IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

COPA: Experimenter-level Container Orchestration for Networking Testbeds

HENRIQUE C. C. DE RESENDE[1], MATIAS A. K. SCHIMUNECK[3], CRISTIANO B. BOTH[2], JULIANO A. WICKBOLDT[3], JOHANN M. MARQUEZ-BARJA[1], (SENIOR MEMBER, IEEE)
[1]University of Antwerp - imec, Sint-Pietersvliet 7, 2000 Antwerp Belgium (e-mail: {henrique.carvalhoderesende, johann.marquez-barja}@uantwerpen.be)
[2]University of Vale do Rio dos Sinos (UNISINOS) Av. Unisinos, 950 - Cristo Rei, São Leopoldo - RS, Brazil (e-mail: cbboth@unisinos.br)
[3]Federal University of Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500 - Agronomia, Porto Alegre - RS, Brasil (e-mail: {makschimuneck, jwickboldt}@inf.ufrgs.br)

ABSTRACT
As Cloud Computing (CC) branched areas such as Multi-access Edge Computing (MEC) and Fog computing are still on growing research interest. The creation of new tools to improve quality and speed the experimentation in such areas is a general interest. In this article, we propose COPA, an experimenter-level container orchestration tool for networking testbeds. This tool provides a friendly interface for the experimenter test container orchestration algorithms which can start, stop, copy, and even migrate a container from one host to another. COPA also includes network/resources monitoring to feed the experimenter's orchestration algorithm so that it can make decisions based on real-time environment information. Furthermore, the experimenter can automatize the experiment scenario setup and deployment by pre-configuring in COPA. This tool helps the experimenter in testing different scenarios and quickly changing experiment parameters. Considering these features, COPA aims to provide an experimentation architecture to deploy and test container orchestration algorithms. Furthermore, we provide a case study explaining how COPA can be a key tool in the MEC and Network Function Virtualization (NFV) experimentation environments. This tool was already deployed in Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory (FUTEBOL) testbeds as part of the control framework and was well validated by the project reviewers and partners.

INDEX TERMS testbed, multi-access edge computing, future networks, container, orchestration.

I. INTRODUCTION
Multi-access Edge Computing (MEC) is predicted to be one of the core paradigms that will support 5G networks. MEC will enable services to provide Ultra-Reliable Low-Latency Communication (uRLLC) by bringing the services or part of them closer to the network edge [1]. Taking the service to the network edge decreases the physical distance to the end-user increasing network bandwidth and decreasing network latency. Moreover, to provide service cover along the network edge, MEC datacenters must be smaller and cheaper than the usual Cloud Computing (CC) datacenter, enabling the construction of several instances. Therefore, these instances with distributed resources sometimes will lack to attend all the demands of service providers. Additionally, the services will need to be orchestrated depending on their requirements among CC datacenters far from the network edge.

MEC's real advancements will bring to network communications rely on the orchestration of either services or Virtual Network Functions (VNFs) to provide uRLLC [2]. The orchestration algorithms can prioritize different characteristics for each type of service. For processing-hungry applications, *e.g.*, image processing, the orchestration algorithm must allocate the most processing powerful resources available. For high network bandwidth usage applications such as video streaming, the orchestration must push the service to the edge of the network to relieve data traffic from the network core. For that reason, the European Telecommunications Standards Institute (ETSI) MEC standard describes a component named "*Multi-access edge orchestrator*", which will receive information from the infrastructure and decide the better way to distribute the services [3]. However, there are many types of applications and several service requirements to be considered when orchestrating. Therefore, it is expected a plethora of research in MEC orchestration.

Based on this rising community interest on MEC, researchers and manufacturers could avail from tools to enhance and help the research on this field. For example, Wang *et al.* and Leonini *et al.* proposed Weevil [4] and SPLAY [5]. These tools offer a distributed-application development environment and enable quick prototyping, deployment, and workload generation in testbeds. For MEC research, a platform to test various orchestration algorithms for service or VNF placement will support the deployment and analysis of the solutions. Orchestration algorithms can be from several types such as linear programming [6], deep learning [7], etc. The output these algorithms decide which actions the orchestrator will take, *e.g.*, migrate, replicate, or even stop a service. When creating an orchestration algorithm, a researcher designs a theory to support its algorithm idea. After, the researcher also needs to evaluate it in simulation [8] or a real-world environment [9] to test the solution's performance. Experiment environments can be used to validate researcher's theories supporting further research advancements in MEC and other areas such as Fog computing [10]–[12]. Testbeds are essential for the future of computer network research because these environments provide ready-to-use access to a variety of physical and virtual resources enabling to test realistic network scenarios [13]. However, because of the complexity of some testbeds, users can find difficulties in using it. Therefore, an easy-to-use, MEC orchestration tool focused on experimentation could increase research possibilities and simplify experiments deployment.

In this article, we present COPA: an experimenter-level container orchestration tool for networking testbeds. COPA was designed to attend the needs of the experimenters to test new orchestration algorithms, *e.g.*, MEC environments. In this way, an experimenter-level testbed layer for container orchestration hands-on experimentation was projected. COPA focuses on providing container orchestration, although MEC virtualization technologies do not include only container but also Virtual Machines (VMs). Containers are the most efficient way of virtualization, having a better performance, such as I/O access and storage virtualization [14], [15]. Furthermore, containers are lighter to migrate and faster to deploy, facilitating orchestration. COPA's main contributions are (*i*) the easy-to-use and simplified container orchestration environment, (*ii*) built-in orchestration algorithms platform enabling the deployment of heterogeneous types of algorithms, (*iii*) easy monitoring data gathering for experiment evaluation, and (*iv*) an architecture unifying testbed experimentation to container orchestration.

COPA differs from other container orchestration solutions such as Open Source MANO [16] and Open Network Automation Platform (ONAP) [17] by bringing a specialized built-in experiment environment with resources and network monitoring of the active hosts. Moreover, COPA enables the experimenter to deploy the network topology and interfacing with the container technologies to reallocate the resources dynamically. COPA communicates to the layer below-named Virtual Infrastructure Management (VIM), where all the container actions can be deployed. As shown in Figure 1, tools such as Kubernetes [18], LXD [19], and Swarm [20] are part of the VIM layer and can provide container management capabilities to the orchestration layer. COPA also focuses on providing a flexible platform for container orchestration, enabling users to deploy different solutions, quickly. Moreover, COPA can use pre-configured experiments in the testbed utilizing different containers management technologies. COPA also offers graphical monitoring of computer resources, and network quality through a friendly Web interface. The experimental environment monitoring feeds the container orchestration algorithms with data and provides real-time environment status emulating a real MEC scenario. Furthermore, COPA stores monitored data for experiment result analysis, which can be exported.

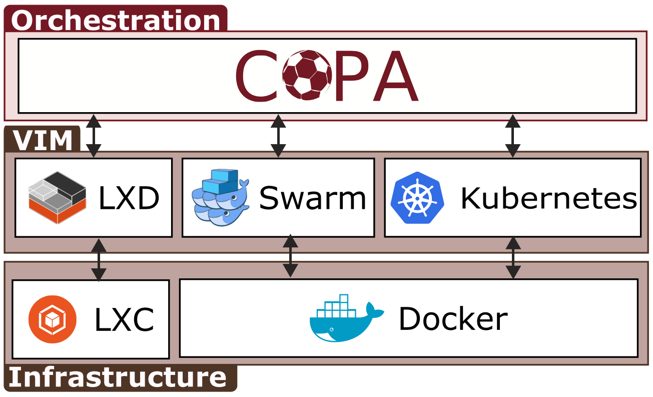


Figure 1: COPA in testbed orchestration layer

The remaining of this article is organized as follows. In Section II, we present the background regarding experimental research and related work focusing on experimenter-level tools and management technologies for containers. The design and components of COPA architecture are explained in Section III. In Section IV, we present a case study where COPA can use its capabilities to provide an easy-to-use experiment environment for MEC orchestration. We evaluate the COPA performance in Section V. Finally, we conclude and present future work in Section VI.

II. BACKGROUND AND RELATED WORK

In this section, we show the state-of-the-art about experimental environments considering virtualization and orchestration as tools used to help the experimenters. We first give a brief introduction to testbeds and virtualization technologies. Afterward, we present research about different types of experimenter-level tools, which help experimenters achieve experimental results in computer networks. Furthermore, we list the most recent container virtualization technologies. Finally, we demonstrate how these two topics could be unified in one solution so that experimenters can benefit from container management solutions in the experimenter-level orchestration layer.

Application	Monitoring	Resource Discovery	GUI	Workflow
Weevil [4]	✓			✓
SPLAY [5]	✓			✓
NEPI [21]	✓			✓
Plush [22]	✓	✓	✓	✓
jFed [23]	✓	✓	✓	
INSTOOLs [24]	✓	✓	✓	✓

Table 1: Experimenter-level Tools

A. TESTBED & VIRTUALIZATION

Experimental research is the bases of new high demand technologies. Researchers utilize experimental study to improve and prove their premises or solutions within state-of-the-art technologies. However, many times, researchers do not have the required equipment or software to enable their research. Some institutions created experimental environments to support investigations, which can be remotely accessed and are called testbeds. These environments are essential for science, making experimentation accessible to the majority of the research community. They help computer science researchers in providing the necessary infrastructure to evaluate its ideas and theories. Usually, these testbeds are built driven by the demands of new emerging technologies and the market. Therefore, different computer experimental environments are mostly specialized in distinct areas, such as programmable networks [25], optical/wireless infrastructure [26], or remote computing architectures [27], [28]. After provisioning the testbed infrastructure, the testbed's owners may provide remote access to its resources. This remote access can vary between the direct access to the experimental resource or through an abstraction layer called Experimenter-level Management. This layer helps the experimenter to understand the topology of its experiment and quickly deploy the necessary resources.

The use of testbeds for remote processing experimentation, as Cloud-oriented applications, is of broad and current interest. CC is a paradigm that enables services over the Internet. Likewise, it may refer to the hardware and systems in the datacenters that provide Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and even a new type of service called Internet of Things as a Service (IoTaaS) [29]. The general purpose of these services is to offer external capabilities of enhance the processing, storage, and compatibility to remote devices and services. In the case of IoTaaS, a set of Internet of Things (IoT) devices operate interconnected with a distributed cloud infrastructure, platform, or software [29]. These resources as a service are currently one of the key concepts in the literature that is pushing forward the virtualization technologies.

Virtualization is one crucial technology that has been widely explored in CC data centers to enable hardware sharing with data and process isolation. Currently, the most popular technology utilized for virtualization is VM [29]. However, recently an alternative for VM, the containers base virtualization, is gaining momentum. The container allows a lightweight in the deployment of services and applications, compared to traditional VMs, due to the sharing of the host kernel with user-space isolation. Containers are the base for the deployment of emerging 5G technologies, as proposed in the Cloud-Radio Access Network (C-RAN) [30], and to deploy Network Function Virtualization (NFV) [31]. However, several container-supported architectures have been developed in recent years. Each architecture is composed of different standards and functionality, which makes one way more challenging to deploy and to evaluate such services in experimental testbeds. To identify successful applicants for experimentation and to better understand the topic, we look at the leading initiatives in the area concerning control tools for testbed experimentation and manipulating applications for containers.

B. EXPERIMENTER-LEVEL MANAGEMENT

Experimenter-level management is an abstraction layer that provides generic and specialized solutions to facilitate the setup, execution, and monitoring of experimentation. Furthermore, it facilitates the development of innovative applications up to the monitoring and automation of testing, decreasing the error rate of configuring, and deploying an experiment. Besides such importance for experimental research, the existing experimenter-level tools do not contemplate the necessary features for the current study areas. To provide a better understanding of this subject of study, we researched some experimenter-level management tools from the last ten years. These researches are summarized in Table 1.

Weevil is a model-driven framework for experimentation on distributed-systems [4]. It provides a tool to describe an application, simulates its workload, and enables repeatability of the testing for the experimenter. However, it is focused on the application-level and does not provide any Graphical User Interface (GUI) or monitoring experimental environment infrastructure and resources. SPLAY is an integrated system that facilitates the design, deployment, and testing of large-scale distributed applications [5]. This tool monitors the whole system, orchestrates the deployment, and provides an environment for easy distributed-system prototyping. Furthermore, it gives a churn manager, which can reproduce the real or synthetic behavior of distributed systems from a file. However, SPLAY monitoring only provides a logging feature for the distributed application, and SPLAY also does not contain any graphical support during the experimentation.

NEPI is a library for emulation and simulation envi-

Application	Service Discovery	API	Autoscaler	Load Balancer	Scheduler	Live Migration
Docker [32]		✓				
Kubernetes [18]	✓	✓	✓	✓	✓	
Swarm [20]	✓	✓		✓		
LXD [19]		✓		✓		✓
rkt [33]	✓	✓	✓	✓	✓	

Table 2: Container Management Solutions

ronment integration that enables experimentation workflow across various testbeds and network simulation tools such as NS3 [21]. It provides tools for network and application-level description, full experiment management, and trace result collecting. Furthermore, NEPI offers an API for a third-party GUI or script. Although it cannot provides metrics during experimentation statistics, neither provides a resource discovery for automatically finding testbed resources. Different from the previous studies, Plush is a configurable application management infrastructure that allows developers to define flows of control needed by their computations [22]. This tool introduces essential characteristics to an experimenter-level environment such as distributed application fault tolerance and liveness improvements utilizing relaxed synchronization primitives. Plush provides a command-line, a GUI, and also provides resource discovery for dynamic application deployment. However, as other experimenter-level tools, Plush focuses on monitoring, deploying, and managing distributed applications in experimental environments, and does not provide an experimental environment resource monitoring.

jFed is an experimentation tool which provides a friendly GUI for testbed management, testing, and experiment automation [23]. This tool provides a list of federated testbeds in its system, enabling the experimenter to reserve, deploy, and connect to the testbed resources. For advanced users, jFed provides components for those who want to perform extensive fully-automated tests of the testbed APIs. One of the significant differences of jFed is it does not offer features from the control framework layer as other experimenter-level tools cited before. All the functions presented must be implemented in the testbed's control framework, provided to external usage by a standard API, and federated to jFed.

Finally, INSTOOLs, an Instrumentation System for Federated and Virtualized Network Testbeds [24], focus on instrumentation and measurement infrastructure for the experiment monitoring. Tsai et al. [31] classify network and resource monitoring as part of the control layer of the testbed. However, INSTOOLs implements a measurement plane instantiating Measurement Controllers and Measurement Points inside the testbed infrastructure to provide environmental information to the experimenter. INSTOOLs presents the experiment data utilizing tables and charts from a Web Interface with the measurement plane. Besides giving a meaningful network and resource monitoring architecture, INSTOOLs lacks on providing experiment replicability and workflow management.

The tools presented in Table 1 answer specific demands on experiment-level management, and some of these tools can even complement each other. However, there are innovative technologies currently being study, such as MEC and NFV, which require the orchestration of the experimental virtualized resources, including features such as migration, load balance, and autoscaling. Therefore, COPA aims to supply the missing features by integrating containers management solutions into the experimental environment. To provide a broad understanding of the existing solutions for container management, we give research about the subject in Section II-C.

C. CONTAINER MANAGEMENT SOLUTIONS

Container management solutions provide a lightweight virtualization environment to deploy virtualized resources. Furthermore, containerization can be attached to experimenter-level management, enabling experimenters to implement their off-the-shelf applications. The benefits of using containers in experiment environments are diverse. In this case, we detail some of these virtualization technologies and relate them to experiment environments. The containers technologies are summarized and compared in Table 2.

Containerization is not a new technology anymore, *e.g.*, Docker popularized it in its founding in 2013 [32]. Since then, Docker took the high part of the market share, turning into the most utilized containerization technology [34]. Single application containerization has been the focus of Docker, providing encapsulate microservices that can communicate with each other through well-defined communication channels. Docker is a simple virtualization technology yet powerful, providing an external API for remote container management besides the containerization itself. Another innovative technology is Kubernetes, *i.e.*, a cluster manager for Docker containers [18]. This cluster manager is a technology built on top of Docker technology, introducing new functionalities such as container auto-scaling, load balancing, and scheduling, enabling the utilization of containerization inside CC data centers. Kubernetes also allowed the service discovery of other new Kubernetes clients in the network. Moreover, this solution provides the configuration of multiple resources from a single point, and it is one of the most promising software for container cluster management.

A more straightforward cluster manager for Docker container than Kubernetes is Docker Swarm [20]. Different from Kubernetes, which was developed by Google, Swarm was developed by Docker Inc. itself. Therefore, the integration of this cluster manager with the lower containerization layers is smoother than the former. Swarm has many similarities with Kubernetes, such as load balance, service discovery, and scheduling modules. However, it is more limited, having lesser options for configuration than Google's solution. Moreover, Swarm does not comprise the auto-scaling module, responsible for providing high stability to the containerized application. Besides its drawbacks, Swarm is an excellent option for simple cluster management and could fit perfectly in an experimental environment.

LXD is an abstraction layer for Linux Containers Libraries and Tools (LXC) [19]. LXC is a straightforward container technology, not providing an API for external management. LXD was implemented on top of LXC to improve Linux containers' experience by enabling remote access through an external API. Different from Docker, LXD treats containers not as microservices, but as a multi-application environment such as VMs. To improve its usage in datacenters, LXD has integration support with OpenStack, *i.e.*, a cloud infrastructure manager [35]. This integration enables OpenStack to deploy containers instead of physical servers or VMs. Furthermore, the differential feature for LXD is its live-migration tool, which empowers the experimentation of several types of containerized services.

Core OS also developed a container technology called rkt, *i.e.*, an application container engine such as Docker. In this case, each container only runs one application, not an entire operating system [33]. The significant advancement of rkt is security. While Docker always has a root privileged daemon which enabled all its features, it can run its containers as unprivileged ones avoiding security breaches. Therefore, an experimental environment that is very security concerned may use rkt as its container virtualization technology.

Based on the crescent utilization of containers, the development of frameworks and layers on top of containers is becoming usual. These frameworks use the containerization characteristics to enable business and research of new kinds of remote computing models such as Function as a Service (FaaS) [36]. One of the tools that deserve mentioning is the OpenFaaS framework [37]. This framework is based on Docker and uses Kubernetes or Swarm as cluster managers to make available its stateless functions over the Internet. Apache OpenWhisk is a framework similar to OpenFaaS [38]. It enables FaaS to utilize container technologies to push the state-of-the-art of remote computing business models. As shown, there are currently new paradigms and technologies created with the emerging of containers such as FaaS and OpenWhisk. Another example of how containers are a trend in virtualization research is the EdgeNET project, which was created aiming to create a Kubernetes-based software-only infrastructure for CC and MEC experimentation [39]. The project provides access to several Kubernetes nodes worldwide and enables the creation and integration of a local Kubernetes node to the EdgeNet cluster. The project can provide at least one node (the experimenter node) close to the network edge, enabling the experimenter to add its node in the project cluster. This scenario characterizes a MEC experimentation because the experimenter's node will be closer to the devices connected in your network. In this way, the experimenter can test remote computing capabilities with lower network latency and higher bandwidth than the other EdgeNet cluster's nodes.

Containerization brings a whole new world not only for the industry but also for academia. However, most of the experiment-level management tools do not have support for container virtualization. Container virtualization technologies seem to take a straight road to service high-availability. However, these technologies lack features that could help with experimentation, such as container live-migration, workflow management, and customizable container orchestration. COPA integrates the experiment-level management and containerization, extending the ability of orchestration from the experimental environment's infrastructure. This integration enables experimenters to run their containers in several configuration patterns and research the application behavior in different conditions through a user-friendly graphical interface. In the next section, we detail how we enhanced the experimentation with containers.

III. EXPERIMENTER-LEVEL CONTAINER ORCHESTRATION FOR TESTBEDS

In this section, we present the COPA's architecture for experimenter-level container orchestration for testbeds. We explain the theoretical references for the layer organization of the architecture and introduce it to the main COPA's components. In the sequence, we detail the fundamental feature of COPA, Orchestration System, which enables the containers orchestration experiments.

A. LAYER ORGANIZATION

One way to organize a system architecture is through layers to group up components that perform a similar role. However, depending on the objective, we can divide the same components differently. In this section, we present a layer organization based on experimental environments, on MEC, and the equivalent layers in COPA architecture. In Figure 2, we map these equivalencies.

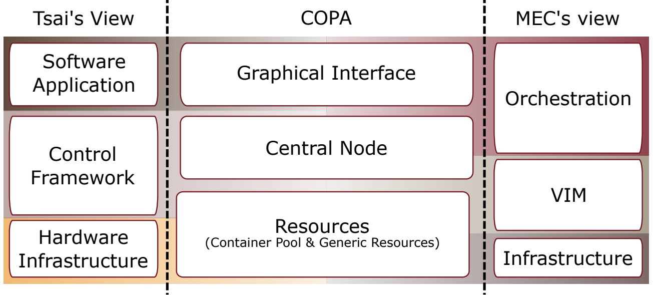


Figure 2: COPA layers classification

Tsai et al. [31] wrote an excellent survey about the control framework for computer networks testbeds. Moreover, the authors classified the different types of testbeds, and they came up with a layer structure for experimental environments. Tsai argues that this environment should be composed of three essential layers. The first layer from top-down is *Software Application layer*, where are all the tools and graphical interfaces to help the experimenter to develop its research. The second layer is the *Control Framework layer*, which is responsible for enabling Resource Management, User Management, Monitoring, besides the communication with the *Software Application layer* and the layer below. The third and last layer is the *Hardware Infrastructure layer*, which includes all the experimental physical resources.

In MEC environments, the layer organization differs from experimental settings because it focuses on orchestrating services and VNFs hosted in its data centers [3]. MEC's first layer, from top-down, is the Orchestration layer. This layer is responsible for choosing where to place or migrate a service based on the current status of the resources and the service's requirements. VIM is the second layer accountable for managing the life cycle of the virtualization technologies, *e.g.*, VMs, and containers. It receives management orders from the Orchestration layer, such as deploying and migrating virtual resources. Furthermore, some VIMs technologies can take over the replication of service, in case of an overload, and respawning it, in case of an error. The last layer is reserved for the infrastructure, representing every physical resource available in a data center and can be managed by the VIM layer.

To provide MEC capabilities in an experimental environment, COPA mix both layer structures into three layers: (*i*) Graphical Interface, (*ii*) Central Node, and (*iii*) Resources, as shown in Figure 2. COPA's Graphical Interface is equivalent to both Tsai's *Software Application layer* by providing tools for experimentation to the user, such as monitoring charts and experiment visualization. Moreover, Graphical Interface includes part of the MEC's orchestration layer into it by providing orchestration configuration and the deployment of algorithms. As Graphical Interface only provides the front-end for the orchestration layer, the Central Node provides the back-end.

The Central Node layer is responsible for executing the orchestration algorithms and providing access to the testbed resources. It is worth to mention that we chose not to rely on any specific VNF description language on COPA. This decision was made to keep the virtualization of the network functions deployment at a container management level. Container technology is currently widespread and this approach can facilitate the understanding of the tool's usage. LXD, the container virtualization used in COPA prototype, relies on profiles[1] to configure resource usage restrictions and security settings for instance. However, for container deployment, LXD relies on previous configured images that it download from image repositories. Furthermore, the Central Node includes experiment support for orchestration features such as experiment automation, monitoring, and data storage. These experiment support features are what Tsai's Control Framework layer is responsible for as well. The Control Framework layer executes the experiment tools in the testbed-side. Moreover, unlike COPA's Central Node layer, the control framework layer also manages the testbed infrastructure and reserves the necessary resources to run the experiment.

[1]https://lxd.readthedocs.io/en/latest/profiles/

Resources are the last COPA layer. It represents all the hardware and virtual infrastructure available in the testbed and some monitoring and experiment automation agents. Therefore, this layer is equivalent to some part of Tsai's Control Framework and some of Hardware Infrastructure. From the MEC perspective, COPA's Resources layer has the same role as VIM and Infrastructure layer by being responsible for hosting virtualized resources. The Resources layer can be classified into two different types. The first type is Container Pool, *i.e.*, a resource capable of running containers, and the second type is Generic Resources that cannot run containers.

Considering the organization of these layers, we aim to provide an experimental MEC environment by providing orchestration configuration with experiment support tools to the experimenter. To better comprehend of the COPA operating, we describe the components within the COPA's layers in Subsection III-B.

B. COMPONENTS

In this subsection, we detail all the components which compose the three-layered COPA's architecture. The components are described from top-down following the order from Figure 3 (A). Orchestration Management, Workflow Management, Monitor, and Data Exportation components (1-4) composes the Graphical Interface layer enabling the user interaction with COPA through the browser. The Central Node's layer counts with Orchestrator, Workflow Controller, Monitor Controller, Data Processor, Software-Defined Networking (SDN) Controller, and Resource Discovery components (5-10). Finally, in Resources, there are Containerized Apps, Monitoring Agent, Workflow Deployer, and Virtual SDN Switch (11-14).

1) Orchestration Management

It is a graphical interface that provides all the functionalities to the experimenter to manage its orchestration algorithms. This component contains management options such as executing, stopping, and removing algorithms of the Orchestrator from the Central Node. Besides the standard management options, the Orchestration Management interface counts with the switch functionality. The switch is necessary because the experimenter can run multiple orchestration algorithms simultaneously. While using the switch, the module tells the Central Node to turn off the running algorithms and start the selected one. Furthermore, the experimenter can upload container images using Orchestration Management, which will send the information to the Orchestrator component, and will communicate to the VIM to deploy the container. The Orchestration Management

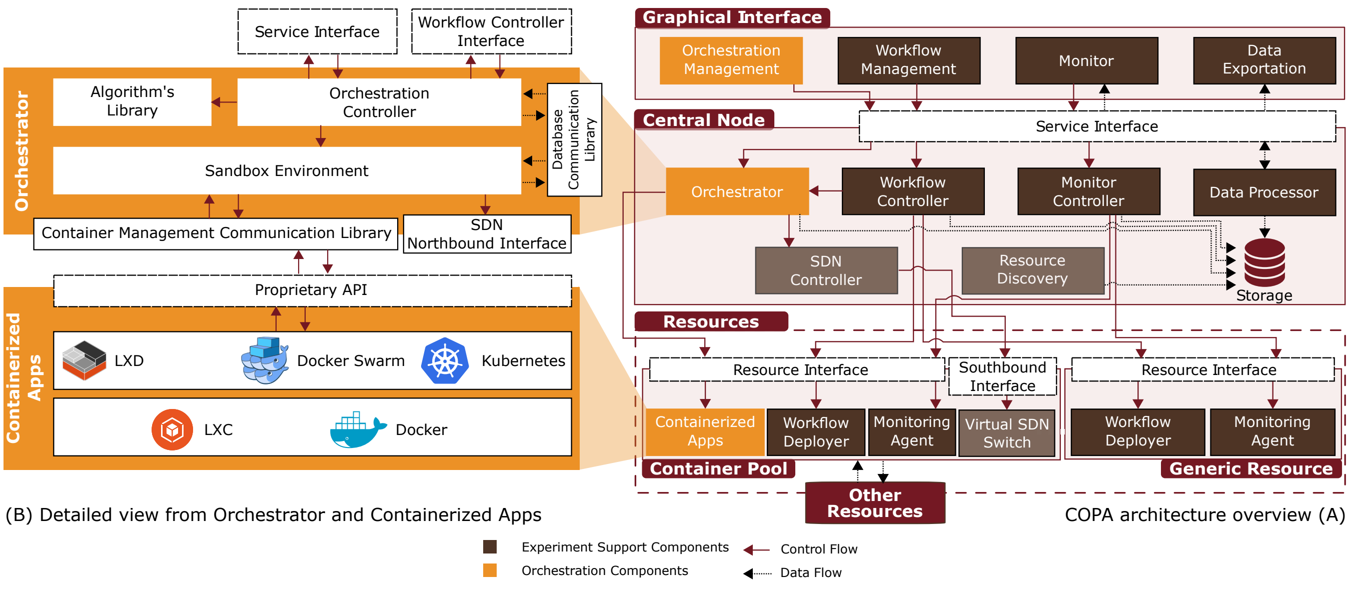


Figure 3: COPA architecture overview (A) and detailed view of the orchestration components (B)

aggregates value to COPA by providing simplified access to built-in orchestration functionalities. Therefore, it enables the adjusted deployment of heterogeneous orchestration solutions.

2) Workflow Management

This component is an interface that provides custom automaticity planning to the experimentation. The experimenter can pre-program a set of commands and time intervals. A scenario can be executed using this component, runs for a specific time, stops the current experimentation, reconfigures the new scenario automatically, and runs in another time interval. In this way, this component can accelerate the experimentation research by reducing the configuration time and providing reproducibility for the experiment. Reproducibility is an essential characteristic for a research analysis since, when evaluating a scenario, researchers need to run the same setup several times, trying to decrease error margins.

3) Monitor

This component provides several metrics charts with data collected by Monitoring Agents, enabling the user's experiment awareness. Following real-time monitoring, the experimenter can detect errors, anomalies, and undesired circumstances. This component is divided into two parts to provide the above characteristics: (*i*) the dashboard, where the experimenter follows the data gathered from Container Pools and Generic Resources such as CPU, Virtual Memory, Storage, etc.; (*ii*) detailed monitoring of Container Pools and Generic Resource, which presents data from containers and network link connectivity among all resources. The objective is to give the metrics available in COPA and its behavior during the experiment to the experimenter. Therefore, the experimenter can design and enhance the orchestration algorithm based on the network's response and the resources during the current orchestration.

4) Data Exportation

It provides an interface to the experimenter with the data stored in the experimental environment. When choosing data exportation methods, the experimenter can select data format, from which resources it needs the information, and which metrics it wants to export. Moreover, this component enables the user to download the experiment workflow and orchestration configuration, allowing the experiment's reproducibility. Furthermore, the exported configuration data can be used as a tool for hands-on lab classes to facilitate the investigation setup by the students.

5) Orchestrator

It allows the experimentation of custom orchestration algorithms by the experimenter. Orchestrator provides an environment to run one or multiples processes to support experimenter orchestration algorithms. In this way, COPA enables the simultaneous execution of intelligence to manage containers or migrate them to new Container Pools. Orchestrator, as can be seen in Figure 3 (B), is composed of five subcomponents: (*i*) Algorithms Library, (*ii*) Orchestration Controller, (*iii*) Sandbox Environment, (*iv*) Container Manager Communication Library, and (*v*) Database Communication Library. Algorithm Library stores default orchestration algorithms and custom algorithms uploaded by the experimenter. Orchestration Controller monitors the experiment configuration in the database and manages the life cycle of the orchestration algorithms by downloading the algorithms from Algorithms Library to its execution in Sandbox Environment. Sandbox Environment is a restricted runtime environment for orchestration algorithms deployment, which avoids malicious scripts to reach critical components of the testbed. Container Manager Communication Library and SDN Northbound interface enable communication from inside of Sandbox Environment to outside resources. The former communicates to the container management services located in the Container Pools. The latter allows interaction with the SDN controller, so that the algorithm can manage the data flow among the containers. Finally, Database Communication Library provides read-only access to the experimental data for decision-making of the custom orchestration algorithms. Furthermore, Orchestration Controller can also manage eventually by Workflow Controller to deploy pre-configured management actions to the orchestration algorithms. The Orchestrator brings a centralized component prepared for deployment of orchestration algorithms. Its usage is simplified through the Orchestration Management interface. The abstraction of heterogeneous interfaces for Container Management solutions, SDN Controllers, and access to data storage helps the deployment of a single script in infrastructures built on top of different solutions. This characteristic facilitates the standardization of the scenarios created by students in several networking testbeds worldwide.

6) Workflow Controller

This controller is the central component of the workflow functionality. It receives the description of the automation process from the Workflow Management and breaks it into small sorted actions, and builds a control unit class. This class maps all the tools and resources necessary to run the experimentation from its beginning till its end. In this mapping are described Workflow Deployers, the orchestration algorithms, the communication actions with Orchestrator, the time intervals, and the routine scripts uploaded by the user to be run. Moreover, Workflow Controller can change the active orchestration script, pause, or run simultaneous scripts. Furthermore, this component can upload scripts to Workflow Deployers and make then run a set of actions inside the experimental resources. These actions can also be succeeded by time intervals, which controls the time for which iteration and parameters change. In summary, the Workflow Controller parses the experiment execution file, maps it to the testbed environment, and dictates the actions that will happen during the experimentation. Therefore, the experimenter applies test automation, which decreases the human error factor when reproducing an experiment several times.

7) Monitor Controller

It is the centralized control of the monitoring functionality to manage multiple Monitoring Agents. Moreover, this component organizes network and resource monitoring, synchronizes the startup of Monitoring Agents, and avoids unnecessary CPU and network utilization at the beginning of the experiment. The connection created between Monitor Controller and Monitoring Agent its strictly for monitoring control, hence Monitor Controller is not responsible for storing the monitoring data collected by Agents. This component adds up to the value of COPA in general because it ensures the management of Monitoring Agents' life-cycle. The data collected by the Monitoring Agents are essential for the decision-making algorithms, which becomes fundamental that we have a component ensuring the execution of the monitoring task.

8) Data Processor

This component is responsible for processing the requests from Data Exportation and the Monitor components. It is common in any client-server architecture to have a component to receive and process the information requested from the front-end applications, such as the Graphical Interface layer. Moreover, when required, Data Processor queries the COPA's database and the data standard format to be shown in the Graphical Interface layer. Furthermore, this component can be used by third-party software to extract data from the experiment providing an API for Graphical Interfaces components. Data Processor is not an innovative component. However, it is a must-have to provide the necessary features for COPA to be up-to-date to the concepts of service-based architectures. Therefore, this component's inclusion is essential and helps in the communication of COPA with other testbed infrastructure software.

9) SDN Controller

This controller centralizes the SDN rules deployment by detaching the network control plane from the data plane. Algorithms deployed by the Orchestrator can have access to this component using the Northbound Interface, which every SDN controller holds for external access. The SDN controller receives the commands from the Northbound interface and deploys them to the respective SDN switches, customizing the network traffic. SDN is fundamental if the experimenter needs to deploy a sequence of VNFs, called Service Function Chaining (SFC). This sequential VNFs needs to have the network traffic steered through them, and the SDN controller can configure the necessary network rules to make it possible.

10) Resource Discovery

It provides automatic discovery of the experiment environment resources. Resource discovery facilitates the experiment setup since the experimenter does not need to register all the resources manually. This characteristic is beneficial in scenarios not controlled by the experimenter, and it does not have direct access to the resources. This component's main action is at the beginning of the experimentation, broadcasting a message to the network, and locating the experimentation resources that are listening to the network. As soon as Resource Discovery founds a resource, it registers in the storage and sends the control of the resource to the Controlling components such as Monitor Controller, Orchestrator, and Workflow Controller. Resource Discovery can also be executed through Graphical Interface, so the experimenter can add new resources after the experiment started. Although the apparent automation brought by the Resource Discovery, this component gets the right information of the Generic Resources and Container Pool for centralized management, such as Orchestrator and Workflow Management, which depends directly on the infrastructure information.

11) Containerized Apps

This component is an abstraction for the container management role. Moreover, it can be any containerization technology, only needing to have an external API for remote management. For some container management characteristics that COPA supports, such as container live-migration, the Containerized Apps may have it implemented so COPA can make it available to the experimenter. The Containerized Apps component is the main block in the testbed's virtualization and resource isolation and experimentation. Based on this component, the MEC application servers, and network function stands and will be managed and orchestrated by the COPA components.

12) Monitoring Agent

This agent composes, beside Monitor Controller and Monitor, the monitoring system of COPA. It is responsible for collecting end-to-end network metrics among the Resources and monitoring computational resources such as CPU, Memory, and Network Workload. This component collects data from connected devices, such as Packet Loss, Signal Strength, and Transmitted and Received bytes, considering any resources which have a wireless communication interface with an access point. The reliability of the Monitoring Agent data is essential for the research results and conclusions. Therefore, extra care should be taken in the tools used for the network and resource monitoring.

13) Workflow Deployer

It is a passive module that listens to the Workflow Controller component for commands to execute. It can be configured and installed in any resource, and besides running commands, it can download full scripts from the Controller and report execution time errors. The support of script download enables the execution of complex routines provided by the experimenters. Furthermore, the execution time report allows experiment environment awareness for the user and helps to debug the experiment routines. Therefore, it is a must-have component to the Workflow functionality works in a centralized way by supporting the actions deployed in Workflow Controller.

14) Virtual SDN Switch

This component enables the redirection and customizes treatment for the network traffic passing by it. Moreover, it communicates with the SDN controller through the Southbound interface to receive SDN Rules and, in some cases, to send monitoring data about the network traffic. In every experiment deployment, the experimenter performs a sequence of actions to run the scenarios, and we call the experiment flow. In our architecture, the experiment flow advances by three main components: Orchestration Management, Orchestrator, and Containerized Apps, which are highlighted in yellow in Figure 3 (A) and (B). The remaining experiment support components support these components, and, in Subsection III-C, we describe their functioning and the communication among them.

C. ORCHESTRATION SYSTEM

In COPA, the experiment flow begins at Graphical Interface after the resources are deployed and already found by the Central Node's Resource Discovery component. After the resources registered, the experimenter can visualize through Monitor the existing Container Pools and Generic Resources. The first action of the experimenter is to create or upload its containers to one or more Container Pools through Orchestration Management. Next, the user writes the algorithm in a standard programming language defined in the COPA implementation. Orchestration Management provides file upload to save one or multiple orchestration algorithms. The component, then, communicates with Orchestrator to send the orchestration algorithm and check the status.

A copy of the algorithm is stored at Algorithms Library when Orchestration Management communicates to Orchestrator sending this algorithm. The experimenter can send management actions, *e.g.*, start/stop the container, from Orchestration Management to Orchestrator to manage one or multiple algorithms. Moreover, COPA saves the current orchestration configuration in the storage. Orchestrator reads the change of configuration status and applies to the orchestration subcomponents by downloading the algorithm from the Algorithm's Library to Sandbox Environment and deploying it.

After being deployed, the orchestration algorithm starts reading the monitoring data from the database and making decisions over the containers. These decisions are made through the Container Manager Communication Library, which delivers the message to the container management technology. Therefore, the container management technology is responsible for doing the changes into Containerized Apps inside Container Pools. We provide a ready-to-use, MEC orchestration environment for experimentation. Orchestration System enables the research of trending areas that require secure virtualization and orchestration of services, concluding the integration of experimenter-level management and container management technologies.

IV. CASE STUDY

This section presents a case study describing how COPA helps the experimenters lead with container orchestration tests. For this case study, we consider a real MEC environment that consists of a small-sized datacenter providing services at the edge of the network and a robust and large-

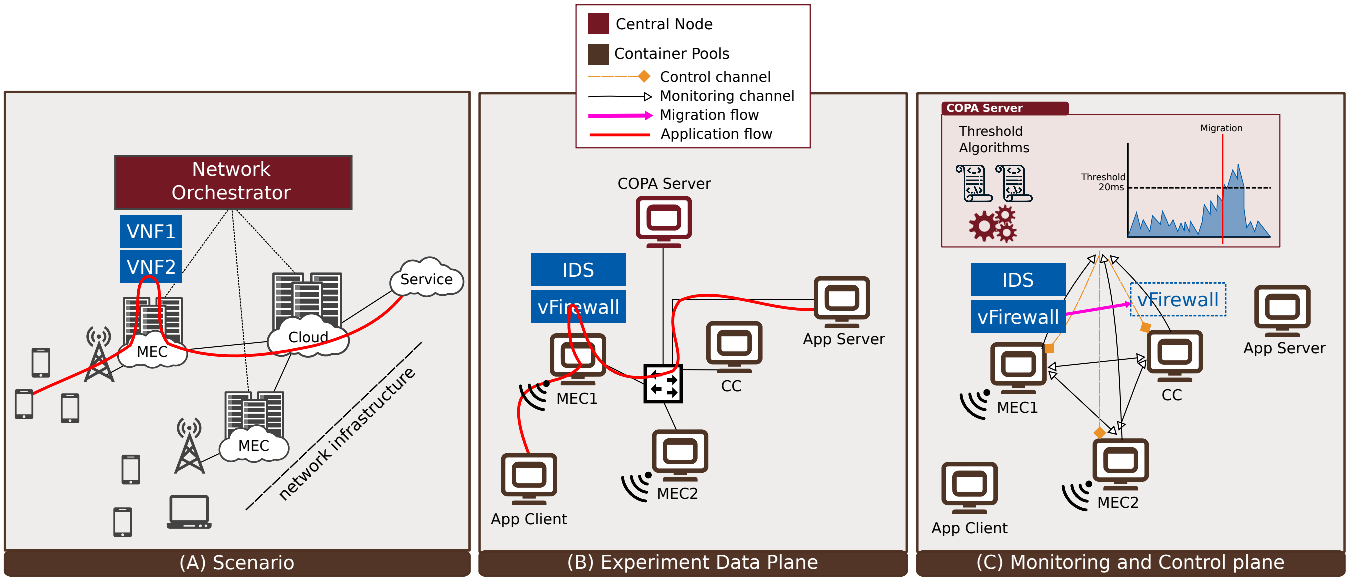


Figure 4: Case study representation

size datacenter specializing in energy efficiency and and large-scale services in CC. Figure 4 (a) shows the scenario with two MECs and a CC. In this case study scenario, two VNFs are deployed in the MEC datacenter at first, and the Network Orchestrator continuously monitors it. This orchestrator's objective is to avoid Quality of Service (QoS) degradation. One of the solutions to prevent this degradation is the orchestrator triggering a migration of one of the VNFs to another MEC or the CC datacenter based on the network's current status and available resources.

COPA supports this scenario emulation by deploying six VMs: one CC datacenter, two MECs datacenters, one for the Orchestrator, one to emulate a client, and one to emulate the server, as can be seen in Figure 4 (b) and (c). VMs for the CC datacenter and the two MECs are deployed as Container Pools with distinct capabilities to emulate the difference in processing power between the edge and core network facilities. The Orchestrator VM is deployed with the Central Node and Graphical Interface for the experiment's network orchestrator. Furthermore, two other generic VMs are deployed to emulate a client-server architecture, establishing a connection, and generating data traffic. Figure 4 (b) depicts the physical connectivity among the resources deployed in the testbed. In this illustration, we detail our case study's application data flow, a virtual Firewall (vFirewall) and an Intrusion Detection System (IDS) are used in MEC1 processing the traffic passing through it.

Figure 4 (c) highlights the existent control channels among the deployed resources to be orchestrated. Each Container Pool, MEC or CC, monitor its available resources such as storage, memory, and CPU usage. Furthermore, the datacenters are responsible for monitoring the network latency and jitter to the other datacenters in the network and the current download and upload network throughput. Considering the data from the monitoring collected, the Container Pool sends this data to the Central Node, where the decision migration is taken.

Over this scenario, the experimenter tests two different types of container orchestration algorithms. These algorithms orchestrate the two different types of VNF: vFirewall and IDS. In this work, we did not aim to provide an evaluation of algorithms effectiveness because there are already other studies that pursue this objective [40]. Therefore, for this purpose, we chose two simple threshold algorithms to describe the functioning of COPA.

The first algorithm, which we call Algorithm A, receives the CPU utilization percentage of the host. In case the CPU percentage goes up 90%, the migration is initiated to a host with the CPU availability under this percentage. The value of 90% was chosen, assuming that both VNFs which will be presented will consume at least 10% of CPU capacity. The second threshold algorithm, which we call Algorithm B, considers the network latency between the hosts and the cost to host the service. If the MEC latency, which is nearer to the users and any other host with lower hosting cost, reaches the latency below 20ms, the migration is triggered by the MEC nearer to the users. Algorithm B assumes which the addition of 20ms in the network latency for this specific service will not interfere in the users' Quality of Experience (QoE).

In this case study, the experimenter manages the scenario connecting to COPA via Graphical Interface to the Central Node. Through Graphical Interface, it is possible to upload the VNFs containers into one of Container Pools, and, then, the experimenter have the possibility to follow the monitoring of the Container Pools. Moreover, COPA presents the list of the containers running in each of the Container Pools, where containers management actions such as start, stop, and migrate are available. Following the VNFs setup, the experimenter uploads the orchestration script into COPA, (*e.g.*, deploying Algorithm A). This operation is performed through Orchestration Management, where the experimenter is also able to manage the available orchestration scripts. Furthermore, after the orchestration script uploaded and deployed, the experimenter checks the behavior of Container Pools using Monitor, and also if there is any container migrating occurring by analyzing the network data traffic. In Figure 5, we detail the sequence diagram of the communication among the COPA layers for (*i*) the creation of a container, (*ii*) the upload of only one orchestration algorithm, (*iii*) the deployment of this algorithm, (*iv*) the orchestration algorithm being fed by COPA with resource monitoring data, and, finally, (*v*) a migration being executed by the Algorithm A.

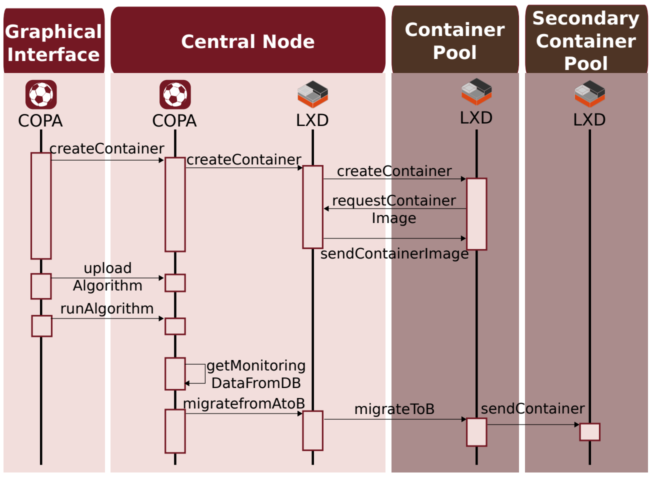


Figure 5: COPA layers sequence diagram for container creation, upload of an algorithm, execution of an algorithm, and migration of a container

There are two possibilities to add algorithms to COPA: Manual or automatically. When deploying manually, the experimenter access the graphical interface and send the algorithm script through COPA API. However, the actions of deployment of a VNF or algorithm can be configured automatically, using the Workflow functionality. The Workflow components enable the experimenter to setup step-by-step the scenario changes, even stopping complete the algorithms and containers, and initiating a completely different one. The automation of the experiment starts by uploading the containers and the algorithms to COPA as usual. However, the experimenter using the Workflow Management starts creating a list of commands, such as deploy container, deploy orchestration Algorithm A, and switch to orchestration Algorithm B. These commands are preceded by the relative time since the experimenter's start and will be executed as configured by the user.

In our case study automated experiment, we configure two VNFs and two orchestration algorithms. Algorithm A and both VNFs will be deployed in the second 0 and stop at the second 59. After that, in the second 60, we return containerized VNFs to its original Container Pools in case they had been migrated. When the migration is done, Algorithm B is started. In this way, we automatically evaluate the same containerized VNFs start position for two different orchestration algorithms.

Case studies such as this one can represent scenarios for the research and introduction to orchestration algorithms for educational purposes. COPA facilitates the setup of such experiments keeping the hardware requirements for the deployment of an orchestrator at the minimum possible. For example, this minimum hardware setup for the VNF orchestration study maximizes access for practical network management courses. Therefore, COPA fills a gap in a market where the orchestrators are built focusing on industry and production rather than education.

V. EVALUATION

Most of the tools used when evaluating a scenario might affect the experimental environment several manners, and with COPA, it is not different. It is essential to understand the behavior of these tools to prevent unexpected variations in the experimental environment. To better understand this section, we first present a high-level comparison between COPA and other orchestration tools. Preaching for transparency, we detail COPA resource consumption behavior to provide a full understanding of what the users can expect when utilizing this tool. Moreover, we describe our experimental environment setup and results.

A. HIGH-LEVEL TOOLS COMPARISON

COPA was developed to be an experimenter-level orchestrator. Given the academic focus, COPA differs in several aspects from other orchestrators available in the market, such as ONAP [17], Open Source MANO (OSM) [16], Tacker [41], and Open Baton [42]. The usage of these orchestrators varies in design and features, which attract different collaborators to the projects. We briefly introduce these four orchestrators, as can be seen in Table 3.

ONAP is an open-source software platform that provides lifecycle management support for new services proving orchestration of physical and virtual network functions. This platform is the merging of two big projects in the same area. One of these projects was lead by AT&T and the other by the Linux Foundation in collaboration with China Mobile, Huawei, and ZTE. Due to the integration of these projects, a restrictive characteristic of ONAP is the high hardware requirements.

OSM is a project led by the OSM Foundation and is designed to align with ETSI NFV information models. This alignment with the standards defined by ETSI implies that every component and interface detailed in the documents are implemented in the stack and should be compatible with every stack that complies with the ETSI standard. However, OSM does not focus on experimenter-level tools.

Tacker is a generic VNF manager and orchestrator maintained by Openstack. This orchestrator is designed to be compatible with the OpenStack virtualization technologies. Tacker is composed mainly of two components, which

Orchestrator	Hardware Requirements	Orchestration Capabilities	Virtualization Technologies	Experimenter-level tools	User Interface	Network Monitoring
COPA	*Light*	Monitoring, Live-migration	Container	✓	✓	Active and passive monitoring
ONAP [17]	**Heavy**	Monitoring, Live-migration, Lifecycle Management, inter-Domain	Virtual Machine & Container		Partial	Passive monitoring for specific technologies
OSM [16]	*Light*	Monitoring, Live-migration,Lifecycle Management	Virtual Machine		✓	Only passive monitoring
Tacker [41]	Medium	Monitoring, Live-migration, Lifecycle Management	Virtual Machine & Container			Only passive monitoring
OpenBaton [42]	*Light*	Monitoring, Live-migration, Lifecycle Management	Virtual Machine & Container		✓	Custom monitoring

Table 3: Orchestration tools comparison

exposes orchestration actions through an API and communicates with the infrastructure driver. This tool works perfectly for who is already proficient in OpenStack for managing cloud servers and upgrading it to use with VNFs.

OpenBaton is another alternative for ETSI compliant orchestrators. This orchestrator is developed by Fraunhofer FOKUS and TU Berlin and is one of the parts of the project 5G Berlin[2]. OpenBaton aims to provide NFV management for cloud servers to support research projects and lead the state-of-the-art.

All the orchestrators aforementioned are designed to expose API and manage the lifecycle of virtualized components such as VNFs. Therefore, these orchestrators connect to VIMs such as OpenStack, Docker, and Kubernetes, responsible for managing any virtualized infrastructure. However, to specialize the orchestration of virtual components, orchestrators such as COPA are fundamental to complement the management intelligence and support specialized configuration and actions to achieve different services/experiment objectives. Aiming to highlight the pros and cons for each orchestrator project and compare them with COPA, as can be seen in Table 3, we selected six different characteristics: hardware requirements, orchestration capabilities, supported virtualization techniques, the support for experimenter-level tools, the availability of a user interface, and network monitoring.

In the hardware requirements, we classified the orchestrators in three types: heavy, medium, and light. In the heavy hardware requirements type, ONAP is the only orchestrator composing this group. ONAP, in its minimum setup, requires 40GB of RAM, 1TB of storage space, and, at least, 25 vCPUs. In the medium type, Tacker is the only group member requiring 8GB of RAM and must be installed with OpenStack. Given the strong attachment with OpenStack as VIM, Tacker was classified in this group. In our most prominent group, the light hardware requirements, we have COPA, OSM, and OpenBaton. OSM requirements are 2 CPUs, 4GB RAM, 20GB disk. OpenBaton requirements are 2 CPUs, 2GB RAM, and 10GB of the disk, COPA minimal hardware requirements are 1 CPU, 1GB RAM, and 10GB of the disk space. These three orchestrators' hardware requirements do not have a massive difference and can be classified in the same type. COPA, OSM, and OpenBaton are the best candidates for simple deployments of orchestration tools while ONAP and Tacker are suitable for large and medium network infrastructures, respectively. Furthermore, a benchmark study for orchestrators was done by Yilma *et al.* [43]. In this study, they compared a setup with ONAP and other with OSM supporting the theory that the former is better designed for high computing power infrastructure. The latter is better for the orchestration of small size infrastructures such as MEC datacenters.

The orchestrators have small differences among capabilities. Monitoring of the virtualized components is an essential feature, and it is present in all of the orchestrators. However, Live-migration capability depends on each VIM in which the orchestrator is deployed. VIMs using VMs are mostly supports live-migration. However, when using containers, hardly the experimenter can use this feature with the orchestrators. COPA using LXD as a container virtualization technique differs from other orchestrators that use Docker and Kubernetes. Therefore, using LXD, COPA can support live-migration for containers. In this context, Inter-Domain is a feature of orchestrators that enable the communication synchronized among orchestrators. The only orchestrator prepared for this communication is ONAP, which is designed for large and industrial operations. The fourth and last orchestration capability is Lifecycle Management, which is an essential feature for orchestrators. In this case, only COPA lead to this challenge in a different methodology. However, COPA does not implement this feature, exposes all the necessary APIs for the implementation. COPA grants primitives for the experimenter to build its solution on top

[2]https://5g-berlin.de/

of the orchestrator. This characteristic enables COPA to be more flexible and straightforward.

The orchestrators support certain virtualization technologies: VM or containers. ONAP, OpenBaton, and Tacker support both technologies, making them great to heterogeneous environments where you have both technologies. However, OSM and COPA have specific focuses. OSM supports only VM, which is a well-consolidated technology, and COPA supports containers, which is a light-weight technology and again focuses on simplicity and light deployment.

The experimenter-level tool is an essential characteristic of orchestration tools for research objectives. The majority of the orchestrators have capabilities to monitor and collect data about their performance and execute actions based on their resource utilization. Nevertheless, COPA is designed to perform sequential experiment steps and activate orchestration scripts with objective performance evaluation. Therefore, it is composed of several graphical interface tools and architectural modules to best experiment orchestration strategies.

The user interface helps experimenters, such as students, visualize the orchestrator's resources and options to perform orchestration actions. Therefore, orchestrators such as COPA, OSM, and OpenBaton offers the user a graphical interface to organize and configure the virtual environment. ONAP is an enormous system and decided to have different graphical interfaces for various modules in its structure, and other modules can be configured only by the command line. As an extension of OpenStack, Tacker does not have a graphical interface, only making available to the user a command-line interface.

The network monitoring for orchestration is fundamental for orchestration intelligence when evaluating the impact of an orchestration decision upon an ongoing service. Therefore, complete monitoring of the network needs to be implemented, and additional network delays take into consideration the orchestration decision-making. Tacker and OSM have passive monitoring of the virtual resources, making available the information about packets sent and received and the network's throughput and for each VNF. ONAP, besides the passive monitoring of the resources, also relies on specific protocols such as Voice Quality, which can inform packet loss, packet delay variation, round-trip delay. Therefore, using packets for this particular technology, it can measure end-to-end network metrics. OpenBaton does not have a complete monitoring system for its resources but enables an external monitoring system such as Zabbix[3]. The customization of the network monitoring helps flexibilization of the system and opens new fronts to evaluate the virtual environment and network. COPA offers the user an active and passive network monitoring of the virtual resources' host. Therefore, it provides the necessary information for the custom orchestration scripts to make the best evaluation of the impact of the migration of the resources in the lifecycle management of the virtual environment and service quality.

In this section, we listed the main options for virtual network orchestrators. The possibilities present different characteristics and focus where ONAP is a robust network orchestrator and the primary choice for deployment in industry. OSM is the second option for most of the studies in network orchestration, and OpenBaton, which was embraced by a big network project. Because of different characteristics, these orchestrators could fill distinct niches, and COPA is not different. COPA has its differences to the other network orchestrators, fills a newly identified gap in research, and enables simple deployments and evaluation of orchestration scripts in universities and testbeds.

B. EXPERIMENTAL ENVIRONMENT SETUP

The evaluation scenario is composed of four VMs: one running Central Node with Graphical Interface, and the other three running Container Pools. The configuration of VMs is detailed in Table 4, which are connected in the same network. COPA's version utilized in this evaluation was developed for the control framework of the Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory (FUTEBOL)[4] project. In the implementation, we used the programming language Python 3.7 mixed with Django Framework [44]. Furthermore, we chose LXD [19] as the container management solution aiming for the live-migration feature. All details, *e.g.*, source-code and tutorials about COPA implementation, can be found at the FUTEBOL's GitLab repository [45].

VMs Configuration	
Description	1 Central Node and 3 Container Pools
OS Version	Ubuntu 18.04.2
Linux Kernel	4.15.0-46-generic
Architecture	x86_64
LXD	3.0.3
CRIU	3.6
Memory	1 GB
CPU Cores	1
CPU Frequency	1.80 GHz,

Table 4: Experiment scenario VMs configuration

This experimental environment setup emulates a MEC scenario with three data centers and a decoupled decision-making unity, Central Node. Based on this setup, we aim to evaluate the number of resources and network capacity COPA requires.

C. QUALITATIVE RESULTS

It is common sense that hard-to-use tools may compromise the new public's adherence to a particular subject, in the researchers' community is not different. Easy-to-use tools may attract interest to the area when new researchers or students look for an area to research. Therefore, simple and easy-to-use tools can be used as a catalyst to increase the

[3]https://www.zabbix.com/

[4]http://www.ict-futebol.org.br/

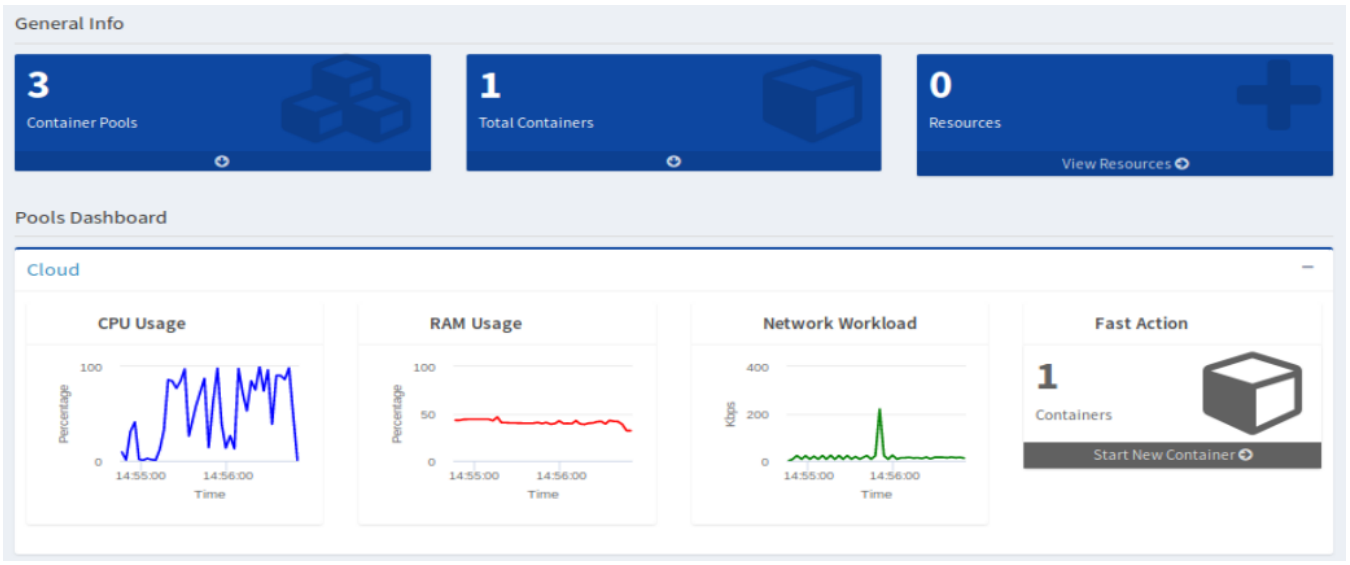
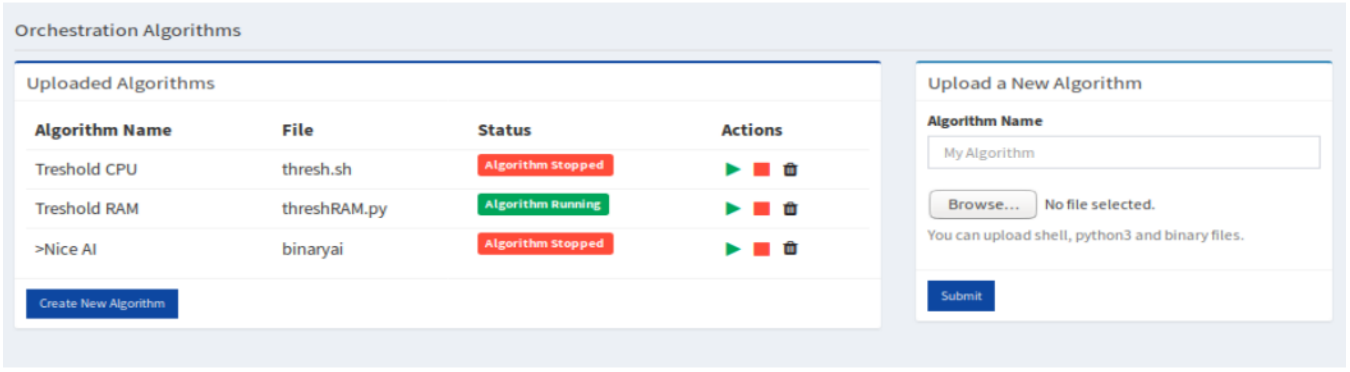


Figure 6: COPA Dashboard elements



Algorithm Name	File	Status	Actions
Treshold CPU	thresh.sh	Algorithm Stopped	▶ ■ 🗑
Treshold RAM	threshRAM.py	Algorithm Running	▶ ■ 🗑
>Nice AI	binaryai	Algorithm Stopped	▶ ■ 🗑

Figure 7: COPA Orchestration Management

number of researchers in a specific area. COPA provides a user-friendly container orchestration graphical interface to improve the research community's adherence in such an important subject that is container orchestration.

The experimenter must know the components that compose the experimental environment and the changes during the time of execution to orchestrate containers. When designing COPA's Graphical Interface, we always looked for a simple way to show the necessary information. When the user enters on the COPA screen, it can already see the available experiment resources through the dashboard, shown in Figure 6. Using the dashboard, the experimenter can inform himself about the number of Container Pools in the experiment, the total number of containers present in the configuration, and the number of the containers on each pool. The dashboard also counts with charts for each Container Pool on CPU, memory, and network throughput utilization. For more information about Container Pools or management actions, the experimenter can go for the dedicated Web page for Container Pool.

In the Container Pool page, the experimenter can manually manage hosted containers and also click on them for more information. On this page, the experimenter also can upload a new container. Moreover, if the Container Pool has a wireless interface, it is also possible to check the connected devices and the Signal-to-Noise Ratio (SNR) concerning each of them. In our experiments, COPA can monitor one Container Pool containing a WiFi network card. One-click away, in the side-menu, and the experimenter can find the orchestration tab, where Orchestration Manager is located. On this page, the experimenter can find the list of already uploaded orchestration algorithms and a section for algorithm upload, as can be seen in Figure 7. Using Orchestration Manager, the experimenter can start, stop, and delete an algorithm. Furthermore, COPA enables the upload of a custom orchestration algorithm. The only requirement to run the algorithm is that Containers Pools have the dependencies for the execution, and the script deploys the orchestration actions through COPA API. The algorithm can be fed with the monitoring data gathered by COPA through COPA API, which is described in the COPA's code repository [45].

D. QUANTITATIVE RESULTS

Our objective is to understand COPA behavior in an experimental environment. Therefore, we deployed two different scenarios, one with COPA running on VMs, and other without COPA. In these two scenarios, we collected three types of data: (*i*) the resources utilization from Central Node and the Container Pool, (*ii*) the network overhead caused by monitoring and control messages in the Container Pool, and (*iii*) the overhead caused in the migration by utilizing COPA API besides directly connecting to LXD API.

For evaluating resource utilization, we gathered data from CPU, Memory, and I/O Disk usage from Central Node and Container Pool, as can be seen in Figure 8 (A) and (B). This experiment was run for 120s and collected every 1 second in the Central Node and all Container Pools. In the Central Node, were gathered 120 samples of every resource, while the Container Pools were gathered 120 samples for each, which in total are 360 samples of every Container Pools' resource. All the means were calculated with a 95% confidence interval. The Central Node's resource evaluation (A) shows a small increase in the CPU, memory, and disk I/O utilization. This increase adds up at least 3.7% of single-core CPU overhead, 14.6% of a 1GB RAM (149.5MB), and 18.72 Disk I/O operations per second on average. The increment is due to the Central Node, which has a Web server with some functionalities such as (*i*) receives requests from the experimenter using Graphical Interface and Monitoring Agents, and (*ii*) saves monitoring data in the Central Node's database. Therefore, like any other Web server, when increasing the number of users or Monitoring Agents, it is expected that COPA will increase resource usage.

In Figure 8 (B), the Container Pool resource utilization is detailed. Compared to a scenario without COPA, Container Pool causes an overhead of 0.22% on CPU, 0.68% on RAM (6.9MB), and no change in Disk I/O operations on average. Therefore, we can say that Container Pool does not cause a significant overhead at its host. Moreover, Monitoring Agent, a component inside Container Pool, can be executed inside resource-constrained devices, such as Raspberry Pis, making them part of the experiment environment. Furthermore, Container Pool can escalate quickly because its resource utilization is not related to the number of devices in the experiment scenario. The only circumstance that can increase in Container Pool is the number of messages changed among Container Pools for monitoring purposes.

The network download and upload of Container Pool in an experiment of 120 seconds, as shown in Figure 8 (C) and (D). The network bandwidth usage pattern changes during the evaluation because the charts' data are from the very beginning of the experimentation when COPA is started. In the 0 seconds, we can observe a high pick in network bandwidth in both charts. This behavior is due to the SSH connection utilized to start the experiment in multiples VMs simultaneously. The initial pick is followed by the second pick, which is the Central Node communicating every LXD server hosted by Container Pool VM. After that, we can notice some seconds without any communication once the Central Node service is waiting for all Container Pools. Next, it starts the monitoring where we can see a communication pattern starts. Besides these patterns, we also can notice some high picks that excel from the communication pattern. We assume that is the keep-alive beacons from SSH utilized to run the experiment or some background traffic generated by the operational system. On average, the experimenter must expect at least 0.52 KBps at the uplink and 0.40 KBps when executing this use case's scenario. Therefore, we can consider that there is no substantial network overhead. Moreover, this control and monitoring messages traffic through the core network is typical using optical fibers and other high bandwidth communication technologies in a MEC scenario.

Finally, we evaluated the difference in response time when migrating a container through COPA API or by Bare LXD API, as shown in Figure 8 (E). Different from the Central Node and Container Pool resources' evaluation, the migration evaluation could not be gathered continuously during the 120s. We evaluated this experiment by migrating an Alpine 3.9 container 25 times randomly between the Container Pools. The mean was calculated with a 95% confidence interval. The results show that the API adds 0.66 seconds on average to the migration command, which can be not relevant for bigger container sizes. Therefore, we consider that COPA API does not add significant overhead in migration container response time.

VI. CONCLUSION

In this article, we present some of the potentials that COPA can play in container orchestration experimentation. Experimenters can configure COPA on the local environment. Moreover, testbed owners can provide COPA as a service for experimenters accessing the facilities. In COPA's graphical interface, users can easily find all the available features in a friendly interface and follow the experimental resources' monitoring. For orchestration algorithm experimentation, COPA provides an interface with the essentials configuration for managing the algorithms at run time. Not all the potential features were developed for this research, such as the Workflow module, and we plan to build it as future work.

Besides the not yet developed components, COPA can be extended to support features for service chaining that is a central area inside container orchestration applied to 5G networks. Furthermore, with the existence of multiple operators working on a collaboration to provide network connectivity, and each of the operators may have different orchestrators for its network. Thinking of that, the research on distributed container orchestration can increase productivity if a tool like COPA provides the right features for the experimentation.

We intend to improve the COPA's monitoring system to provide more information about the network's quality and resources to increase the efficacy of the orchestrators

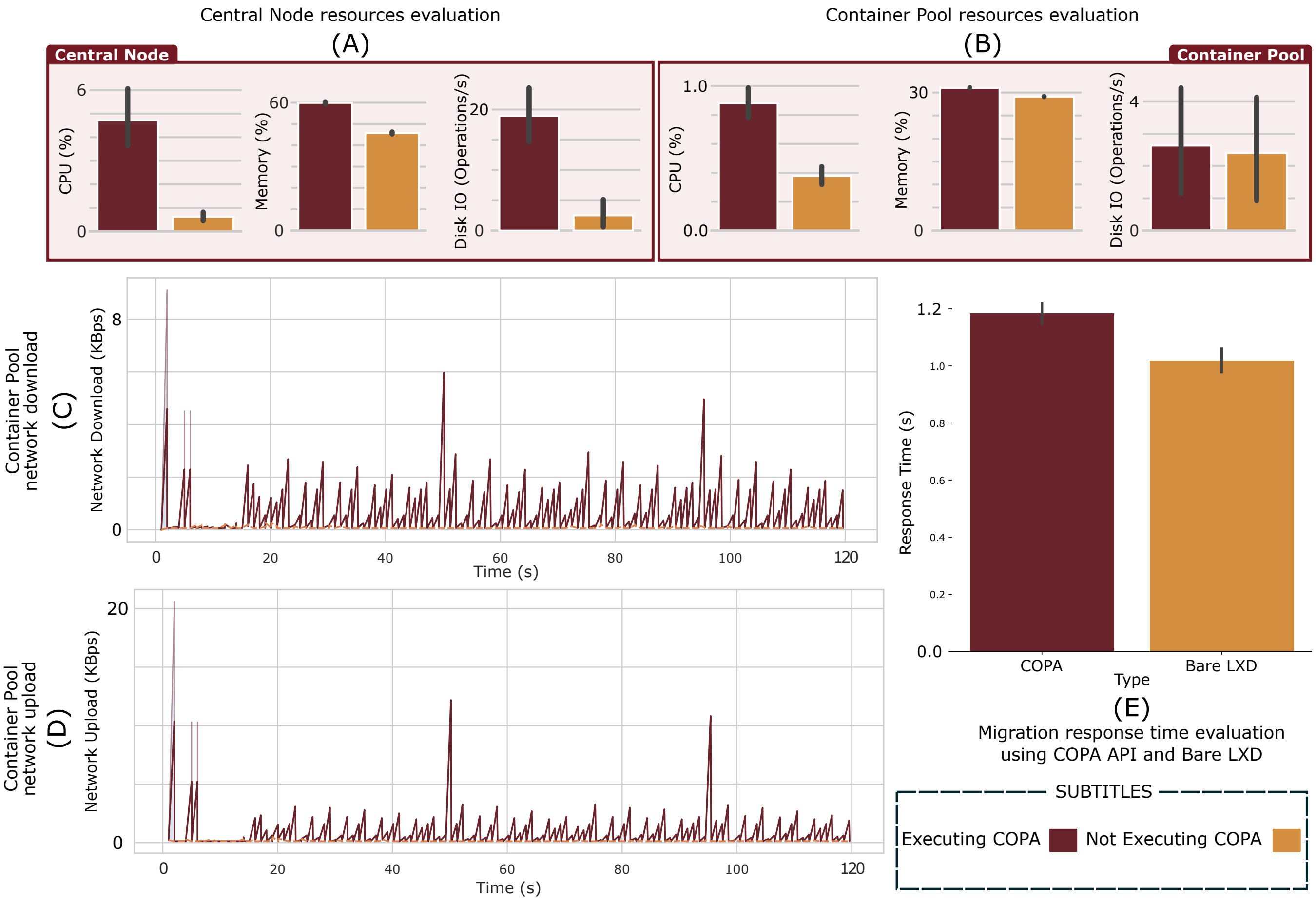


Figure 8: Evaluation charts for Central Node and Container Pool resources, Container Pool's Network (Download/Upload), and COPA API response time overhead.

deployed in COPA. We are already looking forward to supporting a MEC Radio Access Network (RAN) manager software called Low-Latency (LL-MEC), which exposes the RAN information for the upper layer, enabling applications to read this information and improve QoS. This new functionality will that the orchestrators on COPA to make decisions based on the radio network quality.

Another significant enhancement of COPA is to increase container management support. In this case, the experimenter can create its containerized services in any technology they want. Furthermore, it is essential to support different solutions for container management since they have other characteristics, and one experiment can experience better performance when running in a specific technology.

COPA is already available in the project H2020 FUTEBOL's testbeds and ready to use. Experiments of the FUTEBOL project were deployed on top of our solution and demonstrated in project deliverables and results, proving its usage. Finally, the tool is functional and full of potential for further improvements. We hope that it will be beneficial for future experimenters and help advance the research of future networks state-of-the-art faster than before.

ACKNOWLEDGMENT

This work has been performed in the framework of the European Union's Horizon 2020 project 5G-CARMEN co-funded by the EU under grant agreement No. 825012, and project FUTEBOL under grant agreement no. 688941, as well from the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC) through RNP, CTIC, and CNPq. The views expressed are those of the authors and do not necessarily represent the projects. The Commission is not liable for any use that may be made of any of the information contained therein.

References

[1] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A Survey on Service Migration in Mobile Edge Computing," IEEE Access, vol. 6, pp. 23 511–23 528, 2018. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2018.2828102
[2] T. Taleb, S. Member, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," vol. 19, no. 3, pp. 1657–1681, 2017. [Online]. Available: https://doi.org/10.1109/COMST.2017.2705720
[3] "Multi-access Edge Computing (MEC); Framework and Reference Architecture," European Telecommunications Standards Institute (ETSI), Standard, Jan. 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf
[4] Y. Wang, A. Carzaniga, and A. Wolf, "Four enhancements to automated distributed system experimentation methods," Proceedings - International

Conference on Software Engineering, pp. 491–500, 2008. [Online]. Available: https://doi.org/10.1145/1368088.1368155

[5] L. Leonini, É. Rivière, and P. Felber, "SPLAY: Distributed Systems Evaluation Made Simple (or How to Turn Ideas into Live Systems in a Breeze)." Nsdi, vol. 9, pp. 185–198, 2009. [Online]. Available: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/leonini/leonini.pdf

[6] M. Gao, B. Addis, M. Bouet, and S. Secci, "Optimal orchestration of virtual network functions," Computer Networks, 2018. [Online]. Available: https://doi.org/10.1016/j.comnet.2018.06.006

[7] B. Li, W. Lu, S. Liu, and Z. Zhu, "Deep-learning-assisted network orchestration for on-demand and cost-effective VNF service chaining in inter-DC elastic optical networks," Journal of Optical Communications and Networking, 2018. [Online]. Available: https://doi.org/10.1364/JOCN.10.000D29

[8] T. V. Do, N. H. Do, H. T. Nguyen, C. Rotter, A. Hegyi, and P. Hegyi, "Comparison of scheduling algorithms for multiple mobile computing edge clouds," Simulation Modelling Practice and Theory, vol. 93, no. June 2018, pp. 104–118, 2019. [Online]. Available: https://doi.org/10.1016/j.simpat.2018.10.005

[9] D. Sabella, N. Nikaein, A. Huang, J. Xhembulla, G. Malnati, and S. Scarpina, "A hierarchical MEC architecture: Experimenting the RAVEN use-case," IEEE Vehicular Technology Conference, vol. 2018-June, pp. 1–5, 2018. [Online]. Available: https://doi.org/10.1109/VTCSpring.2018.8417826

[10] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," IEEE Internet of Things Journal, vol. 3, no. 6, pp. 854–864, 2016. [Online]. Available: https://doi.org/10.1109/JIOT.2016.2584538

[11] M. Aazam and E. N. Huh, "Fog Computing: The Cloud-IoT/IoE Middleware Paradigm," IEEE Potentials, vol. 35, no. 3, pp. 40–44, 2016. [Online]. Available: https://doi.org/10.1109/MPOT.2015.2456213

[12] M. Yannuzzi, F. Van Lingen, A. Jain, O. L. Parellada, M. M. Flores, D. Carrera, J. L. Perez, D. Montero, P. Chacin, A. Corsaro, and A. Olive, "A new era for cities with fog computing," IEEE Internet Computing, vol. 21, no. 2, pp. 54–67, 2017. [Online]. Available: https://doi.org/10.1109/MIC.2017.25

[13] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," IEEE Communications Magazine, vol. 49, no. 11, pp. 58–67, 2011. [Online]. Available: https://doi.org/10.1109/MCOM.2011.6069710

[14] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015, pp. 386–393, 2015. [Online]. Available: https://doi.org/10.1109/IC2E.2015.74

[15] S. Anish Babu, M. J. Hareesh, J. P. Martin, S. Cherian, and Y. Sastri, "System performance evaluation of para virtualization, container virtualization, and full virtualization using Xen, OpenVZ, and XenServer," Proceedings - 2014 4th International Conference on Advances in Computing and Communications, ICACC 2014, pp. 247–250, 2014. [Online]. Available: https://doi.org/10.1109/ICACC.2014.66

[16] (2020, Sep.) Open Source MANO. web site. European Telecommunications Standards Institute (ETSI). [Online]. Available: https://osm.etsi.org/

[17] (2020, sep) ONAP - Open Network Automation Platform. web site. The Linux Foundation projects. [Online]. Available: https://wiki.onap.org/

[18] (2020, Sep.) Production-grade container orchestration - kubernetes. web site. The Linux Foundation. [Online]. Available: https://kubernetes.io/

[19] (2020, Sep.) Linux Containers - LXD - Introduction. web site. Canonical Ltd. [Online]. Available: https://linuxcontainers.org/lxd/introduction/

[20] (2020, Sep.) Swarm mode overview | Docker Documentation. web site. Docker Inc. [Online]. Available: https://docs.docker.com/engine/swarm/

[21] M. Lacage, M. Ferrari, M. Hansen, T. Turletti, and W. Dabbous, "NEPI: using independent simulators, emulators, and testbeds for easy experimentation," SIGOPS Oper. Syst. Rev., vol. 43, no. 4, pp. 60–65, 2010. [Online]. Available: https://doi.org/10.1145/1713254.1713268

[22] J. Albrecht, C. Tuttle, R. Braud, D. Dao, N. Topilski, A. C. Snoeren, and A. Vahdat, "Distributed application configuration, management, and visualization with plush," ACM Transactions on Internet Technology, vol. 11, no. 2, pp. 1–41, 2011. [Online]. Available: https://doi.org/10.1145/2049656.2049658

[23] (2020, Sep.) jFed. web site. imec. [Online]. Available: https://jfed.ilabt.imec.be/

[24] J. Griffioen, Z. Fei, H. Nasir, X. Wu, J. Reed, and C. Carpenter, "The design of an instrumentation system for federated and virtualized network testbeds," Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012, pp. 1260–1267, 2012. [Online]. Available: https://doi.org/10.1109/NOMS.2012.6212061

[25] J. Struye, B. Braem, S. Latr, and J. Marquez-barja, "The CityLab Testbed - Large-scale Multi-technology Wireless Experimentation in a City Environment : Neural Network-based Interference Prediction in a Smart City," 2018 IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds, pp. 529–534, 2018. [Online]. Available: https://doi.org/10.1109/INFCOMW.2018.8407018

[26] (2020, Sep.) FUTEBOL – Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory. web site. FUTEBOL. [Online]. Available: http://www.ict-futebol.org.br/

[27] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," Computer Networks, vol. 61, pp. 5–23, 2014. [Online]. Available: https://doi.org/10.1016/j.bjp.2013.12.037

[28] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: an overlay testbed for broad-coverage services," ACM SIGCOMM Computer Communication Review, no. 3, p. 3, 2003. [Online]. Available: https://doi.org/10.1145/956993.956995

[29] A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, "Exploring container virtualization in IoT clouds," in Smart Computing (SMARTCOMP), 2016 IEEE International Conference on. IEEE, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/SMARTCOMP.2016.7501691

[30] M. Peng, K. Zhang, J. Jiang, J. Wang, and W. Wang, "Energy-efficient resource assignment and power allocation in heterogeneous cloud radio access networks," IEEE Transactions on Vehicular Technology, vol. 64, no. 11, pp. 5275–5287, 2015. [Online]. Available: https://doi.org/10.1109/TVT.2014.2379922

[31] P. W. Tsai, F. Piccialli, C. W. Tsai, M. Y. Luo, and C. S. Yang, "Control frameworks in network emulation testbeds: A survey," Journal of Computational Science, vol. 22, pp. 148–161, 2017. [Online]. Available: https://doi.org/10.1016/j.jocs.2017.03.003

[32] (2020, Sep.) Docker - build, ship, and run any app, anywhere. web site. Docker Inc. [Online]. Available: https://www.docker.com/

[33] (2020, Sep.) rkt, a security-minded, standards-based container engine. web site. Red Hat, Inc. [Online]. Available: https://coreos.com/rkt/

[34] (2020, Sep.) Docker market share and competitor report | compare to docker, kubernetes, apache mesos | datanyze. web site. Datanyze. [Online]. Available: https://www.datanyze.com/market-share/containerization/docker-market-share

[35] (2020, Sep.) Build the future of open infrastructure. web site. Rackspace Cloud Computing. [Online]. Available: https://www.openstack.org/

[36] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms," Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, vol. 2017-December, pp. 162–169, 2017. [Online]. Available: https://doi.org/10.1109/CloudCom.2017.15

[37] (2020, Sep.) OpenFaaS - Serverless Functions Made Simple. web site. OpenFaaS. [Online]. Available: https://www.openfaas.com/

[38] (2020, Sep.) Apache OpenWhisk is a serverless, open source cloud platform. web site. The Apache Software Foundation. [Online]. Available: https://openwhisk.apache.org/

[39] J. Cappos, A. Rafetseder, M. Hemmings, R. McGeer, and G. Ricart, "EdgeNet: A global cloud that spreads by local action," Proceedings - 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018, pp. 359–360, 2018. [Online]. Available: https://doi.org/10.1109/SEC.2018.00045

[40] A. G. Dalla-Costa, L. Bondan, J. A. Wickboldt, C. B. Both, and L. Z. Granville, "Maestro: An NFV Orchestrator for Wireless Environments Aware of VNF Internal Compositions," in 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), March 2017, pp. 484–491. [Online]. Available: https://doi.org/10.1109/AINA.2017.126

[41] (2020, sep) Tacker - OpenStack NFV Orchestration. web site. OpenStack Foundation. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[42] (2020, sep) Open Baton: an open source reference implementation of the ETSI Network Function Virtualization MANO specification. web site. Fraunhofer Fokus and TU Berlin. [Online]. Available: https://openbaton.github.io/

[43] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source nfv mano systems: Osm and onap," Computer

Communications, vol. 161, pp. 86 – 98, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366420305946

[44] (2020, Sep.) The Web framework for perfectionists with deadlines | Django. web site. Django Software Foundation. [Online]. Available: https://www.djangoproject.com/

[45] (2020, Sep.) COPA: Experimenter-level container orchestration for testbeds. GitLab repository. FUTEBOL – Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory. [Online]. Available: https://gitlab.com/futebol/COPA

HENRIQUE CESAR CARVALHO DE RESENDE is a Ph.D. at the UAntwerpen's Faculty of Applied Engineering (FTI) guided by Professor Marquez-Barja in the group of Wireless Networks focusing his studies in the area of Mobile Edge Computing (MEC) and microservices to provide network end-to-end connectivity. He graduated in Computer Science by the Federal University of Rio Grande do Sul (UFRGS). During his graduation, he participated in several Computer Networks projects such as MEICAN and FE-CoLisEU, both sponsored by RNP, one of the major research institutions in Brazil. During three years, Henrique collaborated with the H2020 FUTEBOL Project which aims to research about wireless-optical network convergence. In this project, he started working with remote computing, and microservices orchestration focused on wireless-optical infrastructures.

MATIAS A. K. SCHIMUNECK is a Ph.D. student in computer networks at the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He received his M.Sc. title from INF of UFRGS in 2017. He holds his bachelor's degree in Computer Science at the University of Santa Cruz do Sul (UNISC) in 2014. His research involves wireless networks, next generation networks, Internet of things, software defined radio, cognitive radio networks, and heterogeneous cloud radio access networks.

CRISTIANO BONATO BOTH is an associate professor of the Applied Computing Graduate Program at the University of Vale do Rio dos Sinos (UNISINOS), Brazil. He coordinators research projects funded by H2020 EU-Brazil, CNPq, FAPERGS, and RNP. His research focuses on wireless networks, next-generation networks, softwarization and virtualization technologies for telecommunication networks, and SDN-like solutions for the Internet of Things. He is participating in several Technical Programme and Organizing Committees for different worldwide conferences and congresses.

JULIANO ARAUJO WICKBOLDT is an associate professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He holds a Ph.D. degree from the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He achieved his M.Sc. degree from UFRGS in a joint project with HP Labs Bristol and Palo Alto. He also received a B.Sc. degree in computer science at the Pontifical Catholic University of Rio Grande do Sul in 2006. Juliano was an intern at NEC Labs Europe in Heidelberg, Germany from 2011 to 2012. His current research interests include cloud resource management and software-defined networking.

JOHANN MARQUEZ-BARJA is a Professor at University of Antwerp, as well as a Professor in IMEC, Belgium. He is leading the Wireless Cluster at IDLab/imec Antwerp. He was and is involved in several European research projects such as CREW, FORGE, WiSHFUL, Fed4FIRE/FAVORITE, Fed4FIRE+, eWINE, CONCORDA, 5G-CARMEN, FLEXNET, FUTEBOL (Technical Coordinator), 5G-Mobix, PROTEGO, InterConnect, and 5G-Blueprint projects (Technical Coordinator). He is a member of ACM, and a Senior member of the IEEE Communications Society and IEEE Education Society where he participates in the board of the Standards Committee. His main research interests are: 5G advanced architectures including edge computing; flexible and programmable future end-to-end networks; IoT communications and applications. He is also interested in vehicular communications, mobility, and smart cities deployments. Prof. Marquez-Barja is co-leading the Citylab Smart City testbed, part of the City of Things programme, and the SmartHighway testbed, both located in Antwerp, Belgium. Furthermore, he is also interested and active on education development, being actively involved in different research actions to enhance engineering education, in particular remote experimentation in e-learning systems. Prof. Marquez-Barja has given several keynotes and invited talks in different major events, as well as received 30 awards in his career so far, and co-authored more than 100 articles. He is also serving as Editor and Guest editor for different International Journals, as well as participating in several Technical Programme and Organizing Committees for several worldwide conferences/congresses.