Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge

Roberto Morabito, Ivan Farris, Antonio Iera, *Senior Member, IEEE*, and Tarik Taleb, *Senior Member, IEEE*

***Abstract*—The constant and fast increase in the number of heterogeneous Internet of Things (IoT) devices that populate everyday life environments brings new challenges to the full exploitation of the computation, memory, sensing, and actuation resources associated to them. In this context, device virtualization solutions and platforms may definitely play a key role in enabling the desired tradeoff between flexibility and performance. This paper focuses on lightweight virtualization technologies for IoT devices, suitably thought to effectively deploy new integrated applications and to create a novel distributed and virtualized ecosystem. Two different frameworks for container-based IoT service provisioning are compared, the one based on a direct interaction between two cooperating devices and the other based on the presence of a manager supervising the operations between cooperating devices forming a cluster. In the latter case, accounting for the growing impetus to move intelligence toward the edge of the network, management features are implemented at the network access point to provide short latency responses. We also introduce the outcomes of a thorough performance evaluation campaign conducted via a real IoT testbed. The measurements, performed by accounting for the constraints of typical IoT nodes, shed light on the actual feasibility of container-based IoT frameworks.**

***Index Terms*—Container virtualization, Internet of Things (IoT), multi-access edge computing, service orchestration.**

I. Introduction

In THE last years, the disruptive nature of the Internet of Things (IoT) paradigm has revolutionized the smartness of our surrounding environment [1]. The exponential growth in IoT devices makes available plenty of resources for computation, storage, sensing, and actuation. At the same time, we are assisting to an increasing spread of relatively inexpensive general-purpose boards, such as Raspberry Pi [2], which allow for easy deployments of a wide range of IoT use cases. Hentschel *et al.* [3] described the early prototypal deployment of a campus-wide sensor network with advanced features compared to traditional solutions, by using commodity single-board devices.

The last few years have also witnessed a growing attention toward lightweight virtualization technologies, such as Docker [4] and LXC [5] containers. These solutions allow for an efficient deployment of virtualized services while requiring a reduced overhead with respect to hypervisor-based virtualization technologies [6]. Additionally, the opportunity of exploiting containers for IoT resource-constrained devices has been demonstrated in [7]. Container-based service provisioning can provide several benefits to heterogeneous IoT environments, allowing to deploy on-demand services according to the nodes capabilities and to dynamically reconfigure the operational behavior of devices. However, to fully exploit the potential of IoT devices, new solutions are necessary to simplify the management and provisioning of integrated IoT applications.

In this paper, we aim at investigating container-oriented operational frameworks for IoT. More specifically, we comprehensively investigate two different frameworks for container-based IoT service provisioning: 1) *pair-oriented*, where two cooperating devices can directly interact with each other and 2) *edge-managed clustering*, where a manager supervises the operations among cooperating devices forming a cluster. In this latter solution, the cluster manager is running on an edge access point, thus providing both connectivity and fast management response to the attached IoT devices.

Indeed, accounting for the momentum shown by multi-access Edge computing [8] and Fog computing [9] paradigms, which move intelligence toward the edge of the network, we believe that management features implemented closer to the end-user device is a key enabling feature for delay-sensitive applications in smart IoT environments.

The main enquiry that underlies the research activity in this paper is: *How can lightweight virtualization and clustering management features allow for fully exploiting the resources offered by IoT devices in pervasive environments?* Giving an answer to this question implies to answer the following sub-questions.

1) Which challenges must be tackled to effectively exploit the resources offered by heterogeneous IoT devices?
2) Which control features should be implemented in the involved devices?
3) Which management framework can better meet the manifold requirements imposed by integrated and critical IoT applications?

Manuscript received January 9, 2017; revised May 9, 2017; accepted June 1, 2017. Date of publication June 9, 2017; date of current version August 9, 2017. This work was supported in part by the FP7 Marie Curie Initial Training Network METRICS Project under Grant 607728. *(Corresponding author: Ivan Farris.)*

R. Morabito is with Ericsson Research, 02420 Kirkkonummi, Finland (e-mail: roberto.morabito@ericsson.com).

I. Farris is with the Department of Communications and Networking, School of Electrical Engineering, Aalto University, 02150 Espoo, Finland, and also with the DIIES Department, University “Mediterranea” of Reggio Calabria, 89100 Reggio Calabria, Italy (e-mail: ivan.farris@aalto.fi).

T. Taleb is with the Department of Communications and Networking, School of Electrical Engineering, Aalto University, 02150 Espoo, Finland, and also with Sejong University, Seoul, South Korea (e-mail: tarik.taleb@aalto.fi).

A. Iera is with the DIIES Department, University “Mediterranea” of Reggio Calabria, 89100 Reggio Calabria, Italy (e-mail: antonio.iera@unirc.it).

Digital Object Identifier 10.1109/JIOT.2017.2714638

Accordingly, in this paper, we thoroughly evaluate container-based solutions in a real IoT environment. In particular, we adopt Docker containers to deploy heterogeneous services on resource constrained IoT devices. Through the setup of a real testbed, we assess the performance in terms of power and resources consumption, with the aim of shedding light on the actual feasibility of such emerging lightweight virtualization approaches.

This paper is organized as follows. Section II browses the related scientific literature to highlight the potential benefits introduced by the use of container-based service provisioning. In Section III we provide a brief overview of enabling virtualization technologies and point out the challenges in resource-constrained IoT environments. The investigated container-oriented solutions are introduced in Section IV. Details on the implemented testbed and experimental results are reported in Section V. Open research areas and conclusions are drawn in Sections VI and VII, respectively.

II. Related Work

The adoption of software-oriented solutions is one of the most promising trends to address the challenges raised by smart IoT environments. The majority of research activities have focused on improving IoT interconnectivity by embracing software defined networking [10], [11]. The next step is definitely the development of a virtualized ecosystems for the provisioning of integrated applications over IoT devices, so as to enable the so-called anything-as-a-service paradigm [12].

To bridge the gap between the high-level requirements of applications and the low-level hardware complexities of IoT nodes, different middleware architectures have been proposed over the last few years in WSN and IoT environments. Research and industrial communities have particularly focused on IoT middleware that not only guarantee a unified access to data generated by IoT devices (such as sensing measurements) but also give opportunities to easily reprogram the operational behavior of IoT nodes. This increased level of flexibility can enable novel IoT scenarios where: 1) on-demand service are deployed according to the nodes capabilities; 2) information exchange and task offloading among cooperating IoT nodes is easily carried out to boost integrated IoT applications; and 3) distribution of IoT service instances is dynamically optimized according to actual workload and IoT nodes capabilities. Different solutions for resource constrained WSNs, leveraging tiny virtual machines, have been developed [13]–[15], for example, based on Java and Python execution environments, to enable programming of sensor nodes and code mobility. However, these solutions are strictly dependent on the underlying virtualization environment. Obviously, this aspect limits the flexibility in application development and potentially implies limiting code dependencies.

In this regard, lightweight virtualization solutions are gaining great momentum as technological enablers of a distributed virtualization infrastructure supporting heterogeneous IoT devices. Recent works have investigated the opportunity to exploit nodes' resources by implementing Docker containers [7], [16]. Container-based service provisioning do not show any strict dependence on a given technology, programming language, or application domain, thus offering the freedom of "*develop once, deploy everywhere*." However, a thorough analysis of methods for container-based application provisioning, not only focusing on computation and storage but also on sensing and actuation capabilities, is still missing. Furthermore, container-oriented management frameworks have been designed by only referring to a Cloud data center environment [17].

IoT platform-as-a-service (PaaS) [18] is emerging as a paradigm complementary to distributed IoT middleware. An exemplary IoT PaaS architecture, aiming at leveraging Cloud models to enable efficient and scalable IoT service delivery, has been defined in [19]. Such a Cloud-based approach may be limiting for nowadays applications, as IoT services requirements are becoming very strict especially in terms of latency and network traffic. To tackle these issues, next-generation network architectures are increasingly considering the multi-access Edge computing paradigm [8], [20] as a means to extend Cloud computing processing capabilities at the edge of the network, by introducing the concept of Cloudlets. By distributing small data centers near to the access points, such as femtocells, several benefits are provided in terms of reduced communication overheads, costs, and latencies [21].

Closer to the IoT domain is Fog computing [9], a similar paradigm intended to provide distributed Cloud environments closer to physical devices and able to support delay-sensitive IoT services such as real-time data analytics [22], [23]. Our container-based approach extends the current IoT PaaS solutions toward the edge of the network. In particular, container virtualization offers new potential benefits in terms of cross-platform deployment, allowing a common execution environment for Cloud, Edge/Fog nodes, and even constrained devices. Indeed, the same container-virtualized instance can efficiently run both at the Edge and in the Cloud. Furthermore, containers can even run on devices characterized by limited computational resources, such as Raspberry Pi. This feature guarantees a transversal interoperability that goes from resource constrained devices up to Cloud architecture. A further advantage introduced by containers is the possibility to isolate, from the underlying system, all the processes running within a virtualized container. This feature promotes the deployment of multitenant platforms, as the same hardware can be shared among different tenants.

Indeed, the potential of emerging mobile Cloud computing (MCC) approaches [24], [25] cannot be underestimated. According to MCC, nearby devices autonomously cooperate to provide the desired services by mutually sharing their resources.

Our envisaged clustering framework is complementary to both Edge and Fog computing paradigms, since control and management functions are implemented at the Cloud-enhanced Edge nodes, whereas task execution is delegated to IoT devices, whose distributed resources allow increasing scalability. The idea of a resource coordinator elected among the mobile nodes was proposed in [26] to manage the matching between application requests and device resources. Habak *et al.* [27] proposed the refactoring of the Cloudlet into a controller, which is in charge of configuring nearby devices to provide collaborative computational services. However, these works do not specifically tackle the heterogeneity of IoT devices, and do not consider any IoT virtualization platform to effectively enable IoT task offloading.

III. Enabling Technologies

Container-based virtualization solutions have gained great popularity in recent years. They leverage some features of the operating system kernel, i.e., namespaces and control groups (cgroups). Linux namespaces isolate processes (i.e., containers) from each other, whereas cgroups allow for reducing the resources, such as CPU, memory, and block device I/O, allocated to each container. Compared to full and para-virtualized approaches, OS-level virtualization (i.e., container virtualization) is directly done in the kernel, thus guaranteeing better performance [6].

Lightweight virtualization technologies introduce interesting features, which make them extremely attractive in IoT environments, such as: 1) fast creation and initialization of virtualized instances; 2) high density of applications, thanks to the small container images; and 3) reduced overhead, while enabling isolation between different instances running in the same host [28], [29].

In this paper, we use application-oriented Docker containers for executing heterogeneous IoT applications. Docker functionalities are based on an underlying container engine, the so-called Docker engine, which is a lightweight containerization technology that includes all the software components devoted to the management of Docker containers. Furthermore, it provides a functional APIs that allows for easily building, management, and removal of a virtualized application. With respect to system-level containers, e.g., OpenVZ and LXC, application-oriented containers better cope with the microservice architecture, which is considered the next big revolution for IoT service deployment [30].

Container orchestration systems represent a core element to ease the deployment and management of multiple containerized applications across a number of physical or virtual hosts, especially for data center environment [28]. The most popular solutions are Kubernetes [31], Docker Swarm [32], and Apache Mesos [33]. In this paper, we focus on Docker Swarm, since is natively integrated with Docker distributions.

A Docker Swarm is a cluster of running Docker engines, which leverage the management features provided by the SwarmKit. Specifically, two different logical entities are defined: 1) the *manager node*, which performs the orchestration functions required to maintain the desired state of the swarm and dispatches units of work called “tasks” and 2) the *worker nodes*, that receive and execute tasks scheduled by the manager. By default, manager nodes are also worker nodes, but it is possible to configure managers to be manager-only nodes. The agent notifies the current state of its assigned tasks to the manager node, so that the manager can continuously monitor the state of the cluster. Each node in the swarm enforces transport level security mutual authentication and encryption to secure its communications with all other nodes.

The standard Docker API, or any tool that already communicates with a Docker engine, can be used to implement swarm management procedures, such as adding and removing nodes, as well as deploying services to the swarm and managing service orchestration. While legacy Docker engine issues container commands, Docker Swarm mode orchestrates “services”, which are the definitions of the tasks to be executed on the worker nodes. Indeed, a central structure of the swarm system is the service concept, whose definition specifies the container image to use and the commands to execute inside the relevant containers, when a service is created. In the case of replicated services, the swarm manager distributes a specific number of replica tasks among the clustered worker nodes based on the desired application requirements.

Although some of the objectives discussed above are common to classic data center environment, IoT introduces additional challenges that deserve specific attention.

1) *Resource Constraints:* IoT devices are typically characterized by extremely reduced capabilities, thus lightweight solutions become essential. Indeed, even if VMs can also enable image-based management, VMs are neither portable nor lightweight as containers. This feature limits the adoption of a virtualization approach. Docker containers seem to guarantee the appropriate tradeoff in terms of flexibility and performance.
2) *Heterogeneity:* To effectively deploy varied services in different IoT devices, a common resource abstraction is essential. Docker engine provides the virtualization features to exploit nodes capabilities in terms of computation, storage, and networking. Notwithstanding, further activities should address sensing and actuation operations through appropriate container configuration.
3) *Wireless Environment:* IoT devices are usually interconnected by wireless networks with likely reduced bandwidth and highly variable channel quality. Therefore, management solutions must be designed so to reduce the control traffic, while avoiding network bottlenecks.
4) *Service Management Complexity:* IoT applications can be highly integrated. This requires the deployment of several computing, sensing, and actuation operations over multiple nodes. Hiding such an underlying complexity in service provisioning is an undoubted key feature to boost IoT application deployment. The capabilities of an IoT client can be, indeed, extremely low and moving the service management burden to a third-party controller may result strictly necessary. However, the introduction of an external controller can introduce new challenges impacting on both delay and resource consumption.

IV. Container-Based IoT Service Provisioning Approaches

This paper aims at tackling the challenges described in the previous section by leveraging Docker-based service

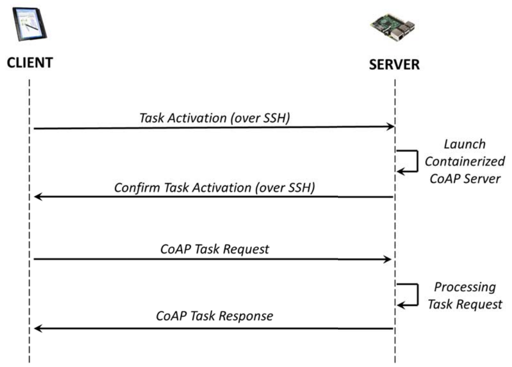


Fig. 1. Workflow of the CPIS approach for clustered IoT environment.

provisioning in wireless resource-constrained IoT environments. In particular, to analyze and identify the impact of Docker management, we consider two exemplary case studies, which are good representatives of a broad range of IoT scenarios. In the first case, the client is a smart device able to directly manage the activation of services by issuing control commands to a collaborating IoT node. In the second scenario, the client is extremely simple and fully relies on an external manager node for service management operations, such as activation/removal, status monitoring, etc. By taking as a reference these case studies, we have designed and developed two different approaches to container-based IoT service provisioning: 1) *pair-oriented mode*, where two cooperating IoT devices directly interact with each other and 2) *edge-managed clustering mode*, where a manager supervises the operations between the cooperating devices forming a cluster. Both approaches are based on Docker containers to deploy heterogeneous applications over IoT devices. In this paper, we assume that all nodes belong to the same private user or public administration; this allows to rely on a “trusted scenario” where services can be safely offloaded to different nodes, according to application criteria and devices, status. The found results can be easily generalizable to a multiowner scenario by providing mechanisms for trustworthiness and security control, which anyway are out of the scope of this paper. The last, widely acceptable, assumption is that all the involved nodes are under the coverage of the same network access point, which can be represented by an Ethernet switch, a Wi-Fi access point, or a cellular femtocell.

A. Container-Based Pair-Oriented IoT Service (CPIS) Provisioning

According to this approach, management operations are based on direct data exchanges between the involved devices. To enable secure transmissions, all control data traffic is exchanged over SSH communications. Indeed, in this paper, we do not focus on scheduling algorithms to choose the best candidate node for executing a specific task. Rather, we aim to investigate on the required procedures to activate a containerized service and to enable the interaction between the device requesting the task (i.e., the client) and the device actually executing the task (i.e., the server) in heterogeneous IoT environments.

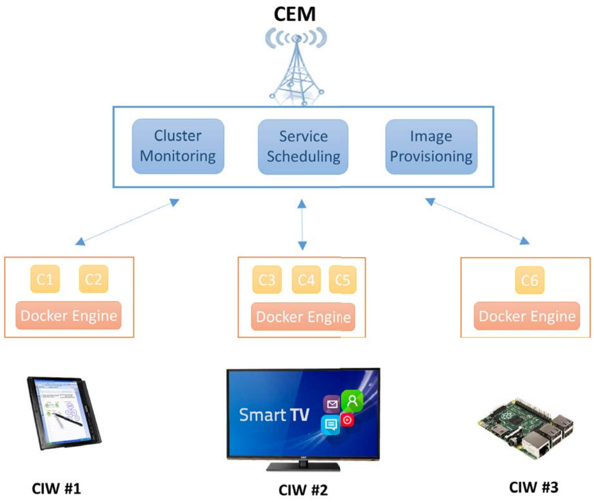


Fig. 2. Container-based architecture for clustered IoT environment.

Fig. 1 shows the basic workflow between a client and a server to achieve container-based IoT service provisioning. First, the client issues the command for task activation on the server, which in turn executes the “containerized” service by leveraging the local Docker engine. To guarantee the desired interoperability in compliance with current IoT standards, constrained application protocol (CoAP) is recommended to define RESTful application interfaces [34], according to a traditional client-server model.

Therefore, after the successful activation of the requested container, interactions between client and server follow the CoAP protocol rules. Once the desired task is completed, the client can also issue the command to stop and remove the container. In this approach, the entire control burden is delegated to the client, which has to comprehensively manage the instantiation, monitoring, and removal of the instance. Furthermore, the control requirements can be even increased when interactions with multiple nodes are required, such as in one of the following situations: 1) the IoT application is composed by multiple modules (for example, with different sensing requests); 2) the IoT client may be in charge of keeping backup service, so to guarantee service continuity and reliability even in case of node failure; and 3) the client can need to scale up or down the service instances by accounting for the resources of IoT nodes and the actual workload. To sum up, this approach can ensure that fast management procedures are achieved through the direct interaction between the cooperating nodes, while all the control features for service lifecycle are implemented in the client.

B. Container-Based Edge-Managed Clustering (CEMC)

According to this second approach, a container-oriented orchestration system provides multiple features to: 1) ease the deployment and monitoring of IoT services over multiple nodes; 2) perform periodical service checking and resource monitoring; and 3) implement replication and auto-scaling policies. Similarly to the Docker Swarm framework, which we use as a reference platform, we consider two different logical nodes: 1) container-oriented edge manager (CEM) and 2) container-based IoT worker (CIW), whose features are presented in the remainder of this section. An exemplary scenario is sketched in Fig. 2, where a CEM controls a cluster of nodes, operating as CIWs and leveraging the virtualization features provided by Docker engine to host containerized IoT services.

1) Container-Oriented Edge Manager Features: In our view, a predominant role has to be played by the network access point. This latter operates as a manager of the clustered devices, by both providing network connectivity and orchestrating integrated IoT applications. Indeed, we believe that network providers are in a predominant position to offer new management services to their IoT customers, by leveraging on their capillary infrastructure and on the emerging cloudification of the Edge networks [35].

The CEM is responsible to handle several cluster management tasks.

- *Maintaining the Cluster State:* The CEM maintains an up-to-date internal state of the entire swarm, accounting for the available resources offered by each associated device and for all the services running within the cluster. In a densely connected environment, multiple CEMs can be deployed over different access points to guarantee fault tolerance features. Indeed, Docker Swarm uses a Raft implementation to maintain a consistent distributed state of the cluster among multiple manager nodes.
- *Service Scheduling:* When a new service is requested, the CEM needs to select the most appropriate nodes to deploy the containerized applications, by matching service requirements and available workers, resources.
- *Service Monitoring:* During the whole application lifecycle, the CEM must monitor the status of relevant containers and, if some failures are detected, new instances must be promptly activated, to guarantee the desired Quality of Experience.
- *Distributing Container-Based Application Images:* To store and distribute Docker images containing all the applications code and dependencies, Docker has introduced public/private Docker registries. The desired flexibility is achieved by enabling the CEM to implement a private registry, to share trusted images among the nodes of the cluster.

2) Container-Based IoT Worker Features: CIWs are devices running instances of Docker engine whose sole purpose is to create, start, and stop containers. These devices require an operating system, whose kernel supports container-based virtualization. Some bootstrap code is necessary to activate Docker in swarm mode, and to automatically join the desired cluster. Once a CIW has joined the cluster, the CEM can deploy containerized applications through the CIW's Docker engine. If the relevant Docker image is not locally available, then the CIW's Docker engine can retrieve it through either a public Docker registry, i.e., the Docker hub, or a private Docker registry running on the CEM.

CIWs can also operate as a proxy or gateway node for extremely resource-constrained nodes, which do not support container-based virtualization yet. In [36], gateways features can be deployed on-demand via Docker containers, to provide integration capabilities to sensor/actuation devices.

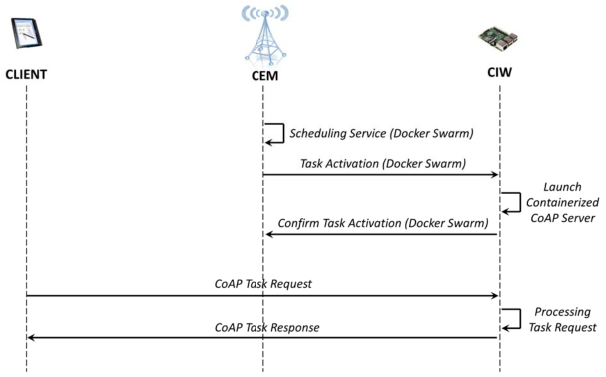

CLIENT
CEM
CIW
Scheduling Service (Docker Swarm)
Task Activation (Docker Swarm)
Launch Containerized CoAP Server
Confirm Task Activation (Docker Swarm)
CoAP Task Request
Processing Task Request
CoAP Task Response

Fig. 3. Workflow of CEMC for clustered IoT environment.

In Fig. 3 an exemplary workflow of container-based edge-managed clustering (CEMC) approach is sketched. After the CEM receives a service requests, it performs the relevant scheduling operation by identifying the device which can better host the requested applications in the cluster. Then, the CEM issues a command to launch the containerized task in the selected CIW. When the container is running, the client can send CoAP task request to the containerized CoAP server running on the CIW, which performs the desired logic operations providing the output in a proper CoAP response packet.

V. Performance Evaluation

Objective of our evaluation campaign is to assess the performance of management and operational features of container-based service provisioning in resource-constrained IoT environments. In this thread, we focus on the analysis of the centralized management introduced by the CEMC approach with respect to the direct interactions envisaged by the CPIS approach. Such an analysis results extremely useful to service designers and engineers for the selection of the most appropriate operational management, according to the application requirements.

A further aspect to investigate is the impact of the used network interface on the communication performance. In particular, in view of the emerging *wireless edge IoT clustering* concept, this paper will assess the feasibility of deploying a container-based solution not only in the presence of devices connected through an Ethernet interface but also in case of wireless connections.

A. Testbed Scenario

For our performance evaluation, we use the Raspberry Pi 3 (RPi) board as an IoT representative of a broad family of smart devices and appliances. Indeed, this board is mass-produced, relatively inexpensive, and can be used for a wide range of IoT applications. Not by chance, over eight million RPi boards have been sold to date and have been increasingly included in research and industrial works [3]. The RPi device has a variety of interfaces for attaching hardware sensor devices. Another key aspect that led us to its choice is the RPi capability in efficiently managing virtualized applications by means of container technologies. The feasibility to execute Docker containerized services on top of an RPi has been shown in [7].

In our experimental setup, an additional RPi provides edge management functionality (in case of CEMC approach). Although a workstation can perform management tasks as well, we have chosen the RPi since its lightweight capabilities are comparable to the limited resources of a generic network access point, such as an Ethernet or a Wi-Fi switch. This way, we demonstrate that even a low-power device can efficiently manage the orchestration of tasks at the edge of the network.

As concern the network configuration in the container-based pair-oriented IoT service (CPIS) modality, the two cooperating devices are interconnected through the same 100 Mb/s Ethernet switch in the wired case, whereas they rely on an ad-hoc Wi-Fi LAN (WLAN) created by the server node in the wireless case. In the CEMC modality, both CEM and CIWs are connected through a 100 Mb/s Ethernet switch (this because the RPi having CEM role offers a single Ethernet interface and is not able to interconnect multiple devices in a wired configuration). In case of wireless configuration, an ad-hoc network is managed by the CEM to provide connectivity to the clustered devices.

From the software perspective, the operating system we use is the image provided by Hypriot,[1] which runs Raspbian Jessie with Linux kernel 4.4.10. The Hypriot image provides a lightweight environment optimized for the execution of Docker container technologies, by also offering dedicated tools for container orchestration like Docker Swarm. Our services are currently implemented in Python, as this language is well supported on the RPi and provides libraries for easily connecting to the various hardware interfaces required by sensors, including interintegrated circuit bus, serial peripheral interface bus, and GPIO pins. For the CoAP server implementation, we use the txThings framework [37], which allows for a fast deployment of CoAP-oriented applications in Python code.

Regarding the measurement tools, the Unix tool *dstat* [38] is used to evaluate the resource usage of the device running the CoAP server. Dstat is a versatile tool that allows for monitoring all the system resources instantly (e.g., CPU utilization, system load, RAM usage, etc.). Furthermore, we measure the power consumption of the RPi during the execution of different tasks by using a voltage meter (USB-1608FS-Plus), which has a resolution of 16 bits. Indeed, since the RPi can be charged via USB, by interrupting the power lines of the USB connection and inserting a measurement shunt in the 5-V line, it is possible to measure the power consumption produced by the device, through indirect measure. The USB measurement tool can simultaneously acquire data from up to eight devices. This feature allows us to monitor the power consumptions of all the devices involved in our scenario. The setup used to investigate the performance of the CEMC approach in wireless configuration is shown in Fig. 4.

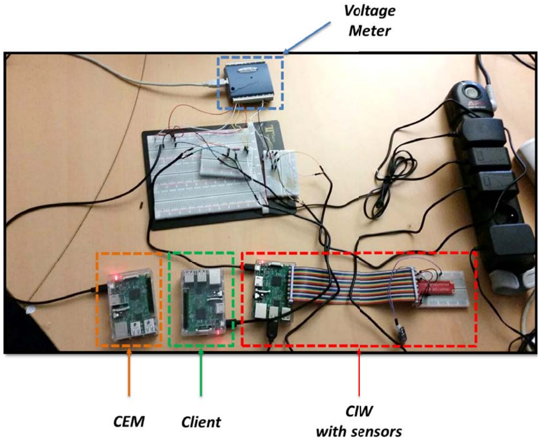


Fig. 4. Testbed environment for CEMC approach in wireless configuration.

TABLE I
POWER CONSUMPTION OF RPI IN IDLE STATE

Network Configuration	Idle Power Consumption
Ethernet	1.323 W
Wireless	1.449 W

B. Experimental Results

To demonstrate that container-based virtualization causes a negligible performance degradation with respect to OS native service execution, for a broad range of IoT applications, we have considered three different heterogeneous IoT tasks. Before going through the analysis of the different scenarios, in Table I we report the power consumption of the RPi measured in Idle state, both for the Ethernet and Wireless configurations. This value can be used as a benchmark to compare the RPi power consumption during a task execution.

1) Computation Tasks: Offloading high computation-demanding tasks is a typical operation of IoT devices. To evaluate this use case, we consider generic-purpose applications with different computation complexity, so that the results can be easily generalized to manifold scenarios, according to the service requirements.

In particular, we implement a CoAP server exposing resources to perform three different operations: 1) *Average*, which computes the average of all the numbers provided in input and its complexity is equal to $O(n)$; 2) *MergeSort*, which allows to order the list of numbers with a complexity equal to $O(n \cdot \log n)$; and 3) *BubbleSort*, which allows to order the list of numbers with a complexity equal to $O(n^2)$.

We measure the response times for both OS-native and Docker-based execution. The response time is measured by considering the elapsed time interval from the instant when the CoAP client begins to transmit the CoAP PUT request (including the input random array of integer numbers) to the instant when it retrieves the CoAP response. Each test is repeated ten times and all the results are shown with a 95% confidence interval. Fig. 5 shows the experimental response times of the three operations for different input sizes (i.e., 1000 and 2000 integer numbers). An increase in the size of number array implies an increase in the response times for each investigated operation. Furthermore, higher response times are measured for the wireless connectivity with respect to the wired networking.

[1] [Online]. Available: https://blog.hypriot.com/

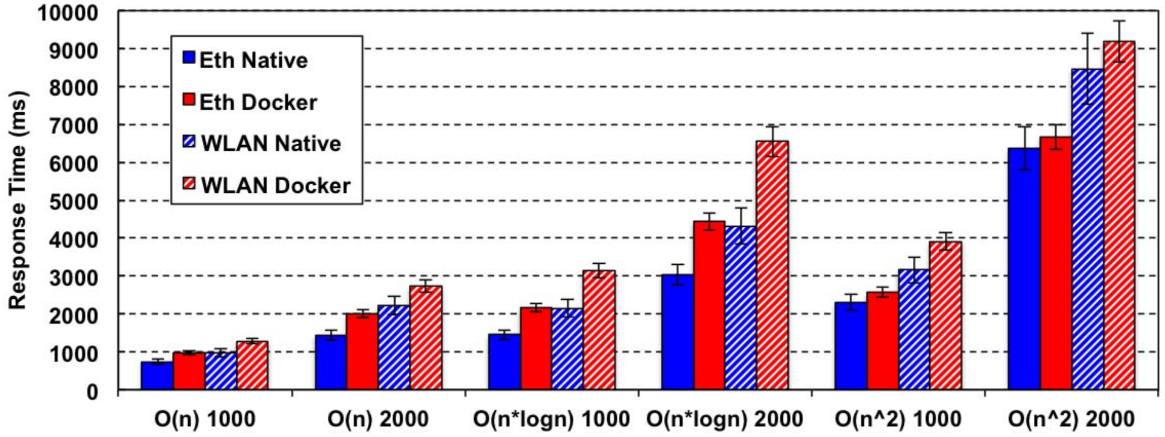
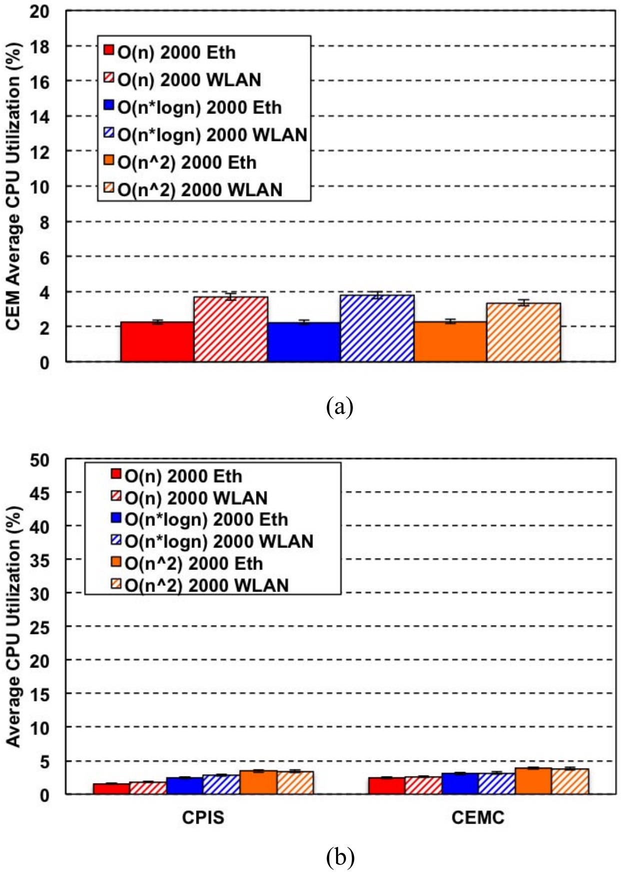


Fig. 5. Response times of computation tasks for different input sizes, comparing the native case to the Docker containerized services, in both Ethernet and WLAN configurations.



(a)

(b)

Fig. 6. Average CPU utilization of (a) CEM and (b) device hosting the CoAP server for different computation tasks.

We also analyze the CPU utilization of the involved devices for the different algorithms. Fig. 6(a) reports the average CPU utilization for the CEM, which performs orchestration features in the CEMC mode. A further measurement campaign has been finalized to analyze the resource performance on the device hosting the containerized server CoAP accounting for both CPIS and CEMC modes. Despite the background management operation provided by the CEM to periodically monitor the server's container status, the average CPU utilization of the device is similar in both CEMC and CPIS modes, as shown in Fig. 6(b).

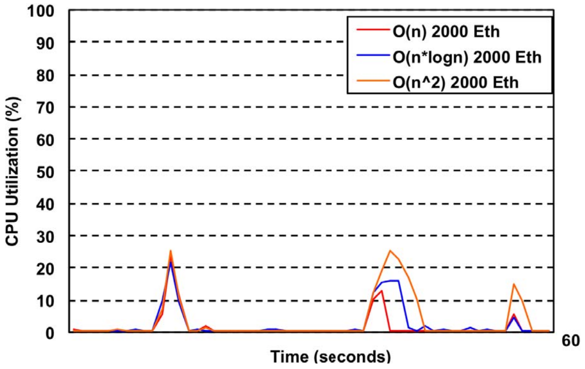


Fig. 7. Comparison of CPU utilization of the device hosting the CoAP server for different computation tasks in CPIS mode.

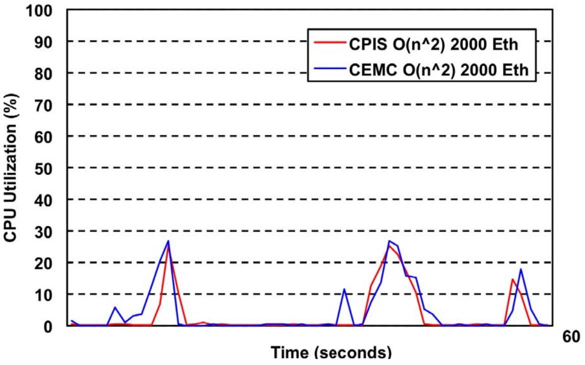
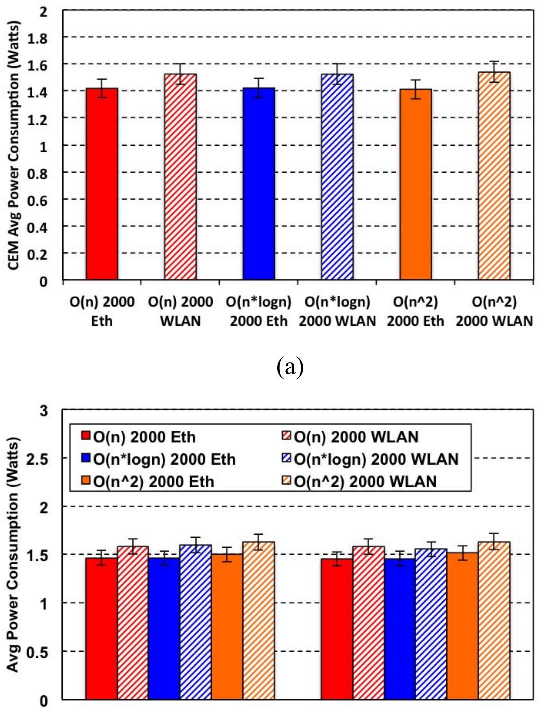


Fig. 8. Comparison of CPU utilization of the device hosting the CoAP server for a computation use case accounting for both CPIS and CEMC modes.

To better characterize the performance of the investigated approaches, in Fig. 7 we show the comparison of the instantaneous CPU utilization for the device hosting the CoAP server, when accounting for different computation tasks in CPIS mode. It can easily be observed that the heavier the complexity of the task, the higher is the relevant CPU utilization experienced by the node. Furthermore, in Fig. 8 we report the instantaneous CPU utilization over the time interval



(a)

(b)

Fig. 9. Power consumption (a) of the CEM, and (b) of the device hosting the CoAP server for different computation tasks.

of 60 s. In particular, in this time interval we include the activation of the containerized CoAP server, the processing of the computation task operation after the reception of the client request, and the final removal of the container. In the CPIS mode, the client itself sends the request to the RPi executing the containerized CoAP server. Whereas, in the CEMC mode, the CEM issues the command to start the container on the CIW. Fig. 8 shows a similar trend for both approaches, with peaks during the activation and removal of the container and, obviously, during task execution. As a consequence, the additional management features provided by the CMEC approach have a negligible impact on the average CPU utilization.

Fig. 9(a) depicts the average power consumption for the CEM, which performs orchestration features in the CEMC approach. As expected, the power consumption is almost equal for all investigated computation tasks; whereas, the wireless case shows a small increase (in the order of 8%) in this parameter compared to the wired case. Fig. 9(b) reports the average power consumption of the device executing the containerized CoAP server for both CPIS and CEMC modes. The reader can observe that: 1) the power consumption smoothly increases when considering algorithms with a higher complexity; 2) the wireless configuration implies a higher power consumption with respect to the wired case; and 3) the power consumption in both CEMC and CPIS modes is similar.

To analyze the cost of the additional management procedures introduced by the CEMC approach, we evaluate the activation times of the container implementing the CoAP server. The measured values, reported in Fig. 10, confirm that the orchestration feature introduces a further delay in terms of activation times. In particular, for the CEMC we consider three different network configurations: 1) Ethernet configuration, where the CEM and the CIWs are connected through a switch (this represents the default wired configuration for the CEMC tests); 2) WLAN setup, where the CEM creates the wireless ad-hoc network the CIWs are attached to; and 3) Ethernet no-switch, where an Ethernet cable is used to directly connect the CEM to a single CIW (this configuration is used only to test the activation times and to better characterize the network delays). The Ethernet no-switch case allows for achieving a lower activation time in the CEMC approach with respect to the WLAN case. The Ethernet case presents the highest activation times due to the additional delay introduced by the communications over the Ethernet switch.

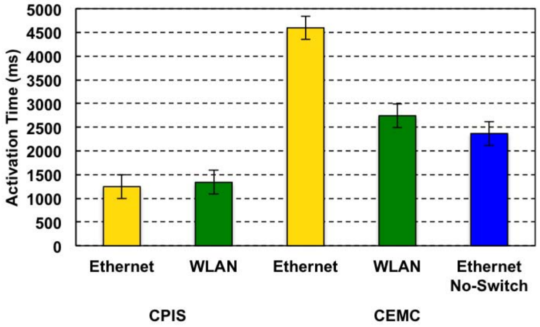


Fig. 10. Activation times of Docker-based containers for different configurations.

To sum up, the experimental results can provide useful guidelines in the choice of the container-based management framework. For delay-sensitive application, the CPIS approach can guarantee a lower delay by exploiting direct interaction between requesting client and devices offering the IoT service. On the other hand, the CEMC management causes a negligible increase of resource consumption, and can better cope with scenarios where clients are resource-constrained and IoT application are highly integrated.

2) Sensing/Actuation Tasks: Sensing and actuation services are the most distinctive features of smart IoT environments. In our framework, to test sensing and actuation operations, we exploit the GPIO interface provided by the RPi board. This interface allows to interact with different sensors and/or actuators. In this paper, six different sensors/actuators are used to perform sensing tasks. Below, we report a more detailed description about the functionality provided by each considered sensor/actuator.

1) *Laser Emitter:* This module is able to emit laser beam, which is widely used in several fields thanks to its good directivity and energy concentration.
2) *Active/Passive Buzzer:* Buzzers are audio signalling devices that can be categorized into active and passive. An active buzzer has a built-in oscillating source, so it will make sounds when electrified, whereas a passive buzzer requires square waves with frequency between 2 and 5 KHz to drive it.
3) *Sound Sensor:* This component detects the sound intensity in the surrounding environment and converts it into electrical signals.

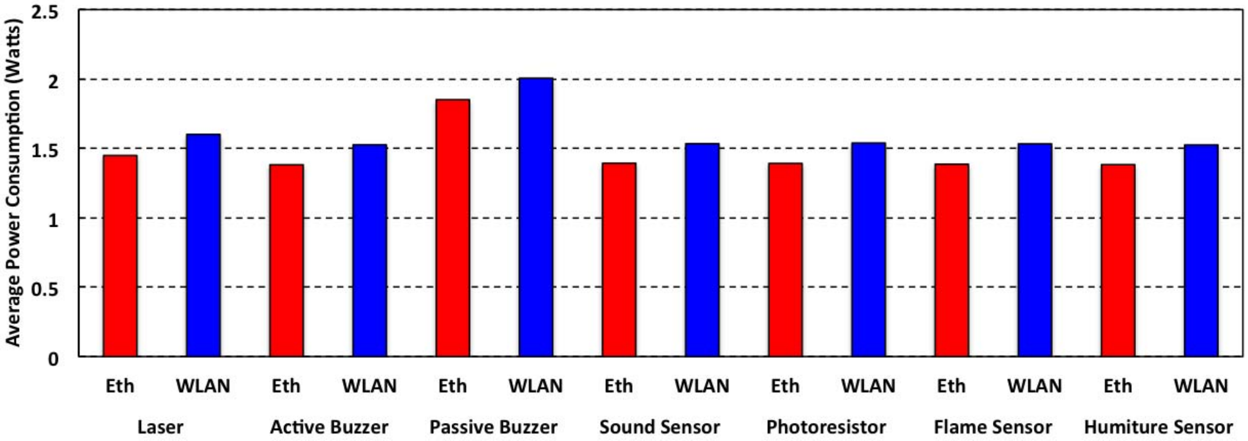


Fig. 11. Average power consumption for sensing/actuation scenarios, in both Ethernet and WLAN configurations.

4) *Photo-Resistor Module:* A photoresistor is a light-controlled variable resistor, i.e., its resistance exhibits photoconductivity and decreases with increasing incident light intensity.
5) *Flame Sensor:* A flame sensor module consists of a flame sensor, resistor, capacitor, potentiometer, and comparator LM393 in an integrated circuit. It can detect infrared light with a wavelength ranging from 700 to 1000 nm. The far-infrared flame probe converts the strength changes of the external infrared light into current changes.
6) *Humidity Sensor:* The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technologies of a dedicated digital modules collection, as well as the temperature and humidity sensing features, allow the product to ensure high reliability and excellent long-term stability.

For this measurement campaign, the client sends a CoAP PUT request for enabling sensing/actuation operation on the CoAP server. Once that the sensor has been activated, the CoAP client issues GET requests toward the server, to either acquire the sensing value measured by the sensor or trigger the actuation service. The GET requests are performed every 5 s within a time interval of 120 s. A final PUT request is sent to turn the sensor off. Unfortunately, Docker Swarm does not have support yet to activate containerized services that require the permission to access modules via GPIO. Therefore, for this use case, all the tests are performed by using the traditional Docker mode, i.e., according to the CPIS approach.

In Fig. 11, we report the average power consumption of the CoAP server for each used sensors. In particular, we can note that the use of the passive buzzer involves a greater power value since it requires square waves with a wide range of varying frequencies. Furthermore, the wireless networking requires a higher power consumption with respect to the wired links. This aspect can be clearly observed in Fig. 12, where the different instantaneous power consumption of Docker-based CoAP server for active buzzer are presented, accounting for both wired and wireless networking. At the beginning and at the end of the observed time interval, peaks of power consumption are, respectively, due to the activation and stopping of the relevant Docker container.

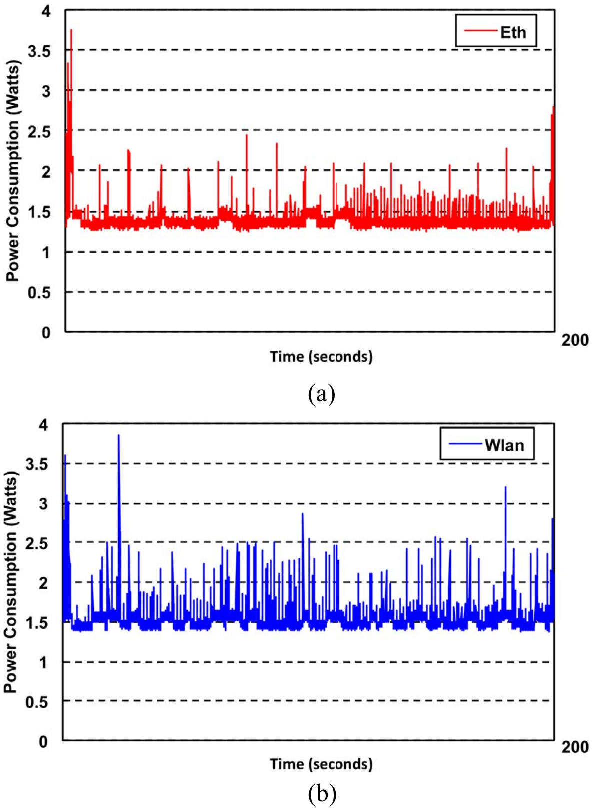


Fig. 12. Power consumption of the device running the Docker containerized CoAP server for active buzzer in (a) Ethernet configuration and in (b) WLAN configuration.

By inspecting the RPi resource usage during the execution of one of the sensing tasks (active buzzer), the lightweight features of the overall operations can be observed. In Fig. 13, although a peak in CPU utilization is reported at the beginning for the container activation, during the remaining part of the sensing operations the CPU utilization is,

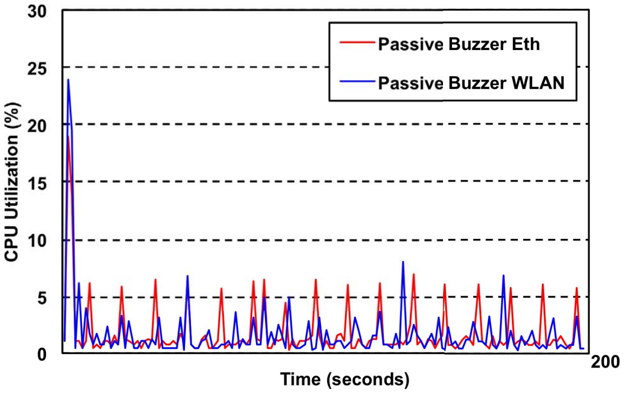


Fig. 13. CPU utilization of Docker containerized CoAP server for active buzzer.

on average, extremely low, i.e., around 1.5%, with major peaks corresponding to the processing of the CoAP GET requests.

3) Video Analytics: The so-called *Internet of Multimedia Things* is a novel paradigm, which envisages a multitude of heterogeneous multimedia things able to interact and cooperate with one another and with other entities connected to the Internet, to facilitate video-based services globally available to end-users [39]. This is why we also performed some measurements in a test scenario in which a USB camera is directly connected to the RPi.

In our testbed, the client performs a CoAP PUT request to activate a video transmission. As soon as the camera (with native resolution equal to 1920×1080) is activated, the video content is encoded in mp4v format, and transmitted through HTTP protocol with a bit-rate of 5 Mb/s. We monitor the performance of the CoAP server during a live streaming session lasting two minutes. To make our scenario more realistic, we consider that ten clients require the video content; these clients are executed in the client RPi.

To also evaluate video processing operations we perform basic video analytics. In particular, a filter of motion-detection is applied to the video content before being transmitted. In another scenario, we apply additional transcoding operation: the video resolution is reduced from 1920×1080 to 320×240 and the bit-rate from 5 Mb/s to 400 kb/s. The overall transcoding process can be considered a heavy workload for the RPi, which in turn generates a higher resource usage, if compared to the Sensing/Actuation tasks. By measuring the maximum system load during the execution of the aforementioned tasks (Fig. 14), we observe that the heaviest system load occurs when the motion-detection filter is applied to the video content and the transmission rate is set at 5 Mb/s. Furthermore, we observe that there is a slight system load increment for the wireless case, and that the transmission bit-rate affects the system load. In fact, even if a motion detection filter is applied, the system load with a transmission rate equal to 400 kb/s is also lower than the base case.

Fig. 15 shows the average power consumption of the RPi server, for the three different cases considered in the video analytics scenario. In particular, the operation with active motion detection and maximum resolution is the most power consuming. The wireless environment implies a small increase in power consumption with respect to wired connectivity.

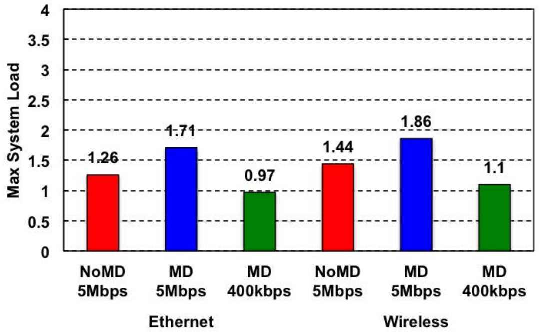


Fig. 14. Max system load of Docker containerized video services with/out motion detection (MD) filter.

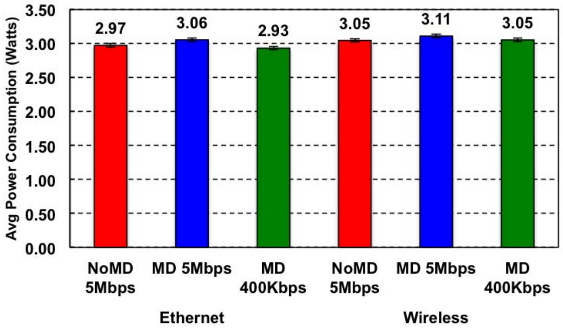


Fig. 15. Average power consumption of Docker containerized video services with/out motion detection (MD) filter.

VI. Open Research Areas

This paper has highlighted that container-based orchestration represents a promising solution to create smart virtualized IoT environments. Nonetheless, we believe that the following features deserve further investigation.

1) *Mobility:* Several use cases consider mobile IoT nodes, which involve a highly time-varying network cluster topology. This can introduce strictly timing requirements in terms of cluster creation and joining, to allow task offloading in a short time interval. Furthermore, mobility introduces two further issues: a) to keep the latency response time short, the CEM may be migrated across different access points, so to be as close as possible to the controlled nodes and 2) the CIWs belonging to the same private/public entity can be deployed under different Edge nodes. In the latter scenario, a distributed management system composed by multiple synchronized CEMs, running on different network access points, is a promising solution to investigate.
2) *Intercluster Service Provisioning:* In [40] the advantages introduced by federations of multiple IoT clusters at the edge of the network has been clearly highlighted. Therefore, in the next future it will be interesting to evaluate the interaction among CIWs belonging to

different clusters that cooperate in providing integrated IoT services.

3) *Semantic Interoperability:* In some IoT contexts—such as smart buildings, smart farms, and so on—the interoperability among different devices can be achieved at the gateway. In [41], the architecture of a gateway able to manage semantic features and to act as an endpoint for the presentation of data to users has been proposed. To provide the desired interoperability, our CEM should be able to appropriately enable interactions between different semantic-based IoT clusters.
4) *Security:* Sharing IoT devices with their relevant resources introduces additional security issues. These will require novel lightweight authentication and trustworthy mechanisms, which need to be integrated with container-based solutions. Furthermore, specific virtual networking schemes should be considered to appropriate manage the sensitive traffic flows among multicluster IoT environments.

VII. Conclusion

In this paper, we evaluated container-based solutions for IoT service provisioning. In particular, we considered two possible operational frameworks: 1) CPIS, which enables direct interactions between devices and 2) CEMC, which introduces a management functionality at the edge of the network to ease the supervision of surrounding devices.

Our analysis in a real testbed demonstrated that lightweight virtualization allows to execute a broad range of IoT applications in both wireless and wired networks while enabling the desired abstraction level. Furthermore, the orchestration framework introduces negligible overhead for small IoT cluster in terms of resource consumption and service activation times, while the advantages in terms of manageability and scalability are remarkable. Finally, a list of promising research areas has been drawn to provide useful guidelines for future works. In the next future, we will address our research toward the evaluation of scalability of the investigated solutions, when accounting for both the impact of multiple containers running on the same device and for an increasing number of devices in the testbed environment.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] *Raspberry Pi*. Accessed on Jan. 2017. [Online]. Available: https://www.raspberrypi.org/

[3] K. Hentschel, D. Jacob, J. Singer, and M. Chalmers, "Supersensors: Raspberry Pi devices for smart campus infrastructure," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, Vienna, Austria, 2016, pp. 58–62.

[4] *Docker Containers*. Accessed on Jan. 2017. [Online]. Available: https://www.docker.com/

[5] *LXC Containers*. Accessed on Jan. 2017. [Online]. Available: https://linuxcontainers.org/

[6] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Tempe, AZ, USA, 2015, pp. 386–393.

[7] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," in *Proc. IEEE INFOCOM Demo*, San Francisco, CA, USA, 2016, pp. 999–1000.

[8] T. Taleb *et al.*, "On multi-access edge computing: A survey of the emerging 5G network edge architecture & orchestration," *IEEE Commun. Surveys Tuts.*, to be published, doi: 10.1109/COMST.2017.2705720.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Edition MCC Workshop Mobile Cloud Comput.*, Helsinki, Finland, 2012, pp. 13–16.

[10] A.-C. G. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Towards a software-defined network operating system for the IoT," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Milan, Italy, 2015, pp. 579–584.

[11] C. Buratti *et al.*, "Testing protocols for the Internet of Things on the EuWIn platform," *IEEE Internet Things J.*, vol. 3, no. 1, pp. 124–133, Feb. 2016.

[12] T. Taleb, A. Ksentini, and R. Jantti, "'Anything as a service' for 5G mobile systems," *IEEE Netw.*, vol. 30, no. 6, pp. 84–91, Nov./Dec. 2016.

[13] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 85–95, 2002.

[14] F. Aslam *et al.*, "Optimized Java binary and virtual machine for tiny motes," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst.*, Santa Barbara, CA, USA, 2010, pp. 15–30.

[15] D. Alessandrelli, M. Petraccay, and P. Pagano, "T-Res: Enabling reconfigurable in-network processing in IoT-based WSNs," in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst.*, Cambridge, MA, USA, May 2013, pp. 337–344.

[16] T. Renner, M. Meldau, and A. Kliem, "Towards container-based resource management for the Internet of Things," in *Proc. Int. Conf. Softw. Netw. (ICSN)*, 2016, pp. 1–5.

[17] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Berlin, Germany, 2016, pp. 202–211.

[18] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016.

[19] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Efficient and scalable IoT service delivery on cloud," in *Proc. IEEE 6th Int. Conf. Cloud Comput. (CLOUD)*, Santa Clara, CA, USA, 2013, pp. 740–747.

[20] I. Farris, T. Taleb, H. Flinck, and A. Iera, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge," *Trans. Emerg. Telecommun. Technol.*, doi: 10.1002/ett.3169.

[21] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," *J. Netw. Comput. Appl.*, vol. 62, pp. 18–40, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2015.11.009

[22] B. Tang *et al.*, "A hierarchical distributed fog computing architecture for big data analysis in smart cities," in *Proc. ASE BigData Social Informat.*, Kaohsiung, Taiwan, 2015, Art. no. 28.

[23] I. Farris *et al.*, "Social virtual objects in the edge cloud," *IEEE Cloud Comput.*, vol. 2, no. 6, pp. 20–28, Nov./Dec. 2015.

[24] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proc. 13th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Hilton Head Island, SC, USA, 2012, pp. 145–154.

[25] T. Penner, A. Johnson, B. Van Slyke, M. Guirguis, and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in *Proc. IEEE Glob. Commun. Conf.*, Austin, TX, USA, 2014, pp. 2801–2806.

[26] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam, "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *Proc. 1st Int. Workshop Mobile Cloud Comput. Netw.*, 2013, pp. 19–26.

[27] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, New York, NY, USA, 2015, pp. 9–16.

[28] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures—A technology review," in *Proc. 3rd Int. Conf. Future Internet Things Cloud (FiCloud)*, Rome, Italy, 2015, pp. 379–386.

[29] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.

[30] A. Sill, "The design and architecture of microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 76–80, Sep./Oct. 2016.

[31] *Kubernetes*. Accessed on Jan. 2017. [Online]. Available: http://kubernetes.io/

[32] *Docker Swarm*. Accessed on Jan. 2017. [Online]. Available: https://www.docker.com/products/docker-swarm

[33] *Apache Mesos*. Accessed on Jan. 2017. [Online]. Available: http://mesos.apache.org/

[34] M. Kovatsch, S. Mayer, and B. Ostermaier, “Moving application logic from the firmware to the cloud: Towards the thin server architecture for the Internet of Things,” in *Proc. 6th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Palermo, Italy, 2012, pp. 751–756.

[35] T. Taleb, “Toward carrier cloud: Potential, challenges, and solutions,” *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 80–91, Jun. 2014.

[36] R. Morabito, R. Petrolo, V. Loscrì, and N. Mitton, “Enabling a lightweight edge gateway-as-a-service for the Internet of Things,” in *Proc. 17th Int. Conf. Netw. Future*, 2016, pp. 1–5.

[37] *txThings—CoAP Library for Twisted Framework*. Accessed on Jan. 2017. [Online]. Available: https://github.com/mwasilak/txThings

[38] *Dstat: Versatile Resource Statistics Tool*. Accessed on Jan. 2017. [Online]. Available: http://dag.wiee.rs/home-made/dstat/

[39] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, “Internet of multimedia things: Vision and challenges,” *Ad Hoc Netw.*, vol. 33, pp. 87–111, Oct. 2015.

[40] I. Farris, L. Militano, M. Nitti, L. Atzori, and A. Iera, “MIFaaS: A mobile-IoT-federation-as-a-service model for dynamic cooperation of IoT cloud providers,” *Future Gener. Comput. Syst.*, vol. 70, pp. 126–137, May 2017.

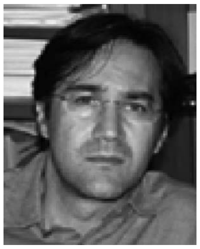
[41] R. Petrolo, R. Morabito, V. Loscrì, and N. Mitton, “The design of the gateway for the cloud of things,” *Ann. Telecommun.*, vol. 72, no. 1, pp. 31–40, 2017.

Roberto Morabito received the master’s degree in computer and telecommunications systems engineering from the University of Reggio Calabria, Reggio Calabria, Italy, in 2013. He is currently pursuing the Ph.D. degree at the Department of Communications and Networking, Aalto University, Espoo, Finland.

He has been with Ericsson Research, Helsinki, Finland, since 2014, and involved in the FP7 ITN METRICS Project. His current research interests include multiaccess edge computing, Internet of Things, and virtualization technologies.

Ivan Farris received the B.Sc. degree in telecommunications engineering and M.Sc. degree in computer and telecommunications systems engineering from the University of Reggio Calabria, Reggio Calabria, Italy, in 2011 and 2013, respectively, where he is currently pursuing the Ph.D. degree in information technology engineering.

He is currently a Researcher with the Department of Communications and Networking, School of Electrical Engineering, Aalto University, Espoo, Finland. His current research interests include Internet of Things, edge computing, and network softwarization.



Antonio Iera (SM’07) received the degree in computer engineering from the University of Calabria, Rende, Italy, in 1991, the Master Diploma degree in information technology from CEFRIEL/Politecnico di Milano, Milan, Italy, in 1992, and the Ph.D. degree from the University of Calabria, in 1996.

Since 1997, he has been with the University of Reggio Calabria, Reggio Calabria, Italy, and currently holds the position of a Full Professor of telecommunications and the Director of the Laboratory for Advanced Research into Telecommunication Systems. His current research interests include next-generation mobile and wireless systems, RFID systems, and Internet of Things.

Tarik Taleb (S’05–M’05–SM’10) received the B.E. degree (with Distinction) in information engineering and M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively.

He is currently a Professor with Aalto University, Espoo, Finland, leading the MOSA!C Laboratory. He was a Senior Researcher with NEC Europe Ltd., Ruislip, U.K., until 2015. Prior to that, he was an Assistant Professor with Tohoku University. His current research interests include mobile core, mobile cloud networking, network function virtualization, software-defined networking, mobile multimedia streaming, and social media networking.

Prof. Taleb is an IEEE ComSoc Distinguished Lecturer.