Virtual Network Function Development for NG-PON Access Network Architecture

Igor Araújo[1], André Brízido[2] and Solange Rito Lima[1*]

[1*]Centro Algoritmi, Universidade do Minho, Braga, Portugal.
[2]Direção de Sistemas de Rede, Altice Labs, Aveiro, Portugal
Emails: solange@di.uminho.pt (corresponding author),
pg39254@alunos.uminho.pt, andre-d-brizido@alticelabs.com.

Abstract

Modern networks urge agility, flexibility, and capacity to cope with the growing demand for media content and applications increasingly oriented toward data consumption. The Central Offices (CO) of telecommunication providers, being a vital aggregator of different access networks, such as optical and mobile, need to be prepared to deal with these demands. The Open Broadband - Broadband Access Abstraction (OB-BAA) architecture fits into the initiative to modernize the Information Technology (IT) components of broadband networks, more specifically the COs. This paper discusses the development of a Virtualized Network Function (VNF) in the context of network security to be integrated as a component of an OB-BAA architecture guided by the Software-Defined Network (SDN) paradigm. More specifically, the authentication and authorization of network equipment within the IEEE 802.1X protocol are applied to Next Generation Passive Optical Networks (NG-PON). The VNF development is based on the Golang language combined with gRPC programmable interfaces for communication between the various elements of the OB-BAA architecture, and then the components were "containerized" and inserted in the Docker and Kubernetes virtualization frameworks of a multinational telecommunications operator. Finally, performance metrics such as computational resource usage (CPU, memory, and network I/O) and execution time of VNF processes were analyzed in usage tests with multiple supplicants and distinct operational modes, to attest to the most promising virtualization scenarios.

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

1 Introduction

Networks have been following the growth of application complexity and user demands for bandwidth, low latency, and high data volumes. Several emerging technologies and applications contribute to this challenging setting, such as Augmented Reality (AR), Ultra-High Definition (UHD) services, Machine-to-Machine (M2M) communication, adding heterogeneous constraints to a demanding network scene.

Network operators have been under pressure by the exponential requirements for flexibility, agility, and scalability. The optical fiber methods of data transmission have brought a big shift in reachability, capacity, and resilience compared with copper cable-based implementations. Moreover, Passive Optical Networks (PON) has stood out against other optical technologies, adding cost and energy savings to the list of benefits. PON and its versions as Gigabit PON (GPON), Next Generation PON (NG-PON), and 10-Gigabit PON (XG-PON), have consolidated as one of the most used broadband access networks from network operators [1]. Further, PON provided a significant expansion of bandwidth per client and network infrastructure quality, which opened new roads for the latest applications and impacted customers' habits in Internet consumption.

Central Offices (COs) are the first "big" hop in terms of infrastructure capacity for most of the broadband access networks. They are located at the network's edges and usually aggregate multiple PON, xDSL, Mobile, and other networks. Therefore, CO shows up as a good opportunity for Communications Service Providers (CSPs) to avail the applications urge demands for low latency, high throughput, and fast processing to generate new incomes [2]. However, to take advantage of this opportunity, it is imperative to modernize CO infrastructures to provide the desired flexibility, adaptability, and availability. To achieve these requirements, network softwarization is proposed, bringing the concepts of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) [3] [4] [5] [6]. In addition, the Open Broadband - Broadband Access Abstraction (OB-BAA) architecture, promoted by the Broadband Forum, aims to modernize the information technology components of broadband networks, more specifically the CO, by proposing a framework with well-defined reference points that could be followed by vendors, operators, and Infrastructure Providers to take advantage of the promoted integration.

Build upon OB-BAA, this work intends to evolve the traditional architecture of the CO, mainly related to broadband infrastructure [1], exploring how CO could benefit from the paradigms of SDN, NFV, and Virtual Network Function (VNF). Framed by a comprehensive workforce across multiple development teams of a major multinational telecommunications operator, this work addresses the development of a VNF for enhancing network security, more precisely the authentication and authorization of network equipment. The integration with distinct VNFs is assured by following the framework established by OB-BAA architecture, which defines well-known protocols, interfaces, and endpoints allowing for seamless integration between multiple asynchronous

developments. In this context, the main contributions of this paper are defined as follows:

- the traditional architecture of COs is clarified. Also, the state-of-art paradigms of SDN, NFV, and VNF are conceptualized in the broadband context, being debated how CO could benefit from them;
- the development of a VNF for Network Access Control based on the IEEE 802.1X protocol is explored for the legitimation of the identity of Optical Network Units (ONU) into the PON domain;
- related to the VNF development, relevant specifications are also outlined along the state-of-art, including the adopted programming language and protocol interfaces used to construct and integrate the VNF with other architectural components;
- the results of performance tests carried out to attest to the most suitable scenarios of VNF development and integration are deeply debated.

This paper is organized as follows: Section 2 provides background concepts and covers related work supporting the proposal; Section 3 presents the architecture integrating the developed security VNF; Section 4 and Section 5 are devoted to performance evaluation and results discussion; Section 6 presents the conclusions.

2 Background Concepts and Related Work

This section presents and interrelates the main concepts and works sustaining the proposed VNF. The debate starts with PON concepts and architecture, followed by a discussion on why and how virtualization and softwarization contribute to next-generation CO. Finally, security in the access layer, the VNF development use case, is detailed.

2.1 Passive Optical Networks (PON)

In the last decade, fiber technology started to expand its capillarity, spreading the concepts of Fiber-to-the-X (FTTx) and Passive Optical Network (PON).

Among the Optical Access Network (OAN) broadband technologies, PONs have excelled rather than others due to their cost-effectiveness. One reason for the advantages and preference by network providers is the fact that PONs avoid active equipment between providers and clients, conversely to Active Optical Network (AON) which needs an optical-electrical-optical at the remote node [1]. This passive approach consequently provides more savings of Capital Expenditure (CAPEX) and Operational Expenditure (OPEX), also because there is no power consumption for equipment throughout the Optical Distribution Network (ODN) [7], at a cost of bandwidth capacity and ability for reaching longer distances, when comparing to the AON scenario.

[1] Throughout the Optical Distribution Network, between the CO and the client, a splitter divides the mainstream of feeder fiber into multiple distribution fibers. The AON splitter is an active powered device, such as a switch or a router, while PONs rely on a passive splitter, which is a non-powered device, made mainly of mirrors and glasses, to divide the multiple client signals.

PON architecture from a high-level point of view is quite simple. It follows mainly a point-to-multipoint architecture that 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, where the aggregator equipment, the Optical Line Termination (OLT), is located. Each port serves multiple points, partitioning the stream of these single wires (feeder fiber) through a splitter, to various cables, each one connected to an Optical Network Unit (ONU) and Optical Network Terminal (ONT). The COs also support a significant number of other services and aggregate other technologies, such as the mobile network infrastructure and more. The second part is the ODN, the infrastructure between the CSP and the customers, which includes the fiber cables, splitting remote nodes (RN), and are all passive elements. Finally, in the customer side, the ONUs/ONTs receive a branch (distribution fiber) from the split feeder fiber. All these components are illustrated in Figure 1.

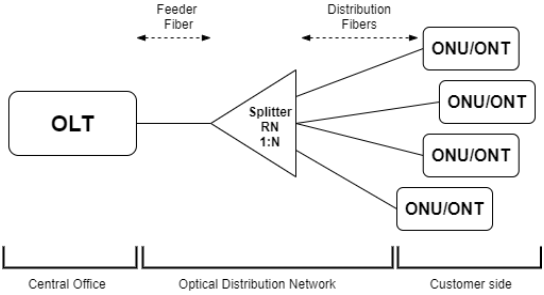

Fig. 1 Components of PON Architecture.

PON characteristics could explain why many carriers and service providers widely adopt this solution. These features are [8]:

- Resiliency - Optical technologies are immune to Electromagnetic Interference (EMI), Electromagnetic Compatibility (EMC), and durable when compared to copper-based access infrastructures. Yet, the ODN infrastructure has a single point of failure at the feeder fiber, requiring high protection against possible harm.
- Performance and Cost-efficiency - PON improves the number of services delivered with quality at an affordable cost - *deliver more with less*. Optical solutions avoid the OPEX costs in the maintenance of optoelectronic and electronic devices along the ODN, reducing dramatically failure rates and repair costs. PON is also a future-proof technology due to its high information-carrying capacity[2], and the mentioned inherent characteristics of optical material [9].

[2]The study reported in [10], proposing an optimization framework and providing a comparison between multiple generations of PON, shows that in scenarios of higher demand of bit rate, newer generations, which tend to be more expensive and with higher capacity, are more cost-effective than the old cheapest ones.

- Convergence and Integration - PON architecture supports distinct wired and wireless network technologies. An example is the Wireless-Optical Broad-Band 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, and these wireless base stations are connected to the ONU [11]. Moreover, PON integrates with copper solutions, for drops near the customer, and with Metro networks, which adapts the traditional PON "tree-and-branch" to ring topology, for Long Range PON (PON).

From the above, it is possible to perceive that Network Operators aggregate a significant number of technologies into a CO. Also, it is the uplink point to the core network. Thus, COs are strategic points to be improved and evolved and to offer new businesses and services. Still, they need to be more agile, flexible, and scalable, providing better management, orchestration, and visibility. To deliver these features, the SDN paradigm appears as a great candidate to leverage COs to the next stage.

2.2 Virtualization in Network Context

To address the evolution in data generation and consumption, network service providers try to improve their network capacity and scale, however, the traditional network architecture imposes constraints for this progression. Some of these limitations include [12]:

- Flexibility - As each vendor promotes its portfolio, their specific hardware and software solutions suffer from restricted/no-compatibility with other vendors.
- Scalability - Traditional on-premises equipment demands physical spaces with power consumption and substantial cabling paths, with scale constraints. On the software side, each vendor generally produces capacity-constrained products, requiring licences for upgrade, also impairing scale.
- Time-to-Market Challenges - Technology is constantly changing, often implying new features that require a vendor's box upgrade or replacement. This impacts time to deliver of new features to clients and affects revenue.
- Lack of Manageability - Well-known management protocols are implemented by almost all vendors, such as Simple Network Management Protocol (SNMP), Syslog, NetFlow. However, each vendor creates its management strategy and proprietary tools, undermining a vendor-agnostic management for fine-grain control and visibility of traditional network equipment.
- Operational Costs - The above impacts the operational costs of traditional network equipment buyers, demanding investment in training and certifications, which locks the client to a particular vendor.

Thus, the traditional network model suffers from blockades that prevent it from advancing together with the applications' demands. The following sections focus on recent paradigms to overcome the raised barriers.

2.2.1 Network Functions Virtualization

Virtualization is the abstraction of operating systems and applications from the physical hardware device. Bringing this definition to the network, it also abstracts the networking nodes from physical topology arrangement, which allows the administrator to group or arrange these elements in a new logical way disregarding their physical location [13].

Although lately network virtualization has been a recurring topic [14], the concept has been used for a long time, namely in Virtual Private Networks (VPNs), Virtual Local Area Networks (VLANs), and Multiprotocol Label Switching (MPLS). These examples of virtualization brought great techniques for network administrators to isolate uncorrelated traffic, protect their integrity, group logically distant nodes, and optimize network usage. Despite that, network hardware devices are still very nested, with restricted functions. So, NFV defines an abstraction approach concerning the use of generic hardware, Commercial Off-The-Shelf (COTS), having all physical capabilities regarding I/O and CPU.

The European Telecommunications Standards Institute (ETSI) and Industry Standards Group (ISG) established some recommendations about NFV, proposing the virtualization of functions from dedicated network equipment into Virtual Machines (VMs), allowing fast, flexible, and ubiquitous deployment [15]. Moreover, network functions become a piece of software, a VNF, portable, and easily upgradeable, rather than specialized network hardware that needs a physical replacement when 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 the business model, enabling providers to offer novel services and products. As drawbacks, the transition from legacy to a virtualized infrastructure imposes upfront costs of acquisition, and complexity exists in managing volatile demands. Additionally, not all software/hardware can be virtualized due to vendor constraints, and combining virtualization with open-source solutions could also be difficult due to the lack of support.

2.2.2 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 can be virtualized, and a VNF can perform each of these virtualized roles. So, enterprises are replacing several boxes, each with its own set of functions, for a consolidated piece of hardware, such as a standard x86 server, with multiple VMs sharing the server resources. In this new network architecture, different vendors can provide distinct VNFs, which eliminates the proprietary hardware constraints, such as the cost of upgrading a box. Furthermore, VNFs improve flexibility by eliminating location constraints for network equipment, allowing to scale and deploy functions near demanding places, e.g., COs, 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. The ETSI provided this framework, which is divided into three main blocks, the infrastructure block, virtualized function block, and management block, as illustrated in figure 2 (Left):

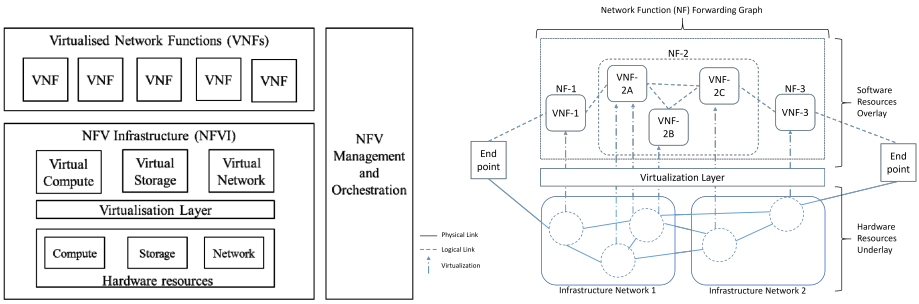

Fig. 2 Left: High-level overview of ETSI NFV framework; Right: Resource allocation to VNFs, overlay/underlay abstraction and forwarding graph [16].

In this framework, the software is so decoupled that one VNF can require hardware COTS resources from different pieces of hardware and a Network Function (NF) can hold multiple VNF. As an example, a firewall NF performs many features, such as Network Address Translation (NAT), Access Control List (ACL), Packet Inspection, etc, and the different VNFs can interoperate throughout the forwarding graph between two endpoints flow. Figure 2 (Right) illustrates this abstraction and independence of software overlay from hardware underlay. Moreover, for each forwarding graph, Management and Orchestration (MANO) can instantiate a different overlay graph and hardware allocation depending on the actual demands and underlay usage.

2.3 SDN-oriented CO Architecture

In the telecommunication and broadband sector, the BroadBand Forum (BBF) and the Open Network Foundation (ONF) are flagships, non-profit organizations working on projects such as 5G, Connected Home, Cloud, and Access Networks.

2.3.1 BroadBand-Forum (BBF)

Regarding the CO environment, the Cloud Central Office (CloudCO) project from BBF proposes an architecture redefinition of the access and aggregation networks hosted into traditional CO. This redefinition includes the use of SDN,

NFV, and cloud technologies to achieve and respond to the existing broadband networks' demands [17].

Inside the CloudCO initiative, the project open-source OB-BAA creates Northbound Abstraction Interfaces (NAI), Core Components and Southbound Adapter Interfaces (SAI) 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) for network operators through NAI, and to connect the access network devices (e.g., OLT, ONU) through SAI [18]. To guarantee portability and modularity, this project 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 NFV Infrastructure (NFVI), using the OB-BAA functions as VNFs, and also to be compatible with legacy components where some specific network functions are still performed by physical nodes, called Physical Network Functions (PNF) [17].

Figure 3 illustrates the OB-BAA architecture, which includes the BAA layer with Northbound/Southbound layers and interfaces. It also includes other services, namely Performance Monitoring, vOMCI Proxy, Control Relay, and examples of attachments to southbound (e.g., Access User plane) and northbound (e.g., Management and Control elements).

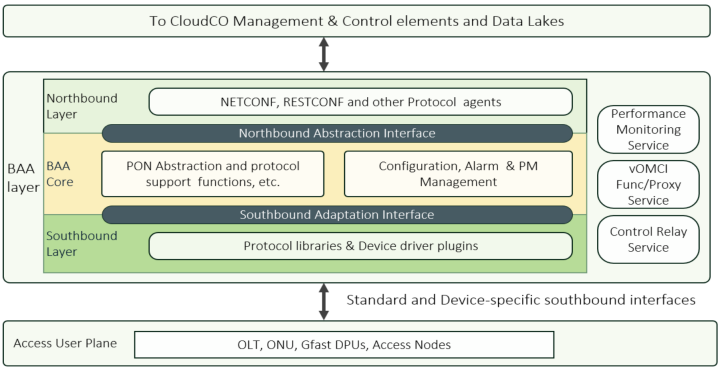

Fig. 3 OB-BAA architecture layers [18]

The BAA Core can deliver a set of functions that could be disaggregated from traditional CO elements into PNF or from legacy devices, and be performed in the BAA Core under an NFVI substrate into the CloudCO environment. Northbound and southbound interfaces enable the communication between SDN Management and Control (M&C) components in the upper layer, and with external devices in the lower layer. Northbound abstraction and southbound adaptation interfaces implement data models, Application Programming Interface (API), libraries, and driver plugins to ensure the compatibility between these interfaces with SDN M&C and the Access Nodes (AN) [18] [17]. In more detail:

Southbound Layer and Interface - the southbound layer, through southbound adaptation interfaces, implements adapters that specify the required data models to integrate the access nodes into the BAA layer. The device adapters can use the southbound protocol libraries and SAI API, or when needed, can implement their specific communication protocol.

Northbound Layer and Interface - the northbound layer implements protocols to establish communication between the BAA layer with multiple network management tools and orchestrators, such as 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, such as 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/from SDN applications, such as AAA, DHCP, or others. 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, a VCA is available in case the access nodes require some adjustment.

vOMCI Function and Proxy - the Virtual ONT Management Control Interface (vOMCI) function has the role of formatting and translating the YANG request/responses to/from Virtual OLT Management Function (vOLTMF) into the OMCI messages [19]. Moreover, the vOMCI function is responsible for encapsulating/de-encapsulating and sending/receiving messages originated/destined to ONUs through the vOMCI Proxy, and then converting the OCMI messages into understandable data vOLT Management Functions (vOLTMF). The vOMCI Proxy is deployed as a microservice. It allows the connection of multiple OLTs and the vOMCI Proxy acting as gRPC endpoints, proxying their communication with the vOMCI Functions. It is also helpful for troubleshooting and management purposes. The proxy aggregate and maintain the associations between the OLTs and corresponding vOMCI function [19].

2.3.2 Open Network Foundation (ONF)

The Open Network 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 solutions 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 [20].

The ONF has projects divided into mobile, broadband, edge cloud infrastructure, and SDN solutions. In the broadband section, the SDN Enabled Broadband Access (SEBA) and Virtual OLT Hardware Abstraction (VOLTHA) projects are under the CORD platform, which has the mission

to transform the edge of the operator network into agile service delivery. Moreover, CORD integrates multiple open-source projects, delivering an open, programmable, and agile platform for network operators [21].

SEBA is a lightweight platform based on a variant of Residential-CORD. It supports virtualized access technologies of the carrier network, such as PON, G.Fast, DOCSIS, and others. It is built with Kubernetes and operationalized with FCAPSI, and OSS integration [22].

VOLTHA relies on the extensive usage of SDN hardware abstraction of COTS hardware instead of dedicated and inflexible equipment, applying this abstraction principle to broadband optical access networks, by 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 turns the access network into an abstract programmable switch. Moreover, it also allows communication with PON hardware devices using vendor-specific protocols through OLT and ONU adapters [23].

2.3.3 BBF and ONF Partnership

The broadband networking operators are constantly looking forward to better solutions for their customers with greater cost-effectiveness and robustness. Having this in mind, the BBF and ONF initiatives stand out and promote SDN-driven solutions that rely on open-source projects, the BBF's BAA and the ONF's vOLTHA [24]. Both solutions intend to abstract different components of the broadband architecture, making them complementary. BAA abstracts the control and management for any Access Node (AN) in traditional or *cloudified* COs. VOLTHA proposes an abstraction of low-level silicon to enable vendor-neutral hardware for AN [24]. Thus, these solutions supply the demands of carriers and operators, providing modern, flexible, and agile infrastructures to deliver improved services.

2.4 Network Security in Access Layer

2.4.1 Framework Extensible Authentication Protocol (EAP)

The Extensible Authentication Protocol (EAP) is a well-defined protocol to carry authentication and authorization messages in a network. The IEEE 802.1X protocol relies on it to provide the exchange message encapsulation framework, more specifically, it uses EAP over LAN (EAPOL) in IEEE 802.3, Point-to-Point Protocol (PPP), IEEE 802.11, or other LAN protocol. EAP typically runs directly over the data link layer, without requiring the Internet Protocol (IP) [25], setting a middleware to the authentication layer where authentication methods could be easily attached. The EAP supports various authentication practices such as token cards, smart cards, preshared keys, Kerberos, one-time passwords, certificates, public key authentication, and others. EAP-MD5, TLS, TTLS, PSK, PEAP, IKEv2 and PEAP-MSCHAPv2 are common examples of EAP methods [26]. A comprehensive list can be found at [27].

2.4.2 IEEE 802.1X Architecture

The IEEE 802.1X protocol (Dot1x), brings security to the link layer for Port-based Network Access Control (PNAC), which 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 unauthorized rogue devices from walking in and compromising the network [28]. Additionally, it is a layer 2 protocol, closer to the client/user and, for this reason, can be deployed at an effective cost compared to other security implementations in upper layers (although could be complementary). Using 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 Authentication, Authorization, and Accounting (AAA) messages, until the user or device is authenticated and authorized (which can be also accounted for future auditing). Therefore, this protocol brings benefits to the network's robustness and security, visibility, and transparency.

There are three main elements in an 802.1X network [29]: *Supplicant* - device that intends to access the network by engaging in an EAP communication exchange between the supplicant and the authenticator (e.g., PCs, Smartphones, CCTV cameras, ONTs, ONUs, etc); *Authenticator* - network equipment where the supplicant will be connected and exchange EAP messages. This device will relay AAA messages between the supplicant and the authentication server. It is commonly called a Network Access Server (NAS) and possible authenticators are switches, wireless access points, OLTs, etc; *Authentication Server* - this element has a credentials database for supplicant AAA, e.g., Remote Authentication Dial-In User Service (RADIUS) Server, Lightweight Directory Access Protocol (LDAP) Server, Active Directory, MySQL Server, etc.

The IEEE 802.1X protocol comprises five phases - initiation, authentication, authorization, accounting, and termination, as illustrated in Figure 6 (excluding 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, sends the first frame *EAP over LAN (EAPoL) Start* over the data link layer, encapsulated into the EAP frame. A DHCP, ARP, DHCPv6, ND, or any packet can also initiate the dot1x process. The authenticator receives this frame and answers with an *EAP Identity Request* frame, inquiring about the supplicant's identity information.
- Authentication - Upon receipt the *EAP Identity Request*, the supplicant initiates the authentication process, sending its identity through the *EAP Identity Response* message to the authenticator. The authenticator forwards the supplicant identity information to the RADIUS authentication server for validation as an Attribute-Value Pairs (AVP) via a *RADIUS Access-Request* message. If the supplicant's identity attribute is found in the database, 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. If so, the supplicant answers with an *EAP Response* accepting the proposed challenge. Then, the authenticator forwards the response over the *RADIUS Access-Challenge EAP Response* message.

- 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 previous steps. The *EAP-Success* message can contain attributes regarding the supplicant's access permission, such as the associated VLAN, a specific Dynamic Access Control List (dACL), and other appropriate features.
- Accounting - The accounting feature is optional, being considered a good practice to configure for auditing reasons. 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, containing information inserted in the AVP field about the session established with the supplicant.
- Termination - The termination occurs when the supplicant sends a *EAPoL-Logoff* packet to the authenticator, which leads to a port status change from the 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.

3 IEEE 802.1X VNF Proposed Approach

This work explores VNF deployment in a PON OB-BAA architecture of a multinational telecommunications operator, integrating SDN concepts. More specifically, the IEEE 802.1X network access control was chosen as the network function to virtualize into the NG-PON domain. The 802.1X VNF was developed to be attached to an NFVI, a fully virtualized platform, or in a hybrid environment with legacy on-premise hardware and an SDN-oriented setting. This section describes the adopted architecture, initially introduced in [30][3], the developing VNF environment, and the test scenarios considered. The practical experiments and tests are divided into three phases, evolving in code maturity and integration with the architecture where this project is embedded.

3.1 Adopted Architecture

Network virtualization concepts are here explored including new initiatives regarding SDN applicability in PON environment. The OB-BAA architecture (see Figure 3) was selected for integration due to the existing partnership and collaboration with the BBF group.

[3]This article extends the previous conference paper by providing an encompassing debate articulating the paradigms contributing to next-generation COs, by detailing the tools, architecture, and phases of VNF development, and by providing results on all the 24 scenarios considered in the evaluation, for processing time and computational resources. Therefore, it sustains and clarifies the most suitable scenarios for VNF development and integration.

As mentioned, the OB-BAA architecture proposes a framework with well-defined reference points that could be followed by vendors, operators, and infrastructure providers to promote flexibility and integration. This framework enables the development of part of the architecture elements, assuring the attachment and interoperation with other parts developed by distinct organizations.

3.2 Supporting Tools

To achieve the proposed goal, first, a simple network topology was emulated for the preliminary studies of IEEE 802.1X. The software development started using a performative programming language suitable to the adopted architecture. Afterward, the network elements become virtualized and containerized into microservices, and the software development shifted to this environment. The tools supporting this progressive approach are discussed next.

3.2.1 Network Emulation Tool

In the first phase of the tests, aimed at exploring the phases of dot1x, the Emulated Virtual Environment Next Generation (EVE-NG) [31] was selected to create the test environment. EVE-NG is a Linux Ubuntu virtual machine that needs to be installed on a hypervisor such as VMware (recommended virtualization platform) or installed on Bare Metal. This tool is clientless, not requiring additional software installation, such as other emulators. Its GUI interface is accessible through HTML5 through its IP address, supporting the entire process of creating topologies and running simulations.

3.2.2 Virtualization Tools

This project is leveraged by the most used virtualization platform tools, Docker and Kubernetes (K8s), which enable containerization of custom and optimized operational systems and software applications for fast delivery [32].

These tools supported the third phase of the project development by leveraging the scale tests, using multiple supplicants and OLT, under the Docker platform. This environment is segmented into parts, according to the core elements 802.1X VNF, authentication server, and other BBF's architecture actors, which are contained in a computational resource pool, an NFVI, typically hosted in Kubernetes. These virtualization tools enable a great opportunity to test, develop, deliver, and put in production solutions, in a fast and scalable way.

3.2.3 Development Languages and Tools

For software development, a goal is to create an application that implements a performative NF, scalable to meet a large number of requests. The adopted OB-BAA architecture, mainly related to CO abstraction, implicitly suggests candidates for this task, namely Java, C, and Golang (GO).

Golang language is a powerful language used to build tools such as Kubernetes, Docker, and Vault. GO is also very suitable and used for microservices, having an outstanding performance due to its concurrency technique based on GO routines and channels, efficient in memory allocation and data [33]. Moreover, GO has a simple language syntax, well-documented codes, comprehensive libraries, garbage collected and compiled into a single binary, among other characteristics, resulting in a great programming language option for networking and gRPC APIs applications for the proposed BAA's architecture.

3.2.4 Monitoring and Visualization Tools

For monitoring and visualization, this project resorts to Prometheus and Grafana. Both tools are open-source and have a big community backing up. Prometheus implements a dimensional data model, collecting and storing metrics and key/value as time-series data from many sources, and has a flexible query language, the PromQL, which allows querying for many visualization tools. Grafana is a common candidate as 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 aggregates them into a single-pane-of-glass.

3.3 Project Phases

This project was divided into phases having as objectives: (i) to devise the virtualization of IEEE 802.1X; (ii) to evolve this knowledge in emulated networks and virtualized scenarios (using the presented tools); (iii) to reach a VNF application to be attached to an NFVI or a hybrid environment.

3.3.1 Phase 1: Deepening into 802.1X protocol

For the first phase an elementary topology is set using EVE-NG, with three dot1x actors: the supplicant, the authentication server, and the authenticator, as illustrated in Figure 4. In this scenario, Dot1x access security, PON broadband access network, VNFs, and correlated subjects were tested, and Dot1x specifications put into practice.

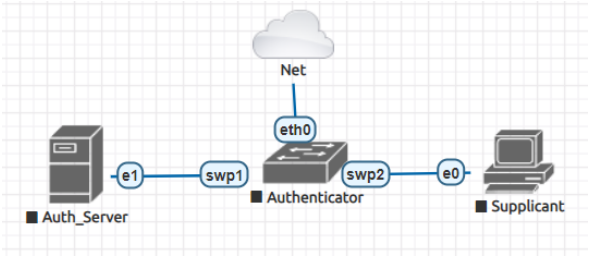

Fig. 4 Simple Dot1x topology.

3.3.2 Phase 2: Proxying EAP messages with Golang

This phase introduced a new component into the topology, a frame EAP proxy called OLT-App. This proxy stays listening to EAP frames from supplicants and forwards them to the authenticator. The proxy is built using Golang and comprises libraries to listen to Ethernet frames and Transmission Control Protocol (TCP) packets in specified host interfaces. The aim is to evolve this EAP proxy module to receive these frames from supplicants and forward them to the Cumulus Linux, which has the authenticator role.

3.3.3 Phase 3: Virtualizing Dot1x elements

In this phase, the Golang development advanced towards more 802.1X protocol actors, namely the 802.1X VNF, which replaced the Cumulus Linux as the authenticator, and OLT-App left the proxy role, now acting as an OLT's embedded software module. Moreover, all the topology elements have been containerized into images, to take advantage of virtualization benefits, such as fast deployment, flexibility of scaling in/out, portability with many infrastructures, among other mentioned advantages.

More specifically, the Google Remote Procedure Call (gRPC) communication was established between the application embedded into OLT (OLT-App) and Control Relay, and between Control Relay and the 802.1X VNF, establishing the first communication channel, the Control Relay Traffic Flow mode. The second channel is the direct communication between OLT-App and VNF, called Straight Traffic Flow mode, as illustrated in Figure 5.

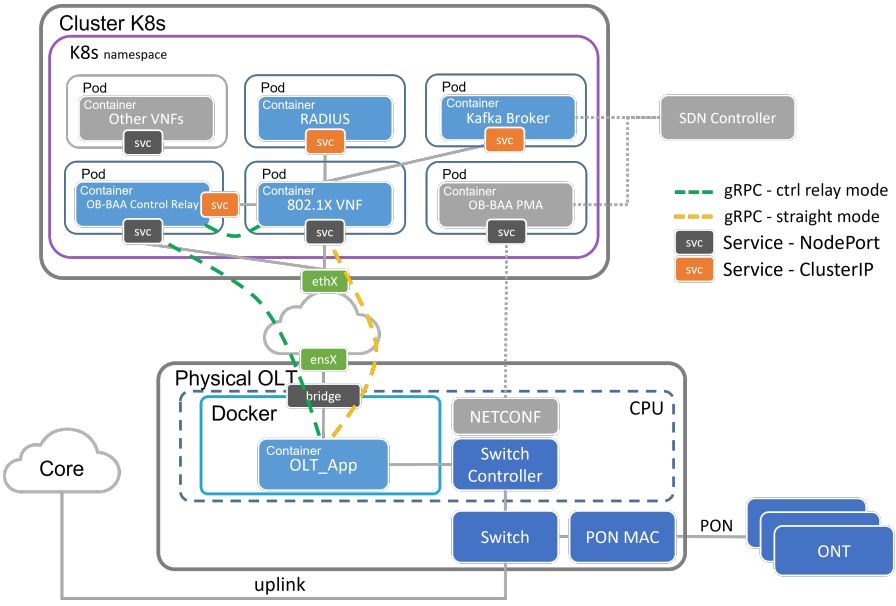

Fig. 5 Diagram of phase 3 topology elements.

The Control Relay mode is preferred because additional information about the AAA process can be shared directly with the BAA framework for future use. Moreover, these actors were 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 the SDN controller, which deploys rules to the infrastructure via Persistent Management Agent (PMA), for example, allowing traffic and setting the appropriated network configurations. Independently the settled way, the OLT-App will receive the EAP messages from supplicants and encapsulate these messages into the gRPC channel with the VNF destination. When the VNF receives the gRPC message, it de-encapsulates the EAP payload which is sent to RADIUS.

The reverse way back path is analogous, and the VNF waits and retrieves a RADIUS response, encapsulates the EAP message into gRPC, and then sends it back to OLT-App to deliver to the supplicant. This routine is repeated for every interaction. During the AAA process, the session-relevant information is forwarded via Kafka message for SDN controller consumption. With this information, the SDN controller can have full interaction between the supplicant's first request for access until the full acceptance and registration.

Details of the SDN Controller, PMA, and other VNF represented in Figure 5 are out of the scope of this paper.

4 Test Scenarios and Results

This section presents the devised test scenarios and the evaluation results obtained from comparing the following cases: (i) the infrastructure runs the supplicant and OLT-App into a physical OLT or a Virtual Machine in a regular COTS server; (ii) the core components run into Kubernetes or Docker, which are the most used open-source types of virtualization containers; (iii) the operational mode where traffic goes from OLT-App to VNF through the Control-Relay element proposed by OB-BAA architecture, which acts as a proxy, or Straight flow between OLT-App and VNF, without using the Control-Relay middleware element, as illustrated in Figure 5. Therefore, the following combinations were studied: Physical OLT/Virtual Machine - Docker/Kubernetes - Control Relay/Straight.

The metrics considered in the evaluation are Processing Time and Computational Resources usage (detailed in Section 4.2), under 30, 60, and 90 supplicants. In initial tests, when monitoring instances (Prometheus and Grafana), and secondary core elements (Kafka and Baa) was not running, 150 supplicants were reached. As a result, 24 test scenarios are evaluated, and each test is repeated 5 times. The obtained results are compiled in Section 4.3.

4.1 *Hosting Infrastructure*

Hardware Specification - The infrastructure supporting the tests divides into two main blocks, the Docker and the Kubernetes environment. For Docker, a VM with 2 vCPUs and 4GB of memory, and a physical OLT with Dual

Core CPUs ARM64 and 4GB of memory was used. For Kubernetes, a namespace into the provider cluster was made available, with 3 physical nodes, each with a 2x Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz (12-core) and 128 GB of memory. In this infrastructure only the Network Access Control VNF is under evaluation, exploring the advantages and limitations of different scenarios of deployment. Although aspects, such as the memory of the physical OLT and the CPU architecture limits the scalability analysis (further debated in Section 5), which would be stressed with more VNFs competing for hardware resources, as a proof-of-concept the comparative quantitative results provide relevant insights on what to expect when using distinct deployment scenarios and virtualization frameworks.

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 that can be deployed consistently [34]. To improve efficiency, lightweight images to instantiate the microservices were considered. For being small, simple, and secure [35], the Alpine Linux image was chosen as the base image for the following microservices: OLT-App, VNF 802.1X, supplicant, Kafka consumer, Monitor (Prometheus and 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 Control Relay, which is a part of core microservices from OB-BAA architecture, the images were built and provided by OB-BAA Github repository [36].

4.2 Performance Evaluation Metrics

4.2.1 Processing Time of Developed Applications

To evaluate time metrics a time counter was settled into the Golang applications code to measure the spent time on each step concerning the AAA processes. Figure 6 summarizes the Dot1x sequence, identifying in blue color the measured intervals between messages. The brackets in red indicate the OLT-App metrics, while the brackets in orange are the metrics on the VNF side. Each supplicant interaction generates a log output in VNF and OLT-App microservices, from which are evaluated the statistics maximum/minimum and average registered time.

4.2.2 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. The cAdvisor is an open-source, community-maintained container image tool led by Google, Intel, VMware, and RedHat, that is used to monitor containers. It can gather, process, and export metrics such as CPU, I/O, and memory usage from the tested containers, which can be afterward caught using Prometheus to plot in Grafana. The Loki stack is a well-pre-defined set of multiple containers which perform their specific functions and are deployed

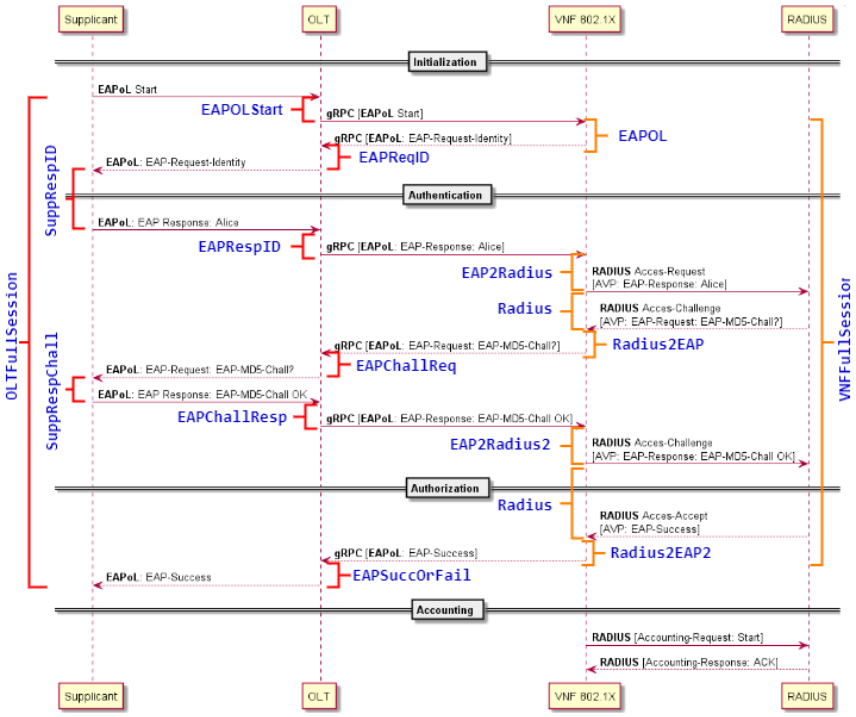

Fig. 6 Dot1x sequence diagram with performance spending time metrics.

together via Helm Charts (packet manager for Kubernetes) [37]. Grafana allows visualizing all gathered data to understand the computational resources consumption - CPU, memory, and network I/O, of each container.

4.3 Evaluation Results

This section presents the results of the authentication and authorization of a group of supplicants using the IEEE 802.1X protocol with the MD5-Challenge authentication method. Firstly, the time metrics illustrated in Figure 6 are collected and summarized, and then, graphical results evaluating CPU, memory, and network I/O are presented.

4.3.1 Processing Time

For brevity, from the eight tables of results obtained (see summary in Table 3) are detailed the results for Physical OLT and Dockers scenario (see Table 1), and for Virtual Machine and Dockers scenario (see Table 2), in both traffic flow modes. In more detail, Processing Time at each Straight and Control Relay modes registered the spent time of each metric and also the rate of processed requests per second. The time values shown are in milliseconds except for the request-per-second metrics. As mentioned, these examples are complemented by the summary provided in Table 3, which highlights the best cases

considering OLT Application Metric Statistics, VNF Metric Statistics, and Core Elements Performance.

Table 1 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

Traffic Flow Modes and Hosting Environments - Table 3 shows that the Control Relay traffic flow mode had a better performance (spent less time on average in most metrics for 6 out of 8 scenarios) for OLT-App and a draw between both modes for VNF metrics. This was not expected because the Control Relay element added a hop throughout the flow. The probable reason is that Control Relay was already put into the production stage, which implies a better construction and well-defined interfaces than OLT-App, which is yet in the development phase.

Docker vs. Kubernetes - Regarding Docker vs. Kubernetes hosting the Core components, Kubernetes (K8s) 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. 47 out of 76 processing time measures showed better performance on the K8s cluster for hosting the infrastructure. Docker showed a better performance for OLT-App running into Physical OLT (72.2% of all metrics) and slightly better for VNF statistics (55% of all metrics). When the OLT-App and Supplicants are hosted in VM, the results between Docker and K8s had a wider difference, and K8s got a significant advantage over Docker; in some metrics, K8s is 5x up to 10x more performative. Considering the overall results, Kubernetes reveals to be more performative than Docker. Although the test scenarios need in

Table 2 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 3 Summary of results

	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			

future to be stressed with large-scale tests, the K8S showed its strength points regarding orchestration.

OLT-App and VNF Metrics Differences - Looking at details in Tables 1 and 2 regarding the differences in spent processing time of OLT-App and VNF, VNF had around a 100x faster processing time than OLT-App's. VNF achieved results in a dozen of microsecond scale, and the OLT-App obtained results of few milliseconds (ms) for a processing step. The average spent time for all combining processes of OLT-App is between 18ms to 40ms, with some isolated tests going up to 131ms. Regarding the spent time for VNF, it ranges 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, between 5000 and 7000 requests per second. These numbers consider only the VNF's spent time because, in its lifecycle, there is

the RADIUS procedure, which is a bottleneck (consuming on average 6ms to 16ms, reaching in some cases up to 1500ms). This difference between OLT-App and VNF is justified by VNF handling multiple messages simultaneously, coming from many authenticators (OLTs).

4.3.2 Computational Resources

For the scenarios considered in Table 3, the computational resources CPU, Memory, and Network I/O (received and sent network traffic) on each element as a result of supplicants' access control were measured for different scales (30, 60, and 90 supplicants). The measured groups include Core Elements (VNF, Control Relay, FreeRADIUS, BAA, and Kafka), OLTs, and Supplicants. The presented graphical results show the Minimum, Maximum, and Average values in the evaluation period, the Current value in the export moment, and the Total (when applicable) aggregate measure in the period.

The objective is to observe the resource consumption as the scale grows and the impact on each element. For brevity, the scale of 60 supplicants is presented for the scenarios: Docker/Kubernetes hosting the core elements in Straight mode (Figures 7 and 8); Docker hosting the core elements in Control Relay mode (9); and Physical OLT/VM in Docker and Straight mode (10 and 11) are illustrated.

Scale Analysis - Network traffic for Core elements scales as expected, as it linear and proportionally increases with the number of supplicants. VNF, FreeRADIUS, and Kafka's Network Traffic also showed proportionality in traffic volumes. not so evident for the Control Relay and BAA, due to the transmission of control data and additional info regarding the connection. Regarding the CPU utilization, only the VNF and FreeRADIUS had a proportional increase near the number of supplicants' growth. Other elements (BAA, Kafka, and Control Relay) showed less consistent behavior. For instance, the Control Relay got a small decrease between 30 and 60 supplicants and an increase for 90 supplicants. The same occurs for memory consumption. The VNF evinced the expected proportionality between tests. OLT and Supplicant results for Network I/O growth also as expected. CPU and memory increased too, however, not proportionally, possibly due to internal running processes which consume resources independently of the supplicants' number.

Comparison Between Hosting Environments (Docker vs. Kubernetes) - Network I/O analyses reveal that the VNF receives significant traffic when compared to the remaining elements, due to the payload related to each supplicant information. The same preponderance is not observed in the sent traffic as only an authentication and authorization response is required. For the Sent, the Kafka element assumes a more determinant role as it streams data flows from many sources, mainly with control content, to other OB-BAA elements not evinced in the present context. Looking at the CPU usage, Docker (in Figure 7) seems to be more resource-saving than K8s (in Figure 8) 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 (Kafka and BAA curves were suppressed for a more detailed analysis of the remaining elements' contribution). Surprisingly, the FreeRADIUS reduced its memory consumption considerably, being more than nine times lower. This behavior could be explained considering that the Kubernetes environment had higher available resources than Docker, and was used more intensively.

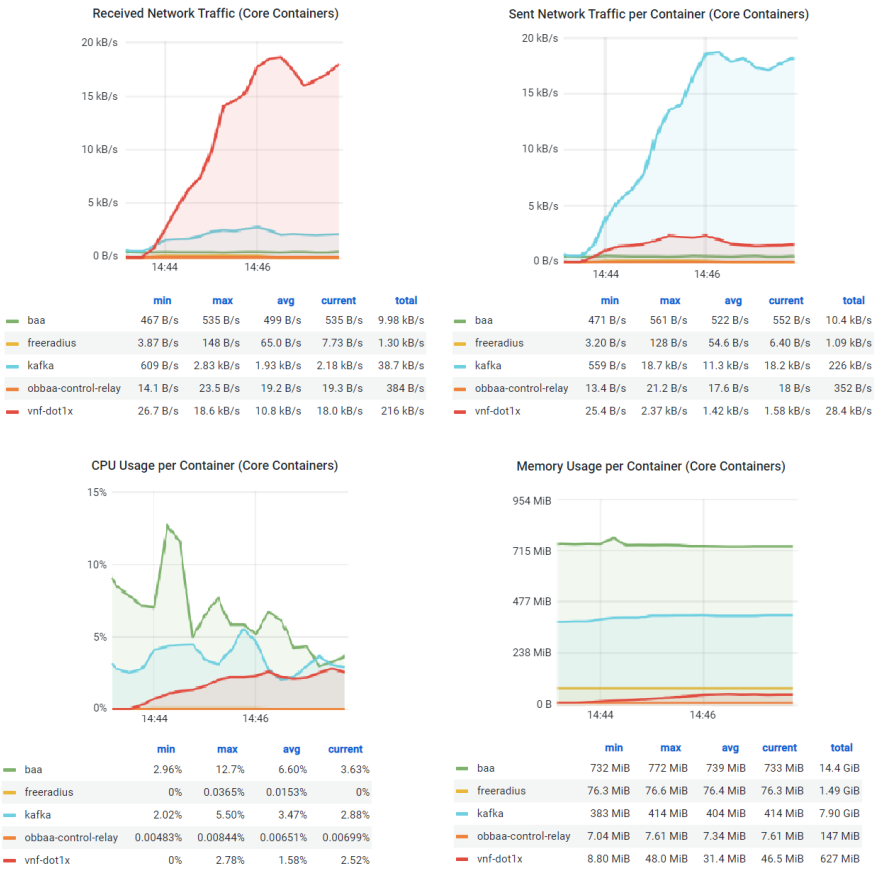

Fig. 7 Core elements hosted into Docker environment in straight mode.

Comparison Between Traffic Flow Modes (Straight vs. Control Relay modes) - This evaluation resorted to test scenarios executed with core elements hosted into Docker, being the supplicants and OLT-App hosted into physical OLT. The comparison between results in Figure 7 and Figure 9, confirm the expectation of a significant impact on Control Relay computational resource usage, without relevant changes in the other elements. CPU and Memory metrics in Figure 9 also have Kafka and BAA curves hidden to allow more detail about other main elements contributing to CPU and memory consumption.

As shown, the Network I/O measured at Control Relay experienced a considerable impact, leveraging the network usage up to 29,5 times, the CPU up to 19,7 times, and memory up to 3,25 times.

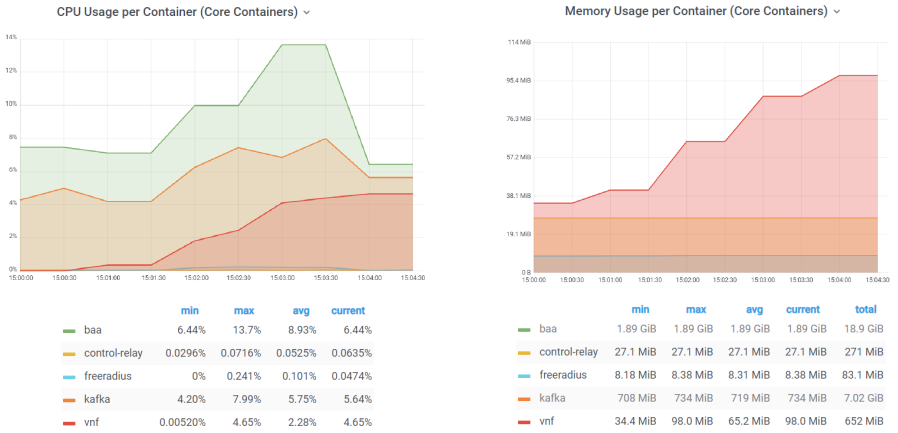

Fig. 8 CPU (Left) and Memory (Right) Core elements hosted into Kubernetes environment in straight mode.

Comparison Between Hosting Environment (VM vs. Physical OLT) for OLT-App and Supplicant - The results of this comparison are illustrated in Figures 10 and 11, considering the scenario running the core elements on Docker with straight traffic mode. In Figure 10 the performance of OLT_App container hosted into a Physical OLT is represented, and in Figure 11 the performance of OLT_Apps hosted in a virtual machine is added. In this last figure, OLT-App1/OLT-App2 denote the separation of 60 supplicants into two 30-supplicant instances.

The results reveal that OLT-App hosted into physical OLT got a higher consumption for all metrics, Network I/O, CPU, and memory. This behavior repeated for all scenarios except the K8s in straight traffic flow mode, where 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. These results also point out that using separate instances to deal with large amounts of supplicants may bring CPU and memory benefits in the long term.

When comparing the Supplicants' results for these scenarios, the Network I/O consumption is higher for VM hosting, similarly in all scenarios. The CPU and memory is high on physical OLT when the core elements are hosted in Docker. When they are on K8s, the Supplicants' CPU is higher on VM deployment. The Supplicants' memory consumption is higher on physical OLT deployment than VM when running core elements on K8s in straight traffic mode.

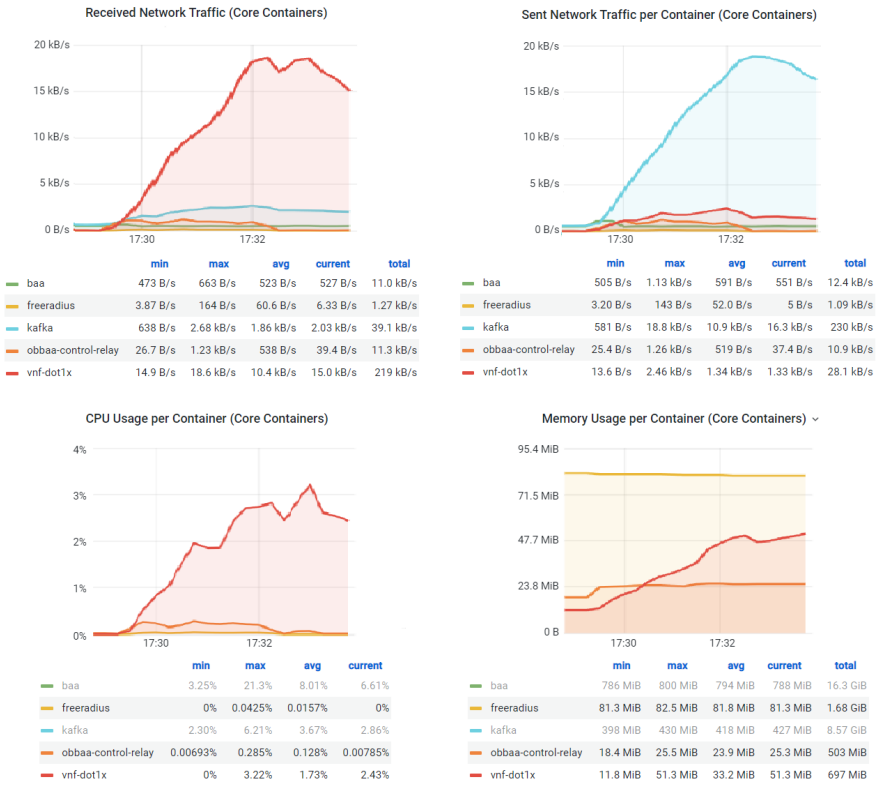

Fig. 9 Core elements hosted into Docker environment in Control Relay mode.

5 Overview

This section synthesizes the main outcomes and constraints of the evaluation described in the previous section.

From the analysis carried out, important results could be inferred, namely the hosting environment for supplicants and the superior performance of OLT-App compared to Virtual Machine hosting. This better performance could be explained due to the higher consumption of Physical OLT hardware rather than VM for the same scalability scenarios. However, the memory hardware of the physical OLT represents a limitation as it does not perform swapping memory allocation. This constraint the scalability tests regarding memory usage, setting a threshold, in addition to the physical OLT ARM64 CPU architecture.

Considering the scalability tests, for the increasing number of supplicants the processing time did not suffer a significant increase for all metrics. The results regarding the computational resources consumption showed to be more predictable, as the main elements who were actively engaged in the tests had, in most cases, increased their consumption with scale growth. Although, this increase is not always proportional and linear to supplicant scaling, for

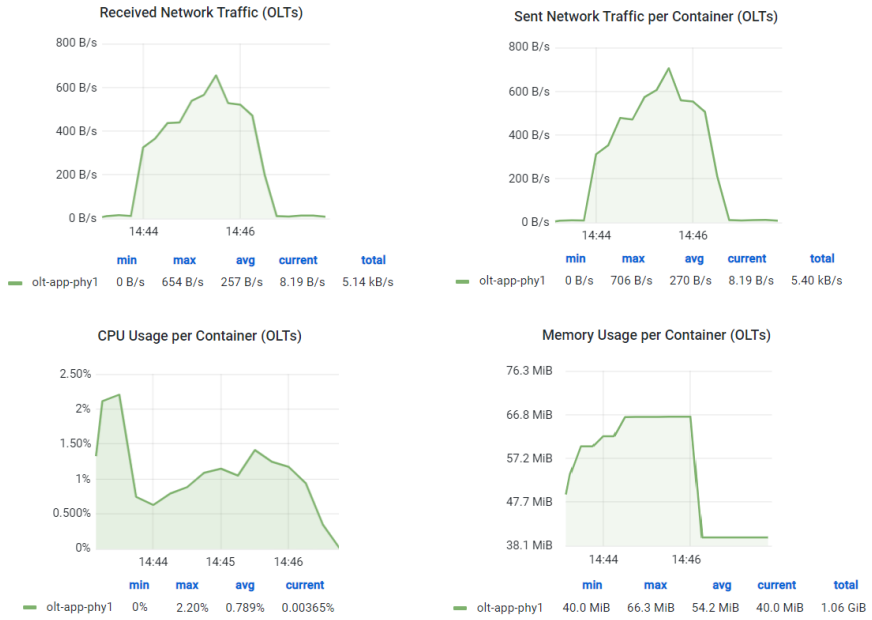

Fig. 10 Physical OLT results for hosting environment of OLT-App and supplicants comparison.

instance, the network traffic and memory of VNF had exponential growth. One reason for this behavior is that supplicants start randomly between 0 and 20 seconds after their container starts, and multiple supplicants can send requests simultaneously. Thus, the FreeRADIUS server may receive near requests and drop packets, resulting in occasional retransmissions from supplicants. As the number of supplicants grows, the probability of this occurrence gets even higher. Still, in VNF Statistics it is possible to quantify how much time the FreeRADIUS consumes compared to the other processes. The VNF process consumes around a dozen microseconds, facing a few milliseconds up to a dozen milliseconds of FreeRADIUS consumption. Thus, the FreeRADIUS is around one hundred times slower than a VNF Process, becoming a bottleneck.

Regarding the traffic flow mode, the results demonstrated that the Control Relay mode is more performative for OLT-App statistics and also on VNF statistics running on Kubernetes environment; the Straight mode achieved better results running on Docker environment. Despite the advantage of Control Relay mode deployment over Straight mode, namely in stability and consistency of results, the differences in several metrics motivate further tests in larger scenarios.

For the hosting environment, Docker and Kubernetes, overall it is not possible to elect a well-defined winner. Although Docker achieved better results in scenarios associated with physical OLT, Kubernetes performed better with OLT on VM. When looking at metrics separately, Kubernetes achieved better

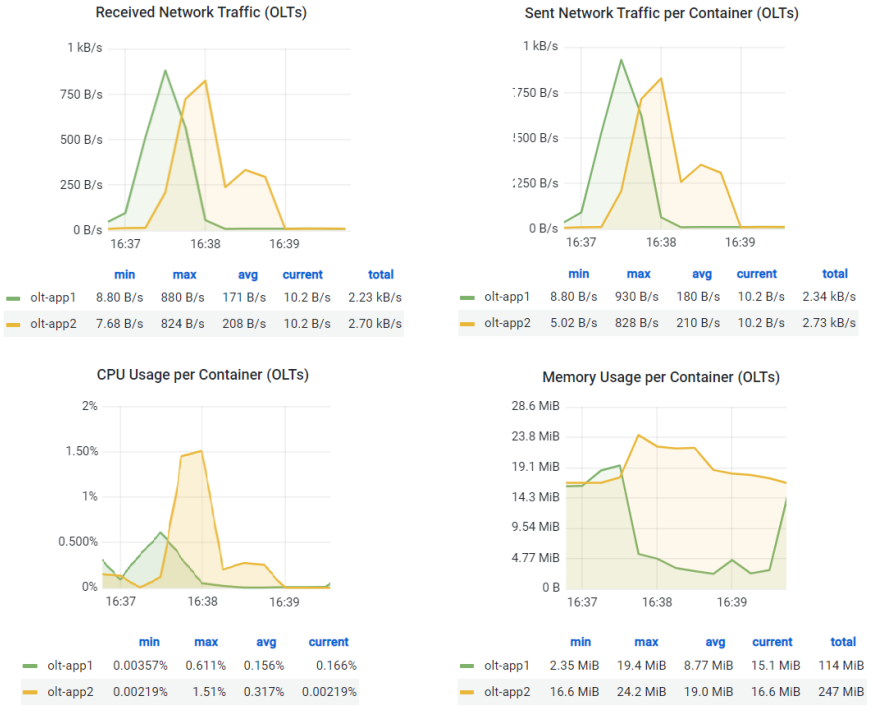

Fig. 11 OLT on Virtual Machine results for hosting environment of OLT-App and supplicants comparison.

performance for more metrics than Docker. The hardware consumption was also slightly better for Kubernetes. Considering that Kubernetes has a robust infrastructure set, it is expectable to achieve better scale results.

In a nutshell, it can be concluded that:

- the developed IEEE 802.1X VNF got the expected results. Processing time results on a microsecond scale were obtained, allowing a VNF instance to process thousands of requests per second;
- the RADIUS server presented itself as a significant bottleneck for the process;
- the utilization of the OB-BAA Control Relay showed to be a 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 showed a feasible deployment for the OLT-App module for communication with VNF. However, the supplicants had also to be executed into physical OLT, consuming computational resources for this. Thus, a better performance running only the OLT-App into physical OLT and the supplicants into physical ONT/ONUs is expected (the required devices were not available for testing);

- the provided tests validated the usage of Docker or Kubernetes as a hosting environment, showing that the proposed approach can benefit from containerized solutions. The quantitative results also revealed the expected behavior in each comparative deployment scenario.

6 Conclusions

This paper explored the development of a security VNF in the OB-BAA context, more precisely an IEEE 802.1X authentication and authorization VNF applied to the NGPON architecture of a major telecommunications operator. Through this case study, this work explores the evolution of the traditional architecture of COs by integrating the benefits of softwarization and virtualization.

After the conceptualization of PON technologies, SDN, NFV, and VNF concepts in broadband networks, the 802.1X protocol (Dot1x) and corresponding VNF proposal was debated. The development employed the Golang programming language due to its robustness and performance. The VNF application implementing the Dot1x authenticator function receives authentication requests from supplicants (ONUs in the real world), consults the supplicants' credentials against a RADIUS server, and authorizes or not their access to the network. The proposal was leveraged by containers hosted into Kubernetes and Docker environments hosting the main elements and testing scalability scenarios with multiple supplicants. The tests results proved the security VNF feasibility with acceptable performance, with microseconds processing time handling thousands of messages per second, and provided insights into the most promising scenarios for VNF deployment.

To fully validate the solution, future tests will consider a more robust and complex Dot1x authentication method, along with higher-scale scenarios, as it is expected that one OLT could support thousands of attached ONU/ONT. The most promising test scenarios will be deployed in real OLT devices, allowing also to evaluate the add-value of the VNF approach facing traditional OLT implementation. As final remark, since the present deployment was aligned with the BBF framework, the VNF can be easily integrated with other OB-BAA components.

Acknowledgements This work has been supported by FCT - Fundação para a Ciência e a Tecnologia, within the R&D Units Project Scope: UIDB/00319/2020.

Author Contributions All authors contributed equally to the study and preparation of the manuscript.

Declarations Ethical, conflict of interest, data and code availability issues are not applicable.

References

[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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Tommaso Muciaccia, Fabio Gargano, and Vittorio M.N. Passaro. "Passive optical access networks: State of the art and future evolution". In: Photonics (2014). 1(4), pp. 323-346. DOI: 10.3390/photonics1040323.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] Bradai, A., Rehmani, M.H., Haque, I. et al. Software-Defined Networking (SDN) and Network Function Virtualization (NFV) for a Hyperconnected World: Challenges, Applications, and Major Advancements. In: Journal of Network and Systems Management, 28, 433–435 (2020). DOI: 10.1007/s10922-020-09542-z

[15] Abel Tong and Kevin Wade. "Guia de NFV e SDN para operadoras e provedores de serviços". In: Ciena (2017), p. 36. URL: https://media.ciena.com/documents/Blue+Planet+Essentials_NFV+and+SDN+Guide_033017_A5_pt_BR.pdf

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

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

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

[19] Broadband Forum. vOMCI Proxy and vOMCI Function. URL: https://obbaa.broadbandforum.org/architecture/vomcipf/ #vomcipf.

[20] Open Network Foundation. ONF Broadband Projects. URL: https://opennetworking.org/onf-broadband-projects/

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

[22] Open Network Foundation. SEBA (SDN Enabled Broadband Access) Introduction. https://wiki.opennetworking.org/display/COM/SEBA.

[23] Open Network Foundation. VOLTHA (Virtual OLT Hardware Abstraction) Introduction. https://wiki.opencord.org/display/CORD/VOLTHA.

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

[25] John Vollbrecht et al. Extensible Authentication Protocol (EAP). RFC 3748. June 2004. DOI:10.17487/RFC3748.

[26] 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.

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

[28] 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/.

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

[30] Igor Araújo, Solange Rito Lima and André Brízido, "IEEE 802.1X Virtual Network Function Development for NG-PON Architecture," 2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2022, pp. 1-6, DOI: 10.23919/SoftCOM55329.2022.9911396.

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

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

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

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

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

[36] BroadBand Forum, Open BroadBand - BroadBand Access Abstraction Github repository. URL: https://github.com/BroadbandForum/obbaa

[37] Helm - The package manager for Kubernetes. URL: https://helm.sh/.