Exploring Container Virtualization in IoT Clouds

Antonio Celesti, Davide Mulfari, Maria Fazio, Massimo Villari and Antonio Puliafito
Department of Engineering, University of Messina
Contrada Di Dio - 98166 Messina, Italy
Email: {acelesti, dmulfari, mfazio, mvillari, apuliafito}@unime.it

***Abstract*—The advent of both Cloud computing and Internet of Things (IoT) is changing the way of conceiving information and communication systems. Generally, we talk about IoT Cloud to indicate a new type of distributed system consisting of a set of smart objects, e.g., single board computers running Linux-based operating systems, interconnected with a remote Cloud infrastructure, platform, or software through the Internet and able to provide IoT as a Service (IoTaaS). In this context, container-based virtualization is a lightweight alternative to the hypervisor-based approach that can be adopted on smart objects, for enhancing the IoT Cloud service provisioning. In particular, considering different IoT application scenarios, container-based virtualization allows IoT Cloud providers to deploy and customize in a flexible fashion pieces of software on smart objects. In this paper, we explore the container-based virtualization on smart objects in the perspective of a IoT Cloud scenarios analyzing its advantages and performances.**

***Index Terms*—Cloud Computing, IoT, Virtualization, Container.**

I. Introduction

The combination between Cloud computing and Internet of Things (IoT) technologies is pursuing new opportunities in delivering services, representing a strategic approach for IT operators of increasing their business. The emerging business perspectives coming from IoT are pushing private, public, and hybrid Cloud providers to integrate their system with embedded devices (including sensors and actuators) in order to provide together with the traditional Infrastructure, Platform and Software as a Services (IaaS, PaaS, SaaS) even a new type of service level, that is called *IoT as a Service* (IoTaaS). As a consequence, new types of providers that combine Cloud computing solutions with IoT are rising. We talk about *IoT Cloud* to indicate a new type of distributed system consisting of a set of IoT devices interconnected with a remote Cloud infrastructure, platform, or software through the Internet able to provide IoTaaS [1].

Resource virtualization is one of the key concepts in cloud computing. Nowadays, several virtualization technologies are available. One of the most popular technologies is the hypervisor virtualization, which requires a virtual machine monitor (VMM) software module on top of a host OS that provides a full abstraction of Virtual Machines (VMs). Recently, a lightweight alternative to the hypervisors is the container-based virtualization, also known as Operating System (OS) Level virtualization. This kind of virtualization partitions the physical machine resources, creating multiple isolated user-space instances. We think that such a technology can revolutionize future IoT Cloud system, bringing the same advantages that the hypervisor virtualization has brought to traditional Cloud system in terms of resource management. In fact, container virtualization can allow IoT Cloud operators to instantiate, relocate, and optimize virtual IoT capabilities in order to control in a more flexible fashion IoTaaS.

In this paper, we explore the container virtualization technology in IoT devices analyzing advantages and performance. In particular, we discuss benefits in adopting virtualization techniques in IoT scenarios both in terms of cloud service management and business opportunities. Then, we describe our experimentation on container virtualization based on a well know embedded device, that is a Raspberry Pi. We investigate the response of the device when no container and several container runs according to several configurations that could be adopted in real use cases, such as a smart mall.

The rest of the paper is organized as follows. Section II describes related works. After an overview on container virtualization techniques provided in Section III, we discuss in Section IV the benefits of adopting such technology in IoT Cloud systems. In Section V, we discuss how can be possible to leverage the container virtualization on Linux-based Single Board Computers (SBCs) considering an IoT device architecture based on the Raspberry Pi model [2] and the Docker container engine. In Section VI, we analyze the overhead introduced by the container virtualization in our experimentation. Section VII concludes the paper.

II. Related Work

The interest of the research community in Cloud based solutions for IoT is proved by the numerous activities in this research field. Nowadays, container-based virtualization presents an interesting alternative to virtual machines in the cloud [3]. Although the concepts underlying containers such as namespaces are very mature, only recently containers have been adopted and standardized in mainstream operating systems, leading to a renaissance in the use of containers to provide isolation and resource control. Linux is the preferred operating system for the cloud due to its zero price, large ecosystem, good hardware support, good performance, and reliability. The kernel namespaces feature needed to implement containers in Linux has only become mature in the last few years since it was first discussed in 2006 [4]. Several articles have focused on container based virtualization technologies by considering cloud computing scenarios. Docker [5] is a lightweight virtualization based on Linux Containers (LXC)

978-1-5090-0898-8/16/$31.00 ©2016 IEEE

that can completely encapsulate an application and its dependencies within a virtual container. In [6], the authors discuss the design and the implementation of cloud system based on Docker, especially intended for a PaaS platform. As motivated in [7], Docker has been deployed within a platform for bioinformatics computing that exploits advanced cloud services. T. Bui investigates the security level of Docker by considering two main areas: (1) the inter-nal security of Docker, and (2) how Docker interacts with the security features of the Linux kernel, such as SELinux and AppArmor, in order to harden the host system. The proposed analysis shows that Docker provides a high level of isolation and resource limiting for its containers using namespaces, cgroups, and its copy-on-write file system, even with the default configuration. It also supports several kernel security features, which help to hardening the security of the host [8].

III. Container Virtualization Overview

Resource virtualization is one of the key concepts in Cloud computing and it refers to the act of creating a virtual (rather than actual) version of something, including but not limited to a virtual computer hardware platform, Operating System (OS), storage device, or computer network resources. Considering such scenarios, virtualization makes use of an intermediate software layer on top of a system in order to provide abstraction of multiple virtual resources. Such virtual resurces are software components known as Virtual Machines (VMs) and they can be viewed as isolated execution contexts.

Nowadays, several virtualization techniques are available. One of the most popular is the hypervisor based virtualization, which requires a Virtual Machine Monitor (VMM) running on top of a host OS that provides a full abstraction of VM. In this case, each VM has its own OS that executes completely isolated from the others. This enables us to execute multiple different operating systems on a single actual host. Examples of Hypervisor Virtualization (HVV) tools include Xen, KVM, Virtual Box, Virtual PC, WM-Ware.

Recently, a lightweight alternative to the hypervisors is the container-based virtualization, also known as Operating System Level virtualization. This kind of virtualization partitions the physical machines resources, creating multiple isolated user-space instances [9].

Figure 1 depicts the key difference between the introduced virtualization technologies. While the hypervisor based virtualization provides abstraction for full guest OSs (one per VM), the container based virtualization works at the OS level, providing abstractions directly for the guest processes. In essence, hypervisor solutions work at the hardware abstraction level and containers operate at the system call layer.

As motivated in [9], all the containers share a single operating system kernel; so the container based virtualization is supposed to have a weaker isolation when compared to hypervisor based virtualization. However, from the point of view of the users, each container looks and executes exactly like a stand-alone OS. Additionally, in a Cloud computing scenario,

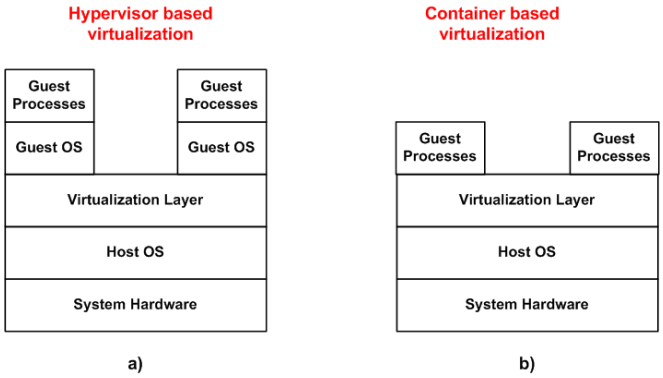

Fig. 1. a) Hypervisor and b) Container based virtualization.

developers can deploy an higher number of containers than VMs on the same physical host.

Recent technological developments have allowed container-based virtualization technology to support SBC (Single Board Computer) devices equipped with a modern Linux kernel supporting a Linux Container Virtualization (LCV) layer. In these scenarios, container based software seems to be an interesting approach to deploy and to customize software applications running on a SBC. More specifically, in the present paper we focus on IoT application scenarios and we define "smart object" a SBC embedded device equipped with a Linux based operating system that runs specialized pieces of software in order to grab and process data from external sensors. For simplicity, in this paper, we will use the terms embedded device, IoT device, smart device, and smart object as synonyms. We intend to distribute multiple smart objects in a complex environment, such as a smart city, where it it is crucial to rely on a Cloud service able to deploy and to customize pieces of software running on target smart objects.

LCV is a lightweight virtualization technology that allows the creation of containers in Linux OS , where different applications can run. Moreover, thanks to a version management system, LCV allows to resolve software dependencies in a flexible way. LCV is a paradigm of virtualization at the OS level, which enables multiple instances of the same OS at the user space, while sharing the kernel of the Host OS. There are two general usage models for containers:

- **Application Container**: a single application runs in the container.
- **System Container**: an instance of user space is booted in a container. This allows multiple isolated instances of user space to run at the same time, each one with its own Init process, process space, file system and network stack.

LCV allows providers to achieve a more flexible setup, configuration, optimization and management of sensing and actuating capabilities. Popular container engine solutions include Docker, LXC, lmctfy, OpenVZ. In the following, we analyze different criteria of interest to evaluate the applicability of virtualization technologies in embedded devices, comparing

the traditional HVV approach with the LCV one.

- **Start-Up-Time**. The container is lighter than a comparable HVV VM so that, in most cases, LCV wins in terms of initial start-up time and application start up time.
- **Dynamic-Runtime Control**. Due to the nature of the concept, LCV is integrated even tighter with the host system, allowing for example to start or kill applications in a container directly from the host while this would have to be done through semaphores or a virtual network/serial connection in a HVV scenario.
- **Speed**. The possibility to directly access devices in a HVV VM can mean a significant speedup compared to accessing it through the same driver running on the host system from a LCV container. Only tests with the specific scenario can show how much the communication overhead (LCV) vs. virtualization overhead (HVV) affects both latency and throughput.
- **Isolation**. LCV does not provide effective isolation on system level at all. HVV on the other hand requires booting its own kernel and user space, therefore allowing effective isolation on system and user level. A malfunction in the hypervisor may still crash the complete system, but code running within the VM will not be able to do this and, for example, allow the host to analyze the crash reasons and restart the VM.
- **Flash Memory Consumption**. This point is pretty obvious. LCV allows sharing both operating system kernel and whatever parts of the user space the developer chooses to be shared while HVV requires the VM images to be stored separately and allows no sharing. So LCV definitely wins in this aspect.
- **Communication Channels Between Virtual and Physical Environments**. Both concepts allow communication through virtual serial or networking interfaces or shared file systems that can be used to place semaphores or messages.
- **Dynamic Resource Assignment or Separation**. Dynamic resource assignment capability is important in load management and fail-over scenarios. It allows, for example to assign additional CPUs to heavily loaded VMs or remove CPUs from idle VMs or Containers. Both solutions are equally suited to perform fail-over scenarios.
- **Direct Hardware Access**. While HVV allows to directly access hardware peripherals from the VM through systems like virtio or vfio, this cannot be done from a LCV container which only has user-space access and therefore requires to have an according driver in the (host) kernel.

Containers on IoT devices is not only a theoretical concept. In fact, more and more manufactures are looking at container engines to simplify the packaging, distribution, installation and execution of complex applications on IoT devices. For example, a popular emerging solution consists in using Docker on multi-core Raspberry PI devices.

From this analysis it is evident how LCV is better than HVV except for two aspects that are isolation and access to physical assets. After that we have motivated the adoption of the container virtualization in IoT devices, in the following, we will analyse how can be possible to utilize it to arrange Stage 1, 2, and 3 scenarios.

IV. Virtual Resource Management in IoT Cloud

We assume that an IoT Cloud provider arranges IoTaaS exploiting both the HVV and LCV concepts. IoT devices interact with a remote Cloud system that is in charge to collect and uniform sensing data coming from heterogeneous IoT devices. Typically, IoT devices run customized software developed with a particular programming language and/or development framework (e.g., C, C++, Phyton, Nodejs, etc). Minimal processing and storage tasks can be performed in

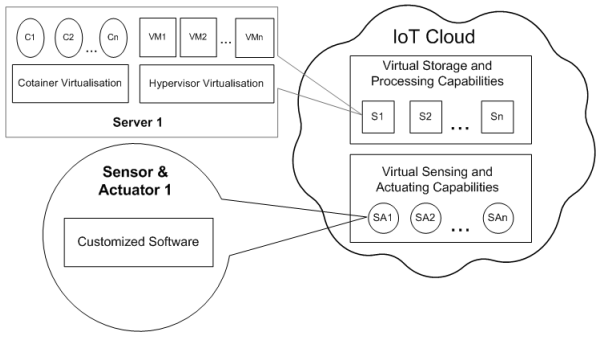

Fig. 2. Example of IoT Cloud exploiting both HVV and LCV virtualization technologies. IoT devices run customized pieces of software.

IoT devices, but the massive data storage and processing tasks (data mining and big data analytics) are performend in the Cloud system that exploits virtualization technologies to elastically scale up/down storage and processing capabilities. Figure 2 shows an example of IoT Cloud that manages, on one hand, storage and processing capabilities and, on the other hand, sensing and actuating capabilities. In particular, storage and processing capabilities are managed by means of HVV, whereas sensing and actuating capabilities are managed by means of customized pieces of software direclty installed on the operating system of IoT devices.

The above approach is currently the most adopted but it is not so flexible considering that often sensing and actuating services need to be tailored according to the changing features of the environment and requirements of users. Recently, a few initiatives have started considering applications deployed in IoT devices by means of containers. Thus, it is possible to take the advantages of both HVV and LCV. Figure 3 shows an example of IoT Cloud exploiting the LCV technology even in IoT devices. In this case, the IoT Cloud system is responsible for the instantiation, optimization, and management of IoTaaS guaranteeing reliability, scalability, Quality of Service (QoS) and Security. Thanks to LCV applied in IoT devices, it is possible to deploy distributed infrastructures, platforms, and applications (in form of IoTaaS) in several distributed containers in a more flexible way. IoT devices interact with a remote Cloud system that is in charge to collect and uniform sensing

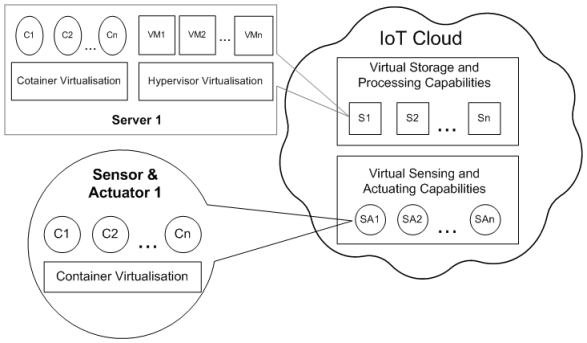

Fig. 3. Example of IoT Cloud exploiting both both HVV and LCV virtualization technologies. Application and services are deployed in a smart way using containers.

data and trigger actuators. All the virtualization technologies are able to hide the underlying physical assets, introducing a logical layer between the physical infrastructure and services. To take advantages of both virtualization technologies, we assume that both LCV and HVV are used in the Cloud datacenter, but only LCV is used in IoT devices. Private IoT Clouds hold their own virtualization infrastructures where several VMs and containers are hosted to provide services to their clients. In a flexible scenario, each IoT Cloud operator can dynamically arrange, optimize, and relocate its own virtualization resources including storage, processing, sensing, and actuating capabilities. Consequently, the IoT Cloud operator will be able to satisfy "on demand", in a flexible fashion, any service allocation request of its clients.

Skeptics on IoT Cloud Federation could ask: why should an IoT Cloud adopt contatiner virtualization for managing IoT resources? In our opinion, the answer is simple: it brings new business opportunities, such as cost-effective assets relocation and optimization, power saving, and on-demand resources provisioning. Furthermore, thanks to container vitualization applied to IoT devices it is possible to build up different overlays distributed systems on the same set IoT devices, each one with a particular purpose. In the following, we provide some examples:

- **Deployment of Distributed IoTaaS**. It is possible to arrange distributed IoTaaS combining services deployed on several IoT devices belonging to different sensor/actuator netorks controlled by the IoT Cloud operator;
- **IoTaaS Relocation and Optimization**. In order to enforce load balancing strategis, an IoT Cloud that manages different sensor/actuators networks can migrate the services and applications deployed in containers among different the IoT devices;
- **IoTaaS Consolidation**. Considering IoTaaS built combining the containers installed in different IoT devices belonging to different IoT Cloud sensor/actuator networks it is possible to apply location-aware services by means of software consolidation strategies. In fact, in some cases it is useful to move services according to a particular application logic;

V. Container Virtualitation in Linux-Based SBC

In this Section, we discuss how it can be possible to use the container virtualization technology in Linux-based Single Board Computers (SBCs). In particular, we focus on an IoT architecture based on a Raspberry Pi B+ model running the Docker container engine. As discussed in the previous Section, in a container-based virtualization, the virtualization layer acts as an application on top of the OS. In this approach, the OS's kernel runs on the hardware node with several isolated guest virtual environments installed on top of it. The isolated guests are called containers. In this Section, we describe the pieces

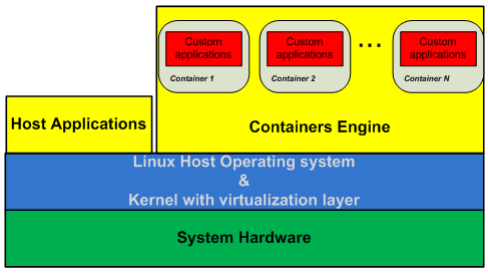

Fig. 4. Container-based virtualization.

of software needed to support the container virtualization by considering a generic Linux system that can be installed in IoT devices. Looking at the Figure 4, a Linux host OS is normally deployed on top of hardware system and its kernel needs to work with a suitable virtualization layer. In this way, the OS-level virtualization does not require an additional hypervisor layer since the virtualization capabilities are part of the host OS. This technique allows us to virtualize servers on top of the host OS itself. Therefore, the overhead produced through the hypervisor mediation is eliminated and it enables near native performance. In addition, the host kernel provides process isolation and performs resource management. This means that even though all the containers are running under the same kernel, each container is a virtual environment that has its own file system, processes, memory, devices, etc. There are host applications located on the top of the Linux kernel: we focus our attention on the containers engine component that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere.

Considering IoT devices, here we mainly focus our attention on considering Linux-based Single Board Computers (SBCs) that include several General Purpose Input Output (GPIO) extensions allowing IoT devices to interact with many different attached sensors and actuators. Figure 5 shows an example of such a system architecture. Starting from bottom, the hardware system consists of a Raspberry Pi B+ model. While the latter board is, in essence, a very inexpensive Linux computer, there are a few things that distinguish it from a general purpose machine. One of the main differences is that the Raspberry Pi can be directly used in electronics projects because it includes GPIO pins right on the board. These GPIO hardware extensions can be accessed for controlling hardware such as

LEDs, motors, and relays, which are all examples of outputs. As for inputs, the used Raspberry Pi can read the status of buttons, switches, and dials, or it can read physical magnitudes like temperature, light, motion, or proximity [10].

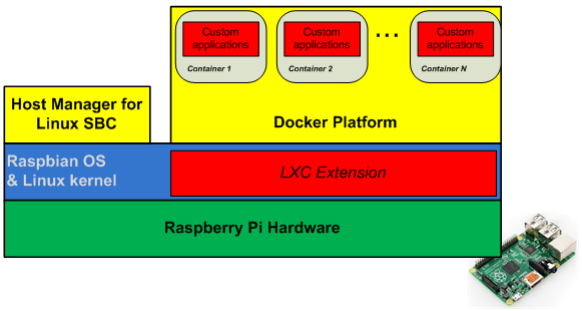

Fig. 5. Software architecture for container-based virtualization deployed on a Raspberry Pi board.

The Raspberry Pi board is equipped with a Raspbian distribution that is the most popular OS for the considered hardware; it also includes customizations that are designed to make the Raspberry Pi easier to use including many different software packages out of the box. In particular, we considered a Raspbian OS supporting the 3.18.8 Linux kernel version that comes with the LXC extensions. As the authors discussed in [11], these extensions represent a container-based OS virtualization that allows us to run multiple Linux instances on a single SBC. With reference to the Figure 5, host applications are deployed on the top of Raspbian OS and Linux kernel.

We consider the Docker Platform as container engine, which is an open platform for developers and sysadmins to build, ship, and run distributed applications. The Docker Engine represents a portable, lightweight runtime and packaging tool. In addition, thanks to Docker Hub, that can be considered as a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, and production environments. As a result, IoT Cloud operators can faster ship and run the same service or application on different IoT devices. Docker is also an open-source implementation of the deployment engine which powers dotCloud, a popular Platform-as-a-Service (PaaS) that directly benefits from the experience gained over several years of large-scale operation and support to hundreds of thousands of applications and databases.

VI. Experiments

In this Section, we investigate the overhead of container virtualization in an IoT device. In particular, we conducted several experiments considering a Raspberry Pi equipped with a Docker container engine. We considered the average response time of a specialized Constrained Application Protocol (CoAP) server implementation running on dedicated Docker's Linux Containers - version 1.6. In particular, we used txThings, i.e., a Python implementation of CoAP that is based on Twisted, an asynchronous I/O framework and networking engine. Our testbed contains two separate Raspberry systems with the following hardware specifications: system on-chip: Broadcom BCM2835, CPU: 700 MHz ARM11 ARM1176JZF-S core, memory: 512 MiB SDRAM, on-board storage: Class 10 micro 8GB SDHC card, on-board network: 10/100 wired Ethernet RJ45 connection. The first device acts as server and it executes different containers at the same time. This configuration is reasonable if we consider a real scenario. For example, considering a smart mall scenario, different services could run at the same time on the same IoT device: a CoAP server could trigger the lighting line to turn on/off light appliance whenever people enter or leave a room, in order to minimize the wasting of energy consumption; another CoAP server could provide offer information to clients that pass close to a shop, thus to encourage clients to enter; a third CoAP server could provide information on where (shops or area in the mall) to find one or more products, and so on.

Each virtual environment embeds a specialized CoAP server which listens on a particular UDP port. The other Raspberry system acts as CoAP client and it runs no Linux Containers. Each CoAP server can serve multiple client requests at the same time and it replies with a simple ACK message. We connected the two systems through a private gigabit ethernet network and we calculated the time between the moment when a client submits a HTTP GET request to a server and the moment when it receives an ACK message Listings 1 shows the command used to start the CoAP server in a Docker container.

```
docker run -t -i --net="host" -p 7777:7777
shop/coapserver python serverm.py 7777
```

Listing 1. Command to start a Docker container with a CoAP server.

Experiments were repeated 30 times in order to consider mean values. In particular, we considered 4 different testbed configurations:

- **conf 1**. 1 CoAP server directly running on Raspian OS;
- **conf 2**. 1 CoAP server running on 1 Docker container A deployed on the Raspian OS;
- **conf 3**. 2 CoAP servers respectively running in parallel on 2 different Docker containers, that are A and B, deployed on the Raspian OS;
- **conf 4**. 3 CoAP servers respectively running in parallel on 3 different Docker containers, that are A, B, ans C, deployed on the Raspian OS;

For each CoAP server, we sent from 1 to 4 presence requests. More specifically, in the case of Raspberry configuration with multiple CoAP servers we sent client requests in parallel.

The histogram depicted in Figure 6 compares the 4 configurations. In particular, we compared the response time of a CoAP server directly running on the Raspian OS (conf 1) with the response time of the CoAP server running on the Docker container A considering configurations 2, 3, and 4. On the x-axis we reported the number of client requests, whereas on the y-axis we reported the time spent to satisfy the request expressed in milliseconds. Considering 1 request, the

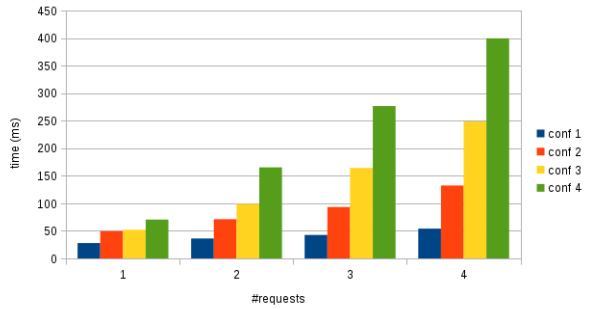

Fig. 6. Response time of the CoAP server receiving a variable number of requests from clients in different testbed configurations.

overhead of container virtualization is roughly 22ms, 24ms, 43ms respectively considering the CoAP server running on container A according to configurations 2, 3, and 4. In this case response times are acceptable. Instead, increasing the number of requests in parallel we observe that such a gap increases. In fact, considering 4 request, the overhead of container virtualization becomes more significant being roughly 78ms, 195ms and 346ms considering the three Docker-based configurations. From such experimental results we can conclude that the overhead of container virtualization is minimum with respect to the overhead due to process the requests by CoAP servers. In addition, we have also to consider the overheard due to single physical network interface that acts as a bottleneck for network traffic.

We can observe how independently from the number of requests sent by clients in parallel, the response times of the CoAP server directly installed on the Raspian OS is always shorter than the response times of CoAP servers deployed on containers. In fact the overhead in term of container virtualization is roughly 280ms

VII. Conclusion

Container-based virtualization is a lightweight alternative to the hypervisor-based approach that can be adopted on smart objects, for enhancing the IoT Cloud service provisioning. Considering different IoT application scenarios, container-based virtualization allows IoT Cloud providers to deploy and customize in a flexible fashion pieces of software on smart objects. In this paper, we studied the container-based virtualization on smart objects in the perspective of a IoT Cloud scenarios specifically focusing on a Raspberry device running the Docker container engine. From our experiments we highlighted overhead introduced by container virtualization is acceptable in a real scenario. In future works we plan to study the parallel processing of a distributed service deployed on multiple containers running on an IoT device.

Acknowledgment

This work was partially supported by EU H2020 BEACON Project under Grand Agreement n.644048.

References

[1] A. Celesti, M. Fazio, M. Villari, M. Giacobbe, and A. Puliafito, "Characterizing iot cloud federation," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications Workshops - Workshop on Cloud Computing Project and Initiatives - CCPI'16*. IEEE Computer Society, March 2016.

[2] M. Maksimović, V. Vujović, N. Davidović, V. Milošević, and B. Perišić, "Raspberry pi as internet of things hardware: Performances and constraints," *design issues*, vol. 3, p. 8, 2014.

[3] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS),*, March 2015, pp. 171–172.

[4] E. W. Biederman and L. Networx, "Multiple instances of the global linux namespaces," in *Proceedings of the Linux Symposium*. Citeseer, 2006.

[5] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.

[6] D. Liu and L. Zhao, "The research and implementation of cloud computing platform based on docker," in *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on*, Dec 2014, pp. 475–478.

[7] M. F. Kacamarga, B. Pardamean, and H. Wijaya, "Lightweight virtualization in cloud computing for research," in *Intelligence in the Era of Big Data*. Springer, 2015, pp. 439–445.

[8] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[9] M. Xavier, M. Neves, F. Rossi, T. Ferreto, T. Lange, and C. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, Feb 2013, pp. 233–240.

[10] M. Richardson and S. Wallace, *Getting Started with Raspberry Pi.* " O'Reilly Media, Inc.", 2012.

[11] N. Memari, S. J. B. Hashim, and K. B. Samsudin, "Towards virtual honeynet based on lxc virtualization," in *Region 10 Symposium, 2014 IEEE*, April 2014, pp. 496–501.