UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

HENRIQUE CESAR CARVALHO DE RESENDE

COPA: A wireline/wireless convergent network monitoring and container manager architecture

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Juliano Araújo Wickboldt

Porto Alegre
July 2018

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Juliano Wickboldt, for all the guidance he provided me. I would also like to thank my first advisor, from my early years of college, Cristiano Both, for introducing me to the academic world, and teaching me important skills to thrive both on my academic and professional life.

I wish also express my sincere gratitude to my network labs colleagues, Matias Schimuneck, Ricardo Pfitscher, Arthur Jacobs, Marcelo Marotta and Iulisloi Zacarias, for all the help they gave me making this project, guiding me through all my mistakes and making sure I stayed focused and motivated. My deepest gratitude to all my university colleagues, specially Felipe Estima da Silveira, for never hesitating to give a helping hand when I needed, and for sharing such great moments through out this long academic years.

I would like to thank my mother, Ana Cândida Costa Carvalho, and my sister, Mariah Costa Carvalho de Resende, for always being by my side support all my life choices and mistakes. I would like to thank Corina Post, my psychiatrist, for her excellent work helping me to overcome my difficulties and always moving forward.

Last but not least, I would like to thank Leticia Caroline Berle, for all the love and support she gave me through the years we have been together, for providing great moments of life and comforting me when everything seemed to go wrong. I am certain I would not have made it this far without the help of these people.

ABSTRACT

New applications are demanding several improvements in the quality of computer network technologies and infrastructure. These new applications that are expected to operate in dynamic environments and computing solutions must provide similar performance even in different network and resources qualities scenarios. Convergent wireless and wireline networks provides mobile and high data rate transfer and solves some of this new requirements such as mobility and low-latency network communication. Also, remote computing such as Cloud and Fog computing has arisen as an approach to deal with latency issues in this context. This group of computing technologies enables several new studies areas. Therefore, to support the research and development solutions for future computer networks, we design COPA, a wireless/wireline network convergent monitoring and container manager architecture for testbeds. This architecture enable the testbeds experimental research by providing real-time wireless and wireline network monitoring and experimenter-level orchestration. The monitoring of the network provides network quality data for decision-making algorithms. Also, we provide a friendly user interface for following the experiment network scenario. While, the experimenter-level orchestration enables the emulation of Cloud and Fog interplay by providing tools for management of virtualized environments such as containers. We also present a use case of COPA. In this use case we developed a smart lighting IoT system that allows control of light bulbs (turn on/off, color and brightness change). This IoT device could be controlled by voice commands which are processed and translated to light bulb language in a remote server. This server could be at a Cloud computing, Fog computing datacenter or close to the device at some kind of smart gateway. We show in this use case, how COPA can provide the necessary data for convergent wireless/wireline networks related research. Finally, we conclude that COPA provide a excellent environment for researches of wireless/wireline convergent network and orchestration algorithms for containerized services.

Keywords: Convergent Monitoring. QoS. Cloud. Testbed. IoT.

COPA: Uma arquitetura para monitoramento de redes sem-fio/cabeadas e gerenciamento de *containers*

RESUMO

Novas aplicações estão demandando muitas melhoras na qualidade das tecnologias e infraestrutura de redes de computadores. É esperado que essas novas aplicações operem em ambientes dinâmicos, e soluções de computação precisam prover uma performance similar mesmo em diferentes qualidades de recursos computacionais ou de rede. Redes convergentes sem-fio e cabeado provem conectividade móvel e alta taxa de transferência de dados solucionando alguns requisitos dessas novas tecnologias tais como mobilidade e baixa latência de comunicação de rede. Soluções de computação remota tais como *Cloud* e *Fog computing* vem sendo utilizadas para melhorar problemas de atraso de rede nesse contexto. Esse grupo de tecnologias de informática habilitam o estudo em diversas novas áreas. Então, para apoiar a pesquisa e desenvolvimento de soluções para redes de computadores do futuro, COPA foi arquitetado, uma arquitetura para monitoramento de redes convergentes sem-fio/cabeadas e gerenciamento de *containers* para laboratórios experimentais. Essa arquitetura habilita a pesquisa em ambientes experimentais provendo monitoramento de rede sem-fio e cabeada em tempo real e orquestração em nível de experimentador. Também é apresentado um uso de caso do COPA. Nesse uso de caso, é desenvolvido um sistema de luzes inteligentes que permite o controle de uma lampada (liga/desliga, cor e intensidade do brilho). Esse dispositivo pode ser controlado por comandos de voz os quais são processados em um servidor remoto e traduzidos para a linguagem da luz inteligente. Esse servidor poderia estar localizado em um centro de dados de *Cloud computing*, *Fog computing* or perto do usuário final em algum tipo de *smart gateway*. Nesse uso de caso, é mostrado como o COPA pode prover os dados necessários para pesquisas relacionadas a convergência de redes sem-fio/cabeadas. Por fim, é concluído que o COPA fornece um excelente ambiente para pesquisas de redes convergentes sem-fio/cabeadas e algorítmos de orquestração de serviços virtualizados.

Palavras-chave: Monitoramento convergente, Qualidade de Serviço, Cloud computing, Redes, IoT.

LIST OF FIGURES

Figure 2.1 Cloud and Fog architecture 15

Figure 3.1 COPA software architecture 20
Figure 3.2 Dashboard mockup 21
Figure 3.3 Monitor Agent state machine. 23
Figure 3.4 Monitor Controller algorithm. 24
Figure 3.5 Detailed Orchestrator Architecture 25

Figure 4.1 COPA dashboard 27
Figure 4.2 Monitor state machine. 34
Figure 4.3 Containers actions 35
Figure 4.4 Containers actions 36

Figure 5.1 Experiment architecture 39
Figure 5.2 Convergent monitoring setup 40
Figure 5.3 Options provided by JFed federation tool 42
Figure 5.4 Allocated COPA resources in JFed 43
Figure 5.5 Monitoring of voice recognition host and container 44
Figure 5.6 Monitoring of end-to-end link between Local and Cloud tier 45
Figure 5.7 Monitoring of IoT device wireless connection 45
Figure 5.8 Automatic migration stages sequence 47

LIST OF TABLES

Table 4.1 Core app code organization 31
Table 4.2 Monitor app code organization 31

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Program Interface
C-RAN	Cloud Radio Access Networks
CPU	Central Processing Unit
CRIU	Checkpoint/Restore In Userspace
CRUD	Create, Read, Update e Delete
CSV	Comma-Separated Values
DCL	Database Communication Library
F-RAN	Fog-Radio Access Network
FUTEBOL	Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory
HTML	Hypertext Markup Language
IaaS	Infrastructure as a Service
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LR-WPAN	Low-Rate Wireless Personal Area Networks
MDC	Micro Data Center
MVC	Model, View, and Controller
NFV	Network Function Virtualization
OWAMP	One-Way Active Measurement Protocol
REST	Representational State Transfer
SaaS	Software as a service
SRE	Service Run Environment
UFMG	Federal University of Minas Gerais
UFRGS	Federal University of Rio Grande do Sul

UNIVBris	University of Bristol
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VM	Virtual Machine

CONTENTS

1 INTRODUCTION..11
2 BACKGROUND AND RELATED WORK..13
2.1 Cloud and Fog Computing...13
2.2 Convergent Networks..14
2.3 Testbed Environments...16
2.4 Related Work...18
3 COPA: THE SOFTWARE ARCHITECTURE..19
3.1 Web Server..19
3.2 Monitoring...22
3.3 Orchestrating...25
4 IMPLEMENTATION...27
4.1 Technologies...27
4.1.1 Django...28
4.1.2 OWAMP...28
4.1.3 LXD...29
4.2 Coding details..29
4.2.1 Web Server..30
4.2.2 COPA Controller...31
4.2.3 COPA Orchestrator...32
4.2.4 COPA Agent...33
4.3 Graphical Interface..34
5 ADAPTIVE CLOUD/FOG FOR IOT...37
5.1 Use case motivation..37
5.2 Showcase set up...38
5.3 Experiment deployment...40
5.4 Experiment visualization..43
6 CONCLUSION AND FUTURE WORK...48
REFERENCES..50

1 INTRODUCTION

New applications are putting computer networks to test several demanding improvements in the quality of computer network technologies and infrastructure. Mobile and low-latency communication are some of these applications requirements. Wireless networks provide mobile communication for devices, while there are some wireline network technologies capable of high data rate transfer. Fast data transference reduces the network latency and provides fast communication between devices. Convergent wireline/wireless networks solves some new applications requirements. However, there are several cases such as Cloud Radio Access Networks [Checko H. Christiansen 2015], and others mixed wireline/wireless network infrastructure models that need to be studied more deeply.

Cloud computing is a paradigm that enables services over the Internet. Retrained resources devices are not able to make complex processing or store too much information. These devices sometimes do not have powerful hardware or, in others situations, needs to decrease the battery usage. Cloud computing enables complex processing, large data storage in this retrained-devices by providing services [Armbrust I. Stoica 2010]. Nevertheless, Cloud computing data center many times may be located far away from the end-user, because of capital expenditure and operational expenditure decisions. This distance from the service and the user causes an increase in the application response time. Nonetheless, Fog computing is a paradigm that enables the services over the Internet providing extended capabilities to restrained-resources devices closer to the user than a Cloud infrastructure [Bonomi R. Milito 2012].

Fog computing infrastructure size and processing power are smaller than a Cloud computing facility. The smaller size of the infrastructure decreases the cost of deploying a computing facility and enables the deployment of Fog facilities in several regions closer to the end-user. Being closer to the user and in the network edge provide a lower network latency than Cloud computing facilities. Fog computing enables remotely executing of latency-sensitive applications computing. However, Fog may not have the sufficient processing power to some computing routines and may delay the applications response time. In this cases, Cloud computing infrastructure may decrease the processing time of the application computing routine sufficiently to compensates the high network latency, decreasing the application response time. Therefore, the interplay between Cloud and Fog computing may solve the deployment of latency-sensitive applications computing routines.

Future networks may count with Cloud and Fog computing over a wireline/wireless convergent network. These computing technologies may work together to provide the best communication performance for the mobile devices. Improving the quality of service of the computer network opens many research areas that need testing. In this case, testbeds are experimental environments to help the research of specifics scenarios. This environment provides resources and tools for deployment of experiments. Testbeds also have the advantage of owning the servers and the network infrastructure of the experiment. By owning it, testbeds can provide tools to monitor and manage its resources, enabling the reproduction of several real-life scenarios. As far as we know, there are not any testbed solution to decrease the time spent with experimentation setup of the interplay between Cloud and Fog computing over a wireless/wireline convergent network researches.

In this monograph, we introduce COPA, a convergent wireline/wireless network monitor and container manager. COPA enables an easy experimentation environment setup for testbeds. Our solution can monitor computer resources, wireless network and wireline network quality. To emulate a Cloud and Fog interplay scenarios, we count on container virtualization technologies. Container virtualization provides a virtual environment similar to Virtual Machines, but with less overhead in the virtualization host. Combining the monitoring tool and the virtualization management, COPA can deploy, manage or even migrate application servers among servers. Therefore, providing COPA in this study, we aim to accelerate and help the research in future networks by providing, also, a friendly user interface.

The remaining of this document is organized as follows. In Chapter 2, we present the background regarding cloud and fog computing, convergent networks, and testbed environments. In Chapter 3, we present the architecture of our solution. In Chapter 4, we present our implementation of the described testbed environment tool. In Chapter 5, we present an use case where our tool is essential to analyze experimental scenarios. In Chapter 6, we conclude this document and present future work.

2 BACKGROUND AND RELATED WORK

In this Chapter, we introduce the background knowledge pertinent to this document, required to fully comprehend our solution. We also present the related work regarding future networks, thus Cloud and Fog-based environments deployed over a convergent wireline/wireless network infrastructure. Also the importance of experimentation in this scenarios.

2.1 Cloud and Fog Computing

Cloud computing is a paradigm that enables services over the internet. Likewise, it may refer to the hardwares and systems in the data centers that provides this services. There are many approaches that Cloud computing can be accessed by a client. Cloud companies may offer a service to the public this is referred as Software as a Service (SaaS). Furthermore, when the Cloud company offers the hardware or a system that runs on the data center the terms used are Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), respectively [Armbrust I. Stoica 2010]. The data center software and hardware is what we call Cloud. In this work, we focus in SaaS since we plan to provide better management solutions to IoT services.

Now a days, there are several computing devices that makes part of our daily routine. It can be a smartphone or a health care sensor for elderly care, this devices have a diversity of characteristics such as battery saving, low processing resources or a small data storage. To achieve this objectives, this devices counts with the structure of large data centers for remote processing and storage. Also, each data center guarantees high-availability of it infrastructure in certain level, turning it targets for big applications companies to trust their services. Big data centers are really expensive to build and several conditions are considered before choosing the right place to initiate the construction. Hence, cloud companies, many times, chose to build their data centers far away from the end-user for several reasons. This distance may increase the network latency between the server application and the final user, and may or may not affects in its quality of experience.

Fog computing is a paradigm that refers to a platform for local computing, distribution and storage to limited-resources devices. Different from Cloud computing, fog does not counts on a large infrastructure, but Micro Data Centers (MDC) geographically distributed or even Smart Gateways located in the very network edge. Fog computing reduced size turns it a restrained processing resource, however it makes possible the deployment of such infrastructure at the network edge. Being at the network edge makes fog closer to the end-user than a standard cloud computing data center. Many companies are trusting its services to this micro data centers trying to reach a better service response time performance for its network latency-sensitive applications. Therefore, cloud computing data centers may not qualify for many of applications requirements, turning fog computing one of the main alternative for latency-sensitive applications.

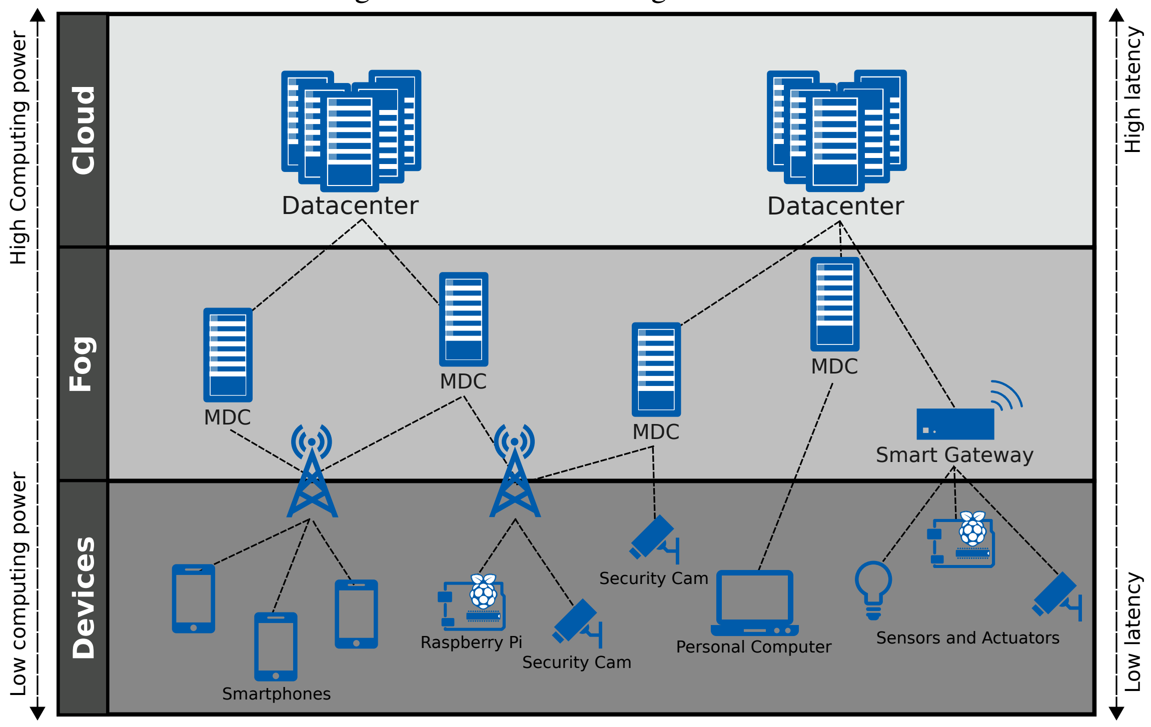
However, fog computing have its downsides. Because of its retrained processing resources, the deployment of massive processing applications may be very costful for the MDCs increasing, then, the service response time. Complex computing routines may turn small data center a bottleneck to remote processing of such applications. Cloud computing data centers, on the other way, may reduce the processing time of such routines. Reducing the processing time considerably may compensates the high network latency characteristics of Cloud data centers. Therefore, it can reduces the response time as whole and may be the best alternative for this kind of applications.

Considering the diversity of applications and the variation of processing power requirement, the interplay between of Cloud and Fog computing data centers may be studied. A very know Cloud and Fog computing architecture is presented at figure 2.1. This architecture shows at the top the Cloud computing data centers providing high processing power and high network latency. Right under it, we have the Fog computing layer. This layer is composed by MDCs and smart gateways, providing low processing power and low network latency to the client applications. The interplay between this two paradigms can be achieved by monitoring the applications resource utilization and network quality. The monitoring data may provide the necessary information to decide where the application server will be better to be deployed in or migrated to.

2.2 Convergent Networks

New services are created frequently with the development of new technologies. The creation of this new services enables the increasing the computer network usage. The increasing of computer network demand only grows and have achieved more than 25 billion of Internet connected devices around the world. In order to support this demand, it needs to be provide an effective network infrastructure. 5G mobile networks may supply

Figure 2.1: Cloud and Fog architecture



Source: Author

part of this demand by providing high network capacity and excellent quality of service. Therefore, most of the modern devices and all kinds of applications are preferring mobile network connectivity from fixed. Because, the wireless technologies are achieving quality of services equivalents of the standards wireline technologies, besides having the mobility advantage.

Nevertheless, there are big urban conglomerates that requires support for connection of great amount of users. 5G wireless access point technologies, then, needs to support this number of users and forward the incoming data to a network communication medium capable to support the quality of service enabled by 5G. Optical fiber enables high data rate and low latency network communication. Hence, the best alternative for this scenario is to use optical fiber as a wireline network communication medium to provide internet connectivity to the 5G wireless network user.

Optical/wireless convergent networks infrastructure may be part of future networks infrastructure. However, uniting this two technologies requires a deeply study. Achieving the quality level of this convergent was a commercial demand and natural move for computer networks. But, providing this communication quality enables others areas of study such as Network Functions Virtualization (NFV) and latency-sensitive applications. NFV aims to migrate network-specific hardware components to software that can be run in generic hardware. This technology aims to reduce capital expenditure to deploy a computer network infrastructure by economizing in hardware and transferring the analogical signal processing to a powerful processing location such as data centers. Also, this technology provides flexibility to provide new network functions solutions by just changing the running software in a centralized center.

Furthermore, latency-sensitive applications could migrate from fixed to mobile network connectivity with the deployment of optical/wireless convergent infrastructures. This migration enables the growth of areas such as Healthcare which have some critical applications that requires quick feedback for the end-user. In this scenario several services could run in a Cloud computing data center and it still would provide a high quality user experience. The deployment of this futuristic infrastructure may be slow in several countries. Many places may not be prepared to spend its time and money in this kind of expensive infrastructure, opting, then, to find others solutions. Thus, in future, we could not always count with only wireless/optical convergence, but with any wireless/wireline infrastructure. To achieve similar results we may need to group up some study areas.

The interplay of cloud and fog computing in this transitional or alternatively scenarios to wireless/optical infrastructures may be a reasonable solution. The research and industry community are already providing experimental scenarios for future network studies and experiments. This places are called network testbeds and are explained in the next section.

2.3 Testbed Environments

Research and development are the bases of new high demand technologies. In many cases, researchers utilizes experimental use cases to prove their premises or solutions. However, the environments of this experiments are not always regular and predictable. Therefore, many times researchers are surprised with non-previewed events or external influences that may compromise the study. But is most of the cases the experimental research is essential for improvement and proof of technologies State-of-the-Art. Nonetheless, testbeds, experimental controlled environments, were created to facilitate the life of our fellows experimenters. Testbeds can vary from hands-on prototype development such as automobile industries to software development environments. In our case, we will focus in computer network testbeds in this work.

Computer network testbeds are mostly specialized in some technology or area such as Internet of Things, wireless network or convergent optical/wireless networks. In a IoT testbed, must be provided IoT devices such as raspberry pies or other hardware capable of communicating through protocols such as low-rate wireless personal area networks (LR-WPANs). For an wireless network testbed it can be provided, for example, Universal Software Radio Peripheral (USRPs) for deploying any kind of wireless protocol. USRPs are generic programmable radios that enables the development of several wireless network protocols. For an convergent optical/wireless network testbed, it is necessary the deployment of any wireless network technology connected to a optical fiber. This optical fiber may provide connectivity to a application servers or any kind of service. Then, it can be researched about this technologies.

Emulab is an example of network testbed. It helps researchers develop, debug and evaluate their systems in a wide range of environments. Emulab is located at Utah, United States of America and counts on two dozen sites. This sites provides resources to help computer science researchers in the fields of networking and distributed systems. Emulab counts with USRPs, 802.11 wireless nodes and tools to create complex network topologies. Also, we have Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory (FUTEBOL) [FUTEBOL 2018] that is a federation of testbeds which its purpose is to provide access to advanced facilities in Europe and Brazil for research and education across the wireless and optical domains. To achieve its objective, FUTEBOL has enabled the access of resources in universities testbeds such as UFRGS, UFMG, UNIVBris and Trinity College Dublin through a federative tool for testbed management, JFed.

FUTEBOL project counts with a diversity of resources. At UFRGS, the federation has Raspberry Pis, Arduinos with Xbee shield to experiment with IoT network protocols and USRPs to reproduce any wireless network protocol. Also, UFRGS FUTEBOL testbed counts with USRPs with optical fiber connectivity enabling the emulation of Radio Over Fiber experimentation or any optical/wireless network convergence experimentation. Not only at UFRGS, FUTEBOL counts we computer servers to deploy Virtual Machines (VMs). VMs are a generic and indispensable resource for any testbed. Through operational system emulation, this resource make possible create network gateways, servers or any virtual resource the experimentation may have.

Testbeds are very important for science, making experimentation easier and trustful. Also, for computer network research and development, it is the only possible option for and low-cost research, since many of the resources needed are very expensive and may be unaffordable for many of experimenters. FUTEBOL testbed will be focused in this work since it is specialized in wireless optical convergence.

2.4 Related Work

There are several factors that affect IoT scenarios, such as devices' features, geographical location, and network architecture. Yannuzzi [Yannuzzi et al. 2014] concludes that the current datacenters' locations will not be able to fulfill the requirements of foreseen IoT applications. Considering mobility, reliable control, and scalability, the author analyzes fog computing as a natural platform for IoT, also addressing the increasing importance of the interplay between fog and cloud in coming years.

Bibani [Bibani et al. 2016] proposes a hybrid fog/cloud architecture in order to provide a feasible IoT solution for low-latency applications such as firefighting. Authors demonstrate their approach, providing evidence that fog and cloud computing can alternate their computing role within the hybrid architecture to meet different application requirements; in particular, to reduce latency, the architecture can allocate the computing processes in the fog, closer to IoT devices.

Confirming these findings, Shi [Shih et al. 2016] presents a solution called Fog-Radio Access Network (F-RAN) which is able to bring efficient computing capability to the edge of the network, the fog, to meet the requirements of ultra-low latency applications. In order to have a comprehensive understanding of the interplay between fog and cloud computing in the IoT context, the convergence of optical and wireless networks must be taken into account. Munõz [Munoz et al. 2016] presents their 5G end-to-end experimental platform which combines heterogeneous optical/wireless networks, distributed cloud, and IoT devices. The authors acknowledge the importance of the fog/cloud interplay and the role of optical and wireless combined networks.

The articles presented in this sections provides a vision of how the computer network researchers are facing some future network solutions. After researching about Cloud and Fog computing, wireless/wireline convergent networks and testbeds, we designed a solution that enables research in such testbed environments scenarios. In the next chapter, we present COPA: a wireline/wireless convergent network monitoring and container manager architecture.

3 COPA: THE SOFTWARE ARCHITECTURE

In this chapter, we present COPA, a wireline/wireless convergent network monitoring and container manager architecture. This software architecture manages and monitors a Cloud and Fog computing scenario that utilizes container-based virtualization in a converged network testbed infrastructure. Furthermore, we show the architecture details per module explaining details about the execution and patterns about the theory behind the development of the software.

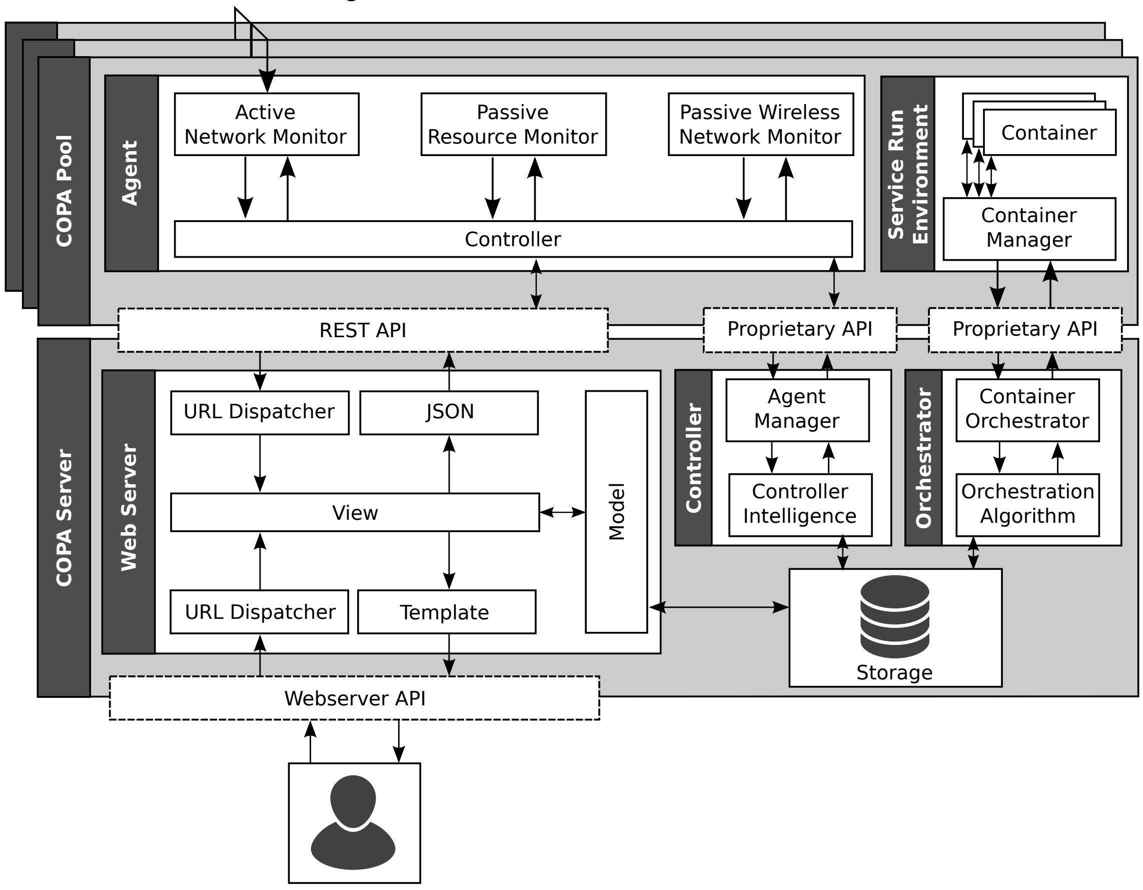
To fulfill all the requirements, COPA is composed of 3 major modules: the orchestrating, the monitoring and the Web Server module. We divided the orchestrating and monitoring modules into two submodules. This division was planned so we could have two distinct levels, the control layer, and the execution layer. The first submodule is the first entity which controls one or more execution layer entities that run on remote servers. The secondary layer, in case of the monitoring module, is called COPA Agent and, in the case of the orchestrator, are represented by the container manager software. Furthermore, all the control layer stays in an entity called COPA Server and the secondaries ones are located in the COPA Pool. Thus, the Web Server module, as the control layers, is located at COPA Server, having just one of this module per experimentation. Figure 3.1 presents all the modules organization and communication among them. We chose a centralized paradigm because of multiple advantages and limitations of the control framework used to reserve resources in the testbed explained in the following sections.

3.1 Web Server

Web Server is the central entity of COPA Architecture. This module is responsible for many tasks as providing a web interface, a REST API and a database for the testbed experiment. The Web interface is an HTML-based API that the user will be able to access via browser, while the REST API is a JSON-oriented interface focused on providing automated access to the server database. The Web Server was designed in a Model, View and Control (MVC) paradigm where it can logically separates the database management layer, the controlling layer and the method of data visualization layer. This organization enables COPA to provide two types of interface.

The Web interface enables the experimenter to follow the COPA Pools monitoring in real time. Besides that, this tool also allows the configuration of the COPA Pools and

Figure 3.1: COPA software architecture

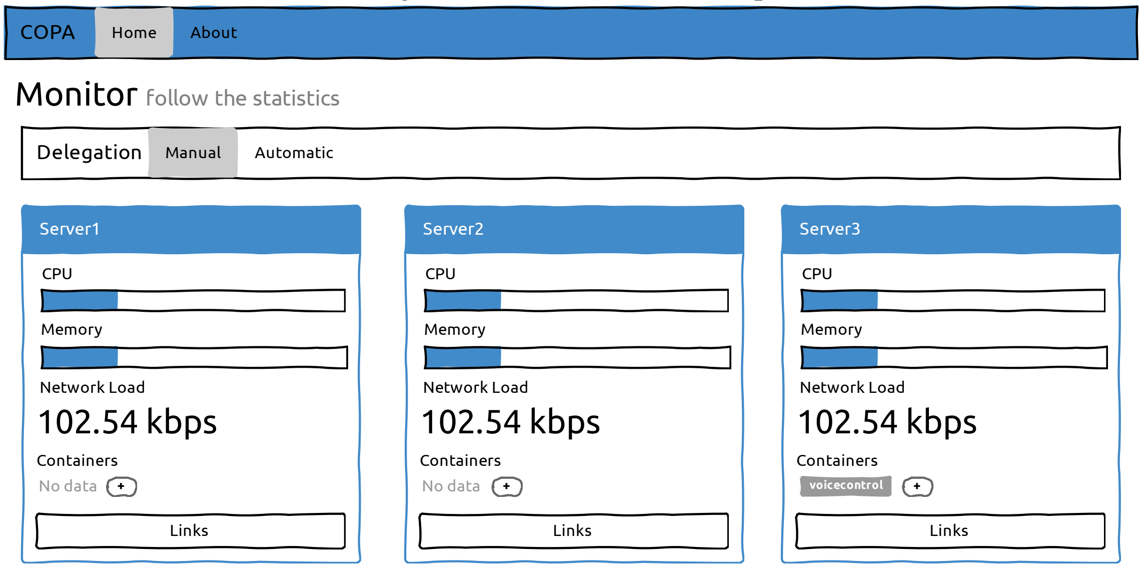


Source: Author

orchestration methods. The REST API provides functions to store, request monitoring data and even container management actions. Besides that, the experimenter could develop a script that uses the REST API to save or gather information from the database. Centralizing data at the Web Server also allows the data consumers applications have a lower response time than requesting each COPA Pool individually to request monitoring data. Having all the data in one place also facilitates future data analysis by the experimenter or the orchestration algorithms.

To provide an easy to use Web interface, we designed a mock-up of COPA's main screen presented in Figure 3.2. The idea was to make available monitoring data in real time for the experimenter and fast orchestration reconfigurability. Therefore, for the orchestration reconfigurability, we designed a delegation bar. In this component, the experimenter could choose between manual delegation or automatic. If the manual option were chosen, the experimenter would have control of the containerized applications. A drag and drop functionality is planned to move containers from one COPA Pool to another manually, providing a user-friendly interface for server migration. If the automatic option were chosen, a drop-down menu that would show up and the experimenter would have to choose among the options presented. In this menu, we would have available the custom orchestration algorithms uploaded by the user and other eventual pre-stored testbed orchestration algorithms. With the automatic delegation activated, the user would be unable to manage the containers manually, and the chosen software intelligence would take control of this task.

Figure 3.2: Dashboard mockup



Source: Author

The end-to-end wireline network monitoring generates many data making its visualization more complex. For the COPA main screen, we planned to keep more simple as possible. Hence, the health status monitoring of COPA Pools was prioritized in this first view. We can see in Figure 3.2 that we have a list of panels components, where each one of the panels represents one COPA Pool. In this panel, we designed to show the resources monitoring of each COPA Pool and the list of containerized applications hosted by them. The first resource data presented is the CPU load that is represented by a percentage bar. Second, we have the memory usage that is represented by the same component that the CPU. Right under, we have the Network Load which is the amount of data passing through the network interfaces of the host. Then, we have the container list which provides the location information of each application and container management as well. Each container in this list has a menu that enables the experimenter to start, stop, freeze, unfreeze and delete a container. Furthermore, there is a form for creating containers which the experimenter can access by clicking on the plus button located inside the container list. To finish, we have the links button which opens a better view of the wireline and wireless end-to-end network measurements.

In this links view, we planned to have the wireline end-to-end network measurements from each COPA Pool to another. The wireline monitoring will be divided into tabs where each one will present information from one end-point COPA Pools. Also, we designed the charts to be updated in real-time as the resource monitoring. Furthermore, we plan to develop a comparative tool to visualize the different links quality. In the same window, the wireless network monitoring will be shown in the case of the COPA Pool have a wireless network interface. This visualization will be organized in tabs too. Each tab will have the information about a connected device.

As we could see, the Web Server module is the base of the architecture. It is responsible for storing data collected by the monitor, provide a graphical user interface for the researcher and provide data and reconfigurability to the orchestrator module. However, we cannot show any data if there is any of it. Therefore, in the next section, we present how we design the monitoring module to provide the data that better represents this network convergent scenario.

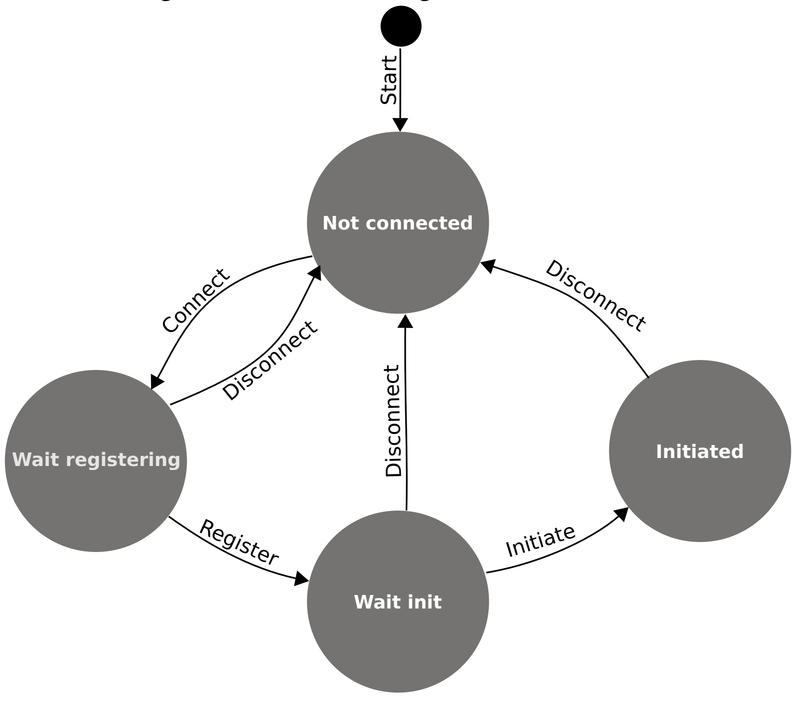
3.2 Monitoring

Some modules had significant influences in the monitoring module design given some characteristics of the testbed environment. The control framework utilized in the FUTEBOL testbed, where we based our software architecture, have some limitations. When provisioning the experimenter slice resources, the control framework does not have some crucial information as what are the IP address of the resources. That influences directly at the Monitor Agent because it needs the information of where it needs to connect to make the end-to-end network quality measurement. Another characteristic of the control framework is the unsynchronized initialization of resources. If we had decentralized monitoring, we would have to utilize or develop a network detection protocol to discover new COPA Pools in the network, causing a development overhead.

Therefore, we propose the monitoring module to be divided into two entities: Monitor Controller and Monitor Agent. The former submodule is a centralized entity for the monitoring module. It can connect to all Monitor Agents through a proprietary API and initiate and synchronize them, while the latter makes the actual network and resources monitoring. Besides that, the Controller can handle the insertion of new COPA Pools by the experimenter. It is made by monitoring the COPA Server database for differences between the data stored in the database itself and what it has stored in the controller data structures.

After defining architectural issues, we elaborated a logical behavior of the Monitor Agent that follows the state machine presented in Figure 3.3. Initially, the Agent is not connected to any Controller and listen to a port waiting for connections. This Agent behavior helps to manage the synchronization of the COPA Pools. After the connection of the Monitor Controller, the Agent waits to receive the information necessary to start the monitoring. Received the data, it waits until all the Agents are in the same state and then receives a command to start the monitoring. When the monitoring is under execution, the agent captures metrics from the wireline network, the resources of the host COPA Pool and, if exists, the wireless network. Finally, at the end of this routine, the Agent sends the monitoring data to the Web Server where it is saved in the database.

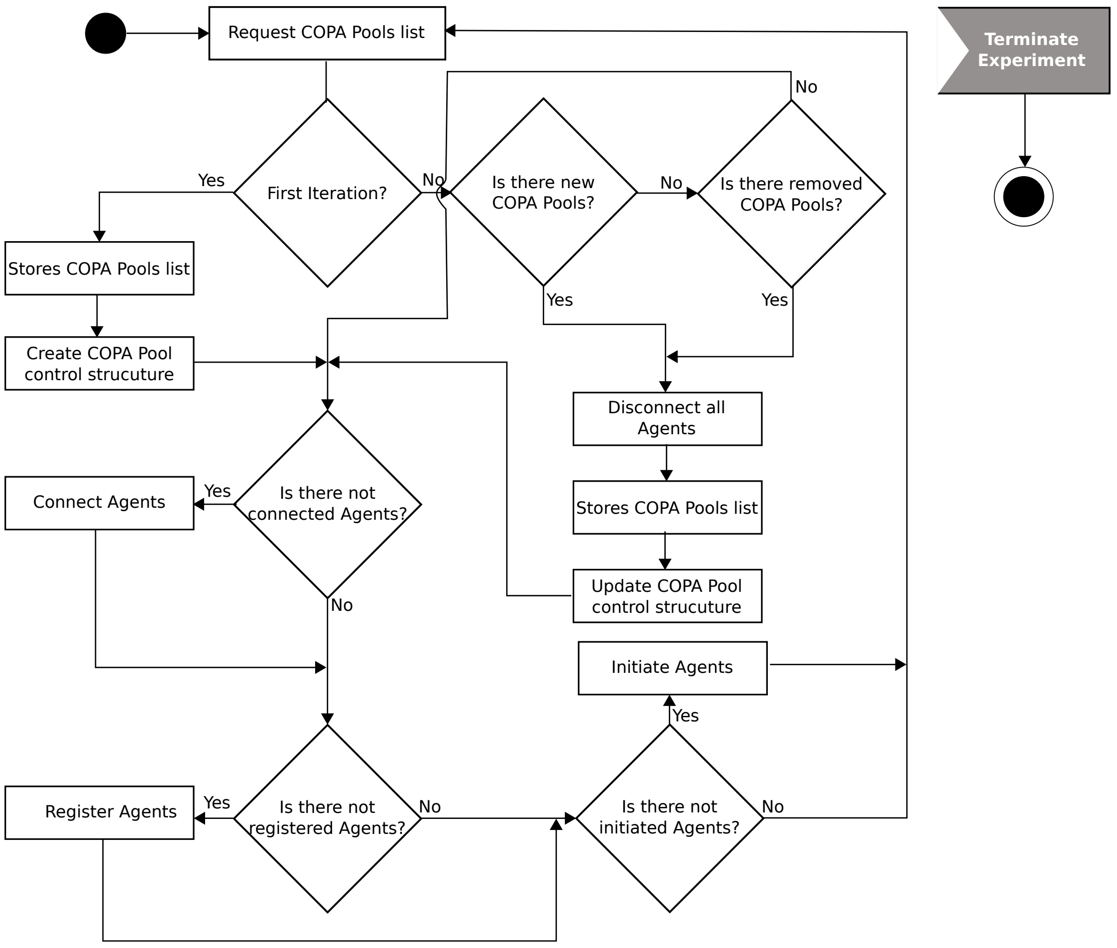
Figure 3.3: Monitor Agent state machine.



Source: Author

The Monitor Controller was planned as presented in the algorithm in Figure 3.4. When initiated, the process accesses the database requesting for the existent COPA Pools. Following, it compares the new information of COPA Pools to the previously collected ones. If it is the first iteration of the algorithm, the Controller only stores the COPA Pools data in a data structure. Next, the controller saves the COPA Pools data in a control structure, where it can make group commands to each Monitor Agent given it state. If there are Agents that are not connected yet, the Controller tries to connect, and them save the current state of this agent given the result of the action. Next, the controller executes the register action distributing data of the connected agents to all agents with the connected state. After the registration of the connected agents, the controller initiates all the registered agents. If there is new COPA Pools, the controller reset all the existent agents by disconnecting to them and redoing all the procedure described above. The same occurs when a COPA Pool is removed from the experiment.

Figure 3.4: Monitor Controller algorithm.



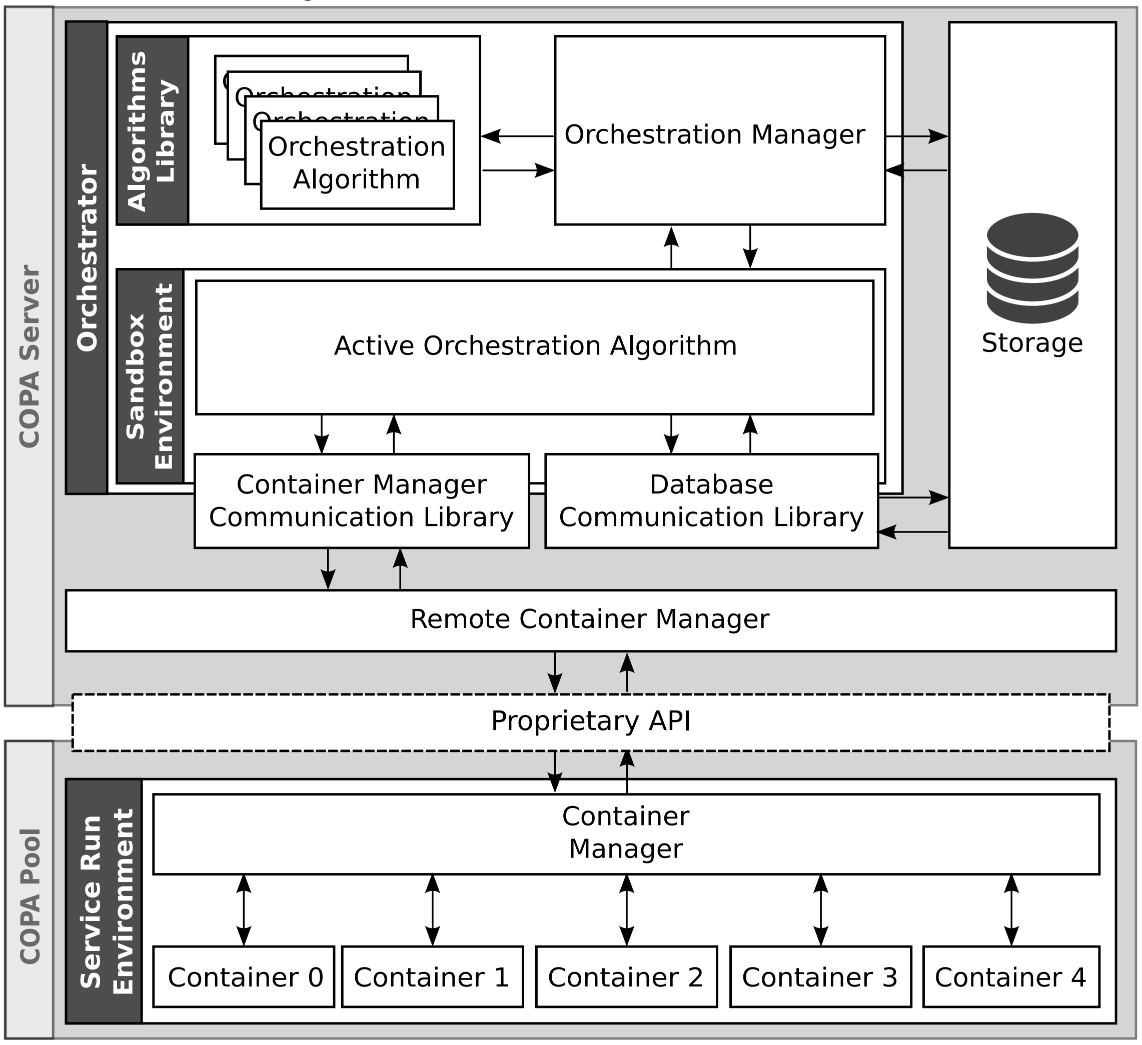
Source: Author

The monitoring module is essential for research and is the focus of this work. Although, there would be no purpose in collecting this information if we could not do anything with it. Therefore, in the next section, we present the orchestrating module.

3.3 Orchestrating

The objective of COPA is not to provide orchestration algorithms to be used in the experimentation. However, to provide a platform where the experimenter can write any service orchestration algorithm and run it without any problem. To achieve this, we present a detailed architecture of the Orchestrating module in Figure 3.5.

Figure 3.5: Detailed Orchestrator Architecture



Source: Author

This module is composed of two major entities: the Orchestrator and the Service Run Environment (SRE). The first is located at COPA Server while the second is placed at COPA Pool. The Orchestrator is divided into three components that provide a generic orchestration algorithm environment. The first component is the Algorithm Library which is responsible for storage the uploaded user algorithms. The second, we have the Orchestration Manager. This component is responsible for monitoring the database for changes in the orchestration configuration and replaces it if necessary. For example, when the user changes the running orchestration algorithm, the Orchestration Manager detects it, consults the Algorithms Library, stops the current active algorithm and executes the selected algorithm.

Third, the Sandbox Environment which provides a secure and isolated environment for the orchestration algorithms to run. This component is essential, because, in COPA, we allow the user to run any code inside COPA. The objective of running user content is for research purposes. Nevertheless, we can not guarantee that the user will not try to find a security breach in the software environment. For that matter, we architecture two Communications Libraries for external operations. One of them is the Container Manager Communication Library. This library provides direct access to the Remote Container Manager which has access to all containers in the COPA Pools. The remote container manager provides management to any containerized service among the COPA Pools. Furthermore, we have the Database Communication Library which enables the Orchestration Algorithm to request monitoring data from Storage. Without this last component, the Orchestration Algorithm would not have any information to base its decision making without this last component.

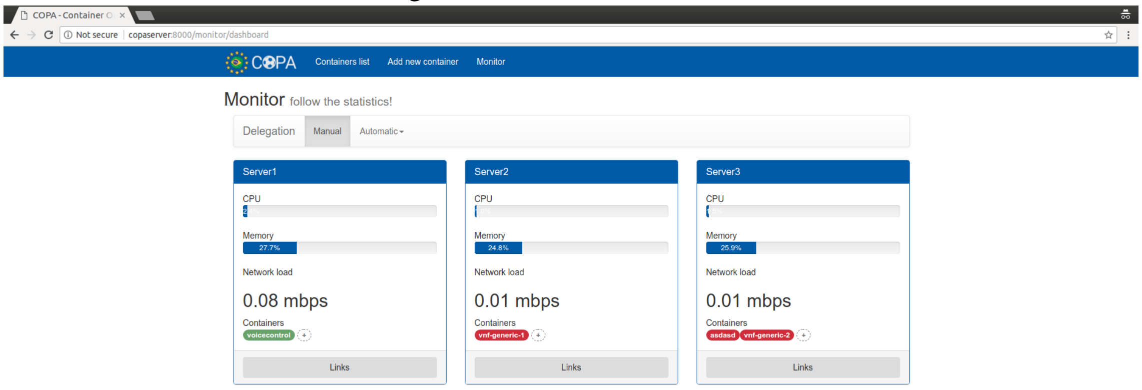
Service Run Environment is a container virtualization technology that provides isolation and management to containerized services. It also provides an open API to connect to others Service Run Environments and be able the exchange information. These connections enable the migration between the COPA Pools and also the Remote Container Manager to manage containers. Inside of SRE, we count with a control layer of the container virtualization technology and the containers itself. This organization enables the experimenter to deploy and manage containerized applications without any problem.

We all know that the theory and practice go hand in hand. Therefore, we presented in this chapter the planning and theory of COPA. We could design a well-defined structure and rules to make the COPA platform possible. In the next chapter, we show how the development was executed and what technologies were used.

4 IMPLEMENTATION

This chapter presents the implementation aspects of the COPA architecture designed in Chapter 3. As we previously stated, the aim of COPA is to allow the monitoring of a wireline/wireless convergent network and management of containerized applications. Considering the challenge inherent to this end, several third-party software were integrated to achieve each of the internal functioning objectives of the architecture. In what follows, we explain our technology decisions and their roles in COPA. In figure 4.1, we present the main screen of COPA.

Figure 4.1: COPA dashboard.



Source: Author

4.1 Technologies

COPA has as its objective to provide an easy-to-use tool for wireline-wireless monitoring of container manager hosts. Hence, it smartly aggregates some technologies to achieve its objectives. In the wireline monitoring, it was chosen One-Way Active Measurement Protocol (OWAMP), an already certificated software to capture one-way-latency and jitter. In the wireless monitoring, we used IW tool [GNU 2017] that is a native Linux tool which provides wireless network metrics for each connected device. The subsections 4.1.1, 4.1.2 and 4.1.3 present the leading technologies used and how they were used in the project.

4.1.1 Django

Django is a Python Web Framework that enabled to implements the web server module, the centralized part of the system architecture. Among many advantages, this framework was chosen because of its fast development and clean, pragmatic design [DJANGO 2018]. This framework is also a Model, View, and Controller (MVC) which is a very well-known organizational method for web servers in general. Therefore, this technology enabled the development of both REST API and the web interface.

To store data, Django provides support to SQLite by default. This database software enables an easy to deploy and portable environment which is very important for running in experimentation. Django helps in the implementation of the web server module by providing different out of the box libraries and structural models. This features helped to speed up the development.

4.1.2 OWAMP

OWAMP is a command line client application, and a policy daemon used to determine one-way latencies and end-to-end jitter measurements between hosts [INTERNET2 2018]. Such a tool allows COPA to extract data about experiments and provide a complete view of testbed network behavior. The version used in this project was the 3.4-10.

OWAMP is a client-server architecture. The server listens to a specific port waiting for a client to initiate the end-to-end network quality measurement. Both client and server were installed in all of the COPA Pools. The server is initiated at boot time of COPA Pool while a COPA Agent executes the client. Each measurement longs at least 10 seconds and returns the maximum, the minimum and the average latency of that period. It is not repeated to say that this software enables a precise wireline monitoring tool and provides two fundamental metrics for this work, latency, and jitter.

4.1.3 LXD

LXD is a next-generation system container manager. Different from others container managers, the objective of LXD is not only provide containerized applications but serve as a light-weight option to Virtual Machines (VM) and provide full servers deployment. COPA is projected for monitoring and orchestrates services, simulating, then, a similar environment of Cloud/Fog computing. Therefore, a key factor in choosing this technology was the implementation of the live-migration method which other containers managers such as Docker has not implemented until this project was written. Furthermore, this technology provides a REST API over a local Unix socket as well as over the network. The REST API enables remote communication that is essential to COPA, so the control layer of the orchestrating module could execute actions from COPA Server to COPA Pool.

All the COPA Pool have an LXD instance to enable the execution of containerized applications. The LXD, in this context, represents the Service Run Environment component of the orchestrating module. Also, COPA Server has an LXD installed in it. However, it does not execute any container in it. The role of LXD installation in COPA Server is providing a container image library, storing pre-made containers images, e.g., ubuntu server. It is important to inform that to run LXD with live-migration, it requires a specific Linux kernel version, LXD, and Checkpoint/Restore In Userspace (CRIU) software: kernel v4.8.0-54, LXD 2.0.9, and CRIU v3.1.

4.2 Coding details

To provide a better understanding of this work, we describe in this section some nuances about the COPA code. We organize this section in four subsections, following the structure of COPA architecture: first, we describe coding of the Web Server; then we explain details regarding the components COPA Controller and COPA Orchestrator of the COPA Server; finally, we discuss coding aspects of the COPA Agent that run in the COPA Pool.

4.2.1 Web Server

For the Web Server component, we use Django framework in the 2.0 version. The Django framework organizes their modules in "apps", and we implement three of them in the web server: containers site, core, and monitor. The first is the default app, which provides files and components that are used by more than one app, e.g., the site template. Second, the core is responsible for providing a REST and HTML-based interface for managing the containers inside the COPA Pools. Finally, the monitor app provides a Create, Read, Update and Delete (CRUD) REST interface for monitoring data gathered by COPA Agents. In the following, we explain the structure of the two important apps: core and

Core app

The core is the container management interface. This app uses the “pylxd” library to provide classes and methods to connect with the local and remotes LXDs in the network. Table 4.1 presents the organization of the Core app. In the left column, we have the HTML-based functions, which are accessible through the browser. While in the right we have the REST API available.

We can see while the HTML-Based have one function per action, the REST API have only one for all. That occurs, because the HTML-Based returns an HTML for each URL, while the REST API have one URL and the actions are sent by HTTP method. For example, if we want to add a new container, we should use the HTTP method POST and the URL that refers to the REST API container function. It is important to highlight that the migrate function, that is present in both user interfaces, depends on secure connectivity among COPA Pools. To migrate a container, the Web Server connects through the pylxd library with the origin LXD and executes a move action to the destination Pool. The remote LXD connection can only occur because of a previous setup of certificates in all COPA Pools LXD.

Monitor app

The monitor app hosts the main COPA interface. Hence, it is one of the most important and the most worked part of the system interface. As the core app, the monitor also exposes a REST API and an HTML-based interface as shown in Table 4.2. One of

Table 4.1: Core app code organization

HTML-Based	REST API
containers_list	container
containers_start	
containers_stop	
containers_delete	
containers_freeze	
containers_new	
containers_migrate	
containers_unfreeze	
containers_info	

Table 4.2: Monitor app code organization

HTML-Based	REST API
dashboard	locus
	dashboard
	dashboardlinks
about	kpilink
	kpiresource
	kpiwireless

the characteristics of this app is the dynamism of its dashboard HTML-based interface. The experimenter can execute actions without changing the webpage. For that matter, we can see that in the HTML-based interface we have only two functions: dashboard and the about page. In turn, the REST API exposes five functions, three for getting and saving network related data in the database (kpilink, kpiresource, and kpiwireless) and two for update data in real time in the dashboard interface (dashboard and dashboardlinks). The former is requested by a javascript loop function to update the dashboard resource monitoring charts. A javascript loop function also requests the latter but in different times.

4.2.2 COPA Controller

The COPA Controller is part of the monitoring module. This component consists in a Python script that runs inside COPA Server and acts in the management of COPA Agents, especially in the initialization process. As COPA Agents run as independent instances of software, when they are started they do not know the existence of their neighbors. In such a context, the Controller is responsible for gather registered Pools from COPA Server and update Agents with neighborhood information. This feature was implemented using a socket python library, and for data transferring we used JSON data format.

In many cases, Pools do not initiate simultaneously. Therefore, another function of the COPA Controller is to synchronize the start of the whole monitoring system, registering each COPA Agent and starting them all together. To do this, we implemented a control class called Monitor which follows the Agent's state machine rules described in the Solution chapter in Figure 3.3. This class can connect to remote COPA Agents and send messages, changing their states automatically.

Another important data structure is the Controller class. This class manages all the instances of the Monitor class, enabling the mass executing of actions as the connect action. For example, when the connectMonitors function from the Controller class runs, the COPA Controller checks all the Monitors instances if any of them have a "not connected" status. If there is, the COPA Controller tries to connect and, if successful, changes to Monitor status to connected. The implementation of this two classes plus the flow of execution as seen in the figure 3.4 of the Solution chapter, enables the Controller to perform all the actions of the algorithm.

4.2.3 COPA Orchestrator

Similarly to COPA Controller, the COPA Orchestrator is a python script running in the background on the COPA Server. Unfortunately, neither the Algorithms Library neither the Orchestration Manager components were implemented. As a proof of concept and for the sake of simplicity, we employ a generic class to implement methods that gather monitoring information from the database. Seven of the eight methods are implemented to provide abstracted data to the orchestration algorithm. Among them, we have two-time interval methods, get and set. The time interval is the referent of how old information the experimenter wants to use to feed the algorithm. For example, the default time interval is two minutes. Then, our algorithm is going to get information about the network links of the last 2 minutes of monitoring. The remaining of this methods implement the Database Communication Library (DCL) of the Orchestrator component.

Representing the DCL, we have a method to get the existing COPA Pools in the experiment. Furthermore, we have implemented the method to get the containers inside this pool, together with the monitoring data of the containers itself provided by LXD platform. Moreover, we have the methods getResources, getLinks, and getWireless which receives the COPA Pool name as a parameter and returns the resources, wireline network monitoring and wireless network monitoring data, respectively. Last but not the least we have the decision algorithm. However, as said before, this part is only an abstract method, giving the opportunity for the experimenter to extend this class and add its code. The downside of this approach is that the experimenter has to extend the Orchestration class adding its python orchestration algorithm. After that, the experimenter has to run the process in the COPA Server manually.

Furthermore, the container manager calls for migrating and creating containers must be made through the implemented Core API which represents the Container Manager Communication Library. The API is well documented in COPA repository [COPA 2017].

4.2.4 COPA Agent

The COPA Agent is also a python script that runs in the background. The module starts at the boot of COPA Pool together with the OWAMP server. When started up, this module includes three important classes: two for passive monitoring and one for active network monitoring.

The first passive monitoring class is for capturing wireless data from the connected devices if a WiFi network is up. To capture this information, the WLInfo class, as it is called, make use of the IW tool. This software is native to Linux, and it gives information per connected device about the signal strength, TX and RX bytes transmitted and TX bytes failed. Moreover, WLInfo parses the response of IW tool and turns into a JSON friendly format. The second class is the Throughput, which is a thread of the Agent process. The role of this class is to calculate the instantaneous network throughput of all the COPA Pool network interfaces. With this information, we can measure the Network Load of the server. Different from WLInfo, Throughput uses a psutil library to gather network data. The Throughput thread is consulted from time to time by the main thread of COPA Agent and sent to COPA Server. In the active monitoring, we have the Owping class. This component manages the use of OWAMP client which calculates the end-to-end one-way latency and the jitter between two COPA Pools.

For control of the monitoring agent, we have two classes: StateMachine and LoopThread. The former works similarly to the state machine presented in the monitoring section of the solution chapter. The difference is that this class, in addition to storing the status of this program, and the rules of changing one state to another, executes some deployment actions. For example, the register method, which executes when each COPA

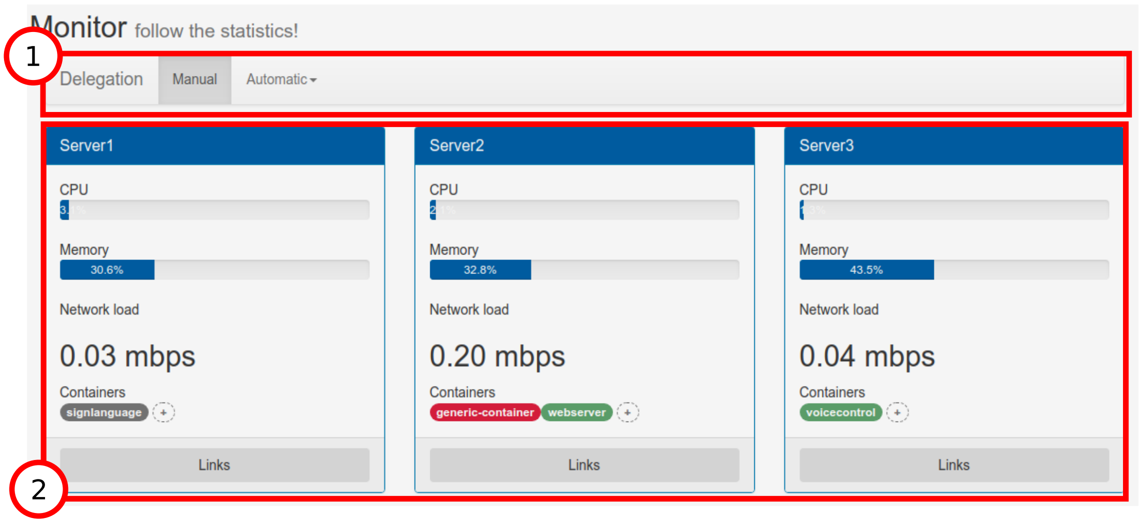
Monitor connects, execute the main control class, LoopThread. LoopThread is a thread of the monitoring script and is responsible for executing the monitoring actions and sending data to the Web server, located at COPA Server. This class contains a loop controlled by a flag. This flag is turned on and off according to the registration and disconnection, respectively, of the COPA Monitor.

4.3 Graphical Interface

The monitor graphical interface of an experiment must be simple and easy to use. We based our interface development in the mockup presented in Chapter 3, Figure 3.2. Given our expertise with the responsive web framework, we used the Bootstrap v2.3.2 for the graphical interface implementation.

The home page of COPA is the dashboard as shown in Figure 4.1. This page is divided into two areas depicted in Figure 4.2. In the top area, a menu bar exhibits orchestration options. By default, the delegation method is manual, and this means the application containers will not migrate to any other Pool unless moves it manually. The other options are in the sub-menu Automatic. The provided orchestration algorithms are just threshold algorithms for CPU, Latency or both together. This automatic options are just proof of concept and must be enhanced.

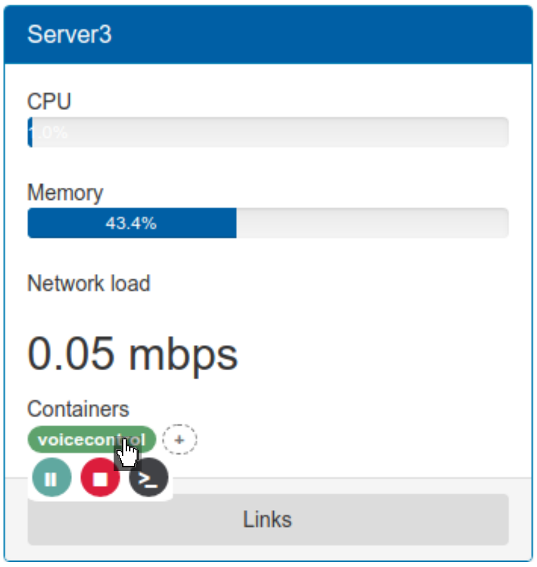
Figure 4.2: Monitor state machine.



Source: Author

In the second area, the web interface presents monitoring information and options. In this division, we have a panel for each COPA Pool in the experiment. Where, charts exhibit resource monitoring data from the COPA Pool, e.g., the percentage of CPU usage, Memory usage and Network Load in megabits per second. Right below, the containers application are listed. Given the implemented functions in Core app located inside the Web server, the web interface can send actions to containers by just making ajax requests to the available API. The actions are available in a submenu that appears when the experimenter hover the mouse through the container name shown in Figure 4.3.

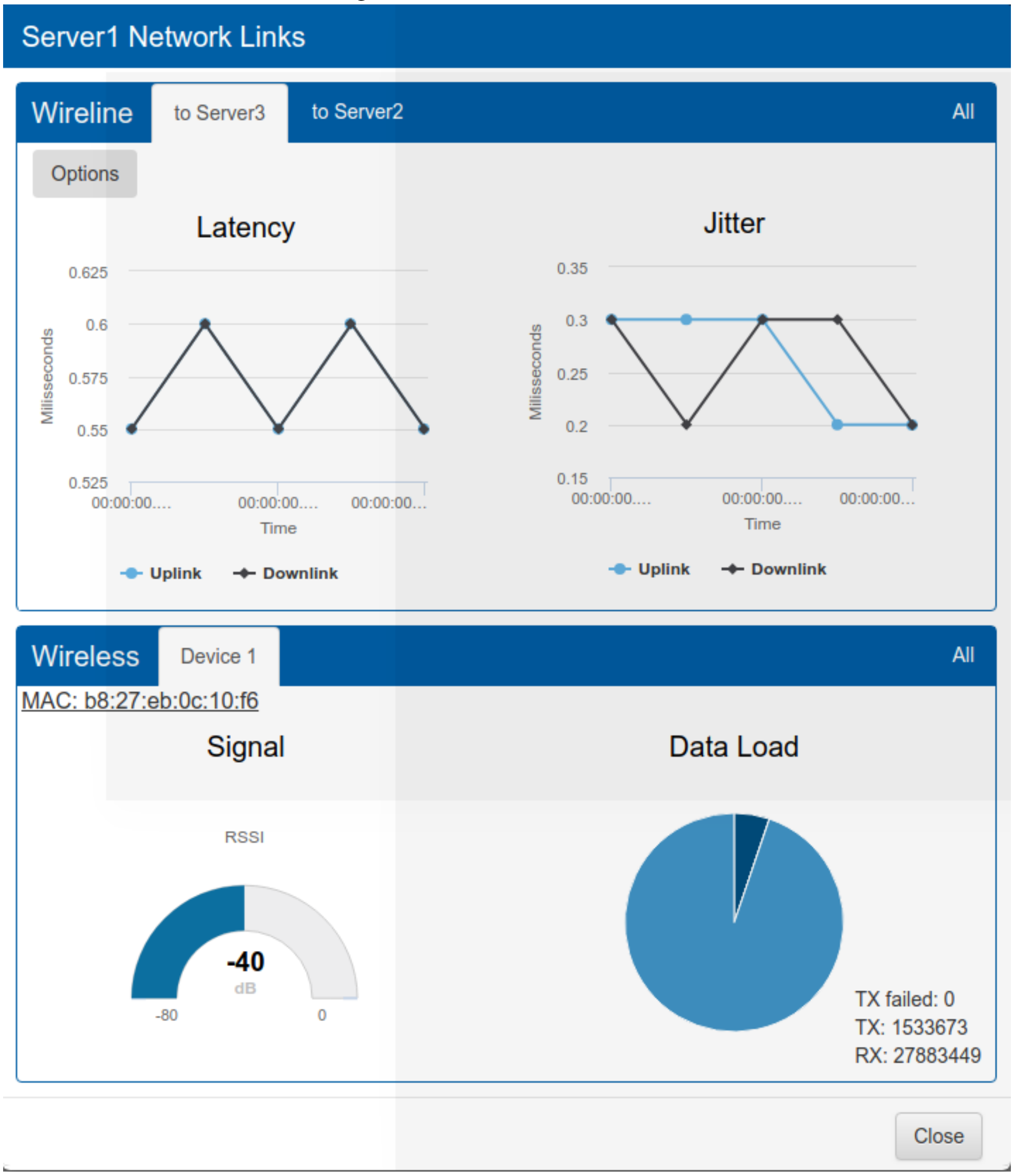
Figure 4.3: Containers actions.



Source: Author

Next, there is the network monitoring modal window in Figure 4.4. In this window, there are two sections. First, we have the wireline monitoring which contains metrics about the link between COPA Pools. The first chart of this section shows the variation of the latency over time. In turn, the second chart represents the variation of the jitter over time. To implement this charts, we use the Highcharts library for chart drawing [HIGHSOFT 2018]. Above this two charts tabs allows the experimenter can navigate through Pools. The following interface section provides wireless monitoring of all connected devices in the WiFi network provided by the Pool. This interface section only is showed up when the Pool has an active wireless interface. The first chart of this section shows the device signal strength. The second chart is about the connected device data load, which provide information about transmission, received bytes and failed transmitted bytes.

Figure 4.4: Containers actions.



MAC: b8:27:eb:0c:10:f6

TX failed: 0
TX: 1533673
RX: 27883449

Source: Author

5 ADAPTIVE CLOUD/FOG FOR IOT

In this use case, we present how COPA can be used in a real wireline/wireless convergent network testbed experiment scenario. The primary research areas are Fog and Cloud computing, wireline/wireless convergence and IoT. Also, we demonstrate how COPA can help tools the research in future networks as a wireline-wireless infrastructure to provide high-quality IoT communication, involving devices ranging from low complexity sensors and actuators (luminosity, and smart light bulbs) to more advanced ones (multimedia sensors and smart glasses). The first section of this chapter validates the idea of this use case. Next, we set up a scenario where the experimentation occurs. Then, we explain how the experiment is deployed inside the FUTEBOL testbed. Finally, we show the monitoring of the metrics collected by COPA and how the experimenter follows the status of experimentation.

5.1 Use case motivation

Using Fog/Cloud in this use case, we could process the workload of different IoT applications, like smart energy metering, security systems, water toxicity sensors, air pollution detectors. These different types of applications impact differently from the wireline and the wireless-network domains. The integration between wireless and wireline networks in this experiment comes from the fact that the wireline network can be used as an efficient transport mechanism for IoT wireless data to be processed in the Cloud, due to the high capacity and low latency provided by wireline transmission technologies as an optical fiber. The experiment demonstrates the importance of resource and network monitoring of the applications by providing information to the interplay between Fog and Cloud computing. This demonstration aims to validate the convergent network monitoring and container management as part of future network infrastructures for IoT deployments. The converged wireline-wireless network monitoring is essential to address the requirements of IoT data transmission and analytics and to offer the highest flexibility and reactivity to changing load demands.

Fog computing enables the reduction of network latency for IoT applications and the decreasing of the total data traffic sent to the core of the network. Fog improves network connectivity because it is located closer to the user. Such performance enhancement can be measured regarding network capacity and latency gains. Also, it is possible to implement containers orchestration intelligence to analyze network and computer resources data and decide whenever the Fog is required. This arrangement enables the dynamic allocation of the locus of computing among the databases according to wireline-wireless network and container host conditions, tailored to latency-constrained IoT applications.

In a previous study, it was demonstrated that a smart light controlled by voice commands could validate the usage of Cloud and Fog [Silva B. Abreu 2017]. The authors conclude in this study that from the processing time point of view, for processing complex voice commands, we could not see a big difference in latency between the Cloud, Fog, or Local. As for simple commands, the processing time is smaller in the tiers closer to the mobile device (local machine or fogs). Although the Cloud could have more processing capabilities, it results in longer latency when there is competition for network resources. Moreover, last, from the response time point of view, a combination of network latency and processing power is assessed showing that Cloud/Fog interplay could improve the application performance.

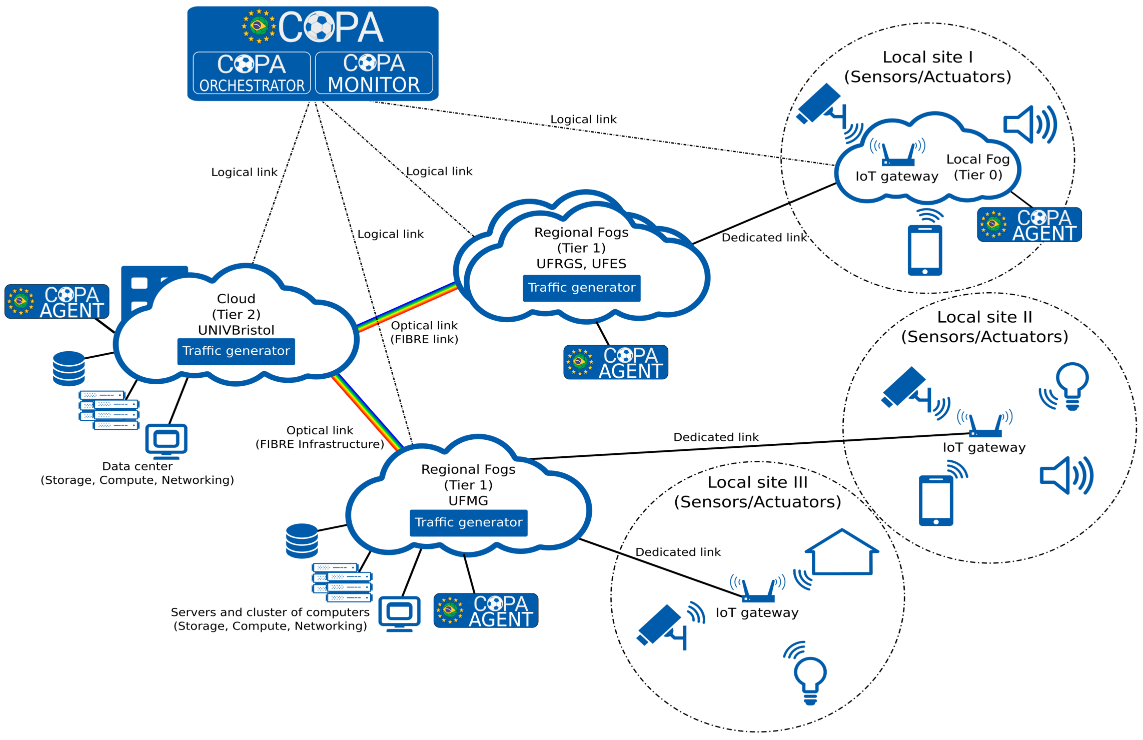
We will use the same IoT application for experimentation. The main objective of this use case is to provide a testbed solution for experimentation which can provide data about a wireline-wireless convergent network and help to assess the interplay between Fog and Cloud to enhance the performance of IoT applications. COPA enables a trade-off between Fog and Cloud when applied to IoT environments while considering different conditions of resource and network monitoring.

5.2 Showcase set up

To set up an experimental scenario, we relied on the FUTEBOL testbed. Therefore, in this use case, we have designed an experimental architecture given the available resources in the FUTEBOL testbed. As in figure 5.1, we have three distinct tiers of processing. The first one is a Local Fog (Local) closer to the application client. Next, with more of processing power than Local and further away from the user, we have the Regional Fog (Fog). Last, we have the Cloud computing datacenter (Cloud) with more computing power, but farther from the client application than Fog. In our scenario, the Local tier is responsible for enabling the wireless network connectivity, turning into a smart wireless gateway. Given the location of the IoT devices, the Federal University of Rio Grande do Sul (UFRGS) was chosen to be our Local tier. The location decision enables the secure configuration and test of the application client. Federal University of

Minas Gerais (UFMG), then, was utilized as a Fog tier. Finally, we used resources at the University of Bristol (UNIVBris) as a Cloud computing given the high latency characteristic.

Figure 5.1: Experiment architecture

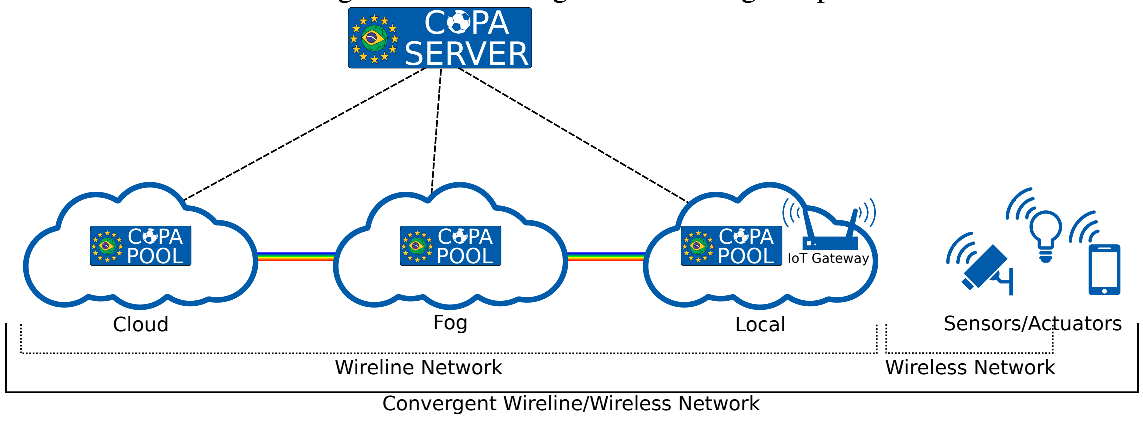

COPA
COPA ORCHESTRATOR
COPA MONITOR
Logical link
Logical link
Logical link
Logical link
Local site I (Sensors/Actuators)
IoT gateway
Local Fog (Tier 0)
COPA AGENT
Dedicated link
Regional Fogs (Tier 1) UFRGS, UFES
Traffic generator
Cloud (Tier 2) UNIVBristol
Traffic generator
Optical link (FIBRE link)
Optical link (FIBRE Infrastructure)
Data center (Storage, Compute, Networking)
Local site II (Sensors/Actuators)
IoT gateway
Dedicated link
Regional Fogs (Tier 1) UFMG
Local site III (Sensors/Actuators)
Servers and cluster of computers (Storage, Compute, Networking)

Source: Author

In figure 5.2, we present the simplified architecture described above. In each tier, we have installed a Virtual Machine (VM) with a COPA Pool enabling the remote processing of the application server. The COPA Server was installed in the Cloud tier, given the higher processing power comparing to the others tiers. Connected in the Local wireless network, we have the IoT application. We have built a multi-colored smart light system in which a person can control the lights through voice commands. In particular, the lights can be controlled on a course three levels: turn on/off, increase/decrease brightness, and change color. This system is a client-server architecture and spans a wireless and a wireline network. The client is connected to a wireless network while the server, where the voice recognition process occurs, stays a remote locus of computing which operates across the wireline network. In this case, processing can occur in any of the presented COPA Pools.

The smart light controlled by voice was created, at the client-side, with a Raspberry Pi 3, an Arduino Uno, an infrared controlled multi-colored light bulb, a protoboard, an infrared emitter and a microphone. In the Raspberry Pi, we used pyaudio library to

Figure 5.2: Convergent monitoring setup



Source: Author

capture the audio and send to the remote processing server. Due to some limitations, Raspberry Pi cannot use the infrared emitter without alterations in the kernel. That occurs because the process that sends data to the infrared emitter is interrupted several times by the CPU process scheduler, making it impossible to send the infrared at the right frequency. Therefore, to execute commands to the smart light, we connected the Arduino with the infrared emitter to the Raspberry. So, the Raspberry can send commands through USB to the Arduino and, then, executes commands to the multi-colored light bulb. In the server-side, we create an LXD container with a Python 3 script utilizing pocket sphinx voice recognition library. This script awaits the client connection, receives an audio file, tries to recognize the command and then send a text command back to the IoT device, which executes the action to the smart light system.

This showcase scenario tries to simulate a Cloud/Fog computing interplay over a wireline/wireless network convergent infrastructure for an IoT application. Then, we will be able, with the usage of COPA, to research about how the degradation of network and processing quality burden the IoT application experience in future networks. Note that, in the future, COPA can be connected to any other application enabling the data collecting of different classes of experimental research.

5.3 Experiment deployment

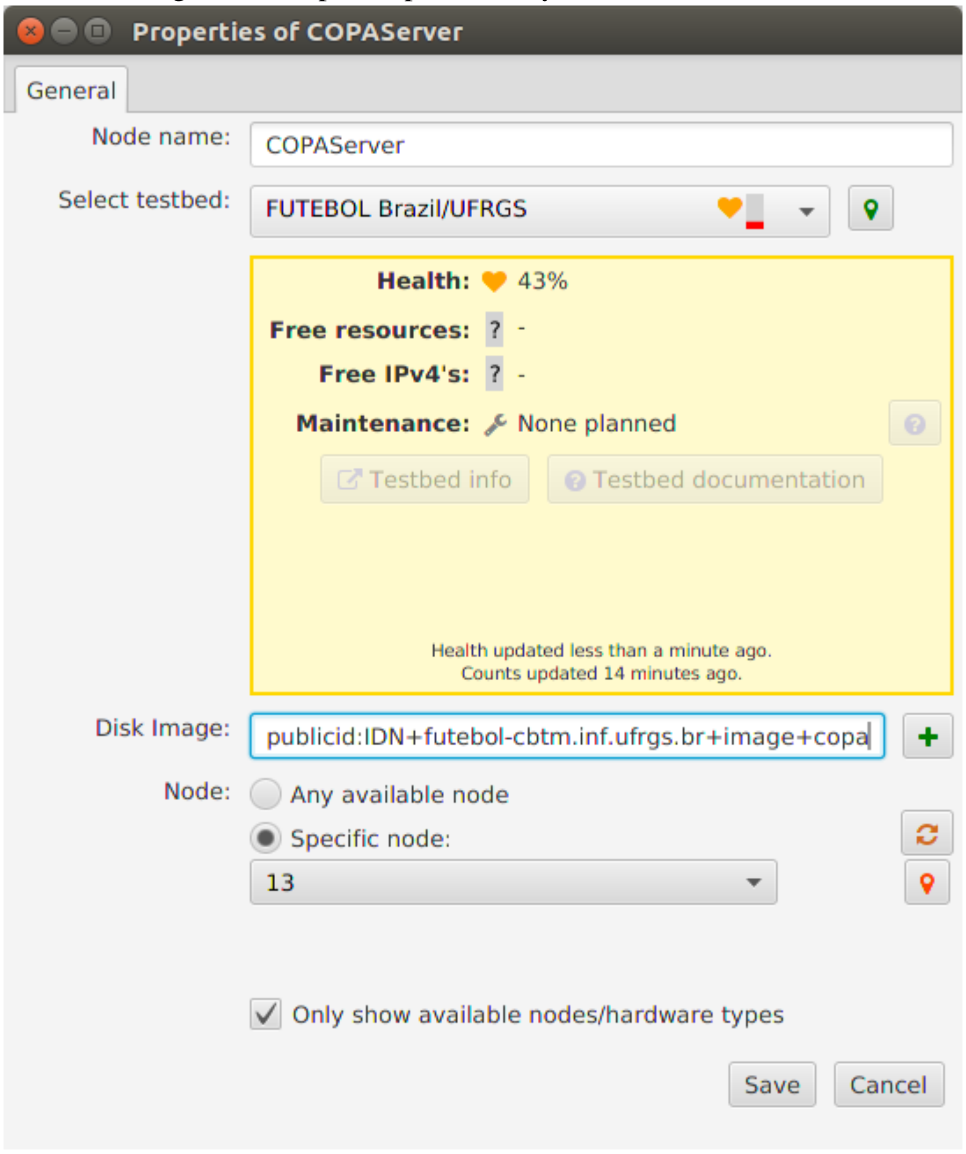
In this section, we present how we deployed this experiment with the smart light application in the FUTEBOL infrastructure. Also, we explain how the metrics collected can be interpreted and how research can be done using COPA as experiment monitoring platform. To initiate our tests we need to request an experiment slice in the testbed. An experiment slice is a group of testbed resources that are isolated physically or logically from the others experimenters. This isolation provides an environment without external influences, making the experimentation more precise and trustworthy. To request a slice, we need a federative tool to intermediate the allocation of resources between the experimenter and the testbed. This tool is capable of making the user authentication and communicates with the testbed control framework. After the authentication, the federative tool provides a list of resources of the federated testbeds. Therefore, the experimenter can choose a set of resources to run his experimentation. In our case, we used jFed, a Java-based graphical user interface for testbed specification and reservation of an experimentation slice. In each chosen resource, we need to choose the testbed where the experimentation will occur, the type of resource, and, in the case of Virtual Machines, the image that we will use. This options are presented in figure 5.3.

After configuring the scenario, we need to allocate the resources and wait for the testbed control framework to set it up. As seen in figure 5.4, we allocated the following resources successfully: one COPA Server, two COPA Pools (one at UNIVBris, one at UFMG), one COPA Pool with the wireless interface at UFRGS and one Raspberry Pi also at UFRGS.

Remote login into the experimenter resources is a must in every testbed. FUTEBOL enables this feature through jFed by creating an SSH tunnel between the resources and the experimenter. Utilizing from this feature, we enable the connection with the COPA Server web interface via a browser. We redirect the port 8000 from the web server located at COPA Server's VM to the experimenter's localhost. Opening the COPA web interface, we can already follow the COPA Pools resource monitoring. The dashboard presents the three allocated COPA Pools, and its status is updated in real-time. After checking the locus of computing functioning, we can confirm the IoT device connectivity by accessing the Local COPA Pool wireless network monitoring.

Afterward, in the experimentation, we need to configure the containerized remote processing server. COPA enables an interface for creating the container with a generic Ubuntu 16.04 image, but for configuration, it is necessary to login into COPA Pool through SSH and configures the container utilizing LXD tools. After creating the container, we are able to monitor and manage the container via COPA interface. Utilizing the manual delegation feature, the experimenter can relocate the containerized application where is better to him, or even stop and delete the container. In our case, we created a container in the Local tier, closer to the user.

Figure 5.3: Options provided by JFed federation tool



Properties of COPAServer

General

Node name: COPAServer

Select testbed: FUTEBOL Brazil/UFRGS

Health: 43%

Free resources: ? -

Free IPv4's: ? -

Maintenance: None planned

Testbed info

Testbed documentation

Health updated less than a minute ago.
Counts updated 14 minutes ago.

Disk Image: publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+copa

Node: Any available node [✓ Specific node:] 13

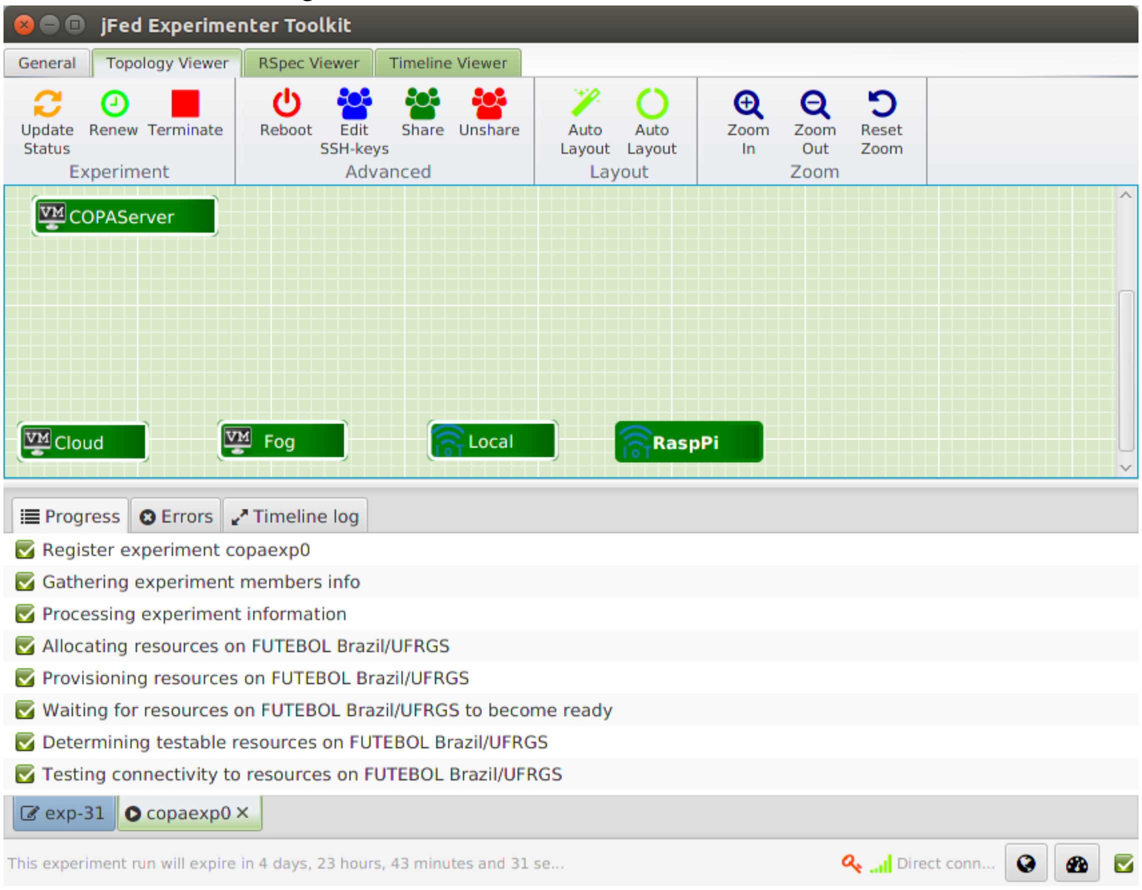
[✓] Only show available nodes/hardware types

Save Cancel

Source: Author

After installing and configuring the application server in the container, we connect the IoT device to the server by executing a script and passing the server IP address as a parameter. The voice recognition application needs a microphone so it can capture the audio. In a testbed, the microphone usage is not applicable since the user is using it remotely. Hence, we developed a script to simulate a microphone into the Raspberry Pi. This script captures the microphone from the experimenter computer and creates a tunnel over the Internet and executes inside the testbed's Raspberry Pi as a virtual microphone, enabling the capture of voice commands. Also, we could create a script to send the pre-recorded audios and, then, automate the experiment. Besides that, we have the smart light available inside the testbed in which the experimenter could not see its status. We could make that the smart light be available only in text-mode. Nevertheless, we create a

Figure 5.4: Allocated COPA resources in JFed



✓ Register experiment copaexp0
✓ Gathering experiment members info
✓ Processing experiment information
✓ Allocating resources on FUTEBOL Brazil/UFRGS
✓ Provisioning resources on FUTEBOL Brazil/UFRGS
✓ Waiting for resources on FUTEBOL Brazil/UFRGS to become ready
✓ Determining testable resources on FUTEBOL Brazil/UFRGS
✓ Testing connectivity to resources on FUTEBOL Brazil/UFRGS

This experiment run will expire in 4 days, 23 hours, 43 minutes and 31 se...

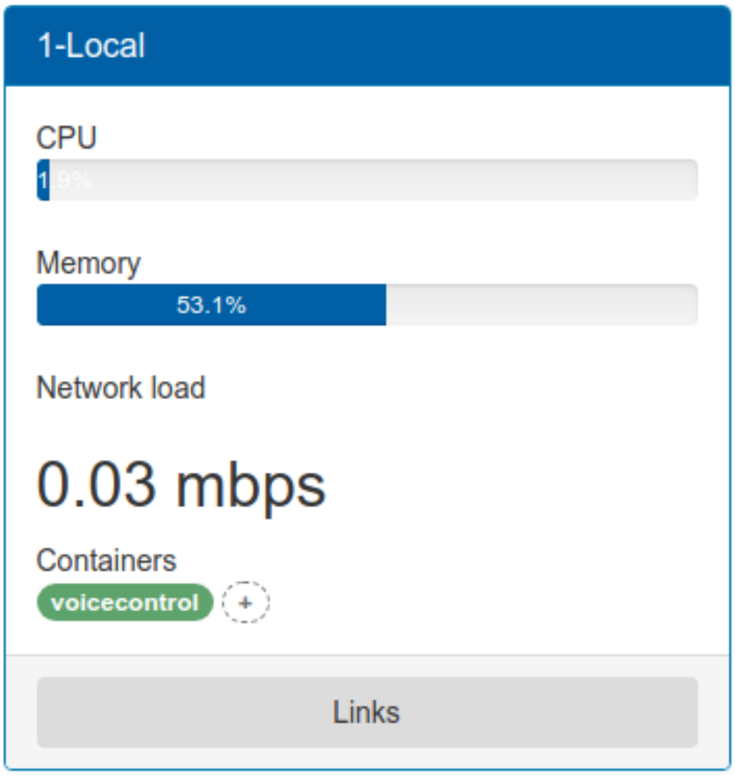
Source: Author

webcam server in the testbed. This webcam server provides images from the smart light system, making possible the experimenter to check into his tests. The webcam server is accessible via SSH port forwarding as the COPA web server. All the resources are installed and ready for experimentation. Now, we can send voice commands to the smart light and see how the IoT application behavior by being executed in three different tiers of computing.

5.4 Experiment visualization

In the COPA dashboard, it is possible to monitor the cloud, fog, and the local computer resources. The experimenter can quickly detect a processing overload or even if the host lacks memory to execute specific computing process. In our experiment, we can use this tool to monitor a simulated overload situation and then evaluates the application performance. The computer resources monitoring of this use case is presented in figure 5.5.

Figure 5.5: Monitoring of voice recognition host and container

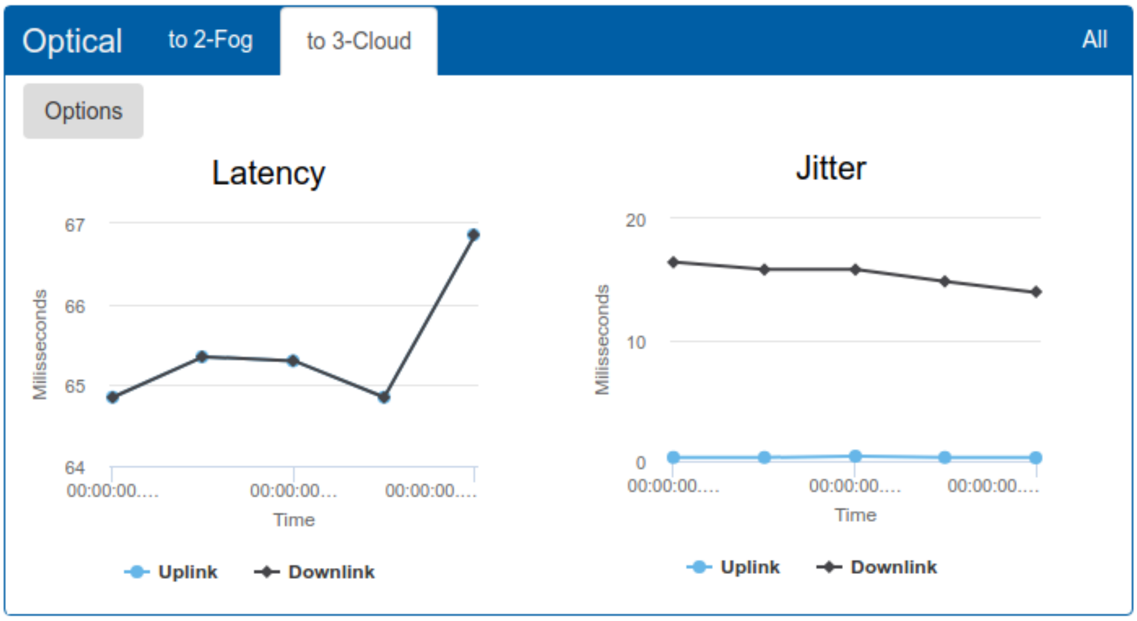


Source: Author

It is important to notice that the convergent monitoring is essential in this experiment to identify the causes of performance issues and decide if it is worth to migrate the processing or not. For example, if the performance degradation of the IoT application is caused by only the wireless link, moving the locus of processing from a Cloud to a Fog at the network edge will not work around the problem. Otherwise, if the locus of processing is facing processing overload issues in a Fog, the migration to a less overloaded Cloud can improve the IoT service. COPA monitors all end-to-end links among the COPA Pools. However, for this use case, the end-to-end link monitoring between the Local and Cloud, and the link monitoring between the Local and Fog is the most important one. That occurs because the IoT device is connected to a wireless network provided by the Local tier, making it the convergent point of the wireless/wireline network. We can see the monitoring between Local and Cloud in figure 5.6. The one-way latency collected shows around 60ms for uplink and downlink, while the jitter for downlink is compromised and for uplink runs smoothly. The jitter downlink and uplink difference could be occurred because of some software background download during the experimentation.

The wireless monitoring of the connected Raspberry Pi is also demonstrated in the figure 5.7. First, we can analyze the first chart of this view and verify that the IoT device is

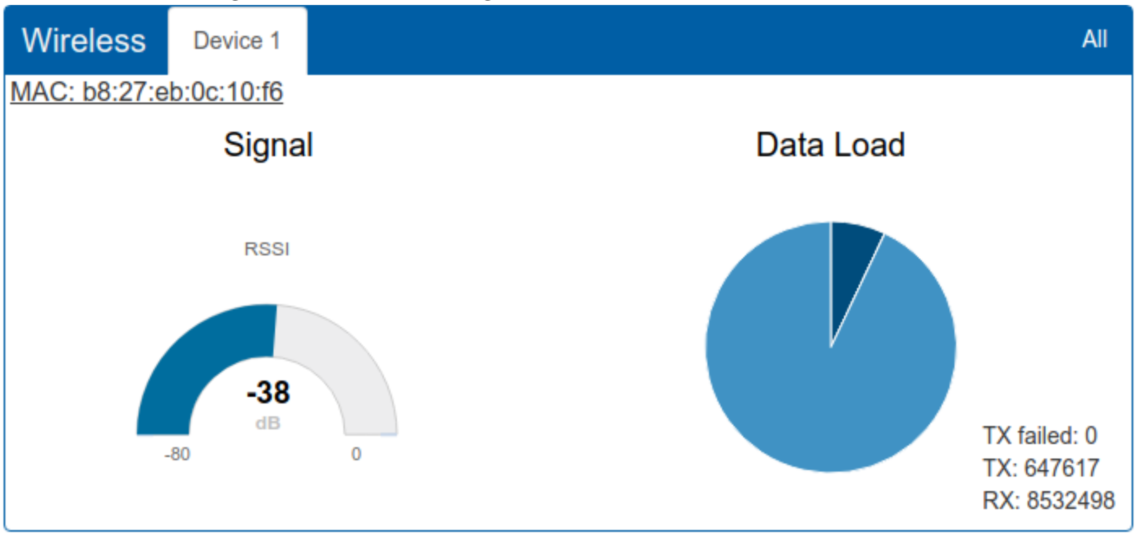
Figure 5.6: Monitoring of end-to-end link between Local and Cloud tier



Source: Author

very near to the wireless access point. While in the second chart, shows how many bytes were received and transmitted to this device from the access point. We can conclude, analyzing this chart, that the Raspberry uploaded many more bytes than downloaded. Moreover, to finish, with zero transmitted bytes failed, we can check the quality of the wireless connection.

Figure 5.7: Monitoring of IoT device wireless connection

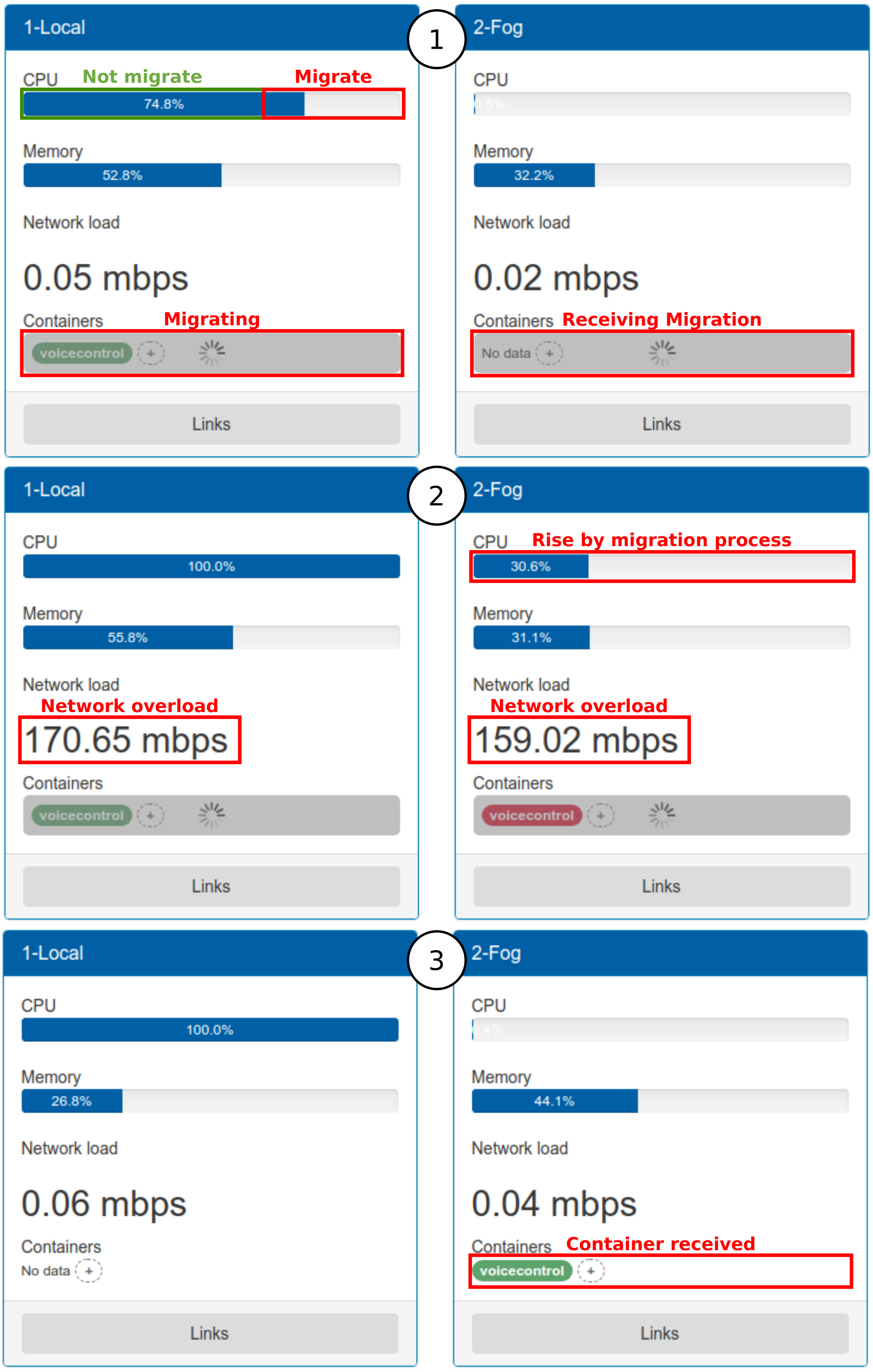


TX failed: 0
TX: 647617
RX: 8532498

Source: Author

To demonstrate the usage of orchestration algorithms, we created a simple threshold algorithm. This algorithm shows how COPA automatic migration can help in the research of orchestration algorithms in future networks too. Our threshold migration solution has as its limit value of 60% of CPU load. Therefore, when the containerized application host reaches a value higher than 60%, the service will be migrated to another locus of computing, as can be seen in the sequence of images in figure 5.8. In the left column, we can see the Local tier with the container, while in the right we present the migration destination, the Fog. In stage 1 of the sequence, the automatic migration is triggered by the CPU overload. In this first stage, we present the areas of CPU bar where the algorithm decides to migrate the container. Also, we can see a grey layer with a loading icon over the container, demonstrating the occurrence of migration. The overload of the Local tier is very common since it lacks the processing power and could easily be overloaded. In stage 2, we detect a network overload because of the transference of the application container. Also, the CPU load of the Fog rises considerably because of the migration. In stage 3, the migration ends successfully. To finish, COPA provides experimentation data to the experimenter through the monitor REST API. The data is available with JSON format and can be easily converted to others format as CSV to create it owns charts in another tool.

Figure 5.8: Automatic migration stages sequence



1-Local

CPU 74.8%

Memory 52.8%

Network load

0.05 mbps

Containers

voicecontrol

Links

2-Fog

CPU

Memory 32.2%

Network load

0.02 mbps

Containers

No data

Links

1-Local

CPU 100.0%

Memory 55.8%

Network load

170.65 mbps

Containers

voicecontrol

Links

2-Fog

CPU 30.6%

Memory 31.1%

Network load

159.02 mbps

Containers

voicecontrol

Links

1-Local

CPU 100.0%

Memory 26.8%

Network load

0.06 mbps

Containers

No data

Links

2-Fog

CPU

Memory 44.1%

Network load

0.04 mbps

Containers

voicecontrol

Links

Source: Author

6 CONCLUSION AND FUTURE WORK

In this work, it was designed a wireless/wireline convergence monitoring and container manager architecture called COPA. This architecture can deploy and migrate containerized applications among several computing levels such as Cloud, Fog computing or even at the network edge. These applications enabled the remote processing for several IoT devices. Also, we researched and developed monitoring and experimenter-level orchestration features for testbed experimentation in future networks. The monitor collects data from the wireless/wireline networks and container host computing resources. COPA enables the usage of this monitoring data in the orchestration module for automatic to manage the experimenter containerized applications. Also, the experimenter could manually migrate the processing application to any of the COPA Pools available through a friendly user interface. Doing it, the researcher could evaluate the deployment of computer routine at different levels of remote computing. Our use case demonstrated how an experiment is executed in a testbed using COPA and how we could analyze the collected metrics. However, several improvements could be made.

Some components of COPA could not be developed in time for this work. These components are from the orchestrator module and should be developed to follow the project planning. The monitor module was developed with completeness but can be improved. External factors should not influence testbeds scenarios. To influence less in the network traffic, the COPA Agent should utilize not only an active monitoring method but a mixed active-passive monitoring method. So, when there is traffic passing through the network interfaces of COPA Pools, would not be necessary to insert more data into the network links. This feature would decrease the influence of the monitor in the user experiment. Also, the addition of network traffic concurrence tool is beneficial for the best representation of experimental scenarios. That is because testbeds most of the times cannot represent a real-world scenario because it lacks physical resources. This tool could simulate network traffic patterns of several IoT applications enabling the research of network concurrence classes in the Internet of Things.

In the Orchestration module, more types of container technologies could be integrated such as Docker, the most famous one of them. This expansion would bring flexibility to COPA platform, enabling the experimenter to bring in the testbed out of the box containerized solutions, not being necessary to adapt to a specific technology. Sometimes, when working with migration algorithms, we wish could classify the applications services, so not all of them follow the same rules. That said, in this module, could be developed more detailed configuration options such create classes of containers and assign them specific rules. This improvement would make possible the refinement of orchestration methods. COPA Orchestrator makes use of live-migration aiming to improve the quality of the service. However, this migration sometimes burdens the network because of the size of the containerized server. So, another feature to think about is to diversify the methods of changing the processing locus of the remote application. COPA could provide not only migration of the service but create another instance of the application server and distribute the network load between them, increasing the research options in the testbeds.

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

REFERENCES

ARMBRUST I. STOICA, M. Z. M. A view of cloud computing. **Communications of the ACM**, ACM, v. 53, p. 50, 2010.

BIBANI, O. et al. A demo of a paas for iot applications provisioning in hybrid cloud/fog environment. In: IEEE. **Local and Metropolitan Area Networks (LANMAN), 2016 IEEE International Symposium on**. [S.l.], 2016. p. 1–2.

BONOMI R. MILITO, J. Z. F. Fog computing and its role in the internet of things. **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**, ACM, p. 13–16, 2012.

CHECKO H. CHRISTIANSEN, Y. Y. A. Cloud ran for mobile networks - a technology overview. **IEEE Communications Surveys and Tutorials**, IEEE, v. 17, p. 405–426, 2015.

COPA. **COPA software respository**. 2017. Available from Internet: <https://gitlab.com/futebol/COPA>.

DJANGO. **Django,the web framework for perfectionists with deadlines**. 2018. Available from Internet: <https://www.djangoproject.com/>.

FUTEBOL. **Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory**. 2018. Available from Internet: <http://www.ict-futebol.org.br/>.

GNU. **iw, a new nl80211 based CLI configuration utility for wireless devices**. 2017. Available from Internet: <https://wireless.wiki.kernel.org/en/users/documentation/iw>.

HIGHSOFT. **Highcharts, Make your data come alive**. 2018. Available from Internet: <https://www.highcharts.com/>.

INTERNET2. **One-Way Ping**. 2018. Available from Internet: <http://software.internet2.edu/owamp/>.

MUNOZ, R. et al. The cttc 5g end-to-end experimental platform: Integrating heterogeneous wireless/optical networks, distributed cloud, and iot devices. **IEEE Vehicular Technology Magazine**, IEEE, v. 11, n. 1, p. 50–63, 2016.

SHIH, Y.-Y. et al. Enabling low-latency applications in fog-radio access networks. **IEEE Network**, IEEE, 2016.

SILVA B. ABREU, E. D. B. E. S. A. Demo abstract: Assessing the impact of fog and cloud computing on an iot service running over an optical/wireless network-an experimental approach. In: IEEE. **2017 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2017**. [S.l.], 2017. p. 950–951.

YANNUZZI, M. et al. Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing. In: IEEE. **2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)**. [S.l.], 2014. p. 325–329.