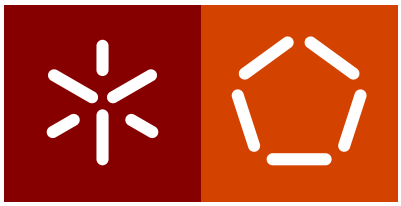
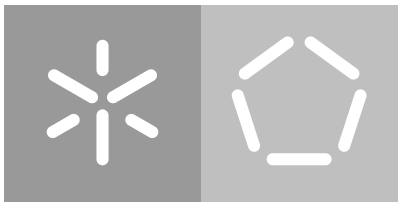


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Igor Virgílio Aquino Martins de Araújo

Virtual Network Function Development for NG-PON Access Network Architecture

January 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Igor Virgílio Aquino Martins de Araújo

Virtual Network Function Development for NG-PON Access Network Architecture

Master's dissertation
in Network and Telematic Services Engineering

Dissertation supervised by
Maria Solange Pires Ferreira Rito Lima
André Domingos Brízido

January 2022

AUTHOR COPYRIGHTS AND TERMS OF USAGE BY THIRD PARTIES

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Igor Virgílio Aquino Martins de Araújo

_____

ABSTRACT

The access to Internet services on a large scale, high throughput and low latency has grown at a very high pace over time, with a growing demand for media content and applications increasingly oriented towards data consumption. This fact about the use of data at the edge of the network requires the Central Offices (CO) of telecommunication providers, to be prepared to absorb these demands. COs generally offer data from various access methods, such as Passive Optical Network (PON) technologies, mobile networks, copper wired and others. For each of these technologies there may be different manufacturers that support only their respective hardware and software solutions, although they all share different network resources and have management, configuration and monitoring tools (Fault, Configuration, Accounting, Performance, and Security management - FCAPS) similar, but being distinct and isolated from each other, which produces huge investment in Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) and can cause barriers to innovation. Such panoramas forced the development of more flexible, scalable solutions that share platforms and network architectures that can meet this need and enable the evolution of networks. It is then proposed the architecture of Software-Defined Network (SDN) which has in its proposal to abstract the control plane from the data plane, in addition to the virtualization of several Network Function Virtualization (NFV). The SDN architecture allows APIs and protocols such as Openflow, NETCONF / YANG, RESTCONF, gRPC and others to be used so that there is communication between the various hardware and software elements that compose the network and consume network resources, such as services AAA, DHCP, routing, orchestration, management or various applications that may exist in this context.

This work then aims at the development of a virtualized network function, namely a VNF in the context of network security to be integrated as a component of an architecture guided by the SDN paradigm applied to broadband networks, and also adherent to the architecture OB-BAA promoted by the Broadband Forum. Such OB-BAA architecture fits into the initiative to modernize the Information Technology (IT) components of broadband networks, more specifically the Central Offices. With such development, it was intended to explore the concepts of network security, such as the IEEE 802.1X protocol applied in NG-PON networks for authentication and authorization of new network equipment. To achieve this goal, the development of the applications was based on the Golang language combined with gRPC programmable interfaces for communication between the various elements of the architecture. Network emulators were initially used, and then the components were "containerized" and inserted in the Docker and Kubernetes virtualization frameworks. Finally, performance metrics were analyzed in the usage tests, namely computational resource usage metrics (CPU,

memory and network I/O), in addition to the execution time of several processes performed by the developed applications.

Keywords: Passive Optical Network, Central Office, Network Function Virtualization, Virtual Network Function, Software-Defined Network,

RESUMO

O acesso aos serviços de Internet em larga escala, alto débito e baixa latência têm crescido em um ritmo bastante elevado ao longo dos tempos, com uma demanda crescente por conteúdos de media e aplicações cada vez mais orientadas ao consumo de dados. Tal fato acerca da utilização de dados na periferia da rede, obriga a que os *Central Offices* (CO) dos provedores de telecomunicações estejam preparados para absorver estas demandas. Os CO geralmente recebem dados de diversos métodos de acesso, como tecnologias *Passive Optical Network* (PON), redes móveis, cabladas em cobre, entre outros. Para cada uma destas tecnologias pode haver diferentes fabricantes que suportam somente suas respetivas soluções de *hardware* e *software*, apesar de todas compartilharem diversos recursos de rede e possuírem ferramentas de gestão, configuração e monitoração (Fault-management, Configuration, Accounting, Performance e Segurança - FCAPS) similares, mas serem distintas e isoladas entre si, o que se traduz em um enorme investimento em *Capital Expenditure* (CAPEX) e *Operational Expenditure* (OPEX) e pode causar barreiras à inovação. Tais panoramas forçaram o desenvolvimento de soluções mais flexíveis, escaláveis e que compartilhem plataformas e arquiteturas de redes que possam suprir tal necessidade e possibilitar a evolução das redes. Propõe-se então a arquitetura de redes definidas por software (*Software-Defined Network* - SDN) que tem em sua proposta abstrair o plano de controle do plano de dados, além da virtualização de diversas funções de rede (*Network Function Virtualization* - NFV). A arquitetura SDN possibilita que API's e protocolos como Openflow, NETCONF/YANG, RESTCONF, gRPC e outros, sejam utilizados para que haja comunicação entre os diversos elementos de *hardware* e *software* que estejam a compor a rede e a consumir recursos de redes, como serviços de AAA, DHCP, roteamento, orquestração, gestão ou diversas outras aplicações que possam existir neste contexto.

Este trabalho visa então o desenvolvimento de uma função de rede virtualizada nomeadamente uma (*Virtual Network Function* - VNF) no âmbito de segurança de redes a ser integrada como um componente de uma arquitetura orientada pelo paradigma de SDN aplicado a redes de banda larga, e aderente também à arquitetura OB-BAA promovida pelo Broadband Forum. Tal arquitetura OB-BAA se enquadra na iniciativa de modernização dos componentes de Tecnologia da Informação (TI) das redes de banda larga, mais especificamente dos *Central Offices*. Com tal desenvolvimento pretende-se explorar conceitos de segurança de redes, como o protocolo IEEE 802.1X aplicado em redes NG-PON para autenticação e autorização de novos equipamentos de rede. Para atingir tal objetivo, utilizou-se desenvolvimento de aplicações baseadas na linguagem Golang aliado com interfaces programáveis gRPC para comunicação entre os diversos elementos da arquitetura. Para emular tais componentes, utilizou-se inicialmente emuladores de rede, e em um segundo momento os componentes foram "containerizados" e inseridos nos frameworks de virtualização Docker e Kubernetes.

Por fim, foram analisadas métricas de desempenho nos testes executados, nomeadamente métricas de utilização de recursos computacionais (CPU, memória e tráfego de rede), além do tempo de execução de diversos processos desempenhados pelas aplicações desenvolvidas.

Palavras-Chave: Passive Optical Network, Central Office, Network Function Virtualization, Virtual Network Function, Software-Defined Network,

AGRADECIMENTOS

Deixo aqui os meus agradecimentos a todos que de alguma forma contribuíram para que esse projeto se torna-se realidade, de forma mais especial:

À meus pais e avós que a todo momento me incentivaram a sair da minha zona de conforto e buscar voos ainda mais longos e desafiantes. Além de proverem condições e a devida educação para que eu pudesse perseguir meus sonhos. Agradeço também ao meu irmão mais novo, que foi fundamental especificamente durante esse período pandêmico em que todos estiveram que estar reclusos, ele esteve ao meu lado, apoiando ao outro em todos os sentidos.

À minha orientadora acadêmica, Professora Solange Rito Lima, e ao meu orientador no âmbito empresarial deste projeto, o Engenheiro André Domingos Brízido, que proporcionou-me a oportunidade de aprender novos conceitos de programação, infraestrutura de redes e computacionais, que agregaram imenso meu currículo. Além do mais, e não menos importante, agradeço imensamente à Altice Labs pela confiança em meu trabalho e também pela oportunidade de fazer parte, mesmo que temporariamente, de sua equipa, e por acreditar que seria possível sim conduzir tal projeto mesmo na situação pandêmica que se passou.

À Universidade do Minho por oferecer um ensino de alto nível, e a por me acolher como aluno.

Aos meus colegas de curso e amigos para a vida, Adriano, Anderson, Leonardo, Matheus e Rosana, pelas trocas de conhecimento, experiências, angústias, e boas risadas.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS

A

ACLS Access Control Lists.

AON Active Optical Network.

APON ATM PON.

AR Augmented Reality.

ATM Asynchronous Transfer Mode.

B

BBF BroadBand Forum.

BPON BroadBand PON.

BSS Business System Support.

C

CAPEX Capital Expenditure.

CO Central Office.

CORD Central Office Re-architected as a Datacenter.

COTS Commercial Off-The-Shelf.

CSP Communications Services Providers.

CSPS Communications Services Providers'.

E

EAP Extensible Authentication Protocol.

EBSE Evidence-Based Software Engineering.

EMC Electromagnetic Compatibility.

EMI Electromagnetic Interference.

EMS Element Management Systems.

EPON Ethernet Passive Optical Network.

ETSI European Telecommunications Standards Institute.

EVE-NG Emulated Virtual Environment Next Generation.

F

FEC Forward Error Correction.

FIB Forwarding Information Base.

FTTX Fiber-to-the-X.

G

GPON Gigabit Passive Optical Network.

H

HFC Hybrid Fiber-Coax.

I

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IOT Internet of Things.

ISG Industry Standards Group.

ITU International Telecommunication Union.

L

LR-PON Long Range PON.

M

M2M Machine-to-Machine.

MAC Medium Access Control.

MANO NFV Management and Orchestration.

MPLS Multiprotocol Label Switching.

N

NAS Network Access Service.

NETCONF Network Configuration Protocol.

NFI NorthBound Interface.

NFV Network Function Virtualization.

NFVI Network Function Virtualization Infrastructure.

NG-PON Next Generation Passive Optical Network.

O

OAN Optical Access Network.

OB-BAA Open Broadband - Broadband Access Abstraction.

OCDM-PON Optical Code Division Multiplexing-PON.

ODN Optical Distribution Network.

OFDM-PON Orthogonal Frequency Division Multiplexing-PON.

OLT Optical Line Termination.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

ONT Optical Network Terminal.

ONU Optical Network Unit.

OPEX Operational Expenditure.

OSS Operation System Support.

P

PMA Persistent Management Agent.

PON Passive Optical Network.

Q

QOS Quality of Service.

R

RADIUS Remote Access Dial-In User Service.

RESTCONF Representational State Transfer Configuration Protocol.

S

SBI SouthBound Interface.

SDN Software-Defined Networking.

SDON Software-Defined Optical Networks.

SEBA SDN Enabled Broadband Access.

SFC Service Function Chain.

SIEPON Service Interoperability EPON.

SLR Systematic Literature Review.

SNMP Simple Network Management Protocol.

SRV6 Segment Routing IPv6.

T

TDM-PON Time Division Multiplexing-PON.

TDMA Time Division Multiple Access.

TWDM-PON Time and Wavelength Division Multiplexing-PON.

U

UDP User Datagram Protocol.

UHD Ultra-High Definition.

V

VLAN Virtual Local Area Network.

VM Virtual Machine.

VNF Virtual Network Function.

VOLTHA Virtual OLT Hardware Abstraction.

VOLTMF Virtual OLT Management Function.

VOMCI Virtual ONT Management Control Interface.

VON Virtual Optical Network.

VPNS Virtual Private Networks.

VR Virtual Reality.

VTEP VXLAN Tunnel EndPoint.

VXLAN Virtual Extensible Local Area Network.

W

WAN Wide Area Networks.

WDM-PON Wavelength Division Multiplexing-PON.

WOBAN Wireless-Optical BroadBand Access Network.

X

XDSL Digital Subscriber Line.

XGPON 10 Gigabit Passive Optical Network.

Y

YANG Yet Another Next Generation.

1

INTRODUCTION

Broadband networks have been evolving since the Internet rose. The optical fibers methods of data transmission have brought a big shift in reachability, capacity, and resilience compared with copper cable-based implementations. Moreover, the *Passive Optical Network* (*PON*) has stood out against other optical technologies. PON adds cost and energy savings to this list of benefits. For these points, PON and its versions as *Gigabit Passive Optical Network* (*GPON*), *Next Generation Passive Optical Network* (*NG-PON*), and *10 Gigabit Passive Optical Network* (*XG-PON*), has consolidated as one of the most used broadband access networks from network operators [1]. Further, the PON provided a significant expansion of bandwidth per client and network infrastructure quality, which opened new roads for the latest applications and impacted customers' habit Internet consumption.

Networks have been following the growth of the application complexity and the user demands for bandwidth, low latency, and volume of transmitted data. However, the networks, specifically the network operators, have been under pressure by the exponential demands from applications for flexibility, agility, and scalability [2]. The customer habit has been changing intensively. For example, the consumption of streaming content, as video and music, videoconferencing, have grown immensely in the last year and the forecast for global Internet traffic is definitely to increase [3]. Several emerging technologies and applications that will contribute to this expansion, such as *Virtual Reality* (*VR*), *Augmented Reality* (*AR*), gaming and video streaming, and other technologies in *Ultra-High Definition* (*UHD*), which demands high bandwidth and data usage, and low latency. Furthermore, the communication *Machine-to-Machine* (*M2M*), *Internet of Things* (*IoT*), Artificial Intelligence, Machine Learning, and Autonomous Vehicles will also add even more constraints to this scenario.

All these points pressure the network operators to handle high complexity and various data types, with their specific requirements. Considering the access network infrastructure provided by network operators to residential, enterprises, and mobile end users, it could be viewed as a bottleneck to the previously cited demands' growth. More specifically, the *Central Office* (*CO*) is the aggregation point infrastructure for access networks. Network operators could take advantage of CO's proximity to the end-users to offer an infrastructure that supplies the demand for ultra-low latency application responses [2]. Due to this scenario of the constant need for agility, flexibility, and adaptability imposed by applications on the *Communications Services Providers'* (*CSPs*) access networks, the *Software-Defined Networking* (*SDN*)

and *Network Function Virtualization* (*NFV*) paradigm appears as a promising solution to accommodate this scenario and seize this new opportunity.

Explaining the concepts of SDN and NFV briefly, they propose the decoupling from traditional hardware, e.g., routing, firewalling, switching, load balancing, and others with negligible customization and restricting scalability. SDN and NFV instead use to a pool of virtualized *Commercial Off-The-Shelf* (*COTS*) infrastructure which brings more flexibility and customization. One example of the most impacted sectors is the network operators. They have extensive infrastructures and may suffer from high *Capital Expenditure* (*CAPEX*) and *Operational Expenditure* (*OPEX*) embedded into unyielding traditional appliances. A specific affected infrastructure are the COs from PON structure, which is a promisor location for deployment of NFV Infrastructure due to the proximity with access nodes. So this characteristic would expand the capabilities of a significant amount of sensible applications mentioned before.

1.1 MOTIVATION AND OBJECTIVES

The COs are the first "big" hop in terms of infrastructure capacity for most of the access networks. They are located at the network's edges and usually aggregate multiple PON, xDSL, Mobile, and other kinds of networks. Due to this, the CO shows up as a good opportunity for CSP to avail the application's urge demands for low latency, high throughput, and fast processing to generate new incomes [2]. However, to take advantage of this opportunity, these demands require an imperative modernization of CO infrastructures to provide the desired flexibility, adaptability, and availability. To achieve these requirements, is proposed a softwarization bringing the concepts of SDN and NFV [4][5][6].

This work intends to present the traditional architecture of the Central Office, mainly related to PON infrastructure [1]. Also, it will be conceptualized with the state-of-art paradigms of SDN, NFV, and *Virtual Network Function* (*VNF*), and how CO could benefit from them. Moreover, after conceptualization, the development of a VNF is proposed to contribute to the *Altice Labs* next-generation Central Office. More specifically, is intended to create a VNF for Network Access Control for the legitimation of the identity from *Optical Network Unit* (*ONU*) into the PON domain. Related to this work, a few other specifications are also outlined along the state-of-art, including the adopted programming language and protocol interfaces used to construct and integrate the VNF to other architecture components [7].

To summarize the objectives of this master dissertation, they are synthesized as follow:

- Study the traditional PON architecture in Central Office, its limitations, new opportunities, and requirements to seize the opportunity of the novel demands;
- Study the paradigm of SDN/NFV and VNF, and how these concepts could modernize toward to softwarization of Central Offices;

- Develop a security IEEE 802.1X VNF to control the access of ONUs/ONTs into the PON domain;
- Evaluate in different scenarios the performance and feasibility of the developed VNF for the sake of future implementations.

Thus, this dissertation intends to investigate more deeply the state-of-the-art of the PON ecosystem and how it has been moving from traditional architecture to the modern paradigm of using SDN, virtualization, and programmable hardware, and the specific communication/programmable interfaces. Furthermore, this work will walk through a security Virtual Network Function development applied to a PON architecture's element.

1.2 DISSERTATION LAYOUT

The dissertation is organized into six chapters as described below:

In Chapter 1 - Introduction - is introduced some current emerging applications demands that impose pressure over the whole network, for more flexibility, adaptability, and agility. The opportunity for Central Office to modernize itself to seize these demands to create new revenues and keep up-to-date with these new application trends is also explored.

In Chapter 2 - Research Methodology And Bibliography Review - is conducted a *Systematic Literature Review* (*SLR*) to begin the bibliographic reference repository with important contributions to this dissertation. Thus, there is described all the applied methodology and obtained results.

In Chapter 3 - State-of-the-Art - all the main theoretic concepts and technology descriptions used in this dissertation are presented. The Passive Optical Network and Central Office are core motivator topics, which influences the other points. The next is the Software-Defined Network, Network Function Virtualization, Virtual Network Function applied to PON, and finally the Network Security issue.

In Chapter 4 - Methodology - is outlined the followed methodology used for the VNF development, including the employed tools, installation steps, programming languages, virtual topologies, and a possible VNF's deployment procedure on realistic infrastructure.

In Chapter 5 - Development and Results - are presented the results of all development, including the simulation results, showing the proposal of the VNF development and its applicability in a real scenario.

In Chapter 6 - Conclusions - are reviewed all the outcomes from the developed project, the difficulties, positive results, the best contributions, a comparison between the objectives and the results. Finally, the possible future works related to this thematic are pointed out.

2

RESEARCH METHODOLOGY AND BIBLIOGRAPHY REVIEW

In the last decades, the number of researches and their quality improved a lot, one possible reason for that could be the beginning of the Internet era. Before that, the availability of study materials, as articles, journals, and academic researches, was challenging to achieve. These new researches were restricted to the sole or limited source of references. It could result in poor material or even an extended period to conclude some research due to the lack of accessible content.

Nowadays, there are plenty of research sources databases to find many points of view from whatever thematic, which could sometimes stress finding out the most suitable research to aggregate to your study. Indeed, this new range of options is much better than before. Nevertheless, the toilsome work is still there, so to alleviate the pain, some research review methods were developed to filter the best, relevant and notorious resources to be used as solid bibliography for new studies and projects.

Among these methodologies, this chapter intends to introduce the *Systematic Literature Review (SLR)* that was chosen to support the construction of this master's dissertation bibliography.

This work intends to gather the state-of-the-art related to the central theme of this master dissertation, which is concerned about the study and the development of *Network Function Virtualization* (*NFV*) for *Passive Optical Network* (*PON*) access networks, following the programmability and automation tendency, brought by *Software-Defined Networking* (*SDN*), for traditional networks. Hence the goal of this chapter is to rely on SLR methodology to get an effective result of a better literature collection fit.

2.1 SYSTEMATIC LITERATURE REVIEW METHODOLOGY

One of the preeminent methods introduced mainly in medicine is evidence-based research and practice due to this domain being driven to investigation and experimentations. The goal of this method is to gather many scientific experiments and results about some topic the get a final, and better conclusion rather than a single medical advice [8].

In [8] pointed out the success of the Evidence-Based methodology and questioned its applicability in the Software Engineering domain, the *Evidence-Based Software Engineering* (*EBSE*).

It was observed in the following work [9] that a systematic review could be very effective to reach significant evidence about the researcher's theme, helping to find out which resources could not support and could support their hypotheses. This work also described some advantages, disadvantages, general features, steps, and many valuable pieces of information that many authors followed.

Since then, the SLR methodology has been improved, resulting in a very satisfactory outcome. In this dissertation, the search process for bibliographic references will be based on the SLR methodology as a relevant auxiliary component of this work.

The SLR methodology proposed by [9] and cited by [10] well-defined several steps to be followed that will also be used in this work.. Each step is illustrated in Figure 1

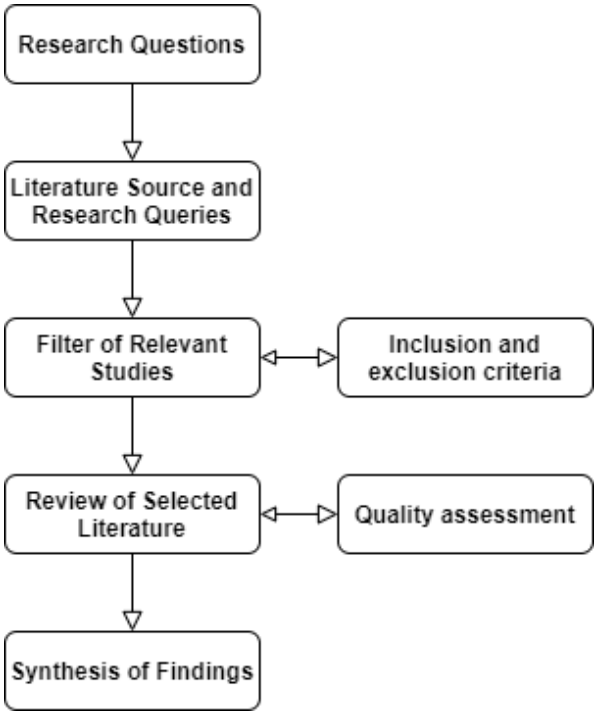


Figure 1: Conducted SLR steps (adapted from [10]).

2.2 RESEARCH QUESTIONS

The *Research Questions* are inquiries that were formulated to narrow the themes and subjects in answers that are composed of bibliographical references about the *Research Questions*.

(RQ1): ***What are the advantages of using the SDN/NFV architecture in PON networks?***

The SDN/NFV paradigm has become a synonym of programmability, flexibility, adaptivity, scalability, and many other attributes that traditional networks by now have willing for. Over time the way that we consume and demand the network as a whole has grown immensely. The operators need an agile network infrastructure that can quickly adjust to this ever-increasing demand for new services while assuring reliability and resilience. SDN/NFV is considered as an alternative to achieve this agility. Therefore, this question will address these SDN/NFV concepts and how they apply to PON and benefit from them.

(RQ2): ***Which are the suitable programming languages for VNF modules to use in a high-performance network?***

In this contemporary context of SDN and NFV, there is a concept of decoupling network functionalities from traditional network devices where there are usually a vast amount of functionalities that most of the time aren't used. So SDN and NFV propose the creation of modules for each function, namely VNF, that would be used, and each of them could be contained and consume a fraction of a pool of infrastructure resources instead of having a dedicated and idle computational resource. For this decoupling, these applications and modules should have an excellent performance, so the intent is to verify possible programming languages candidates for the development.

(RQ3): ***What are the best programmable interfaces between VNF and the whole infrastructure?***

The previous research questions raised issues about the modularization of network functions and the importance of standardization of technologies to maintain compatibility between them. For now, this question intends to address these thoughts into the programmable interface between these network applications, how they communicate and are compatible with each other, even constructed in different programming languages. Moreover, how the devices use these interfaces for management, monitoring, and configuration messages in an industry-defined data model structure.

2.3 LITERATURE SOURCE AND SEARCH TERMS

After establishing the research questions, we can move forward to the next step of the SLR procedures, the literature sources, and research queries. Following the [9] guidelines and [10] interpretation, the next step was selecting which literature sources we used to look for

the articles, papers, and other scientific literature that will help to answer the research questions. After doing the literature sources definition and listing essential keywords related to the previous research questions, the synonyms for each keyword were identified [10].

2.3.1 *Literature Source*

Given that the theme of this dissertation is related to the areas of network and informatic engineering, it was decided to start the primary research for the dissertation through the IEEE databases, specifically using the IEEEXplore. The reason is that this is an organization that brings a large number of relevant publications in this area. But it was also used other sources listed as follows. At Table 1 could be seen the list of selected Literature Sources:

Table 1: Literature sources

Resource type	**Resource name**
Online databases	ACM DigitalLibrary, IEEEXplore, ScienceDirect, Web Of Science
Others	Materials recommended by professionals in the field

2.3.2 *Search Terms*

Then, the first findings helped to find relevant keywords and then form a list of research terms, and for each term, derived some synonymous and alternative spellings. Below there are the main chosen keywords related to the dissertation proposal.

1. ***Passive Optical Network***- the PON is a quite adopted network for *Fiber-to-the-X* (*FTTx*), and this work discussed the state of the art of this technology;

2. ***Network Function Virtualization*** - NFV is a complementary topic to the SDN. It proposes to utilize *Commercial Off-The-Shelf* (*COTS*) hardware instead to use a specific one for some restricted range of functions like firewalls, load balancers, routers, and others. Thus the central point for NFV adoption is to virtualize all these roles, supported by a pool of infrastructure to deliver these functions;

3. ***Software Development*** - most of the presented terms are definitely software-oriented, so it is crucial to investigate more about how these trends are conceived in terms of software development.

4. ***Programmable Interface*** - after gathering these last topics related to softwarization and virtualization, was also intended to study the possible methods, namely which programmable interfaces as APIs. These APIs could attach different functions, modules, or plugins developed for different organizations in distinct programmable languages.

After defining the collection of key terms, the next phase is to generate some synonyms and variations from the keywords. Hereafter is outlined at Table 2 the list of terms with their variations and synonymous:

Table 2: Research terms

Terms	Synonyms and correlated terms
Passive Optical Network	*PON*, *GPON*, *NG-PON*
Network Function Virtualization	Virtual Network Function, Software-Defined Access Node
Software Development	Network Development, Programming Language
Programmable Interface	Representational State Transfer, API, Data Model, Remote Procedure Call, NETCONF

After defining these terms, a string could be formed using a Boolean expression (**ORs** and **ANDs**) to determine what is or is not related to each term to insert into the search engines. In this way, the engines of literature sources will look for the best matches for this string. Could be seen below the whole constructed string:

("Passive Optical Network" ***OR*** *PON* ***OR*** *GPON* ***OR*** *"NG-PON")* ***AND*** *("Network Function Virtualization"* ***OR*** *"Virtual Network Function"* ***OR*** *"Software-Defined Access Node")* ***AND*** *("Software Development"* ***OR*** *"Network Development"* ***OR*** *"Programming Language")* ***AND*** *("Programmable Interface"* ***OR*** *"Representational State Transfer"* ***OR*** *"Remote Procedure Call"* ***OR*** *"API"* ***OR*** *"Data Model"* ***OR*** *"gRPC"* ***OR*** *"NETCONF").*

Later, some experimentations with the entire string above show that the approached themes are not frequently correlated to each other, which returned no results at the search engines and databases. For this reason, the search was decoupled into multiple queries, one for each research question, using a shorter string with fewer terms and synonyms and variations. Right below is presented each personalized string for each research question:

- **Research question 1's string** - *("Passive Optical Network" OR PON OR GPON OR "NG-PON") AND ("Network Function Virtualization" OR "Virtual Network Function" OR "Software-Defined Access Node")*
- **Research question 2's string** - *("Network Function Virtualization" OR "Virtual Network Function" OR "Software-Defined Access Node") AND ("Software Development" OR "Network Development" OR "Programming Language")*
- **Research question 3's string** - *("Network Function Virtualization" OR "Virtual Network Function" OR "Software-Defined Access Node" OR "SDN") AND ("Programmable Interface" OR "Representational State Transfer" OR "Remote Procedure Call" OR "API" OR "Data Model" OR "gRPC" OR "NETCONF")*

2.4 FILTER OF RELEVANT STUDIES

This section mapped some strategies to filter among the found articles and papers, which of them are more suitable for this dissertation theme. Although the Research Queries made a tremendous funnel work, some next steps should be made to refine this source of bibliographic materials to find the ones with the most relevance [9]. These maneuverings follow exclusion and inclusion criteria that it's explained below:

Inclusion criteria

- If was found the same study in multiple tools, only the most refreshed version will be kept;
- Results in the year range of 2008 and 2021;
- Results that was cited at least two times, if it isn't a recent study (around last one year ago).

Exclusion criteria

- Studies that are a non-related study area (e.g. medicine, biology, politics, and others);
- Materials that don't consider the relationship with Passive Optical Networks, Network Function Virtualization or Software-Defined Network.

Due to the characteristics of the intercorrelated areas issued in this dissertation, was conducted three search queries. Then was applied the inclusion and exclusion criteria for each research query. After this procedure resulted in 37 pre-selected studies. The fine grain results could be consulted at Table 3 and Table 4.

Table 3: Found and filtered studies per search query.

Source	Found Studies	Filtered Studies
Search Query 1		
ACM DL	24	3
IEEEXplore	17	2
ScienceDirect	38	8
Web Of Science	17	2
Subtotal	96	15
Search Query 2		
ACM DL	57	2
IEEEXplore	21	3
ScienceDirect	54	4
Web Of Science	16	3
Subtotal	148	12
Search Query 3		
ACM DL	26	3
IEEEXplore	60	3
ScienceDirect	30	4
Web Of Science	36	5
Subtotal	152	15
Total	396	42

Table 4: Found and filtered studies in total.

Sum of All Research Queries Findings			
Source	**Found Studies**	**Filtered Studies**	**After Duplicate Exclusion**
ACM DL	107	8	6
IEEEXplore	98	8	8
ScienceDirect	122	16	13
Web Of Science	69	10	10
Total	396	42	37

2.5 REVIEW OF SELECTED LITERATURE

The filtered results returned a substantial amount of selected studies, the subsequent step was to make qualitative analysis to determine the most relevant references.

2.5.1 *Quality Assessment*

To help the quality assessment was used two relevant classification sites that evaluate the source of these studies, they are:

- ***CORE (The Computing Research and Education Association of Australasia)*** - Association of computer science departments of universities from New Zealand and Australia that classify a huge amount of published academic conferences and journals worldwide. Their site can be reached at `http://portal.core.edu.au/conf-ranks/` for conferences and `http://portal.core.edu.au/jnl-ranks/` for journals;
- ***SCImago (SCImago Journal Country Rank)*** - Web portal which collects data from Scopus databases. All the gathered information is translated into quality indicators to evaluate a vast quantity of study areas. Its site can be reached at `https://www.scimagojr.com`.

Based on these sources and their ranking method, was attributed a weight for each ranking due to classifying the found studies. Down below there are the condition grades (CG) for each ranking:

- **CG1** - has grade A* at CORE. Received a weight of 2.0;
- **CG2** - has grade A at CORE or Q1 at SCImago. Received a weight of 1.5;
- **CG3** - has grade B at CORE or Q2 at SCImago. Received a weight of 1.0;
- **CG4** - has grade C at CORE or Q3 at SCImago. Received a weight of 0.5;
- **CG5** - has no grade at CORE or SCImago or less than Q3. Received a weight of 0.0.

Note: *The condition grades are applied to only the quality assessment 1 (QA1), which is defined hereafter.*

With these condition grades established, now are listed the quality assessment (QA) questions, which also received condition weighted grades (CWG), as listed below:

- **CWG1** - study *has completely satisfied* the QA, received weight 1.0;
- **CWG2** - study *has partially satisfied* the QA, received weight 1.5;
- **CWG3** - study *hasn't satisfied* the QA, received weight 0.0;

Finally, after defining the grade conditions, the Quality Assessments with the respective feasible answers and its corresponding CWG are as follow:

- ***QA1*** - does the study belongs to a well-classified conference or journal?
  (accordingly to CGs ranking)
- ***QA2*** - does the study discuss VNF into Passive Optical Networks?
  (yes / no)
- ***QA3*** - does the study has a comprehensive approach to some dissertation topics?
  (yes / partially / no)
- ***QA4*** - does the study answers some of the Research Questions?
  (yes / partially / no)
- ***QA5*** - does the study discuss more than one Research Question?
  (yes / no)
- ***QA6*** - does the study has comparatives between technologies and return a valuable conclusion?
  (yes / no)
- ***QA7*** - does the study has a great value for this thesis' approach?
  (yes / partially / no)
- ***QA8*** - how many citations it study has?
  (more than 100 citations: 2.0 / less or equal than 100 and more than 50: 1.5 / less or equal than 50 and more than 25: 1.0 / less or equal than 25 and more than 5: 0.5 / less or equal than 5: 0.0)

In these qualification procedures, each selected study, resulted from Section 2.4, was classified and had its total grade depending on how it answered the Quality assessments. After this classification, the studies with a total score under 5.5, the medium rate from filtered studies, were excluded from the selected studies list, labeled from S01 to S18. Although the exclusion score constraint, there were some studies that were included (S12 and S13), even with a low score, because they have relevant content that wasn't found in other ones. The following Table 5 demonstrate the final list of selected studies with their respective grades, and also the Table 6.

Table 5: Quality Assessment score for last selected studies.

Study No.	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	Total Score
S01	1.5	1	1	1	1	1	1	1	**8.5**
S02	1.5	1	1	1	0	1	1	2	**8.5**
S03	1.5	1	1	1	0	1	1	1	**7.5**
S04	1.5	0	1	1	1	1	1	1	**7.5**
S05	1.5	1	0.5	1	0	1	1	1	**7**
S06	1.5	0	1	0.5	0	1	1	2	**7**
S07	1.5	0	1	0.5	0	1	1	2	**7**
S08	1.5	0	1	0.5	0	1	1	1.5	**6.5**
S09	2	1	0.5	1	0	0	1	1	**6.5**
S10	1.5	0	1	1	0	1	1	1	**6.5**
S11	1.5	0	1	0.5	0	1	1	1	**6**
*S12	1	1	1	0	0	1	1	0	**5**
*S13	0	1	0.5	1	0	1	1	0	**4.5**

In addition, a table with the respective research questions discussed for each study presented can be viewed below. It is noticed that there are few studies that address more than one Research Question, this fact is due to the fact that each RQ has its own Research Question, that is, a separate research was formed for each RQ. As previously mentioned, this methodology was adopted due to the difficulty of correlating the themes covered in this dissertation work. These results are shown at Table 6.

Table 6: Answered Research Question per selected study.

Study No.	Reference	Reference #	Research Questions		
			RQ1	RQ2	RQ3
S01	Bonfim et al., 2019	[11]	1	2	
S02	Thyagaturu et al., 2019	[12]	1		
S03	Chowdhury et al., 2019	[13]	1		
S04	Latif et al., 2020	[14]			3
S05	Khalili et al., 2017	[15]	1		
S06	Mijumbi et al., 2016	[16]	1		
S07	Yi et al., 2018	[6]	1		
S08	Abbas et al., 2016	[17]	1		
S09	Peterson et al., 2019	[5]	1		
S10	Ventre et al., 2018	[7]			3
S11	Demirci et al., 2019	[4]	1		
S12	Memon et al., 2020	[1]			
S13	Bruschi et al., 2018	[18]	1		

2.6 SYNTHESIS OF FINDINGS

This last step of SLR involves summarizing the most important findings of each final selection of studies filtered in previous sections. These synthesis are presented below:

Bonfim et al. **[11]** - The study made a Systematic Literature Review of NFV/SDN architecture integrations. It proposes studying the state-of-the-art NFV/SDN architecture, their architectural design, and possible improvements. The author defined the following three research questions: *i) In which environments are the integrated NFV/SDN solutions applied?*; *ii) What are the problems that such integrated NFV/SDN solutions are trying to solve?*; *iii) What are the differences among the design architectures of integrated NFV/SDN solutions?*. This work was conducted in 2016 and found 1,644 articles, with 907 duplicates, after that in the first phase was selected 138 studies. In stage 2, 88 studies were selected for a deeper quality analysis that resulted in 48 studies. Was also added 26 articles that were recommended by experts, which totalized 76 articles. After this selection procedure, the author used the selected studies to answer the research questions. Then was concluded that Cloud Computing is the dominant scenario for NFV/SDN implementations due to flexibility and scalability [11]. Also was identified the utilization growth of *Open Network Operating System* (*ONOS*) SDN controller, and the decline of citation of OpenFlow 1.0 for NFV/SDN architectures in the overall publications. Another observed point is that the SDN/NFV architecture has the interconnecting VNFs/VNFCs as a primary focus for all areas. Finally, was claimed the attention to the cost issue, and the SDN has been used to reduce the CAPEX and OPEX costs.

Thyagaturu et al. **[12]** - The study claims that although the SDN-based optical communication characteristics pose some challenges, it also holds great potential. So the author presented a comprehensive survey along this thematic of *Software-Defined Optical Networks* (*SDON*), focusing on the infrastructure, control, and application layers, as well as orchestration. The paper found studies that examined the optical transmission and switching suitable for SDN-controlled operation and investigated the performance of some optical monitoring frameworks. It also analyzed the *Virtual Optical Network* (*VON*) over physical infrastructure and the impact on physical and *Medium Access Control* (*MAC*) layers of access networks. Additionally, the survey showed works in *Quality of Service* (*QoS*), access control and security, energy efficiency, and failure recovery. Finally, reported studies in orchestration areas that addressed the coordination mechanisms across multiple layers and multiple network domains.

Chowdhury et al. **[13]** - This study agrees with the fact that NFV is an enabler for softwarization. However, the massive use of VNF can generate a repeated execution of redundant functionalities, which could raise an overhead issue. So the authors intend to study this thematic chaining of VNF, *Service Function Chain* (*SFC*), which could be very impactful for operational complexity and expenditure. They addressed the academic and industry point of view on this topic. Moreover, settled a use-case of a large-scale cloud application using microservices to study all the impacts like resource allocation, monitoring, fault tolerance,

language for VNF and SFC composition, communication primitives for VNF composition, and such other aspects. In conclusion, the study reinforces the use of microservices architecture on the NFV ecosystem to speed up innovation in this area. Finally, is pointed out that this topic is still in its infancy

Latif et al. **[14]** - This study surveyed comprehensively the interface communication/programming protocols used for SDN, namely the *SouthBound Interface* (*SBI*), *NorthBound Interface* (*NFI*), and east/westbound interfaces. The study classified the finding into directional-communication properties and subclassified them based on the functionality. As a result, it was concluded that OpenFlow still has a presence in SBI studies. On the other hand, the NBI and east/westbound interfaces have more diversity and a different approach that isn't well-established in a common protocol, being used to communicate between applications, enabling the operators to control, configure, and program the network.

Khalili et al. **[15]** - The authors raised an introduction of the Software-Defined Network paradigm to the *Ethernet Passive Optical Network* (*EPON*), that includes the *Service Interoperability EPON* (*SIEPON*) architecture. They proposed to apply SDN to SIEPON to separate and virtualize the control and management functions of the OLT, together with OAM functions, building an OLT around an OpenFlow switch. This proposal allows to optimize resource allocation and service availability, thus enhancing SLA and QoS requirements. Results also presented an improvement in the average throughput of the ONUs in downstream and upstream channels and the average data packet delay. So, this project seems to be a flexible alternative for multi-tenant and multi-provider PON networks.

Mijumbi et al. **[16]** - The authors present a deep research about NFV, revealing it state-of-the-art, and also correlate with SDN, which is closely connected to NFV topics. These are described in the study as having well-defined architecture design, specification and standardization efforts, research projects, commercial products, and early NFV proof of concept implementations. Finally, the future perspective to Service Providers are discussed.

Yi et al. **[6]** - In this study, the author presents a comprehensive survey about NFV, explaining the motivation, history, terminology, standardization, and comparison with the traditional approach with use-cases. It is also conceptualized the VNF's algorithms, placement, scheduling, migration, chaining, and multicast. The study finalizes with future challenges that technology might face.

Abbas et al. **[17]** - The author in this study presents a literature review of *Passive Optical Network* (*PON*) focusing on the Next-Generation PON, specifically on stage 2 PON (NG-PON2). It is defined the first generations of PON and compares them with the newer proposal. Then is also discussed the main hurdles which NG-PON2 has to handle, as capacity, cost, compatibility, and other topics.

Peterson et al. **[5]** - In this study, the Edge network is the main topic. It is concerned about the great opportunity to innovate at the edges that industry and Service Providers

have with the current momentum of Internet demands. These demands are related to the increasing addressable market for IoT, the increasing complexity of applications that require a very low latency, which obligates the infrastructure to be closer to the end-user. This point forces a "cloudification" of the edges (on-premise, on-vehicle, in the cell tower, in the Central Office, or distributed across a metro area, or all of the above [5]) to satisfy these demands. The study address many concepts that are part of this edge modernization, and also some current project, such as *Central Office Re-architected as a Datacenter* (*CORD*). So this work finish with a "call-to-action", defining a plan to achieve and seize this mentioned opportunity. This plan suggests three actions that are: *i) Focus on the Access-Edge*; *ii) Participate in Open Source*; *iii) Operationalize the System*.

Ventre et al. **[7]** - This study discusses the programmable interfaces that could be implemented in a SDN *SouthBound Interface* (*SBI*). More specifically, it is concerned about APIs for *Segment Routing IPv6* (*SRv6*) applied on *Wide Area Networks* (*WAN*). The authors propose the design and implementation of the Southbound API between the SDN controller and the SRv6 device. Furthermore, defined a data model and four different implementations of the API, respectively, based on gRPC, REST, NETCONF, and remote Command Line Interface (CLI) [7]. They also created a testbed to compare these API candidates, where was evaluated the CPU and memory utilization and response time. The result was that gRPC and REST got the best results.

Demirci et al. **[4]** - This work discusses the context of SDN and NVF, focusing on placement area for VNFs in this context. It reviewed the state-of-the-art studies and categorized them based on the defined taxonomy. This study results in a developed taxonomy for the classification of VNF placement solutions in SDN. There were stated nine different dimensions for classifying VNF placement. They are: *role of the functions, placement approaches, optimization objectives, optimization methods, flow handling strategies, application domain, used data, optimization tools, and simulation tools* [4]. They also discussed some mathematical and algorithmic approaches to the VNF placement problem. Finally, they listed some future works and evidenced the security challenges.

Memon et al. **[1]** - The authors in this study discussed an extensible study of the last 10 years (2010-2020) of the evolution of *Passive Optical Network* (*PON*) technology. The study analyzed around 3,381 publications. With this huge database, this work might serve as a valuable guide for PON researchers. This work resulted in a rich study with many graphs and principal terms, authors, and projects that have great relevance to the PON scene.

Bruschi et al. **[18]** - The study presents an evaluation of the growing demand for cloudification to the edges into Mobile Edge Computing, which is driven by IoT and 5G emerging technology. It focuses on analyzing the best placement for these micro-data centers (microDC) in the urban environment. This study takes into account the tradeoff between *Quality of Service* (*QoS*) and the Service Providers costs. One example of possible placement is at the Central Offices in a PON architecture. One gap that the study address is how much proximity

these MECs should have to the end-user. So they obtained results from a case of a microDC deployment, considering a 5G implementation, simulating the impact of using a Central Office as microDC to supply this 5G deployment. This resulted in a latency decrease, but the cost is high, so the less scattered design is still preferable for the near future [18].

2.7 SUMMARY

In this chapter, a systematic literature review was conducted, which had a significant contribution to making a preliminary selection of the most relevant articles related to the main themes of this dissertation. The steps of the adopted methodology were detailed, citing the research question, defining literature sources, search terms, filter the relevant studies, quality assessment, and finally the synthesis of finding. Also, some adjustments were required to fit with the theme's peculiarity and get a non-obvious correlation between the dissertation's theme, which explicitly influences the number of search queries and the number of found studies.

After the review of the selected articles were found extensive studies that proposed distinct points of view about the context of SDN/NFV/VNF, and each ended in a valuable result.

Related to PON theme, were selected fewer related studies. However, they also contribute to the dissertation, which correlates the network and application trends that could impact the central offices and PON technologies.

For the programmable interfaces was found even fewer contents. Also, some of the selected works didn't achieve the medium quality assessment grade due to the lack of ranking conference/journal registry and the infancy of this topic. But, besides that, they were included because their content would be valuable for this project, since this dissertation's practical part is to directly handle programmable interfaces for communication between VNFs and other topology elements.

Thus, the researches carried out in this chapter helped to consolidate an initial set of valuable references and contributed to visualizing a roadmap for the following chapters.

3

STATE OF THE ART

This chapter conceptualizes most of the central technologies used in this work and how they interact. The initial section defines the crucial characteristics of passive optical networks. Afterward, the following sections present the notion about virtualization of IT components, which opened the road for novel ideas in multiple areas in IT, for example, hypervisors, Cloud Computing, and SDN. Lastly, some concepts about network security regarding access control methods are introduced, such as the protocol IEEE 802.1X.

3.1 PASSIVE OPTICAL NETWORKS (PON)

The broadband sector has been growing over the years, allowing many applications to evolve and offer new services for end-users, as triple-play services (i.e., data, voice, and multimedia). The legacy broadband technologies, as *Digital Subscriber Line* (*xDSL*) telephone wire cable, were inefficient in many aspects, such as reachability, operational expenditures, and low capacity. To overcome these characteristics, optical technology introduced a significant shift to broadband transmission lines. Initially, these previous lines were mainly composed of copper wires between service providers' facilities to residential and enterprise customers. Afterward, these medium infrastructures evolved to a *Hybrid Fiber-Coax* (*HFC*), which part of these lines between customer and network providers were fiber cable and near to the subscribers, in the last mile, with coax cables. Due to this hybrid architecture, it had spread the concepts of *Fiber-to-the-X* (*FTTx*), and the fiber technology started to expand its capillarity, namely the *Passive Optical Network* (*PON*) architecture because of the characteristics described in this section.

3.1.1 *Introduction*

The fiber-based technology, specifically the PON, debuted around 1980. Still, the related costs were prohibitive at those times compared to copper materials, even with the list of benefits over traditional mediums. Since then, the technology was evolving, and in the late 1990s, the optical scene had changed and wiped some elements from architecture to reach a more concise design [19]. Little by little, standards emerged due to the work of two relevant orga-

nizations, *International Telecommunication Union* (*ITU*) and *Institute of Electrical and Electronics Engineers* (*IEEE*), who started parallel projects and guided the evolution and democratization of fiber-optical projects.

The PON evolution could be divided into three phases, which are well recognized by its technology enhancements. These phases are: deployed PON; next-generation stage 1 (NG-PON1); and next-generation stage 2 (NG-PON2). Moreover, NG-PON3 is already mapped and could be integrated soon as a fourth phase. Along the following paragraphs are briefly described relevant characteristics for each stage [17].

A prominent transmission protocol leveraged the first phase and a primary relevant PON standard at that time, the *Asynchronous Transfer Mode* (*ATM*), resulted in the *ATM PON* (*APON*) standard (ITU-T G.983), followed by its unfolding, the *BroadBand PON* (*BPON*) [20]. This version specified a 622 Mbps downstream and 155 Mbps upstream data rate, reaching 20 km between OLT and 32 ONUs. In 2001, the *Ethernet Passive Optical Network* (*EPON*), IEEE 802.3ah standard, was released with a higher data rate of 1.0 Gbps for down and upstream, but serving 16 ONUs per OLT port reaching 20 km between them [19]. Contemporaneous to EPON, the GPON came to compete for market share. GPON was ratified in 2003 with the standard ITU-T G.984, bringing more capacity and scalability, typically operating in 2.4 Gbps for downstream, 1.2 Gbps for upstream, and split ratio up to 64 ONUs over to 60 km of distance, but typically deployed up to 20 km [19]. Both specifications, EPON and GPON, have similar services offered to the customers and main infrastructure components and technical characteristics as both are *Time Division Multiple Access* (*TDMA*) based. However, there are highlighted differences regarding the data link, and physical layers [17], whereas how the traffic is encapsulated to downstream and upstream for both PON variances.

Concerning of the second phase (NG-PON1), it brought a significant jump of capacity related to data rates for both standards, which achieve 10 Gbps of data rates. The EPON evolved to XG-EPON, and the GPON developed to XG-GPON. In the first evolving, in 2009, the XG-EPON (defined in IEEE 802.3av) inherited some characteristics, yet had changed physical layer aspects concerning wavelength range, line code, and *Forward Error Correction* (*FEC*) [17]. Hence a meaningful change was that it could enable symmetric and asymmetric data rates. Like XG-EPON, the XG-PON (defined in ITU-T G.987 series), in 2010, also changed the wavelength range, line code, and FEC, which enhanced the performance and provided both symmetric and asymmetric data rates too.

The third phase, in 2013, the NG-PON2 data rates (defined in ITU-T G.989 series) went even further, typically allowing 40 Gbps for downstream and 10 Gbps for upstream. Moreover, it has the reachability of 40 Km maximum passive fiber and up to 60 km with reach extenders, and with a split ratio of 1:64 [21]. To achieve these gains lots of multiplexing technologies were proposed, as high-speed *Time Division Multiplexing-PON* (*TDM-PON*), *Wavelength Division Multiplexing-PON* (*WDM-PON*), *Optical Code Division Multiplexing-PON* (*OCDM-PON*), *Orthogonal Frequency Division Multiplexing-PON* (*OFDM-PON*), and hybrid technologies as

Time and Wavelength Division Multiplexing-PON (*TWDM-PON*), which had a great adoption due to cost-effective architecture [17] [19] [21]. Each option is more suitable for a specific scenario, and each one should avail the trade-off of several parameters, i.e., capacity, reachability, and split ratio. A good feature of NG-PON2 is compatibility with legacy generations as GPON and XG-PON, which allows seamless migration to new standards and with lower capital expenditure for network providers.

3.1.2 *Architecture and Components*

Among the *Optical Access Network* (*OAN*) broadband technologies, the Passive Optical Networks has excelled rather than others due to its cost-effectiveness. One reason for these advantages and preference by network providers is the fact that PONs don't need active equipment between providers and clients, in contrary to *Active Optical Network* (*AON*) that does need an optical-electrical-optical at the remote node. This passive approach consequently provides more saving of CAPEX and for OPEX also, because there is no power consumption for equipment throughout the *Optical Distribution Network* (*ODN*) [21].

The PON's architecture from a high-level point of view is quite simple. It follows mainly a point-to-multipoint architecture and could be separated into three parts, the Central Office, the Optical Distribution Network, and the Customer side. The first part is located in the *Communications Services Providers* (*CSP*) facilities, namely in the CO, where there is aggregator equipment, the *Optical Line Termination* (*OLT*). Each its port serves multiple points, partitioning the stream of these single wires (feeder fiber) through a splitter, to various cables, each one connected to a *Optical Network Unit* (*ONU*) and *Optical Network Terminal* (*ONT*). The COs also have a significant number of other services and aggregate other technologies, as mobile network infrastructure and more. The second part is the ODN, in which there is all the infrastructure between the CSP and the customers, where includes the fiber cables, splitting remote nodes (RN), and are all passive elements. Finally, the customer side, where the ONUs/ONTs receive a branch (distribution fiber) from the split feeder fiber. All these components are illustrated in Figure 2.

As introduced before, the Network Operators aggregate a significant number of technologies in the Central Offices. Moreover, it is a facility that could implement a similar datacenter infrastructure with compute, storage, and networking, as a traditional cloud provider. Also, it is the uplink point to the core network. Thus it could be beneficial for Network Providers and Operators to offer new business and services. Still, they need to be more agile, flexible, and scalable, provide better management, orchestration, and visibility. To deliver these features, the *Software-Defined Networking* (*SDN*) paradigm seems to be a great candidate to leverage the Central Offices to the next stage. The Section 3.4 will cover more deeply other components of CO and the latest initiatives concerning CO modernization, and how SDN is helping the CO's improvements.

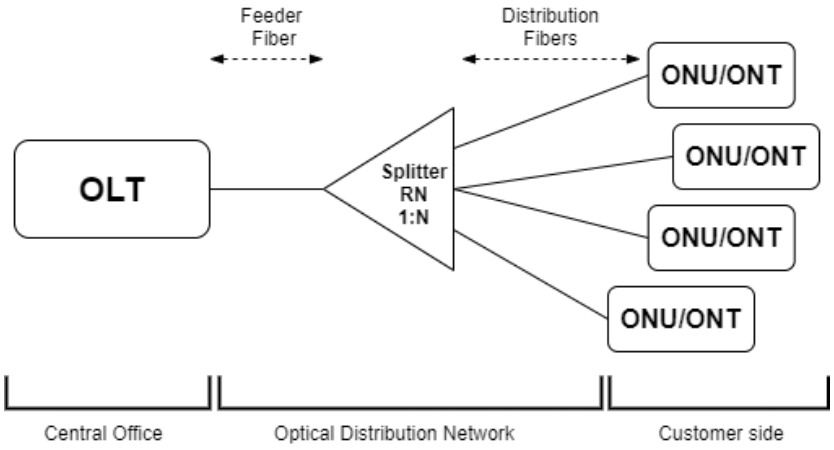


Figure 2: Components of PON Architecture (adapted from [17]).

3.1.3 *Benefits*

PON's characteristics could explain why many carriers and service providers widely adopt its solution. These features are [20]:

Performance - Regarding performance, it is closely related to capacity, amount of services delivered with quality at an affordable cost. Moreover, it is "deliver more with less". Thus the PON architecture increased the performance generation by generation, expanding the information-carrying capacity, reducing costs, and reaching longer distances.

Resiliency - Several aspects of optical technologies as a whole could be mentioned, when comparing to copper-based access infrastructure, as coaxial and twisted pair cables. One of them is the immunity from *Electromagnetic Interference* (*EMI*) and *Electromagnetic Compatibility* (*EMC*), which can cause a huge disturbance in transmission flow through copper conductors or elevate the cost if it is used shielded cables [22]. Another material physical feature of optical fibers against copper cable is durability. Moreover, the copper is less resistant to the environment and chemical impacts and oxidizes faster than silica glass does [23]. Although these positive points, there is a certain vulnerability that providers should be aware of to keep the network reliability. For example, in the ODN infrastructure, there is a single point of failure at the feeder fiber. Thus it should be very protected against possible harms.

Cost-efficiency - Regarding the cost issue, the optical fiber had a very high and prohibitive cost at the early stages of technology. Nowadays, it is still a non-cheap solution, but the gain in performance, quality, capacity, and other aspects turn into a very optimal choice for more and more scenarios, even those of FTTH. Moreover, optical solutions haven't some OPEX cost, as maintenance of optoelectronic and electronic devices along the ODN, reducing dramatically the failure rates and repair cost associated with these devices, and all the infrastructure and facilities cost around them. Among the optical technologies options, the PON stands out even more than others as AON, e.g., the PON solution typically doesn't need active devices

on ODN at the cabinets, and this devices' power consumption. The PON is also a future-proof technology due to its high information-carrying capacity, and the previous inherent characteristics of the optical material [19]. Thereby, it's expected that the cost per bit will fall even more with this capacity growth. For instance, the study reported in [24] proposed an optimization framework and provides a comparison between multiple generations of PON. This study had shown that scenarios of higher demand of bit rate, newer generations, which tend to be more expensive and with higher capacity, are more cost-effective than old cheapest ones.

Convergence and Integration - The demand for connectivity as a whole is just increasing, and this is pushing broadband technologies and network operators to evolve. So, the PON architecture has been used to support other technologies which rely on it and converge their traffic to a PON infrastructure for wired and wireless networks. An example is the *Wireless-Optical BroadBand Access Network* (*WOBAN*), which provides a more cost-effective solution and a less invasive deployment for subscribers. The customers get access through a wireless front-end infrastructure (e.g., WiMAX, LTE, Wi-Fi, or a combination of these), and these wireless base stations are connected to the ONU [25]. Moreover, PON also has been integrating with other solutions as copper, for drops near to the customer, composing a heterogeneous solution, for example, with PON and DSL or Coax cables [25]. Another integration is the PON with Metro networks, which adapts the traditional PON "tree-and-branch" to ring topology, where it is committed to *Long Range PON* (*LR-PON*).

It was possible to perceive that the Central Offices might be the main aggregator facilities that join these heterogeneous networks. Thus, the CO appears here as a great candidate to be improved and evolved, which could benefit all the networks that usufruct the COs. These potential benefits will be covered in Section 3.4, after an introduction of SDN and virtualization concepts which might be a prerequisite to perceive the potential benefits of Cloud Central Offices approaches.

3.2 VIRTUALIZATION OF IT COMPONENTS

An excellent way to understand why using some technology is to understand its past and the old difficulties and problems it proposes to solve. Thus, this section introduces a brief history of computers, data centers, and their evolution and virtualization conception that drove Cloud solutions and modern computational architectures. After that, it will reach the networking domain, pointing out some traditional characteristics that halt the network from following the application demands. Furthermore, still in networking scope, it will cover recent paradigms about *Virtual Network Function* (*VNF*) and *Network Function Virtualization* (*NFV*). At the end of this section, there will get a proper introduction subject for the following sections regarding the *Software-Defined Networking* (*SDN*) and applicability to broadband evolves.

3.2.1 *Data Centers History*

The first ages of computing technology known as nowadays with graphical UI and a good memory debuted around 1970 when Xerox launched the Xerox Alto, the first desktop computer with these characteristics. Until those times, the mainframes were costly, and few enterprises with great demands and capital could afford them. Thus the majority of companies were used to having PC desktops to execute the daily workload.

Although Moore's Law has been confirmed constantly at the beginning of computer evolution, between 1970 and 1980, and the desktop became very common, the mainframes were still not financially affordable for numerous enterprises. So these workloads were growing, and the expenditure on computing decreased, which stimulated data centers' creation to house dedicated and optimized machines for processing these demands centrally. As a result, most of them chose to outsource this specific type of infrastructure.

Year by year, the society has been more data-driven, stimulated by more and more rapidly and richly information consumption. It forces the companies to invest in IT infrastructure and guarantee that these premises will be available all the time [26]. This necessity of multiple servers and backup servers for redundancy in most scenarios. So having a dedicated server for some applications will be idle most of the time in computational processing. At the same time, the consumed energy, cooling, facility's space, and other costs will keep billing. Thus, this inefficiency forces the vendors and enterprises to seek a better way of deploying this infrastructure, which consists of sharing these resources.

The resource sharing already had taken place at that mainframe's times, where the computational power was scarce and expensive. Moreover, this "sharing" concept was grown, and new manners of segmentation evolved to divide and isolate one use from another. This segmentation reaches the storage from an IT perspective, where the hard disks could be partitioned into multiple smaller disk slices. Also, the network started to be possible to transmit in the same wire various users' traffic from a different sector that shouldn't communicate with each other, using the *Virtual Local Area Network* (*VLAN*) concept. Further, the CPU was shared. It was a single-core CPU processing a task at once, after it evolved to a multi-core architecture. Beyond that, the CPU was also divided logically to be used for multiple jobs. So, this resource sharing culminated in the virtualization concept, which is very used nowadays, and concerns this dissertation's main topics.

3.2.2 *Virtualization*

Server Virtualization

Due to the inefficiency of dedicated servers for each specific application, a better deployment solution surged to fill this gap. Before this solution, these dedicated servers were built with

one hardware generally composed of a CPU and Memory, Storage, and Network interfaces, with an operational system (OS), installed on it depending on the application requirements. Looking more deeply into traditional server architecture, as already said, the server's hardware has an OS that has mapped all the hardware drivers, and it reserves all the resources for an installed application, like as shown in Figure 3.

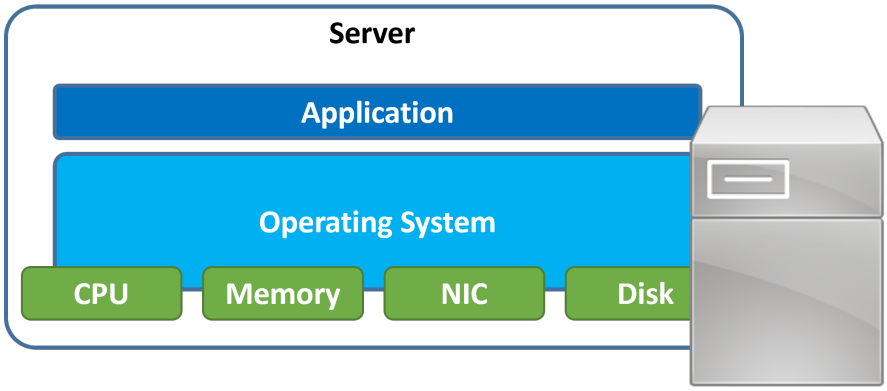


Figure 3: Traditional Server.

To fill this gap, the virtualization concept took place. The virtualization idea is to share the computational resource that would be underutilized in the host with more than a single application which would never reach the peak of the utilization of this server. The virtualization paradigm adds a new layer between the hardware and OS to create micro servers which use a part of available hardware power. These micro servers are called as *Virtual Machine* (*VM*), and the new layer is the hypervisor.

Hypervisors

The hypervisors allow the system administrator to host multiple applications in distinct VMs with different OS and segment them from each other, with their specific computational power, storage, and network. This method can enhance the overall efficiency of the server, which could keep running more closely to full capacity (respecting a safe margin to avoid crashes) instead of wasting idle resources with a single running application [26].

Another benefit brought by virtualization is the ability to have portable VMs. It means that one application hosted into a VM can be easily migrated to another data center in a couple of hours compared to the older approach, which could take weeks or months for this job [26]. Thus, the hypervisors have their implementations to execute and facilitate this task.

The hypervisor can be called an Operational System for Operational Systems. It is software installed into a server or desktop which gains access to manage the I/O all over the host's hardware, CPU, memory, storage, and network. Generally, there are two kinds of hy-

pervisor, Type 1 and Type 2. The first one is the software embedded directly on the hardware, not requiring a previously installed operating system, and it is called a bare-metal hypervisor. Due to this feature being straightforward on hardware, it has better performance and is recommended for servers [26]. On the other hand, hypervisor Type 2 is installed over an existing OS, as a hosted hypervisor, like Windows, macOS, and Linux. This type is less performative than type 1, and it usually runs on desktops and laptops.

Thus, the hypervisor has been performing a crucial task for virtualization, enabled great advances in the IT world, and provided considerable business growth. It also a key tool for hype technologies and new concepts as Cloud computing and *Software-Defined Networking* (*SDN*), which are covered further in this dissertation.

3.2.3 *Cloud Computing*

The virtualization method, as already said, brought new opportunities in many IT areas. One of them is Cloud Computing, and sometimes called a virtualized data center. Clouds comprise totally in virtualization, and it results in many benefits for IT. Some of them are listed below [26]:

- **CAPEX savings -** Due to improved utilization of the servers' hardware, less equipment is bought compared to traditional architecture. Moreover, the cloud provider can buy equipment on a large scale, reducing, even more, the costs. Regarding these savings the providers can be passed to their clients.

- **Faster Deployment -** All the environments are virtualized and managed through a hypervisor, who will allocate the hardware resources for each demanded application in a couple of minutes. It is no more necessary to instantiate new physical hardware for a new application. In just a few clicks, a new environment is up and running, and in the same way, it can be shut down and terminated.

- **Testing and Development -** In an IT area, various teams depend on the infrastructure to get their job done. An example is a development and operational teams that constantly test new features and applications, and the environment provisioning tasks must be fast to deliver on time their jobs.

- **On-demand self-service -** This characteristic allows anyone to provision a computational service as needed, with no need for IT personas. Of course, it isn't a simple job. A skillset would be required to deploy an efficient, secure, scalable, and robust solution into the cloud environment. But, the point is, the deployment procedures are much simpler and faster in the Cloud than in on-premises implementations.

- **Ubiquitous Network Access -** Differently from the on-premises data center, which usually demands specialized equipment like routers or firewalls to get remote and secure

access from outside the internal network. The Cloud, by default, allows this remote access natively, transparently for the user, with no need to instantiate specific equipment to provide this access, unless specific deployments, integrations, and security measurements.

- **Pay per Use -** In traditional data centers, the costs are usually paid upfront for equipment, disregarding all the OPEX costs, like cooling, energy, and others that are billed independently of the computational uses. On the other hand, in a cloud scenario, you pay only for your use on-demand. There are upfront and other payment options when is well-known previously workloads. Thus, cloud solutions eliminate the cost of ownership, and the Cloud's clients outsource their computational demands as they need [26].

- **Rapid Elasticity -** This is a feature in which many businesses can take part and enables many of them. Some companies need a heavy computational use for a specific workload, which a mighty and expensive server would perform. In this case, it would be impracticable to maintain this cost on-premises because all this power would be idle most of the time. So the Cloud provides the opportunity to instantiate a considerable machine to perform a heavy job and shut this machine down when finished the task, avoiding a waste of money.

  Thus, it is possible to adapt the needed IT resource to follow the businesses demands, comprising automation, programmability, flexibility, and rapidity that Cloud offers. Moreover, it allows small companies to develop advanced outcomes that could now be affordable, which wouldn't be feasible if they had to buy and maintain their resources.

- **Location-independent Resource Pooling -** The Cloud, since the beginning, has the concept of resource sharing and optimizing the use of infrastructure to the maximum. To provide this, the cloud companies, by default, have their resources as a pool to allocate for many different clients in the same physical server, storage, network, database, and so on. The segmentation and virtualization are very well-done by them to faultlessly allocate a piece of resource to each client with conflicts. Due to this, their infrastructure is hosted independently of the location for Cloud's clients, unless it isn't specified previously, and the resources are considered as a pool. Although, the client can pinpoint where their solutions are hosted and even pay more for a dedicated and exclusive server furthermore a dedicated rack.

The Cloud has become an inevitable solution for many companies of multiple niches and scales. The providers brought the missing tool to allow many enterprises to grow, scale, advance. This tool stimulated the evolution of the applications, and it sums into pressure on networks, which didn't follow this unfolding at the same pace. The following topic will discuss the traditional network difficulties and the alternative for that.

3.2.4 *Traditional Network Architecture*

Over the past years, the IT area has evolved a lot. The computational power increased, better ways of utilization have arrived, the cloud solution was born, applications have become even more complex, and with all of this scenario, the data generation and consumption have exploded. All of these put some weight over the network, requiring more bandwidth capacity, less latency, jitter, and losses, driven mainly by newer applications as high video quality, *Internet of Things* (*IoT*) implementations, autonomous vehicles, and so on [27].

To address this evolution, the network service providers try to improve their network capacity and scale, but the traditional network architecture imposes some constraints for this progression. As following are cited, some of these limitations [27]:

- **Flexibility -** Each vendor intends to promote its portfolio. For this reason, they produce a specific set of hardware to perform a particular function. In addition, they also develop complementary software specifically for it and with no compatibility or very restrict set of functions, with any other vendor. From a clients' perspective, when they adopt some vendor's solution, it restricts their deployed features, just for the vendor capabilities, disabling any client customization, resulting in a limited and inflexible network.

- **Scalability -** As mentioned before, traditional on-premises equipment demands a physical space with power consumption and substantial cabling paths, which aren't scalable compared with the vanguard approach. On the software side, each vendor generally produces a capacity-constrained product for some feature that can sometimes be upgraded with a license purchase, otherwise not even this way. So this fact reduces the traditional network scalability.

- **Time-to-Market Challenges-** Technology is constantly changing, and this often implies new features that require a vendor's box upgrades or replacement to offer the latest releases. But infrastructure changes are pretty complex and costly. So, this can impact the time to deliver new features for the enterprise's clients and effects revenue outcomes.

- **Lack of Manageability -** There are well-known management protocols implemented by almost all vendors, such *Simple Network Management Protocol* (*SNMP*), Syslog, Netflow, and others. However, each vendor creates its management strategy and proprietary tools and protocols. This fact disables a vendor-agnostic management tool to usufruct a fine-grain control and visibility about these traditional network equipment.

- **Operational Costs -** Hence, these last arguments impact the operational costs for traditional network equipment buyers due to their specific tools, which demands investment in training and certifications to operate that implementation. Furthermore, these investments lock the client into a particular vendor.

Thus, the traditional networks model has suffered from some blockades that prevent from advancing together with exponential applications demands. The following topics focus on recent paradigms which try to overcome some of the raised barriers.

3.2.5 *Network Virtualization*

First Network Virtualization Phase

As mentioned before, virtualization has revolutionized the data center and computational world. Then its context has become more efficient and has opened great opportunities for the IT area.

Recapitulating the virtualization definition, is the abstraction of operating systems and applications from the physical hardware device. Bringing this definition to the network, it's also abstraction the networking nodes from physical topology arrangement, which allows the administrator to group or arrange these elements into new logical way disregarding from their physical location [26].

Although lately network virtualization has been a recurring topic, the concept has been used for a long time, namely into *Virtual Private Networks* (*VPNs*), *Virtual Local Area Network* (*VLAN*), and *Multiprotocol Label Switching* (*MPLS*). These examples of virtualization indeed brought great techniques for network administrators to isolate uncorrelated traffics, protect their integrity, group logically distant nodes, and even optimize network usage. Although the network landscape has already implemented virtualization and has been widely used protocols for a long time, network hardware devices are still very much nested, with restricted functions such as firewalls, load balancers, routers, and switches.

Novel Network Virtualization Phase

Traditional network virtualization is still essential and will continue to be used. Therefore the highly demanded scene incite modernization to the network equipment to be more flexible, scalable, compatible with multi-vendors, and more manageable. Some advances have already been happening, such as network automation through APIs and scripts developed in programming languages (e.g., Python), which promote more scalability and visibility.

Virtualization has brought lots of benefits, still, also inefficiencies surged. With server VM proliferation and mobility, traffic flow between them becomes very scattered. In contrast, the network appliances kept stationed at the edges of the data centers [26]. In large data centers and cloud providers, VMs in different networks and VLANs are used to communicate with each other. So the VLAN has some matters, first one is the number of VLAN IDs, which is limited to 4096. Another problem is that VLAN is a layer 2 protocol and relies on Spanning-Tree Protocol (STP), and to avoid loops, it disables ports, decreasing the network capacity in this way.

To overcome these communication issues, another network virtualization protocol was conceived, the *Virtual Extensible Local Area Network* (*VXLAN*). VXLAN borrows some encapsulation features from VLAN, which encapsulate the Ethernet frames within layer 3 packets. More specifically, the VXLAN's technic comprises in encapsulation of MAC address in *User Datagram Protocol* (*UDP*) and send this packet throughout layer 3 network. This feature provides the layer 2 abstractions to VM, allowing them to be hosted anywhere, even in different data centers. The fundamental requirement is the connectivity between tenants. Thus, the VXLAN creates an overlay network that VMs don't need to be aware of the physical network's (underlay) minutiae [26]. This overlay is based in tunnels, called *VXLAN Tunnel EndPoint* (*VTEP*), that a hardware or software element instantiates these tunnels and makes the encapsulation and de-encapsulation job [28]. The VTEP also maps the local LAN segment of each endpoint with a bridging transport IP network, and then a VXLAN segment is created. Through this segment, the endpoint's MAC addresses are learned via ARP protocol and also mapped into VTEP. Down below, in Figure 4 there is a VXLAN simple exemplification of its architecture.

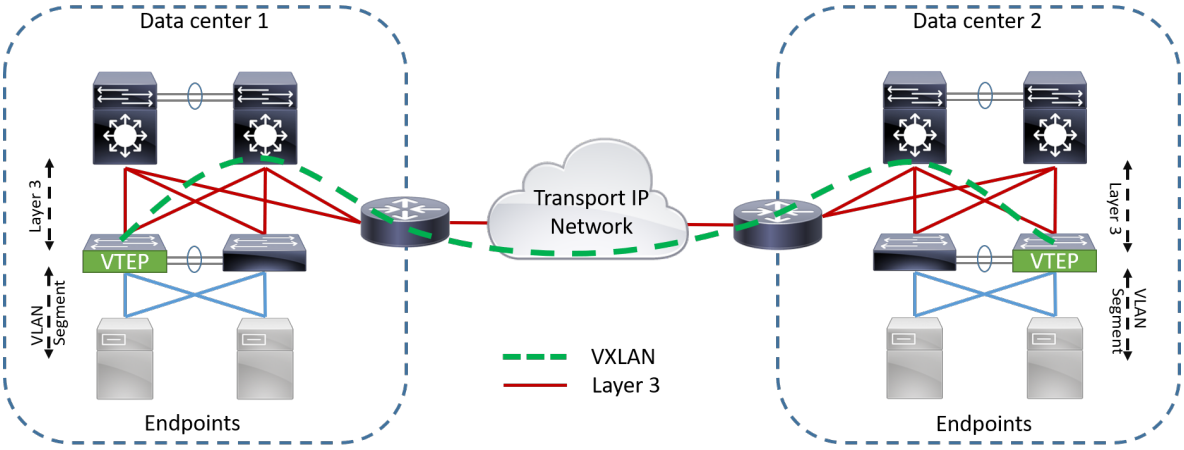


Figure 4: Simplified VXLAN architecture (adapted from [29]).

Moreover, in Figure 5 there is the packet format after VXLAN encapsulation. This packet gets a total of 50 (up to 54) bytes long, where the L2 Ethernet frame is encapsulated with a VXLAN header and forms an overlay overhead part. The underlay is also provisioned, composed of a UDP header and the outer IP and MAC addresses belonging to VTEP interfaces.

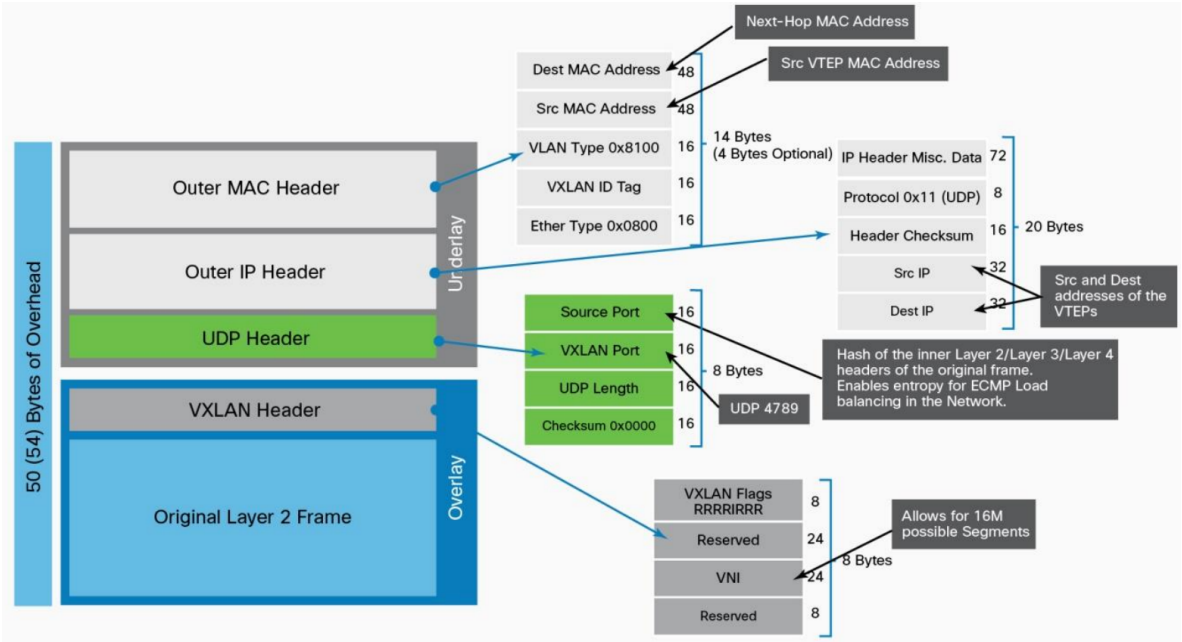


Figure 5: VXLAN's Packet format [29].

Despite these ultimately improves, the actual moment suggests virtualization analogous to data centers has done, which abstract the OS and applications from the hardware. So the NFV defines a similar approach concerning the use of generic hardware, a Commercial Off-The-Shelf (COTS), that have all physical capabilities regarding I/O and CPU. The *European Telecommunications Standards Institute* (*ETSI*) and *Industry Standards Group* (*ISG*) established some recommendations about NFV, which proposes the virtualization of functions from dedicated network equipment like firewalls and load balancers into VMs, allowing fast, flexible, and ubiquitous deployment [30]. Moreover, the network functions become a piece of software, called VNF, portable, and easily upgradeable, rather than specialized network hardware that needs a physical replacement when it gets obsolete and performs just one central role. These scalability, agility, and efficiency features promote great opportunities for innovation in terms of the technology itself and business model, enabling providers to offer novel services and products.

Virtual Network Functions

One of the main elements of the NFV environment is the VNF. The traditional network functions from firewalls, load balancers, routers, and other network equipment are virtualized, and a VNF can perform each of these virtualized roles. So the enterprises are replacing several boxes, each with its own set of functions, for a consolidated piece of hardware as a standard x86 server with multiple VMs sharing the server resources.

Into a new network architecture, different vendors can provide one distinct VNF. This fact eliminates the proprietary hardware constraints, such as the cost of upgrading a box and its inherent spending.

Furthermore, the VNFs bring flexibility, eliminating the location constraint of some network equipment. It allows to scale and deploy these functions near demanding places, e.g., central offices, remote branch offices, data centers, Point-of-Presence, etc.

Service providers and mobile operators are considered catalyst players for network virtualization due to their interest in delivering faster time-to-market and innovative services and products to their clients with agility and lower costs.

The freedom from these constraints imposes a well-defined architecture framework to allow a concise coupling among these heterogeneous elements.

NFV Framework

The traditional network architecture is well-defined due to vendors' hardware and software being developed to be tidily integrated into each other. On the other hand, in NFV's world, many vendors specialize in software and network functions developments, unlike much other hardware construction. This decoupling of solutions conception forces a very detailed and standardized framework with multiple touchpoints to be followed by software and hardware vendors as guidance to guarantee their compatibility and integration [27].

The service providers, who have this recurrent yearning for agility, innovation, and standardization, formed the *European Telecommunications Standards Institute* (*ETSI*), and in 2012 created a consortium, the *Industry Standards Group* (*ISG*) to standardize the NFV into a framework. This framework establishes guidance to hardware and software vendors to guarantee interoperability between them.

One of the established key principles was to leverage the virtualization throughout IT components into COTS. To achieve this principle, they suggest some recommendations, which are [31]:

- **Decoupling -** Abstract and decoupling the network-specific functions from dedicated hardware, becoming these functions as software allowing portability to a standard COTS.
- **Flexibility -** Open the possibility to deploy these functions in different places looking forward to the most efficient one, into a cloud, Central Office, data center, or an NFVI-PoP, depending on the business desired outcome.
- **Dynamic Operation -** Providing granular control and monitoring of operational parameters of network functions state.

After these statements, a high-level NFV framework was defined. The framework is divided into three main blocks, called infrastructure block, virtualized function block, and management block., which are well-defined by ETSI as following the Figure 6:

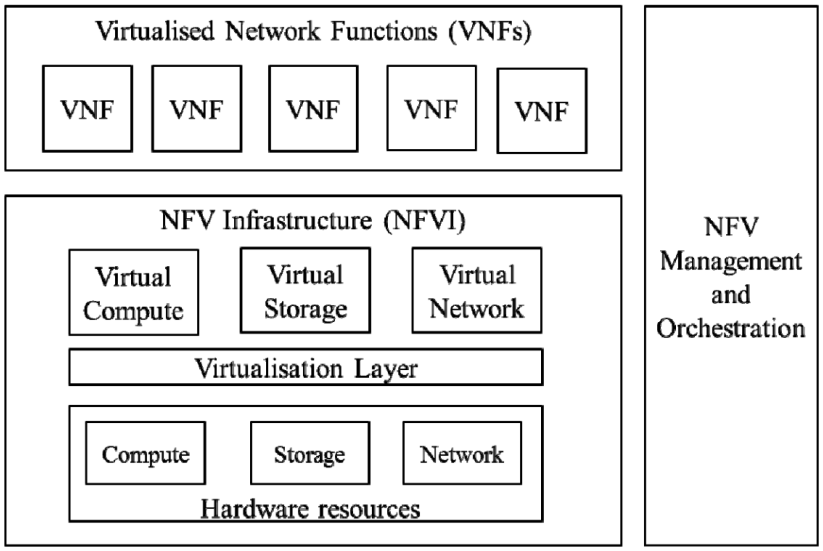


Figure 6: ETSI NFV framework, high-level overview [31].

A brief explanation of each block is described below:

- ***Network Function Virtualization Infrastructure*** **(*NFVI*) -** The NFVI block is concerned about all the computational, storage, connectivity hardware resources that are virtualized hosting virtual machines, composing a pool resource to be demanded by the upward and sideward blocks.
- ***Virtual Network Function*** **(*VNF*) -** This block contains the network functions which were virtualized into software, and are hosted into VM offered by NFVI block.
- ***NFV Management and Orchestration*** **(*MANO*) -** MANO is an adjacent block that has the management role and interacts with both previous blocks, orchestrating the allocation of resources and demands all over the infrastructure lifecycle.

In this framework, the software is so decoupled that one VNF could require hardware COTS resources from different pieces of hardware. As well, a Network Function (NF) can hold multiple VNF. As an example of a firewall NF, it performs many features, like NAT, ACL, Packet Inspection, etc. A different VNF can operate each one throughout the forwarding graph between two endpoints flow. in Figure 7 there is a representation about this abstraction and independence of software overlay from hardware underlay. Moreover, the forwarding graph is also shown, and for each forwarding graph, the MANO could instantiate a different overlay graph and hardware allocation depending on the actual demands and underlay usage.

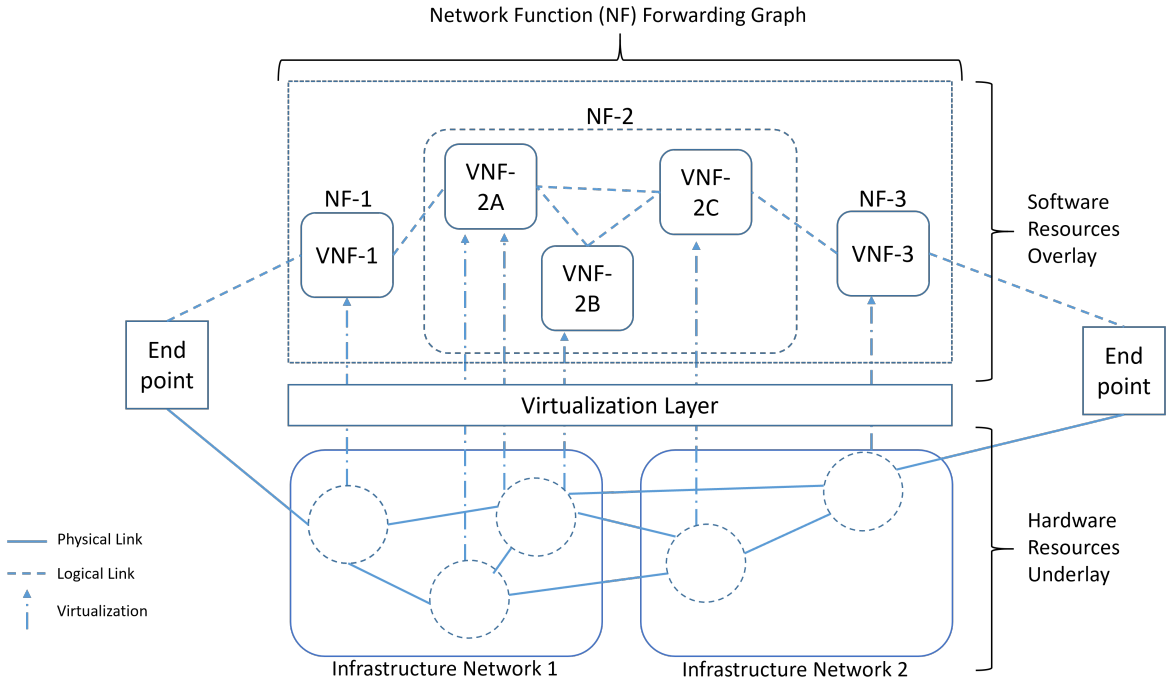


Figure 7: Resource allocation to VNFs, overlay and underlay abstraction, and forwarding graph [31].

When it is said that MANO performs the allocation task, there is a set of specific components inside MANO that has their role in this enginery. Each of these components is aware of each layer, VNF, NFVI, and a topper layer of Operational and Billing Support System (OSS/BSS).

The ETSI NFV architecture defines these deeper functional blocks, the Virtualized Infrastructure Manager (VIM), Virtualized Network Function Manager (VNFM), and NFV Orchestrator (NFVO). For each of these management and orchestration tools and managed blocks, there are reference points with well-defined specifications to ensure the already mentioned compatibility among multi-vendors deployment.

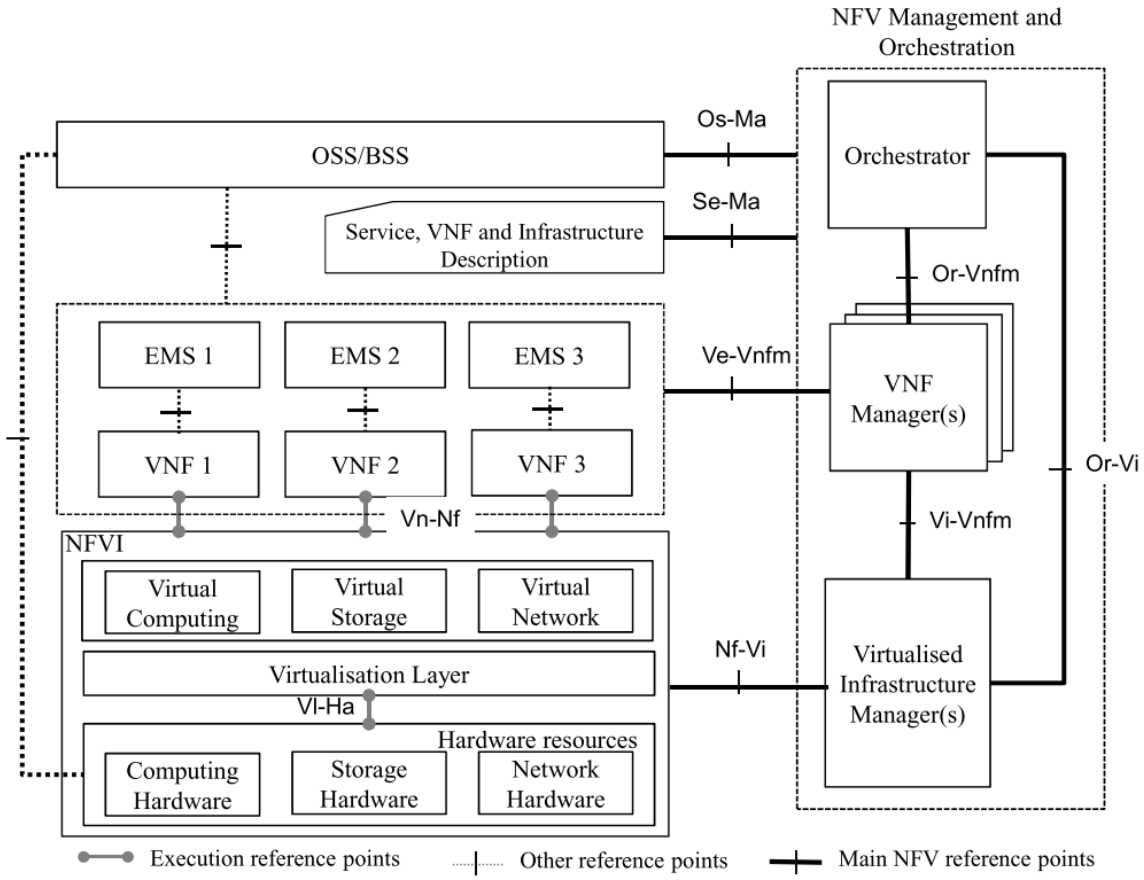


Figure 8: ETSI NFV framework reference points [31].

These signed reference points at Figure 8 are described below:

- **Os-Ma -** This reference point (RP) is positioned between NFVO and the OSS/BSS blocks. It handles all of the management data related to usage, accounting, and analytics for NFVI.
- **Se-Ma -** This one refers to VNF deployment, forwarding graphs, and the NFV infrastructure information model.
- **Ve-Vnfm -** This RP regards VNF lifecycle management, including the exchange of configuration and state information.
- **Nf-Vi -** Used to allocate a virtualized resource answering demands from upward layers. Also, inform about the state concerning the virtualized infrastructure
- **Or-Vnfm -** This is responsible for handling the VNF Managers' requests for authorization, allocation, validation, and reservation. Moreover, it provides configuration information to the VNF manager to organize the good function of VNF forwarding graphs.
- **Or-Vi -** This is due to setting a virtualized hardware allocation or reservation by the orchestrator, and for configuration and information state exchanges between both blocks.

- **Vi-Vnfm -** This RF sets the information exchange between VNFM and VIM to update requests for VM running a VNF.
- **Vn-Nf -** This RF is related to setting the requirements regarding the hardware lifecycle, performance, and portability to attend to the VNF's demands.
- **Vl-Ha -** This RF establishes an interface between the virtualization layer and the hardware's pool resource to manage them, independently from the hardware platform, to offer virtual resources to VNFs.

These show the high-level specification for the industry, in general, could apply the NFV concept into their sector, like broadband, mobile network, and many other segments. This first lead from the ETSI NFV framework stimulated new initiatives combined with SDN architecture to be developed in those sectors. For example, the Broadband Forum is a consortium that intends to modernize the Central Offices to take advantage of these new paradigms of virtualization and "softwareization". All these initiatives promote the modernization of technology itself and enable the providers to deliver better service and solutions to their clients with more efficiency, lower costs, customization, agility, and flexibility.

Summary

The NFV framework has the objective to standardize the relationship among multiple components of an NFV environment. ETSI framework did this first important step which enabled many vendors to create their implementation and NFV's component blocks following guidance to allowing compatibility to each other vendor's implementation.

One of the most benefited players is the service providers, who won't be locked in a specific vendor, allowing them to customize the network implementation accordantly to their business needs, or even develop their own homemade solution, thanks to the opensource initiatives of this context.

3.3 Software-Defined Network (SDN) Paradigm

The last section presented some constraints related to the traditional IT infrastructures and network architectures. More specifically, the paradigm of dedicated equipment deploys a restricted function versus COTS (commodity platforms), which can host different virtualized functions depending on the current demands.

Additionally, Software-Defined Networking reinforces the paradigm of virtualized infrastructure, functions, and the network itself; hence it's complementary to NFV and VNF topics. SDN has the main characteristic of the network control plane detachment from the data plane, which these planes are coupled in traditional devices.

3.3.1 *Introduction*

Traditional network devices, independently of their network role, have some common functional planes embedded into them. These planes communicate with each other using proprietary or open APIs. Each plane has specific roles, which are described as follow:

- **Control plane -** This plane is responsible for deciding what happens with data flow if it is allowed, denied, queued, or any other manipulation to that.
- **Forwarding plane -** This role implements the decision from the control plane. Hence it will discard, forward, queue the data traffic based on control plane instruction.
- **Management plane -** The management plane is responsible for dealing with the network device itself regarding configuration, fault monitoring, and resource management.
- **Operation plane -** The operational plane deal with the state of all of the components of the network device, retrieving health information that is constantly used by the management plane.

Exemplifying these planes, in router device, the configuration applied, which refers to management plane, results in a defined routing protocol to be used in some interface. This routing protocol belongs to the control plane, which will set the instructions for moving the traffic forward, thus translating into a routing table, such as a Routing Information Base (RIB). After that, when data traffic arrives at the router, the forwarding plane will consult the settled directions in the routing tables. If there is proper instruction on how to handle the traffic, the forwarding plane will deliver the data to the appropriate place. Finally, all over this process, the operational plane will keep logging and monitoring the device activities.

Though the planes are confined into each device and distributed, some management tools add an upper layer and centralize the management and monitoring of these devices using some proprietary or non-proprietary protocols, like SNMP, NETCONF, and Netflow, for instance.

SDN Principles

As already presented, the traditional network devices have their control plane in a distributed model. However, this approach could bring some constraints. This distributed control could be inefficient for topology elements to know the big picture in large and complex deployments. Protocols can be optimized if network elements get the state from another node. For example, QoS applications and security measures could be more effective when is known the state of involved nodes for a specific data flow [27].

Due to this fact, SDN proposes the approach to decouple the control plane from network devices, offloading them and centralizing the controlling task to a central device or a cluster.

This central entity is called the SDN Controller, which has a complete state of the controlled topology and, based on that, can make a better decision to define the instructions to the nodes. Although the controller is a central element, it could be set as a cluster, performing a high availability deployment and scaling as needed. Moreover, it can be placed into the COTS infrastructure, and NFV and VNF can be deployed.

Since the control plane moved out from the forwarding/data plane in network devices, it became merely a piece of forwarding equipment that follows the instructions from the controller. Thus, in the network node with no more control plane and its RIB, it left the in data plane just *Forwarding Information Base* (*FIB*). In this way, after the centralized control plane formed its RIB, it passed the nodes' instructions to populate their FIBs. When unknown traffic arrives on the node's interfaces, it's usually sent to the control plane to decide what action should be taken [32].

SDN also defines the communication between the controller, forwarding, and applications planes should be established through open industry standards. Consequently, this fact allows vendor-agnostic deployments, brings more flexibility, and lower costs implementations. Based on these, the SDN communication architecture between layers and planes is subdivided into Northbound and Southbound protocols/API. in Figure 9, there is a graphic representation of the relationship between these planes and layers into SDN architecture.

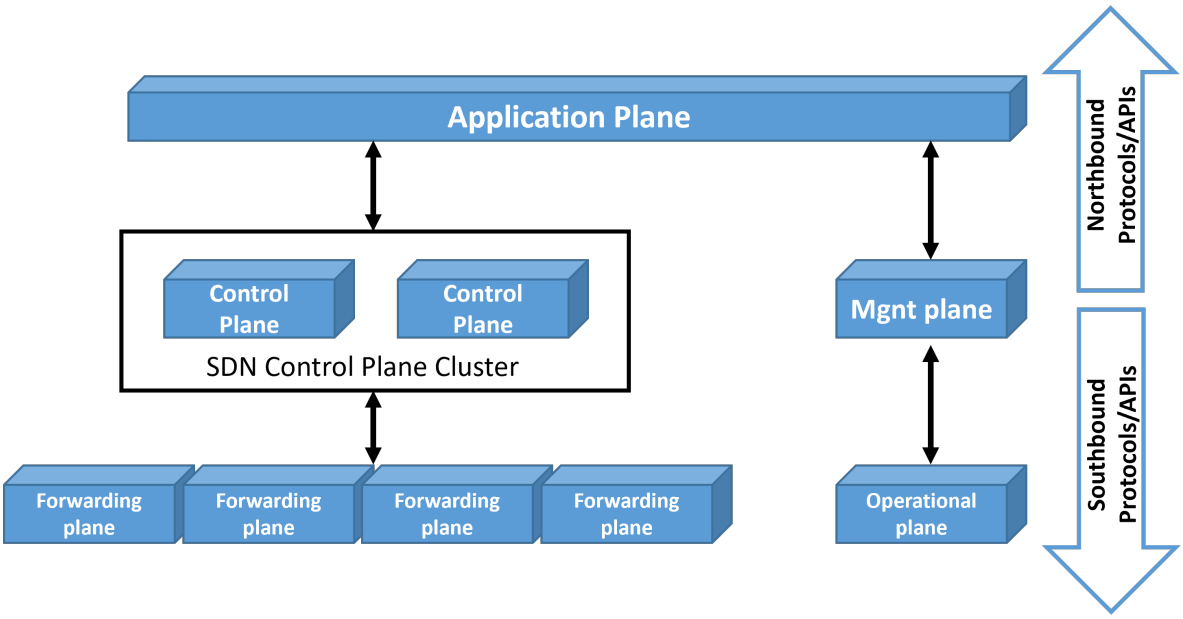


Figure 9: SDN Planes and Northbound/Southbound Protocols [27].

3.3.2 *Benefits and Advantages*

SDN brings a new paradigm that uses concepts from virtualization, cloud environments, software development, and open-source. Initially, SDN didn't turn the service providers'

and vendors' attention to it, because the problems that SDN tries to resolve weren't critical as nowadays. However, this scenario has changed, motivated by massive data consumption and more complex applications, the network industry struggled to become more flexible, programmable, open, scalable, and less expensive.

Programmable and Automated Network

The necessity for a more agile network in deployment, scalability, recovery, and lower operational costs naturally forces the network to become more programmable. This means that some traditional ways of managing, operating, configuring, and maintaining the network, mostly made manually or with much human intervention, are obsolete and should be replaced by automated and programmable processes.

Thus, the SDN brought the ability to attach customized software applications to the network, which can perform different roles automatically depending on the network behavior and necessity. Since the SDN controller has a holistic view of the network, it can call each application as needed in an automated way through northbound protocols and APIs.

Virtual Networks

As well as servers' world was optimized with virtualization concepts, which opened great opportunities and progress, this also applied to the networks. Some benefits that were already mentioned before in Section 3.2, and using some of them, for example, the abstraction of physical network hardware and create logical segmented layers, in other words, overlay networks. These overlay nets could be considered as slices. One could have their own forwarding rules applied to a set group of users and could consume different combinations of applications from the application plane, or even VNF and NFVs.

Therefore, this creates a scalable environment, which COTS hardware can be used and grow as needed. Further, the hardware underlay can be shared for multiple tenants, which is an excellent characteristic for service providers and network operators.

Vendor Agnostic Architecture

The traditional best-in-class network solutions usually are proprietary, and as much as they implement standard protocols, each of them deploys in its own way, becoming an operational issue imposed by vendor restrictions. Furthermore, often the compatibility with other vendors might be limited.

SDN architecture promotes and uses standard and open protocols, eliminating the vendor's lock-in and limitations. Moreover, the new SDN's proposed architecture of decoupling the control plane from the data plane consolidates the network brain into a central entity. The

data plane equipment has to receive instructions and report through a common and standard API/protocol. Thus these API/protocols should be open and vendor agnostic.

Unburden of Network Devices

The central intelligence of the traditional network devices belongs to the control plane, which consequently consumes a significant amount of computational resources to calculate the multiple protocols algorithms. Hence, this usual computational usage is offloaded from network nodes to the SDN controller in the SDN architecture. This approach turns the network node into a simple data forwarder, and when it needs an instruction, route, rule, etc., the SDN controller is called.

In this way, the controller performs the demanding computational workload, which can run into COTS hardware. This can significantly reduce the CAPEX and even more regarding the network nodes that don't need specific ASIC chips to control plane processing.

3.3.3 *Protocols*

Southbound Protocols

The southbound communication between the SDN controller and the forwarding devices is divided into two groups. First is the category in which the control plane talks directly with the forwarding plane. These are called SDN control protocols. On the other hand, in the second, the control plane influences the forwarding plane through the management plane [27].

SDN Control Plane Protocols

The main examples of SDN Control Plane Protocols are the OpenFlow and Path Computational Element Communication Protocol (PCEP).

- **OpenFlow -** The OpenFlow was the first opensource SDN control protocol to communicate the Controller instruction to the forwarding devices. Using the OpenFlow(OF), the controller can alter the forwarding table of the OF-enable switches. The OF-enable switch should have three components, the Flow table, Secure Channel, and OpenFlow protocol. The Flow table provides the rules for forwarding the data, the Secure Channel ensures secure communication between controllers and forwarders, and the protocol provides communication with external controllers [14].

- **PCEP -** This protocol is often used for MPLS and Segment Routing implementations. It uses a Label Switch Path and sets the forwarding rules into Path Computation Client (PCC) defined by Path Computational Elements (PCE) after PCC's requests [7].

Management Plane Protocols

On the management plane hand, there are protocols to configure and monitor the network devices which influence the forwarding plane. Additionally, there are network devices that are SDN enable. However, they still have their control plane attached to them. Thus some protocols could influence their behaviors. A few examples are the *Network Configuration Protocol* (*NETCONF*) and *Representational State Transfer Configuration Protocol* (*RESTCONF*).

- **NETCONF -** The NETCONF protocol is an initiative from *Internet Engineering Task Force* (*IETF*) to support network configuration through a programmatic way and in a client-server model, where the server is the network device to be configured and the client is the application. It was defined syntax, and the vendor could add their add-ons. This syntax is formatted using *Yet Another Next Generation* (*YANG*) data model [33].

- **RESTCONF -** This is quite similar to NETCONF, and both use the YANG and client-server model. However, the RESTCONF had the RESTful API as inspiration, relying on simple operations like GET, PUT, POST, and DELETE methods.

Northbound Protocols

The northbound interface (NBI) is the communication between the application layer and the SDN controller, and it is also a bridge for applications and management and control planes.

The communication between these planes is simple communication between software applications. This simplicity allows using RESTful APIs or programming languages like Python, Golang, Java, Ruby, or C++. Comparing the Southbound Interface (SBI) with NBI, the southbound is more defined and oriented to SDN controller and network devices communication (mainly through OpenFlow protocols). In contrast, the NBI is more flexible and accepts multiple protocols and programming languages to build applications that run into the network. Thus, this permits different kinds of applications and SDN controllers.

3.4 SDN-ORIENTED CENTRAL OFFICE ARCHITECTURE

In Section 3.1 was introduced some concepts of Passive Optical Network and its components, furthermore, was mentioned the evolution and opportunities of Central Offices have been meeting with NFV, SDN, and Cloud paradigms.

Actually, these opportunities are driven by the actual demands from application and extensive consumer usage, which pressured the vendors and service providers to offer more flexible, agile, scalable, and affordable solutions. Due to this, consortiums and industry organizations have been formed by service providers, vendors, and the community, in general, to focus on leading initiatives to promote innovation to IT. Regarding the telecommunication and broadband sector, there are *BroadBand Forum* (*BBF*) and the *Open Networking Foundation* (*ONF*) organizations, who are the flagship in this sector.

3.4.1 *BroadBand-Forum (BBF)*

The BBF is a non-profit organization composed of industry's broadband operators and vendors. This organization has some working groups divided by projects as 5G, Connected Home, Cloud, and Access Networks [34].

Regarding the Central Office environment, the BBF has the "Cloud Central Office (CloudCO)" project. The CloudCO proposes an architecture redefinition of the access and aggregation networks hosted into traditional Central Offices. This redefinition includes the use of SDN, NFV, and Cloud technologies to achieve and respond to the existing broadband networks' demands [34].

Inside the CloudCO initiative, there is the *Open Broadband - Broadband Access Abstraction* (*OB-BAA*) project. This is an open-source project designed to create Northbound Abstraction Interfaces (NAI), Core Components, Southbound Adapter Interfaces (SAI) layers to permit integration with multiple management and control elements (e.g., *Business System Support* (*BSS*), *Operation System Support* (*OSS*), *Element Management Systems* (*EMS*), SDN Management and/or Control, and others) for network operators through NBI, and to connect the access network devices (e.g., OLT, ONU, DPUs, and others) via SAI [35].

To guarantee the portability and modularity of this project, it was designed to be virtualized, containerized, and use standard data models to ensure the compatibility between multiple elements and vendors. Moreover, it was outlined to be leveraged by NFVI, using the BAA functions as VNFs, and also to be compatible with legacy components where some specific network functions are still performed by physical nodes, which can be called Physical Network Functions (PNF) [36].

The OB-BAA architecture diagram illustration can be viewed down below in Figure 10, which has the BAA layer with Northbound/Southbound layers and interfaces. It also has

other services, i.e., Performance Monitoring, vOMCI Proxy, Control Relay, and finally examples of attachments to southbound (e.g., Access User plane) and northbound (e.g., Management and Control elements).

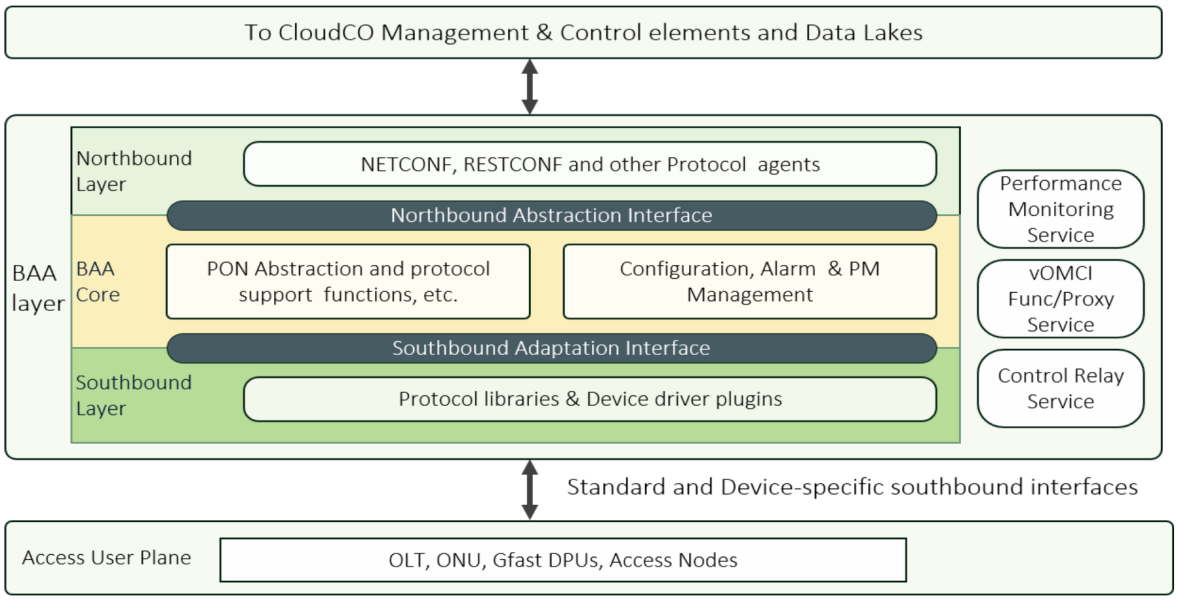


Figure 10: OB-BAA architecture layers [35].

BAA Core

The BAA Core can delivers a set of functions that could be disaggregated from traditional central office elements into PNF or from legacy devices and be performed by BAA Core under an NFVI substrate into the CloudCO environment.

BAA Layer also implements northbound and southbound interfaces to enable communication between SDN Management and Control components in the upper layer with external devices in the lower layer. In both northbound abstraction and southbound adaptation interfaces, implement data model, API, libraries, and driver plugins to ensure the compatibility between these interfaces with SDN M&C and the Access Nodes (AN) [35] [36].

Southbound Layer and Interface

The southbound layer implements adapters through southbound adaptation interfaces that specify the required data models to integrate the access nodes to the BAA layer. The device adapters can use the southbound protocol libraries and SAI API, or if it is needed, can implement their communication protocol [35].

Northbound Layer and Interface

The northbound layer implements protocols to establish communication between the BAA layer with multiple network management tools and orchestrators, like SDN M&C, BSS/OSS, etc. These protocols are usually NETCONF, RESTCONF, and RESTful APIs. However, it can be updated and redefined as needed.

Control Relay Services

The Control Relay service has the function to relay packets, like a proxy, from an access node to an application endpoint in the SDN M&C, and vice-versa.

This communication could flow from the Standard or Vendor Control Adapter (VCA) receiving encapsulated packets to SDN applications, like AAA, DHCP, or others. As well, the inverse path is also implemented. The OB-BAA provides this communication through gRPC/GPB interfaces, the Control Protocol Adapter for the northbound layer, and Standard Control Adapter (SCA) for the southbound layer. Also, is available a VCA in case the access nodes require some adjustment [35].

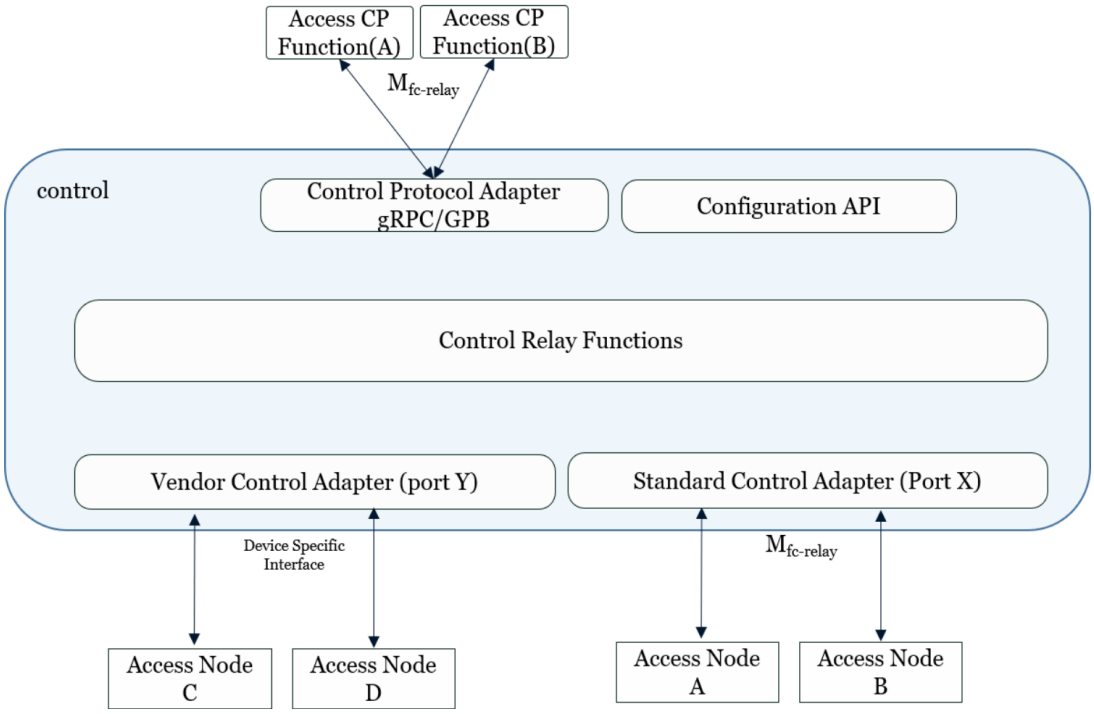


Figure 11: OB-BBA's Control Relay architecture diagram details [35].

The **Control Protocol Adapter** makes an interface between the Access SDN Management & Control application and the BAA layer, encapsulating the message into GPB payload and using the gRPC protocol for this communication, for its compatibility, easy implementation, and performance.

In the **Control Relay Functions** there is the action to receive the packets from the southbound layer through SCA or VCAs. To perform this task, the function "packetInCallback" is employed.

For **Standard Control Adapter** the gRPC also was chosen to establish the communication from the BAA layer and the access nodes.

On the other hand, the **Vendor Control Adapter** the vendors can create their protocol. However, they must implement the "packetOutCallback" function that BAA Core uses to send a packet to a device.

vOMCI Function and Proxy

The *Virtual ONT Management Control Interface* (*vOMCI*) function has the role for formatting and translating the YANG request/responses and to/from *Virtual OLT Management Function* (*vOLTMF*) into the OMCI messages [37]. Moreover, the vOMCI function is responsible for encapsulating/de-encapsulating and sending/receiving messages originated/destined to ONUs through vOMCI Proxy, and then converting the OCMI messages into understandable data vOLT Management Function (vOLTMF).

the vOMCI Proxy is deployed as a microservice. It allows the connection of multiple OLTs and the vOMCI Proxy acting as gRPC endpoints and proxies their communication with the vOMCI Functions. It is helpful for troubleshooting and management purposes. The proxy aggregate and maintain the associations between the OLTs and corresponding vOMCI function [37].

Performance Monitoring Service

In BAA architecture, the performance monitoring (PM) is deployed by a microservice divided into the PM Collector and DataHandler Module.

The PM Collector gets monitoring data through IPFIX messages from managed pANs and formats these data to DataHandler, who subscribes to Collector.

The DataHandlers are Karaf modules that subscribe to the PM Collector stream. The DataHandler (DH) is composed of Persistence Manager DH and Logger DH. The Persistence Manager does the subscription to the IPFIX stream reformatting the data into tag & counters and updating them in a typical data lake, the InfluxDB in this case. The Logger DH is also responsible for subscribing to the IPFIX collection stream and logging it into the BAA application log [38].

3.4.2 *Open Network Foundation (ONF)*

The *Open Networking Foundation* (*ONF*) is a non-profit consortium that has the mission to catalyze the transformation of the networking industry. This consortium leverages open-source adoption, network disaggregation, SDN/NFV/Cloud solution to reach this transformation.

Service providers, industry, universities, and many relevant companies integrate the member board of ONF, and this partnership produced plenty of outcomes. ONF has led the SDN development to the networking world and has produced great Reference Design processes that have already been used in production environments [39].

The ONF has projects divided into mobile, broadband, edge cloud infrastructure, and SDN Solutions. In the broadband section, which is the main topic of this dissertation, there are the SEBA and VOLTHA projects. Both projects are under CORD which has the mission to transform the edge of the operator network into agile service delivery. Moreover, the CORD integrates multiple open source projects, delivering an open, programmable, and agile platform for network operators [40].

SEBA

The *SDN Enabled Broadband Access* (*SEBA*) is a lightweight platform based on a variant of Residential-CORD. It supports virtualized access technologies of the carrier network, like PON, G.Fast, DOCSIS, and others. It is built with Kubernetes and operationalized with FCAPSI, and OSS integration [41].

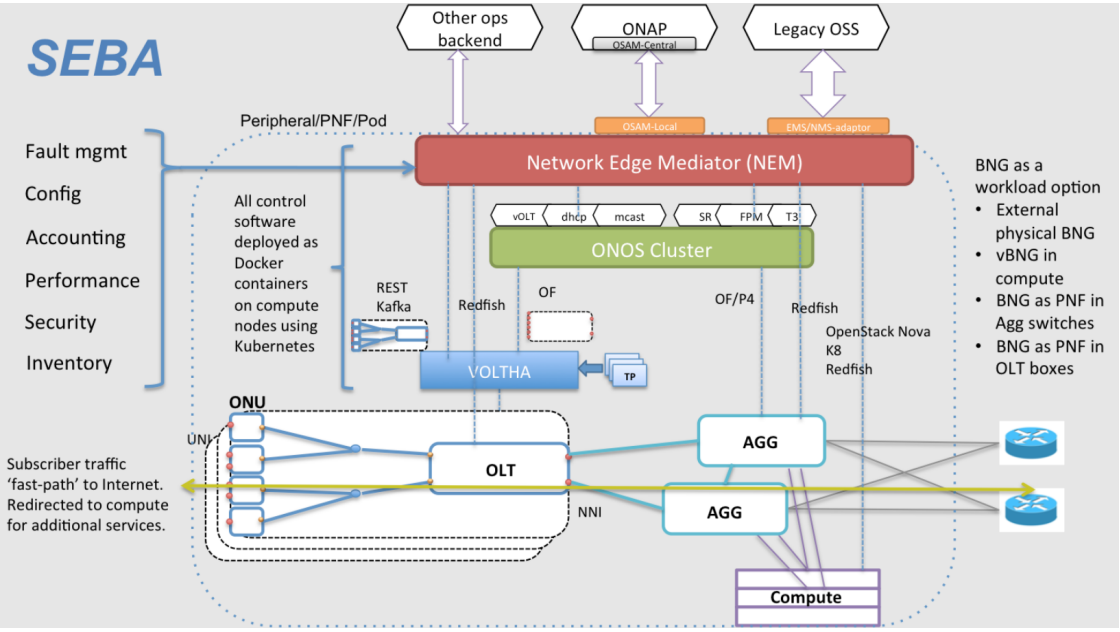


Figure 12: SEBA Architecture [41].

VOLTHA

The *Virtual OLT Hardware Abstraction* (*VOLTHA*) relies on hardware abstraction SDN's extensive usage of COTS hardware instead of dedicated and inflexible equipment. VOLTHA project applies this abstraction principle to broadband optical access networks, moving the control plane of OLT to a northbound SDN controller. Further, keeping the hardware as a white box with the data plane on the southbound side, turning the access network look like an abstract programmable switch. Moreover, it also allows communication with PON hardware devices using vendor-specific protocols through OLT and ONU adapters [42].

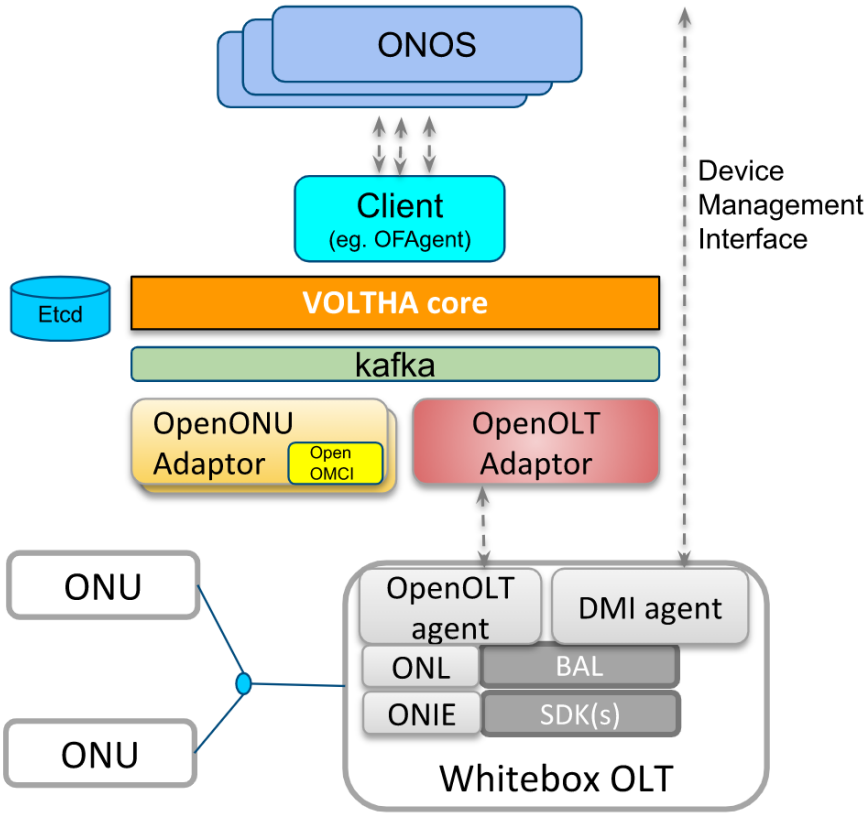


Figure 13: VOLTHA architecture [42].

3.4.3 *BBF and ONF Partnership*

The broadband networking operators are constantly looking forward to better solutions for their clients with greater cost-effectiveness and robustness. Regarding this, the BBF and ONF initiatives stand out and promote SDN-driven solutions. They both rely on open-source projects, the BBF's Broadband Access Abstraction (BAA) and the ONF's Virtual OLT Hardware Abstraction (VOLTHA) [43].

Both solutions intend to abstract some and different components of the broadband architecture, making them complementary. The BAA abstracts the control and management for any Access Nodes (AN) in traditional or "cloudified" central offices. Elseways, the VOLTHA proposes an abstraction of low-level silicon to enable vendor-neutral hardware for AN [43].

Thus, these solutions might supply the carriers and operator demands who may modernize and get the more flexible and agile infrastructure to deliver better services to their customers.

3.5 NETWORK SECURITY IN ACCESS LAYER

In computer networks, specifically in Local Access Network (LAN), the concept of elements' hierarchy that composes them is often used in its various topologies. A widely adopted hierarchical model divides into layers of access, distribution, and core.

The access layer is the one that provides initial connectivity to users, whether via a wired or wireless network. RJ45 Ethernet cables are usually used in wired networks between the user's device (PC, laptop) and the first network asset in the access layer (switches, hubs, modems, among others). For this reason of being "an entrance door" for regular users, this could be a potential "backdoor" for rogue access.

Making an analogy with an enterprise building that anyone who wants to enter the facilities must verify their identification. In other words, the first step to getting into the building facilities is to authenticate using some method, usually with a personal ID or company badge in this case.

Moreover, beyond the identification, these people should have different access levels. For example, suppose a person called Alice, a known employee from ABC Company, her role is software developer, for this, she's authorized to enter the office facilities. Still, her entrance is denied in the data center room. On the other hand, Bob is also an employee, but he is a Network Support Engineer, so his role implies accessing IT assets in the data center room. A third situation is with John, who is not an employee, he is a guest, and he came to ABC Company to have a meeting with Bob. As could be imagined, his facility's access privilege is near the lowest possible. John is just authorized to visit meeting rooms and social places. Here is an example of authorization levels.

Another important security feature is logging every access that each user had made so far, log when Alice, Bob, and John got in and got out of each place inside the Company. This is relevant for accountability.

Assuming that happened a security issue, as a stolen computer from meeting room A, an investigation procedure would be initialized by verifying who entered this room at some time range. Hence, the access control logging of this room would be a relevant tool for mitigation of this problem.

Computer network environments could have a similar approach as stated above. Likewise, as illustrated, a network context could have the same routine of entrance intention, although many networks don't implement robust security measurements at the access layer. Enterprises could often see a scene in which an Ethernet cable is plugged into an RJ45 socket, Internet and intranet access is granted instantaneously. It reveals how vulnerable a company could be if not invested in security in the access layer's first defense line.

3.5.1 *Framework Extensible Authentication Protocol (EAP)*

The *Extensible Authentication Protocol* (*EAP*) is a well-defined protocol to carrier authentication and authorization messages in a network. The 802.1X relies on it to provide this exchange message encapsulation framework, more specifically use EAP over LAN (EAPoL), in an 802.3 Ethernet, PPP, 802.11 WLAN, or other LAN protocol. EAP typically runs directly over the data link layer without requiring IP [44].

The EAP supports various authentication methods such token cards, smart cards, pre-shared keys, Kerberos, one-time password, certificates, public key authentication, and others. Here below are a few examples of EAP methods [45]. A comprehensive list could be found at [46].

- EAP-MD5
- EAP-TLS
- EAP-TTLS
- EAP-PSK
- EAP-PEAP
- EAP-IKEv2
- PEAP-MSCHAPv2

The EAP framework set a layer between the data link layer and the authentication layer, creating an interface between them and allowing multiple authentication methods to be used independently of the data link option. In this way, a new authentication method could be attached easily to the data link using the EAP layer. in Figure 14 there is an illustration of this layers interaction.

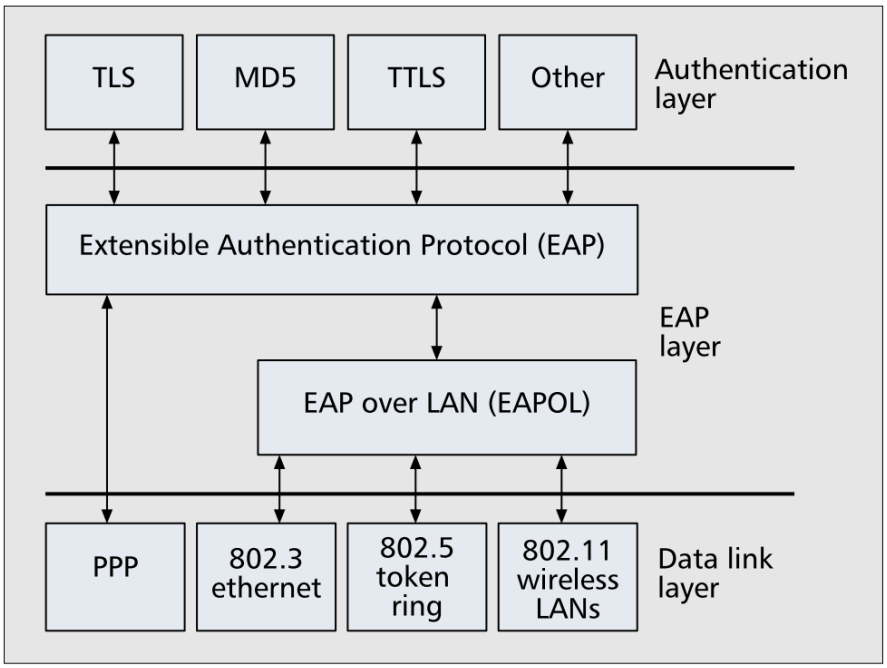


Figure 14: EAP framework layer [45].

3.5.2 *Remote Access Dial-In User Service (RADIUS)*

An enterprise environment with many users demands an access control database to manage and centralize the authentication, authorization, and even the accounting of these users. The *Remote Access Dial-In User Service* (*RADIUS*) is an AAA protocol used to provide network access [47].

The RADIUS has a client/server architecture, where the client is the *Network Access Service* (*NAS*), e.g., switches, access points, controllers, and others also called as an authenticator in 802.1X terminology. The RADIUS client is responsible for passing user information to the RADIUS server. The server is the RADIUS server itself, the authentication server in 802.1X terminology. This server is responsible for receiving user connection requests, authenticating the user, and delivering the necessary information for the client to allow connectivity [47].

The network security of communication between client and RADIUS server is granted by authentication with a shared secret not sent over the network, and any user passwords are sent encrypted, avoiding a user's password snooping [47].

3.5.3 *IEEE 802.1X Architecture*

The 802.1X is a protocol defined by IEEE that brings security at the link layer for port-based network access control (PNAC) that relies on the identity of users or/and devices that could

be trying to gain access to a network service. This protection method could prevent some unauthorize rogue devices from walking in and compromising the network [48].

Additionally, it's a protocol that belongs to layer 2. It is closer to client/user, and for this reason, could be deployed at an effective cost compared to other security implementations in upper layers. However, they could be complementary to each other.

With 802.1X, the access network devices (e.g., switches, access points) assume that all access ports are disabled, except for EAP frames that will transport the AAA messages. Until the user or device is authenticated and authorized, moreover also could be accounted for future audit. This protocol could bring some benefits for the network's robustness and security, visibility, and transparency.

Main Components of 802.1X Protocol

There are usually three elements in an 802.1X network [49]:

- **Supplicant –** Device that intent to access the network by engaging in an Extensible Authentication Protocol (EAP) communication exchange between the supplicant and the authenticator. E.g., PCs, Smartphones, CCTV cameras, ONTs, etc.
- **Authenticator –** Network equipment where the supplicant will be connected and exchange EAP messages. This device will relay authentication, authorization, and accounting (AAA) messages between supplicant and authentication server. Commonly called a Network Access Server (NAS). Some examples of possible authenticators are switches, wireless access points, OLTs, etc.
- **Authentication Server –** Element has a credentials database for supplicant authentication, authorization, and accounting. E.g., RADIUS Server, LDAP Server, Active Directory, MySQL Server, etc.

How Does 802.1X Protocol Works

The IEEE 802.1X protocol, also called dot1x for abbreviation, has five phases, initiation, authentication, authorization, accounting, and termination.

- **Initiation -** In this phase, the supplicant's access port is blocked for every frame and packet, except for EAP frames. So when the supplicant intends to connect to the network, it will send the first frame *EAP over LAN (EAPoL) Start* over the data link layer, encapsulated into the EAP frame. Or even a DHCP, ARP, DHCPv6, ND, or any packet can initiate the dot1x process. The authenticator receives this frame and answers it with an *EAP Identity Request* frame, inquiring about the supplicant's identity information.
- **Authentication -** Upon receipt of the *EAP Identity Request*, the supplicant can thus initiate the authentication process in which it sends its identity through the *EAP Identity Response* message to the authenticator.

The authenticator will then forward the supplicant's identity information to the authentication server to validate this information. Messages exchanged with the authentication server use RADIUS messages using a client/server architecture between the authenticator and the RADIUS authentication server. Then, the *EAP Identity Response* message is transmitted to the authentication server as an Attribute-Value Pairs (AVP) via a *RADIUS Access-Request* message.

Authentication server when receiving the *Access-Request* message, will check the supplicant's identity attribute if there is an entry from it on the database. If there is, the server will send a *RADIUS Access-Challenge* to the authenticator, who will ask the supplicant, through *EAP Request EAP Method*, if it complies with the current authentication EAP method or not. If so, the supplicant answers to the authenticator with an *EAP Response* accepting the proposed challenge. Then, the authenticator forwards the response over the *RADIUS Access-Challenge EAP Response* message.

Note: *The messages exemplified above are regarding the EAP method EAP-MD5, which is one of the most straightforward methods. Other methods might have additional procedures.*

- **Authorization -** After completing the previous phase, the authentication server sends the *RADIUS: Access-Accept: EAP-Success* message to the supplicant via RADIUS message to the authenticator, similarly to what was done in the other previous steps. Into *EAP-Success* message, could contain attributes regarding the supplicant's access permission, like which associated VLAN, a specific Dynamic Access Control List (dACL), others appropriated features.

- **Accounting -** The accounting feature is optional, but it is considered good practice to configure it for auditing reasons. Since accounting is an optional setting, it must be explicitly configured when you want to use it. The message that starts the process is *RADIUS: Accounting-Request: Start* and is started by the authenticator. This initial message is sent to the authentication server, and it can contain various information inserted in the AVP field about the session established with the supplicant. Below is the Table 7 with the main attributes (AVP) that may contain in this message:

Table 7: Dot1x Accounting Attribute Value Pairs (AVP).

ACCOUNTING ATTRIBUTES	OFFICIAL ATTRIBUTE'S NAME
Attribute[1]	User Name
Attribute[4]	NAS IP Address
Attribute[5]	NAS Port
Attribute[8]	Framed IP Address
Attribute[25]	Class Group
Attribute[30]	Called Station ID
Attribute[31]	Calling Station ID
Attribute[40]	Acct Status Type
Attribute[42]	Acct Input Octets
Attribute[43]	Acct Output Octets
Attribute[44]	Acct Session ID
Attribute[45]	Acct Authentic(Start)
Attribute[46]	Acct Session-Time
Attribute[49]	Acct Terminate-Cause ()
Attribute[61]	NAS Port Type

- **Termination -** The termination can occur when the supplicant sends a *EAPoL-Logoff* packet to the authenticator, then the port status change from authorized state to the unauthorized state. Moreover, if accounting is enabled, the authenticator also sends a termination instruction to RADIUS to close the accounting session.

in Figure 15 there is a visual summary of described process above.

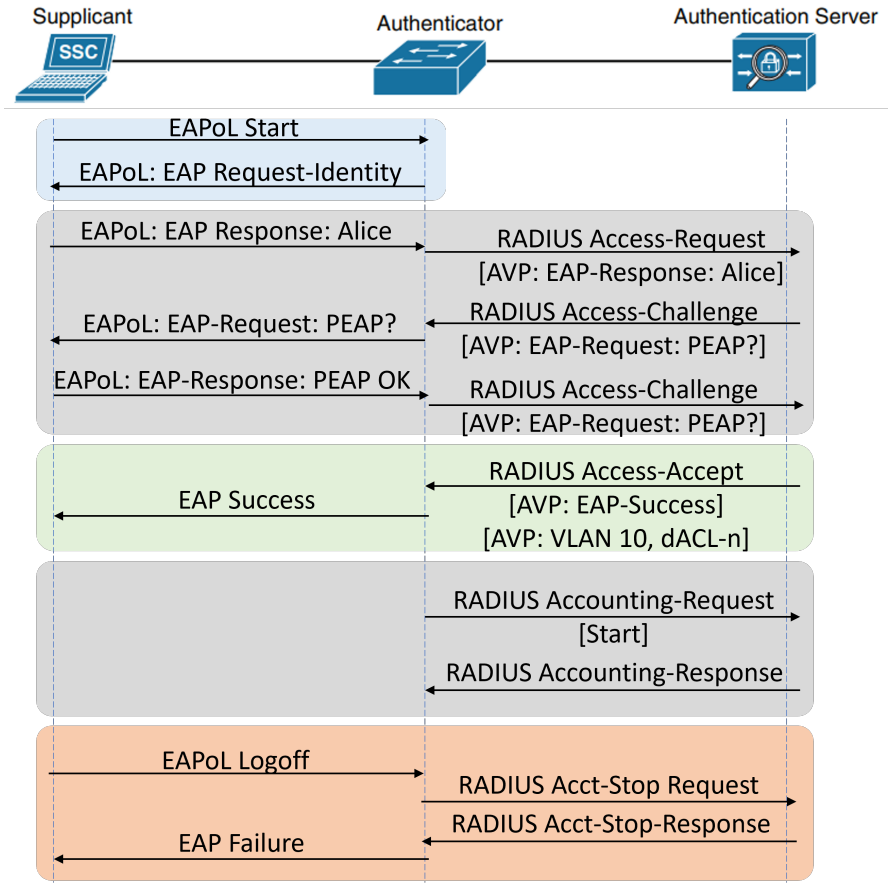


Figure 15: Dot1x process illustration.

3.6 SUMMARY

This chapter briefly introduced the PON technology and its evolution over the years. Furthermore, the main components and characteristics were mentioned, highlighting the core aspects and vantages over other broadband network solutions. The relevance of PON for broadband networks and the perspectives related to this technology's infrastructure also could be observed, suggesting the next infrastructure generations could leverage the business outcomes.

The chapter also discourse about how IT has evolved a lot through the years, so part of this evolution relies on how the computational power is used more efficiently. A relevant component was highlighted, virtualization, which is one relevant component to reach this efficientness. Further, how this concept has influenced the datacenters, Cloud solutions, and networks.

Was also conceptualized the SDN paradigm and how it is becoming relevant and reaching the broadband networks. Then, some proposed architectures by the most relevant con-

sortiums formed by services providers, network operators, and industry are proposing the network transformation through the SDN concepts adoption.

Finally, the chapter described the IEEE 802.1X protocol, which is implemented in this project as a VNF to support the AAA role of a PON network into BBF architecture.

4

IEEE 802.1X VNF PROPOSED APPROACH

This chapter describes the Dot1x proposed approach, adopted architecture, and procedures executed in the project. It is relevant to remark that all the work developed in concern of this masters' thesis is conducted in an enterprise environment, namely with ***Altice Labs*** and inserted into an internal Software-Defined Network project. The theme of this work is related to issues that are correlated with ***Altice Labs'*** routine and future possible projects or products. Due to this fact, it is disclaimed that some terms, names, procedures, or codes could have been simplified, replaced, or hidden if these are considered internal-only or sensible information.

The practical experiments and tests are divided into three phases. Each one has evolved from the last phase regarding the simulation environment, code maturity, and integration with the architecture where this project is embedded.

The first section of this chapter has pointed the architecture adopted for conceiving the project as a whole. Further, in next section, are listed supporting tools and the motivation for their use. Afterward, the structure of each phase for this project is covered after some limitations and scalability issues were observed during the project's development. Subsequently, the performance evaluation metrics for these scenarios were detailed and measured, and their relevance.

4.1 ADOPTED ARCHITECTURE

This project proposes to explore the network virtualization concepts, namely the Virtual Network Functions, for this goal was selected the *Passive Optical Network* (*PON*) architecture, including its new initiatives regarding SDN applicability into this context. Even more specifically, the IEEE 802.1X (dot1x) protocol was chosen as the network function to be virtualized into the PON domain. The dot1x has multiple authentication EAP methods, authorization steps, and accounting attributes. However, a more surface authentication method was chosen, namely the EAP-MD5, for initial project development. In addition, other techniques were listed, but for lack of utile time, it was settled as future works, as described in conclusion Chapter 6.

As mentioned in Chapter 3 the Central Offices, which aggregate the PON and many other technologies, have been evolving to adapt to modern demands and take advantage of new opportunities. Was also cited few organizations who are the flagships in this scenario, namely the *BBF* and *ONF*, with projects *OB-BAA* and *SEBA*, respectively. Both organizations are composed of many interested operators, companies, and industry, who gather to develop new technologies and the next steps towards to the evolution of broadband technologies. Into this project context, the OB-BAA's architecture was selected for this work, which was already explained in Chapter 3, due to Altice Labs' partnership and collaboration with the BBF group.

The OB-BAA architecture proposes a framework with well-defined reference points that could be followed by vendors, operators, and Infrastructure Providers to take advantage of the solution and could be integrable between them. This framework enables them to develop part of architecture elements and take for granted that it will attach and interoperate with other parts developed for other organizations.

4.2 SUPPORTING TOOLS

To achieve the proposed approach, a few tools helped in this task. First of all, a simple network topology was emulated for starting the preliminary studies of IEEE 802.1X. Then, furthermore, software development started, using a specific performative programming language suitable to adopted architectures. Afterward, the network elements have become virtualized and containerized into microservices, and the software development shifted to this environment. This dissertation leveraged some tools for monitoring, automating, troubleshooting, and reporting throughout these phases.

4.2.1 *Network Emulation Tool*

At the first phase of the tests, the goal was to understand the main concepts and phases of protocol 802.1X. This project selected one elementary topology and a typical scenario for this step: three dot1x actors, the supplicant, authenticator, and server authentication. An example could be seen in Figure 16 below.

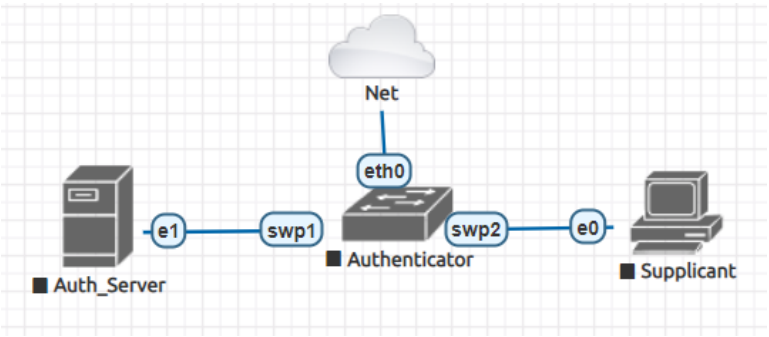


Figure 16: Simple Dot1x topology.

The network emulation tool *Emulated Virtual Environment Next Generation* (*EVE-NG*) [50] was selected to create this environment. EVE-NG is a Linux Ubuntu virtual machine that needs to be installed on a hypervisor such as VMware (recommended virtualization platform), or even installed on Bare Metal. After the installation process, images of the elements that will be in the simulation environment are included. The elements that can be emulated in this tool are usually network devices, such as routers, switches, servers, load balancers, firewalls, management tools, etc. Still, there is also the possibility of emulating operating systems like Linux and Windows in several distributions. These images can be of 3 types. It is a network emulator with a wider range of support for images of network device operating systems. These images are Dynamips (Cisco IOS emulation), IOL (IOS on Linux as known as IOU), and finally, QEMU images, which include Linux, Windows, and several other operating systems.

The entire initial configuration of the EVE-NG is done via the Command Line Interface. The inclusion of the images can be done via the SCP protocol. It is transferred as QEMU images, for example, to the EVE-NG virtual machine in its indicated directory.

The tool itself is clientless, which is not required additional software installation like other emulators. You need to access its GUI interface via HTML5 with its IP address. Then the entire process of creating the topologies and simulation is done via this graphical interface. To access, configure, and use the network elements of the standard topologies can be done via VNC or Telnet, depending on the type of the emulated node. In addition, the emulator allows the Wireshark application to capture packets in any network interface of the topology elements, which can be of great value for the research that is intended to be done.

Reason for choice

Among the reasons for choosing EVE-NG is that it has a freeware version, which allows the inclusion of several open operating systems such as Linux and its various distributions. Licensed operating systems are not intended to be used within the scope of the project. Another determining factor in choosing this simulation environment is building a network topology analogous to a realistic setting, with physical connections between the topology elements

using their respective network interfaces. In addition to having an easy-to-use graphical interface, which allows a clear demonstration of the idealized topology. As already mentioned, the possibility of using Wireshark is a positive factor that helps to understand the types of traffic transmitted on different network interfaces. This tool also allows the creation of several networks connected to virtualizer interfaces, such as VMware Workstation Player, to conceive a communication of the created topology with another determined virtual machine or with the host itself with the Internet.

4.2.2 *Virtualization Tools*

As long as been discussed, the virtualization paradigm has evolved and reached a wide range of Information Technology areas. Since application to network functions, which is the primary topic of this project.

Hence this project is leveraged by the most used virtualization platform tools, the **Docker** and **Kubernetes**, which enables containerization of custom and very optimized operational systems and software applications for fast delivery [51].

These tools were used at the third phase of the project development when was supposed to leverage the test's scale, using multiple supplicants and OLT, using the Docker platform. This environment segmented into parts, which are the core elements like 802.1X VNF, authentication server, and the other BBF's architecture actors, which are contained into a computational resource pool, a *Network Function Virtualization Infrastructure* (*NFVI*), typically hosted into Kubernetes.

Reason for choice

As already mentioned before, these virtualization tools enable a great opportunity to test, develop, deliver, and put in production fastly, scalably, and saving a substantial number of computational resources against traditional infrastructure with dedicated network devices for each specific function, either compared to conventional virtual machines.

Another reason to elect these tools is that both are free and could be installed on most Operational Systems, like Linux, Windows, and macOS. Moreover, they support image container interchange between Docker and Kubernetes platforms seamlessly.

Although the benefits there was identified limitation during the project progress, which will be detailed further at the section 4.3.

4.2.3 *Development Languages and Tools*

For the software development part, one of the project's goals is to create an application that implements a network function. Such NF must be performative and scalable to meet a large number of requests. Furthermore, such network function is expected to be performed in

an *Commercial Off-The-Shelf* (*COTS*) to abstract traditional network equipment, following the concept of VNFs.

Another requirement that came with the adopted architecture, mainly related to Central Office abstraction, the OB-BAA suggests some candidates for this task implicitly. Some examples are Java, C, and Golang (GO).

Reason for choice

Golang language is a powerful language used to build great tools like Kubernetes, Docker, and Vault. Moreover, it was created by Google and used by them to solve their problems. GO is also very suitable and used for microservices. Accordantly [52] the first three kinds of developed software are Website, Utilities, and IT Infrastructure. The GO has outstanding performance due to its concurrency technique based on goroutines and channels, which is very economical in memory allocation and data. Hence, more units will process with less memory and faster than some languages like Python or Java.

Moreover, GO has a simple language syntax, well-documented codes, comprehensive libraries, garbage collected, compiled into a single binary, etc.

So, the sum of these few characteristics results in a great programming language option for networking and gRPC APIs applications for proposed BAA's architecture.

4.2.4 *Monitoring and Visualization Tools*

Regarding the monitoring and visualization tools, this project used Prometheus and Grafana. Both tools are open-source and have a big community backing up.

The Prometheus implements a dimensional data model, collecting and storing metric names and key/value as time-series data from many sources. Prometheus has a flexible query language, the PromQL, which allows querying for many visualization tools.

Grafana is a habitual candidate for a visualization tool. This tool makes it possible to create multiple dashboards for different metrics, visualizing them with several kinds of graphs, tables, and appearances. Grafana allows querying various sources and aggregating them into a single-pane-of-glass.

4.2.5 *Complementary Tools*

Was used a specific tools for sniffing packets, developing, logging, and reporting. For sniffing, the classic Wireshark and the TCPdump for Linux hosts were employed. For Golang development, the VS Code was used due to its rich base of extensions and plugins attached to it and the capability to integrate a terminal console for SSH remote access.

4.3 PROJECT PHASES

Firstly, this project was divided into phases where the objective was to understand the basis of IEEE 802.1X concepts, software development, and network virtualization. Secondly, was to evolve this knowledge to put them into practice through emulated networks and virtualized scenarios using the tools presented in sections before. Finally, reach a developed VNF application that could be attached to an NFVI, a fully virtualized platform, or even in a hybrid environment with legacy on-premise hardware together with SDN oriented setting.

Thus below is a brief description of these three phases and how they were conducted. Moreover, in Chapter 5, was discussed in detail the most relevant outcomes.

4.3.1 *Phase 1: Deepening into 802.1X protocol*

The first phase consists of the initiation of the studies around the key concepts that the project intends to approach. These concepts are the access security protocol IEEE 802.1X, the PON broadband access network, the Virtual Network Functions, and correlated subjects. Beyond this primary study, was also done the simulations in this initial phase, mainly to put in practice the 802.1X reviewed specifications. As mentioned before, the EVE-NG was used for this task, and a simple topology was made with the main actors of 802.1X architecture, the supplicant, authentication server, and authenticator. Down below there are slight specifications for each element of this scenario:

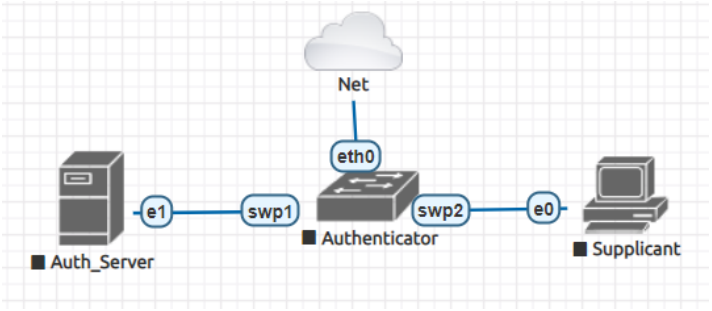


Figure 17: Initial Dot1x topology of phase 1.

Supplicant

For the supplicant, was used a *Linux Ubuntu Server 20.04* distribution, which is a lightweight operating system that doesn't consume many computational resources. To perform the dot1x supplicant role was installed the package *wpa_supplicant* to be able to initiate the process IEEE 802.1X with *EAP over LAN Start* frame and respond to the subsequent ones from the authenticator.

Authenticator

The authenticator is also a Linux kernel. More specifically, it's an open network operating system called *Cumulus Linux*, a product initially open source and software-defined oriented, bought by NVIDIA company recently in early 2020. Although the NVIDIA purchase, they still provide and support a free version, namely the *NVIDIA Cumulus VX*, which is a virtual appliance, that can be installed as a KVM-QEMU or a VM into many hypervisors, like *VirtualBox*, *VMware vSphere ESXi*, and others. In this project, the KVM-QEMU version was used, which was emulated into the EVE-NG platform. This version of the system was chosen because firstly is freeware and satisfies the test requirements, which are a network device that emulates layer 2 TCP/IP communication and mainly supports IEEE 802.1X implementations as an authenticator.

Authentication Server

The authentication server also comprises a Linux server with the FreeRADIUS package installed on it. The FreeRADIUS is a well-known freeware software that answers the project demands regarding the complexity and authentication functionalities despite its response time scalability issues observed at the third phase.

As mentioned before, a trivial scenario was used. Hence, a simple authentication method was chosen to be simulated, so the configuration file was specified to set the EAP-MD5 authentication mode, the IP from the authenticator, the shared key between them, and the supplicant's user and password.

4.3.2 *Phase 2: Proxying EAP messages with Golang*

After the introductory first phase around the Dot1x characteristics, personas, and roles, we introduced the first elements of Golang development in the next step. To do so, a new component was introduced into topology, which is a frame EAP proxy called OLT-App. This proxy stays listening to EAP frames from supplicants and forwards them to the authenticator, as can be seen in Figure 18.

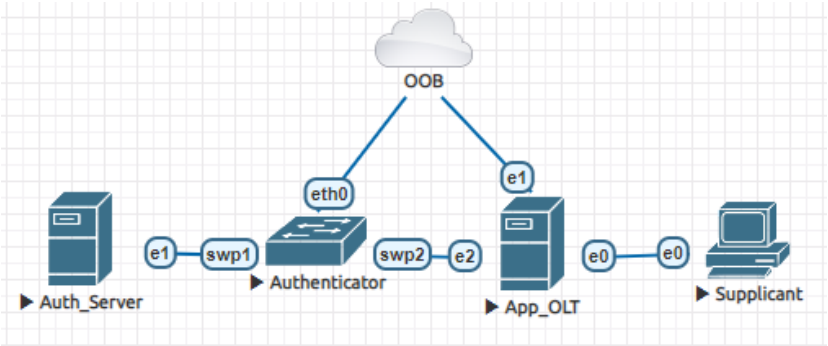


Figure 18: Refreshed topology with the EAP proxy.

The cloud illustration in Figure above represents an Out-Of-Band network for managing these devices via SSH.

The proxy is built using Golang and comprises libraries to listen to network Ethernet frames and TCP packets in some specified host interface. This second phase had the objective of starting the Golang development toward NFV creation and module application to run inside the OLT for communication establishment with NFV in the next stage. Beyond the proxy, the last components, the supplicant, authenticator, and authentication server are still into this scenario.

The intention is to evolve this EAP proxy module to receive these frames from supplicants and forward them to an 802.1X VNF, which will have the authenticator role, via a programmable interface as gRPC and receive responses through it.

4.3.3 *Phase 3: Virtualizing Dot1x elements*

In the third phase, the Golang development had advanced towards more 802.1X protocol actors, namely the 802.1X VNF, which replaced the Cumulus Linux as the authenticator, and improvements in OLT-App left the proxy role. Moreover, all the topology elements have been containerized into images, which can enjoy virtualization benefits, such as fast deployment, the flexibility of scaling out/in, the portability with many infrastructures, and other advantages mentioned before.

More specifically, the gRPC communication was established between the Control Relay and the application embedded into OLT and between the 802.1X VNF. Moreover, these actors had integrated into OB-BAA architecture, namely with Control Relay and Kafka broker. In addition, the Kafka messages produced by 802.1X VNF could be consumed by SDN Controller, which could deploy some rules to the infrastructure via *Persistent Management Agent* (*PMA*), for example, allowing the traffic and setting the appropriated network configurations.

A diagram with these components and their interfaces and correlations could be seen in Figure 19.

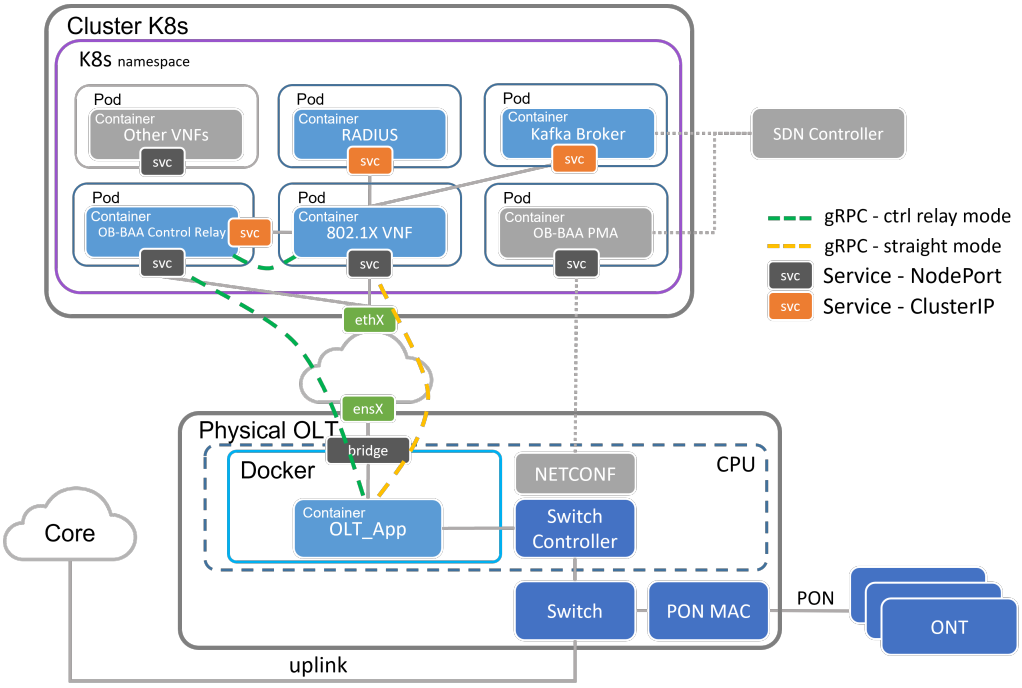


Figure 19: Diagram of phase 3 topology elements.

The diagram in Figure 19 exemplifies a scenario where the OLT-App is already embedded into a physical OLT hardware. However, for the sake of tests and more scalability, was deployed a virtualized environment with Docker. This approach is represented in Figure 20 below.

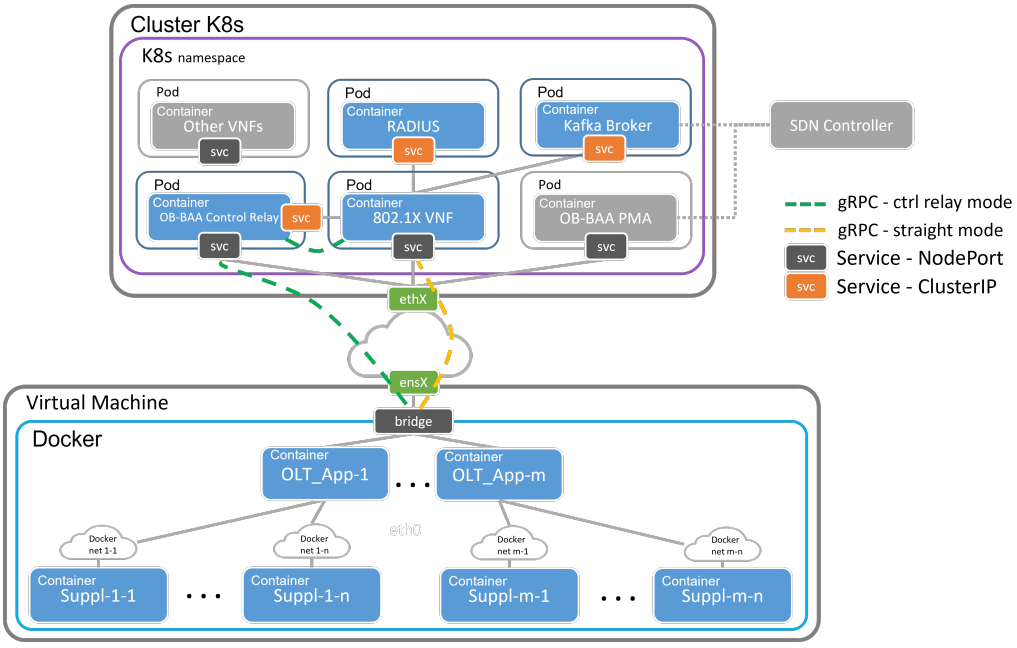


Figure 20: Diagram of phase 3 topology elements with virtualized OLT environment.

One significant point is that in both scenarios, with or without real physical OLT illustrated in Figures 19 and 20, the OLT-App always receives the EAP packages from supplicants through Ethernet interfaces. This characteristic allows seamless migration from a simulated environment to a realistic deployment of OLT-App. Thus, the simulation test is considered valid for this pointed out the reason.

This 3rd phase has implemented two ways of communication between OLT-App and 802.1X VNF. As seen in Figure 20 above, one path could be set straight mode through 802.1x VNF directly or pass throughout Control Relay and then go to VNF. The Control Relay mode is preferred because addition information about the AAA process can be shared directly with BAA framework for future reasons.

Independently the settled way, the OLT-App will receive the EAP messages from supplicants and then encapsulate this message into the gRPC channel with the VNF destination. When the VNF receives the gRPC message, it unencapsulated the EAP payload and then sent to RADIUS. The reverse way back path is analogous, and the VNF waits for a RADIUS response, retrieve the RADIUS response, encapsulate the EAP message into gRPC and then sends back to OLT-App to deliver to the supplicant. This routine is repeated for every interaction.

During the AAA process, this session's relevant information is forwarded via Kafka message for SDN controller consumption. With this information, the SDN controller could have full interaction between the supplicant's first request for access until the full acceptance and registered of him. This allowance is summarized in the sequence diagram in Figure 21 below:

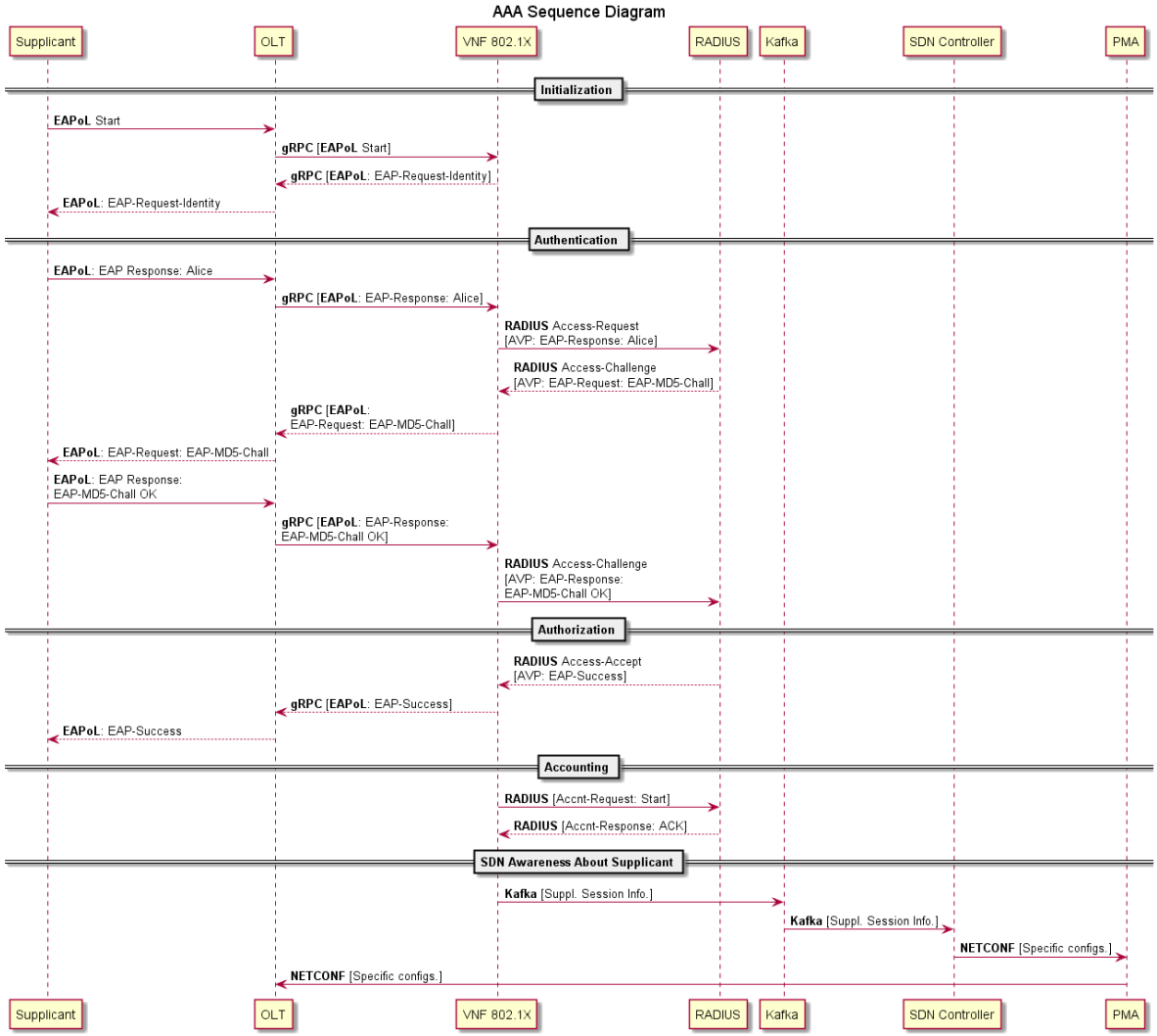


Figure 21: Sequence Diagram of phase 3 interaction

4.4 LIMITATIONS AND SCALABILITY ISSUES

This section pointed out the found limitations regarding the testbeds, used tools, machine restrictions, and performance limitations.

4.4.1 *Layer 2 Communication between Supplicants and OLTs*

Ideally, it was thought to build the entire test scenario hosted on a Kubernetes cluster. However, given a characteristic of the communication between supplicants and OLT, the supplicant sends multicast frames on the network, and the OLT must listen to them. It made such an implementation impossible in K8s. In the next chapter, the Section 5.1.4 illustrates the eight designed scenarios simulated in this project.

In the K8s, two hypotheses were put to the table. The first one is which OLT and supplicants would be implemented in the same Pod. In this case, both containers would have the same localhost IP and MAC addresses. Still, it was not possible due to this network characteristic, in which the address of the source could not be listening for frames emitted by itself.

The second hypothesis, also discarded, would place supplicants and OLTs in separate Pods, but this would also not be possible. The point is that the communication is L2 multicast type, and such frames are not propagated between Pods in the L3 domain in traditional configurations of the K8s.

Thus, the solution was to provide this infrastructure in a Docker environment, where distinct broadcast networks could be created between each OLT-Supplicant pair. These networks are MACVLAN type, and for all created subinterfaces, each one is associated with a VLAN and internal interface on the host that hosts the Docker framework. These subinterfaces are attached to the corresponding OLT-Supplicant pairs.

In this way, each element has IP and MAC addresses distinct from each other. So the multicast communication goes on as expected.

Note: It was supposed to be possible to place all supplicants connected to the same broadcast network, and an OLT interface is also attached to this network, but it was verified that in this way, the supplicants who are listening to all other traffic from others when receiving such frame start to restart the authentication process, which results in a loop. It has not yet been possible to remedy this problem, and for this issue, it was decided to create multiple separate broadcast networks, one for each OLT-Supplicant peer.

4.4.2 *Limit of Subinterfaces in Linux Container*

An OLT supports up to thousands of ONU/ONT (supplicants) connected in a realistic environment. Although, in the designed test environment, a limitation was found to create up to 30 interfaces in the image (Linux Alpine) of the container used for OLT provisioning in the environment Docker. Thus, together with the previous restriction, the environment was provisioned with one supplicant connected to each OLT interface, thus totaling a maximum amount of 30 supplicants per OLT.

4.4.3 *Docker-compose performance*

Given the setup presented, we tried to scale such an environment with multiple OLTs and supplicants. For such a task in Docker, it is intuitive to think about using Docker-compose, which is a multi-container descriptor. Still, other tools can also perform a similar job of provisioning a container environment, as is the case with Terraform and others. However, we initially chose to use the intuitive method. Although it has been demonstrated that there is a topology segmentation between Docker and Kubernetes environment, due to ease of tests and "tunings" in the architecture, it started with the entire architecture in a Docker environment.

With that, two docker-compose files were created, one for the instantiation of the most supplicant and OLT parts and another for the central elements of the topology. To assist in the creation of these files, as it is expected to do several tests with different amounts of OLTs and supplicants, bash scripts were also created in which the number of supplicants is inserted, and the script divides this amount by the maximum capacity of supplicants per OLT (30 supplicants). In non-multiple quantities of 30, the surplus is allocated to a new OLT. Thus, it is possible to create different test scenarios very quickly.

Despite the agility in creating deployment files, their execution with Docker-compose when the number of containers is high, above 150 containers, constant errors occur. The most frequent errors are "OCI runtime" and "failed to add interface vethXXXX to sandbox", and in addition to these, the creation process is stopped being parked in the docker-compose deployment screen, and only part of the containers are instantiated.

So far, no definitive and conclusive answer has been found about such observed problems. However, it has been found and verified that docker-compose does not have a high performance for deploying multiple containers in this test scale on a host with computational resources used (4GB of RAM).

Besides this point, as M different networks are being used for N supplicants, it is an unusual configuration for "containerized" environments in Docker, which may compromise the solution's performance. An alternative to such a problem with errors was using Docker-

compose, using the classic command "docker run" and its arguments and attributes. Due to the lack of scalability of this command, bash scripts were also created to replicate the command N times for N containers and create networks and connect them to the respective containers. This approach proved to be more efficient than docker-compose concerning errors and crashes, which became much less frequent. So, this last deployment method has been adopted since then.

4.4.4 *Limit on the number of containers for computing resources*

As mentioned before, the virtualization method used by Docker and Kubernetes is more efficient than the traditional way, but there is always a limit. In this case, after several combinations of quantities of containers were found that the used host to run the Docker framework could instantiate 180 supplicant containers and 6 OLTs for the hosted Docker environment VM's computational configuration, which is 4GB of RAM. Thus, more than the number of containers exceeds 180, the machine's memory gets fully used.

This number of 180 is reached when running only the OLTs and supplicants. But when the Kafka consumer and the Grafana and Prometheus machines are added, this number dropped to 90 supplicants and 3 OLTs.

4.5 SUMMARY

This chapter presented some details around the proposed implementation approach of protocol IEEE 802.1X as an application of BBF's broadband architecture, which follows the new trend of "softwarerization" of the networks through SDN adoption. Therefore, the chapter discussed the employed tools to perform the project and its usage through the implemented phases and how they evolved. Finally, was pointed out a few identified limitations concerning the implementation itself.

5

TEST SCENARIOS AND RESULTS

This chapter intends to consolidate the outcomes of the proposed approach exposed in the last Chapter 4. Moreover, this chapter will explain the test scenarios of the implemented solution and further discuss the performance metrics to evaluate it. Lastly, the subsequent section will present the achieved results, analyze them, and close with a major section to solidify the impressions and results.

5.1 TEST SCENARIOS

The tests consist in to simulate Dot1x supplicant authentication and authorization processes implemented in this project following the steps already described in the sequence diagram shown in Figure 21. The next subsections describe in more detail the test characteristics.

5.1.1 *Hosted Infrastructure*

As mentioned in Chapter 1, this project ran in an enterprise environment, namely with *Altice Labs*, which provided all the necessary infrastructure to complete this work.

Hardware Specification

The used infrastructure divides into two main blocks, the Docker and the Kubernetes environment. For Docker, has been made available a virtual machine with 2 vCPUs and 4GB of memory. Concerning Docker yet, a physical OLT was available for testing with Docker environment on it, with Dual Core CPUs ARM64 and 4GB of memory. However, it is a shared resource with other researchers, adding a constraint to the test scalability.

For Kubernetes, the Altice Labs made available a namespace into their cluster, which has 3 physical nodes each with a 2x Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz (12-core) and 128 GB of memory.

Container Images

Containers leverage all the instances used in the tests into Docker and Kubernetes environments. Hence, these containers are composed of base images, which are immutable static files with executable code and can be deployed consistently [53].

For better efficiency, the project sought the most lightweight images possible to instantiate the microservices. For this task, the Alpine Linux image was chosen. As the maintainer claims, the Alpine distribution is small, simple, and secure. It consumes only 8MB for the container image and can add the most relevant packages with package manager APK. The userland binaries are compiled as Position Independent Executable (PIE) with stack smashing protection [54]. So, the Alpine is the base image for the following microservices: OLT-App, VNF 802.1X, supplicant, Kafka consumer, Monitor (Prometheus & Grafana), and OLT-App embedded into physical OLT settled the Alpine in version ARM64. For RADIUS Server, was used a pre-built image from FreeRADIUS.

Regarding the BAA, Control Relay, which is a part of Core microservices from OB-BAA architecture, the images are built and provided by own OB-BAA repository at OB-BAA's Github repository. The OB-BAA also uses the Kafka and Zookeeper, and their images are from Confluent Inc. Github repository.

Beyond this image, another one was deployed to perform the metric collector function for containers. This is the cAdvisor (Container Advisor) image maintained by community leading for Google, Intel, VMware, and RedHat [55]. For physical OLT, due to ARM architecture, was used the Zcube image, which implements the cAdvisor for ARM [56]. The cAdvisor is a running daemon that collects, aggregates, processes, and exports information about running containers and exposes these metrics to Prometheus [57]. The Figure 22 resumes the correlation between the containers, cAdvisor, Prometheus, and Grafana.

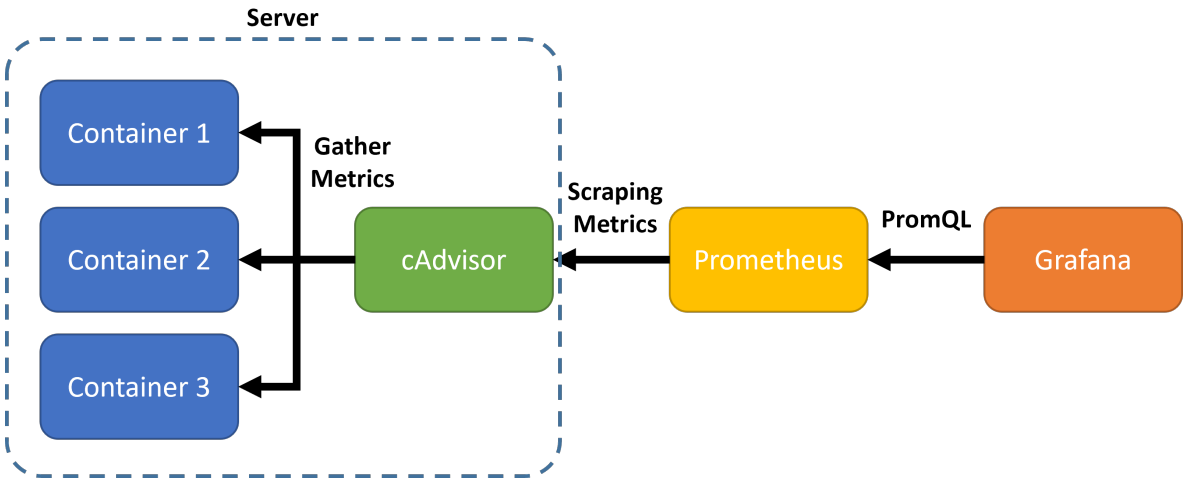


Figure 22: CAdvisor integration.

The Kubernetes environment used the same image, the Alpine Linux, for VNF 802.1X, the BAA and Control Relay from OB-BAA repository, RADIUS from FreeRADIUS. Concerning the monitoring in Kubernetes, a chart from the Grafana community repository was used to implement a monitoring stack to query the metrics from Kubernetes pods.

5.1.2 *Scalability Test*

The objective is to test different quantities of running supplicants and OLT-App until they reach the infrastructure limits. With given hardware specifications and with all minimum required instances to run the tests without running out of memory space, it was possible to run up to 90 supplicants simultaneously. In initial tests, when the monitoring instances as Prometheus and Grafana were not running, this quantity reached 120 supplicants.

Thus was made tests for three different quantities, 30, 60, and 90 supplicants, for each combination of operation mode and infrastructure selection.

5.1.3 *Operational Modes*

The figures 19 and 20 have shown the two operational modes, which differ between them how the traffic goes among OLT-App and VNF. So the traffic can go passing through Control Relay, which acts as a proxy, or the traffic can flow straight between OLT-App and VNF.

5.1.4 *Combination of Scenarios*

Considering the exposed characteristics above, a set of test scenarios with multiple combinations can be defined. So the scenes are a combination of which infrastructure runs the supplicant and OLT-App (into physical OLT or Virtual Machine), the core components into Kubernetes Cluster or Docker into Virtual Machine, and the Operational Mode (traffic passing through Control Relay or straight mode). This results in the following combinations listed and illustrated in Figures below:

Physical OLT / Docker / Control Relay

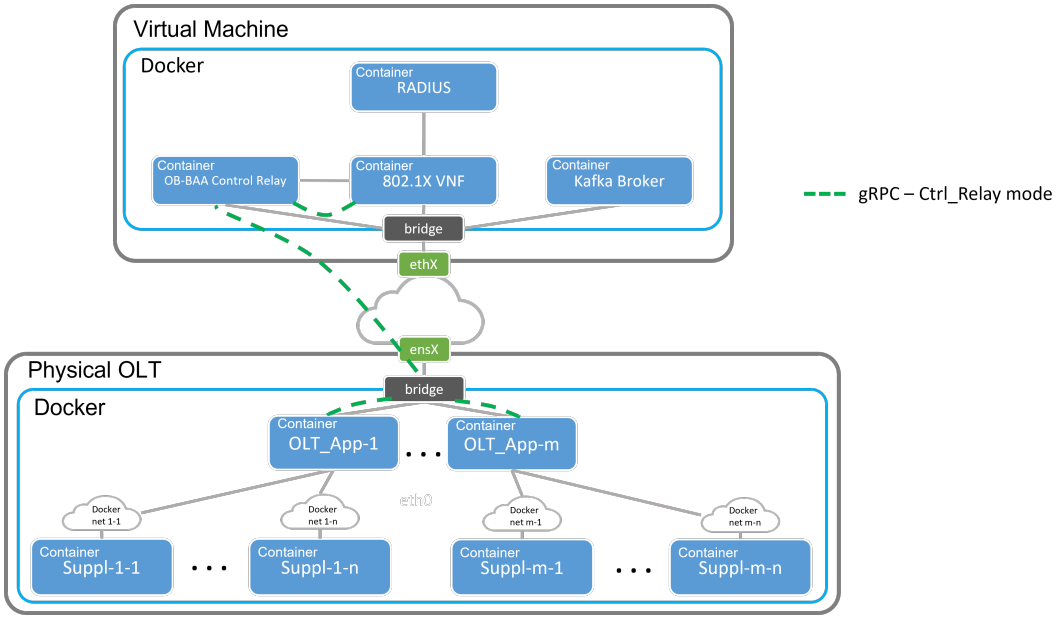


Figure 23: Diagram of scenario Physical OLT / Docker / Control Relay

Physical OLT / Docker / Straight

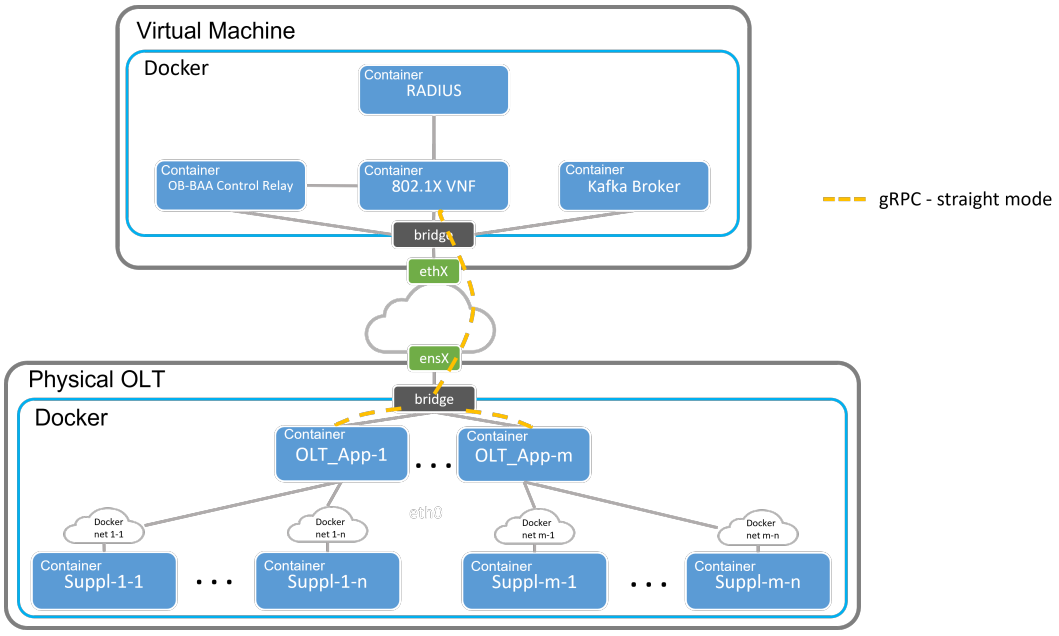


Figure 24: Diagram of scenario Physical OLT / Docker / Straight

Physical OLT / Kubernetes / Control Relay

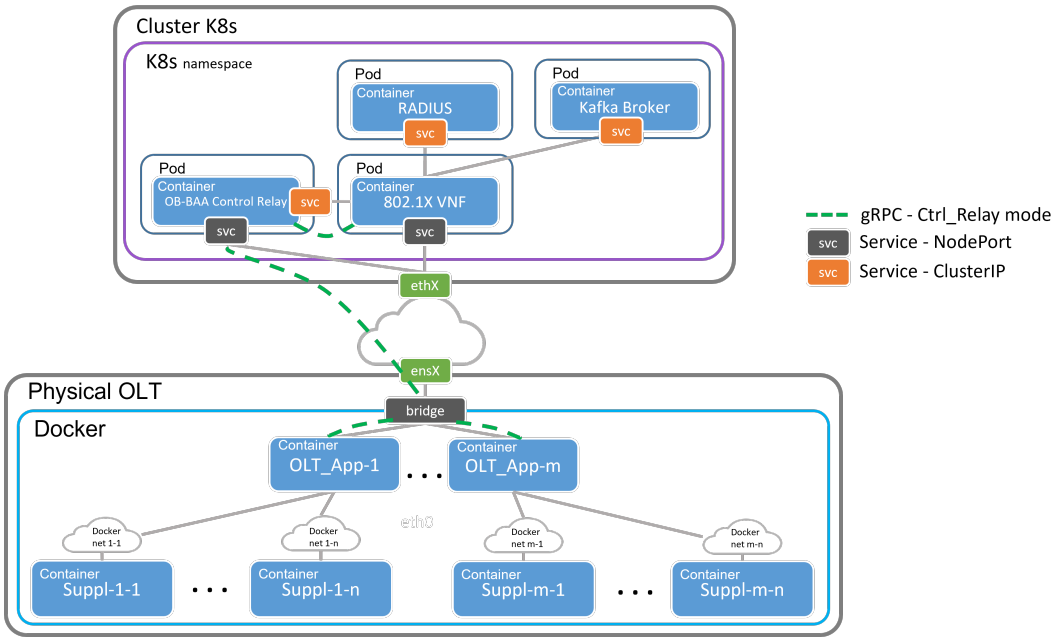


Figure 25: Diagram of scenario Physical OLT / Kubernetes / Control Relay

Physical OLT / Kubernetes / Straight

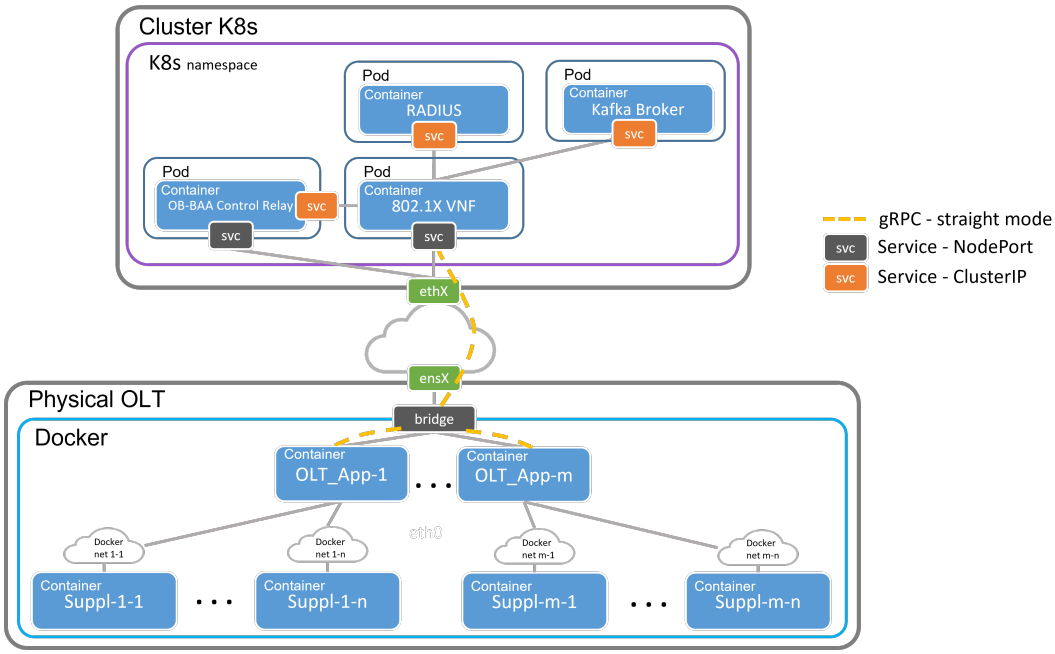


Figure 26: Diagram of scenario Physical OLT / Kubernetes / Straight

Virtual Machine / Docker/ Control Relay

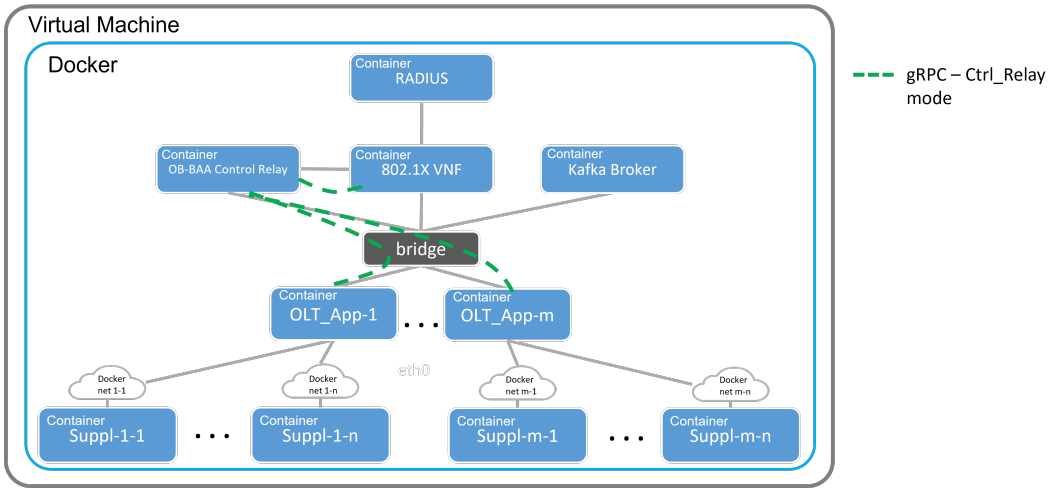


Figure 27: Diagram of scenario Virtual Machine / Docker/ Control Relay

Virtual Machine / Docker / Straight

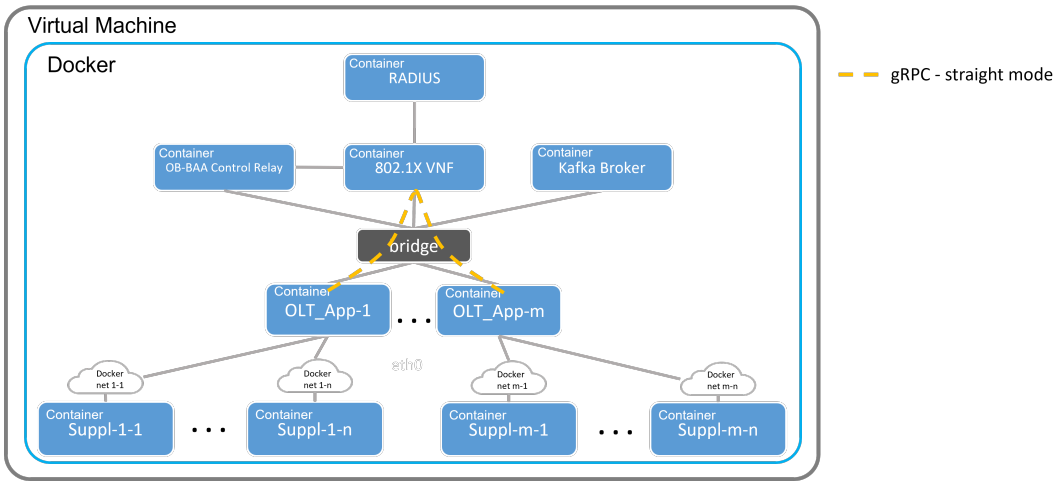


Figure 28: Diagram of scenario Virtual Machine / Docker / Straight

Virtual Machine / Kubernetes / Control Relay

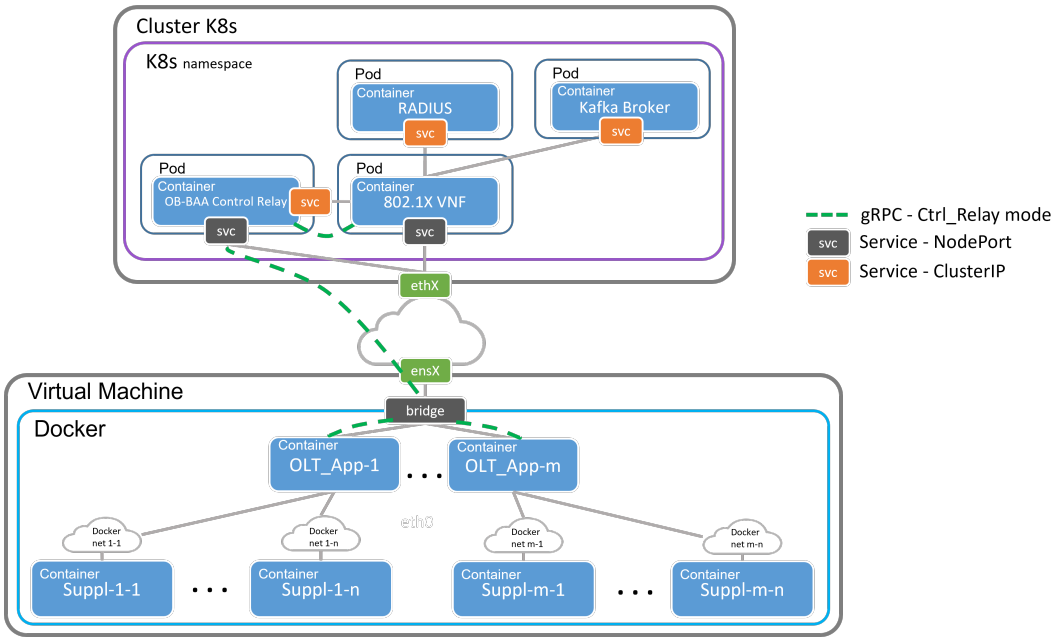


Figure 29: Diagram of scenario Virtual Machine / Kubernetes / Control Relay

Virtual Machine / Kubernetes / Straight

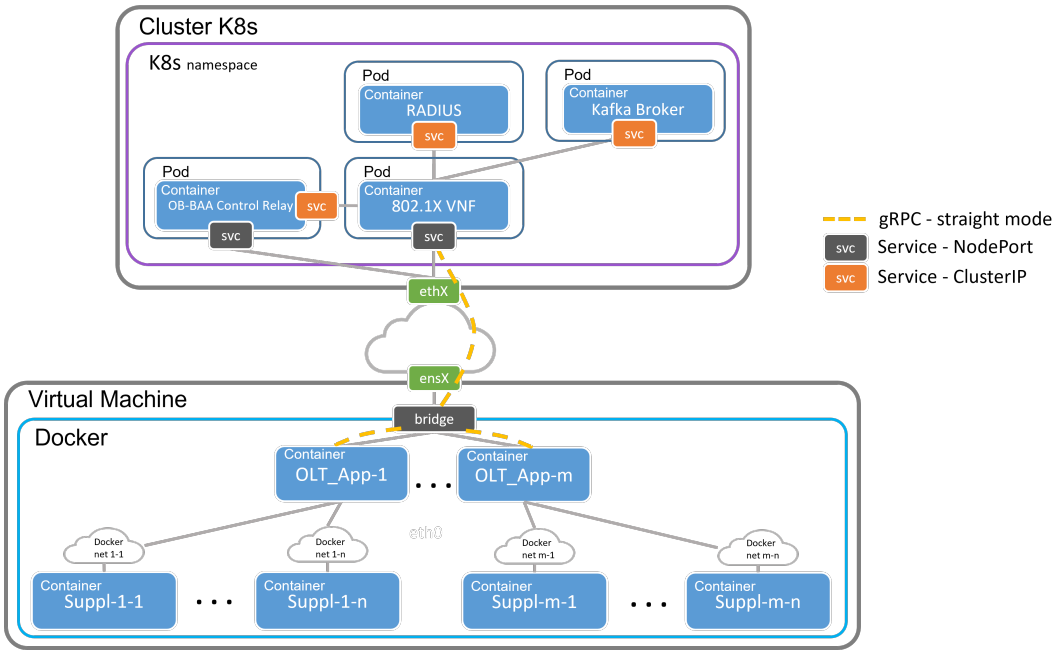


Figure 30: Diagram of scenario Virtual Machine / Kubernetes / Straight

Furthermore, each combination was tested for 30, 60, and 90 supplicants, which ended in 24 tests scenarios. Moreover, each one was made five tests for every 24 scenarios, resulting in 120 tests. The results are compiled in the Result Analysis section.

5.2 PERFORMANCE EVALUATION METRICS

This section exposed the chosen performance metrics to evaluate the impact of developed applications.

5.2.1 *Processing Time of Developed Applications*

To evaluate the metrics regarding the proposed implementation were settled time counter into the Golang applications codes to measure the spent time on each step concerning the AAA processes. In Figure 31 Chapter the summarized previous Dot1x sequence diagram (Figure 21) with identification in blue color words the measured intervals between messages, the brackets in red indicated the OLT-App metrics, while the brackets in orange means the metrics on VNF side.

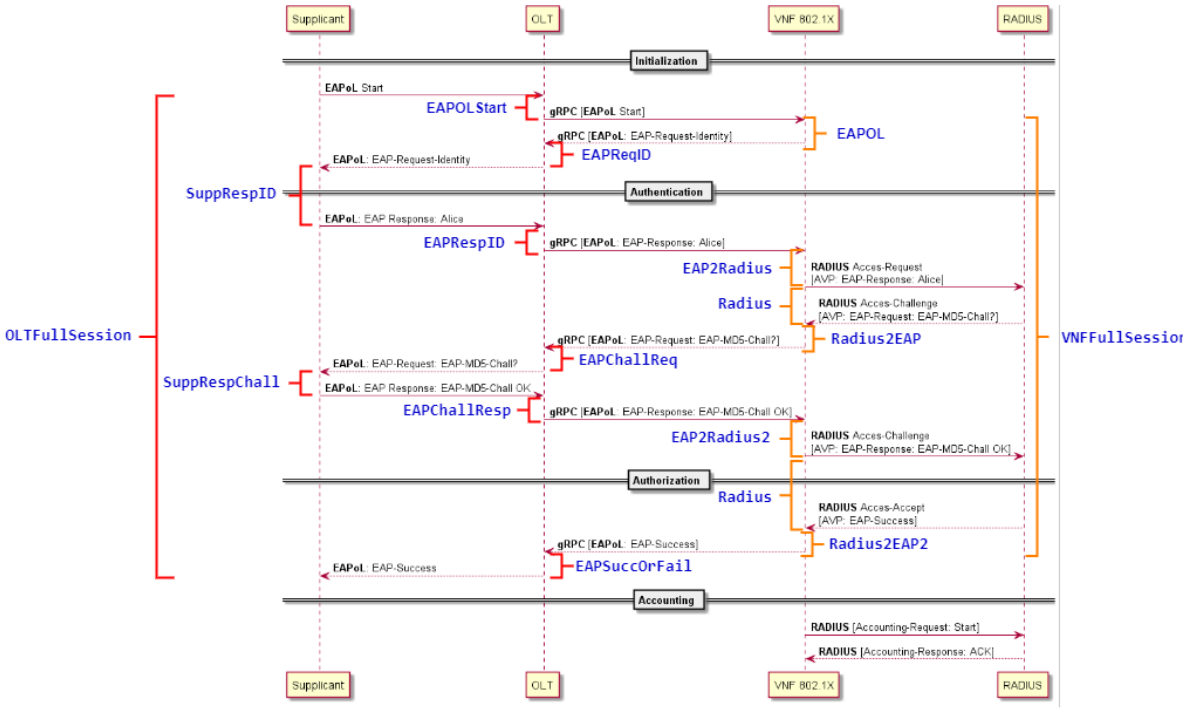


Figure 31: Dot1x sequence diagram with performance spending time metrics

For each supplicant interaction generates a log output in VNF and OLT-App microservice. At the end of the simulation, a script is run to consolidate the time metrics for each message illustrated in Figure 31. For each metrics are calculated the following statistics:

- maximum registered time;
- minimum registered time;
- average registered time.

Then, a CSV file is generated resuming each test. So the consolidated metrics are the maximum, minimum, average for all supplicants sets (30, 60, and 90 pieces) of each test.

5.2.2 *CPU, Memory and Network I/O*

This project leverages Prometheus scraping data from the cAdvisor container for the Docker environment and the Loki-Grafana stack for Kubernetes tests. This Loki stack is a well predefined set of multiple containers which perform their specific functions and are deployed together via Helm Charts (packet manager for Kubernetes) [58]. Grafana can visualize all gathered data to understand the computational resources consumption CPU, memory, and network I/O of each container.

5.3 RESULT ANALYSIS

This section presents the results of the tests which simulate the authentication and authorization of a group of supplicants using the IEEE 802.1X protocol with the MD5-Challenge authentication method. Initially, there are tables with collected metrics mentioned in the diagram at the Figure 31. The second part of the results shows graphs with the test's CPU, memory, and network I/O.

As specified before, eight different scenarios were executed for three scales of tests (for 30, 60, and 90 supplicants). Each of these was repeated five times on average, resulting in approximately 120 running simulations. Tables were generated for VNF results and another for OLT_app for each test, which results in 240 tables.

For brevity, the five tests for each scale were made an average of them, resulting in 48 columns tables. In addition, another metric average for each scale was calculated to compare the scenarios better. In summary, there were 48 columns table, plus 16 for the averages. These columns results were organized into eight tables with eight columns each. One of these tables are shown at Table 8 and 9, and the all of eight tables are at the Appendix.

These eight tables' results were divided into two groups. The first four tables contain the metrics collected in OLT_app, nine metrics (the table rows) with their minimum, maximum, and average for time and request-per-second. On the other hand, the next four tables are related to VNF metrics in the second group, ten metrics (the table rows) with their minimum, maximum, and average for time and request-per-second. The time values shown at the tables are in milliseconds except for the request-per-second metrics.

For each table quartet, there are two tables regarding tests with the **OLT application** and **supplicants** hosted into **Virtual Machine**, and the other two with the **OLT application** and **supplicant** hosted into **physical OLT**. Furthermore, for each couple of tables, one had the **Core microservices** running on **Docker** in a virtual machine and another couple into the **Kubernetes** environment. Finally, for each hosting environment, there is one test with traffic flowing between **supplicants** and **Dot1x VNF** through **Control Relay** and another test with **Straight flow** mode between them.

Each table there is a comparison between Traffic Flow Mode average performances. Moreover, the last column shows the result of this comparison. The Table 8 is an example of this shrunken statistics. All of the eight tables mentioned in the last paragraphs are in the Appendix, Chapter A.

5.3.1 *Processing Time (Tables)*

To evaluate the results, the average of each metric was compared between the **Traffic Flow Modes** collected at each mode (Straight and Control Relay). These metrics registered the spent time of each metric and also the rate of processed requests per second. Based on these results and comparisons, the tests were evaluated.

OLT Application Metric Statistics

The tables could infer some analysis. Firstly, comparing the performance of **Control Relay Flow Mode** against the **Straight Flow Mode**, the Control Relay mode had better results for **OLT application metrics**.

VNF Metric Statistics

In the **VNF metrics**, there was a draw between the two flow modes, which is more performative for Physical OLT and Virtual Machine implementation. The **Control Relay Flow Mode** had a better performance for both implementations (Physical OLT and VM) when associated with the Core elements running on Kubernetes. On the other hand, the **Straight Flow Mode** had better results for also for both implementations when deployed with the Core elements into the Docker environment.

Core Elements Performance

The **Core elements**, the 802.1x VNF, the Control Relay, RADIUS Server, BAA, Kafka, and Zookeeper, were deployed in two possible ways, into the **Kubernetes** and the **Docker** environments. So, for the same test run in both tools, it is expected to get different results.

Regarding the tests, the **Kubernetes** environment was way more efficient (lower process time) than **Docker** (77.8% of the tests on OLT statistics and 95% on VNF statistics) when

combined with supplicants and OLT-App running on Virtual Machine. The Physical OLT had a better result with the **Docker** environment for OLT statistics (72.2%) and slightly better for VNF statistics (55%).

In Table 10 resumes the last comments about *OLT Application Metric Statistics*, *VNF Metric Statistics*, and *Core Elements Performance*.

Table 8: OLT-App Metrics Statistics for Scenario with: Physical OLT, Docker and both Traffic Flow Modes

Measuring Point	OLT-App Statistics								
Suppl. & OLT App Hosting	OLT PHYSICAL								
Core Hosting Environment	DOCKER								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Avg.**	**30**	**60**	**90**	**Avg.**	**Best Mode**
EAPOLStart max. Time (ms):	6,058	18,139	16,779	13,659	11,454	13,899	7,640	10,997	
EAPOLStart min. time (ms):	1,327	1,400	1,627	1,451	1,480	1,537	0,481	1,166	
EAPOLStart avg. time (ms):	**2,128**	**3,043**	**3,558**	**2,910**	**3,023**	**3,262**	**1,090**	**2,458**	**CTRL_RELAY**
EAPReqID max. time (ms):	16,501	73,761	49,229	46,497	18,229	70,667	13,240	34,045	
EAPReqID min. time (ms):	1,668	2,775	3,899	2,781	1,642	2,747	1,337	1,909	
EAPReqID avg. time (ms):	**4,186**	**7,148**	**8,799**	**6,711**	**4,085**	**7,887**	**2,755**	**4,909**	**CTRL_RELAY**
EAPRespID max. time (ms):	18,440	19,553	21,356	12,664	9,445	32,777	8,707	16,977	
EAPRespID min. time (ms):	1,126	2,171	3,598	1,099	1,302	2,255	0,435	1,331	
EAPRespID avg. time (ms):	**2,786**	**4,911**	**7,844**	**2,566**	**2,671**	**5,958**	**1,216**	**3,282**	**STRAIGHT**
EAPChallReq max. time (ms):	25,383	57,078	34,297	38,919	35,617	57,072	33,043	41,911	
EAPChallReq min. time (ms):	1,659	2,808	3,802	2,756	1,720	2,840	1,336	1,965	
EAPChallReq avg. time (ms):	**5,602**	**7,776**	**7,704**	**7,027**	**6,240**	**8,921**	**4,742**	**6,634**	**CTRL_RELAY**
EAPChallResp max. time (ms):	27,281	697,524	3015,315	1246,707	27,740	708,524	11,506	249,257	
EAPChallResp min. time (ms):	0,978	671,383	2009,625	893,995	1,238	671,521	0,415	224,391	
EAPChallResp avg. time (ms):	**3,475**	**674,985**	**2028,896**	**902,452**	**4,551**	**676,051**	**1,226**	**227,276**	**CTRL_RELAY**
EAPSucc max. time (ms):	19,498	62,390	48,506	43,465	26,536	53,322	13,176	31,011	
EAPSucc min. time (ms):	1,663	2,789	3,859	2,770	1,662	2,792	1,309	1,921	
EAPSucc avg. time (ms):	**4,698**	**7,105**	**8,760**	**6,854**	**5,242**	**7,133**	**3,257**	**5,211**	**CTRL_RELAY**
Suppl. max. spent-time (ms):	164,489	184,648	1085,696	478,278	1001,821	929,353	462,226	797,800	
Suppl. min. spent-time (ms):	21,523	24,232	23,974	23,243	19,695	21,957	8,875	16,842	
Suppl. avg. spent-time (ms):	**43,173**	**47,159**	**55,096**	**48,476**	**89,800**	**79,162**	**27,662**	**65,541**	**STRAIGHT**
Max. proc. request-per-sec.:	88	65	52	69	83	63	16	54	
Min. proc. request-per-sec.:	14	8	12	11	13	8	2	8	
Avg. proc. request-per-sec.:	**54**	**38**	**33**	**42**	**48**	**34**	**9**	**31**	**STRAIGHT**
OLT spent time (ms):	22,875	31,756	38,482	18,210	25,812	35,820	14,287	25,307	**STRAIGHT**

Table 9: VNF Metrics Statistics for Scenario with: Physical OLT, Docker and both Traffic Flow Modes

Measuring Point	VNF Statistics								
Suppl. & OLT App Hosting	OLT PHYSICAL								
Core Hosting Environment	DOCKER								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOL max. time (ms):	0,133	0,101	0,207	0,147	0,119	0,268	0,591	0,326	
EAPOL min. time (ms):	0,016	0,015	0,016	0,016	0,017	0,013	0,014	0,015	
EAPOL avg. time (ms):	**0,034**	**0,030**	**0,032**	**0,032**	**0,033**	**0,037**	**0,040**	**0,037**	**STRAIGHT**
EAP2Radius max. time (ms):	0,199	0,157	0,132	0,163	0,463	0,197	0,155	0,272	
EAP2Radius min. time (ms):	0,029	0,026	0,023	0,026	0,024	0,026	0,023	0,025	
EAP2Radius avg. time (ms):	**0,055**	**0,049**	**0,055**	**0,053**	**0,058**	**0,046**	**0,048**	**0,051**	**CTRL_RELAY**
Radius max. time (ms):	6,662	1001,316	1002,398	670,126	12,436	6,373	1001,612	340,141	
Radius min. time (ms):	0,683	0,595	0,587	0,622	1,395	0,675	0,627	0,899	
Radius avg. time (ms):	**1,490**	**9,676**	**7,137**	**6,101**	**3,064**	**9,595**	**7,228**	**6,629**	**STRAIGHT**
Radius2EAP max. time (ms):	0,060	0,247	0,057	0,121	0,040	0,111	0,169	0,106	
Radius2EAP min. time (ms):	0,014	0,013	0,013	0,013	0,014	0,014	0,012	0,013	
Radius2EAP avg. time (ms):	**0,023**	**0,026**	**0,026**	**0,025**	**0,024**	**0,026**	**0,026**	**0,025**	**STRAIGHT**
EAP2Radius2 max. time (ms):	0,154	0,129	0,171	0,151	0,068	0,097	0,394	0,187	
EAP2Radius2 min. time (ms):	0,020	0,021	0,020	0,020	0,023	0,020	0,020	0,021	
EAP2Radius2 avg. time (ms):	**0,042**	**0,040**	**0,045**	**0,042**	**0,037**	**0,039**	**0,046**	**0,041**	**CTRL_RELAY**
Radius max. time (ms):	6,662	1001,316	1002,398	670,126	12,436	6,373	1001,612	340,141	
Radius min. time (ms):	0,683	0,595	0,587	0,622	1,395	0,675	0,627	0,899	
Radius avg. time (ms):	**1,491**	**9,605**	**7,105**	**6,067**	**3,051**	**9,529**	**7,196**	**6,592**	**STRAIGHT**
Radius2EAP2 max. time (ms):	0,092	0,059	0,149	0,100	0,060	0,192	0,080	0,111	
Radius2EAP2 min. time (ms):	0,013	0,011	0,013	0,012	0,013	0,013	0,007	0,011	
Radius2EAP2 avg. time (ms):	**0,023**	**0,022**	**0,028**	**0,024**	**0,022**	**0,025**	**0,025**	**0,024**	**STRAIGHT**
FullSession max. time (ms):	95704,69	1069,79	1078,22	32617,57	241365,35	467,19	1364,67	81065,74	
FullSession min. time (ms):	11,343	13,327	16,270	13,646	14,665	13,776	15,684	14,708	
FullSession avg. time (ms):	**25,577**	**46,021**	**41,027**	**37,542**	**35,964**	**72,090**	**63,859**	**57,304**	**STRAIGHT**
Max. proc. request-per-sec.:	9075	10542	9611	9743	9464	9245	10652	9787	
Min. proc. request-per-sec.:	2931	2443	2694	2689	1800	2823	1248	1957	
Avg. proc. request-per-sec.:	**6272**	**6542**	**5857**	**6223**	**6483**	**6432**	**6044**	**6320**	**CTRL_RELAY**
VNF spent time (ms):	0,181	0,195	0,115	0,164	0,134	0,147	0,143	0,142	**CTRL_RELAY**

Traffic Flow Modes and Hosting Environments

Table 10 shows that the Control Relay traffic flow mode had a better performance on average for 6 out of 8 scenarios. Furthermore, 47 out of 76 metrics performed better on Kubernetes Cluster for hosting the infrastructure. Docker had a better performance in combination with supplicants and OLT-App running into physical OLT.

Looking at the result when the physical OLT was employed, the performance between Docker and K8S implementations are not much disparate; in general, they are very close results, but Docker was slightly better. When the OLT-App and Supplicants are hosted into VM, the results between Docker and K8s had a wider difference, and K8S got a large advantage over Docker; in some metrics, K8s is 5x up to 10x more performative.

Looking to the overall results, Kubernetes is more performative than Docker. Although the test scenarios are not so relevant in terms of scalability, flexibility, and high availability, the K8S could show its strength points regarding orchestration. It still proves a better tool for most executed tests. Kubernetes was not used in all the components because this work faced a network communication limitation between supplicants and OLT-App, where a multicast layer 2 communication was required, and the K8S was not able to do, so the Docker was employed instead.

OLT-App and VNF Metrics Differences

Another conclusion about the Tables 8 and 9 is the differences concerning the spent processing time by each OLT-App metrics and by VNF's. Looking closer at both results, VNF had around a 100x faster processing time than OLT-App's. The VNF had results in a dozen of microsecond scale, and the OLT-App got results in a few milliseconds (ms) for a processing metric step.

The average spent time for all combining processes of OLT-App are between 18ms to 40ms, with some isolated tests going up to 131ms. Regarding the spent time for VNF, it goes from 0,130ms to 0,211ms. Concerning the average number of process requests per second, the OLT could handle 31 up to 137 requests per second. The VNF instead reached, on average, the range between 5000 and 7000 requests per second. These numbers consider only the VNF's spent time because, into its lifecycle, there is the RADIUS procedure, which is a bottleneck due to its high spent time, consuming on average 6ms to 16ms, but in some cases up to 1500ms. This difference between OLT-App and VNF is expectable because the VNF would handle multiple messages simultaneously, coming from many authenticators (OLTs).

Table 10: Add caption

	References	OLT App Statistics				VNF Statistics			
Suppl. and OLT App Hosting		PHYSICAL OLT		OLT VM		PHYSICAL OLT		OLT VM	
Core Hosting Environment		K8S	DOCKER	K8S	DOCKER	K8S	DOCKER	K8S	DOCKER
Table		1	2	3	4	5	6	7	8
Better Mode per Table	Results	CTRL_R.	CTRL_R.	CTRL_R.	CTRL_R.	CTRL_R.	STRAIGHT	CTRL_R.	STRAIGHT
Better environment		DOCKER (13/18)		K8S (14/18)		DOCKER(11/20)		K8S (19/20)	
Better env for Supp and OLT-App		PHYSICAL OLT				PHYSICAL OLT			

5.3.2 *Computational Resources (Graphs)*

This section presents the results for using the computational resources, the CPU, memory, Network Received data, and Network Sent data, regarding all the main elements of the environment. As already mentioned before, all the visual graphics were collected using the Prometheus and Grafana tools.

Eight scenarios were tested, and they are presented at Combination of Scenarios subsection. For each of them, were executed tests with 30, 60, and 90 supplicants. These tests have generated graphs and divided them into three groups:

1. Core elements (VNF, Control Relay, FreeRADIUS, BAA, and Kafka);
2. OLT(s);
3. Supplicants.

The Supplicants' statistics were summed into a single curve to consolidate this group impact and facilitate the visualization of each supplicant's graphs. In some graphic figures, specifically about the core elements, the less relevant components (Kafka and BAA) were hidden to visualize other curves better.

To avoid an overload of images were selected the scenario with 60 supplicants to insert in this dissertation.

Performance Comparison Between Different Scales

This part shows a comparison between the three different scales (30, 60, and 90 supplicants). The graphics are from Core components, OLT, and Supplicants, showing the metric results of Network I/O, CPU, and memory.

The objective here is to observe the increase of the resource consumption throughout the scale growing. For this task was chose the scenario randomly: the physical OLT hosting the OLT-App and supplicants, Docker hosting the core elements, and the straight mode traffic flow settled (**Physical OLT / Docker / Straight mode**).

Core Elements Metrics

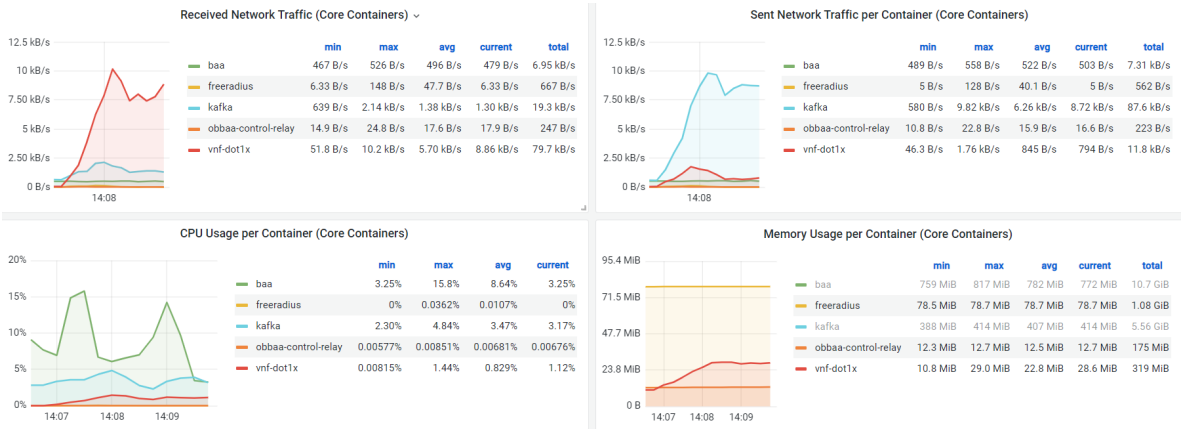


Figure 32: Consolidated metrics for Core elements for 30 Supplicants Scale.

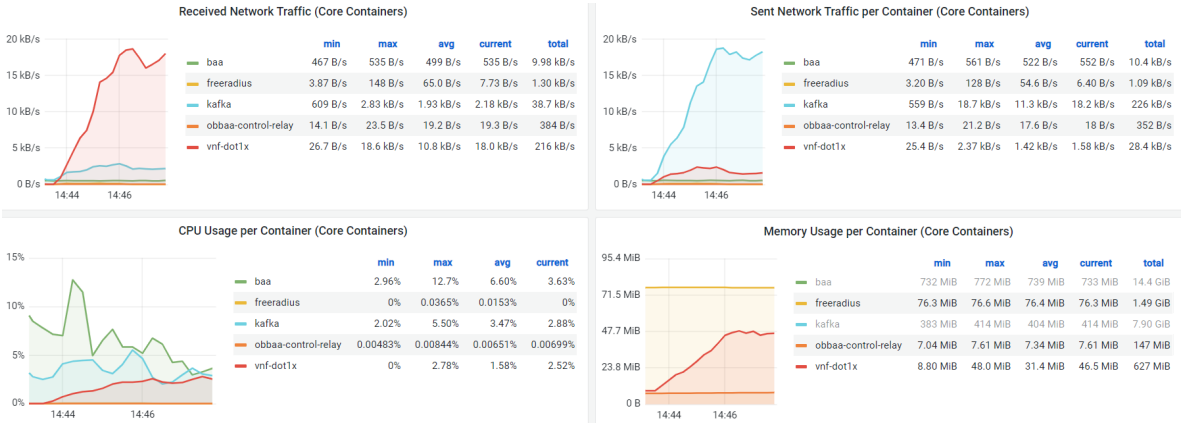


Figure 33: Consolidated metrics for Core elements for 60 Supplicants Scale.

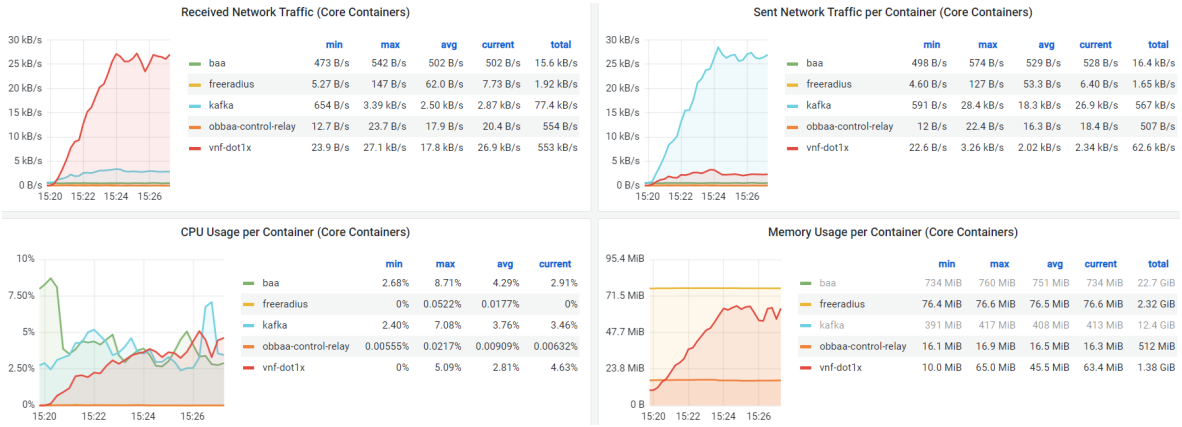


Figure 34: Consolidated metrics for Core elements for 90 Supplicants Scale.

The received network traffic for Core elements through the three scalability tests illustrated in Figures 32, 33, and 34 had shown an expected behavior once the number of supplicant scales, its traffic had a linear and proportional increase compared to increase of supplicants. At the same figures, the VNF's, FreeRADIUS, and Kafka's Network Traffic curves also had shown an increase of traffic volume at a relative proportionality as the number of supplicants increases. For Control relay and BAA, the network traffic is not so proportional as others because they transmit control data regarding the connection information and some info about them.

Regarding the CPU utilization, still looking at Figures 32, 33, and 34, only the VNF and FreeRADIUS had a proportional increase near the number of supplicants' growth. The other elements had a discord behavior from supplicants' increasing. The BAA had a decrease in usage. The Kafka got the same average usage for 30 and 60 supplicants' tests and a slight increase for 90. The Control Relay got a decrease between 30 and 60 supplicants and an increase for 90 supplicants.

Concerning memory consumption, the BAA, FreeRADIUS, and Kafka did not present any changes in consequence of supplicant scaling, keeping the consumption constant through all three tests. The Control Relay had a discord behavior against the supplicant scaling. When the test grew from 30 supplicants to 60, the Control Relay memory usage decreased, and this usage rose when tested with 90 supplicants. Regarding the VNF, it had the expected reaction, a usage growth proportionally between the tests.

OLT Metrics

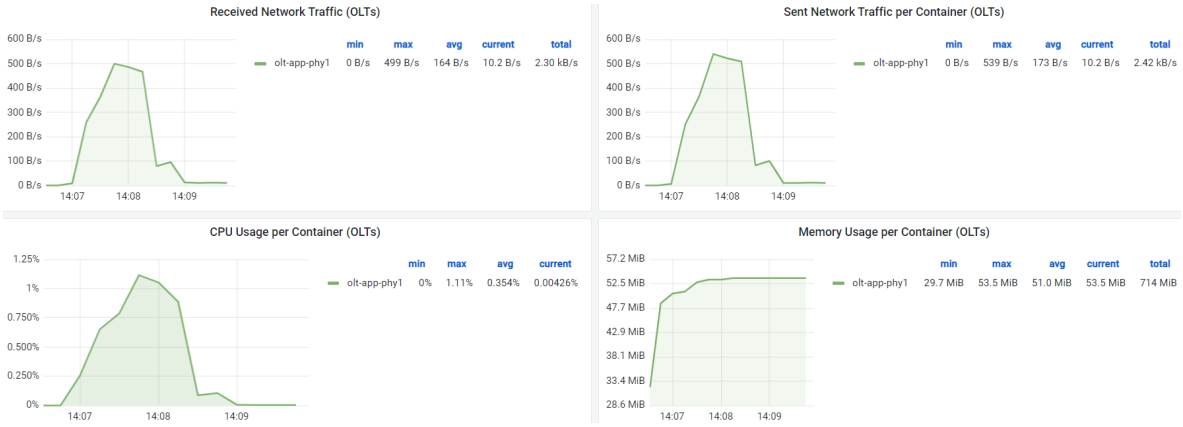


Figure 35: OLT's consolidated metrics for 30 Supplicants Scale.

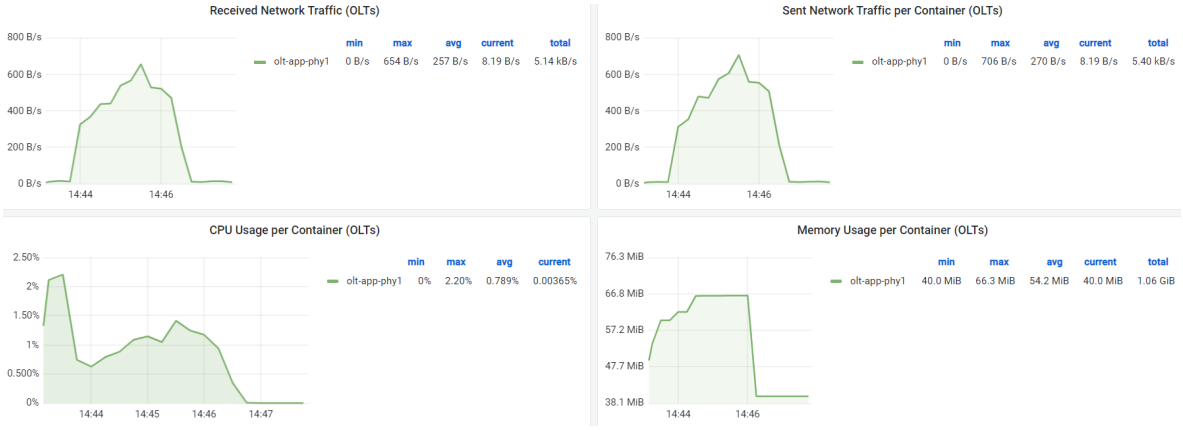


Figure 36: OLT's consolidated metrics for 60 Supplicants Scale.

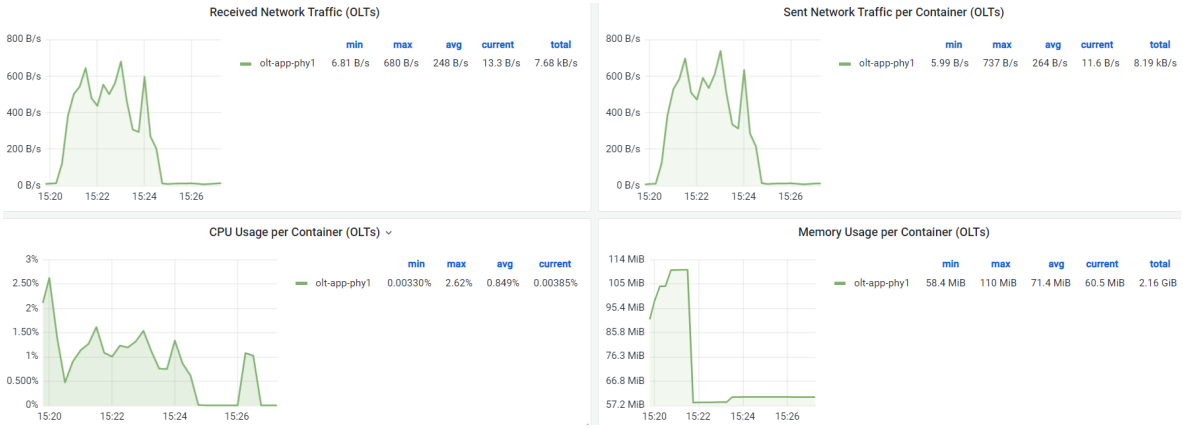


Figure 37: OLT's consolidated metrics for 90 Supplicants Scale.

The Figures 35 36 37 show the OLT's results. Regarding the Network I/O volume, it had grown proportionally as expected. As expected, the CPU and memory increased too; however, both metrics did not grow proportionally, and doubling the supplicants did not double the CPU and memory. Possibly due to internal running processes which consume resources independently to supplicants' number.

Supplicants Metrics

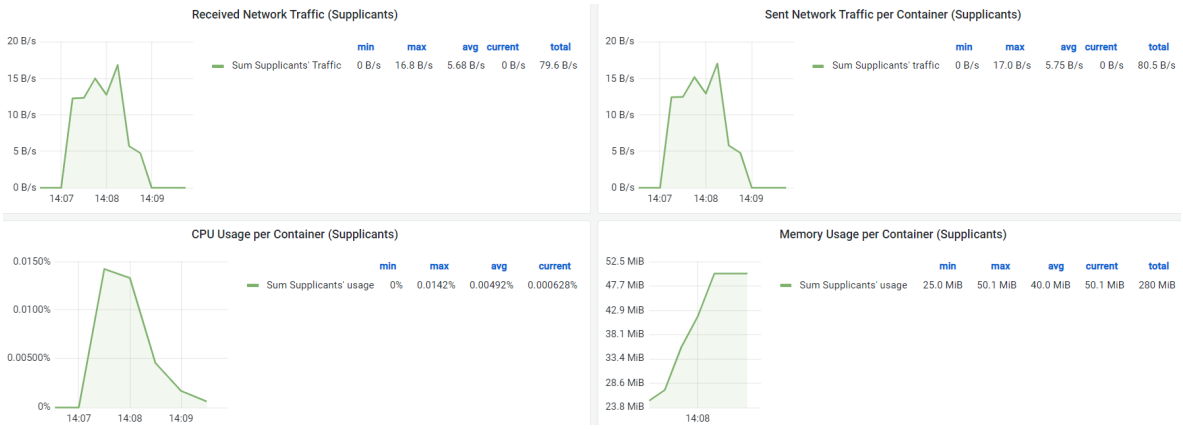


Figure 38: Supplicants' consolidated metrics for 30 Supplicants Scale.

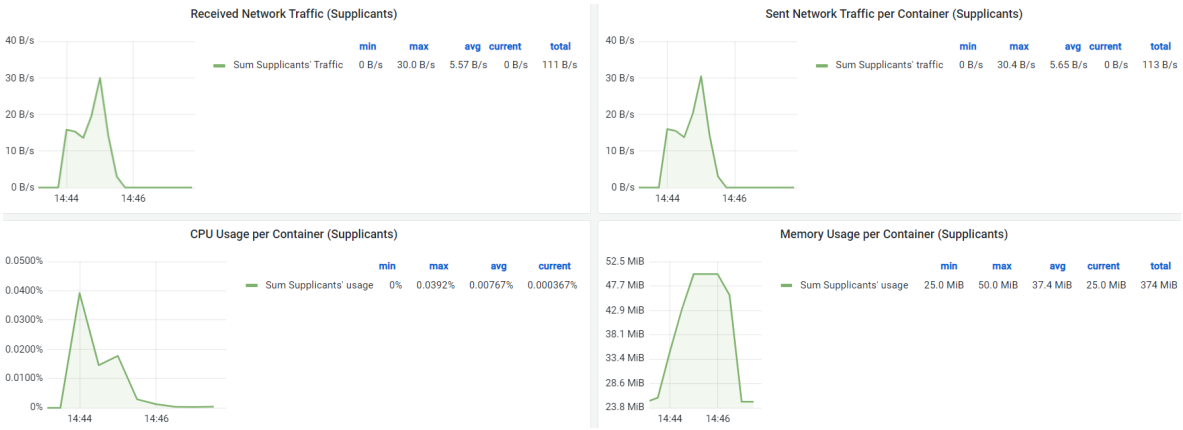


Figure 39: Supplicants' consolidated metrics for 60 Supplicants Scale.

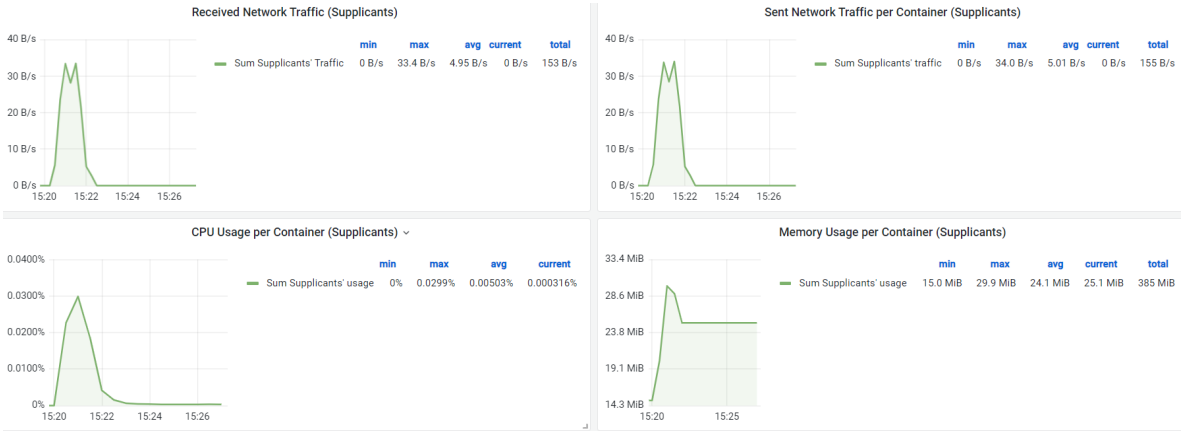


Figure 40: Supplicants' consolidated metrics for 90 Supplicants Scale.

The Figures 38 , 39 , and 40 show that Supplicants' behavior is nearly the same as presented for OLT's. So the Network I/O volume had grown linearly. The CPU increased from 30 to 60 supplicants, and for 90 supplicants, it had a slight decrease, contrary to the expected. Moreover, the memory had a relevant increase from 30 to 60 supplicants, and from 60 to 90 had an increase, but was a very discrete one.

Comparison Between Hosting Environments (Docker vs. Kubernetes)

In this subsubsection there is a comparison between the Kubernetes hosting method against the Docker for the same quantity of supplicants per test (60 supplicants). The following graphs in Figures 41, 42, and 43 are only regarding the Core elements because only they had been hosted in both environments (K8S and Docker).

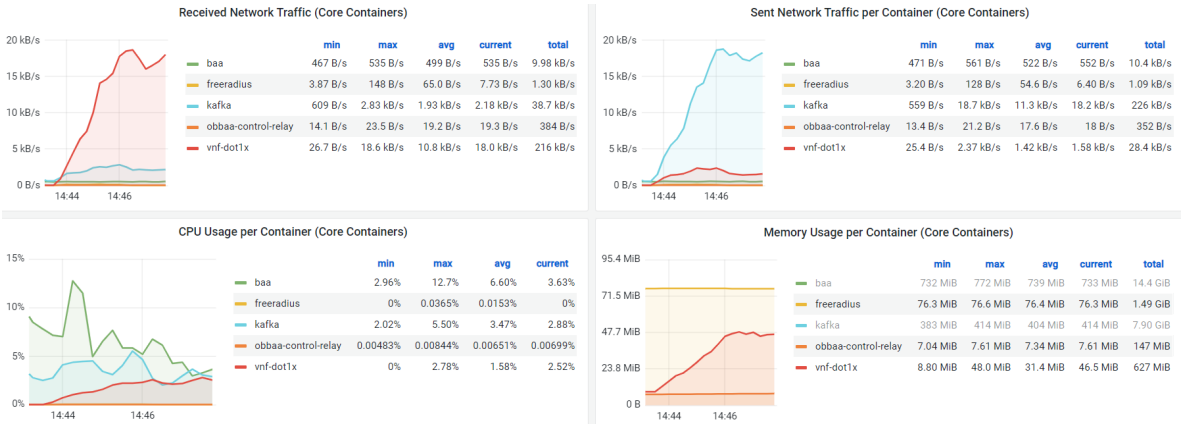


Figure 41: Core elements hosted into Docker environment in straight mode.

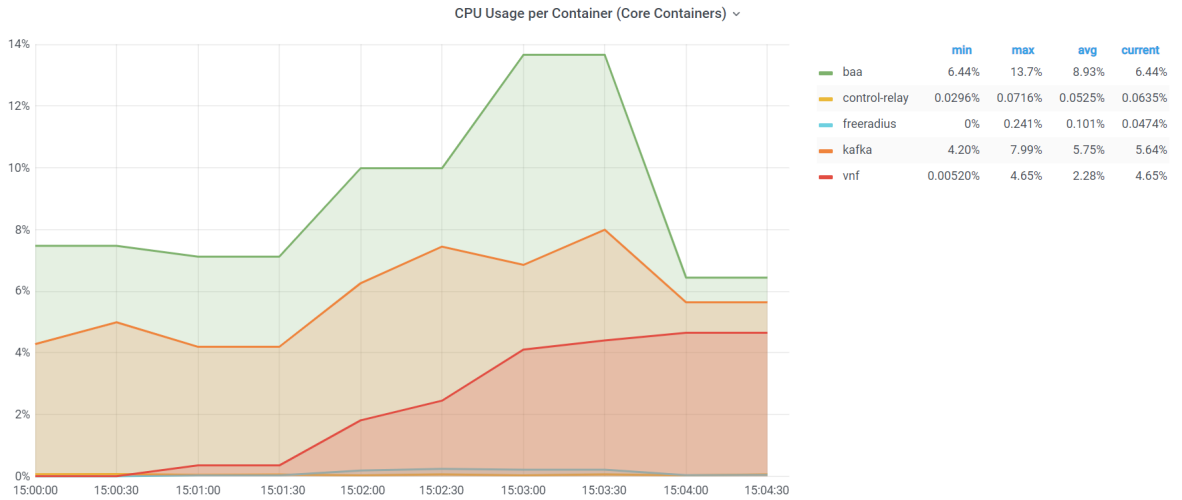


Figure 42: CPU Core elements hosted into Kubernetes environment in straight mode.

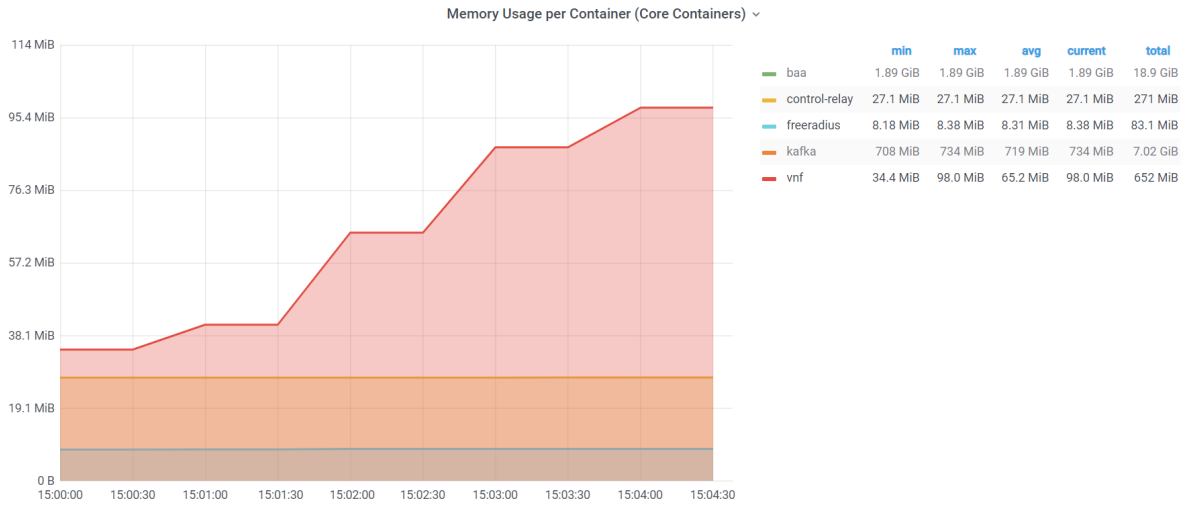


Figure 43: Memory Core elements hosted into Kubernetes environment in straight mode.

Looking at the CPU usage, Docker in Figure 41 seems to be more resource-saving than K8s, in Figure 42 for most elements. Regarding the memory results, these differences are even higher. The VNF, Control Relay, and BAA more than doubled the memory usage in K8s, and the Kafka almost got doubled too. Surprisingly, the FreeRADIUS reduced its memory consumption considerably, more than nine times lower. This behavior could be explained if, considering that the Kubernetes environment in this project has higher available resources than Docker, it could be used more intensively.

Comparison Between Traffic Flow Modes (Straight vs. Control Relay modes)

Another comparison is regarding the behavior of tests performed with each traffic flow mode, the Straight and the Control Relay mode. For this were selected the test scenarios executed with core elements hosted into Docker, moreover the supplicants and OLT-App were hosted into physical OLT. So the comparison is between the Figure 41 and the next Figure 44 below:

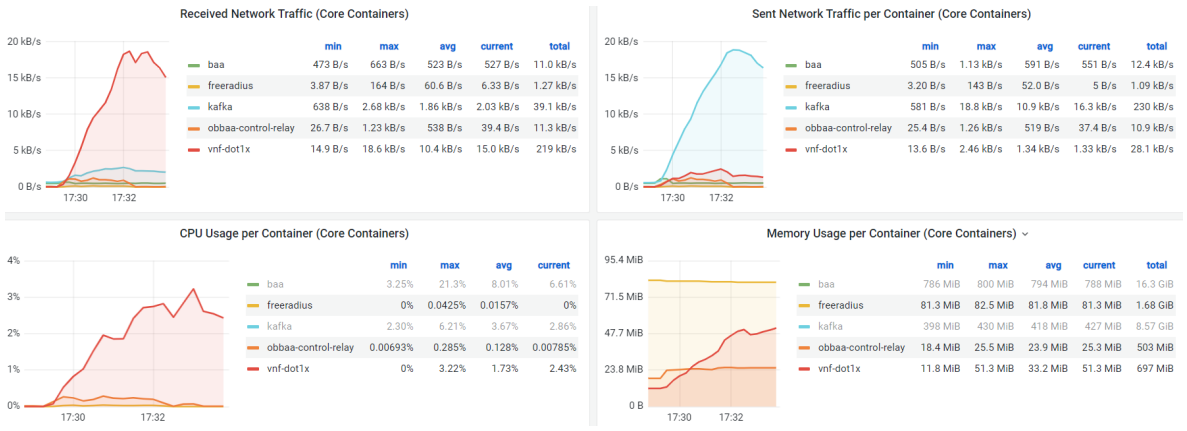


Figure 44: Core elements hosted into Docker environment in Control Relay mode.

The expectation for this comparison was to see a significant impact on Control Relay computational resource usage and no relevant changes in the other elements. So this expectation was achieved, the network I/O measured at Control Relay experienced a considerable impact, leveraging the network using up to 29,5 times, the CPU up to 19,7 times, and memory up to 3,25 times.

Comparison Between Hosting Environment (VM vs. Physical OLT) for OLT-App and Supplicant

The following comparison looks at the OLT-App and supplicants hosted into physical OLT scenario results for a realistic execution versus a scenario where these actors are hosted into a VM. The both Figures 45 and 46 below show the scenario running the core elements on Docker with straight traffic mode, the first hosted into physical OLT and the second into Virtual Machine.

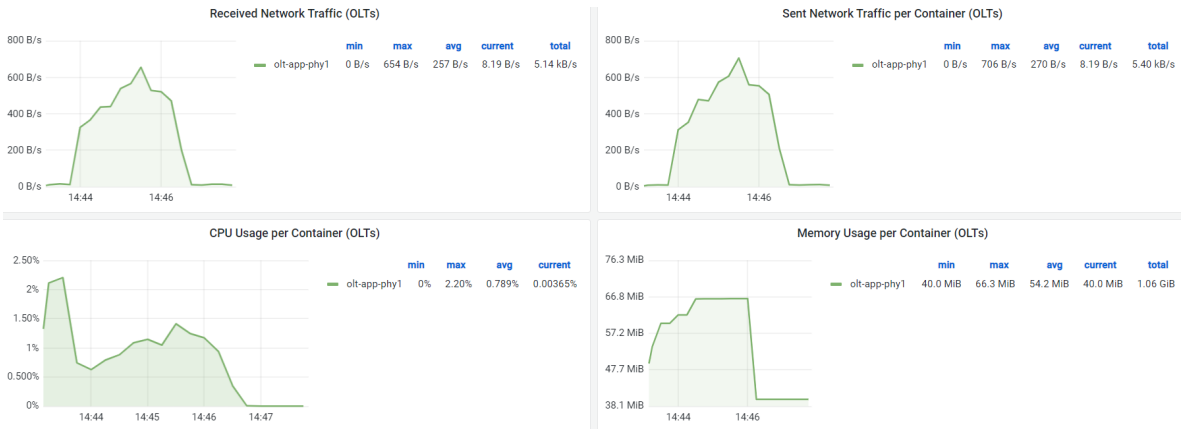


Figure 45: Physical OLT results for hosting environment of OLT-App and supplicants comparison.

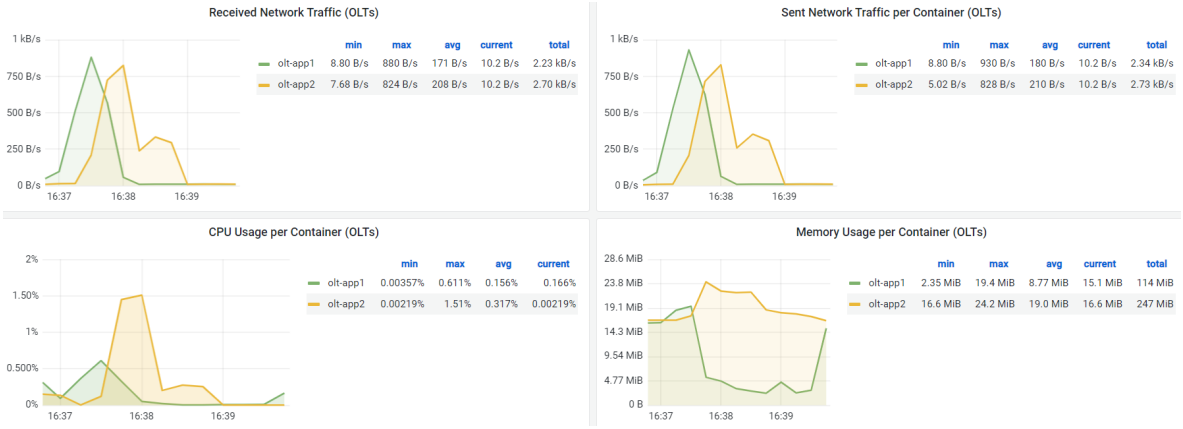


Figure 46: OLT on Virtual Machine results for hosting environment of OLT-App and supplicants comparison.

The OLT-App hosted into physical OLT got a higher consumption for all metrics, Network I/O, CPU, and memory. This behavior repeated for almost all scenarios unless for the K8s in straight traffic flow mode. This scenario with the OLT-App hosted into VM got a higher Network consumption, but the CPU and memory were still higher for the physical OLT hosting scene.

The following graphs in Figures 47 and 48 show the supplicant results for the same scenario (Docker in straight traffic mode), the first graph running on physical OLT, and the second on Virtual Machine.

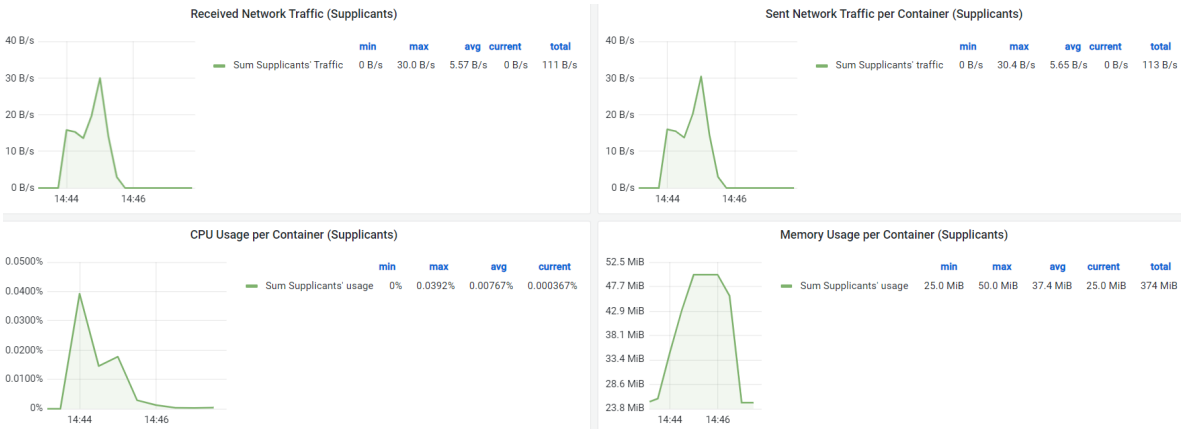


Figure 47: Results for Supplicants on physical OLT hosting environment of OLT-App and supplicants comparison.

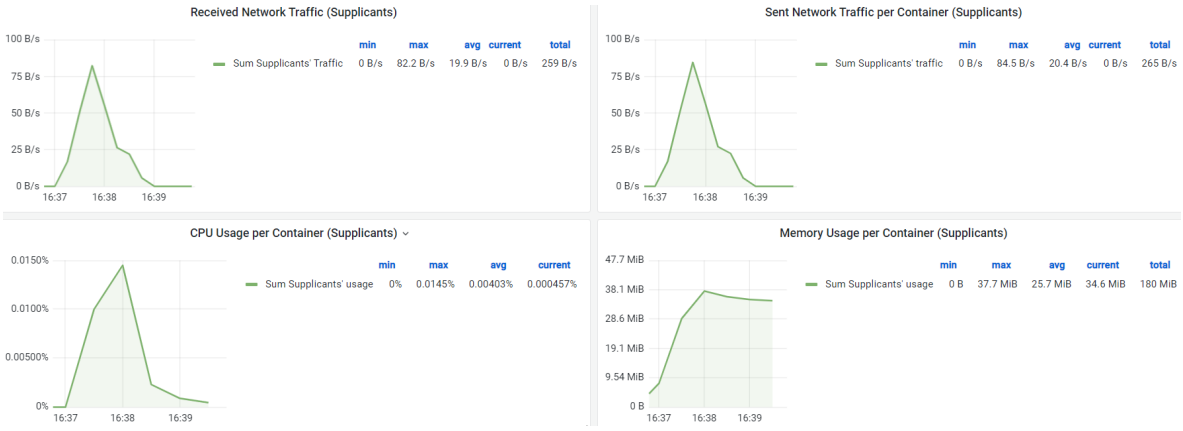


Figure 48: Supplicants on Virtual Machine results for hosting environment of OLT-App and supplicants comparison.

Comparing the Supplicants' results for these scenarios, the Network I/O consumption got higher values for the virtual machine hosting, which repeated for the other scenarios. The CPU and memory get high values on physical OLT when the core elements are hosted into Docker. When they are on K8s, the Supplicants' CPU is higher on Virtual Machine deploy-

ment. Moreover, the Supplicants' memory consumption is higher on physical OLT deployment than Virtual Machine when running into the scenario with core elements on K8s in straight traffic mode.

5.4 OVERVIEW

The last section made a result analysis and comparison between the proposed scenarios. Thus, here are the outcomes from this analysis.

The memory hardware of the physical OLT has a limitation because it does not perform swapping memory allocation. This constraint forces the scalability of tests to be careful with memory consumption, so the tests were based on this threshold. Moreover, the physical OLT has an ARM64 CPU architecture. However, beyond these points, regarding the table results, the hosting environment for supplicants and the OLT-App had better performance than Virtual Machine hosting. Hence, this better performance could be explained due to exactly higher consumption for Physical OLT hardware rather than VM for the same scalability scenarios.

Comparing the multiple scalability test, for 30, 60, and 90 supplicants, the processing time did not suffer an significant increase for all metrics. In some cases happened a decrease in processing time, becoming more performative. However, the results regarding the computational resources, the consumption were more predictable, for the main elements who were actively engaged on the tests had increased their consumption in most cases. Nevertheless, this increase in some cases was not proportional and linear to supplicant's scaling. Instead, it was exponential in some situations.

For instance, the network traffic and memory of VNF had exponential growth. Between the 30, 60, and 90 supplicants scales, the consumption doubled for each stage. One supposed reason for this behavior is that multiple supplicants could send the request simultaneously. It might also be noted, a script was configured to start the supplicants randomly between 0 and 20 seconds after its container started. Thus, the FreeRADIUS server could receive these requests near to which other. Hence these packets could be dropped, occasioning retransmission from supplicants. Furthermore, for more supplicants, the probability of this occurrence gets even higher.

Still regarding this issue about the FreeRADIUS Server, into result tables of VNF Statistics is possible to see how much time the FreeRADIUS consumes compared to the other process metrics. The VNF's process consumes around a dozen microseconds, and rather the FreeRADIUS consumes a few milliseconds up to a dozen milliseconds instead. So the FreeRADIUS is about 100 times slower than a VNF Process, becoming a bottleneck for the simulations.

Regarding the traffic flow mode, was identified that the Control Relay mode was more performative for the test on OLT-App statistics and also on VNF statistics when running on

Kubernetes environment, the straight mode was better running on Docker environment. Although the advantage of Control Relay mode deployment over Straight mode, the difference between them in several metrics was not ample in most of the results. Additionally, the Straight mode got some metric results much discrepant than Control Relay results. So the Control Relay setting seems more stable than Straight mode. Including one more element between the OLT and VNF traffic could be associated to add more delay. However, the result shows that this statement is not necessarily true; this third element could be acting as a buffer and stabilizing the burst traffic, but more tests in larger scenarios should be done to reach an accurate conclusion.

The hosting environments, the Docker and Kubernetes, did not have a well-defined winner overall. Although, Docker got better results in scenarios associated with physical OLT, while Kubernetes was better associated with OLT on VM. However, looking at the metrics separately, Kubernetes got a better performance for more metrics in total than Docker. The hardware consumption was slightly greater for Kubernetes than Docker. So considering that Kubernetes has a robust infrastructure set, it is expectable to also better results.

In a nutshell, it conclude that:

- the developed IEEE 802.1X Virtual Network Function got expected results. It gets process time results on a microsecond scale, allowing a VNF instance to process thousands of requests per second;
- the RADIUS server, however, presented as a significant bottleneck for the process;
- the utilization of the OB-BAA's Control Relay showed as stabilizer element, in a "proxy" role for the traffic between OLT and VNF. Even its addition to the network brought lower process time;
- the physical OLT tests had shown as a feasible deployment for the OLT-App module for communication with VNF. However, in simulations, the supplicants had to be also executed into physical OLT, consuming computational resources for this. Thus is expected to have an even better performance running only the OLT-App into physical OLT and the supplicants into physical ONT/ONUs, which were not available devices for testing during this project;
- the test validated the usage of Docker or Kubernetes as a hosting environment, showing that the proposed approach can benefit from containerized solutions.

5.5 SUMMARY

This chapter passed through the multiple test scenarios, combining different scales, hosting environments, and traffic flow modes, to simulate the developed VNF proposed in this work.

The chapter also presented the metrics to evaluate the simulation tests. Finally, was presented the main results achieved at the end of this project and its analysis.

6

CONCLUSION

This chapter resumes the developed project and the identified future works that complement the actual output.

6.1 RESUME OF CURRENT WORK

The study of the new network paradigms such as SDN, NFV, and the VNF was the focus of this project. Due to modern applications pressure demands, the network have changed so far, and these paradigms would answer these needs. Thus, considering this scenario, the project had a great value partnership with *Altice Labs* and intended to study this network advancements context applied to PON and develop a security VNF. Precisely, a function to implement the IEEE 802.1X authentication and authorization, relevant to *Altice Labs*' projects. Furthermore, this project is framed within the framework of the OB-BAA project from the BBF, one of the organizations that lead the adoption and development of new network technologies in broadband solutions, where *Altice Labs* contributes for.

After the initial studies concerning the PON technologies, the SDN, NFV, VNF concepts, and the 802.1X protocol (Dot1x), this project turned to the development phase. In this step, the development employed the Golang programming language due to its robustness and performance. So, this work developed a Golang VNF application that implements the Dot1x authenticator function, which receives authentication requests from supplicants, who would be in a real-world the *Optical Network Unit* (*ONU*), then consults the supplicants' credentials against a RADIUS server, and authorize or not their access to the network. This project focused on validating at least one authentication EAP method used by the Dot1x protocol. This method was the EAP-MD5, which is one of the most straight options with simpler deployment. s

For the simulations and test, the project leverage containers hosted into Kubernetes and Docker environments to host the main elements and simulate some scale scenarios with multiple supplicants. Ideally, the test would be more realistic if executed into physical supplicants (ONUs). However, there was no available real equipment for testing.

The main objective to develop a security VNF was accomplished. Moreover, the tests proved that performance was acceptable, with microseconds for processing time and handling thousands of messages per second. However, as already mentioned, the chosen Dot1x authentication method was a simple one, and a more robust and complex method should be tested to fully validate the solution's viability. Furthermore, the deployment was aligned with the BBF framework, so that this VNF could have an easier match and integration with OB-BAA's project components.

Beyond the VNF development itself, this work was a great opportunity to explore the Golang programming language, which many players are using. Moreover, the container concept is also used widely. It proved its value and the importance of the well-consolidated platforms to manage these containers, namely the Kubernetes and Docker. Without the container would be very difficult to create the proposed scenarios and test.

6.2 FUTURE WORKS

This project had developed a starting point for future improvements and implementations.

Regarding the possible future works concerning this project's topic, other authentication EAP methods could be implemented, e.g., EAP-TLS, EAP-TTLS, EAP-GTC, EAP-MSCHAP-V2, EAP-FAST EAP-PSK, and others. Also, the implementation of an authorization function on OLTs to grant or deny access to supplicants' connected interfaces. Moreover, checking what privilege of access these supplicants would have, setting VLANs and *Access Control Lists* (*ACLs*), for instance. Furthermore, the Accounting phase is also considered future work to account for the access logs of authenticated and authorized users.

In a traditional PON network, one OLT could have up to thousands of ONU/ONT attached to it. Thus, it would be valuable to run the simulations with higher-scale scenarios, near reality, to validate the performance of the developed VNF and OLT-App.

Finally, another theme for future work is deploying the Dot1x supplicant in the ONU/ONT equipment and rerun the tests with these physical devices.

A

APPENDIX

A.1 METRICS RESULTS: TABLES

Table 11: OLT-App Metrics Statistics for Scenario with: Physical OLT, Kubernetes and both Traffic Flow Modes

Measuring Point	OLT-App Statistics								
Suppl. & OLT App Hosting	OLT PHYSICAL								
Core Hosting Environment	K8S								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOLStart max. Time (ms):	6,058	18,139	23,972	16,056	5,183	17,062	19,303	13,849	
EAPOLStart min. time (ms):	1,327	1,400	1,348	1,358	1,338	1,262	1,315	1,305	
EAPOLStart avg. time (ms):	**2,128**	**3,043**	**4,263**	**3,145**	**3,801**	**2,471**	**2,579**	**2,950**	**CTRL_RELAY**
EAPReqID max. time (ms):	16,501	73,761	62,500	50,921	16,805	60,044	69,408	48,752	
EAPReqID min. time (ms):	1,668	2,775	4,025	2,823	1,333	2,788	3,901	2,674	
EAPReqID avg. time (ms):	**4,186**	**7,148**	**9,669**	**7,001**	**3,751**	**6,547**	**11,077**	**7,125**	**STRAIGHT**
EAPRespID max. time (ms):	18,440	19,553	22,933	20,309	14,432	16,089	17,067	15,863	
EAPRespID min. time (ms):	1,126	2,171	3,991	2,429	1,181	1,173	0,906	1,087	
EAPRespID avg. time (ms):	**2,786**	**4,911**	**8,372**	**5,356**	**2,697**	**2,807**	**4,172**	**3,225**	**CTRL_RELAY**
EAPChallReq max. time (ms):	25,383	57,078	80,865	54,442	21,901	53,715	51,730	42,449	
EAPChallReq min. time (ms):	1,659	2,808	3,885	2,784	1,665	2,764	3,852	2,761	
EAPChallReq avg. time (ms):	**11,129**	**7,776**	**10,329**	**9,745**	**4,611**	**7,128**	**10,173**	**7,304**	**CTRL_RELAY**
EAPChallResp max. time (ms):	20,544	697,524	2373,408	1030,492	29,238	18,688	19,930	22,619	
EAPChallResp min. time (ms):	0,636	671,383	2012,448	894,822	1,044	1,113	0,902	1,020	
EAPChallResp avg. time (ms):	**4,684**	**674,985**	**2022,563**	**900,744**	**3,457**	**2,871**	**4,657**	**3,662**	**CTRL_RELAY**
EAPSucc max. time (ms):	19,498	62,390	41,948	41,279	22,582	43,700	40,172	35,485	
EAPSucc min. time (ms):	1,663	2,789	3,839	2,764	1,669	2,764	3,821	2,751	
EAPSucc avg. time (ms):	**4,698**	**7,105**	**9,647**	**7,150**	**4,822**	**6,860**	**9,088**	**6,923**	**CTRL_RELAY**
Supplicant max. spent-time (ms):	164,489	184,648	484,026	277,721	173,584	196,004	1004,937	458,175	
Supplicant min. spent-time (ms):	21,523	24,232	28,225	24,660	21,375	24,186	29,108	24,889	
Supplicant avg. spent-time (ms):	**43,173**	**47,159**	**56,106**	**48,813**	**41,950**	**46,182**	**71,272**	**53,134**	**STRAIGHT**
Max. proc. request-per-sec.:	88	65	47	67	95	72	56	74	
Min. proc. request-per-sec.:	14	8	5	9	16	10	7	11	
Avg. proc. request-per-sec.:	**54**	**38**	**27**	**40**	**57**	**43**	**31**	**43**	**CTRL_RELAY**
OLT spent time (ms):	22,875	31,756	42,280	32,304	21,262	28,554	37,974	29,263	**CTRL_RELAY**

Table 12: OLT-App Metrics Statistics for Scenario with: Physical OLT, Docker and both Traffic Flow Modes

Measuring Point	OLT-App Statistics								
Suppl. & OLT App Hosting	OLT PHYSICAL								
Core Hosting Environment	DOCKER								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOLStart max. Time (ms):	6,058	18,139	16,779	13,659	11,454	13,899	7,640	10,997	
EAPOLStart min. time (ms):	1,327	1,400	1,627	1,451	1,480	1,537	0,481	1,166	
EAPOLStart avg. time (ms):	**2,128**	**3,043**	**3,558**	**2,910**	**3,023**	**3,262**	**1,090**	**2,458**	**CTRL_RELAY**
EAPReqID max. time (ms):	16,501	73,761	49,229	46,497	18,229	70,667	13,240	34,045	
EAPReqID min. time (ms):	1,668	2,775	3,899	2,781	1,642	2,747	1,337	1,909	
EAPReqID avg. time (ms):	**4,186**	**7,148**	**8,799**	**6,711**	**4,085**	**7,887**	**2,755**	**4,909**	**CTRL_RELAY**
EAPRespID max. time (ms):	18,440	19,553	21,356	12,664	9,445	32,777	8,707	16,977	
EAPRespID min. time (ms):	1,126	2,171	3,598	1,099	1,302	2,255	0,435	1,331	
EAPRespID avg. time (ms):	**2,786**	**4,911**	**7,844**	**2,566**	**2,671**	**5,958**	**1,216**	**3,282**	**STRAIGHT**
EAPChallReq max. time (ms):	25,383	57,078	34,297	38,919	35,617	57,072	33,043	41,911	
EAPChallReq min. time (ms):	1,659	2,808	3,802	2,756	1,720	2,840	1,336	1,965	
EAPChallReq avg. time (ms):	**5,602**	**7,776**	**7,704**	**7,027**	**6,240**	**8,921**	**4,742**	**6,634**	**CTRL_RELAY**
EAPChallResp max. time (ms):	27,281	697,524	3015,315	1246,707	27,740	708,524	11,506	249,257	
EAPChallResp min. time (ms):	0,978	671,383	2009,625	893,995	1,238	671,521	0,415	224,391	
EAPChallResp avg. time (ms):	**3,475**	**674,985**	**2028,896**	**902,452**	**4,551**	**676,051**	**1,226**	**227,276**	**CTRL_RELAY**
EAPSucc max. time (ms):	19,498	62,390	48,506	43,465	26,536	53,322	13,176	31,011	
EAPSucc min. time (ms):	1,663	2,789	3,859	2,770	1,662	2,792	1,309	1,921	
EAPSucc avg. time (ms):	**4,698**	**7,105**	**8,760**	**6,854**	**5,242**	**7,133**	**3,257**	**5,211**	**CTRL_RELAY**
Supplicant max. spent-time (ms):	164,489	184,648	1085,696	478,278	1001,821	929,353	462,226	797,800	
Supplicant min. spent-time (ms):	21,523	24,232	23,974	23,243	19,695	21,957	8,875	16,842	
Supplicant avg. spent-time (ms):	**43,173**	**47,159**	**55,096**	**48,476**	**89,800**	**79,162**	**27,662**	**65,541**	**STRAIGHT**
Max. proc. request-per-sec.:	88	65	52	69	83	63	16	54	
Min. proc. request-per-sec.:	14	8	12	11	13	8	2	8	
Avg. proc. request-per-sec.:	**54**	**38**	**33**	**42**	**48**	**34**	**9**	**31**	**STRAIGHT**
OLT spent time (ms):	22,875	31,756	38,482	18,210	25,812	35,820	14,287	25,307	**STRAIGHT**

Table 13: OLT-App Metrics Statistics for Scenario with: Virtual Machine, Kubernetes and both Traffic Flow Modes

Measuring Point	OLT-App Statistics								
Suppl. & OLT App Hosting	OLT VM								
Core Hosting Environment	K8S								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOLStart max. Time (ms):	8,146	12,472	539,480	242,031	5,451	7,648	178,756	149,970	
EAPOLStart min. time (ms):	0,952	0,889	1,038	0,939	0,933	0,882	0,840	0,883	
EAPOLStart avg. time (ms):	**2,485**	**2,617**	**28,184**	**15,822**	**1,971**	**2,257**	**13,103**	**11,833**	**CTRL_RELAY**
EAPReqID max. time (ms):	8,782	10,357	457,199	191,004	2,570	5,477	92,972	97,174	
EAPReqID min. time (ms):	0,609	0,673	0,575	0,610	0,634	0,597	0,563	0,594	
EAPReqID avg. time (ms):	**1,337**	**1,885**	**20,109**	**9,043**	**1,125**	**1,091**	**5,655**	**5,177**	**CTRL_RELAY**
EAPRespID max. time (ms):	9,842	15,878	130,416	56,452	4,898	6,537	9,522	22,658	
EAPRespID min. time (ms):	2,753	3,564	3,377	2,586	0,716	0,651	0,644	0,666	
EAPRespID avg. time (ms):	**4,831**	**6,370**	**13,057**	**7,116**	**1,386**	**1,478**	**1,846**	**2,229**	**CTRL_RELAY**
EAPChallReq max. time (ms):	698,017	8,733	103,269	231,251	4,171	3,376	76,479	49,753	
EAPChallReq min. time (ms):	671,644	0,544	0,564	168,321	0,568	0,540	0,534	0,544	
EAPChallReq avg. time (ms):	**674,700**	**1,283**	**4,397**	**171,539**	**1,096**	**0,880**	**3,484**	**2,808**	**CTRL_RELAY**
EAPChallResp max. time (ms):	1384,89	2261,98	2723,47	1601,82	7,287	6,768	13,567	16,141	
EAPChallResp min. time (ms):	1342,82	1678,31	2013,20	1258,74	0,650	0,608	0,625	0,628	
EAPChallResp avg. time (ms):	**1349,20**	**2030,37**	**2080,16**	**1365,60**	**1,451**	**1,396**	**2,108**	**1,907**	**CTRL_RELAY**
EAPSucc max. time (ms):	7,348	5,381	233,827	94,305	3,041	5,524	84,269	55,875	
EAPSucc min. time (ms):	0,533	0,537	0,552	0,533	0,553	0,529	0,512	0,526	
EAPSucc avg. time (ms):	**1,090**	**1,000**	**10,143**	**4,626**	**0,997**	**0,972**	**4,466**	**3,176**	**CTRL_RELAY**
Supplicant max. spent-time (ms):	45,869	214,79	1253,61	638,15	517,18	261,35	650,66	616,88	
Supplicant min. spent-time (ms):	11,147	11,050	11,478	11,282	12,170	11,236	12,173	11,758	
Supplicant avg. spent-time (ms):	**18,797**	**39,453**	**136,433**	**74,318**	**35,739**	**25,903**	**65,973**	**57,551**	**CTRL_RELAY**
Max. proc. request-per-sec.:	142	147	146	167	218	239	234	231	
Min. proc. request-per-sec.:	38	32	16	33	69	58	28	50	
Avg. proc. request-per-sec.:	**88**	**89**	**76**	**96**	**140**	**149**	**127**	**137**	**CTRL_RELAY**
OLT spent time (ms):	12,562	13,183	58,656	31,571	7,807	7,813	29,052	21,639	**CTRL_RELAY**

Table 14: OLT-App Metrics Statistics for Scenario with: Virtual Machine, Docker and both Traffic Flow Modes

Measuring Point	OLT-App Statistics								
Suppl. & OLT App Hosting	OLT VM								
Core Hosting Environment	DOCKER								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOLStart max. Time (ms):	281,49	1289,74	1934,24	1168,49	231,09	1789,50	1934,24	1318,28	
EAPOLStart min. time (ms):	0,60	0,60	0,57	0,59	0,60	0,56	0,57	0,58	
EAPOLStart avg. time (ms):	**13,91**	**122,53**	**180,24**	**105,56**	**8,45**	**141,81**	**180,24**	**110,17**	**STRAIGHT**
EAPReqID max. time (ms):	60,71	81,12	392,10	177,97	4,81	262,43	392,10	219,78	
EAPReqID min. time (ms):	0,62	0,60	0,55	0,59	0,54	0,56	0,55	0,55	
EAPReqID avg. time (ms):	**3,51**	**4,93**	**13,10**	**7,18**	**1,17**	**12,98**	**13,10**	**9,08**	**STRAIGHT**
EAPRespID max. time (ms):	486,21	1266,21	482,47	744,96	321,35	158,25	482,47	320,69	
EAPRespID min. time (ms):	1,70	1,59	0,38	1,23	0,42	0,40	0,38	0,40	
EAPRespID avg. time (ms):	**25,10**	**84,01**	**38,58**	**49,23**	**10,39**	**11,25**	**38,58**	**20,08**	**CTRL_RELAY**
EAPChallReq max. time (ms):	33,96	45,20	106,91	62,02	17,74	42,74	106,91	55,80	
EAPChallReq min. time (ms):	0,56	0,54	0,54	0,55	0,51	0,52	0,54	0,52	
EAPChallReq avg. time (ms):	**2,90**	**3,07**	**5,72**	**3,90**	**1,42**	**2,94**	**5,72**	**3,36**	**CTRL_RELAY**
EAPChallResp max. time (ms):	4578,08	5886,44	275,28	3579,93	14,16	47,77	275,28	112,40	
EAPChallResp min. time (ms):	2018,71	2024,19	0,39	1347,76	0,36	0,38	0,39	0,37	
EAPChallResp avg. time (ms):	**2327,17**	**2411,17**	**29,14**	**1589,16**	**1,64**	**5,43**	**29,14**	**12,07**	**CTRL_RELAY**
EAPSucc max. time (ms):	95,68	87,64	63,16	82,16	7,14	28,51	63,16	32,94	
EAPSucc min. time (ms):	0,55	0,54	0,52	0,54	0,51	0,51	0,52	0,51	
EAPSucc avg. time (ms):	**5,42**	**3,75**	**3,12**	**4,10**	**1,08**	**1,98**	**3,12**	**2,06**	**CTRL_RELAY**
Supplicant max. spent-time (ms):	1505,38	1965,57	4438,08	2636,34	2130,76	3285,39	4438,08	3284,74	
Supplicant min. spent-time (ms):	7,73	8,99	10,84	9,19	8,95	9,18	10,84	9,66	
Supplicant avg. spent-time (ms):	**118,26**	**302,24**	**583,64**	**334,72**	**166,37**	**253,20**	**583,64**	**334,41**	**CTRL_RELAY**
Max. proc. request-per-sec.:	210	220	286	239	302	293	286	293	
Min. proc. request-per-sec.:	6	14	14	11	26	11	14	17	
Avg. proc. request-per-sec.:	**89**	**96**	**131**	**105**	**154**	**137**	**131**	**140**	**CTRL_RELAY**
OLT spent time (ms):	50,84	177,93	165,58	131,45	24,13	82,08	165,58	90,60	**CTRL_RELAY**

Table 15: VNF Metrics Statistics for Scenario with: Physical OLT, Kubernetes and both Traffic Flow Modes

Measuring Point	VNF Statistics								
Suppl. & OLT App Hosting	OLT PHYSICAL								
Core Hosting Environment	K8S								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOL max. time (ms):	0,045	0,083	0,101	0,076	0,177	0,343	0,364	0,295	
EAPOL min. time (ms):	0,017	0,017	0,016	0,017	0,017	0,017	0,016	0,016	
EAPOL avg. time (ms):	**0,032**	**0,033**	**0,033**	**0,033**	**0,036**	**0,036**	**0,032**	**0,035**	**STRAIGHT**
EAP2Radius max. time (ms):	0,411	0,228	0,112	0,250	0,247	0,115	0,153	0,172	
EAP2Radius min. time (ms):	0,023	0,016	0,018	0,019	0,023	0,016	0,016	0,018	
EAP2Radius avg. time (ms):	**0,055**	**0,043**	**0,039**	**0,046**	**0,050**	**0,041**	**0,038**	**0,043**	**CTRL_RELAY**
Radius max. time (ms):	11,937	9,930	340,838	120,902	8,781	7,863	805,921	274,188	
Radius min. time (ms):	2,087	1,949	1,738	1,925	1,811	1,935	2,100	1,949	
Radius avg. time (ms):	**4,405**	**4,569**	**5,971**	**4,982**	**3,628**	**3,668**	**12,979**	**6,758**	**STRAIGHT**
Radius2EAP max. time (ms):	0,121	0,056	0,090	0,089	0,063	0,061	0,097	0,074	
Radius2EAP min. time (ms):	0,014	0,014	0,014	0,014	0,014	0,013	0,013	0,014	
Radius2EAP avg. time (ms):	**0,032**	**0,027**	**0,030**	**0,030**	**0,029**	**0,027**	**0,030**	**0,029**	**CTRL_RELAY**
EAP2Radius2 max. time (ms):	0,075	0,072	0,248	0,132	0,068	0,203	0,152	0,141	
EAP2Radius2 min. time (ms):	0,016	0,015	0,015	0,015	0,019	0,015	0,016	0,016	
EAP2Radius2 avg. time (ms):	**0,037**	**0,033**	**0,041**	**0,037**	**0,037**	**0,036**	**0,037**	**0,037**	**CTRL_RELAY**
Radius max. time (ms):	11,937	9,930	340,838	120,902	8,781	7,863	805,921	274,188	
Radius min. time (ms):	1,995	1,949	1,738	1,894	1,811	1,935	2,100	1,949	
Radius avg. time (ms):	**4,372**	**4,551**	**5,954**	**4,959**	**3,615**	**3,656**	**12,930**	**6,734**	**STRAIGHT**
Radius2EAP2 max. time (ms):	0,062	0,303	0,078	0,148	0,095	0,151	0,071	0,106	
Radius2EAP2 min. time (ms):	0,011	0,011	0,011	0,011	0,013	0,011	0,011	0,012	
Radius2EAP2 avg. time (ms):	**0,028**	**0,028**	**0,028**	**0,028**	**0,029**	**0,025**	**0,026**	**0,027**	**CTRL_RELAY**
FullSession max. time (ms):	66487,7	43874,1	100281,4	70214,4	201311,1	93595,0	37504,5	110803,5	
FullSession min. time (ms):	15,714	17,816	19,760	17,763	15,834	17,329	20,822	17,995	
FullSession avg. time (ms):	**28,386**	**35,318**	**41,855**	**35,186**	**32,115**	**32,780**	**59,633**	**41,509**	**STRAIGHT**
Max. proc. request-per-sec.:	10260	10033	10928	10407	9068	10342	9994	9801	
Min. proc. request-per-sec.:	2906	2356	2870	2711	2414	2822	2851	2696	
Avg. proc. request-per-sec.:	**6083**	**6597**	**6219**	**6300**	**6044**	**6673**	**6522**	**6413**	**CTRL_RELAY**
VNF spent time (ms):	0,177	0,144	0,174	0,165	0,148	0,139	0,149	0,145	**CTRL_RELAY**

Table 16: VNF Metrics Statistics for Scenario with: Physical OLT, Docker and both Traffic Flow Modes

Measuring Point	VNF Statistics								
Suppl. & OLT App Hosting	OLT PHYSICAL								
Core Hosting Environment	DOCKER								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOL max. time (ms):	0,133	0,101	0,207	0,147	0,119	0,268	0,591	0,326	
EAPOL min. time (ms):	0,016	0,015	0,016	0,016	0,017	0,013	0,014	0,015	
EAPOL avg. time (ms):	**0,034**	**0,030**	**0,032**	**0,032**	**0,033**	**0,037**	**0,040**	**0,037**	**STRAIGHT**
EAP2Radius max. time (ms):	0,199	0,157	0,132	0,163	0,463	0,197	0,155	0,272	
EAP2Radius min. time (ms):	0,029	0,026	0,023	0,026	0,024	0,026	0,023	0,025	
EAP2Radius avg. time (ms):	**0,055**	**0,049**	**0,055**	**0,053**	**0,058**	**0,046**	**0,048**	**0,051**	**CTRL_RELAY**
Radius max. time (ms):	6,662	1001,316	1002,398	670,126	12,436	6,373	1001,612	340,141	
Radius min. time (ms):	0,683	0,595	0,587	0,622	1,395	0,675	0,627	0,899	
Radius avg. time (ms):	**1,490**	**9,676**	**7,137**	**6,101**	**3,064**	**9,595**	**7,228**	**6,629**	**STRAIGHT**
Radius2EAP max. time (ms):	0,060	0,247	0,057	0,121	0,040	0,111	0,169	0,106	
Radius2EAP min. time (ms):	0,014	0,013	0,013	0,013	0,014	0,014	0,012	0,013	
Radius2EAP avg. time (ms):	**0,023**	**0,026**	**0,026**	**0,025**	**0,024**	**0,026**	**0,026**	**0,025**	**STRAIGHT**
EAP2Radius2 max. time (ms):	0,154	0,129	0,171	0,151	0,068	0,097	0,394	0,187	
EAP2Radius2 min. time (ms):	0,020	0,021	0,020	0,020	0,023	0,020	0,020	0,021	
EAP2Radius2 avg. time (ms):	**0,042**	**0,040**	**0,045**	**0,042**	**0,037**	**0,039**	**0,046**	**0,041**	**CTRL_RELAY**
Radius max. time (ms):	6,662	1001,316	1002,398	670,126	12,436	6,373	1001,612	340,141	
Radius min. time (ms):	0,683	0,595	0,587	0,622	1,395	0,675	0,627	0,899	
Radius avg. time (ms):	**1,491**	**9,605**	**7,105**	**6,067**	**3,051**	**9,529**	**7,196**	**6,592**	**STRAIGHT**
Radius2EAP2 max. time (ms):	0,092	0,059	0,149	0,100	0,060	0,192	0,080	0,111	
Radius2EAP2 min. time (ms):	0,013	0,011	0,013	0,012	0,013	0,013	0,007	0,011	
Radius2EAP2 avg. time (ms):	**0,023**	**0,022**	**0,028**	**0,024**	**0,022**	**0,025**	**0,025**	**0,024**	**STRAIGHT**
FullSession max. time (ms):	95704,69	1069,79	1078,22	32617,57	241365,35	467,19	1364,67	81065,74	
FullSession min. time (ms):	11,343	13,327	16,270	13,646	14,665	13,776	15,684	14,708	
FullSession avg. time (ms):	**25,577**	**46,021**	**41,027**	**37,542**	**35,964**	**72,090**	**63,859**	**57,304**	**STRAIGHT**
Max. proc. request-per-sec.:	9075	10542	9611	9743	9464	9245	10652	9787	
Min. proc. request-per-sec.:	2931	2443	2694	2689	1800	2823	1248	1957	
Avg. proc. request-per-sec.:	**6272**	**6542**	**5857**	**6223**	**6483**	**6432**	**6044**	**6320**	**CTRL_RELAY**
VNF spent time (ms):	0,181	0,195	0,115	0,164	0,134	0,147	0,143	0,142	**CTRL_RELAY**

Table 17: VNF Metrics Statistics for Scenario with: Virtual Machine, Kubernetes and both Traffic Flow Modes

Measuring Point	VNF Statistics								
Suppl. & OLT App Hosting	OLT VM								
Core Hosting Environment	K8S								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOL max. time (ms):	0,094	0,065	0,091	0,086	0,074	0,524	0,157	0,211	
EAPOL min. time (ms):	0,013	0,013	0,012	0,013	0,017	0,015	0,012	0,015	
EAPOL avg. time (ms):	**0,033**	**0,028**	**0,029**	**0,030**	**0,034**	**0,038**	**0,030**	**0,033**	**STRAIGHT**
EAP2Radius max. time (ms):	0,105	0,491	0,498	0,299	0,090	0,138	0,270	0,147	
EAP2Radius min. time (ms):	0,016	0,017	0,017	0,016	0,020	0,015	0,016	0,017	
EAP2Radius avg. time (ms):	**0,041**	**0,045**	**0,045**	**0,042**	**0,044**	**0,040**	**0,038**	**0,040**	**CTRL_RELAY**
Radius max. time (ms):	11,047	348,739	1004,416	591,995	405,308	409,928	607,532	606,684	
Radius min. time (ms):	1,952	1,887	1,321	1,675	1,650	1,724	1,755	1,631	
Radius avg. time (ms):	**3,947**	**13,969**	**23,495**	**16,242**	**9,896**	**8,153**	**12,069**	**14,394**	**CTRL_RELAY**
Radius2EAP max. time (ms):	0,081	0,081	0,922	0,288	0,057	0,070	0,057	0,061	
Radius2EAP min. time (ms):	0,012	0,012	0,013	0,012	0,014	0,012	0,013	0,013	
Radius2EAP avg. time (ms):	**0,030**	**0,028**	**0,037**	**0,030**	**0,029**	**0,024**	**0,025**	**0,026**	**CTRL_RELAY**
EAP2Radius2 max. time (ms):	0,153	0,112	0,086	0,119	0,087	0,184	0,083	0,116	
EAP2Radius2 min. time (ms):	0,017	0,013	0,015	0,015	0,019	0,016	0,016	0,016	
EAP2Radius2 avg. time (ms):	**0,037**	**0,032**	**0,034**	**0,034**	**0,039**	**0,037**	**0,033**	**0,036**	**STRAIGHT**
Radius max. time (ms):	11,047	348,739	1004,416	591,995	405,308	409,928	607,532	606,684	
Radius min. time (ms):	1,952	1,887	1,321	1,675	1,650	1,724	1,745	1,629	
Radius avg. time (ms):	**3,933**	**13,884**	**23,383**	**16,119**	**9,782**	**8,109**	**12,003**	**14,257**	**CTRL_RELAY**
Radius2EAP2 max. time (ms):	0,187	0,107	0,103	0,123	0,053	0,183	0,141	0,115	
Radius2EAP2 min. time (ms):	0,010	0,011	0,010	0,010	0,012	0,010	0,010	0,011	
Radius2EAP2 avg. time (ms):	**0,030**	**0,027**	**0,025**	**0,027**	**0,025**	**0,023**	**0,029**	**0,025**	**CTRL_RELAY**
FullSession max. time (ms):	37,760	383,161	2291,817	1311,299	512,964	488,498	809,517	1131,817	
FullSession min. time (ms):	9,117	8,833	8,758	8,875	9,868	8,703	9,394	9,202	
FullSession avg. time (ms):	**15,589**	**37,348**	**116,683**	**62,981**	**32,453**	**26,003**	**43,390**	**56,495**	**CTRL_RELAY**
Max. proc. request-per-sec.:	12693	11394	11142	11610	9236	11282	11132	10683	
Min. proc. request-per-sec.:	2539	2011	982	2304	3907	2302	2650	3136	
Avg. proc. request-per-sec.:	**6796**	**7034**	**6708**	**6907**	**6119**	**6967**	**7292**	**6873**	**STRAIGHT**
VNF spent time (ms):	0,111	0,133	0,129	0,130	0,167	0,154	0,146	0,150	**STRAIGHT**

Table 18: VNF Metrics Statistics for Scenario with: Virtual Machine, Docker and both Traffic Flow Modes

Measuring Point	VNF Statistics								
Suppl. & OLT App Hosting	OLT VM								
Core Hosting Environment	DOCKER								
Traffic Flow Mode	STRAIGHT				CONTROL RELAY				
Number of Supplicants	**30**	**60**	**90**	**Average**	**30**	**60**	**90**	**Average**	**Best Mode**
EAPOL max. time (ms):	0,069	0,115	5,426	1,870	0,103	1,579	5,426	2,369	
EAPOL min. time (ms):	0,017	0,013	0,008	0,013	0,015	0,006	0,008	0,010	
EAPOL avg. time (ms):	**0,029**	**0,030**	**0,076**	**0,045**	**0,032**	**0,052**	**0,076**	**0,053**	**STRAIGHT**
EAP2Radius max. time (ms):	7,119	164,465	366,349	179,311	1,757	7,082	366,349	125,063	
EAP2Radius min. time (ms):	0,025	0,025	0,019	0,023	0,023	0,018	0,019	0,020	
EAP2Radius avg. time (ms):	**0,304**	**5,112**	**3,210**	**2,875**	**0,110**	**0,174**	**3,210**	**1,165**	**CTRL_RELAY**
Radius max. time (ms):	1050,57	1872,98	1508,91	1477,49	1037,41	692,30	1508,91	1079,54	
Radius min. time (ms):	0,73	0,97	0,62	0,77	0,66	0,64	0,62	0,64	
Radius avg. time (ms):	**35,15**	**93,26**	**138,87**	**89,09**	**53,46**	**91,71**	**138,87**	**94,68**	**STRAIGHT**
Radius2EAP max. time (ms):	2,132	33,244	1,670	12,349	0,177	87,012	1,670	29,620	
Radius2EAP min. time (ms):	0,011	0,011	0,005	0,009	0,006	0,005	0,005	0,005	
Radius2EAP avg. time (ms):	**0,093**	**1,105**	**0,037**	**0,412**	**0,028**	**1,304**	**0,037**	**0,456**	**STRAIGHT**
EAP2Radius2 max. time (ms):	2,869	300,955	65,284	123,036	0,143	10,770	65,284	25,399	
EAP2Radius2 min. time (ms):	0,021	0,021	0,017	0,020	0,019	0,017	0,017	0,018	
EAP2Radius2 avg. time (ms):	**0,133**	**4,571**	**0,599**	**1,768**	**0,040**	**0,176**	**0,599**	**0,271**	**CTRL_RELAY**
Radius max. time (ms):	1050,569	1872,979	1508,910	1477,486	1037,411	692,299	1508,910	1079,540	
Radius min. time (ms):	0,726	0,974	0,624	0,775	0,659	0,644	0,624	0,642	
Radius avg. time (ms):	**34,612**	**92,405**	**137,917**	**88,311**	**52,726**	**91,097**	**137,917**	**93,914**	**STRAIGHT**
Radius2EAP2 max. time (ms):	0,058	21,579	3,961	8,533	0,064	3,644	3,961	2,556	
Radius2EAP2 min. time (ms):	0,013	0,013	0,007	0,011	0,010	0,009	0,007	0,008	
Radius2EAP2 avg. time (ms):	**0,023**	**0,653**	**0,097**	**0,258**	**0,022**	**0,079**	**0,097**	**0,066**	**CTRL_RELAY**
FullSession max. time (ms):	1238,83	2523,41	4858,47	2873,57	3049,29	1907,84	4858,47	3271,87	
FullSession min. time (ms):	5,84	8,08	8,31	7,41	6,28	5,99	8,31	6,86	
FullSession avg. time (ms):	**98,13**	**256,65**	**636,07**	**330,28**	**266,68**	**365,90**	**636,07**	**422,88**	**STRAIGHT**
Max. proc. request-per-sec.:	10022	9985	13186	11065	10363	13939	13186	12496	
Min. proc. request-per-sec.:	1240	1574	799	1204	1884	56	799	913	
Avg. proc. request-per-sec.:	**6284**	**6591**	**7376**	**6750**	**6418**	**7097**	**7376**	**6964**	**CTRL_RELAY**
VNF spent time (ms):	0,143	0,170	0,211	0,175	0,204	0,152	0,211	0,189	**STRAIGHT**

A.2 GRAPH METRICS RESULTS

Scenario 1: Physical OLT / Docker/ Control Relay

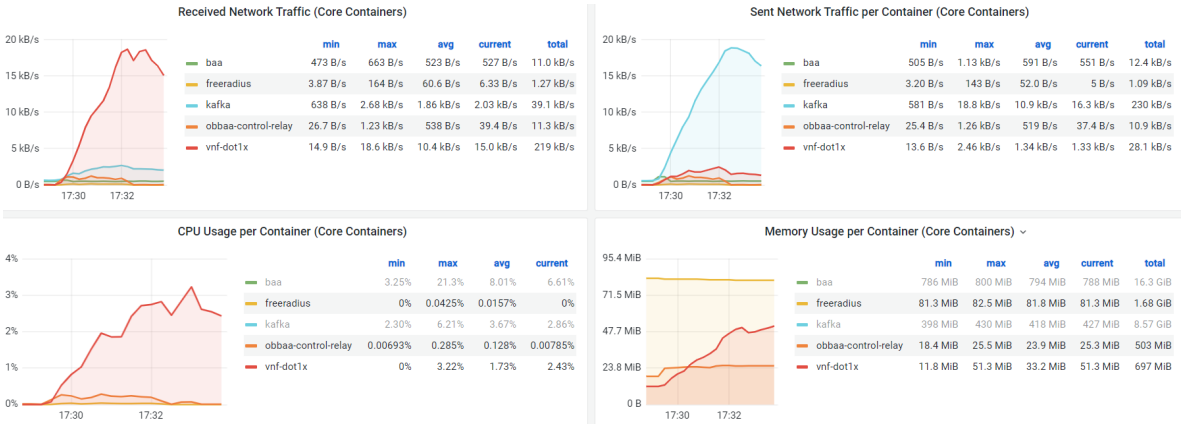


Figure 49: Scenario 1 - Consolidated metrics for Core elements hosted into Docker and traffic in Control Relay mode.

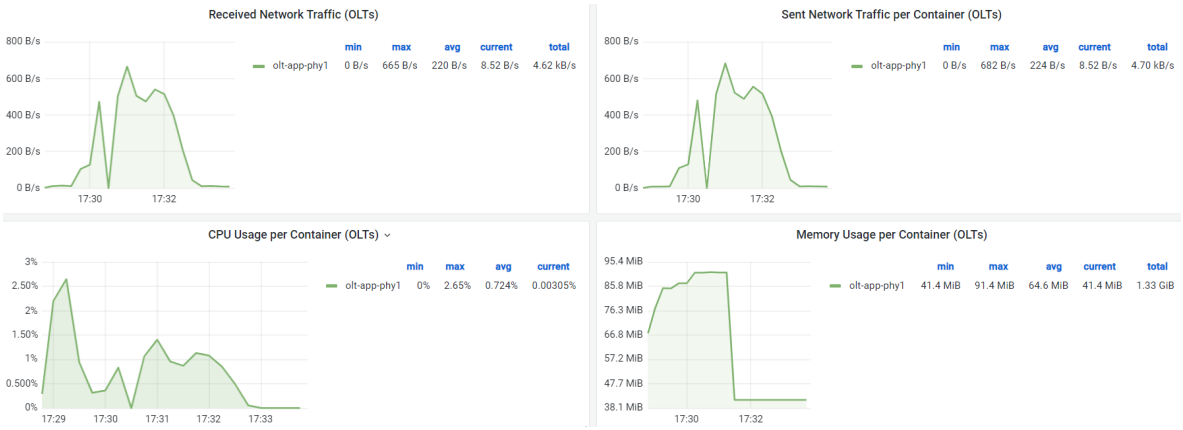


Figure 50: Scenario 1 - Consolidated metrics for OLT-App into physical OLT and traffic in Control Relay mode.

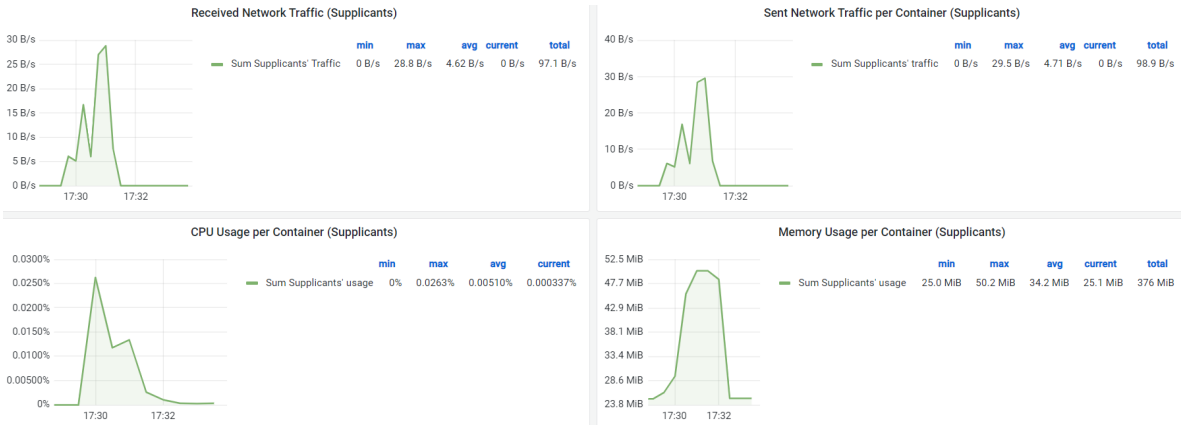


Figure 51: Scenario 1 - Consolidated metric for Supplicants hosted into Docker and traffic in Control Relay mode.

Scenario 2: Physical OLT / Docker / Straight

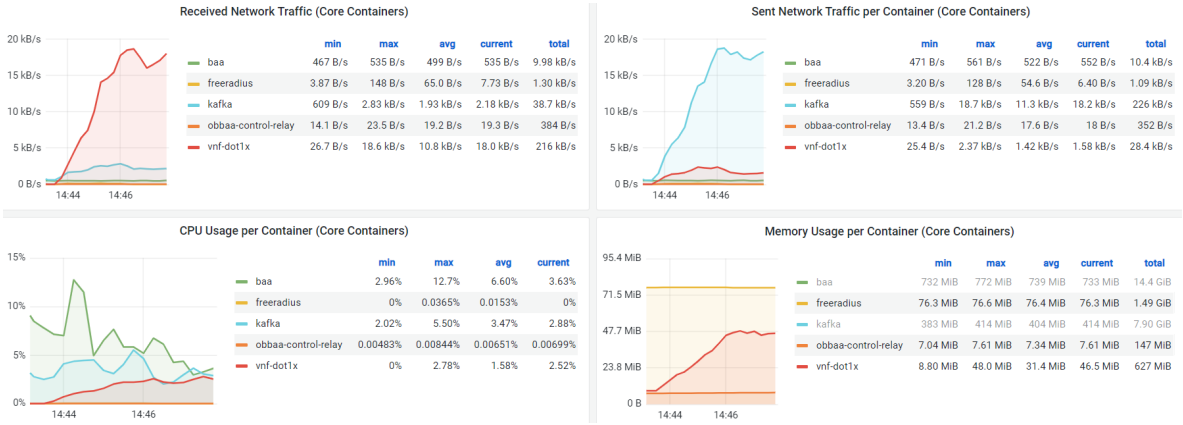


Figure 52: Scenario 2 - Consolidated metrics for Core elements into Docker and traffic in straight mode.

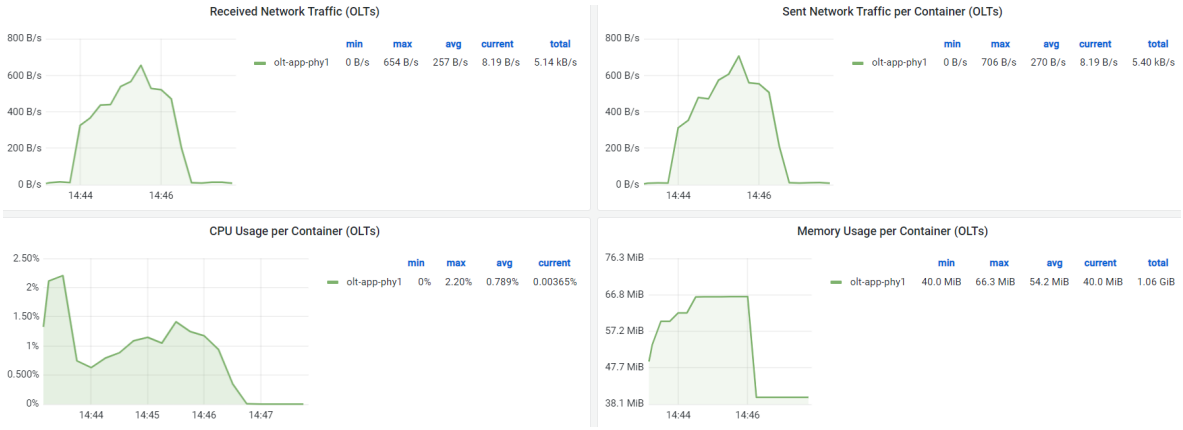


Figure 53: Scenario 2 - Consolidated metrics for OLT-App into physical OLT and traffic in straight mode.

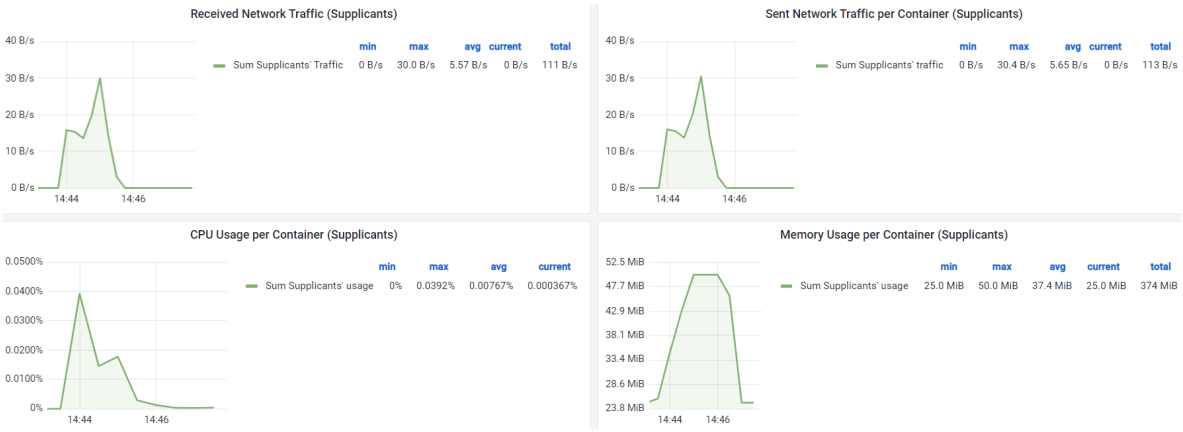


Figure 54: Scenario 2 - Consolidated metrics for Supplicants hosted into Docker and traffic in Straight mode.

Scenario 3: Physical OLT / Kubernetes / Control Relay

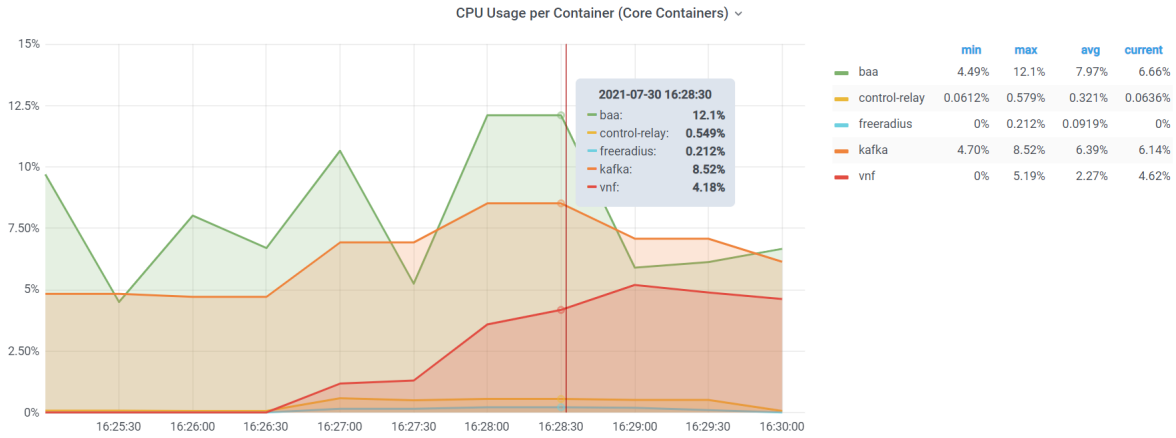


Figure 55: Scenario 3 - CPU metric for Core elements hosted into K8S and traffic in Control Relay mode.

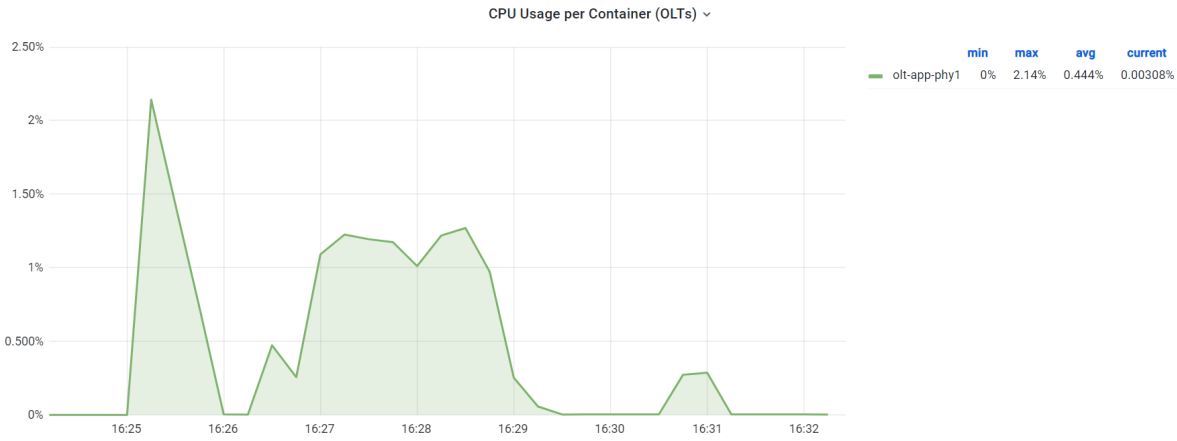


Figure 56: Scenario 3 - CPU metric for OLT-App into physical OLT and traffic in Control Relay mode.

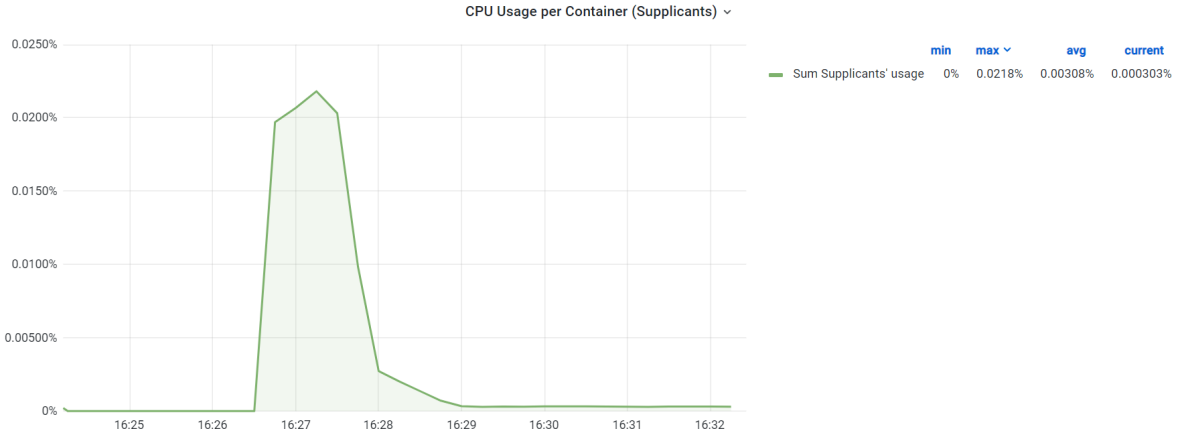


Figure 57: Scenario 3 - CPU metric for Supplicants hosted into K8S and traffic in Control Relay mode.

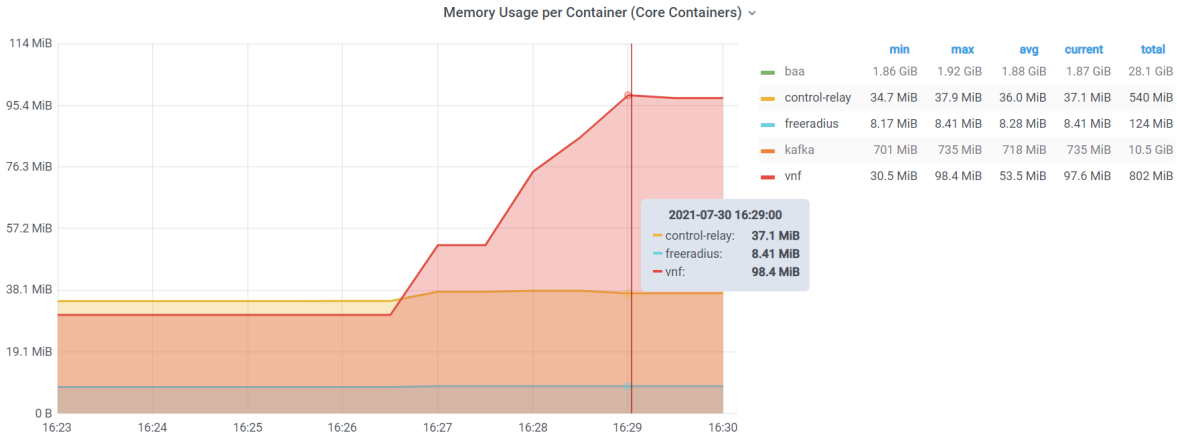


Figure 58: Scenario 3 - Memory metric for Core elements hosted into K8S and traffic in Control Relay mode.

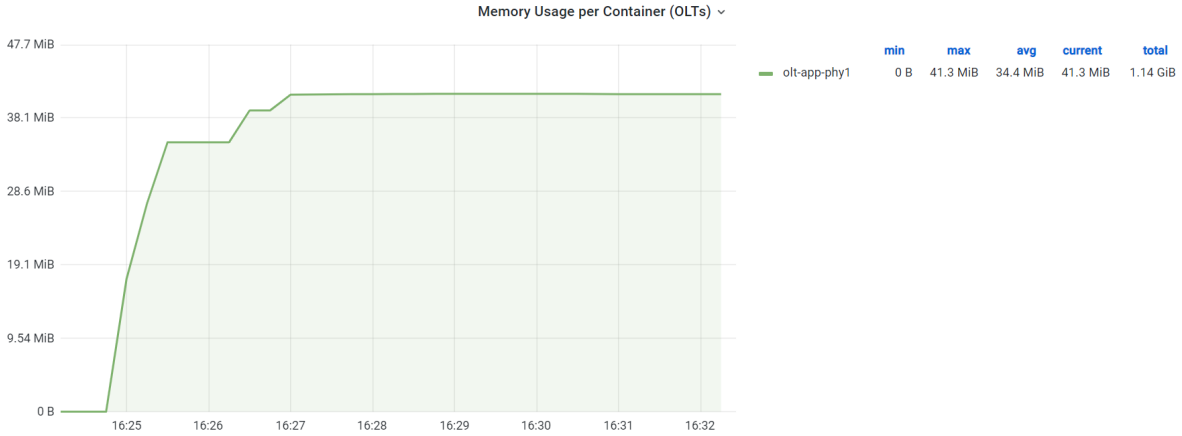


Figure 59: Scenario 3 - Memory metric for OLT-App into physical OLT and traffic in Control Relay mode.

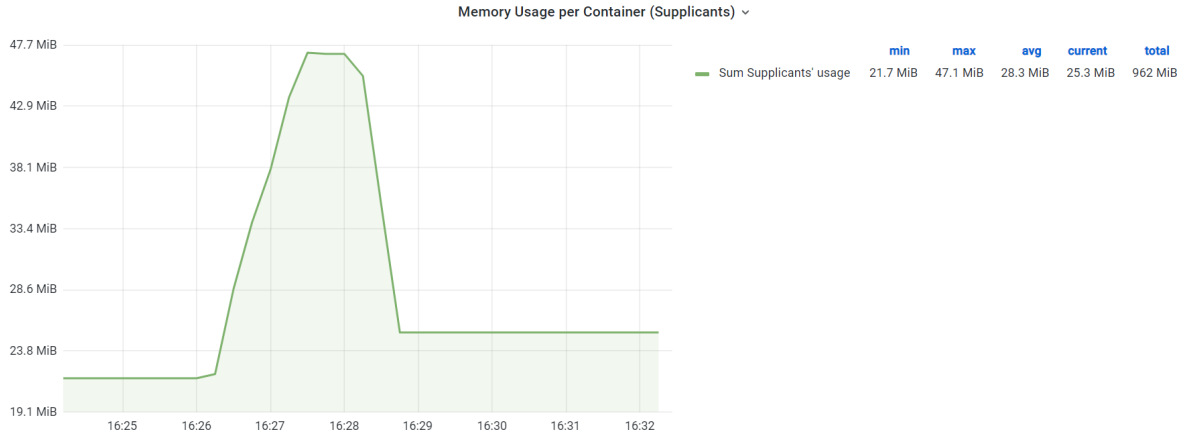


Figure 60: Scenario 3 - Memory metric for Supplicants hosted into K8S and traffic in Control Relay mode.

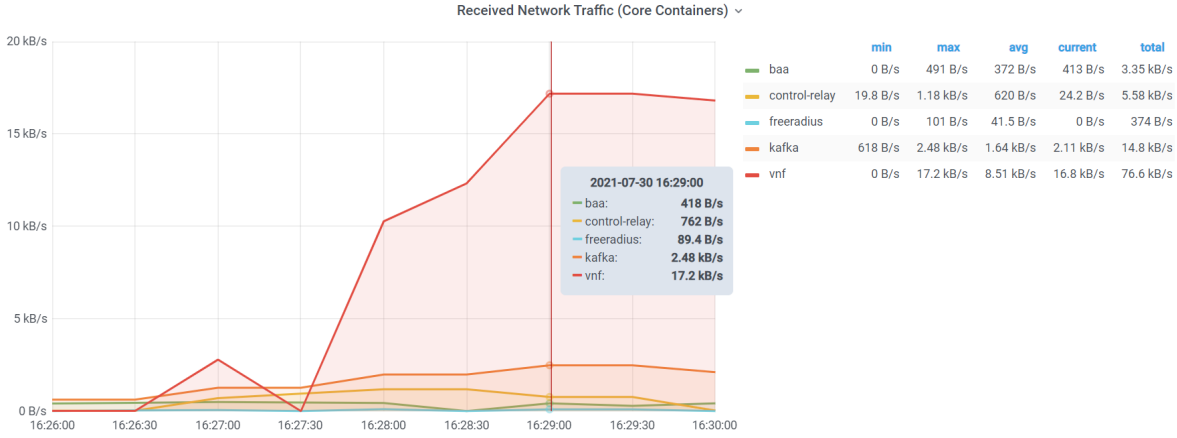


Figure 61: Scenario 3 - Network received traffic for Core elements hosted into K8S and traffic in Control Relay mode.

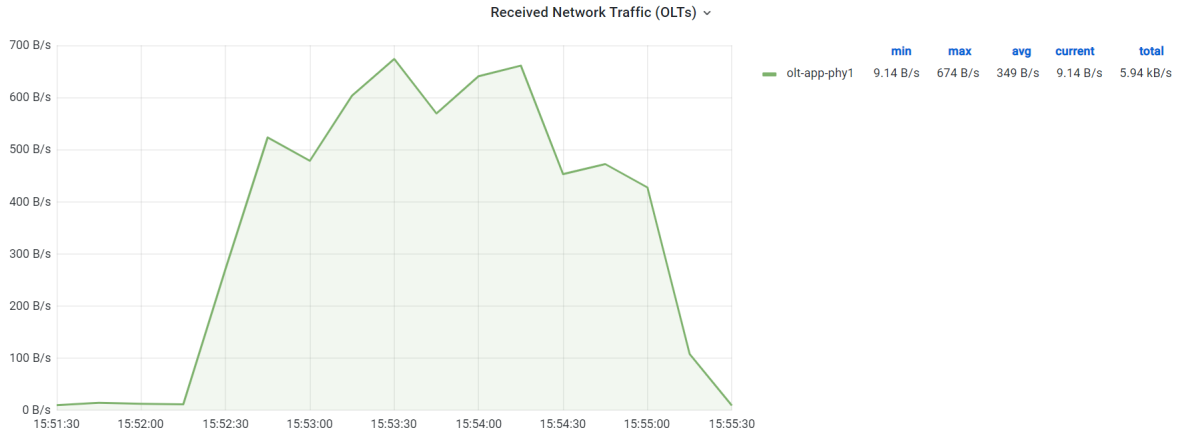


Figure 62: Scenario 3 - Network received traffic for OLT-App into physical OLT and traffic in Control Relay mode.

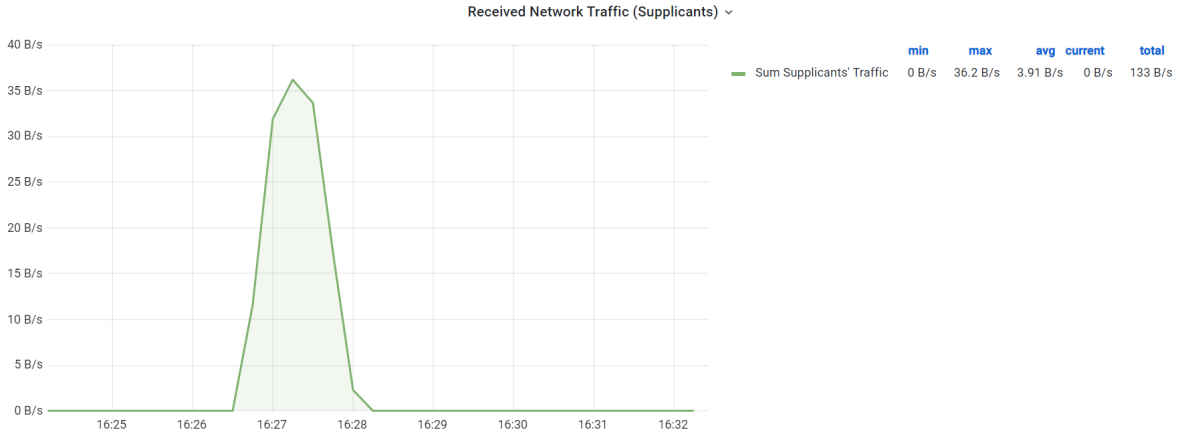


Figure 63: Scenario 3 - Network received traffic metric for Supplicants hosted into K8S and traffic in Control Relay mode.

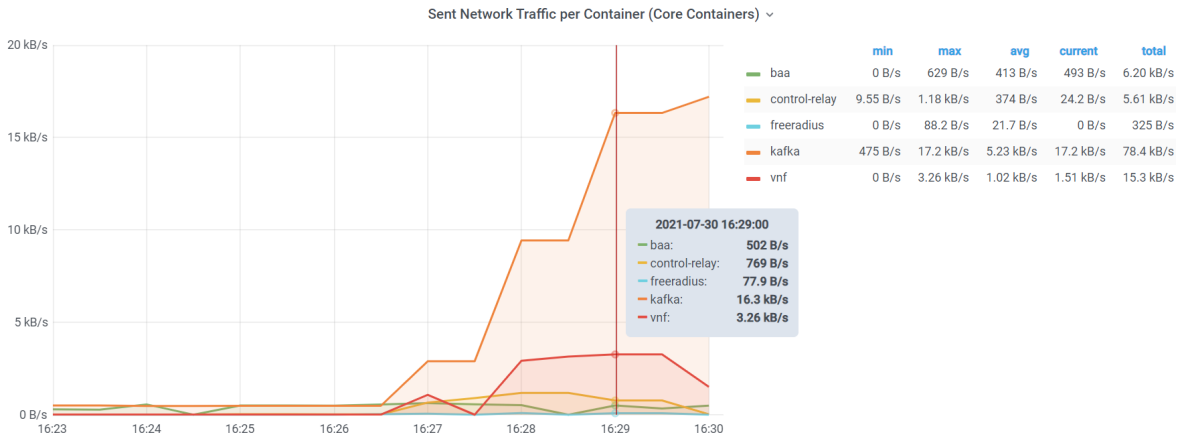


Figure 64: Scenario 3 - Network sent traffic metric for Core elements hosted into K8S and traffic in Control Relay mode.

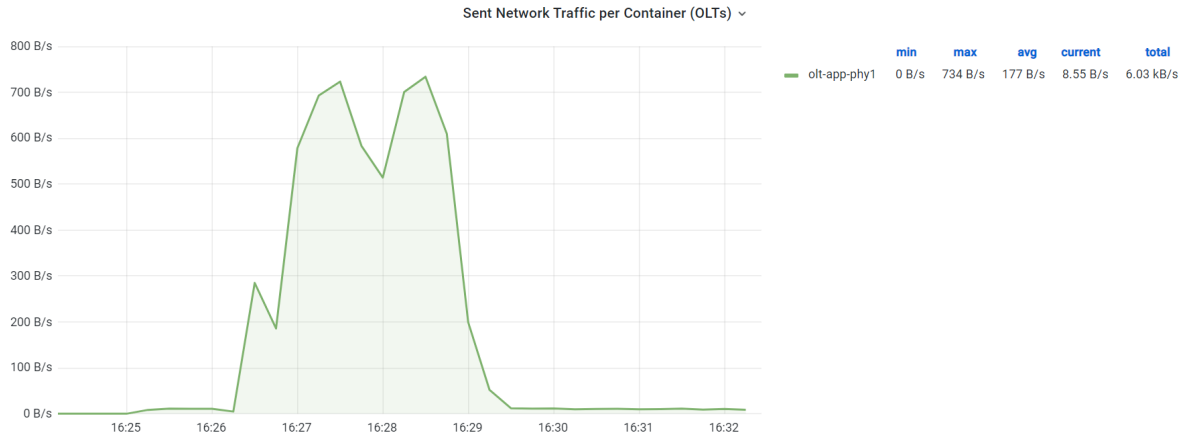


Figure 65: Scenario 3 - Network sent traffic metric for OLT-App into physical OLT and traffic in Control Relay mode.

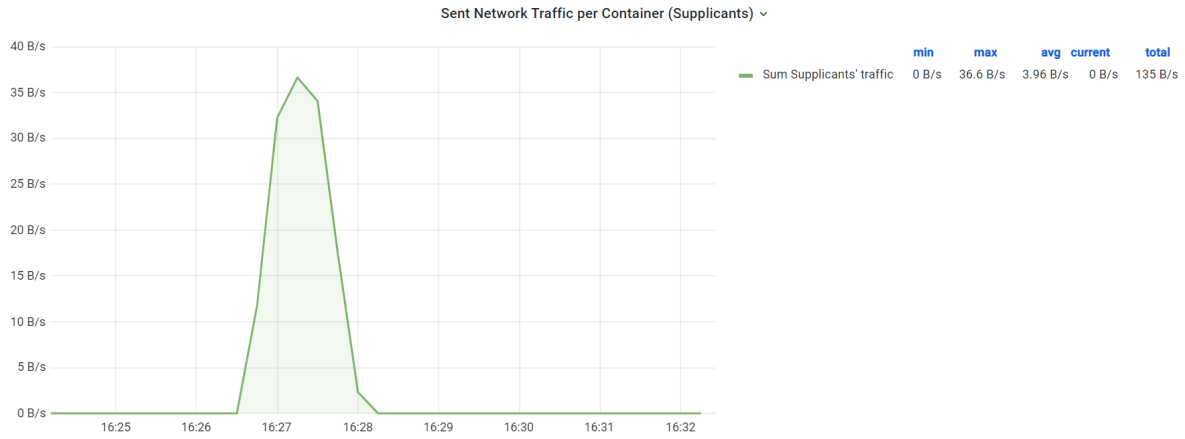


Figure 66: Scenario 3 - Network sent traffic metric for Supplicants hosted into K8S and traffic in Control Relay mode.

Scenario 4: Physical OLT / Kubernetes / Straight

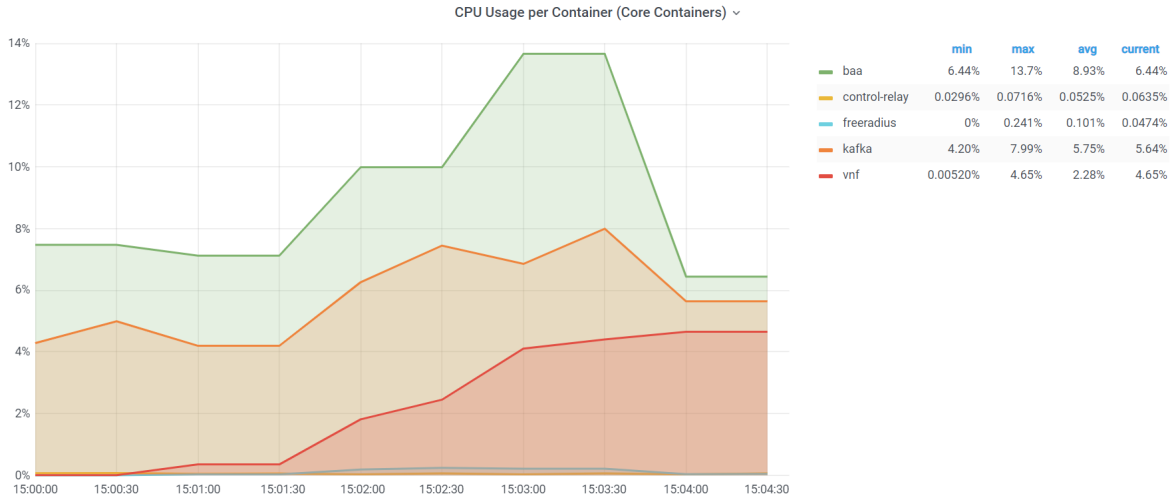


Figure 67: Scenario 4 - CPU metric for Core elements hosted into K8S and traffic in Straight mode.

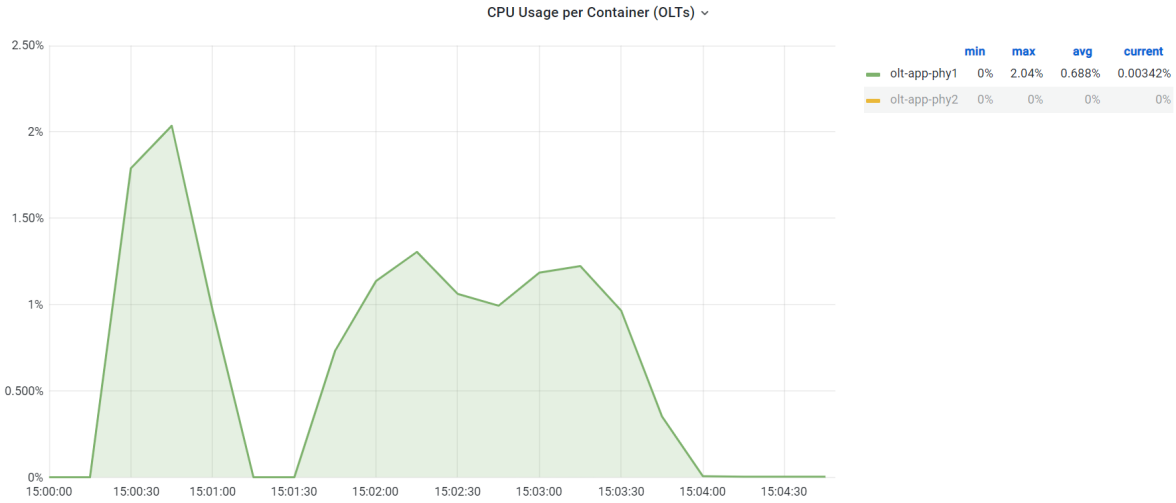


Figure 68: Scenario 4 - CPU metric for OLT-App into physical OLT and traffic in Straight mode.

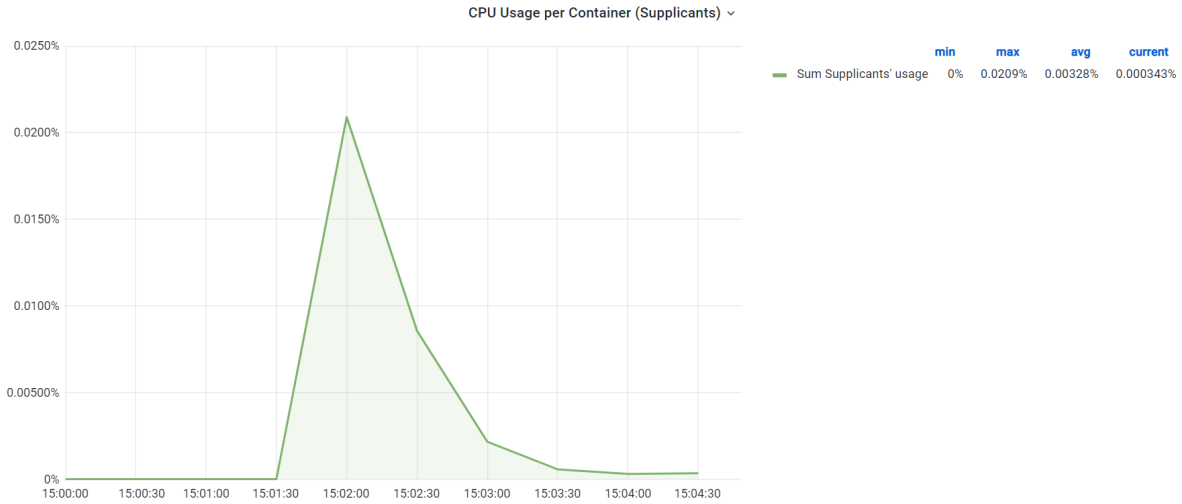


Figure 69: Scenario 4 - CPU metric for Supplicants hosted into K8S and traffic in Straight mode.

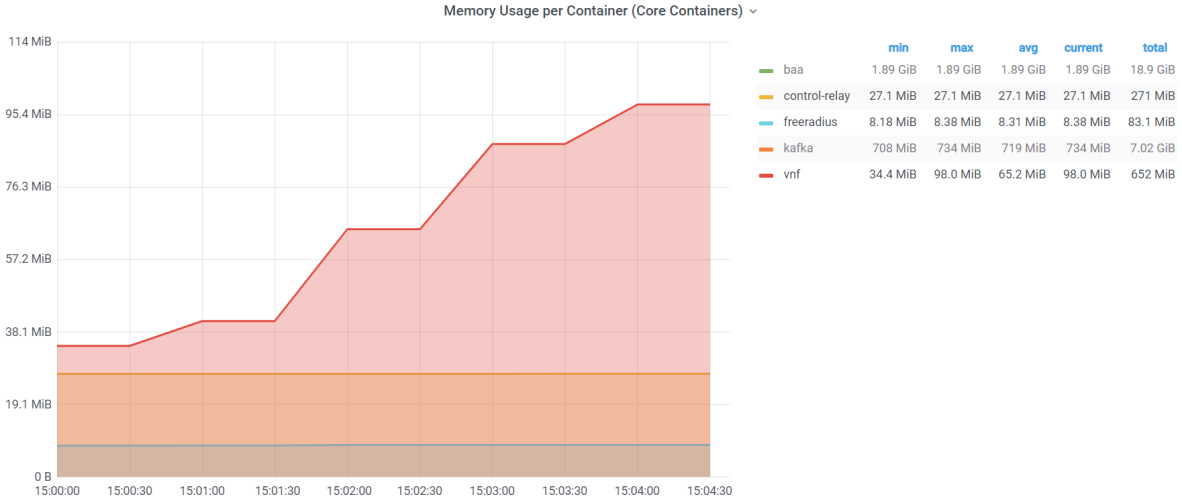


Figure 70: Scenario 4 - Memory metric for Core elements hosted into K8S and traffic in Straight mode.

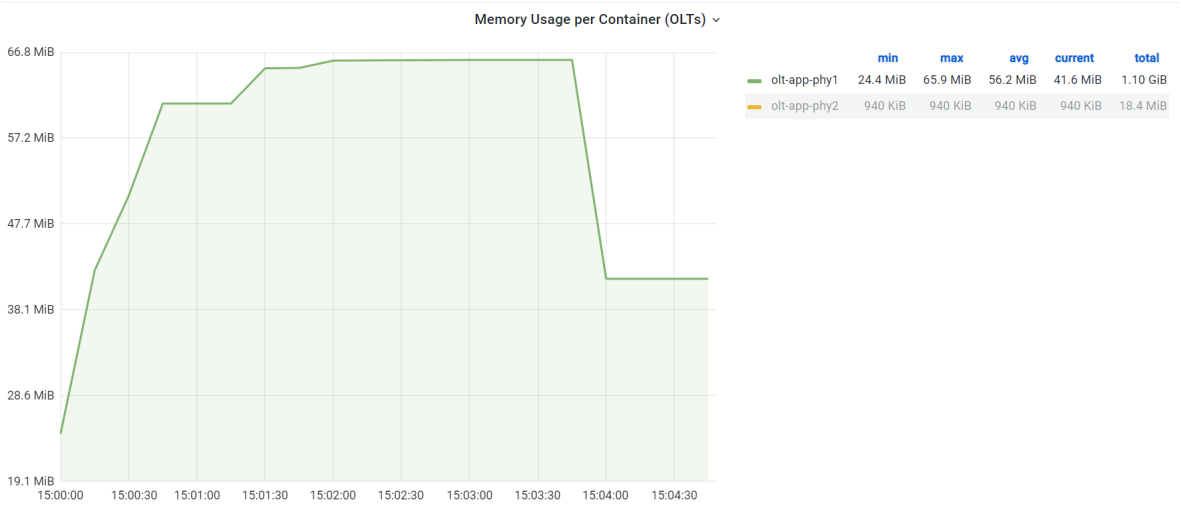


Figure 71: Scenario 4 - Memory metric for OLT-App into physical OLT and traffic in Straight mode.

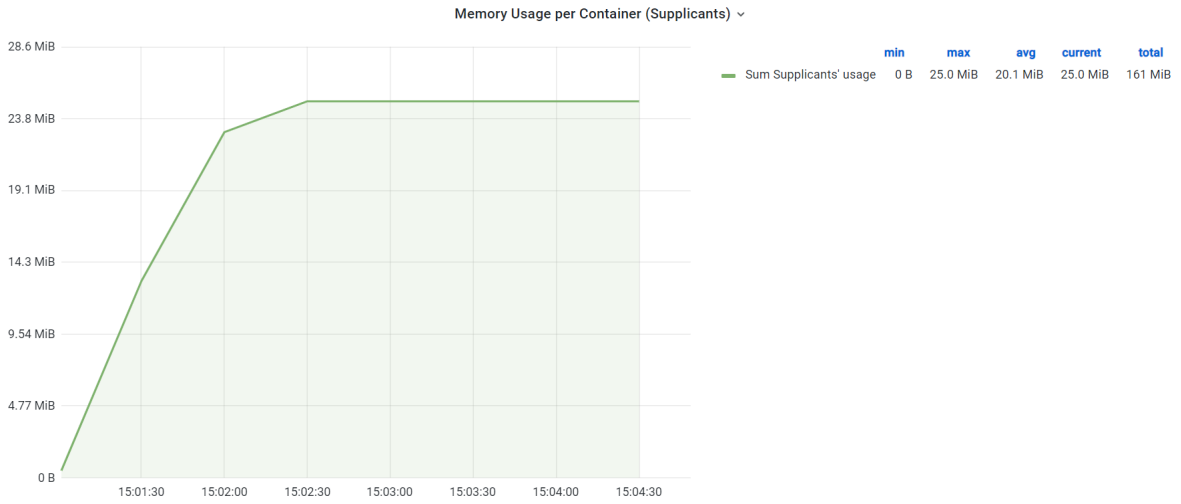


Figure 72: Scenario 4 - Memory metric for Supplicants hosted into K8S and traffic in Straight mode.

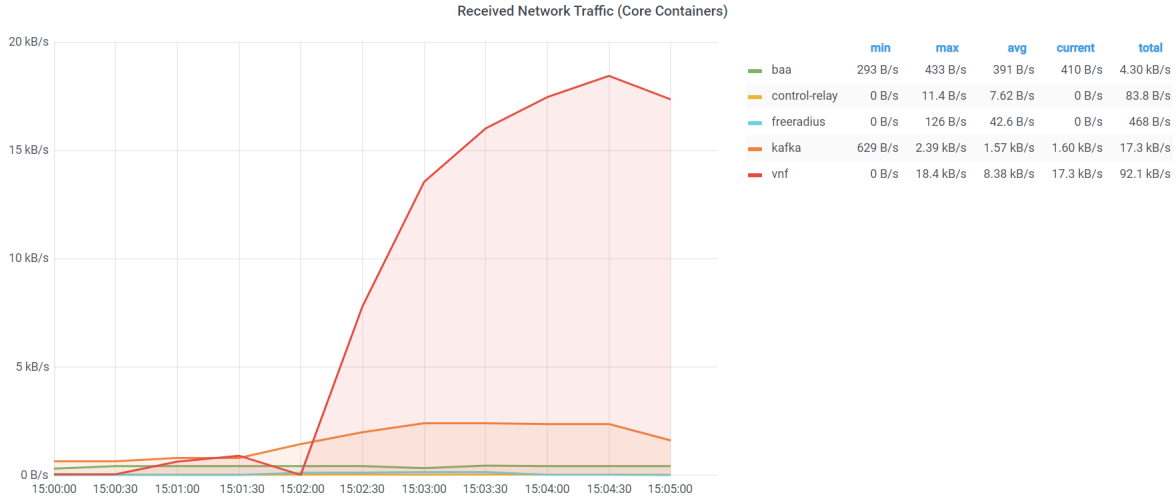


Figure 73: Scenario 4 - Network received traffic for Core elements hosted into K8S and traffic in Straight mode.

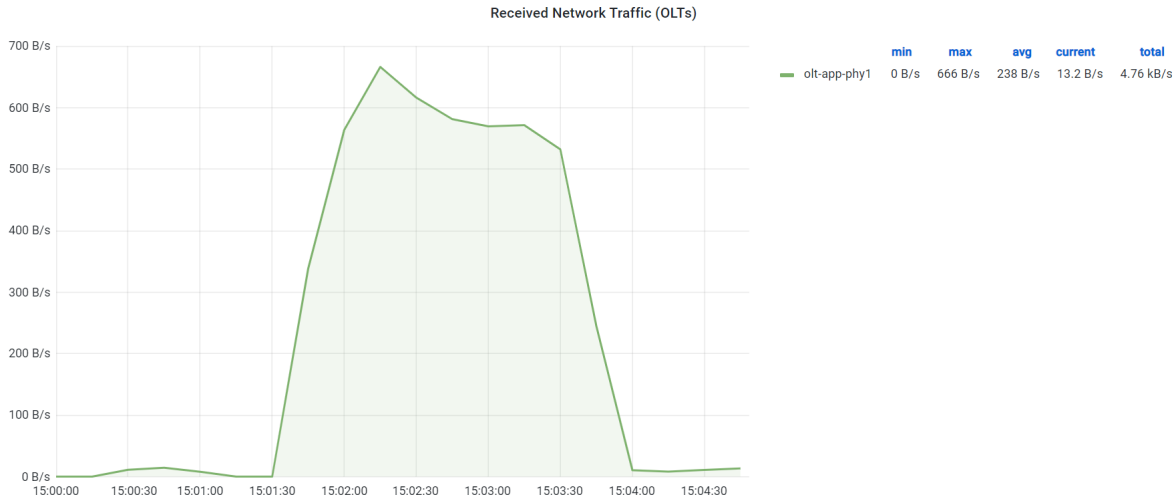


Figure 74: Scenario 4 - Network received traffic for OLT-App into physical OLT and traffic in Straight mode.

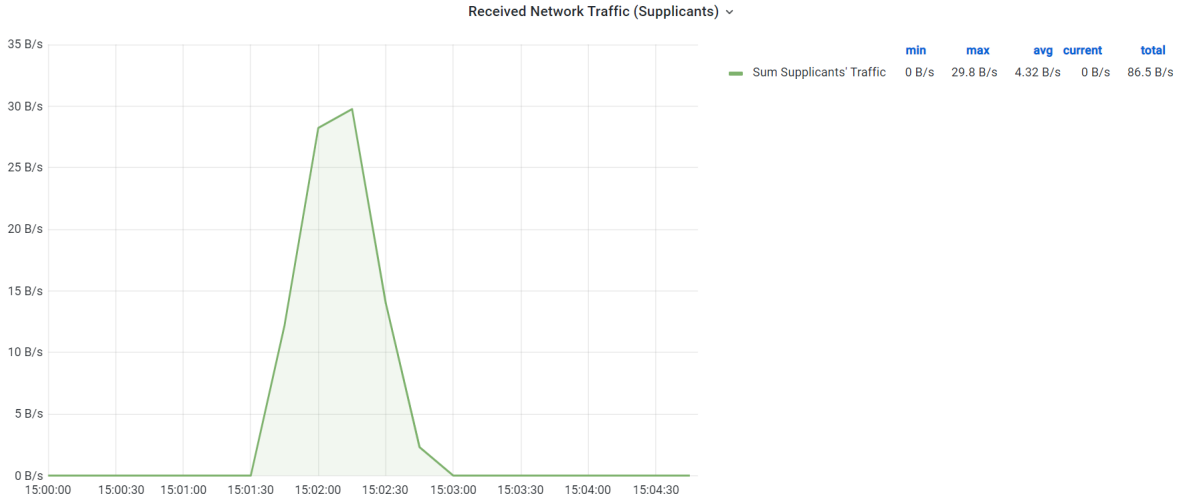


Figure 75: Scenario 4 - Network received traffic metric for Supplicants hosted into K8S and traffic in Straight mode.

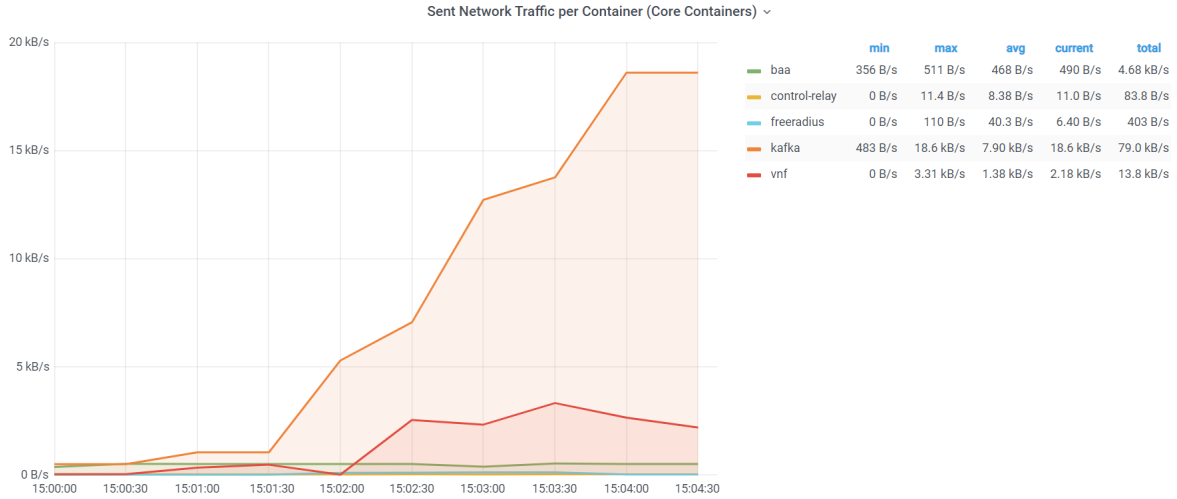


Figure 76: Scenario 4 - Network sent traffic metric for Core elements hosted into K8S and traffic in Straight mode.

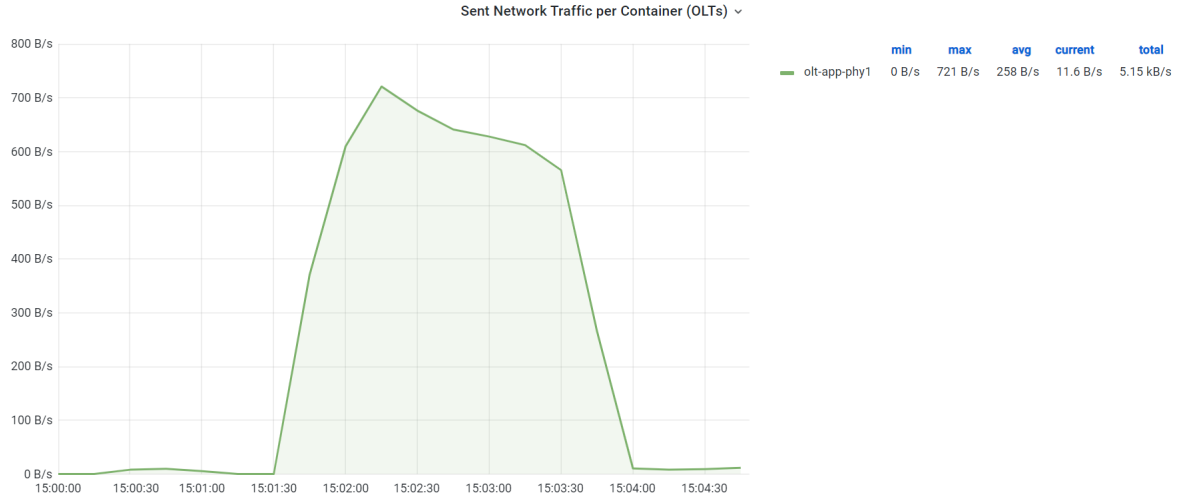


Figure 77: Scenario 4 - Network sent traffic metric for OLT-App into physical OLT and traffic in Straight mode.

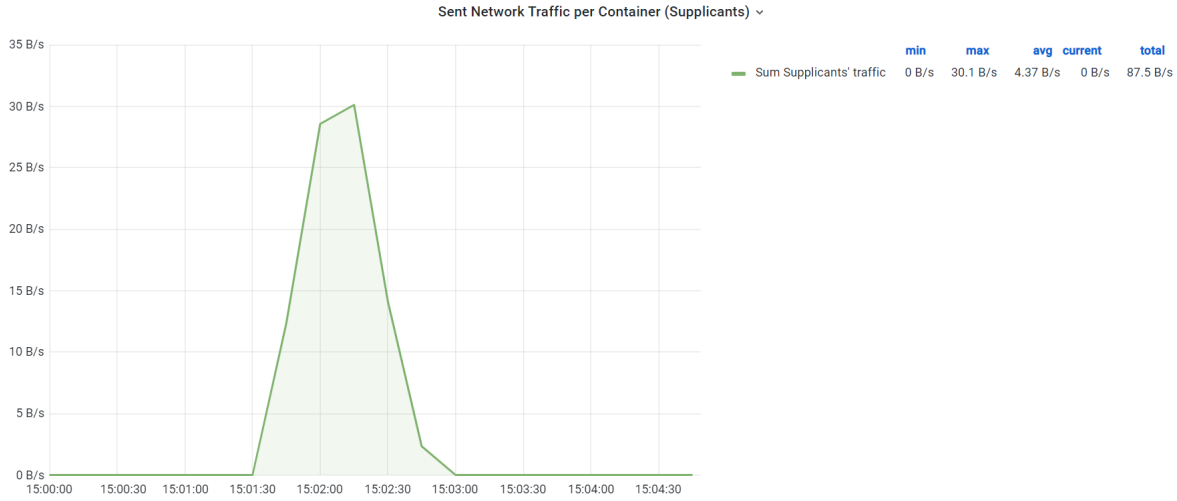


Figure 78: Scenario 4 - Network sent traffic metric for Supplicants hosted into K8S and traffic in Straight mode.

Scenario 5: Virtual Machine / Docker / Control Relay

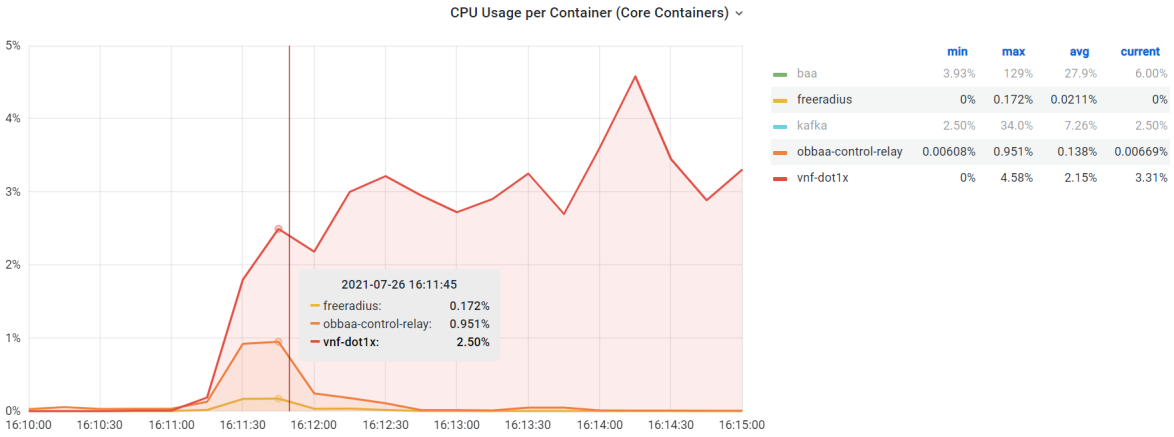


Figure 79: Scenario 5 - CPU metric for Core elements hosted into Docker and traffic in Control Relay mode.

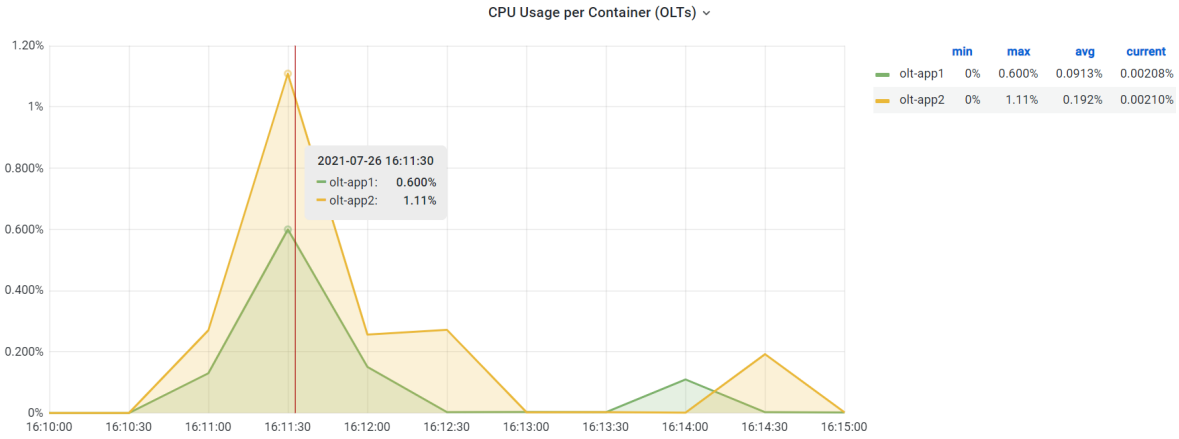


Figure 80: Scenario 5 - CPU metric for OLT-App into physical OLT and traffic in Control Relay mode.

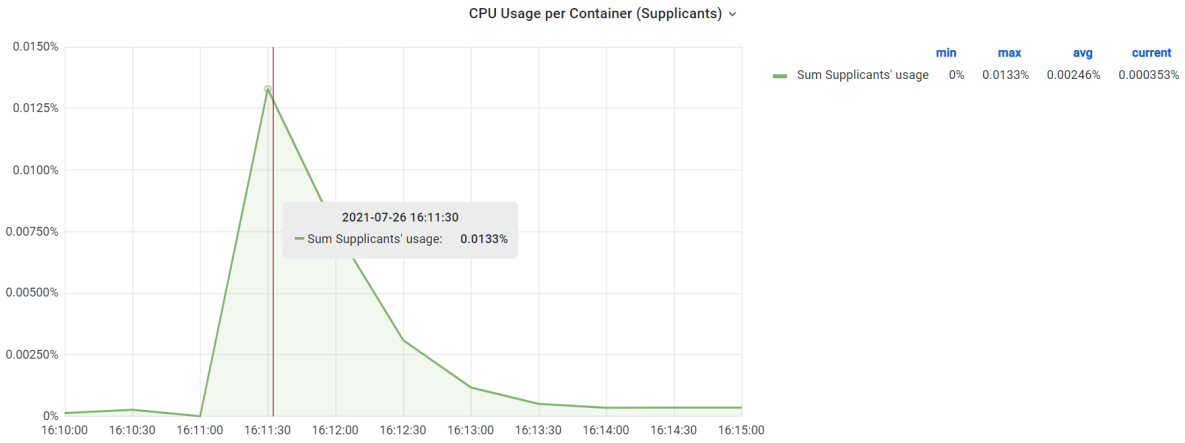


Figure 81: Scenario 5 - CPU metric for Supplicants hosted into Docker and traffic in Control Relay mode.

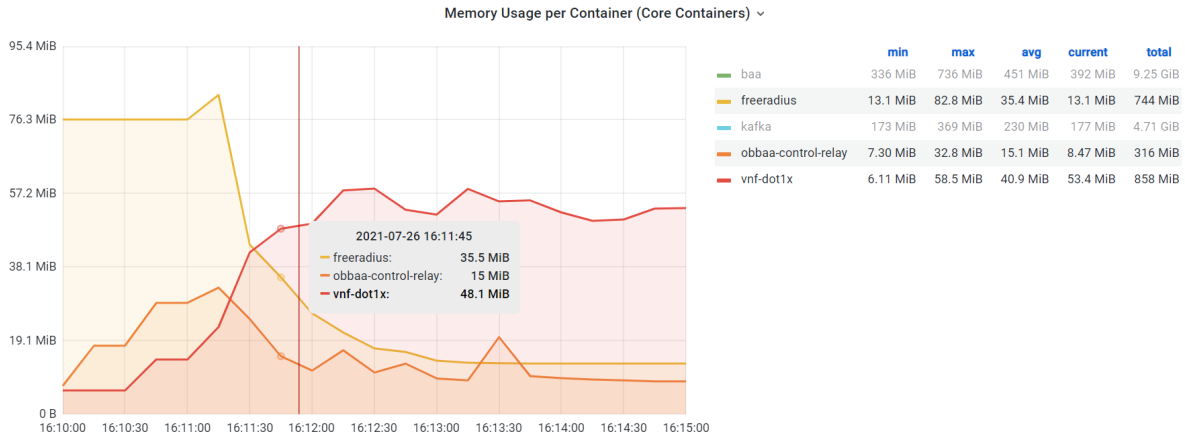


Figure 82: Scenario 5 - Memory metric for Core elements hosted into Docker and traffic in Control Relay mode.

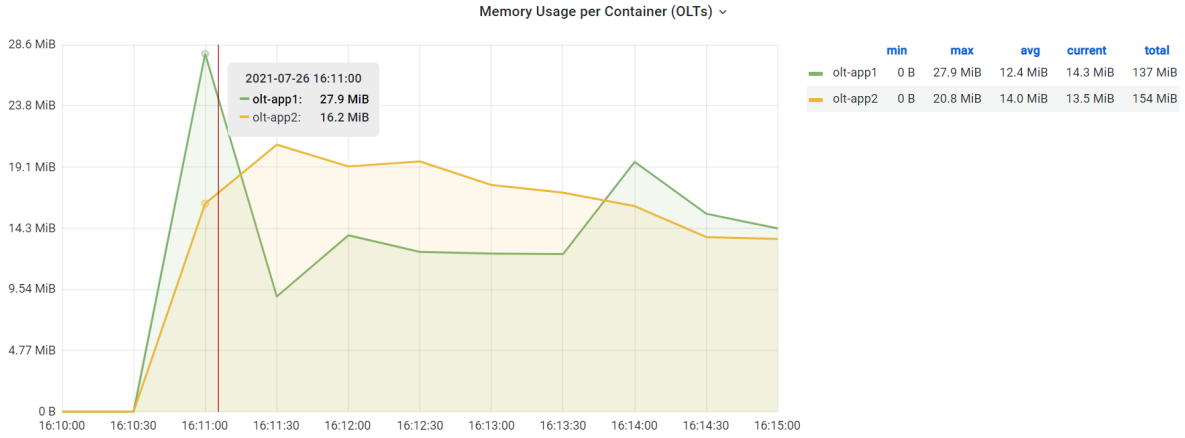


Figure 83: Scenario 5 - Memory metric for OLT-App into physical OLT and traffic in Control Relay mode.

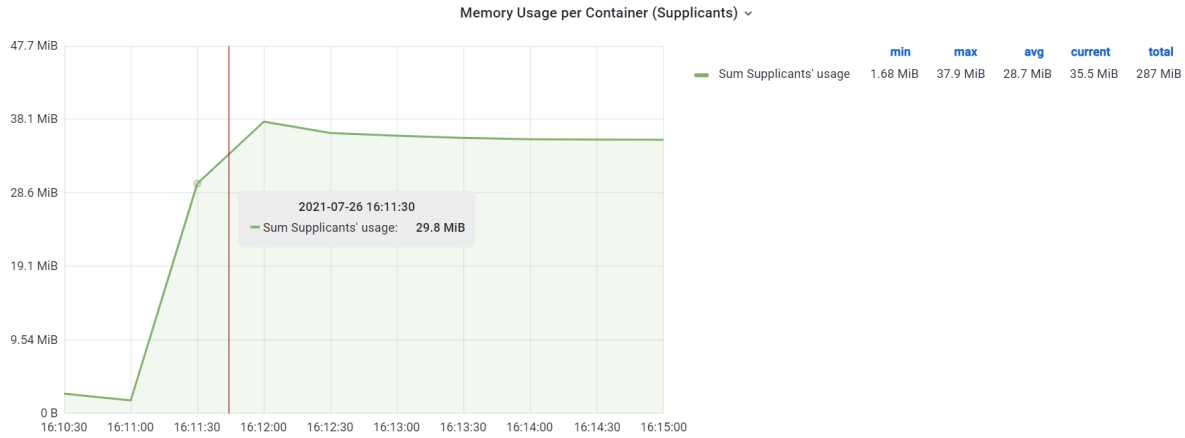


Figure 84: Scenario 5 - Memory metric for Supplicants hosted into Docker and traffic in Control Relay mode.

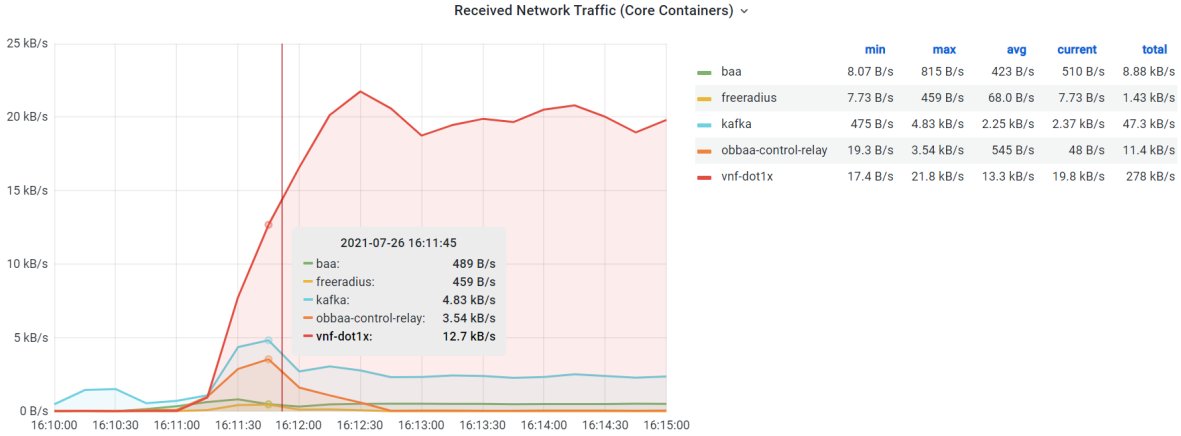


Figure 85: Scenario 5 - Network received traffic for Core elements hosted into Docker and traffic in Control Relay mode.

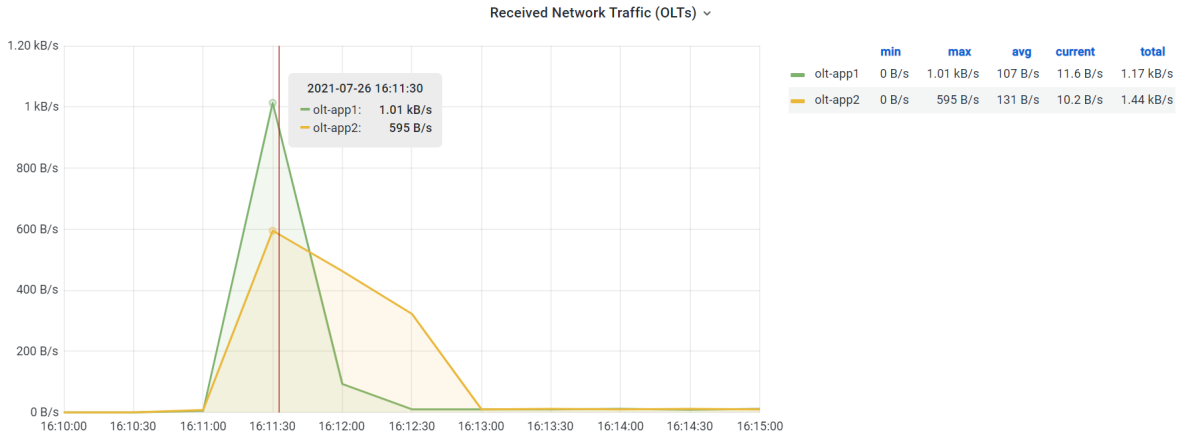


Figure 86: Scenario 5 - Network received traffic for OLT-App into physical OLT and traffic in Control Relay mode.

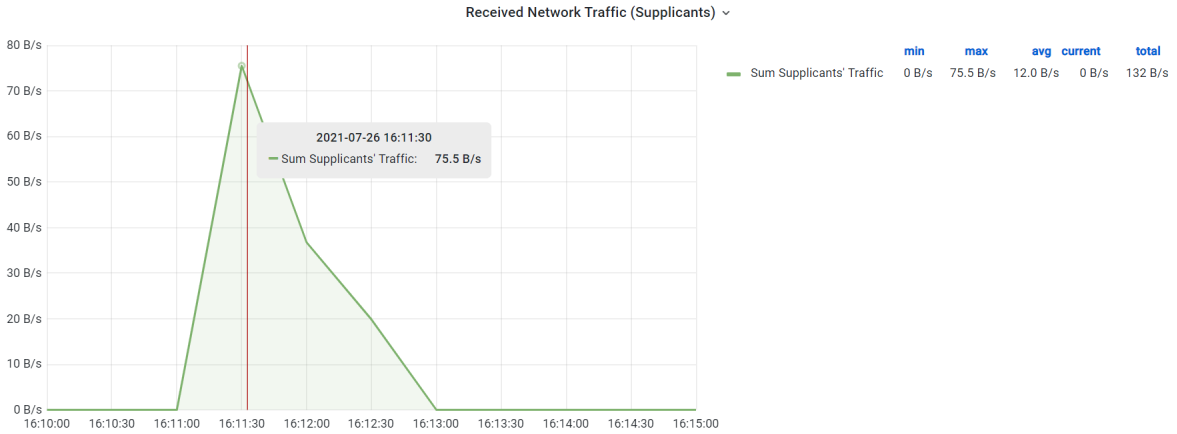


Figure 87: Scenario 5 - Network received traffic metric for Supplicants hosted into Docker and traffic in Control Relay mode-

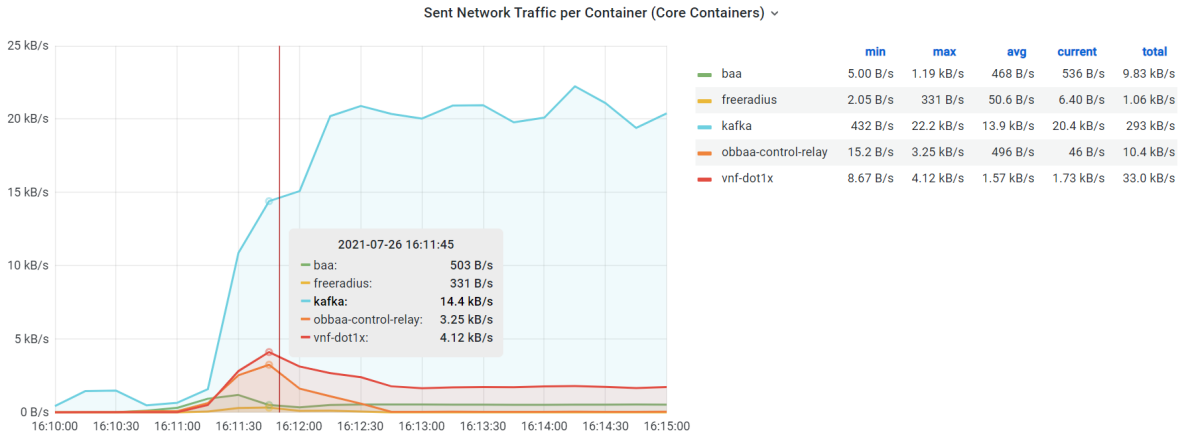


Figure 88: Scenario 5 - Network sent traffic metric for Core elements hosted into Docker and traffic in Control Relay mode.

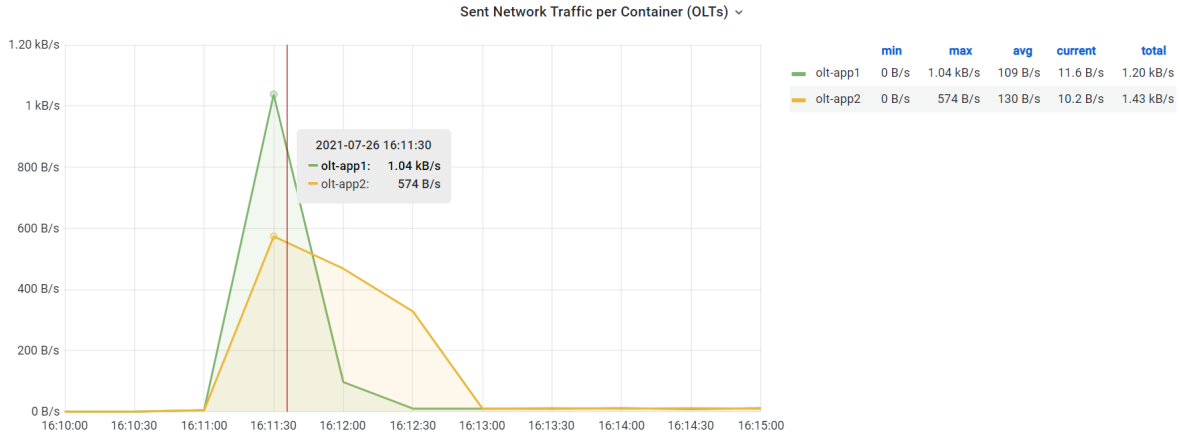


Figure 89: Scenario 5 - Network sent traffic metric for OLT-App into physical OLT and traffic in Control Relay mode.

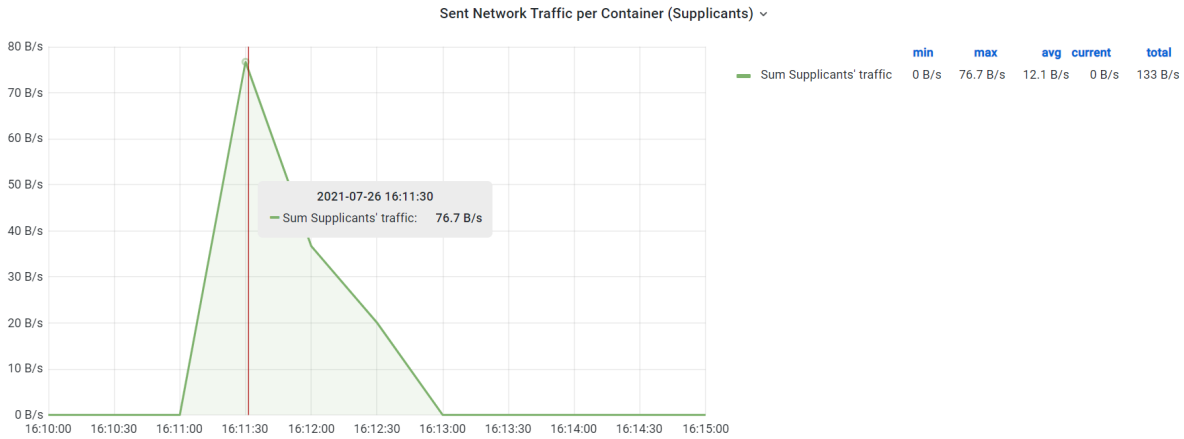


Figure 90: Scenario 5 - Network sent traffic metric for Supplicants hosted into Docker and traffic in Control Relay mode.

Scenario 6: Virtual Machine / Docker / Straight

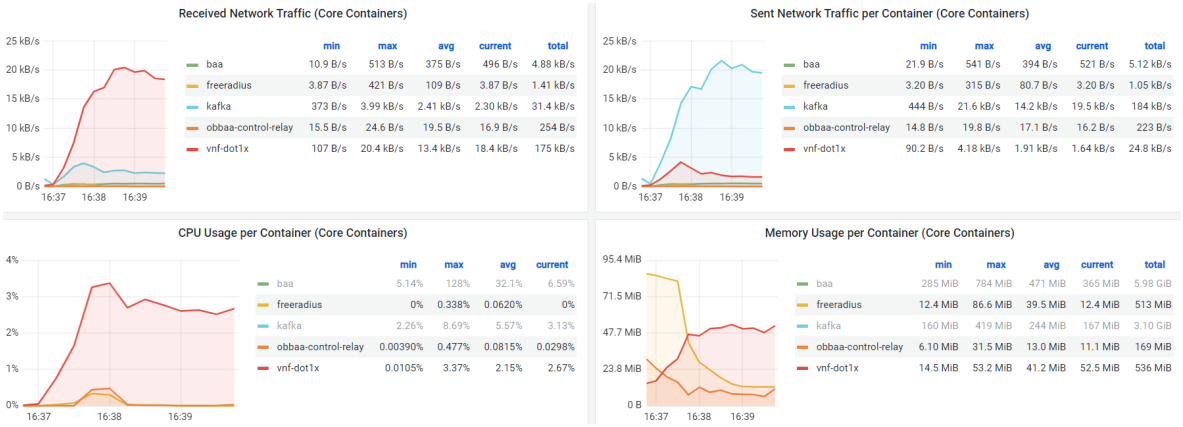


Figure 91: Scenario 6 - Consolidated metrics for Core elements hosted into Docker and traffic in Straight mode.

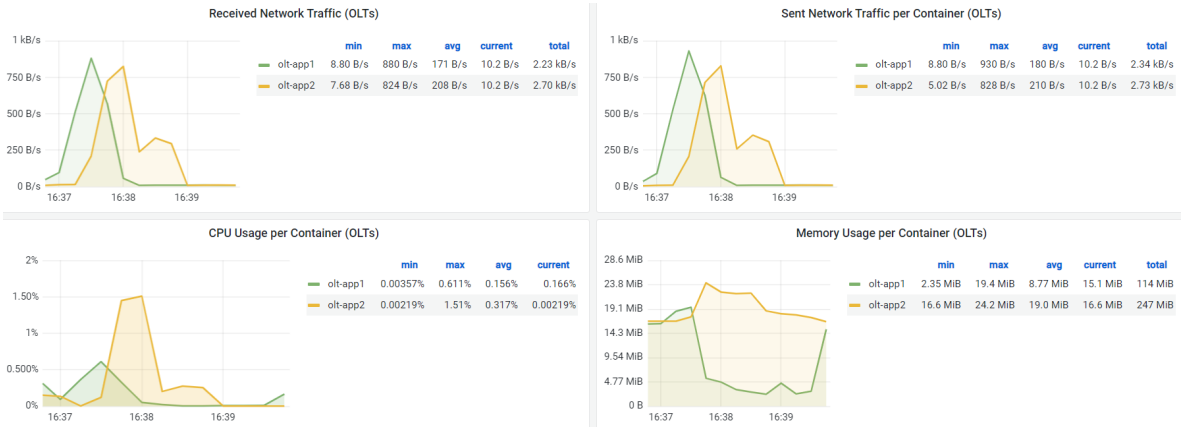


Figure 92: Scenario 6 - Consolidated metrics for OLT-App into physical OLT and traffic in Straight mode.

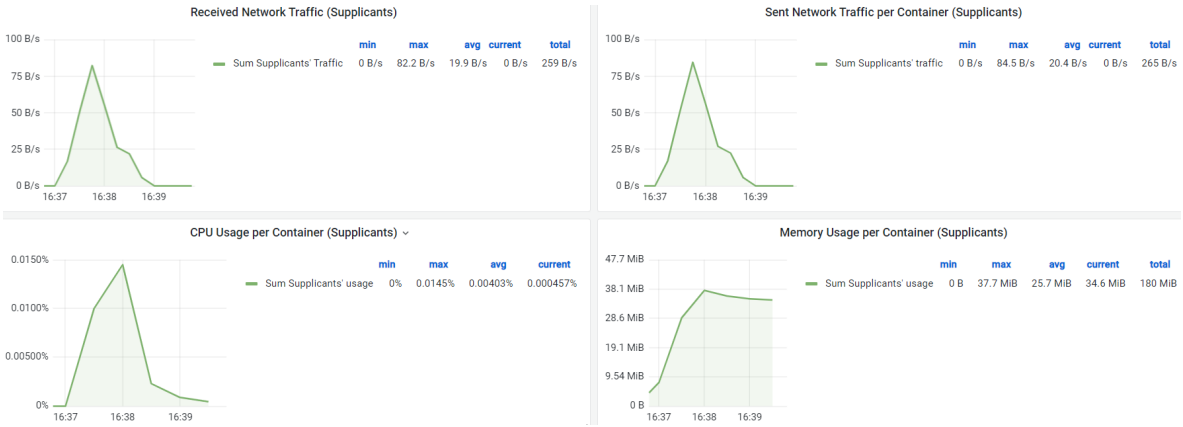


Figure 93: Scenario 6 - Consolidated metrics for Supplicants hosted into Docker and traffic in Straight mode.

Scenario 7: Virtual Machine / Kubernetes / Control Relay

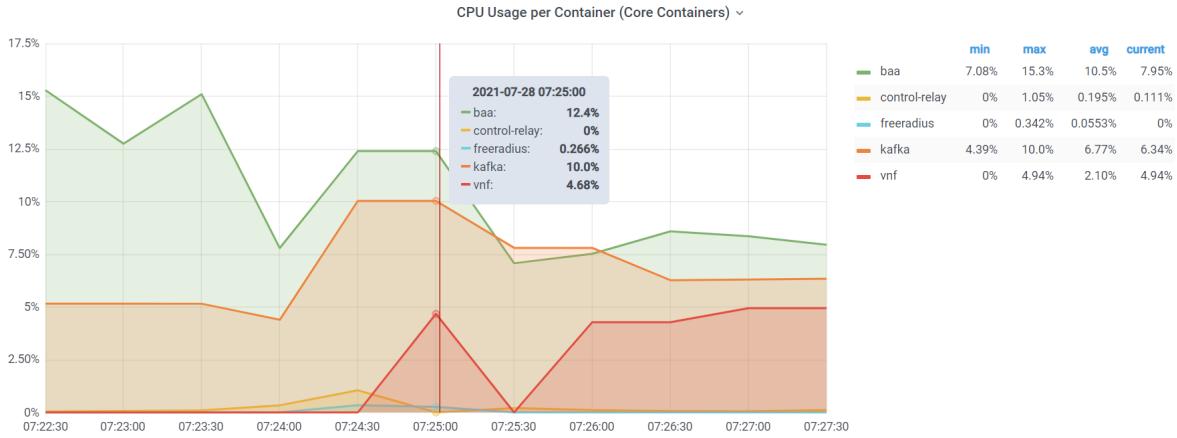


Figure 94: Scenario 7 - CPU metric for Core elements hosted into K8S and traffic in Control Relay mode.

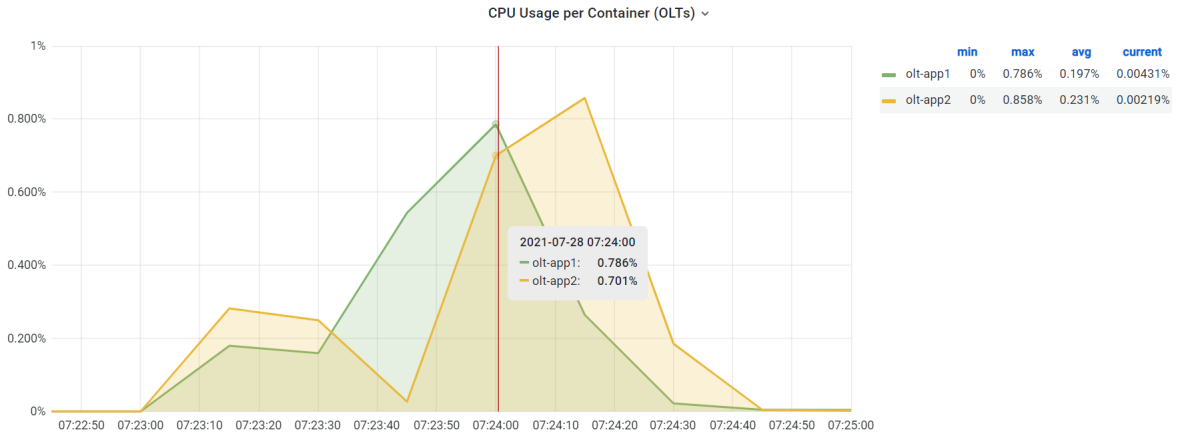


Figure 95: Scenario 7 - CPU metric for OLT-App into physical OLT and traffic in Control Relay mode.

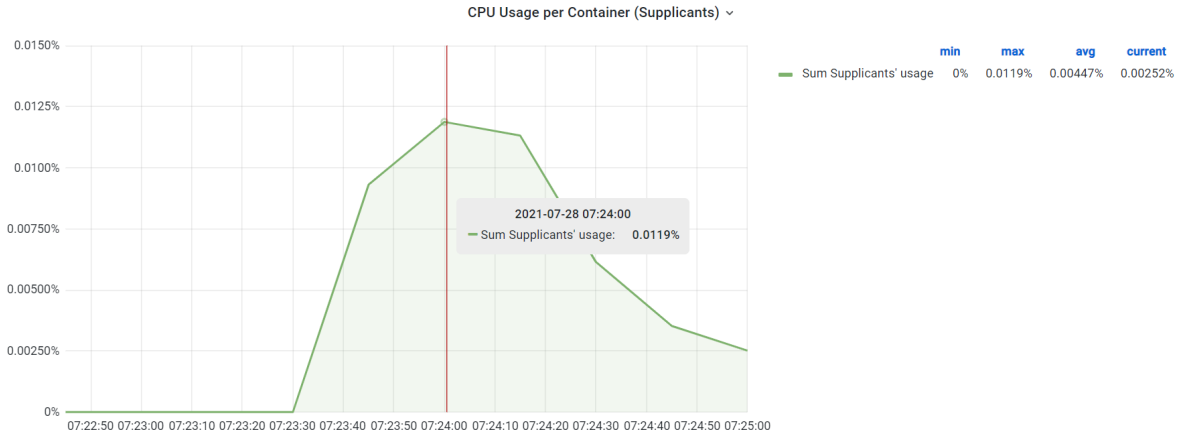


Figure 96: Scenario 7 - CPU metric for Supplicants hosted into K8S and traffic in Control Relay mode.

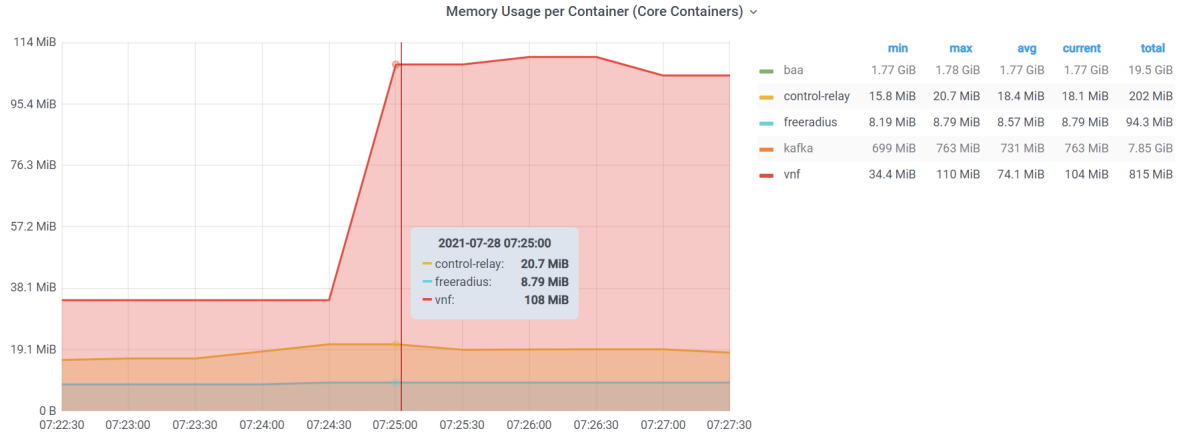


Figure 97: Scenario 7 - Memory metric for Core elements hosted into K8S and traffic in Control Relay mode.

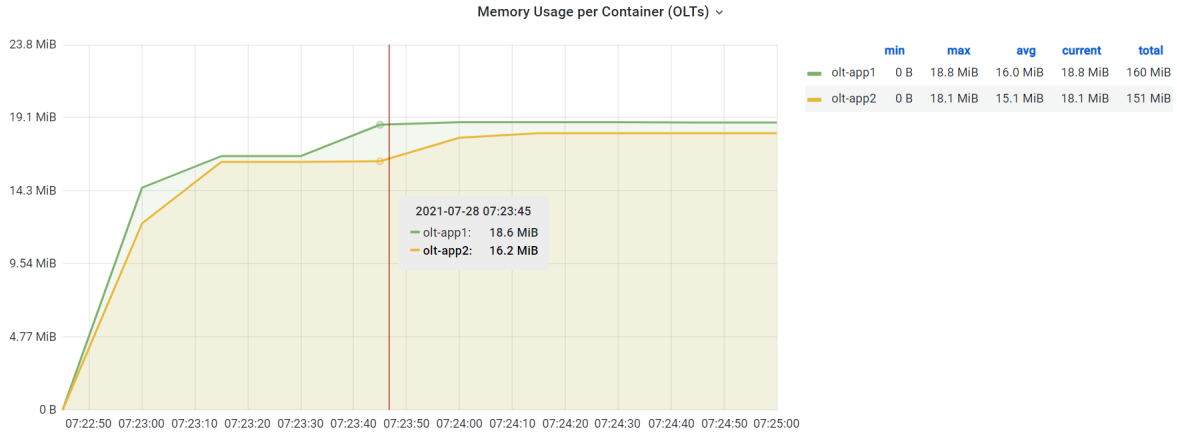


Figure 98: Scenario 7 - Memory metric for OLT-App into physical OLT and traffic in Control Relay mode.

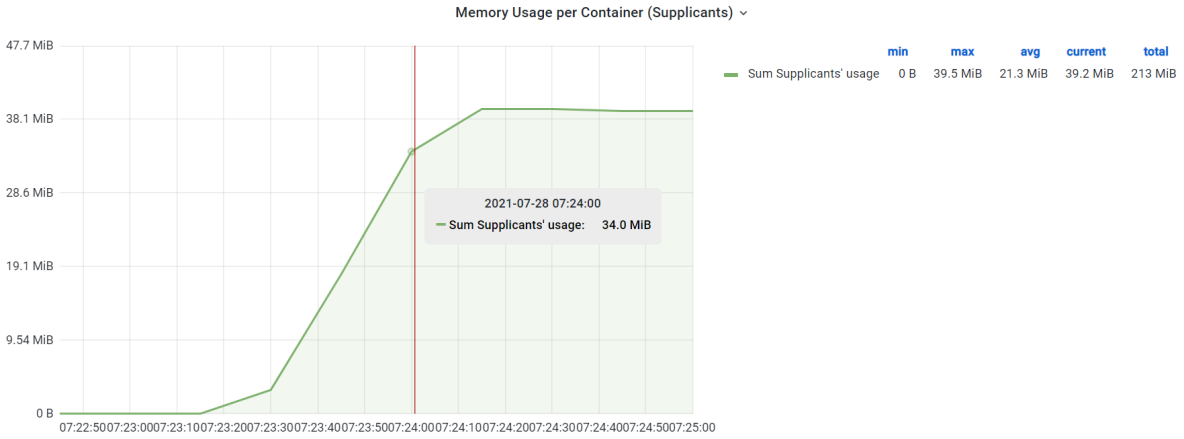


Figure 99: Scenario 7 - Memory metric for Supplicants hosted into K8S and traffic in Control Relay mode.

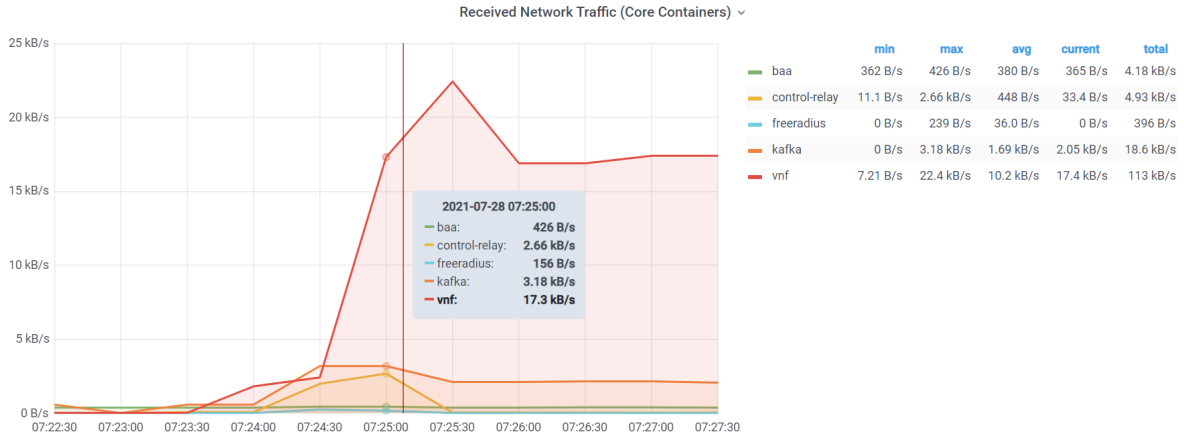


Figure 100: Scenario 7 - Network received traffic for Core elements hosted into K8S and traffic in Control Relay mode.

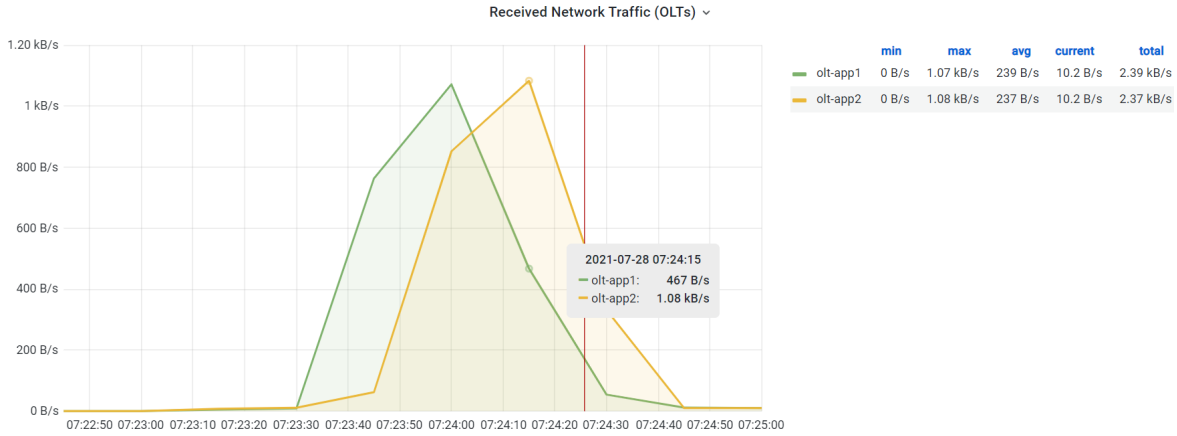


Figure 101: Scenario 7 - Network received traffic for OLT-App into physical OLT and traffic in Control Relay mode.

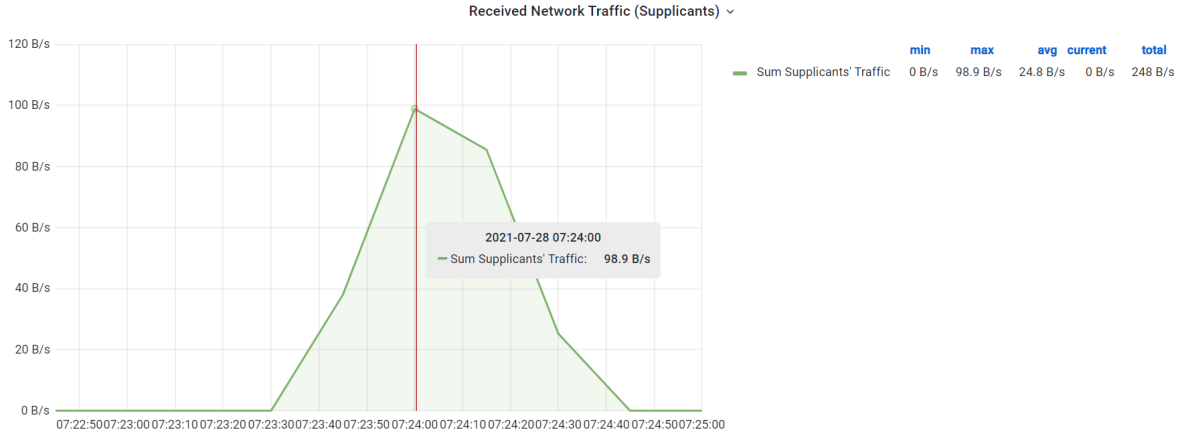


Figure 102: Scenario 7 - Network received traffic metric for Supplicants hosted into K8S and traffic in Control Relay mode.

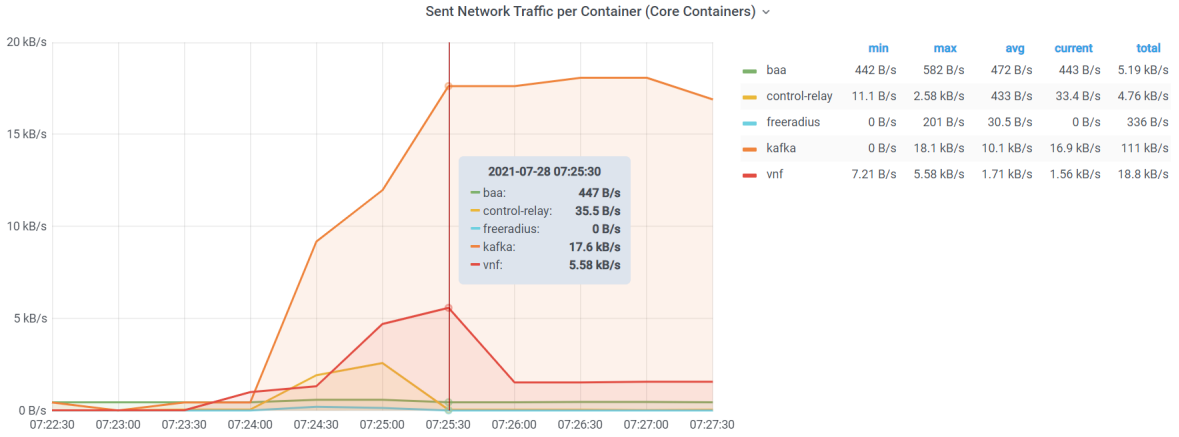


Figure 103: Scenario 7 - Network sent traffic metric for Core elements hosted into K8S and traffic in Control Relay mode.

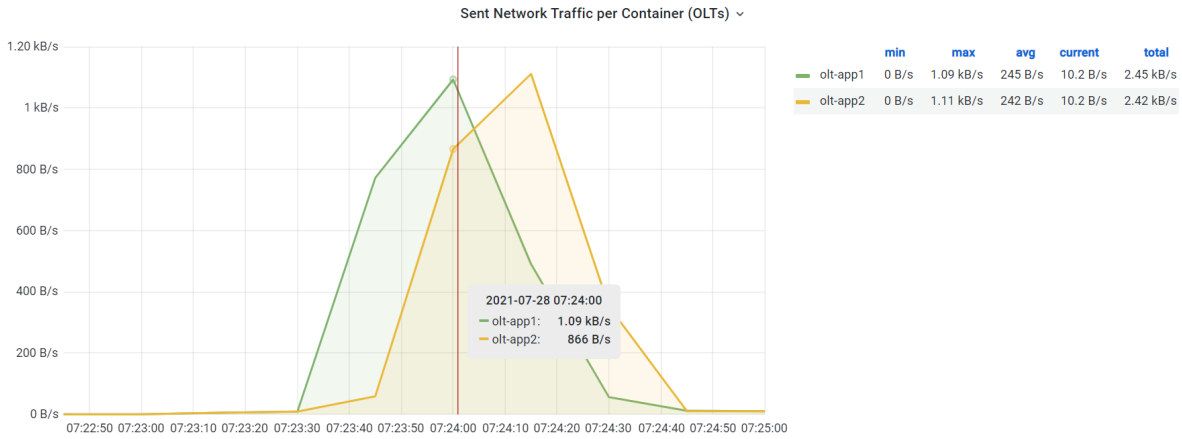


Figure 104: Scenario 7 - Network sent traffic metric for OLT-App into physical OLT and traffic in Control Relay mode.

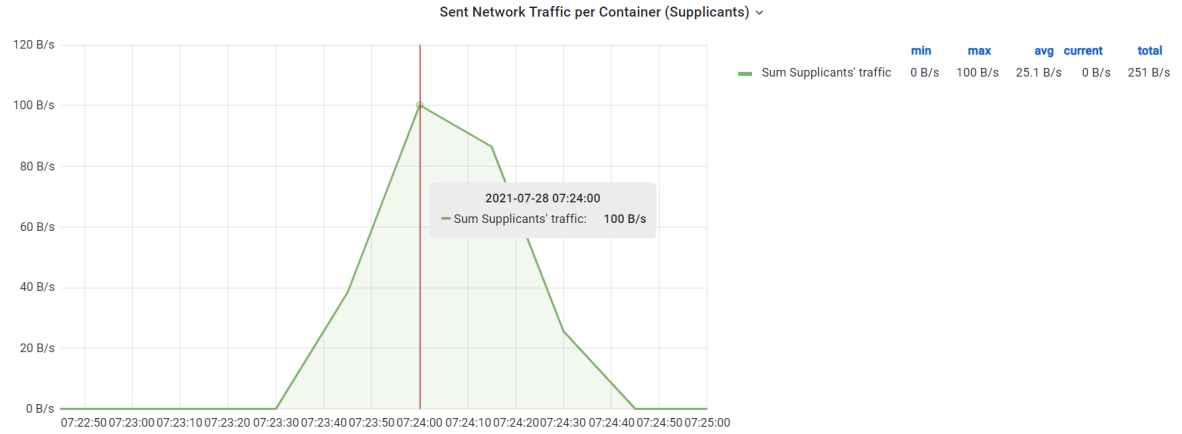


Figure 105: Scenario 7 - Network sent traffic metric for Supplicants hosted into K8S and traffic in Control Relay mode.

Scenario 8: Virtual Machine / Kubernetes / Straight

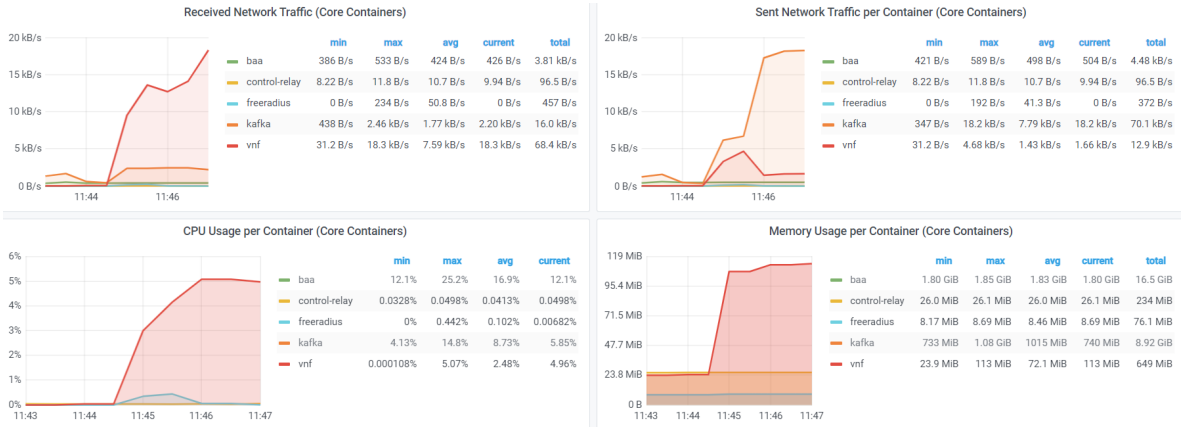


Figure 106: Scenario 8 - Consolidated metrics for Core elements hosted into K8S and traffic in Straight mode.

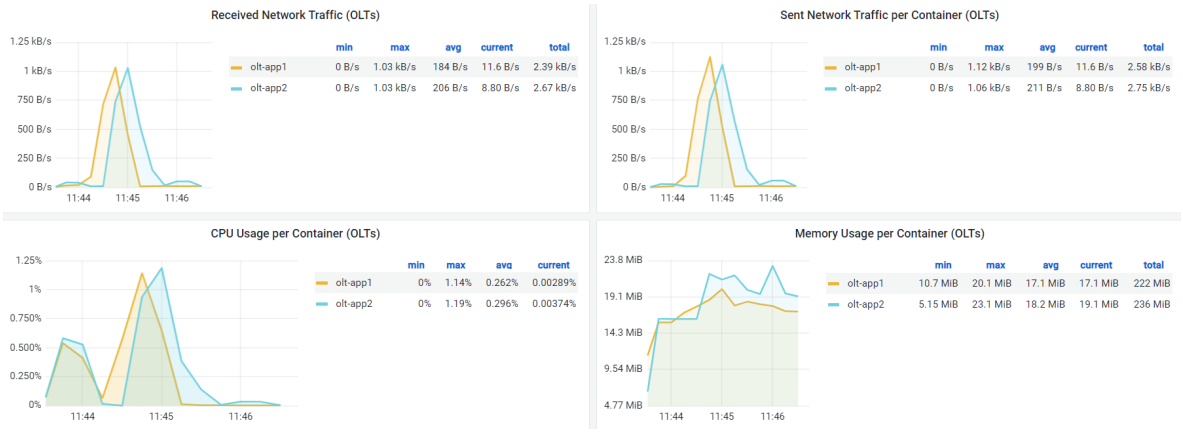


Figure 107: Scenario 8 - Consolidated metrics for OLT-App into physical OLT and traffic in Straight mode.

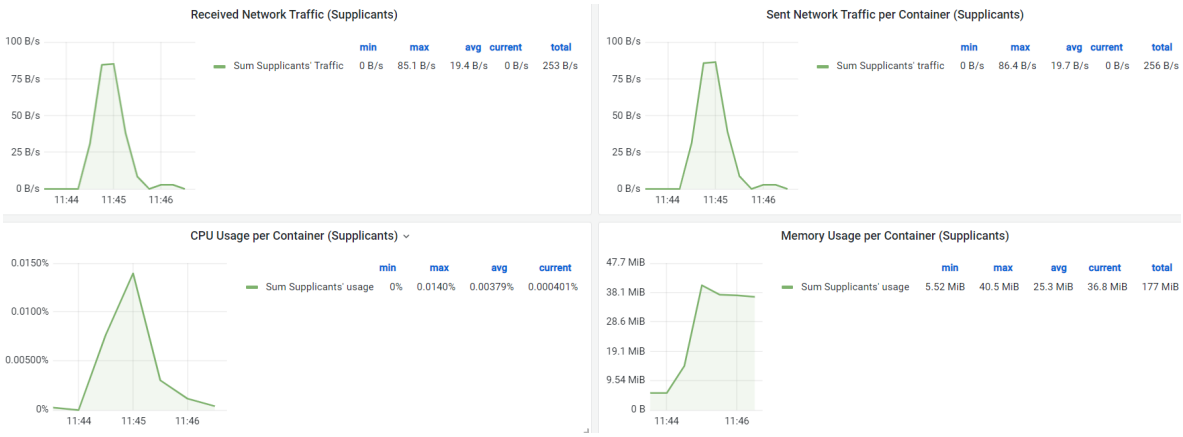


Figure 108: Scenario 8 - Consolidated metrics for Supplicants hosted into K8S and traffic in Straight mode.

BIBLIOGRAPHY

[1] Kamran Ali Memon et al. "A Bibliometric Analysis and Visualization of Passive Optical Network Research in the Last Decade". In: *Optical Switching and Networking* 39 (2020), p. 100586. ISSN: 1573-4277. DOI: https://doi.org/10.1016/j.osn.2020.100586. URL: http://www.sciencedirect.com/science/article/pii/S1573427720300229.

[2] Rui Calé Martins, André Brízido, and Miguel Santos. "InnovAction 2018 Altice - The digital transformation of the central office". In: (2018), pp. 159–171.

[3] Cisco. "Cisco Annual Internet Report (2018–2023)". In: 2020.3 (Mar. 2020), p. 4. ISSN: 13613723. DOI: 10.1016/S1361-3723(20)30026-9. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf.

[4] Sedef Demirci and Seref Sagiroglu. "Optimal placement of virtual network functions in software defined networks: A survey". In: *Journal of Network and Computer Applications* 147 (2019), p. 102424. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.102424. URL: https://www.sciencedirect.com/science/article/pii/S1084804519302760.

[5] Larry Peterson et al. "Democratizing the Network Edge". In: *SIGCOMM Comput. Commun. Rev.* 49.2 (2019), pp. 31–36. ISSN: 0146-4833. DOI: 10.1145/3336937.3336942. URL: https://doi.org/10.1145/3336937.3336942.

[6] Bo Yi et al. "A comprehensive survey of Network Function Virtualization". In: *Computer Networks* 133 (2018), pp. 212–262. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.01.021. URL: http://www.sciencedirect.com/science/article/pii/S1389128618300306.

[7] P L Ventre et al. "SDN Architecture and Southbound APIs for IPv6 Segment Routing Enabled Wide Area Networks". In: *IEEE Transactions on Network and Service Management* 15.4 (2018), pp. 1378–1392. DOI: 10.1109/TNSM.2018.2876251.

[8] Barbara A. Kitchenham, Tore Dyba, and Magne Jorgensen. "Evidence-based software engineering". In: *Proceedings - International Conference on Software Engineering* 26 (2004), pp. 273–281. ISSN: 02705257. DOI: 10.1109/icse.2004.1317449.

[9] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. Durham: University of Durham, 2007. DOI: 10.1145/1134285.1134500. arXiv: 1304.1186.

[10] Sofiane Amara et al. "Group formation in mobile computer supported collaborative learning contexts: A systematic literature review". In: *Educational Technology and Society* 19.2 (2015), pp. 258–273. ISSN: 14364522.

[11] Michel S Bonfim, Kelvin L Dias, and Stenio F L Fernandes. "Integrated NFV/SDN Architectures: A Systematic Literature Review". In: *ACM Comput. Surv.* 51.6 (2019). ISSN: 0360-0300. DOI: 10.1145/3172866. URL: https://doi.org/10.1145/3172866.

[12] A S Thyagaturu et al. "Software Defined Optical Networks (SDONs): A Comprehensive Survey". In: *IEEE Communications Surveys Tutorials* 18.4 (2016), pp. 2738–2786. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2586999.

[13] Shihabur Rahman Chowdhury et al. "Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges". In: *IEEE Network* 33.3 (May 2019), pp. 168–176. ISSN: 0890-8044. DOI: 10.1109/MNET.2019.1800082. URL: https://ieeexplore.ieee.org/document/8688711/.

[14] Zohaib Latif et al. "A comprehensive survey of interface protocols for software defined networks". In: *Journal of Network and Computer Applications* 156 (2020), p. 102563. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102563. URL: http://www.sciencedirect.com/science/article/pii/S1084804520300370.

[15] Hamzeh Khalili et al. "A proposal for an SDN-based SIEPON architecture". In: *Optics Communications* 403 (2017), pp. 9–21. ISSN: 0030-4018. DOI: https://doi.org/10.1016/j.optcom.2017.07.009. URL: http://www.sciencedirect.com/science/article/pii/S0030401817305898.

[16] Rashid Mijumbi et al. "Network Function Virtualization: State-of-the-Art and Research Challenges". In: *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS* 18.1 (2016), pp. 236–262. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2477041.

[17] Huda Saleh Abbas and Mark A. Gregory. *The next generation of passive optical networks: A review*. May 2016. DOI: 10.1016/j.jnca.2016.02.015.

[18] R Bruschi et al. "Evaluating the Impact of Micro-Data Center (μDC) Placement in an Urban Environment". In: *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2018, pp. 1–7. DOI: 10.1109/NFV-SDN.2018.8725627.

[19] Meet Kumari, Reecha Sharma, and Anu Sheetal. "Passive Optical Network Evolution to Next Generation Passive Optical Network: A Review". In: *2018 6th Edition of International Conference on Wireless Networks Embedded Systems (WECON)*. Institute of Electrical and Electronics Engineers Inc., Nov. 2018, pp. 102–107. ISBN: 9781538670507. DOI: 10.1109/WECON.2018.8782066.

[20] Ivica Cale, Aida Salihovic, and Matija Ivekovic. "Gigabit passive optical network - GPON". In: *Proceedings of the International Conference on Information Technology Interfaces, ITI* (2007), pp. 679–684. ISSN: 13301012. DOI: 10.1109/ITI.2007.4283853.

[21] Tommaso Muciaccia, Fabio Gargano, and Vittorio M.N. Passaro. *Passive optical access networks: State of the art and future evolution*. Dec. 2014. DOI: 10.3390/photonics1040323.

[22] Paul W Shumate. "Fiber-to-the-Home: 1977–2007". In: *Journal of Lightwave Technology* 26.9 (2008), pp. 1093–1103.

[23] Frank Effenberger and Tarek S. El-Bawab. "Passive Optical Networks (PONs): Past, present, and future". In: *Optical Switching and Networking* 6.3 (July 2009), pp. 143–150. ISSN: 15734277. DOI: 10.1016/j.osn.2009.02.001.

[24] Germán V Arévalo, Roberto C Hincapié, and Roberto Gaudino. "Optimization of multiple PON deployment costs and comparison between GPON, XGPON, NGPON2 and UDWDM PON". In: *Optical Switching and Networking* 25 (2017), pp. 80–90. ISSN: 1573-4277. DOI: https://doi.org/10.1016/j.osn.2017.03.003. URL: http://www.sciencedirect.com/science/article/pii/S1573427716300522.

[25] Glen Kramer et al. "Evolution of optical access networks: Architectures and capacity upgrades". In: *Proceedings of the IEEE* 100.5 (2012), pp. 1188–1196. ISSN: 00189219. DOI: 10.1109/JPROC.2011.2176690.

[26] Jim Doherty. *SDN and NFV Simplified. A Visual Guide to Understand Software Defined Networks and Network Function Virtualization*. Person, 2016, p. 299. ISBN: 978-0-13-430640-7.

[27] Rajendra Chayapathi, Syed Farrukh Hassan, and Paresh Shah. *Network Functions Virtualization (NFV) with a Touch of SDN*. Addison-Wesley, 2017, p. 336.

[28] Edison F. Naranjo and Gustavo D. Salazar Ch. "Underlay and overlay networks: The approach to solve addressing and segmentation problems in the new networking era: VXLAN encapsulation with Cisco and open source networks". In: *2017 IEEE 2nd Ecuador Technical Chapters Meeting, ETCM 2017* 2017-January (2018), pp. 1–6. DOI: 10.1109/ETCM.2017.8247505.

[29] Max Ardica and Patrice Bellagamba. "VXLAN Multipod Design for Intra-Data Center and Geographically Dispersed Data Center Sites". In: (2016), pp. 1–58.

[30] Abel Tong and Kevin Wade. "Guia de NFV e SDN para operadoras e provedores de serviços". In: *Ciena* (2017), p. 36.

[31] Janet E. Burge et al. "An Architectural Framework". In: *Rationale-Based Software Engineering* 1 (2008), pp. 241–254. DOI: 10.1007/978-3-540-77583-6_17.

[32] Thomas D. Nadeau and Ken Gray. *SDN Software Defined Networks*. 2013, p. 352. ISBN: 978-1-449-34230-2.

[33] Rob Enns et al. *Network Configuration Protocol (NETCONF)*. RFC 6241. June 2011. DOI: 10.17487/RFC6241. URL: https://rfc-editor.org/rfc/rfc6241.txt.

[34] Broadband Forum. *About Broadband Forum*. URL: https://www.broadband-forum.org/about-bbf.

[35] Broadband Forum. *Open Broadband - Broadband Access Abstraction*. URL: https://obbaa.broadband-forum.org/overview/#overview.

[36] Broadband Forum. *TR-384 Cloud Central Office Reference Architectural Framework*. Tech. rep. January. 2018, pp. 1–80.

[37] Broadband Forum. *vOMCI Proxy and vOMCI Function*. URL: https://obbaa.broadband-forum.org/architecture/vomcipf/#vomcipf.

[38] Broadband Forum. *Open Broadband-Broadband Access Abstraction (OB-BAA) Overview*. URL: https://obbaa.broadband-forum.org/overview/#overview.

[39] Open Network Foundation. *2020 State of the ONF*. URL: https://opennetworking.org/news-and-events/blog/2020-state-of-the-onf/.

[40] Open Network Foundation. *CORD (Central Office Re-architected as a Datacenter) Indroduction*. URL: https://opennetworking.org/cord/.

[41] Open Network Foundation. *SEBA (SDN Enabled Broadband Access) Indroduction*. URL: https://wiki.opennetworking.org/display/COM/SEBA.

[42] Open Network Foundation. *VOLTHA (Virtual OLT Hardware Abstraction) Indroduction*. URL: https://wiki.opencord.org/display/CORD/VOLTHA.

[43] Open Network Foundation. *Relationship between BAA and VOLTHA open source projects for automating the access network for any operator deployment*. URL: https://www.broadband-forum.org/marketing/download/BAA_VOLTHA_open_source_projects.pdf.

[44] John Vollbrecht et al. *Extensible Authentication Protocol (EAP)*. RFC 3748. June 2004. DOI: 10.17487/RFC3748. URL: https://rfc-editor.org/rfc/rfc3748.txt.

[45] Jyh Cheng Chen and Yu Ping Wang. "Extensible Authentication Protocol (EAP) and IEEE 802.1x: Tutorial and Empirical Experience". In: *IEEE Communications Magazine* 43.12 (2005), S26–S32. ISSN: 01636804. DOI: 10.1109/MCOM.2005.1561920.

[46] Internet Assigned Numbers Authority (IANA). *Extensible Authentication Protocol (EAP) Registry*. URL: http://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml.

[47] Pat R. Calhoun and Dr. Bernard D. Aboba. *RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)*. RFC 3579. Sept. 2003. DOI: 10.17487/RFC3579. URL: https://rfc-editor.org/rfc/rfc3579.txt.

[48] Institute of Electrical and Electronics Engineers (IEEE). *IEEE Standard for Local and metropolitan area networks–Port-Based Network Access Control*. URL: https://1.ieee802.org/security/802-1x/.

[49] Fortinet. *What Is 802.1X Authentication?* URL: https://www.fortinet.com/resources/cyberglossary/802-1x-authentication.

[50] EVE-NG. *Emulated Virtual Environment Next Generation*. URL: https://www.eve-ng.net/index.php/documentation/.

[51] Docker. *Docker overview*. URL: https://docs.docker.com/get-started/overview/.

[52] Jetbrains. *Developer Ecosystem Survey 2020*. URL: https://www.jetbrains.com/lp/devecosystem-2020/go/.

[53] Aqua Cloud Native. *Container Images: Architecture and Best Practices*. URL: https://www.aquasec.com/cloud-native-academy/container-security/container-images/.

[54] Alpine Linux. *Alpine User Handbook*. URL: https://docs.alpinelinux.org/user-handbook/0.1a/index.html.

[55] cAdvisor. *cAdvisor GitHub Repository*. URL: https://github.com/google/cadvisor.

[56] ZCube. *ZCube cAdvisor Container Image*. URL: https://github.com/ZCube/cadvisor-docker.

[57] Pramod Shehan. *Containers metrics with Prometheus and Grafana Tutorial*. URL: https://pramodshehan.medium.com/containers-metrics-in-prometheus-and-grafana-389555499eb8.

[58] Helm. *Helm Webpage*. URL: https://helm.sh/.