going through the analysis of the operations performed by the gateway, we provide further details about the implementation. In particular, we specify the main components virtualized with Docker containers and stored as Docker images in the gateway itself.

As shown in Figure 5, our proposal is characterized of three main components:

- A web server used to expose services to the Internet. We chose **WildFly** [7] an application server written in Java, which runs on multiple platforms.
- A search server in which all sensor data is stored. We chose **Elasticsearch** [8] because it provides a distributed search engine with an HTTP web interface and schema-free JSON documents.
- A third component that serves as orchestrator among all the components.

[7] `http://wildfly.org`
[8] `https://www.elastic.co`

The aforementioned parts are *dockerized*, so as to ensure an isolated environment. The main benefits introduced by the virtualization of these components will be clearer from the analysis of the different use cases presented below.

5.1 Gateway interactions

Below, we present a subset of possible interactions between the entities that define our scenario. The analysis suggests how the dynamic allocation of services in the gateway - by means of containers - brings several benefits from the energy perspective. All the container instances running on the gateway (e.g. databases) can be dynamically allocated when required, without being constantly active.

Moreover, we have seen all the potentialities of our design regarding backup functionalities and capacity of storing the network status at given times.

5.1.1 Sensors - Gateway

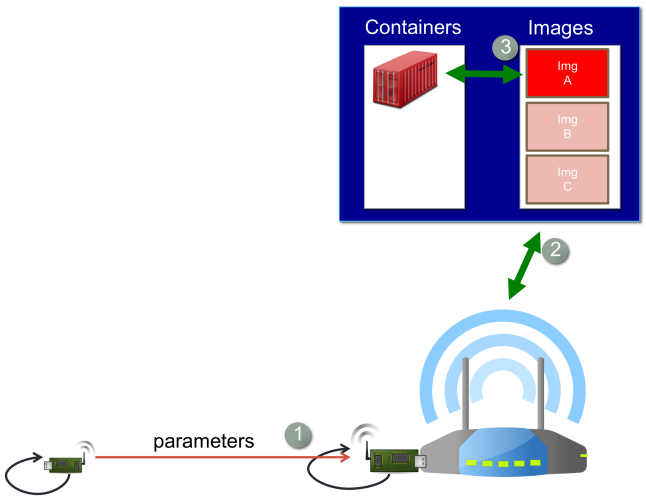


Fig. 6: Sensors - Gateway interaction.

We analyze now the simple case of a sensor that wants to transmit its updated parameters to the gateway (Fig. 6). In this case, a database containing the latest update data is stored within a cover Docker image. When the gateway receives updated information by the sensor, a Docker container is launched and the new values are stored into the database. Once this transaction is completed, the updated Docker image is saved locally in the gateway.

5.1.2 User - Gateway

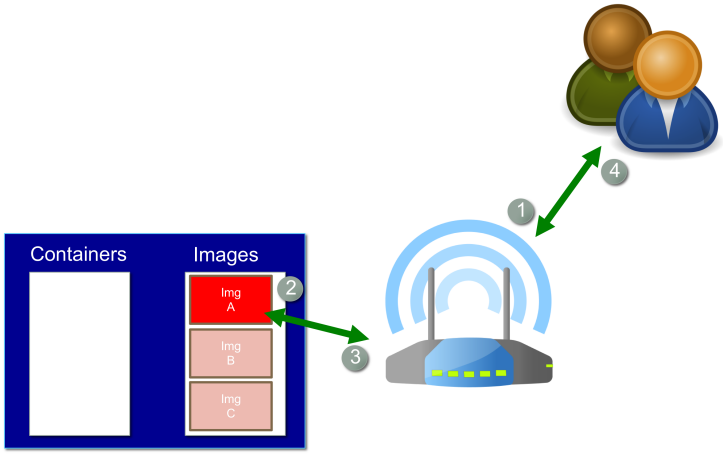


Fig. 7: User - Gateway interaction.

This second case (Fig. 7) can be considered closely related to the first; users, remotely connected, want to access data stored in the database. In this scenario, the users can easily download the latest updated Docker image - which is stored on the gateway - and then have access to data. If the image download is not strictly necessary at the user level, data can be read on the gateway - from the Docker image directly -. Contrarily to the previous case, no containers are started in the gateway, avoiding further processing operations.

5.1.3 Sensors - User

The last case (Fig. 8), describes a requested started user-side. For example, a user wants to know the temperature measured in the "Room 105". Thanks to the semantic annotation, the gateway knows directly to who asking for the information, and it forwards the request to the specific node.

5.2 Hardware

We choose a Raspberry Pi 2 [9] to act as the gateway (Figure 9b). It is equipped with a quad-core ARM Cortex-A7 CPU, with 1 GB of RAM. Furthermore, it has 4 USB ports, 1 Ethernet port, and 40 GPIO pins. One important reason that has led us to use a Raspberry Pi 2 as a gateway is the possibility to run containers on top of it[10].

[9] `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`
[10] `http://resin.io/blog/docker-on-raspberry-pi/`

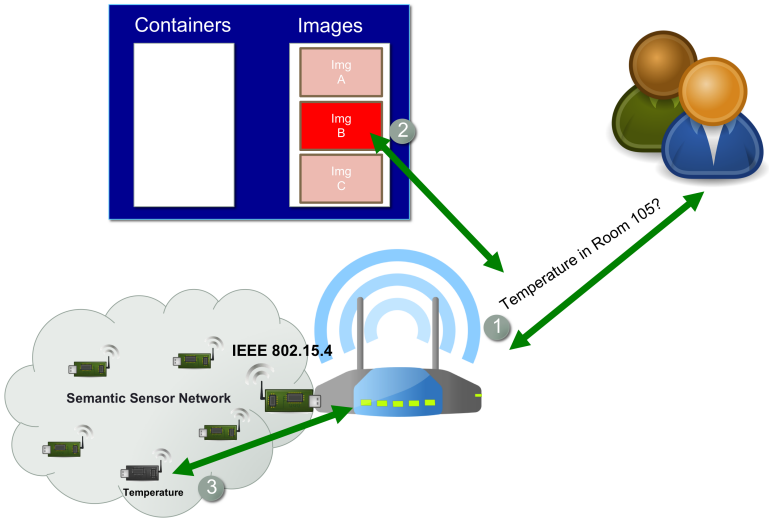
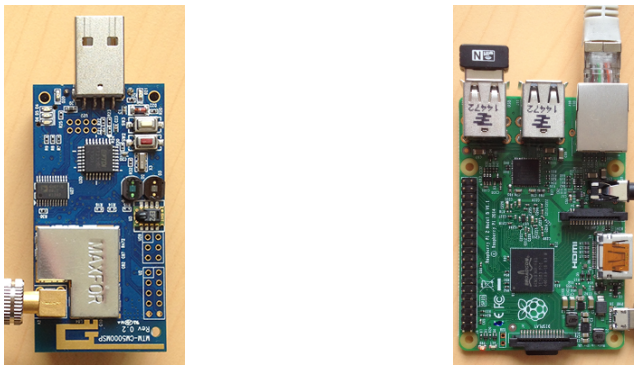


Fig. 8: Sensors - User interaction.



(a) Sensor. (b) Gateway.

Fig. 9: Hardware structure of network nodes.

As sensor node, we used a Maxfor node (Figure 9a). As shown in Table 1, it embeds a MSP430 as the CPU and CC2420 as the radio frequency module; moreover, this node is integrated with temperature and light sensors.

5.3 Evaluation

The validation of our proposal covers two different aspects. First, we evaluate how a Raspberry Pi 2 reacts - in terms of performance - to specific workloads generated by applications running within Docker containers. Then, we present

Table 1: Sensor node specifications.

Parameter	*Specification*
CPU	MSP430
RF Chip	Texas Instruments ®CC2420
Frequency Band	2.4 GHz − 2.485 GHz
Transfer Rate	250 Kbps
RF Power	-25 dBm − 0 dBm
Range	120m [outdoor]; 20-30m [indoor]
Sensor Temperature & Humidity	Sensirion ®SHT11
Sensor Light	600 nm peak

a first performance evaluation of our platform while performing very simple tasks (e.g., running the web server).

5.3.1 General Performance

In the first analysis, we use benchmark tools built to generate different types of workloads capable to challenge a specific hardware segment in the Raspberry Pi 2. In order to estimate the overhead produced by the presence of a virtualization layer, we tested *CPU*, *Memory*, and *Disk I/O*. The *native performance* - i.e. running the benchmark tool without including any virtualization layer - is used as a reference for comparison.

Table 2: Benchmark results.

	CPU	**Memory**			**Disk I/O**	
	Execution Time (seconds)	*memcpy (MiB/s)*	*dumb (MiB/s)*	*mcblock (MiB/s)*	*Read (MB)*	*Write (MB)*
Native	434.074	598.22	70.93	601.55	123.25	82.172
Docker	446	562.05	70.43	570.51	107.062	74.719

Table 2 shows the results of the benchmark analysis. The sysbench [11] tool is used to test CPU and Disk I/O performance. The results of the CPU test demonstrate an existing difference between the native and the Docker case. However, we can observe how the container engine introduces a negligible impact in terms of CPU performance, with a percentage difference in the order of 2.67%. To test the Memory I/O performance, we use the Unix command *mbw* [12], which determines the available memory bandwidth by copying large arrays of data in memory, and performing three different tests (memcpy, dumb, and mcblock). Similarly to the CPU case, native and container performance can be considered comparable, with a max percentage difference of 6.04% during the memcpy test. In the Disk I/O evaluation, the benchmark is set in

[11] `http://manpages.ubuntu.com/manpages/wily/en/man1/sysbench.1.html`
[12] `http://manpages.ubuntu.com/manpages/wily/en/man1/mbw.1.html`

order to execute random read/write operation. A performance degradation of Docker compared to the native case can be observed from the results. This difference remains in the order of roughly 10%.

5.3.2 Gateway Performance

The second evaluation aims to compare the performance of our gateway while operating in two different configurations: *(i)* when the components of the gateway are virtualized by using Docker container, *(ii)* when the components are running on top of the operating system, without any virtualized application (Figure 10).

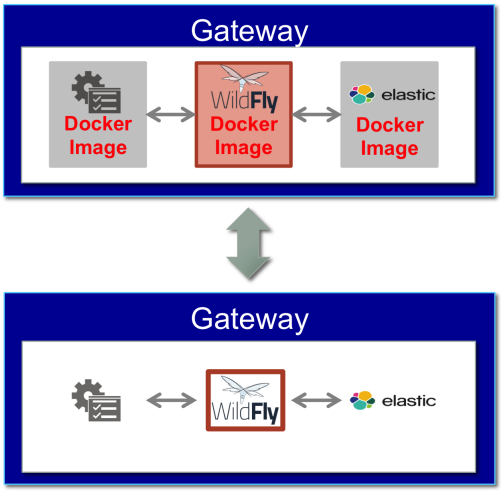


Fig. 10: Docker virtualization vs. No virtualization.

We have been monitoring the two setup - running on two different Raspberry Pi 2 - for 24 hours. Only the performance of the web-server in idle state have been analyzed.

As shown in Figure 11, the *Memory footprint* is slightly bigger when the single components are virtualized. Such result is coherent whit the analysis reported in the aforementioned section, and it indicates a negligible impact of the virtualization layer in terms of memory performance. Nevertheless, it is interesting to observe that the performance of the single web server are better when using Docker container. Although further investigation are needed, this result is due to the optimized configurations of the Docker environment, which allows to save in terms on used memory.

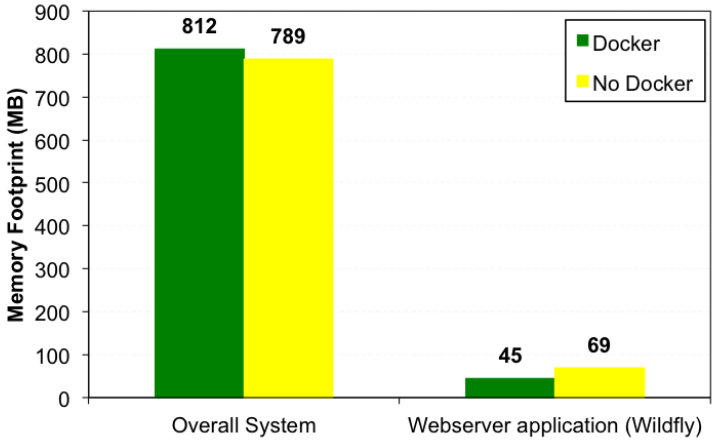


Fig. 11: Memory footprint performance (average).

5.4 Future Work

As next achievement, we aim to connect our gateway to the VITAL platform. As initially described, VITAL's goal is to federate multiple IoT platforms by using semantic web technologies. As shown in Figure 12, in order to be integrated into VITAL, each IoT system should expose a Platform Provider Interface (PPI). This is defined as a set of RESTful web services used by VITAL to be able to retrieve:

- information about the system;
- information about the services exposed by the system;
- information about the sensors managed by the system;
- observations made by the sensors that the system manages.

Moreover, we aim to investigate more sophisticated techniques in order to better manage the virtualization of processes on the gateway itself.

6 Conclusion

In this paper we have proposed the design of a gateway for the Cloud of Things. CoT is a recent paradigm which aims to better use distributed resources by putting them together and enabling the horizontal integration of various Internet of Things platforms and applications. The proposed gateway takes advantages by the emerging virtualization technologies which are modeled to guarantee a lightweight and dense deployment of services.

We also provided information about the hardware we used and we illustrated some interactions between the gateway and different actors of our scenario. In the future we plan to connect our gateway to the VITAL platform

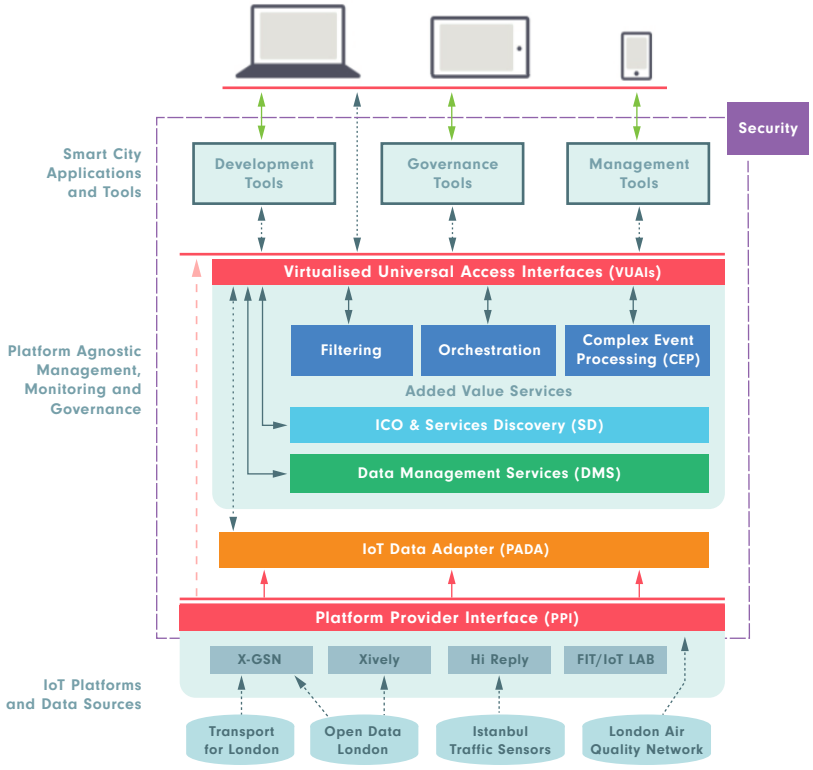


Fig. 12: VITAL architecture.

and to adopt prediction algorithms on data production between the gateway and sensors in order to reduce the number of communications between them, and therefore, to lower the battery consumption and interference.

We expect that the proposed design may help the development of new Internet of Things applications, while pointing out research directions and solutions.

Acknowledgements This work is partially supported by CPER Nord-Pas-de-Calais/DATA, by the European Community in the framework of the VITAL FP7 project under contract number FP7-SMARTCITIES-608662, and by the FP7 Marie Curie Initial Training Network (ITN) METRICS project (grant agreement No. 607728).

References

1. Ashton, K.: That 'Internet of Things' Thing. RFiD Journal (2009)
2. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of MCC - 1st Workshop on Mobile Cloud Computing, p. 13. Helsinki, Finland (2012)
3. Chen, H., Jia, X., Li, H.: A brief introduction to iot gateway. In: Proceedings of ICCTA - International Conference on Communication Technology and Application, pp. 610–613 (2011)
4. Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Le Phuoc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web **17**, 25–32 (2012)
5. Costantino, L., Buonaccorsi, N., Cicconetti, C., Mambrini, R.: Performance analysis of an lte gateway for the iot. In: Proceedings of WoWMoM - International IEEE Symposium on World of Wireless, Mobile and Multimedia Networks, pp. 1–6 (2012)
6. Datta, S.K., Bonnet, C., Nikaein, N.: An iot gateway centric architecture to provide novel m2m services. In: Proceedings of IEEE World Forum on Internet of Things (WF-IoT), pp. 514–519 (2014)
7. Diaz, H.D., Ortega, J.F.M., García, A.C., Rodríguez Molina, J., Cifuentes, G.R., Jara, A.: Semantic as an interoperability Enabler in Internet of Things. In: Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems, pp. 315–342 (2013)
8. Doukas, C., Maglogiannis, I.: Bringing iot and cloud computing towards pervasive healthcare. In: Proceedings of IMIS - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 922–926 (2012)
9. Farris, I., Militano, L., Nitti, M., Atzori, L., Iera, A.: Federated edge-assisted mobile clouds for service provisioning in heterogeneous iot environments. In: Proceedings of WF-IoT - 2nd IEEE World Forum on Internet of Things. Milan, Italy (2015)
10. Gaddour, O., Koubâa, A.: RPL in a nutshell: A survey (2012). DOI 10.1016/j.comnet.2012.06.016
11. Golchay, R., Mouël, F.L., Frénot, S., Ponge, J.: Towards bridging iot and cloud services: proposing smartphones as mobile and autonomic service gateways (2011)
12. Gronbaek, I.: Architecture for the internet of things (iot): Api and interconnect. In: Proceedings of SENSORCOMM - 2nd International Conference on Sensor Technologies and Applications, pp. 802–807 (2008)
13. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future Generation Computer Systems **29**(7), 1645–1660 (2013)
14. Guoqiang, S., Yanming, C., Chao, Z., Yanxu, Z.: Design and Implementation of a Smart IoT Gateway. In: Proceeding of GreenCom and iThings/CPSCom - International IEEE Conference on Green Computing and Communications, and International IEEE Conference on Internet of Things and Cyber Physical Social Computing, pp. 720–723. Beijing, China (2013)
15. Gyrard, A., Datta, S.K., Bonnet, C., Boudaoud, K.: A semantic engine for internet of things: cloud, mobile devices and gateways. In: Proceedings of IMIS - 9th International IEEE Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. Santa Cantarina, Brazil (2015)
16. IEEE 802.15 WPAN Task Group 4 (TG4): IEEE 802.15.4-2006. Available at: `http://www.ieee802.org/15/pub/TG4.html`
17. Kushalnagar, N., Montenegro, G., Culler, D.E., Hui, J.W.: Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944 (Proposed Standard). Tech. rep., IETF (2007)
18. Melen, J., Beijar, N., Jimnez, J.: Distributed cloud for capillary networks. Tech. rep. (2015). URL `http://http://www.ericsson.com/research-blog/5g/distributed-cloud-for-capillary-networks/`

19. Merlino, G., Bruneo, D., Distefano, S., Longo, F., Puliafito, A.: Stack4Things: Integrating IoT with OpenStack in a Smart City context. In: Proceeding of SMARTCOMP - International Workshop on Smart Computing Workshops, pp. 21–28. IEEE, Hong Kong, China (2014)
20. Morabito, R., Kjällman, J., Komu, M.: Hypervisors vs. Lightweight Virtualization: a Performance Comparison. In: Proceedings of IC2E - International IEEE Conference on Cloud Engineering. Tempe, Arizona, US (2015)
21. Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A., et al.: Mobile-edge computing introductory technical white paper. Mobile-edge Computing (MEC) industry initiative (2014)
22. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Context Aware Computing for The Internet of Things: A Survey. IEEE Communications Surveys & Tutorials **16**(1), 414–454 (2014)
23. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Sensing as a service model for smart cities supported by Internet of Things. European Transactions on Telecommunications **25**(1), 81–93 (2014)
24. Petrolo, R., Loscrì, V., Mitton, N.: Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. Transactions on Emerging Telecommunications Technologies (2015). DOI 10.1002/ett.2931
25. Petrolo, R., Loscri, V., Mitton, N.: Confident-based Adaptable Connected objects discovery to HArmonize smart City Applications. In: Proceedings of IFIP WD - Wireless Days. Toulouse, France (2016)
26. Reports, I.I.: The Internet of Things. Tech. rep. (2005)
27. Sachs, J., Beijar, N., Elmdahl, P., Melen, J., Militano, F., Salmela, P.: Capillary networks a smart way to get things connected - Ericsson. Tech. rep. (2014). URL `http://www.ericsson.com/news/140908-capillary-networks_244099436_c`
28. Shelby, Z., Chauvenet, C.: The IPSO Application Framework draft-ipso-app-framework-04. Tech. rep. (2012). URL `http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf`