Journal of Network and Systems Management (2023) 31:14
https://doi.org/10.1007/s10922-022-09702-3

Empowering SDN-Docker Based Architecture for Internet of Things Heterogeneity

Intidhar Bedhief[1] · **Meriem Kassar**[1] · **Taoufik Aguili**[1]

Received: 19 January 2022 / Revised: 13 September 2022 / Accepted: 27 October 2022 /
Published online: 22 November 2022

Abstract

Internet of Things (IoT), an emerging technology, connects billions of smart devices to the Internet through heterogeneous communication technologies enabling novel services and applications. This complex and variable environment encounters several challenges such as managing and controlling the network and IoT applications, programming IoT devices, responding to some specific Quality of Service requirements of certain applications. The current IoT network based on the complex, static, and closed traditional architecture represents a serious concern, especially with the amounts of data generated from the huge and heterogeneous IoT devices. In this paper, we present a new model for IoT architecture that brings together the benefits of Software Defined Networking (SDN) architecture and containerization (i.e., Docker) to address most IoT challenges and abstract the underlying communication and hardware subsystem. The SDN concept decoupled the control plane from data plane which implies a centralized global network view and provides flexibility in network configuration and resource management. Moreover, containerization technique through the docker in the device brings the portability and scalability of the applications. First, we propose, an SDN-based architecture for IoT. Then, with lightweight virtualization of IoT device supplied by Docker, we empower the functionalities and effectiveness of the SDN based architecture solution to challenge the IoT heterogeneity. Furthermore, we use the smart supermarket as a use case to validate our SDN-Docker based architecture in a real IoT scenario. The experimental results validate the architecture implementation.

Keywords Internet of Things (IoT) · Software defined networking (SDN) · Containerization · Docker · Scalability · Heterogeneity · Architecture · Smart supermarket · Virtualization

✉ Intidhar Bedhief
Intidhar.Bedhief@enit.utm.tn

Extended author information available on the last page of the article

1 Introduction

The Internet of Things has created a digital revolution by connecting all existing things around us to the Internet emerging novel applications and services. The thing varied from laptops, smart-phone, home appliances, surveillance equipment, industrial system, e-health device, to any sensor and actuator that can be geographically anywhere and connected over any wireless communication technology such as Bluetooth, ZigBee, WiFi, or cellular network at any time [1]. The spread of IoT smart devices integrated with their abilities of sensing, computing, and communicating in real time is making every aspect of human life smarter and more efficient; such as the shopping experience that becomes seamless thanks to the IoT. Thus, we can either have a virtual shopping experience ended by getting all you need to your home without getting out or moving from the chair. Otherwise, we go to the supermarket, get a real smart shopping experience, and at the end, we just walk out. The last experience is guaranteed due to the widespread cameras and sensors that detect and transmit data to the system that charges us while we walking out.

The IoT devices (cameras, sensors, etc.) are connected to each other and to the Internet in order to exchange data. The connection could be maintained for a limited period of time while providing specific services. The IoT environment has a very dynamic topology and scale network. Consequently, IoT needs a flexible network architecture. As well, the amount of data generated and traffic represents a major bottleneck for the network. In addition, the heterogeneity, complexity, and proprieties limitation that characterize the devices represents an issue for management and performance optimization of the network. The traditional network devices are vertically integrated: the control and data planes are coupled and embedded in the same network device. Therefore, the integration of new networking features can be harder. So, the traditional network architecture that seems to be rigid, complex, and closed can not be a solution even with the introduction of a structure as a "gateway" [2]. In fact, the deployment of a huge number of IoT devices, with different designs and protocols in a complex environment, increases problems of mismatch in protocols and capabilities that affect applications. Sequentially, the application needs to be scalable and efficient especially, for the real-time type. So, to meet the IoT requirements in terms of heterogeneity, flexibility, and interoperability, a new architecture vision is needed. Researchers focused on recent technologies such as Software Defined Networking (SDN) or containerization (e.g., Docker) as potential solutions.

SDN is an emerging technology that guarantees network flexibility and agility answering IoT requirements in terms of heterogeneity and flexibility. The SDN concept of decoupling the network control plane from the data plane enables the network programmability and centralizes the network control providing a global view. The direct programmability of the network allows to configure, manage, optimize, and secure the forwarding devices dynamically. Docker is the container technology that allows developers to isolate their applications from their environment and gives them the ability to innovate with their chosen tools, application stacks, and deployment environments. Hence, Docker is a lightweight approach that provides portability for developers to create and deploy an application without complexity.

On one hand, SDN has been used along with IoT in various studies [3–5] addressing different IoT challenges related to the network performance optimization. On the other hand, Docker has been used with IoT devices especially the gateway taking advantage of the application portability and scalability [6–8]. In literature, the two technologies are used separately for IoT. We considered, in our previous work [9], a first idea that introduces the concept of combining them together in the IoT architecture. However, several questions arise when applying SDN and Docker in IoT. It is important to identify the IoT challenges that emerge the integration of new technologies in the IoT architecture. Secondly, it is a key to specify the need of SDN for IoT and answers the questions of why SDN for IoT and what it brings to IoT. Thirdly, it is crucial to identify the needs of docker with SDN for IoT. In this paper, in order to answer all those questions, we study the effectiveness of SDN-Docker architecture for IoT heterogeneity; and we test it in the smart supermarket use case. Further, firstly, we reveal the main IoT challenges. Secondly, we give an overview of the two technologies SDN and Docker. Then, we propose an SDN-based architecture for IoT. Finally, we describe and evaluate it.

The rest of the paper is organized as follows. Section 2 provides a background on IoT challenges and the network softwarization techniques such as SDN and the dockerization. Also, it reviews the state of the art of integration SDN and IoT in a first part and IoT and Docker in the second one. Section 3 describes our proposed SDN-Docker based architecture for IoT. Section 4 describes the implementation of our proposed architecture in the case of a smart supermarket and evaluates its performances. Section 5 concludes our paper.

2 Background and Related Work

In this section, we firstly present the IoT general characteristics and issues. Secondly, we introduce the softwarization techniques (i.e., SDN and Docker) as well as their related research to IoT. Finally, we discuss related researches on SDN and Docker for IoT.

2.1 IoT Challenges

With technology evolution (Web service, Cloud, ICN (Information-Centric Networking), etc.) and the integration of IoT in various domains, many definitions have been released by different organizations and researchers describing the concept of IoT. Although the variety of its definition, some common features of the IoT system have been pointed out: the variety of the "thing" (physical object or virtual entity), interconnection of things, the network ubiquity (i.e., availability of the network "anytime" and "anywhere"), and heterogeneity of communication technologies.

The IoT system is made up of very heterogeneous and resource-constrained embedded devices. It is mainly characterized by the heterogeneity that can be present in devices, networks, applications, services, and/or data. IoT devices can have heterogeneous architecture, they can be based on different MCU (MicroController

Unit) architecture (e.g., Intel 8051/52, ARM, x86, AMD64) and system resources. They are equipped with at least one network interface and they can come up with different networking communication technologies (e.g., WiFi, 4G/LTE, Ethernet, 5G, etc.) [10]. Besides, the network connectivity of the IoT system is dynamically variable: The IoT network composes of a huge number of devices where some are always connected while others connect and disconnect for a period of time. This is due to energy saving or variable network conditions. Likewise, the running applications are various and dynamic in terms of latency, bandwidth, and deployment environment. All of the above characteristic remarks have led to the huge advancement of IoT services and applications, but also reveal many problems that could be summarized in heterogeneity. Here, the main challenge is finding flexible layered architecture that handles billions or trillion of connecting devices and their dynamic topology. Hence, heterogeneity can be accompanied by flexibility and interoperability challenges for an IoT environment. It is therefore challenging to manage the network and control the huge and heterogeneous number of devices that exchange various information through heterogeneous communication protocols. Also, the IoT system has to be able to add new devices, services, and functions for customers without negatively affecting the quality of existing network services, which is a challenge. The main IoT challenges are cited in Table 1.

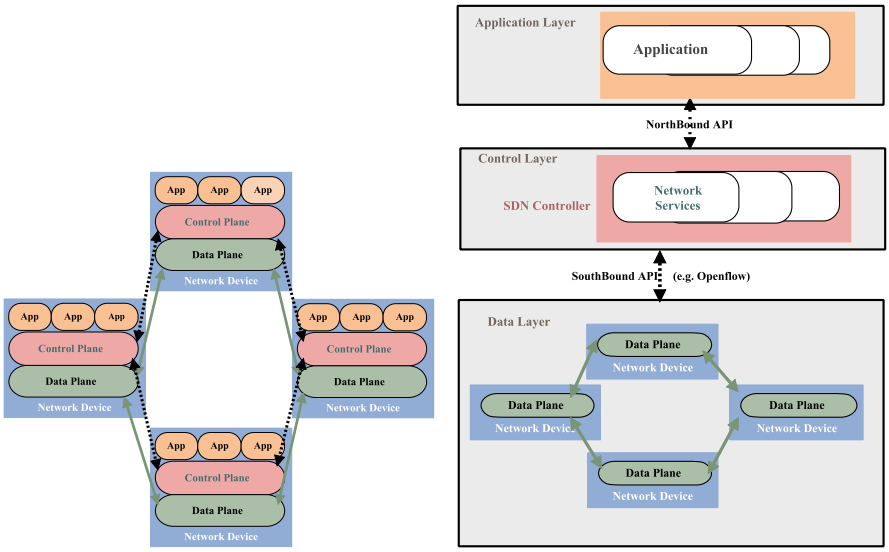
2.2 Software-Defined Networking for IoT

In traditional network architecture, the network device hardware that is responsible for the connectivity is built along with the controller, as shown in Fig. 1a, that contains the forwarding rules. This design was considered early the best way to guarantee network resilience, as a crucial design goal [11]. In today's existing communication network, the control plane needs to be managed anytime from anywhere. So, for a dynamic, large, and complex network such as the IoT network, the model with a static architecture could not fit the previously mentioned requirements.

Instead, SDN architecture, shown in Fig. 1b, separates the network control from the data forwarding functions in order to make network management more flexible. Figure 1 highlights the difference between the two architectures. SDN enables the networking devices on the data plane to be dynamically controlled and configured

Table 1 IoT challenges

Challenges	Description
Heterogeneity	Management of various devices/services/technologies
Scalability	Ability to add new devices, services, and function
Security	Protection mechanism that provides privacy, integrity, confidentiality, and authenticity
Interoperability	Interaction and communication between various technologies
QoS	Guarantee IoT application and service QoS (bandwidth, latency,)
Flexibility	Dynamic management and configuration of IoT devices



(a) Traditional Network Architecture **(b)** SDN Architecture

Fig. 1 Traditional vs SDN network architecture

by the controllers on the control plane. Network applications (i.e., routing, switching, monitoring, firewalls, load balancing, etc.), in the Application layer, can be programmed and managed by SDN controller through the Northbound API (ex. REST API). The Control layer is decoupled from the Data layer and it is logically centralized having a global view of the whole network. The controller can be one physical entity or multiple entities (i.e., physically distributed). Then, the controller guides the network devices to forward the packets or flows through the Southbound API (ex. OpenFlow).

Splitting the control and the data plane gives the flexibility, dynamicity, and cost-effectiveness to the network architecture to adapt to today's applications such as IoT applications [12]. Indeed, SDN architecture enables a global view of the entire network, including its resources. Besides, it improves the network flexibility through programmability i.e., new network services can be provided, the priority of packets can be detected or even block it [13].

Integrating SDN in the IoT architecture has been proposed as a solution to resolve traditional network issues and network management optimization by many researchers. Several papers [3–5] have surveyed the integration of SDN in IoT in many perspectives. In our paper, we discuss works that considered such integration on heterogeneity and architecture challenges. The discussed solutions can be arranged in two sets: the first one focuses on the SDN controller [14–16]. The second one proposed instead a framework [17–19].

Regarding the first approach, Wu et al [14] proposed Ubiflow, the first software defined IoT system which uses a decentralized SDN controller for flow control and mobility management in urban multi-network. It divides an urban scale into different

geographic partitions controlled by a physically distributed SDN controller. Also, the authors, in their paper [15], presented an original SDN controller design in IoT multi-network that enables flexible, effective, and efficient management on task, flow, network, and resources. Martinez et al. [16] proposed an SDN based architecture for heterogeneous devices connecting with each other using IPv6 and creating an IoT controller over the SDN controller in order to simplify the management.

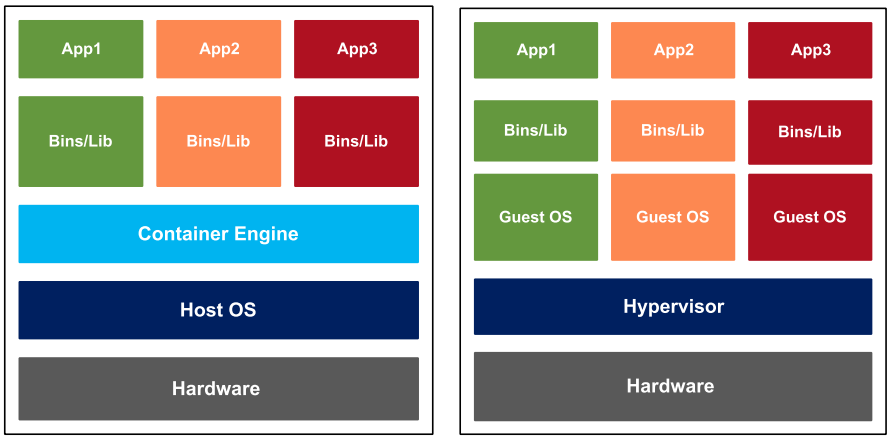
The second approach used a framework such as SDIoT, a software defined based framework for IoT in order to challenge traditional system architecture [17]. Huang et al. [18] proposed a framework to manage and control the IoT network by exploiting the benefits of SDN. The proposed framework focused on Machine to Machine (M2M) transactions. The authors of the paper [19] proposed a framework called SHIOT that classifies the incoming user requests based on an open ontology. SHIOT utilizes the Lagrange relaxation theory to find the optimal path to forward the requests to the destination nodes.

SDN plays an essential role in the management of heterogeneous network. But, applying only SDN in IoT is it enough to overcome the previously mentioned issues? Some researchers empower SDN with NFV such as the contribution of Ojo and al. [20]. Tomovic and al. propose a Fog node solution along with SDN to solve the IoT challenges [21]. In this paper, we propose an SDN-Docker based architecture for IoT, presented in Sect. 3 and evaluated in Sect. 4.

2.3 Dockerization for IoT

Containerization is an operating-system-level virtualization concept that packages an application and all its dependencies, configurations, and other details to be easily deployed. So, it runs rapidly and reliably from one computing environment to another [22]. Containers and hypervisors (i.e., virtual machines) are both two virtualization techniques that share the same benefits of resource isolation and allocation, but they provide different abstraction levels regarding virtualization and isolation. Hypervisor virtualizes hardware and device drivers providing an abstraction of physical hardware. Thus, it generates a higher overhead. As shown in Fig. 2b, the entire guest OS (Operating System) is virtualized on a host OS enabling the simple use of several OSs and making the operating environment heavy and slow. In contrast, containers provide an abstraction at the application layer avoiding such overhead by implementing process isolation at the operating system level. A single container instance combines the application with all its dependencies as shown in Fig. 2a. It runs as an isolated process in user space on the host OS, while the OS kernel is shared among all the containers [23].

In our proposed architecture, we will apply Docker, as a container-based virtualization platform [22] that provides portability for developers to develop, deploy, and run applications without complexity. Since docker virtualizes the OS without adding overhead, it provides an easier and faster way to deploy IoT applications. It offers portability through packaging applications with the necessary resources. Docker containers are created using base images that can include just the OS fundamentals, or it can consist of a sophisticated pre-built application stack ready for launch. When



(a) Container-based Architecture **(b)** Hypervisor-based Architecture

Fig. 2 Container vs Hypervisor based architecture

building images with Docker, each action taken (i.e., command executed, such as apt-get install) forms a new layer on top of the previous one. Commands can be executed manually or automatically using Dockerfiles. Each Dockerfile is a script composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image to create a new image.

Container-based virtualization through Docker has been applied to the IoT. It was proposed in order to get over the deployment issues especially for distributed applications and to utilize its advantages in terms of portability of services and applications. Firstly, some works [24–26]studied the impact of running docker container on IoT devices in terms of performance. Morabito et al. [24] compared the performances of a running docker on a Raspberry Pi 2 to native performance (i.e., without including any virtualization layer) in terms of CPU, Memory I/O, Disk I/O, and Network I/O. They concluded that the impact of the container virtualization layer is almost negligible. In another paper [26], one of the previous authors extended the work in a way that he analyzed in depth the performance of running docker on Raspberry Pi 3, Odroid C1+, Odroid C2, and Odroid XU4. Also, Ruchika [25] evaluated the performances of running IoT applications in a container and over the host aiming at proving the ability of container technology to satisfy the requirement of the IoT application. In a second step, there were researches that proposed docker technology for IoT edge and gateway to alleviate its functionalities. Xu et al. [8] proposed an architectural design and the implementation of docker-based framework for SDN switch. The framework enabled auto-docking/undocking of applications at the edge switches and managed the storage, computing, and networking resources of the switch. Also, Morabito et al. [6] presented and tested a Lightweight Edge Gateway for the Internet of Things (LEGIoT) architecture that enhances basic gateway features through two different virtualized modules: one charged the communication with the Internet and the other managed the exchanges with sensors. Alam

et al. [7] proposed a modular and layered architecture that oers containerized services and micro-services. The modularity and the orchestration in the architecture that is supplied by Docker, simplified the management and enabled the distributed deployments.

As it can be seen, according to the mentioned works, the Docker technology is used along with IoT devices especially in the Edge thanks to its potentiality in the field of distributed applications and data processing services.

2.4 Related Work

SDN has been proposed, mostly, as a solution to resolve traditional network issues and network management optimization for IoT networks [27]. Similarly, Docker is used all along with Fog and Edge nodes thanks to its advantages in terms of portability of services and applications. However, SDN and Docker have been also applied together in IoT for network management issues. Authors in their paper [28] proposed Aloe, an SDN control plane orchestration framework for IoT. The proposed framework uses a Docker container to support lightweight migration capable in-band controllers. Furthermore, Yang et al. [29] propose a simulation experiment platform for the IoT called ImuLab. The imuLab platform is based on OpenStack, containers, and the existing IoT emulator (VMNet). Besides, Okwuibe et al. in [30] study the energy and latency trade-offs obtained by combining various technologies (SDN, Edge, IIoT, and Docker). Also, in [31], they propose an approach called SDRM (Software Defined Resource Management) that manages dynamically resources for IIoT and proposes an integration of SDN along with edge and cloud computing. The work [32] proposes an IoT-aware multilayer transport software defined networking and edge/cloud orchestration architecture.

Florita et al. [33] proposed a resilient system that is dependent on SDN and container to ensure the reliability and functionality of IoT devices when internet access is inconsistent or slow. Fawwaz et al. [34] proposed a distributed MQTT broker for edge resources to reduce network traffic and data delivery latency. Authors [35] propose CRMS (Cloudless Resource Monitoring System), a self-configuration orchestrator for advanced IoT applications. The proposed system can monitor the different performance aspects of the resource allocation among multiple hosts of a fog computing system interconnected by SDN. Bolatti et al. [36] propose a portable IoT network monitoring system using Docker container and bridge networking with SDN.

In the previously cited related work, Docker is mostly used to enhance the scalability of the SDN control plane and the edge layer in order to optimize the IoT environment performances.

3 SDN-Docker Based Architecture for IoT

In this section, to address IoT challenges previously cited, we propose a new architecture for IoT based on two described technologies: SDN and Docker. First, we present our general SDN-based architecture for IoT. Then, we add the docker to our architecture.

3.1 SDN-Based Architecture for IoT

Our main objective for developing the IoT architecture is to enhance the flexibility and programmability of the network and so it will be easier to introduce new IoT devices and services. Moreover, IoT architectures have been defined to bring the necessary flexibility to interconnect heterogeneous devices. Several architectural models have been presented in the literature. The basic model is a three-layer consisting of Perception, Network, and Application Layers. However, in recent works, some other models, where they introduce the middleware layer between the network and application layer and five-layers model that contains objects, objects abstraction, service management, application, and business layers, have been proposed in order to add more abstraction to the IoT architecture. Still, with the traditional network architecture, the management, control, and monitoring of the IoT network is extremely complex and rigid. However, introducing the SDN model in the IoT architecture brings a new abstraction level. The SDN-based architecture provides a clear separation of control and data plane, which is the key enabler for more flexibility and programmability in the IoT network.

Figure 3 depicts our proposed SDN-based architecture for IoT. We propose a layered architecture with more abstraction of the network layer where we integrate the model SDN. Moreover, SDN enhances the flexibility and agility of the IoT network by adopting the flexibility of dynamic configuration guaranteed by the programmability of the control plane.

As shown in Fig. 3, our proposed SDN based architecture for IoT composes of four layers:

- *Perception layer* contains various IoT devices like sensors, actuators, mobiles, laptops, and every smart embedded electronic devices. The main assignment is capturing and collecting information from the surrounding environment. Then, the IoT device establishes a connectivity with other heterogeneous devices to share data or within the gateway.
- *Edge/Fog layer* where Edge node provides more powerful computation and processing resources for IoT devices. While, Fog node brings the cloud closer to the thing. So, the Edge/Fog layer can be considered as an intermediate layer that provides an extra resources layer. Fog and edge nodes can be switches, gateways computers, and I/O devices that provide storage, computing, and connectivity services. They can contain also SDN switches and controllers.

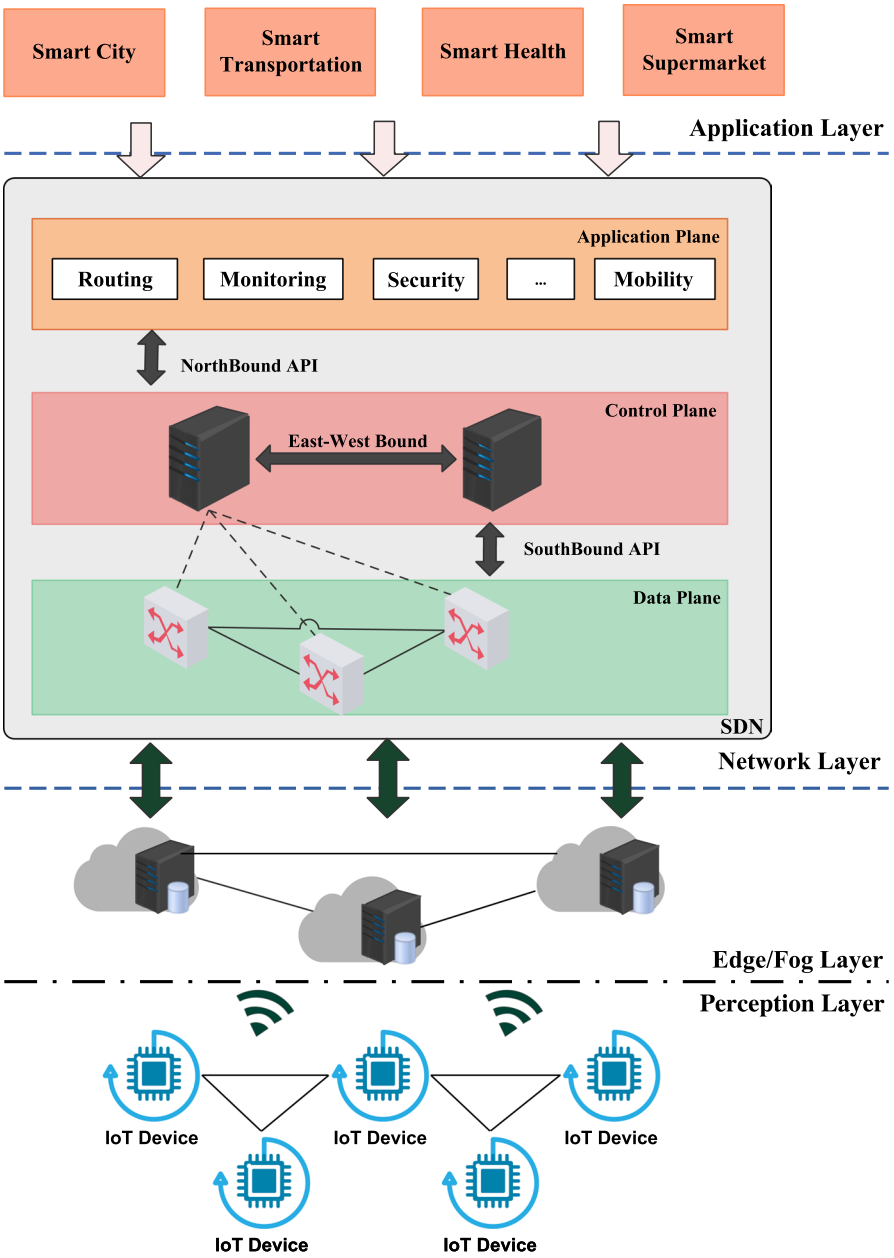


Fig. 3 SDN based Architecture for IoT

- *Network layer* virtualizes the IoT access network devices. Thus, we guarantee the abstraction of this layer. With the concept of SDN that separates the control actions from the forwarding, the network administration, control, and

monitoring can be automatic and dynamic. This layer contains the SDN controllers and SDN switches (i.e., Open Virtual Switches OVS, Open Flows Switches OFS). The SDN controller can be centralized or distributed according to the network requirements.

- *Application layer* represents the IoT applications or services such as Smart Home, Smart city, Smart health, or Industrial IoT. It is the most visible part of the architecture to the end user. The application layer is where data and collected information are managed and processed.

With applying SDN based model in the IoT architecture, we guarantee the programmability of the IoT network and its flexibility. Although it simplifies the management and control of the network, SDN needs more enhancements to cope with the IoT environment requirements. Therefore, we integrate the Docker technology in our proposed architecture.

3.2 Our Proposed Architecture

IoT covers an heterogeneous system with various devices connected to the Internet and to each other without human intervention. IoT architecture should be flexible. Thus, the two technologies SDN and Docker provide best practices and experience to manage and control IoT networks and applications. Using SDN for network management will simplify its configuration and control. And applying Docker will provide application portability and hardware abstraction. As depicted in Fig. 4, our proposed SDN-Docker based architecture for IoT shows the connection of IoT device (integrating Docker) through the SDN enabling network that is monitored and controlled by the SDN controller.

Furthermore, we propose to encapsulate the IoT application and services such as smart city, smart transportation, and others into a container software (i.e., Docker). The use of containerization gives many advantages. First, containerization

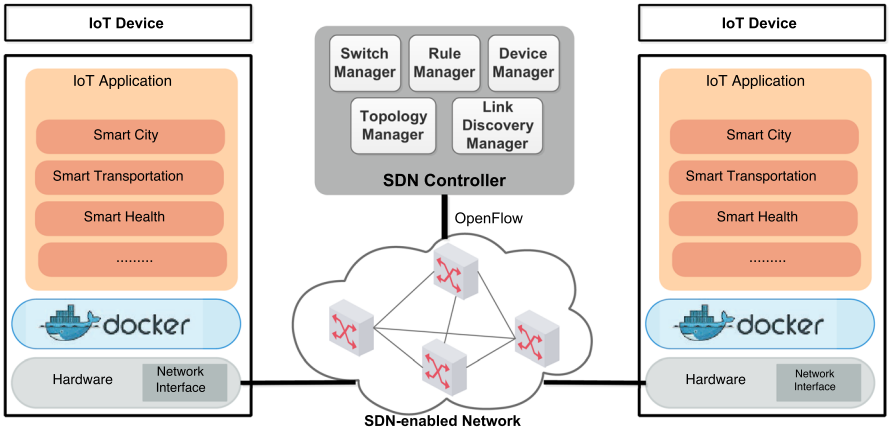


Fig. 4 SDN-Docker based Architecture for IoT

guarantees the scalability of the deployment of applications and services. With docker, it is easier to integrate new IoT device and to update the application of the existing one. Second, docker guarantees the abstraction of the hardware. Third, docker is a standard lightweight open-source software.

We apply SDN for the IoT network in which devices are connected through the SDN network.

The architecture components shown in Fig. 4 are the following:

- *IoT Device* is heterogeneous hardware that uses its network interface to communicate with other IoT devices. It can be a sensor, mobile phone, laptop, or any other physical entity that has computation, communication, and storage capabilities. The device hosts a Docker container that runs the IoT application.
- *SDN Controller* is the brain of the network, and it can be a one central controller or several controllers executing the control functions cooperatively. In this paper, we use one centralized controller. The controller has a global view of the network. The SDN controller configures switches and gateways with rules for forwarding data to the corresponding devices. It is responsible also for topology management.
- *SDN-enabled Network* contains SDN switches that are responsible for data forwarding in the network.

As we detailed previously, the key requirement in the IoT environment is the control and the configuration of heterogeneous entities and networks. SDN and Docker, as emerging technologies for IoT, provide an abstracted interface to upper layers by covering underlying layers. So, Docker-based approach provides higher flexibility and less overhead. It is applied to provide entity management while SDN provides a dynamic configuration and management of the network.

4 Test Scenario and Results

In this section, we present the IoT use case scenario "Smart supermarket" where our architecture can be applied. So, firstly, we describe the use case scenario. Then, we present the deployment scenario and test environment. Finally, we discuss the experimental results.

4.1 Use Case "Smart Supermarket"

Smart supermarket is a new innovation that uses IoT devices to collect physical data and transmit them to a digital platform making the shopping experience seamless for the customers. The application of smart supermarket is being smarter all along with the evolution of IoT. Therefore, the technology used in the smart supermarket has evolved from control technology using RFID and sensors to "just walk out" technology that uses further artificial intelligence, machine learning, and image recognition. Thus, IoT sensors can be associated with things used in the supermarket like

freezers and shelves to control and collect data. The temperature sensor can be associated with a freezer to control temperature and send an alarm when it gets to the limit. Also, using IoT, smart shelves can transmit collected data to the responsible for analysis. The information, for example the most appeal product, helps in making a decision for the arrangement of products. Recently, the supermarket has been using sensors and cameras to track customers and detect chosen items, then automatically charge them when they leave. Figure 5 shows the supermarket use case. Many companies such as Amazon, Kroger, and Intel implemented smart supermarkets. Further, with the COVID-19 pandemic, the smart supermarket application is getting more attention, especially with autonomous checkout technology to avoid the line and keep safe.

In this paper, we propose an SDN-Docker based architecture to enhance the flexibility and programmability of the network. The smart supermarket is the best application to apply our architecture. In the supermarket, once the customer enters, he gets identified. Then, there will be multiple cameras that will detect him and follow him along the shopping process. Besides, there will be sensors to detect goods that have been chosen. Then, collected data will be sent to the system. Hence, the topology, as well as application deployment, will be dynamic and scalable. Our proposed architecture will be applied in this scenario to enhance the application and network management and deployment.

4.2 Test Environment

To evaluate the SDN-Docker based architecture for IoT smart supermarket, we run an emulation using Mininet-WiFi, Containernet, and the ONOS controller. We have carried out this emulation using three virtual machines (Oracle VM VirtualBox) that have been installed on a HP Pavilion Gaming laptop that has Intel Core i7-9700 CPU 2.60 GHZ and 16 GBytes of memory. In the first one, we implemented Mininet-Wifi to create the network topology. In the second, we deployed ONOS controller. In the third machine, we implemented Containernet.

ONOS [37] is an open source SDN controller joined to the Linux Foundation in October 2015. Its first prototype was released by Open Networking Lab in 2012. It is written in Java and provides a distributed SDN applications platform. ONOS

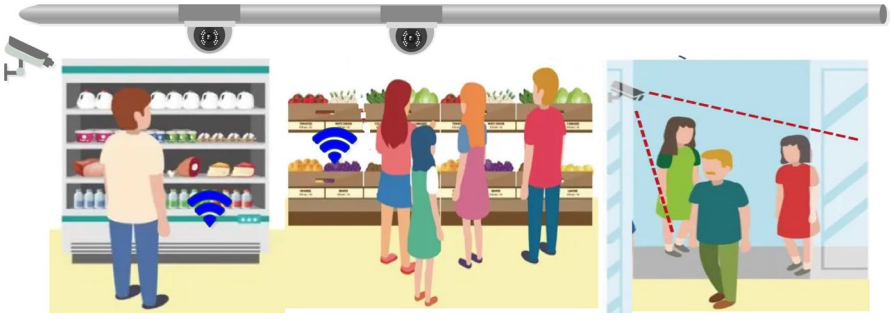


Fig. 5 Smart supermarket

provides availability, scalability, and high performance that enables the management of the distributed network. It can function as a single entity as well as multiple servers.

Mininet-WiFi [38, 39] is an extension of Mininet [40, 41] that supports further virtualized WiFi stations and Access Points (AP). Mininet is a container-based network emulator that runs various types of hosts such as end terminals, switches, routers, and links on a single Linux kernel and it provides the capability of instantiating a set of virtual nodes, which can be connected to form any arbitrary network. In Mininet, the host behaves like a real machine in which the running programs can send packets through an Ethernet and/or a WIFI interface behaving like a real interface, with a given link speed and delay. It is also able to create OpenFlow enabled switches required for the SDN environment.

Containernet is a branch of Mininet network emulator. It enables the use of Docker containers as hosts in emulated network topologies. With containernet, Docker containers can be connected to the topology (i.e., to switches, other containers, or legacy Mininet hosts). Also, it enables executing commands inside containers by using the Mininet CLI. In addition, Containernet enables adding and removing hosts as well as containers/Docker to a running Mininet topology. And, it permits to control traffic links (i.e., delay, bandwidth, loss, jitter).

4.3 Experimental Scenarios and Results

The smart supermarket defines a dynamic topology where there are multiple cameras and sensors that detect and transmit data. Once the client enters into the market, he

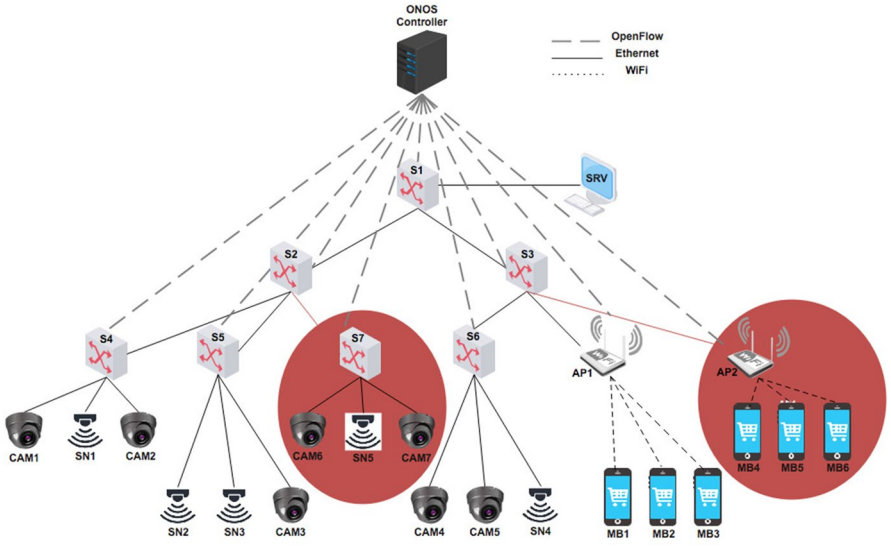


Fig. 6 Experimental architecture of the first scenario

gets identified and followed by IoT devices (i.e., cameras and sensors). So, the topology will get bigger during the shopping time while more devices will be involved in the reading movement and choices of the client. Figure 6 shows the experimental topology of the "Smart supermarket" and Table 2 summarizes the emulation parameters. To evaluate the performance of our proposed SDN-Docker based architecture, we run an emulation using two scenarios. The objective of the first scenario is to prove the scalability and flexibility of the network. While the second scenario aims to prove the application flexibility and scalability.

4.3.1 First Scenario

The first scenario represents a dynamic topology implemented with Mininet-WiFi. The topology as shown in Fig. 6 contains, in the first place, six Open vSwitch (OVS) and one Access Point (AP). They are all connected to a centralized remote controller (ONOS) which manages the traffic flow from a second VM. Moreover, the topology of Fig. 6 can be considered a hierarchical model which is suitable for achieving synchronization in IoT. More, synchronization, as defined in [42, 43], is the coordination of a group of devices to harmonize the execution of a task by time alignment. So, related to our topology, we can find in the lower level the IoT devices which send a request to the OVS (second level) to forward a packet. The OVS in its turn asks the controller for a path. So, the controller which is the single point of synchronization sends back the rules. Besides, it sends periodically LLDP packets to check the network status.

There are five camera hosts as clients holding a video server based on VLC media player named respectively CAM1, CAM2, CAM3, CAM4, and CAM5. There are three sensors, SN1, SN2 and SN3. The topology contains also three mobiles, MB1 MB2 and MB3. SRV represents the receiver of all flows. We have configured the bandwidth values of each link on Mininet-WiFi: the links between CAMs and the OVS are sets to a bandwidth of 10 Mbits/s. The links between SNs and OVS are set to a bandwidth of 250 Kbits/s. At the running time, we expand the topology (i.e., the added nodes highlighted in the red zone). Firstly, we add an OVS named S7 connected to the controller. We add two cameras named respectively CAM6 and CAM7 and a sensor SN5 connected to S7. Secondly, we add an access point named AP2

Table 2 Emulation parameters

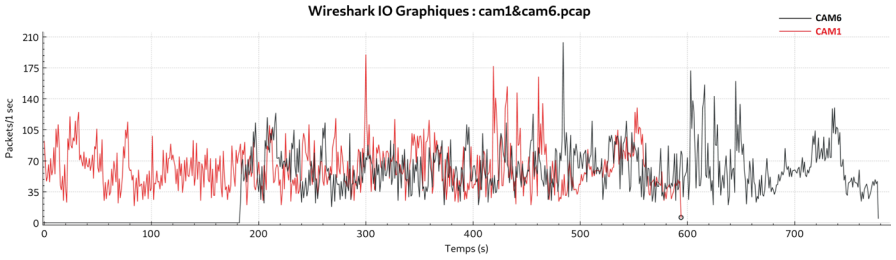
Parameters	Values
Emulator	Mininet 2.5 And Containernet
SDN controller	ONOS
Video server and client	VLC
Traffic generator	iPerf
Emulation Time	800 seconds for scenario 1
	200 seconds for scenario 2
Number of Hosts	19
Number of Switches	9

connected to the controller. We add MB4, MB5, and MB6 connected to AP2. The different IoT devices (CAMs and SNs) follow a time alignment that we define during the emulation. Besides, the controller is centralized and it synchronizes the state of the network. The emulation of the topology lasts about 800 s.

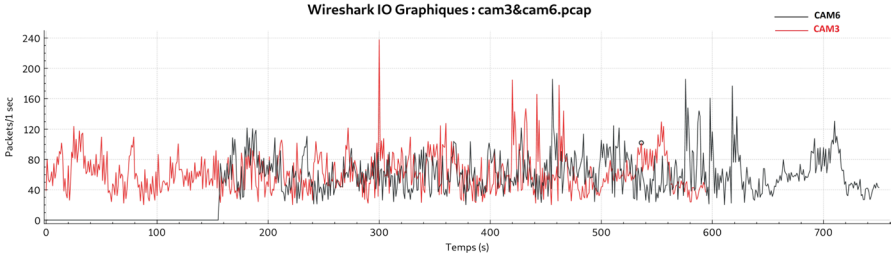
Two types of traffic are used: (i) An animation video, Big Buck Bunny, with a high-definition resolution and its duration of 9.57 mins has been used for streaming over the cameras ; (ii) The data traffic is generated using iPerf. The traffic is captured using tcpdump command and the results are analyzed using the network protocol analyzer, Wireshark.

The emulation starts by streaming video using VLC over CAM1 to SRV and CAM3 to MB3 as well as generating data traffic using iPerf from SN1 and MB1 to SRV. So, a video traffic started between CAM1 and SRV. CAM1 runs VLC media player as client to stream the video using RTP (Real-Time Transport Protocol), which uses the UDP (User Datagram Protocol) protocol. SRV is listening to the video on the port 5004. The same, a video traffic started between CAM3 and MB3.

At the running time, we scale the topology, for the first time, by adding the OVS S7 and the devices CAM6, SN4, and CAM7. CAM6 starts streaming the video flows from the adding moment (i.e., at t = 180s) to SRV. Figure 7a and b represent respectively the amount of packets transmitted from CAM1 to SRV and CAM3 to MB3 compared to the traffic generated from CAM6 to SRV. As we can observe, in the two figures, the transmission rate has an excessive variation. Still, they have the same behavior. CAM1 and CAM3 have the highest number of the transmitted packet at second 300 that get over 175 packets for CAM1 and 240 packets for CAM3.

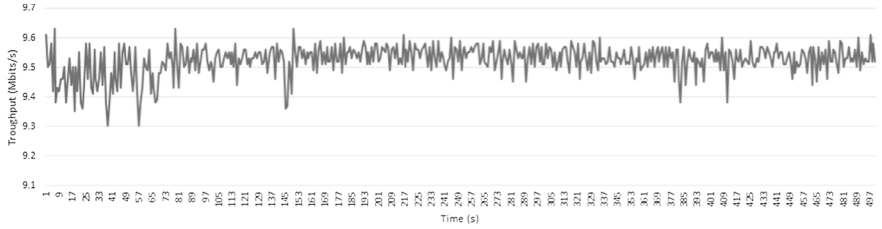


(a) Transmitted packet rate of CAM1 and CAM6 to SRV

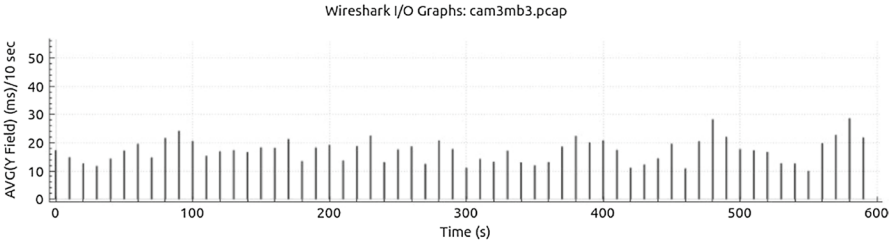


(b) Transmitted packet rate of CAM3 to MB3 and CAM6 to SRV

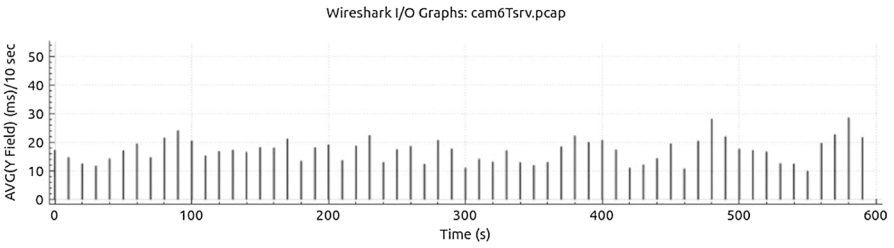
Fig. 7 Video traffic during emulation time



(a) Video streaming delay of CAM1



(b) Video streaming delay of CAM3

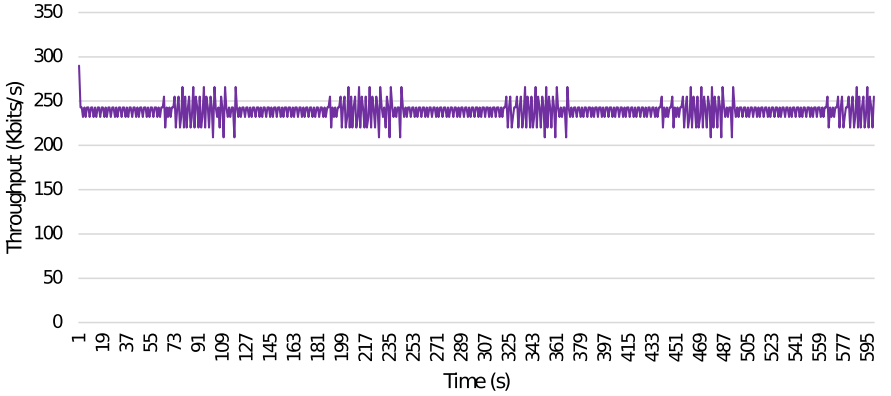


(c) Video streaming delay of CAM6

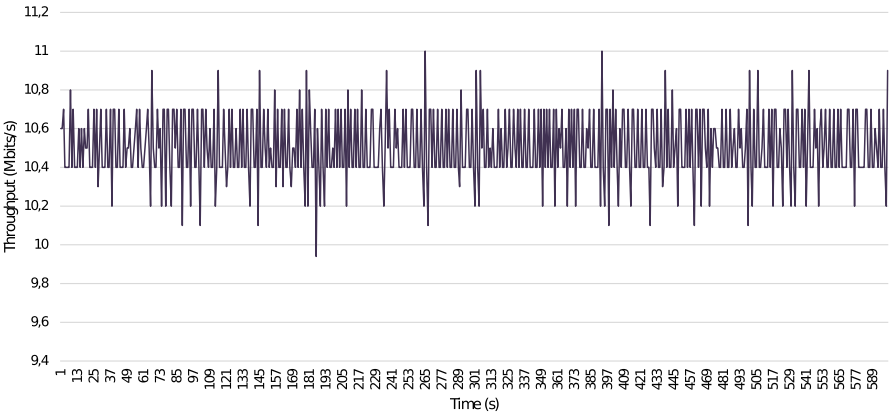
Fig. 8 Video streaming delay

Moreover, Fig. 8a and c represent respectively the delay of streaming a video over CAM1 and CAM6 and listening by SRV on port 5004 and 5006 respectively. Whereas, Fig. 8b shows the delay of video streaming between CAM3 and MB3. As we can see, the delay of the three cameras varied between 10 ms and 30 ms.

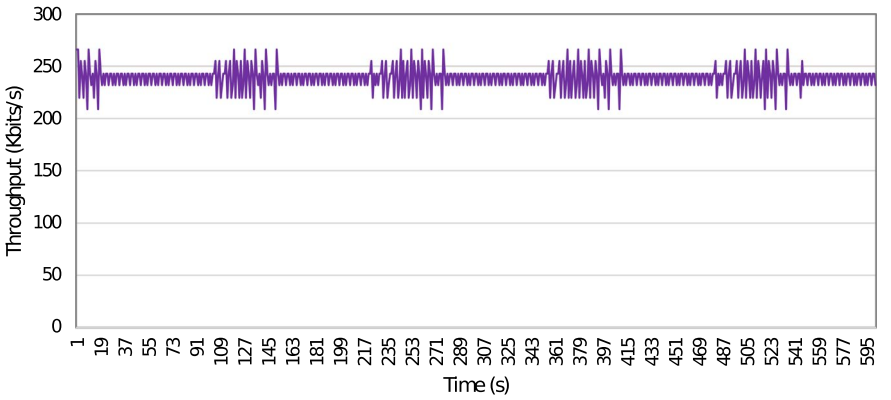
As we mentioned before, a data traffic is started by sensors and mobiles nodes, at the same time, with the video traffic generation. Figure 19 shows the throughput of the transmitted data from the sensors and mobiles to the server (SRV) and the exchanged data between different IoT devices. Figure 9a represents the throughput results between SN1 and SRV which is maintained stable with an average of 240 Kbits/s. Figure 9b represents the throughput of exchanged data between MB1 and SRV which is varied between 10.4 and 10.6 Mbits/s. Figure 9c shows the throughput between two IoT devices, SN2 and MB3. The average



(a) Throughput of data traffic between SN1 and SRV



(b) Throughput of data traffic between MB1 and SRV



(c) Throughput of data traffic between SN2 and MB2

Fig. 9 Data traffic at the emulation start time

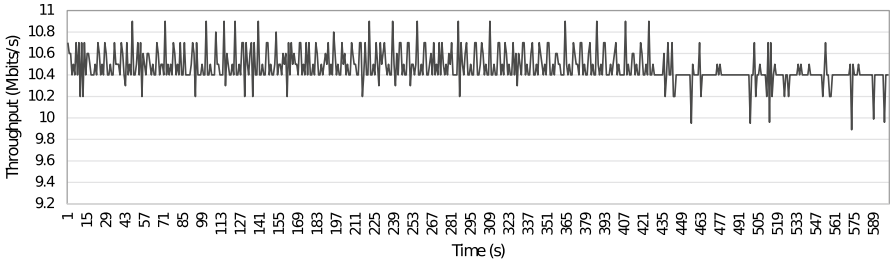
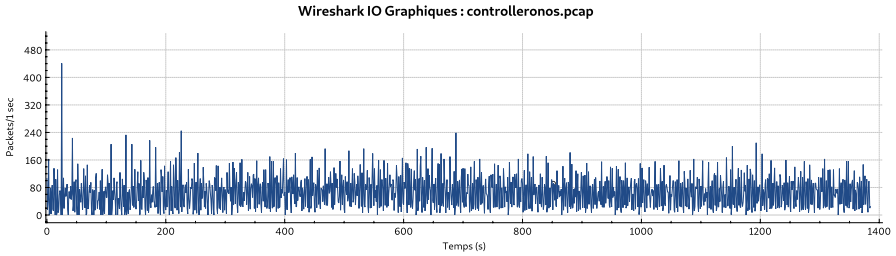
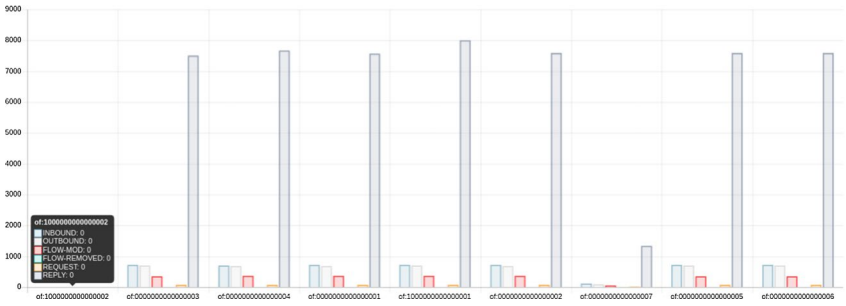


Fig. 10 Throughput of data traffic between MB5 and SRV at the emulation running time



(a) OpenFlow exchanged packets between ONOS controller and OVSs



(b) Number of different OpenFlow messages for different OVSs

Fig. 11 ONOS controller monitoring

throughput generated between IoT devices is 240 Kbits/s, which is more stable in comparison to the evaluation in Fig. 9a.

We scale the topology for the second time and we add AP2, MB4, and MB5. MB5 starts sending data flow to SRV at the adding moment. Figure 10 represents the throughput generated between MB5 and SRV that is stable with an average of 10.5 Mbits/s. We observe that the variation in the throughput started to decrease from the 430 s, which is the end time of previous emulations.

Throughout the emulation time, we captured the OpenFlow traffic between the topology (i.e., Mininet-Wifi) and the Controller (ONOS). The rate of the exchanged traffic is presented in Fig. 11a. The height number of the exchanged packets is 440

packets recorded around 20 s which is the time of starting the emulation. The other peaks did not get over 240 packets and they are corresponding to the time of the topology change.

Also, we run the monitoring application over ONOS to see the recorded messages that have been exchanged between the running switches and the controller at a specific moment. Figure 11b represents the amount of exchanged OpenFlow packets at the time of adding the second access point (AP2). As expected, the value of all exchanged AP2 messages are set to zero. The switch S7 has the lowest value since it is just added to the topology (100 s before AP2). While for the other switches, the number of reply packets overcome 7000 packets and reached the 8000 messages for AP1 .

In summary, these all results validate the flexibility and scalability qualities of our proposed architecture. The performances of the various devices (i.e., CAM1, CAM3, SN1, SN2, and MB1) have not been decreased while we add new devices and links during the emulation. Even the quality of the streamed video is maintained all along the emulation time.

4.3.2 Second Scenario

In the second scenario, we will use Containernet [44] to emulate our topology since we will use Dockers instead of hosts. The topology, as shown in Fig. 12, consists of 7 Open vSwitch (OVS) and 12 Docker hosts where 6 of them (CAM1 to CAM6) will be considered as Docker hosts and 6 considered as Docker sensors (SN1 to SN6). The OVS is controlled and managed by the ONOS controller. The bandwidth of links between S3 and S6 and between S2 and S4 is set to 10 Mbits/s. Also, the

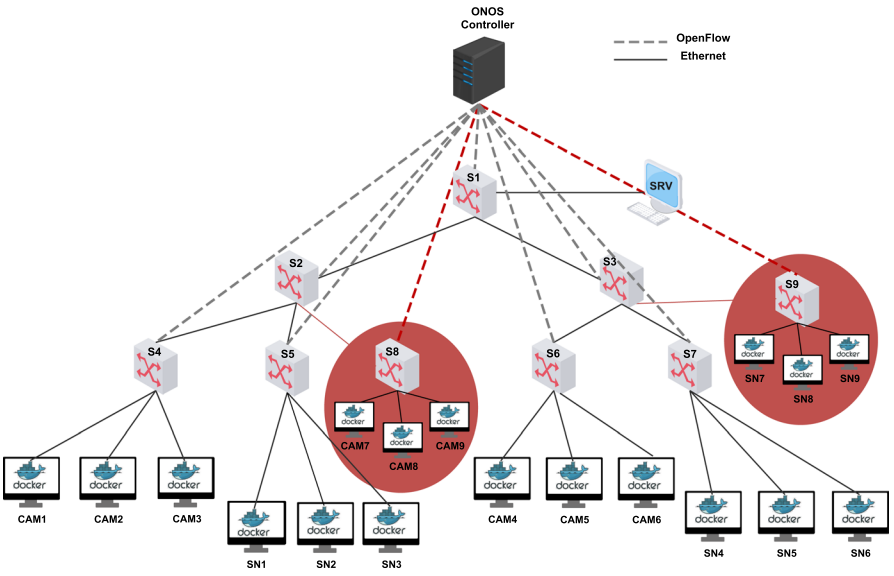


Fig. 12 Experimental architecture of the second scenario

bandwidth of the links between S2 and S5 and between S3 and S7 is set to 250 Kbits/s. The emulation of the second scenario lasts 200 s and we used the emulator containernet. The dockers of CAM1 to CAM6 and sensors SN1 to SN6 are implemented as Docker containers while SRV remains an ordinary host. To evaluate the flexibility and the scalability of the topology, we run two applications with two different protocols. We use MQTT (Message Queuing Telemetry Transport) protocol for the IoT data transmission and TCP (Transmission Control Protocol) for data traffic generation.

MQTT is an application protocol that runs over the TCP/IP protocol using the publish-subscribe pattern. For the docker containers of CAM1 to CAM6, we create a DockerFile where we build an Ubuntu image, install all the dependencies, and the iPerf application to generate the TCP traffic. For the docker containers of sensors, we create a second DockerFile where we build another Ubuntu image, install dependencies, and install the Mosquitto, the MQTT broker, and MQTT client.

Like the first scenario, the IoT devices containing dockers (CAMs and SNs) follow a time alignment during the emulation. The centralized controller synchronizes the state of the network. So, at the starting emulation time, we activate the MQTT broker on SRV. Then, we take SN1 and SN6 respectively as subscriber and publisher. Hence, SN6 starts publishing the message which is the temperature, and SN6 subscribes to request it from the broker (SRV) who is listening at port 1883.

Figure 13 represents the throughput evaluated during emulation time. The average throughput generated between SRV, SN1, and SN6 is about 1400 Bits/s. As we can observe there is a period couple of peaks. They represent ping request and ping response between once the broker and the publisher and the other between broker and subscriber.

At the same time, we start the generation of TCP traffic with iPerf on CAM2 to SRV and from CAM1 to CAM6. Figure 14a and b represent respectively the throughput between CAM2 and SRV and CAM1 and CAM6. The two figures have the same behavior as they maintain stability with an average throughput equal to 9 Mbits/s.

Figure 15a and b represent respectively the RTT delay between CAM1 and CAM6 and between CAM2 and SRV. We can see that differently from the throughput where they have almost the same value, they have also different delay values. The delay between CAM1 and CAM6 is varied between 0 and 15 ms while the delay between CAM2 and SRV is varied between 0 and 5 ms.

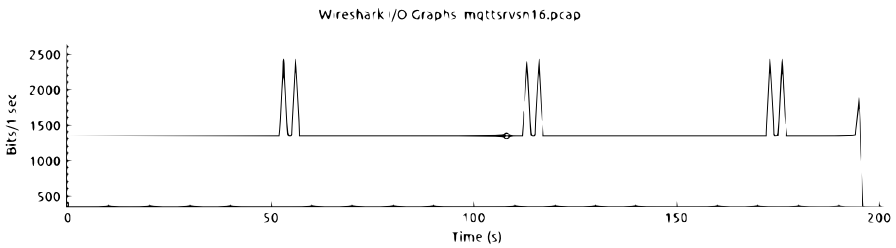
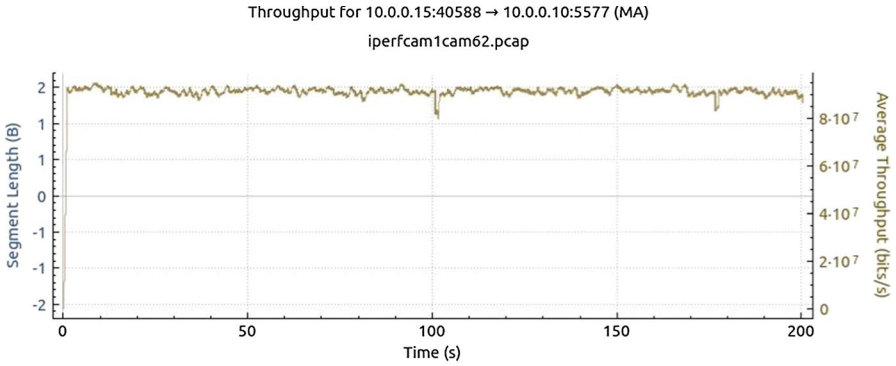
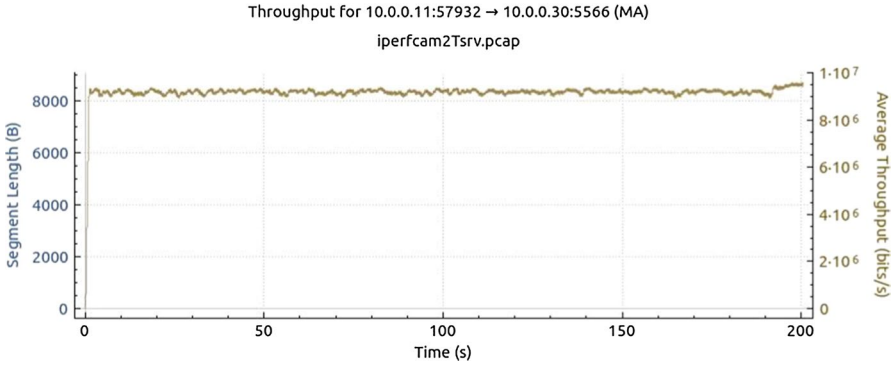


Fig. 13 MQTT throughput



(a) Throughput of data traffic between CAM1 and CAM6

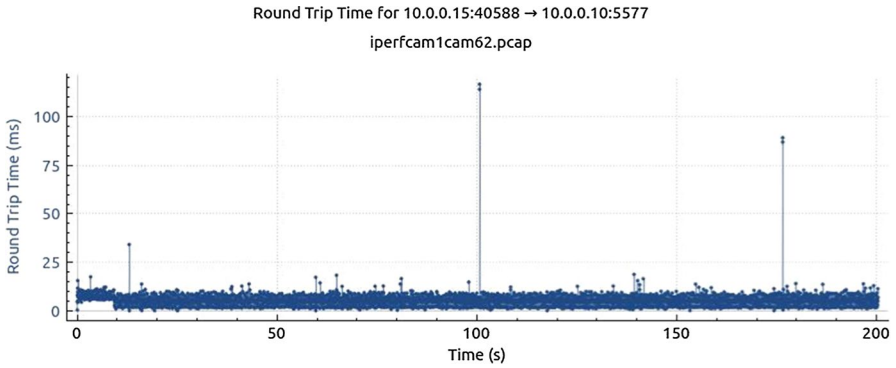


(b) Throughput of data traffic between CAM2 and SRV

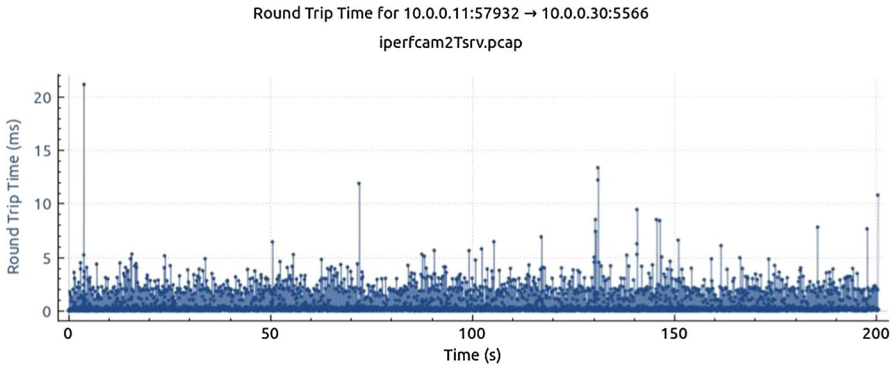
Fig. 14 Throughput evaluation of docker hosts

In the second scenario, we also captured the OpenFlow message exchanged between Containernet where we implement our topology and ONOS. Figure 16 shows the amount of packets exchanged. As shown in the figure, the highest number of the exchanged packets is 80 packets.

In the second part of the second scenario, we test the scalability of the SDN-Docker-based architecture. We define a time alignment for the IoT devices (docker hosts) to follow during the emulation. So, we start again with the topology in Fig. 12 where all nodes are docker containers. We start the evaluation by generating TCP traffic over CAM1 to SRV, activating the MQTT broker on SRV, and starting the publishing of temperature (i.e., message) by SN1. Then, we scale the topology, for the first time at t=30s, by adding the OVS S8 and the three docker hosts CAM7, CAM8, and CAM9. CAM7 starts generating data to SRV. Figure 19a represents the throughput of exchanged data between CAM1 and SRV. The average throughput generated is about 9.53 Mbits/second. Also, Fig. 19b represents the throughput generated between the added docker hosts



(a) Delay between CAM1 and CAM6



(b) Delay between CAM2 and SRV

Fig. 15 Delay evaluation of docker hosts

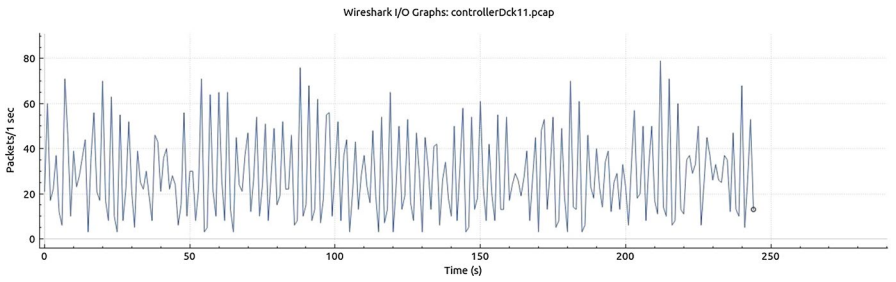


Fig. 16 OpenFlow message exchanged between ONOS and OVSs

CAM7 and SRV. The average throughput generated is 9.52 Mbits/s. Similarly, we generate a TCP traffic between SN5 and CAM2. The average generated throughput represented in Fig. 19c is 241 Kbits/s. We can compare the results with those in Fig. 9 of the first scenario (i.e., SDN-based architecture). We

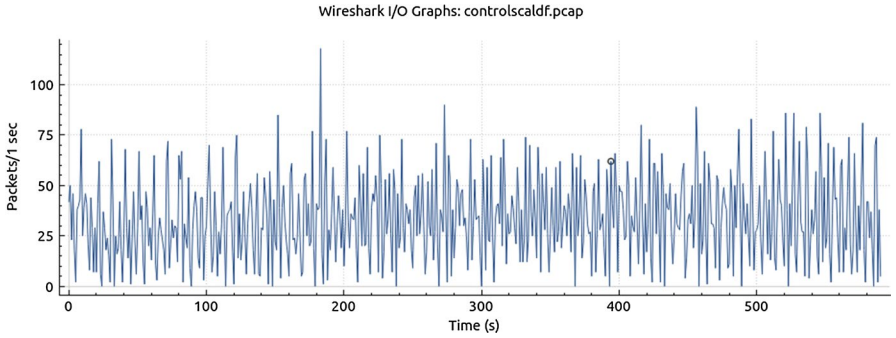


Fig. 17 OpenFlow message exchanged between ONOS and OVSs at scale time

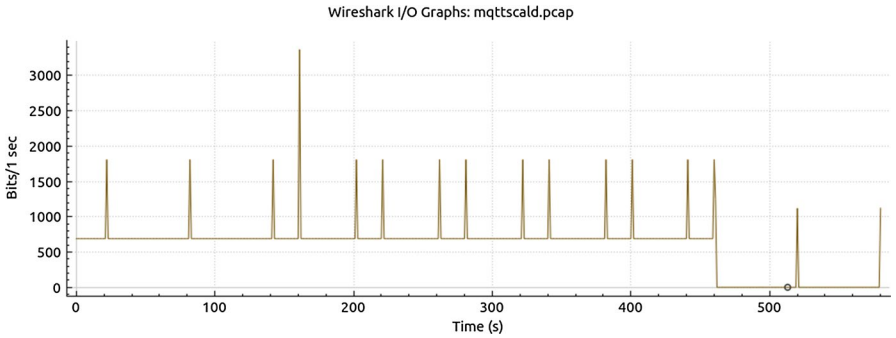
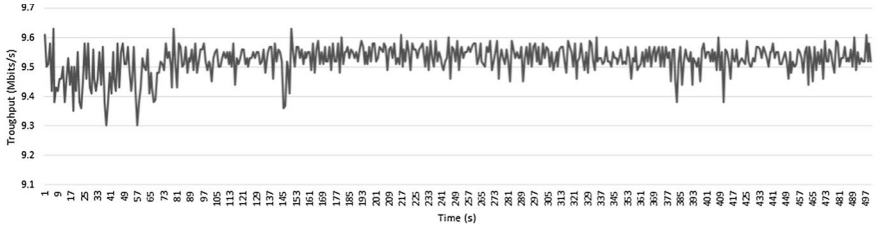


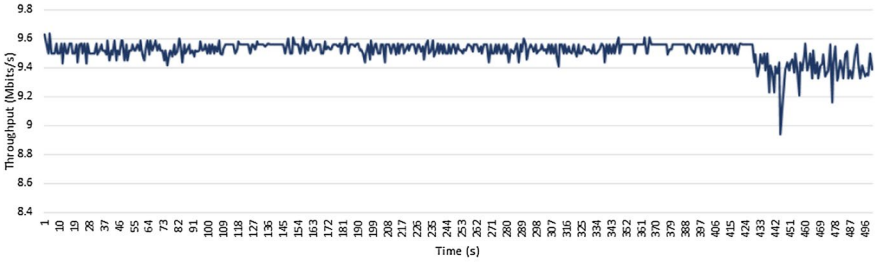
Fig. 18 MQTT throughput at scale time

notice that the throughput decreased. But, compared to the results of the first part of the second scenario where there is no scalability, we notice that there is no impact. Next, we scale the topology for the second time (100 s after the first scale) and we add S9, SN7, SN8, and SN9. At the adding time, SN8 subscribes and requests the message (temperature) from the broker (SRV). Figure 18 represents the flow rate between SN1, SRV, and SN8. We can notice that the peaks are not a couple and that is because the publishing and the subscribing are alternatives. Finally, Fig. 17 represents the amount of exchanged OpenFlow packets between ONOS and the switches. We notice that the flow is more intensive and the peak gets over 100 packets.

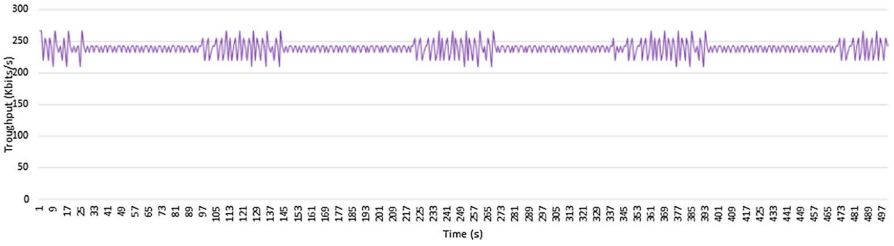
To summarize, in the second scenario, we deployed a topology where the hosts are Dockers. In those Dockers, we build two different images of Ubuntu. Then on top of them, we installed in one iPerf and in the other Mosquitto and MQTT client and server. Next, we start the emulation by generating flows from different nodes. The obtained results validate our objective in terms of scalability and flexibility of the architecture.



(a) Throughput of data traffic between CAM1 and SRV



(b) Throughput of data traffic between CAM7 and SRV



(c) Throughput of data traffic between SN5 and CAM2

Fig. 19 Data traffic at scale time

5 Discussion and Conclusion

In this paper, we proposed a scalable and flexible architecture in order to face IoT challenges such as heterogeneity and scalability. Firstly, we study the IoT challenges as well as the network softwarization techniques (i.e., SDN and containerization). Secondly, we describe an SDN-based architecture for IoT. Finally, we proposed an SDN-Docker based architecture for IoT. Furthermore, we use the smart Supermarket use case to validate our architecture. Indeed, the topology of the Smart supermarket is very dynamic and scalable which is considered the best application of our proposed architecture. In this paper, we focused on testing our SDN-based architecture as well as our SDN-Docker-based architecture. The obtained results demonstrate the flexibility and scalability of both architectures. Furthermore, it demonstrates that the

use of SDN improves the management of the heterogeneous and scalable IoT network since the performance does not decrease when scaling the network of both scenarios. Also, the results of the second scenario in which we apply docker in the IoT devices demonstrate the application scalability. Although, we notice a decrease in the throughput performance when we use docker in the devices compared to the native devices (i.e., the first scenario) but it remains stable during the scalability of the network.

This paper is considered as our work's basis and it can be extended to answer other IoT requirements. Certainly, some questions still remained such as the impact of the architecture on the delay especially for some IoT applications such as IIoT which is time-sensitive [45]. Additionally, the use of docker and containerization in general in the IoT device can be discussed. Therefore, we continue enhancing our architecture. We already started with boosting the Docker to the Edge/fog node in order to reduce the delay and to answer the IIoT requirements in [46].Also, we are working on applying the containerization in the Fog node so that we apply the face recognition and movement detection applications on captured videos. Besides, with the use of Kubernetes, as an orchestrator of Dockers, we can guarantee the auto-scaling of applications. At the same time, in our future work, we intend to manage the mobility of IoT. Therefore, the work of Moufakir et al. [47] can be considered in our research. Finally, we are working on distributing the SDN control plane to guarantee more network scalability.

References

1. Eleonora, B.: The internet of things vision: key features, applications and open issues. Comput. Commun. **54**, 1–31 (2014)
2. Mishra, P., Puthal, D., Tiwary, M., Mohanty, S.P.: Software defined IoT systems: properties, state of the art, and future research. IEEE Wirel. Commun. **26**(6), 64–71 (2019)
3. Bera, S., Misra, S., Vasilakos, A.V.: Software-defined networking for internet of things: a survey. IEEE Internet Things J. **4**(6), 1994–2008 (2017)
4. Alam, I., Sharif, K., Li, F., Latif, Z., Karim, M., Biswas, S., Nour, B., Wang, Y.: A survey of network virtualization techniques for internet of things using sdn and nfv. ACM Comput. Surv. (CSUR) **53**(2), 1–40 (2020)
5. Tayyaba, S.K., Shah, M.A., Khan, O.A., Ahmed, A.W.: Software defined network (sdn) based internet of things (iot) a road ahead. In: Proceedings of the International Conference on Future Networks and Distributed Systems, pp. 1–8 (2017)
6. Morabito, R., Petrolo, R., Loscri, V., Mitton, N.: Legiot: a lightweight edge gateway for the internet of things. Future Gener. Comput. Syst. **81**, 1–15 (2018)
7. Alam, M., Rufino, J., Ferreira, J., Ahmed, S.H., Shah, N., Chen, Y.: Orchestration of microservices for iot using docker and edge computing. IEEE Commun. Mag. **56**(9), 118–123 (2018)
8. Xu, Y., Mahendran, V., Sridhar, R.: Sdn docker: enabling application auto-docking/undocking in edge switch. In: Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE (2016)
9. Bedhief, I., Kassar, M., Aguili, T.: Sdn-based architecture challenging the iot heterogeneity. In: Proceedings of the 2016 3rd Smart Cloud Networks & Systems (SCNS). IEEE, pp. 1–3 (2016)
10. Zikria, Y.B., Yu, H., Afzal, M.K., Rehmani, M.H., Hahm, O.: Internet of Things (IoT): Operating System, Applications and Protocols Design, and Validation Techniques. Elsevier, Amsterdam (2018)
11. Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)

12. Bedhief, I., Kassar, M., Aguili, T.: From evaluating to enabling sdn for the internet of things. In: Proceedings of the 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA). IEEE, pp. 1–8
13. Li, Y., Su, X., Ding, A.Y., Lindgren, A., Liu, X., Prehofer, C., Riekki, J., Rahmani, R., Tarkoma, S., Hui, P.: Enhancing the internet of things with knowledge-driven software-defined networking technology: future perspectives. Sensors **20**(12), 3459 (2020)
14. Wu, D., Arkhipov, D.I., Asmare, E., Qin, Z., McCann, J.A.: Ubiflow: mobility management in urban-scale software defined iot. In: Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, pp. 208–216 (2015)
15. Qin, Z., Denker, G., Giannelli, C., Bellavista, P., Venkatasubramanian, N.: A software defined networking architecture for the internet-of-things. In: Proceedings of the Network Operations and Management Symposium (NOMS), 2014 IEEE (2015)
16. Martinez-Julia, P., Skarmeta, A.F., et al.: Empowering the internet of things with software defined networking. In: Proceedings of the FP7 European research project on the future Internet of Things (2014)
17. Jararweh, Y., Al-Ayyoub, M., Darabseh, A., Benkhelifa, E., Vouk, M., Andy, R.: Sdiot: a software defined based internet of things framework. J. Ambient Intell. Hum. Comput. **6**, 453–461 (2015)
18. Huang, H., Zhu, J., Zhang, L.: An sdn_based management framework for iot devices. In: Proceedings of the 25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). IET, pp. 175–179 (2013)
19. Tran, H.A., Tran, D., Nguyen, L.G., Ha, Q.T., Tong, V., Mellouk, A.: Shiot: a novel sdn-based framework for the heterogeneous internet of things. Informatica **42**(3), 507 (2018)
20. Ojo, M., Adami, D., Giordano, S.: A sdn-iot architecture with nfv implementation. In: Proceedings of the 2016 IEEE Globecom Workshops (GC Wkshps). IEEE, pp. 1–6 (2016)
21. Tomovic, S., Yoshigoe, K., Maljevic, I., Radusinovic, I.: Software-defined fog network architecture for iot. Wireless Pers. Commun. **92**(1), 181–196 (2017)
22. What is docker ? https://www.docker.com/what-docker
23. Morabito, R., Kjällman, J., Komu, M.: Hypervisors vs. lightweight virtualization: a performance comparison. In: Proceedings of the 2015 IEEE International Conference on Cloud Engineering. IEEE, pp. 386–393 (2015)
24. Morabito, R.: A performance evaluation of container technologies on internet of things devices. In: Proceedings of the IEEE Infocom 2016. IEEE (2016)
25. Ruchika, V.: Evaluation of docker for iot application. Int. J. Recent Innov. Trends Comput. Commun. **4**(6), 624 (2016)
26. Morabito, R.: Virtualization on internet of things edge devices with container technologies: a performance evaluation. IEEE Access **5**, 8835–8850 (2017)
27. Ja'afreh, M.A., Adhami, H., Alchalabi, A.E., Hoda, M., El Saddik, A.: Toward integrating software defined networks with the internet of things: a review. Clust. Comput. **2021**, 1–18 (2021)
28. Chattopadhyay, S., Chatterjee, S., Nandi, S., Chakraborty, S.: Aloe: An elastic auto-scaled and self-stabilized orchestration framework for iot applications. In: Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, pp. 802–810 (2019)
29. Yang, R., Zhang, J.: imulab: Internet of things simulation platform based on openstack and container technology. In: Proceedings of the 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS). IEEE, pp. 927–932 (2021)
30. Okwuibe, J., Haavisto, J., Harjula, E., Ahmad, I., Ylianttila, M.: Sdn enhanced resource orchestration of containerized edge applications for industrial iot. IEEE Access **8**, 229117–229131 (2020)
31. Okwuibe, J., Haavisto, J., Kovacevic, I., Harjula, E., Ahmad, I., Islam, J., Ylianttila, M.: Sdn-enabled resource orchestration for industrial iot in collaborative edge-cloud networks. IEEE Access **9**, 115839–115854 (2021)
32. Muñoz, R., Vilalta, R., Yoshikane, N., Casellas, R., Martínez, R., Tsuritani, T., Morita, I.: Integration of iot, transport sdn, and edge/cloud computing for dynamic distribution of iot analytics and efficient use of network resources. J. Lightwave Technol. **36**(7), 1420–1428 (2018)
33. Florita, N.J.B., Sesdoyro, M.L.R., Valdez, J.T.C., Guinto, R.F., Tan, W.M.: Iot resiliency through edge-located container-based virtualization and sdn. In: Proceedings of the 2021 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob). IEEE, pp. 39–44 (2021)

34. Fawwaz, D.Z., Chung, S.-H., Ahn, C.-W., Kim, W.-S.: Optimal distributed mqtt broker and services placement for sdn-edge based smart city architecture. Sensors **22**(9), 3431 (2022)
35. Le, D.T., Großmann, M., Krieger, U.R.: Cloudless resource monitoring in a fog computing system enabled by an sdn/nfv infrastructure (2022)
36. Bolatti, D.A., Todt, C., Scappini, R., Gramajo, S.: Network traffic monitor for ids in iot. In: Proceedings of the Conference on Cloud Computing, Big Data & Emerging Topics. Springer, pp. 43–57 (2022)
37. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., et al.: Onos: towards an open, distributed sdn os. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, pp. 1–6 (2014)
38. Fontes, R.R., Afzal, S., Brito, S.H., Santos, M.A., Rothenberg, C.E.: Mininet-wifi: Emulating software-defined wireless networks. In: Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM). IEEE, pp. 384–389 (2015)
39. Fontes, R.d.R., Rothenberg, C.E.: Mininet-wifi: a platform for hybrid physical-virtual software-defined wireless networking research. In: Proceedings of the 2016 ACM SIGCOMM Conference, pp. 607–608 (2016)
40. Introduction to Mininet. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
41. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, pp. 1–6 (2010)
42. Olaniyan, R.: Synchronization schemes for internet of things and edge intelligence applications. PhD thesis, McGill University (Canada) (2021)
43. Olaniyan, R., Maheswaran, M.: Multipoint synchronization for fog-controlled internet of things. IEEE Internet Things J. **6**(6), 9656–9667 (2019)
44. Peuster, M.: Containernet. https://containernet.github.io/
45. Balasubramanian, V., Aloqaily, M., Reisslein, M.: An sdn architecture for time sensitive industrial iot. Comput. Netw. **186**, 107739 (2021)
46. Bedhief, I., Foschini, L., Bellavista, P., Kassar, M., Aguili, T.: Toward self-adaptive software defined fog networking architecture for iiot and industry 4.0. In: Proceedings of the 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). IEEE, pp. 1–5 (2019)
47. Moufakir, T., Zhani, M.F., Gherbi, A., Bouachir, O.: Collaborative multi-domain routing in sdn environments. J. Netw. Syst. Manag. **30**(1), 1–23 (2022)

Intidhar Bedhief has a Computer Science engineering degree from the Faculty of Sciences of Tunis, University of Tunis El Manar, Tunisia in 2012. She received her master degree in Communication Systems in 2016 from the National Engineering School of Tunis (ENIT), University of Tunis El Manar, Tunisia, where she is currently pursuing her Ph.D. in Communication Systems Laboratory (LR-Sys'Com). Her research interests are the Internet of Things (IoT), Software Defined Networking (SDN), Network Virtualization, and Network Orchestration.

Meriem Kassar received the B.E degree in Computer Networks and Telecommunications from the National Institute of Applied Sciences and Technology, INSAT, Tunisia, in 2004, the M.S and Ph.D degrees in Networks from University of Paris 6, UPMC, in 2005 and 2009 respectively. She is currently Assistant Professor at the National Engineering School of Tunis (ENIT), the University of Tunis El Manar, Tunisia. She is a senior researcher at Communication Systems Laboratory (LR-Sys'Com). Her research interests include Computer Networks, Heterogeneous Wireless Networks, Mobility Management, Software-Defined Networking (SDN), Internet of Things (IoT) and D2D in 5G networks.

Taoufik Aguili received a Ph.D. degree in telecommunications from INSA, France, in 1990. He is currently a full Professor at the National Engineering School of Tunis (ENIT), the University of Tunis El Manar, Tunisia. He is also the head director of Communications Systems Laboratory (SysCom) and the coordinator of the master degree in communications and information technology at the National Engineering School of Tunis (ENIT), Tunisia. His research interests include modeling of microwave systems and nano-devices, numerical methods in electromagnetics, electromagnetic wave phenomena in layered media, integrated transmission lines, waveguides and antennas, and leaky-wave phenomena. He has published over 250 papers.

Authors and Affiliations

Intidhar Bedhief[1] · Meriem Kassar[1] · Taoufik Aguili[1]

Meriem Kassar
Meriem.Kassar@enit.utm.tn

Taoufik Aguili
Taoufik.Aguili@gmail.com

[1] Communication Systems Laboratory LRSys'Com, National Engineering School of Tunis (ENIT), University Tunis El Manar, CampusUniversitaire Farhat Hached El Manar, Le Belvédère, BP 37, 1002 Tunis, Tunisia