1 2 9 0

UNIVERSIDADE D
COIMBRA

Ricardo Mendes Figueiredo

VIRTUAL BORDER NETWORK GATEWAY (vBNG)

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Paulo Simões and Professor Vasco Pereira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2023

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Ricardo Mendes Figueiredo

Virtual Border Network Gateway (vBNG)

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Prof. Paulo Simões and Prof. Vasco Pereira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September of 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Departamento de Engenharia Informática

Ricardo Mendes Figueiredo

Virtual Border Network Gateway (vBNG)

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Paulo Simões e Professor Vasco Pereira apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro of 2023

Acknowledgements

This work is funded by the project POWER (grant number POCI-01-0247-FEDER-070365), co-financed by the European Regional Development Fund (FEDER), through Portugal 2020 (PT2020), and by the Competitiveness and Internationalization Operational Programme (COMPETE 2020). It was also partially supported by FEDER, via the Competitiveness and Internationalization Operational Program (COMPETE 2020) of the Portugal 2020 framework, in the scope of Project Smart5Grid (POCI-01-0247-FEDER-047226).

Gostaria de agradecer ao Professor Doutor Paulo Simões e Professor Doutor Vasco Pereira, os meus orientadores, pelos conhecimentos e competencias que me transmitiram ao longo da elaboração deste trabalho.

Quero também agradecer ao Professor Doutor Tiago Cruz e ao Mestre Jorge Proença pela disponibilidade em tirar duvidas quando inevitavelmente alguma coisa corria mal.

Quero dar um especial agradecimento aos meus pais e às minhas irmãs, por me terem dado esta oportunidade, por terem feito todos os possíveis para que eu possa ter um futuro promissor, e por sempre me terem deixado espaço para seguir o meu próprio caminho e por terem feito o seu melhor para que eu me tornasse alguém em quem me possa orgulhar.

Por último, mas não menos importante, queria agradecer à minha namorada, Raquel Ferreira, por ainda me aturar até hoje, por acreditar em mim mesmo quando eu não o fazia, por sempre estar lá para me apoiar quer com carinho ou com a franqueza que eu merecia. Por ser a mulher mais forte que conheço, e por me inspirar a ser uma pessoa melhor. Tenho muito orgulho em ti, e obrigado por seres a pessoa incrível que és.

Cofinanciado por:

Abstract

This document reports the work done in the author's Master's dissertation in Informatics Engineering, specialization in Software Engineering, of the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

The increased demand for high-speed broadband network connectivity has been creating issues for network service providers as their systems, due to their inflexibility, struggle to keep up with the ever-more dynamic market. This in turn creates a scenario where network service providers require ever-more resilient and flexible solutions for network infrastructures at lower prices, putting pressure on hardware manufacturers. This scenario also creates a demand for highly qualified personnel, capable of designing and operating such systems. This along with the competitive market, are just a few of the factors that lead Altice Labs to create the **POWER** project, in which this internship is contextualized.

The goal of this work is to design and validate a Virtual Border Network Gateway (vBNG) proof-of-concept supported by Network Function Virtualization (NFV) and Software-Defined Networking (SDN).

At the end of this internship the objectives were met successfully, with a prototype for the vBNG having been developed. Future work concerns the iteration of this prototype, implementing more of the Border Network Gateway (BNG) functionality and further exploring the tools' features to empower the vBNG.

In this document, all phases of work done during this process are described, including familiarization with the technologies, the state of the art which includes possible approaches discovered, the study of requirements, architecture design, implementation and validation of the prototype.

Keywords

Border Network Gateway, Broadband Remote Access Server, Software Defined Networking, Network Function Virtualization, Virtual Network Function

Resumo

Este documento relata o trabalho realizado pelo estagiário no estágio de Mestrado em Engenharia Informática, especialização em Engenharia de Software, do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

A crescente necessidade de conectividade de rede de banda larga de alta velocidade tem criado problemas para as operadoras de serviços de rede, uma vez os seus seus sistemas, devido à sua inflexibilidade, têm dificuldades em acompanhar um mercado cada vez mais dinâmico. Isso, por sua vez, cria um cenário em que as operadoras de serviços de rede exigem soluções cada vez mais resilientes e flexíveis para as suas infraestruturas de rede a preços mais baixos, pressionando assim os fabricantes de hardware. Este cenário também gera uma necessidade por pessoal altamente qualificado, capaz de desenhar e operar tais sistemas. Estes, juntamente com o mercado competitivo, são apenas alguns dos fatores que levaram a *Altice Labs* a criar o projeto **POWER**, no qual este estágio se enquadra.

O objetivo deste trabalho é projetar e validar uma prova de conceito de um Virtual Border Network Gateway (vBNG), suportado por Network Function Virtualization (NFV) e Software-Defined Networking (SDN).

No final deste estágio foram cumpridos com sucesso os objetivos, tendo sido desenvolvido um protótipo do vBNG. O trabalho futuro prende-se com a iteração deste protótipo, implementando mais funcionalidades do Border Network Gateway (BNG), e explorando mais das funcionalidades das ferramentas utilizadas para melhorar o vBNG.

Neste documento serão descritas todas as fases do trabalho realizadas ao longo deste processo, desde a familiarização com as tecnologias, ao estado da arte que inclui algumas abordagens descobertas, ao estudo de requisitos, desenho da arquitetura, implementação e validação do protótipo.

Palavras-Chave

Border Network Gateway, Broadband Remote Access Server, Software Defined Networking, Network Function Virtualization, Virtual Network Function

Contents

Acronyms

AAA authentication, authorization and accounting.

ACL Access Control List.

API Application Programming Interface.

ASIC Application-Specific Integrated Circuit.

BNG Border Network Gateway.

BRAS Broadband Remote Access Server.

CGNAT Carrier-Grade Network Address Translation (NAT).

CH Could Have.

CLI Command Line Interface.

COTS Commercial Off-The-Shelf.

CPU Central Processing Unit.

DC Datacenter.

DHCP Dynamic Host Configuration Protocol.

DPI Deep Packet Inspection.

ETSI European Telecommunications Standards Institute.

FPGA Field-programmable gate array.

GB gigabyte.

Gbps gigabits per second.

GUI Graphical User Interface.

HAL Hardware Abstraction Layer.

IGMP Internet Group Management Protocol.

IPoE IP-over-Ethernet.

IPTV Internet Protocol Television.

ISP Internet Service Provider.

KVM Kernel-based Virtual Machine.

MANO Management and Orchestration.

Mbps megabits per second.

MH Must Have.

NAT Network Address Translation.

NFV Network Function Virtualization.

NFVI Network Function Virtualization (NFV) Infrastructure.

NFVO NFV Orchestrator.

NS Network Service.

NSD Network Service Descriptor.

NSH Network Service Headers.

ONAP Open Network Automation Platform.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OS Operating System.

OSM Open Source MANO.

OVS Open vSwitch.

P4 Programming Protocol-independent Packet Processors.

PBR Policy-Based Routing.

PoC Proof-of-Concept.

PPP Point-to-Point.

PPPoE Point-to-Point Protocol over Ethernet.

QoS Quality of Service.

SDN Software-Defined Networking.

SFC Service Function Chaining.

SFF Service Function Forwarder.

SH Should Have.

SNAT Source Network Address Translation.

SoA	State of the Art.
SP	Sub-project.
TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
vBNG	Virtual Border Network Gateway.
VDU	Virtual Deployment Unit.
VF	Virtual Function.
VIM	Virtualised Infrastructure Manager.
VM	Virtual Machine.
VNF	Virtual Network Function.
VNFD	VNF Descriptor.
VNFM	Virtual Network Function (VNF) Manager.
VoIP	Voice over Internet Protocol.
VPN	Virtual Private Network.

List of Figures

List of Tables

Chapter 1

Introduction

According to [Hu et al., 2020], a Border Network Gateway (BNG), or Broadband Remote Access Server (BRAS) "routes traffic to and from broadband remote access devices such as digital subscriber line access multiplexers (DSLAM) on an Internet Service Provider's (ISP) network". The Border Network Gateway (BNG) is an integral component as it provides customers an access point to connect to the broadband network, allowing them access to their provider's services, as figure 1.1 demonstrates. Its tasks include authentication, authorization and accounting (AAA), managing Point-to-Point Protocol over Ethernet (PPPoE) sessions, assigning IP addresses, and controlling Quality of Service (QoS) among others [Kundel et al., 2019].

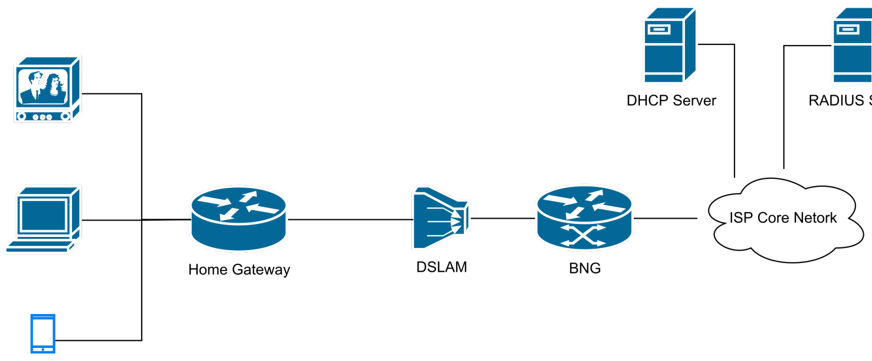


Figure 1.1: BNG placement in the network, connecting subscriber to the Internet Service Provider (ISP)'s core network, based on [Cisco, 2022]

Traditionally, the BNG is deployed in specialized proprietary hardware, similarly to other components of an operator infrastructure. This factor, coupled with its complexity increases the costs for the operators both for deployment and management. Since such devices are purpose-built, they cannot provide much in terms of flexibility either, and introducing new features is a long process, which in some cases can only be deployed when the vendor releases a new version of the appliance.

One solution for this problem is to virtualize the BNG. A Virtual Border Network Gateway (vBNG) could be deployed in Commercial Off-The-Shelf (COTS) hardware, reducing costs, and a disaggregated BNG approach could provide the desired flexibility and scalability. An approach in virtualizing the BNG is Network Function Virtualization (NFV) which combined with other technologies such as Software-Defined Networking (SDN) would allow for easier, more scalable, and cost-effective solutions. With this approach, the different functions of the BNG could be implemented as Virtual Network Functions (VNFs) and deployed on COTS hardware, and new instances could be deployed as the needs grow.

The main objective of the work here presented is to design a Proof-of-Concept (PoC) of such a vBNG, following the concepts of Network Function Virtualization (NFV) and Software-Defined Networking (SDN), implement it using available open-source tools when possible, and to validate the proposed solution.

1.1 Context

This internship falls within the scope of the research project **"POWER - Empowering a digital future"**, Sub-project (SP) 2 - Future Networks. This project aims to expand Altice Labs' portfolio of products and services with new innovative solutions, mostly based on cloud and cognitive technologies, towards an integrated offer concept, enabling new approaches to the market, fully aligned with four strong technological vectors of transformation: 5G networks, Edge/Cloud computing continuum, artificial intelligence, data-driven technologies, and business models. The project is divided into five sub-projects, SP2 being one of them.

The aim of SP2 is to support the evolution of the current Altice Labs' network solutions, by creating new approaches using emergent network technologies, increasing competitiveness in current markets, and being able to achieve new market segments. To this end SP2 identifies five dimensions of evolution, amongst them the one this internship falls within, the creation of new solutions for virtualized networks that take advantage of NFV and SDN technologies to reach new levels of operational efficiency and flexibility in the access network.

The virtualization of the BNG is one of the project goals. Being the main component that connects the subscriber to the ISP core network, its management is a complex task. As the number of subscribers grow, and new services are developed, it may require new configurations and software/hardware updates. Additionally, the deployment and update of a BNG must have minimal to no impact on the service, which frequently results in BNG updates being a long task to be completed. These requirements may be easier to meet using virtualized solutions, as they can improve flexibility on its deployment and management.

1.2 Goals

The main goal of this work is to design, implement and test a PoC for a vBNG using the concepts of SDN and NFV. The PoC will include only a set of the main functionality, as a complete BNG implementation would be too extensive for the length of this thesis. This subset of functions includes AAA, Dynamic Host Configuration Protocol (DHCP), and Network Address Translation (NAT), and will be deployed using an NFV platform, into a virtual network managed by SDN technology.

1.3 Document Structure

The document is organized as follows:

- **Chapter 2, Background** - In this chapter, the main concepts for this work, BNG, SDN and NFV are introduced and examples of software related to each topic are presented.
- **Chapter 3, State of the Art** - In this chapter, the role of each technology in the context of the vBNG is presented, as well as examples of vBNG solutions.
- **Chapter 4, Planning** - This chapter presents the work plan as well as the methodology used and the identified risks.
- **Chapter 5, Preliminary Work** - Here it is presented the work done during the first semester to analyse the behavior of Open Network Operating System (ONOS), when deployed in a geographically distributed scenario.
- **Chapter 6, Requirements** - In this chapter, the analysis of the requirements for the planned functionality is presented.
- **Chapter 7, System architecture** - This chapter presents the designed system architecture.
- **Chapter 8, Implementation** - This chapter details the prototype implementation.
- **Chapter 9, Testing** - This chapter details how the prototype was validated.
- **Chapter 10, Conclusion** - This chapter concludes the document and discusses future work.

1.4 Contributions

This thesis contributes with an analysis of the viability to implement a BNG using virtualized functions, a proposed solution, instructions on how to set it up and next steps that could be taken to improved the solution proposed. These results will also contribute to the project POWER deliverables.

The research efforts of this thesis also lead to the following contributions to open-source projects:

- **Open Source MANO (OSM) Documentation PR#117** - Fixed an error in the documentation. The page explaining how to deploy the example VNFs was linking to the wrong Virtual Machine (VM) images, causing errors when deploying, creating confusion in new users.
- **VyOS Phabricator Task T5418** - Presented an error in the documentation stating there is a limitation when configuring the PPPoE Server that does not exist.
- **VyOS Documentation PR#1052** - Fix the aforementioned error in documentation.

Chapter 2

Background

This chapter presents the background for the virtualization of the Border Network Gateway (BNG) by over-viewing three technologies, namely Software-Defined Networking (SDN), Network Function Virtualization (NFV) and Service Function Chaining (SFC).

Section 2.1 will give an overview of SDN and its components. Next, NFV will be introduced in section 2.2, followed by section 2.3 explaining SFC.

2.1 Software-Defined Networking

In the scope of networking, SDN is a paradigm of network design that eases network configuration and management, providing increased flexibility and resource scaling according to application and data needs, while also reducing costs [Benzekki et al., 2016].

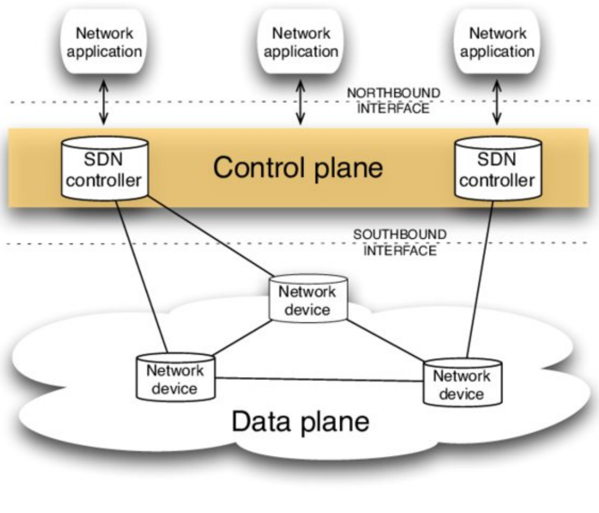


Figure 2.1: Simplified SDN architecture, from [Nencioni et al., 2017]

In figure 2.1, a simplified view of the SDN architecture can be observed. As stated by [Proença et al., 2019], SDN separates the control plane from the forwarding plane, which traditionally were tightly coupled in routers, and moves it to a logically centralized software-based controller [VMWare, 2022].

The data plane is described in section 2.1.1, and the control plane in section 2.1.2. Openflow, a popular protocol used for the communication between SDN controllers and forwarding devices is presented in section 2.1.3. Finally, in section 2.1.4, Open Network Operating System (ONOS), a SDN controller, is presented.

2.1.1 Data Plane

In an SDN network, the data plane consists of forwarding devices, responsible for moving the network packets. As an example, this can be done using OpenFlow-enabled switches (see subsection 2.1.3). These devices are responsible for orchestrating traffic forwarding according to the configuration established in the control plane. This orchestration is enabled by the Southbound interface, which connects the two planes, allowing forwarding devices to receive the forwarding rules from the controllers using the OpenFlow protocol.

2.1.2 Control Plane

The control plane is responsible for making decision on how the network behaves, such as defining forwarding rules used by the forwarding devices. It is composed of the following elements [Freitas et al., 2018]:

- **Controller** - Software stack responsible for managing almost every aspect of the network. The controller essentially can manage the network topology, access the state of each network device and manage flow rule installation in the forwarding devices. The controller is the critical component of a SDN network, which could create a scalability issue, in the case of the controller not being able to handle the load of managing so many devices [Benzekki et al., 2016]. A common technique used to mitigate this issue and therefore achieve scalability is clustering, in which multiple controller work in tandem, each managing a subset of devices, and propagating any change in the network to the rest of the cluster [Benzekki et al., 2016],[Berde et al., 2014]. Some examples of SDN controllers include ONOS, OpenDaylight, Ryu, and SDN-C.

- **Northbound Interfaces** - Application Programming Interfaces (APIs) connecting the control layer to the applications, typically through REST. These APIs allow applications, for example, to set network protocols or access network state through an abstraction layer.

- **East/Westbound interfaces** - Interfaces that connect each controller in a cluster, used for synchronization of the network state.

- **Southbound interfaces** - Interfaces through which the controller communicates with the forwarding devices, typically through the OpenFlow protocol

[Singh, 2023]. Used for tasks such as managing flow rules and the state of forwarding devices.

2.1.3 OpenFlow

OpenFlow is one of the main southbound communication protocols used in SDN [Lantz, 2017a]. This protocol specifies how SDN controllers and forwarding devices communicate.

Although it provides basic configuration capabilities, such as "...bringing a port up or down, or configuring meters..." [Lantz, 2017a], its primary use is to enable the installation of forwarding rules in the data plane by the controllers in the form of flow table entries. Flow table entries map packet attributes (source IP, destination IP, among others) to an action (such as forwarding the packet, dropping it, go to another table, among others) and are stored in flow tables. When a switch receives a packet, it performs a lookup in the first flow table and executes the action on the first rule matching the received packet. In case no such entry is found, a table-miss occurs, and the packet is processed by a table-miss flow entry. This entry may specify that the packet and the packet is to be sent to the controller, which in turn sends the missing flow table entries to the switch, allowing it to process the packet.

2.1.4 Open Network Operating System

The Open Network Operating System (ONOS) project [ONF, 2022] is an open-source SDN controller implementation. Its development was focused on creating a "distributed platform for scale-out and fault tolerance" [Berde et al., 2014]. To this end, ONOS was designed to be deployed in clusters, where multiple instances manage the same network, distributing the work amongst all instances.

ONOS is highly modular, exposing JAVA APIs, allowing development of applications to extend the ONOS's core functionality [Lantz, 2017b]. As an example, the application *Mastership Load Balancer* periodically balances the connected devices across all online controllers [opennetworkinglab, 2018]. These applications can be activated, deactivated, installed and uninstalled at runtime. REST APIs are also exposed but should not be used for high-performance applications since they are slower [Lantz, 2017b].

Lastly, ONOS provides a Command Line Interface (CLI) as well as a Graphical User Interface (GUI), the latter supporting extensions through applications, allowing for easy visualization and management of the network topology.

2.2 Network Function Virtualization

NFV is the concept of separating network functions usually coupled to proprietary hardware and deploying them on Commercial Off-The-Shelf (COTS) hardware as software instances, typically in the form of containers or Virtual Machines (VMs) [Yi et al., 2018]. The "increasing costs of energy, capital investment challenges and the rarity of skills necessary to design, integrate and operate increasingly complex hardware-based appliances" were some of the motivations for European Telecommunications Standards Institute (ETSI) to introduce this concept [ETSI, 2012]. Moreover, ETSI also pointed out that NFV could enable targeted and tailored services according to customer needs. Beyond these issues, the dependence on proprietary hardware also increases the time to market and limits innovation in the industry [Hawilo et al., 2014], while the flexibility provided by NFV allows for targeted and tailored services [Han et al., 2015].

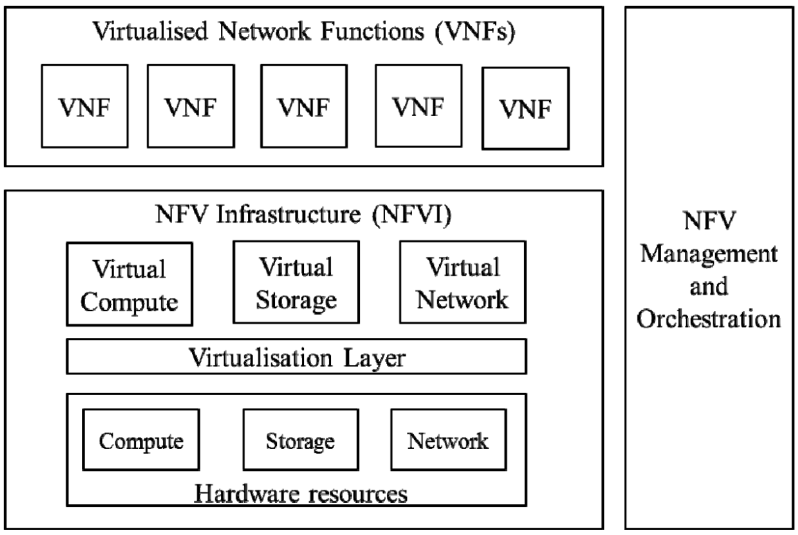


Figure 2.2: High level NFV Framework, [ETSI, 2014a]

The NFV framework, depicted in figure 2.2, is mainly composed of the following elements:

- Virtual Network Functions (VNFs), the software implementation of the functions to be virtualized.
- NFV Infrastructure (NFVI), composed of all the hardware infrastructure, the virtualization layer, and virtual infrastructure which provides the environment where the VNFs will be executed.
- Management and Orchestration (MANO), responsible for managing the resources available in the NFVI and the VNFs lifecycle, such as instantiating and terminating them.

NFV MANO can then be divided into its three main components: the Virtualised Infrastructure Manager (VIM), the VNF Managers (VNFMs), and the NFV Orchestrator (NFVO). The VIM orchestrates the NFVI within its domain, allowing multiple VIMs to exist, each with its own domain. It manages its resources, and keeps an inventory of VMs. The VNFs manages the lifecycle of the VNFs, for instance, instantiation and termination. The NFVO interfaces with the previously mentioned elements and is responsible for coordinating them.

In subsection 2.2.1, a few solutions for NFV orchestration are presented and section 2.2.2 presents the concept of catalogs in the context of NFV.

2.2.1 NFV Platforms

Over the years, the framework proposed by ETSI has been gaining more acceptance and is being implemented by a few prominent open-source projects. One example is OpenStack [OpenInfra, b], a platform started in early 2010 for managing distributed compute, storage, and network resources. It is based on a modular architecture, and in 2016 [OpenInfra, a] it introduced the Tacker [OpenInfra, c] component, the official OpenStack implementation of the ETSI MANO architecture, enabling the deployment and management of VNFs on an OpenStack network. Another project is Open Network Automation Platform (ONAP), started in 2017 and hosted by the Linux Foundation by merging the *OpenECOMP* and *Open-Orchestrator* projects. This project aims to develop a platform for "orchestration, management, and automation of network and edge computing services for network operators, cloud providers, and enterprises" [The Linux Foundation]. Finally, Open Source MANO (OSM), started around 2016, is an effort by ETSI to develop a MANO software stack aligned with ETSI NFV, built from the now defunct *Open MANO* project. Table 2.1 presents a summary of these projects.

Table 2.1: NFV MANO solutions

Product	**Function**	**Description**
Openstack	VIM	Software stack for cloud management.
Tacker	NFVO and VNFM	Official OpenStack's implementation of a generic VNFM and NFVO.
ONAP	NFVO and VNFM (supports third-party VNFMs)	MANO platform hosted by the Linux Foundation, adds extra functionality to complement the ETSI-NFV architecture.
OSM	NFVO and VNFM	MANO implementation hosted by ETSI, aligned with ETSI-NFV architecture.

2.2.2 Catalog

In the context of NFV, catalogs are essentially data repositories. In [ETSI, 2014b], ETSI defines the following:

- **VNF Catalogue**: Repository containing all loaded VNF packages, including images, manifest files and descriptors.
- **Network Service (NS) Catalogue**: Repository containing all NS templates and relevant data, such as descriptors, information regarding forwarding, and service chaining.

- **NFV Instances repository**: This repository contains information on all deployed VNFs and NSs. The records in this repository are updated to reflect the state of each instance.
- **NFVI Resources repository**: This repository contains information about the hardware and software resources that are available for use in a network, useful for resource reservation, allocation, and monitoring purposes.

2.3 Service Function Chaining

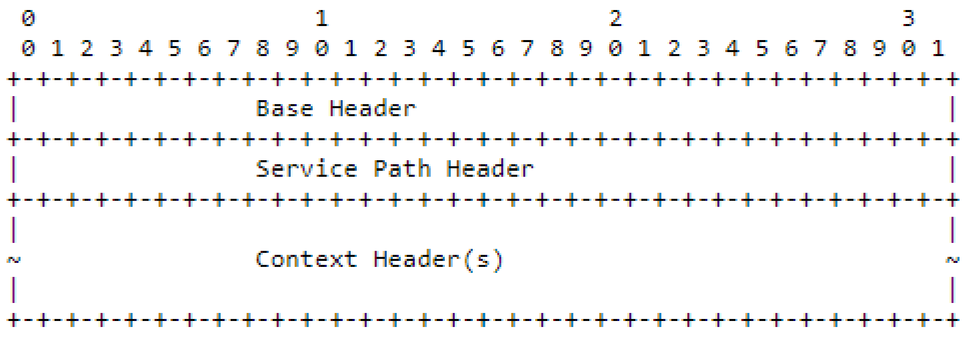
In a telecommunications network, traffic often passes through multiple functions, such as firewalls, Deep Packet Inspection (DPI), Network Address Translation (NAT), and load balancers, among others. Service Function Chaining (SFC) is a technique that leverages the capabilities of SDN to enable the chaining of multiple VNFs into a service chain. In the context of this work, SFC could allow each function of the BNG to be stored as independent VNF images which would then be chained together into a disaggregated Virtual Border Network Gateway (vBNG) service. As an example, the functions of Point-to-Point Protocol over Ethernet (PPPoE) server, NAT and firewall could be deployed as separate Virtual Functions (VFs), and chained together such that the packets sent from the clients are received by the PPPoE function, forwarded to the firewall function to inspect outgoing and incoming traffic and then sent through the NAT function that does the address translation.

In this section, the various elements of SFC are described. Section 2.3.1 provides an overview of Network Service Headers (NSH), the packet encapsulation used in SFC, and sections 2.3.2 and 2.3.3 will introduce the Classifier and Service Function Forwarder (SFF) components, respectively.

2.3.1 Network Service Header

The Network Service Headers (NSH) is a protocol for packet encapsulation, specified in RFC 8300 [Quinn et al., 2018]. It consists of a series of headers that provide information to VNFs and Service Function Forwarders (SFFs) about the service chain that the packet is part of. It addresses several limitations of SFC, documented in [Quinn and Nadeau, 2015], such as network services deployments being tied to the network topology, among others.

The NSH is composed of a Base Header, a Service Path Header, and optional Context Headers, as shown in Figure 2.3. The Service Path Header is composed of only 2 fields: the Service Path Identifier, a value set by the classifier which uniquely identifies the service function chain the packet belongs to, and the Service Index, which provides the location of the packet within a chain, meaning at which step of the chain the packet is to avoid a packet being processed by a VNF multiple times. The Base Header provides some information about the NSH headers and the payload. The optional Context Headers provide some metadata to be shared between the classifier and VNFs.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Base Header                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Service Path Header                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                Context Header(s)                              ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.3: Network Service Header, from RFC 8300 [Quinn et al., 2018]

2.3.2 Classifier

The classifier is an important component in SFC, as it matches packets to service chains using an Access Control List (ACL). If a match is found, it encapsulates them in NSH packets, sending them along the correct service chain. [OpenDayLight]

OpenStack's Neutron plugin includes the functionality of a SFC classifier. [OpenStack, 2022b]. Older versions of OpenDaylight controller also included an SFC classifier [OpenDayLight], but seems to have dropped it.

2.3.3 Service Function Forwarder

The SFF is a function responsible for forwarding traffic along the service chain. To accomplish this it uses the data in the NSH encapsulation. Examples of SFF implementations include OpenVSwitch and FD.io, a collection of projects aiming to provide high-performance networking solutions for virtualized infrastructure, providing a variety of components such as a virtual switch, a packet accelerator, and a service proxy.

Chapter 3

State of the Art

This chapter presents the NFV-enabled vBNG, and how each of the different technologies presented in the previous chapter is important to bring it to fruition. Moreover, it presents the State of the Art (SoA) on BNG virtualization, including current proposals and approaches for implementing it.

3.1 The NFV-enabled vBNG

In this document, a NFV-enabled vBNG is proposed. This section details how NFV, SDN and SFC technologies, which were previously presented in Chapter 2, could be leveraged to achieve flexibility and scalability.

3.1.1 The role of NFV

As described in section 2.2, NFV is the concept of separating the network functions from the underlying hardware. This would be the main role of NFV for the vBNG, to separate the BNG functions from the hardware, and deploy them as VNF instances. This approach would not only allow the vBNG to be deployed in COTS hardware (reducing costs), but also create a more scalable and flexible BNG solution as instantiating a new VNF instance is substantially easier than installing new hardware.

NFV MANO is a key component for the solution, as the task of managing such a system must be highly automated and coordinated to be viable since there is a large number of functions that need to work together to make the vBNG a reality. MANO is responsible for instantiating and configuring the various vBNG functions, scaling them as needed, monitoring them, and orchestrating the SDN subsystem to ensure that the various VNFs composing the network service have connectivity.

3.1.2 The role of SDN

The role of SDN (see section 2.1) in the context of the vBNG is simpler to explain than NFV but just as important, it provides the connectivity to the various VNFs comprising the solution. By separating the control and data planes, and creating a logically centralized network control and management, SDN provides a way for NFV to configure the network according to the service's needs. This enables the vBNG to have its resources scaled according to the network traffic. The controllers can also be used to manage Quality of Service (QoS) and traffic shaping through OpenFlow rules they install in the data plane.

3.1.3 The role of SFC

In the context of vBNG, SFC (see section 2.3) could provide a mechanism to steer the traffic through the various VNFs. The NSH headers provide NFV and SDN a way to identify which network service a packet belongs to, and to forward it through the service chain.

3.2 Approaches

The following sections will provide an overview of different approaches when implementing a vBNG.

3.2.1 ClickOS

In [Bifulco et al., 2013] and [Dietz et al., 2015] the authors propose the implementation of a vBNG in ClickOS [NEC Laboratories Europe]. ClickOS is a lightweight VM based on the click modular router [Kohler]. ClickOS is made with the purpose of developing virtual middleboxes using it. "ClickOS virtual machines are small (5MB), boot quickly (about 30 milliseconds), add little delay (45 microseconds) and over one hundred of them can be concurrently run while saturating a 10Gb pipe on a commodity server."[NEC Laboratories Europe]. ClickOS is also very modular, allowing users to effortlessly connect the over 300 stock elements [NEC Laboratories Europe], or create their own for their specific use case. In this approach, all functionality would be implemented as a single VM, although the authors mention it would be possible to migrate some functionality to other devices in case the workload was too big. However, it is not clear how this migration would happen. This approach also does not appear to be integrated into any NFV platform, but considering it runs on top of the XEN hypervisor, a hypervisor supported by OpenStack [OpenStack, 2019], integration should be possible.

3.2.2 BNG-HAL

In [Figueiredo and Kassler, 2021] the authors propose an Hardware Abstraction Layer (HAL) for a disaggregated BNG, based on the architecture presented in [Forum, 2020]. This architecture applies to a BNG the same principles of abstracting control and data planes, in order to achieve a more scalable and easier to configure solution. Mirroring SDN, TR-459 separates the control functions and data forwarding into two separate components. In [Figueiredo and Kassler, 2021] the authors suggest the introduction of a HAL with the goal of standardizing the way the different components communicate.

3.2.3 ONOS

A vBNG application for ONOS was presented in [Lin and Hart, 2015]. Its main functionality is providing clients with private IP addresses and installing intents in the SDN network to allow internet access to the hosts. This solution is presented for the sake of exemplifying a BNG implemented as an SDN controller application and will not be taken into consideration for the work of this thesis, as at the time of writing ONOS is being migrated into a new architecture, and XOS (one of the application's dependencies) is part of CORD, an Open Networking Foundation (ONF) project, that has been deprecated [Thengvall and Indermitte, 2021a].

3.2.4 VyOS

VyOS [Thengvall and Indermitte, 2021b] is an open-source network operating system based on Debian. The project started in 2013 as a fork of Vyatta, by a community of developers, being actively supported at the time of writing. The project boasts features relevant to a vBNG, including PPPoE subscriber session management, IP-over-Ethernet (IPoE) subscriber session management, subscriber authentication using a RADIUS, QoS, firewall, Virtual Private Network (VPN) and NAT.

3.2.5 Open Network Automation Platform

ONAP [The Linux Foundation] is an open-source platform for automating, orchestrating and managing VNFs and services. ONAP can be used to create a disaggregated vBNG by automating the deployment and management of VNFs that provide vBNG functions. ONAP also provides tools for chaining different VNFs, allowing the vBNG to be easily tailored according to the requirements. To this end, the ONAP project includes a SDN controller based on OpenDaylight, SDN-C, not requiring any external controller to function.

3.2.6 Open Source MANO

OSM [ETSI] is an open-source project to develop a NFV MANO software stack, more aligned with ETSI NFV than ONAP. Similarly to ONAP, it can be used to create a disaggregated vBNG by automating the deployment and management of VNFs. It also provides tools for chaining VNFs. However, it does not provide a SDN controller, relying on external ones, such as ONOS.

3.2.7 P4

In [Osiński et al., 2020] and [Kundel et al., 2019] the authors present P4-vBNG and P4-BNG respectively. These are two implementations of the vBNG's data plane implemented using the Programming Protocol-independent Packet Processors (P4) [ONF] programming language. The P4 language is a high-level domain-specific language, designed for a specific set of tasks [Fowler, 2019], with the purpose of specifying how network devices should process packets. It allows programmers to specify behaviors independent of the underlying hardware.

In these proposals, the authors use programmable Application-Specific Integrated Circuits (ASICs) or Field-programmable gate arrays (FPGAs), a specialized microchip and a microchip consisting of a matrix of logic blocks that can be configured by the user to implement custom logic designs [Xilinx] respectively. These are used to offload data plane packet processing to specialized hardware, while allowing the control plane to be deployed in COTS hardware, similar in principle to SDN. Such an approach would allow greater performance than running all functions of the vBNG on COTS hardware, as shown by the authors in [Kundel et al., 2019] by comparing the solution proposed to a standard x86 Linux server and showing greater results.

In [Kundel et al., 2021] the authors present OpenBNG, an extension of P4-BNG [Kundel et al., 2019] the authors present a P4 BNG data plane implementation and a queueing chip based on FPGA. In this research paper, the author suggests adding an additional FPGA to the previous solution, for packet queueing, as P4 is not designed for this use case.

3.3 Chapter Summary

This chapter started by exploring the role of SDN, NFV and SFC in the implementation of the vBNG. It then presented solutions for the implementation of the proof-of-concept, both implementations of the VF (such as ClickOS and VyOS) and supporting components such as MANO (OSM and ONAP). The approaches are summarized in table 3.1.

Table 3.1: Comparison of BNG Virtualization Approaches

Proposal	**Type**	**Status**	**Notes**
ClickOS	Proof-of-concept study	Released	No longer under active development
BNG-HAL	Architecture proposal	Proposed architecture	Architecture only, no implementation available
ONOS	Open-source network OS	vBNG support deprecated	Currently migrating to new architecture
VyOS	Open-source network OS	Under active development	Provides the necessary functionality
ONAP	Open-source NFV Project	Under active development	More powerful, but much higher resource requirements and harder to setup
OSM	Open-source NFV Project	Under active development	Easier to deploy and setup, provides the necessary functionalities, but relies on external VIM
P4-vBNG	Research paper on deploying a P4 vBNG data plane in an ASIC	Released	Provides better performance than linux based solutions but relies on external control plane and requires specialised hardware
P4-BNG	Research paper on deploying a P4 vBNG data plane in an ASIC	Released, including implementation source-code	
OpenBNG	Extension of P4-BNG that enhances the solution using an additional FPGA	Released	

Chapter 4

Planning

In this chapter, the planning of the internship is presented. This includes the methodology, the work plan for both the first and second semesters, and risks that might impact the plan.

4.1 Methodology

For this project, an Agile project management methodology was deemed the most fitting, as it allows incremental development and better adaptability to the project's progress compared to other methodologies such as Waterfall [Chaudhary]. With this in mind, a simplified version of Scrum [Schwaber and Sutherland, 2020] was chosen. The following are some of the rules defined for the methodology:

- At the beginning of the project, the tasks were defined.
- Two week long *Sprints* or development cycles.
- Biweekly meetings with the internship advisors, Prof. Paulo Simões and Prof. Vasco Pereira, where the progress was reviewed and the tasks for the next sprint were defined.
- Biweekly meetings with Altice Labs to report on the project's progress.
- Frequent contact was maintained between the author and the advisors to allow quick adaptation to issues.
- In December, the meetings with the advisors were changed from biweekly to weekly.

To assist the management, a Google Drive folder was shared between the author and advisors, where the milestones and meeting logs were kept.

4.2 Timeline

In this section, the estimations for the various tasks are presented, as well as the actual time taken for each task during this work. The identified tasks are the following:

- **Task 1** - Introduction to the project, terminology and themes
- **Task 2** - ONOS test definition
- **Task 3** - Setting up the scenario for testing, including the time to get familiar with the tools and various iterations over it.
- **Task 4** - Building the tools to gather and analyze the results from the tests, refining them, and using them.
- **Task 5** - Familiarization with the state of the art
- **Task 6** - Requirement Elicitation
- **Task 7** - Preliminary System Architecture
- **Task 8** - Intermediate Report Writing
- **Task 9** - Architecture and Requirements Refinement
- **Task 10** - vBNG test plan definition
- **Task 11** - Exploratory Work
- **Task 12** - Integration of VyOS into the platform
- **Task 13** - Configuration of RADIUS server and VyOS
- **Task 14** - Migration of RADIUS to SQL database
- **Task 15** - Testbed setup, performance tests
- **Task 16** - Dissertation and Technical Report writing
- **Task 17** - Troubleshooting, setup of second testbed and second round of performance tests

The Exploratory Work (Task 11) consisted of further researching the tools identified in section 3.3, as there were some lingering questions at the beginning of the second semester, regarding SFC support and viable approaches. Once these questions were closed, meaning the most viable approach was identified, the tools were selected (section 8.1), and experimentation with deploying VNFs started.

Figure 4.1 presents the plan created at the beginning of the second semester, while figure 4.2 presents the actual work done. Some clear delays are visible, that unfortunately hindered the ability to fully test the proposed solution. The main cause of delays were the following:

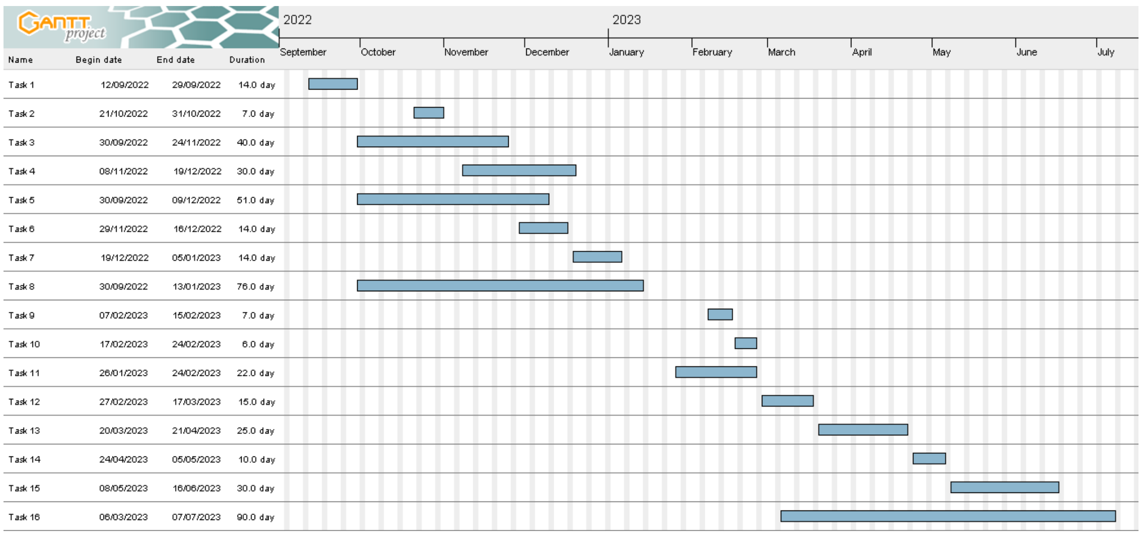
- Connecting VNFs with the exterior - By default, MicroStack sets an Open vSwitch (OVS) bridge that grants connectivity between the VNFs and the exterior. Due to the vBNG acting as a gateway a second bridge was necessary to isolate the traffic between the access network and the Internet Service Provider (ISP) network, however this process is not well documented, with the author resorting to adapting MicroStack's configuration scripts [Giessen et al., 2020] to achieve this.

- Integrating VyOS into the platform - While simply deploying VyOS to the platform is a straight-forward process, to take advantage of the the OSM and MicroStack's functions, cloud-init support was necessary to enable automatic configuration during initial startup. Due to the author having no prior knowledge of the tool, this process took more time than expected.

- PPPoE and cloud-init - VyOS and cloud-init handle configuration changes differently, the former being able to commit multiple changes at the same time while the latter handling one command at the time, this cause issues configuring the PPPoE server. This issue was exacerbated by the fact that it would cause VyOS's configuration to fail and locking the author out of the machine, unable to access the log files to troubleshoot the problem. While a workaround was eventually found, it took more time than initially planned.

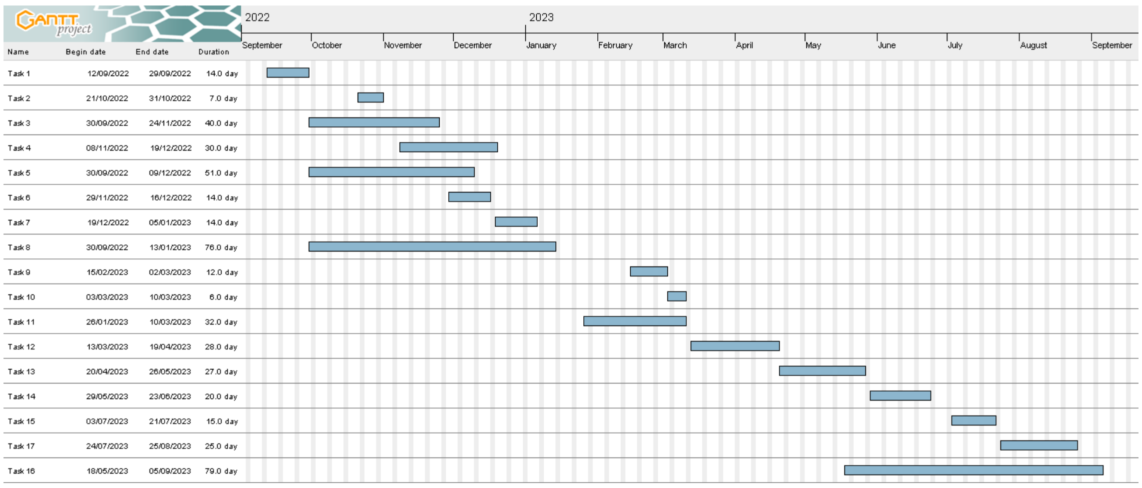
- Low performance - Due to several factors, including the use of nested virtualization and limitation in hardware virtualization, the initial performance of the prototype was much worse than expected, requiring a new testbed to be setup to mitigate those issues, as well as new round of tests to be executed.

The Gantt charts were generated using [GanttProject], a larger version of these figures is available in Attachment A.



Name	Begin date	End date	Duration
Task 1	12/09/2022	29/09/2022	14.0 day
Task 2	21/10/2022	31/10/2022	7.0 day
Task 3	30/09/2022	24/11/2022	40.0 day
Task 4	08/11/2022	19/12/2022	30.0 day
Task 5	30/09/2022	09/12/2022	51.0 day
Task 6	29/11/2022	16/12/2022	14.0 day
Task 7	19/12/2022	05/01/2023	14.0 day
Task 8	30/09/2022	13/01/2023	76.0 day
Task 9	07/02/2023	15/02/2023	7.0 day
Task 10	17/02/2023	24/02/2023	6.0 day
Task 11	26/01/2023	24/02/2023	22.0 day
Task 12	27/02/2023	17/03/2023	15.0 day
Task 13	20/03/2023	21/04/2023	25.0 day
Task 14	24/04/2023	05/05/2023	10.0 day
Task 15	08/05/2023	16/06/2023	30.0 day
Task 16	06/03/2023	07/07/2023	90.0 day

Figure 4.1: Planned Work Timeline



Name	Begin date	End date	Duration
Task 1	12/09/2022	29/09/2022	14.0 day
Task 2	21/10/2022	31/10/2022	7.0 day
Task 3	30/09/2022	24/11/2022	40.0 day
Task 4	08/11/2022	19/12/2022	30.0 day
Task 5	30/09/2022	09/12/2022	51.0 day
Task 6	29/11/2022	16/12/2022	14.0 day
Task 7	19/12/2022	05/01/2023	14.0 day
Task 8	30/09/2022	13/01/2023	76.0 day
Task 9	15/02/2023	02/03/2023	12.0 day
Task 10	03/03/2023	10/03/2023	6.0 day
Task 11	26/01/2023	10/03/2023	32.0 day
Task 12	13/03/2023	19/04/2023	28.0 day
Task 13	20/04/2023	26/05/2023	27.0 day
Task 14	29/05/2023	23/06/2023	20.0 day
Task 15	03/07/2023	21/07/2023	15.0 day
Task 17	24/07/2023	25/08/2023	25.0 day
Task 16	18/05/2023	05/09/2023	79.0 day

Figure 4.2: Real Work Timeline

4.3 Risks

The table 4.1 presents the identified risks for the project. Each risk is assigned a value for likelihood, impact, and severity. Likelihood and Impact are scaled between 1 (Low) and 5 (High). Severity is obtained by the product of Likelihood and Impact.

Table 4.1: Risks

ID	Risk Description	Likelihood	Impact	Severity	Mitigation Plan
1	Lacking documentation for the chosen tools.	1	5	5	Verify before choosing a tool if it is well supported and the community is active and helpful.
2	Bad understanding of the project's goals. Could result in badly defined requirements and architecture.	3	5	15	Biweekly meetings where the author's questions are exposed.
3	Project goals be too complex to be achievable in the specified timeframe.	3	5	15	Scale down the requirements to a manageable point.
4	System architecture does not cover all requirements.	1	5	5	Refine the architecture.
5	Author is not familiar with the technologies, causing a bad understanding of the objectives and a bad proposed solution.	3	5	15	Have a period of familiarization with the technologies to be used in the project.

6	Tools not compatible with each other.	3	5	15	Have an exploration phase in the beginning of the second semester where the compatibility between the tools is analysed.

Chapter 5

Preliminary Work

In this chapter, the work done to assess how ONOS behaves in a geographically distributed scenario is presented. Section 5.1 presents the test scenario definition. Section 5.2 presents an overview of how the testbed was set up. In section 5.3 the specifications for the various tests are presented. Section 5.4 presents the results obtained. Section 5.5 showcases various difficulties the author was presented with during the execution of these activities. Finally, section 5.6 contains a discussion of the activities performed.

The following tests involving ONOS were performed in collaboration with Altice Labs, as part of an exploratory work regarding ONOS behavior when deployed in a geographically distributed environment.

5.1 Test scenario

As the goal of the tests was to understand how ONOS behaves in a geographically distributed scenario, where the connections may be less than ideal, the testbed must support the simulation of such an environment. As a result, the testbed presents the following requirements:

- The testbed must simulate at least three regions.
- The SDN controller must be distributed across multiple regions.
- Must provide a way to manually define the parameters of the connections between the different regions.
- Each region must have multiple controller nodes to test fault tolerance.

Given these requirements, the scenario simulates three geographically distributed regions: Aveiro, Coimbra, and Faro. Each of these regions contains three nodes (nine in total). Each node is equipped with an Atomix instance to enable clustering, an ONOS instance, and an OVS instance. OVS is an open-source OpenFlow-enabled virtual switch and as such able to be integrated into a SDN network.

The tests were conducted using ONOS version 2.7.0 with a hotfix made by the author to stop it from crashing when all controllers in a region are shut down, Atomix version 3.1.5, and finally OVS version 2.9, deployed through Mininet [Mininet] version 2.3.0.

A transparent bridge connects the different regions, providing traffic shaping capabilities through *iproute2* [The Linux Foundation, 2022]. To automate the tests, *iproute2* is controlled using a script with pre-defined profiles for different latency values.

Figure 5.1 demonstrates the scenario topology.

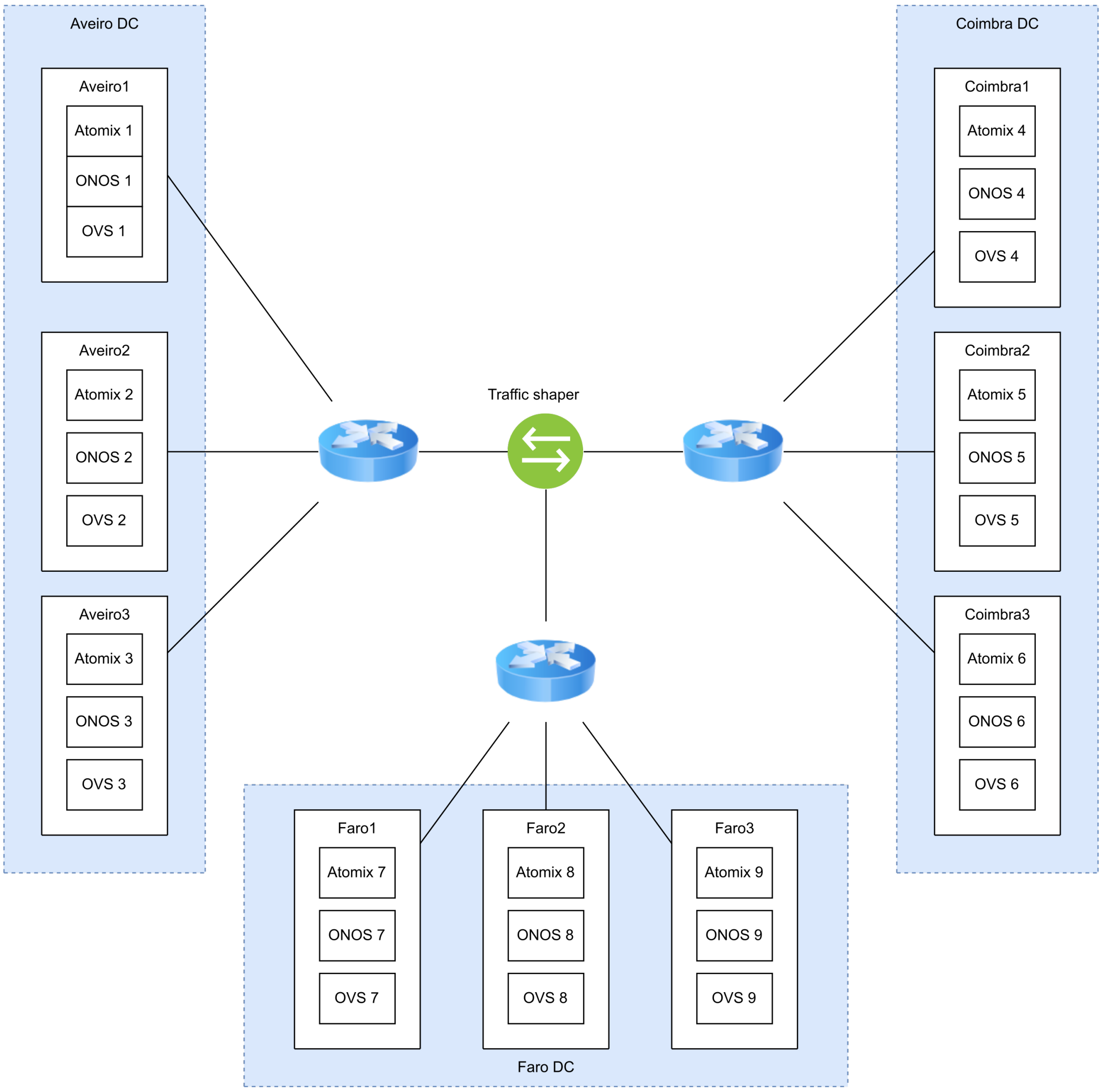


Figure 5.1: Simplified view of the Datacenters (DCs)

5.2 Testbed Setup

This section presents the setup of the testbed for the tests (described in section 5.3)

5.2.1 Datacenter Node

Each DC node is a VM running Ubuntu 22.04 LTS, running the ONOS, OVS and Atomix instances in Docker [Docker] containers.

ONOS Cluster

The initial configuration files were obtained by following the guide provided by the ONOS team to automate cluster creation [Zaballa, 2020]. These configuration files were then modified to include the relevant IP addresses.

The cluster is deployed with the official Atomix Docker image, version 3.1.5 and a modified version of ONOS version 2.7, the latest version at the time of writing, had a bug causing a *NullPointerException* to occur in case all nodes in a region go offline. As a result, it was impossible to test the scenario, as part of the test's objective is to understand how ONOS chooses which controllers to master the newly-orphaned switches (see section 5.5).

Open vSwitch

To deploy the OVS instances a custom Docker image was created, built from the Mininet [Mininet] Docker image [iwaseyusuke, 2022]. The main difference between the custom and the original images is that the custom runs Mininet at startup, with the topology provided in the shared volume. The topology used in each instance is a simple OVS switch that connects to all 9 controllers, with two hosts connecting to the switch, for debugging purposes. It should also be noted that for these tests, OpenFlow version 1.3 was specified in OVS configuration instead of the default version 1.5, this is due to while both OVS and ONOS support version 1.5 some discrepancy exists between the two, causing message parsing errors in ONOS.

5.2.2 Traffic Shaper

The Traffic Shaper bridge is a Kali Linux 2021.4 VM connecting the different DC sites. It is set up with four network interfaces, one for each DC plus a management interface, which are then bridged using a simplified version of the script in [Backer, 2016]. The version used, instead of parsing a file to load configurations, has them hardcoded, which can then be chosen by a simple argument.

5.2.3 Docker Registry

To ease deployment of the custom Docker images to the various DC nodes, a new server was deployed running the official Docker registry image [Team, 2022], where the images are stored. This server is connected to the DC nodes through the management interfaces, so traffic between these machines is not forwarded through the bridge, meaning that no shaping to this traffic happens.

5.3 Test Specification

This section presents the specification for the conducted tests. In order to avoid repetition, a few notes should be kept in mind:

- All tests were conducted in 4 variations, each one introducing different amounts of latency between the three different regions to simulate a geographically distributed scenario, where the connection properties are not ideal. These variations are (in milliseconds): 0, 250, 500, and 750.
- Unless explicitly stated otherwise, the latency values are set for all links connecting the different regions.
- Each variation of the tests is executed with 30 repetitions unless explicitly stated otherwise.
- The term *load* will be used to describe the number of devices connected to a controller, as in ONOS's Wiki [Vachuska, 2019].
- The term *Default network configurations* will be used to describe a configuration where no latency is artificially added, functionally identical to the 0-millisecond variation of the tests.
- The outputs of each test are part of every ONOS, OVS, and Docker instance's logs, which are collected at the end of each iteration, starting at a specific moment specified in each test. These logs are then parsed using *gawk* [arnold, 2022] to a normalized format to facilitate analysis.
- At each step the times are relative to the previous step.

5.3.1 Test 1 - Stable scenario

For this test the objective is to assess the stability of the masterships within one region when the scenario is stable, meaning every controller, switch, and Atomix instance is online, and the controllers' loads are balanced. This test, unlike the others, is not executed with 30 repetitions, only 1 per variation.

The steps for this test are as follows:

1. Testbed is started, with default network configurations.
2. After 3 minutes, the iteration's latency is applied.
3. The logs are saved from this point onward.
4. After 3 hours, the test is stopped.

5.3.2 Test 2 - Controllers failure in one region

The objective of this test is to assess how the cluster behaves when all controllers of a region are shut down.

The steps for this test are as follows:

1. Testbed is started, with default network configurations.
2. After 3 minutes, the iteration's latency is applied.
3. The logs are saved from this point onward.
4. After 15 seconds, all ONOS instances deployed in Aveiro are shut down.
5. After 5 minutes, the test is stopped.

5.3.3 Test 3 - Latency tolerance awareness

The objective of this test is to check the behavior of device mastership when the link quality degrades. For this test, the Coimbra DC will not have ONOS instances available, and its OVS are controlled by Aveiro's ONOS instances. The link between Coimbra and Aveiro will have its latency values increased to check how the device mastership behaves.

The steps for this test are as follows:

1. Testbed is started, with default network configurations.
2. After 1 minute, ONOS instances in Coimbra are shutdown, and the Coimbra's OVS instances' masterships are set to Aveiro's ONOS instances 1, 2 and 3.
3. The logs are saved from this point onward.
4. After 5 minutes, the iteration's latency is applied only to the link connecting Coimbra and Aveiro.
5. After 5 minutes, the test is stopped.

5.3.4 Test 4 - Latency tolerance awareness (cont.)

This test is similar to test 3 but with 2 key differences: the latency is changed on the link connecting Aveiro and Faro and Coimbra OVS instances are moved to Faro and not Aveiro.

The steps for this test are as follows:

1. Testbed is started, with default network configurations.

2. After 1 minute, ONOS instances in Coimbra are shutdown, and the Coimbra's OVS instances' masterships are set to Faro's ONOS instances 1, 2 and 3.
3. The logs are saved from this point onward.
4. After 5 minutes, the iteration's latency is applied only to the link connecting Faro and Aveiro.
5. After 5 minutes, the test is stopped.

5.3.5 Test 5 - Shutting down two ONOS instances from a region

This test is a variation of Test 2 (see section 5.3.2). Its main objective is to assess how the cluster behaves when two ONOS instances from one region are shut down, either by changing the OVSs' mastership to the remaining ONOS instance or to another region.

The steps for this test are as follows:

1. Testbed is started, with default network configurations.
2. After 3 minutes, the iteration's latency is applied only to the link connecting Coimbra and Aveiro.
3. The logs are saved from this point onward.
4. After 15 seconds, two ONOS instances deployed in Aveiro are shutdown (instances 1 and 2).
5. After 5 minutes, the test is stopped.

5.3.6 Test 6 - Shutting down two ONOS instances from a region (cont.)

This test is a variation of test 5. In this test, the *Mastership Load Balancer* app was disabled in ONOS. The goal of this test is to verify that the behavior observed in the result of test 5 (see section 5.4.5) is caused by the app. To accommodate this change a manual balance is made so that both tests start in a balanced state.

The steps for this test are as follows:

1. Testbed is started, with default network configurations.
2. After 3 minutes, the iteration's latency is applied only to the link connecting Coimbra and Aveiro.
3. A load balance is triggered.
4. The logs are saved from this point onward.

5. After 15 seconds, two ONOS instances deployed in Aveiro are shutdown (instances 1 and 2).

6. After 5 minutes, the test is stopped.

5.4 Results

In this section, the results obtained from the tests specified are presented, as well as some notes and hypotheses based on them.

5.4.1 Test 1 - Stable scenario

At 0 and 250 ms of latency introduced, nothing worth noting is observed, the scenario stays in a stable state with no changes to the network.

At 500 ms of latency introduced, the scenario shows some unexpected behaviors. Because the aveiro1 controller finished booting first it ended up mastering the 9 switches in the beginning. Once the other controllers finished booting, aveiro1's ONOS balanced the scenario, but it is clear it is not following the regional rules (it sends OVS instanced to regions other than their own, for instance sending coimbra1's OVS instance to the Faro region). This initial erroneous balancing cascades into 2:30 minutes of the scenario trying to balance itself while at the same time trying to move the switches to the correct region. After this period the scenario stabilizes and no further events of interest are observed.

At 750 ms of latency introduced, the scenario was not able to stabilize during the 3 hours of simulation. At multiple points in the simulation, it is observable OVS instances having their mastership changed to other regions despite no loss of connection with the current master being registered. This phenomenon also appears at moments when ONOS controllers in the same region are available and when the load is balanced between the various controllers. While no reason was discovered it is hypothesized this phenomenon occurs due to the high latency creating instability in the scenario. This phenomenon appears to be caused by the latency, as it's not observed, or at least not to this extent, at lower latencies.

5.4.2 Test 2 - Controllers failure in one region

Mastership change - elapsed time

These results are used to assess the time for the cluster to balance the masterships when all the ONOS instances from one region (Aveiro) are shut down.

In figure 5.2 is visible the measured averages of time the ONOS cluster takes to balance the scenario, to be exact the figure displays the average time of each of ONOS' attempts at balancing the scenario. This discrepancy is due to ONOS,

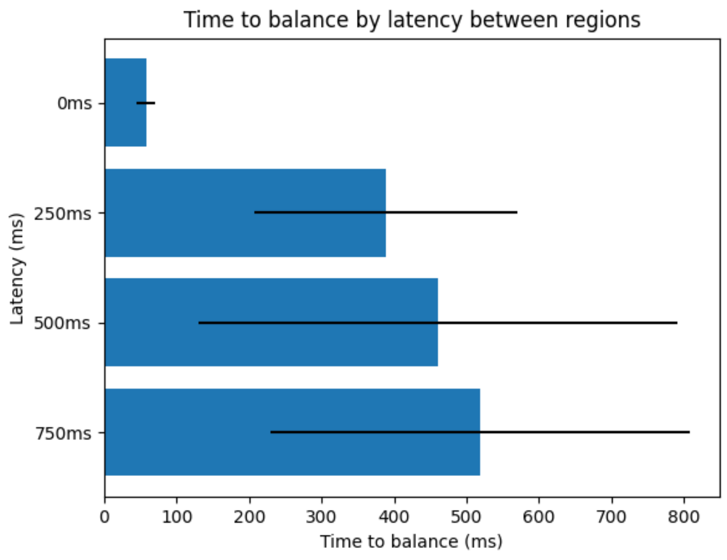


Figure 5.2: Time taken to perform the mastership balancing operation. The blue bar presents the average, while the black line shows the standard deviation.

under certain conditions, which are met in this test, not being able to determine that the scenario is balanced, which will be discussed later in this document.

As expected, the time the cluster takes to balance the system increases with the added latency, especially when increasing the latency from 0ms to 250ms. For the remaining latency values the cluster was able to balance the system within a reasonable time frame, with the biggest registered value being 997.437ms

Mastership change – region awareness

These results are used to analyze how the mastership associations change when the ONOS instances from Aveiro are shut down.

The measured results are displayed in figures 5.3 and 5.4. While figure 5.3 accounts for the mastership changes of all OVS instances, figure 5.4 only accounts for Aveiro's OVS instances. Both figures show a small number of mastership changes to Aveiro's ONOS instances. This is due to the script used to automate the tests shutting down Aveiro's ONOS one by one, possibly creating the opportunity for the orphan OVS instances to transition to the remaining ONOS in the region. Further analysis of the logs revealed that in the 750ms iterations, some of these changes into Aveiro are made to OVSs outside of the region, and also happen before any ONOS instance is shutdown, possibly being caused by instability due to the high latency.

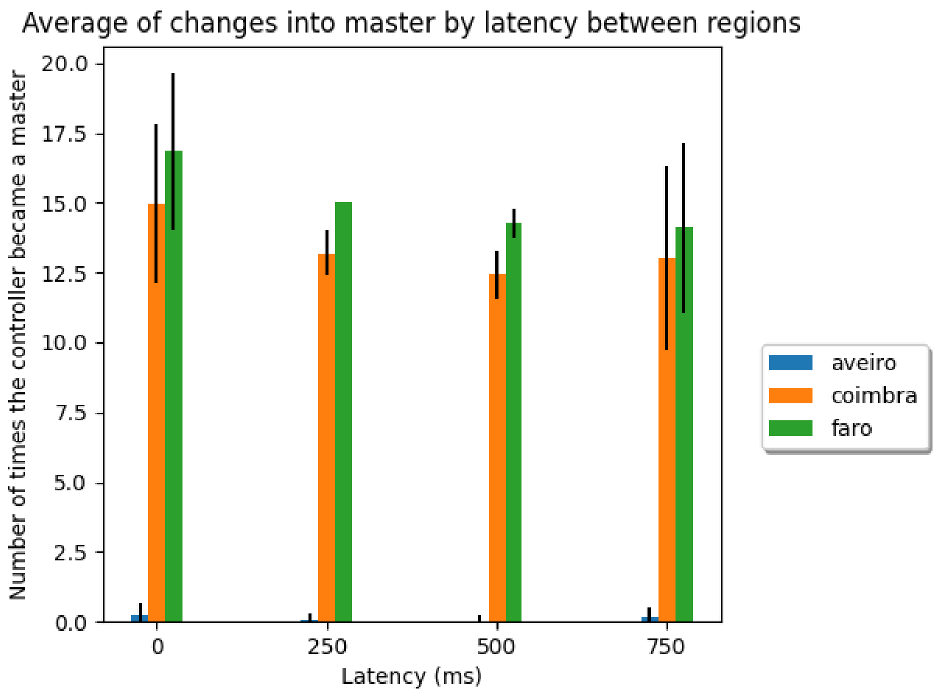


Figure 5.3: Number of times each ONOS instance became the master of an OVS instance.

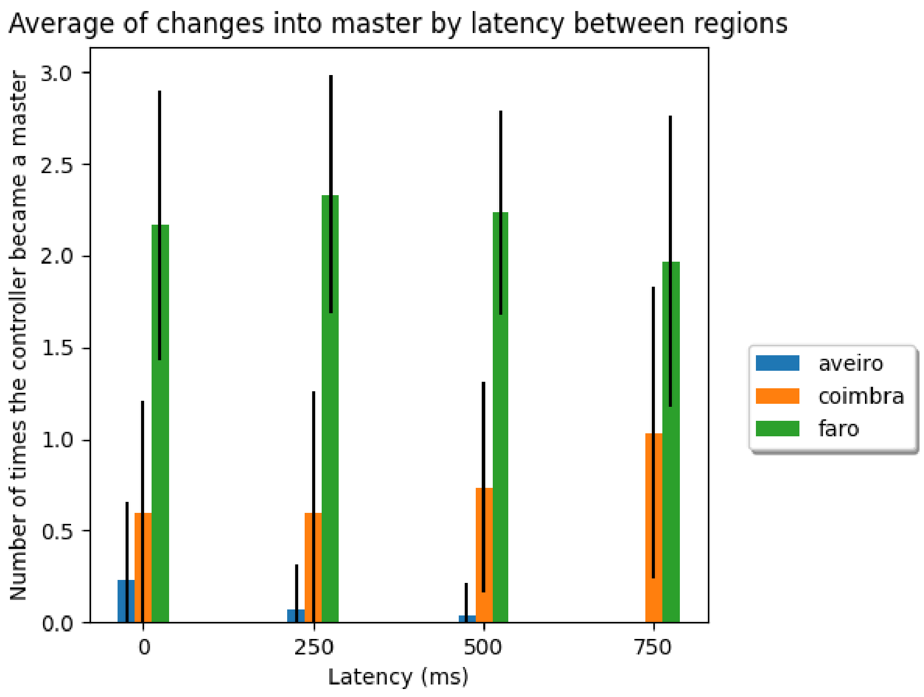


Figure 5.4: Number of times each ONOS instance became the master of one of Aveiro's OVS, grouped by region. For this graph, only the first mastership change of each OVS in each iteration was counted.

Figure 5.4 reveals that the orphan OVS have a tendency to have their mastership changed to Faro. This behavior is unexpected and the reasons for such could not be determined. ONOS' documentation refers that "A candidate node is selected from the pool of known standby nodes for a device" and that "... this pool is a ordered list of NodeIDs in preference order." [Koshibe and Olkhovskaya, 2016], but the factors in determining preference are not defined.

Detecting mastership connection fail

These results will assess how much time the OVS instances take to detect that the ONOS instance is no longer available.

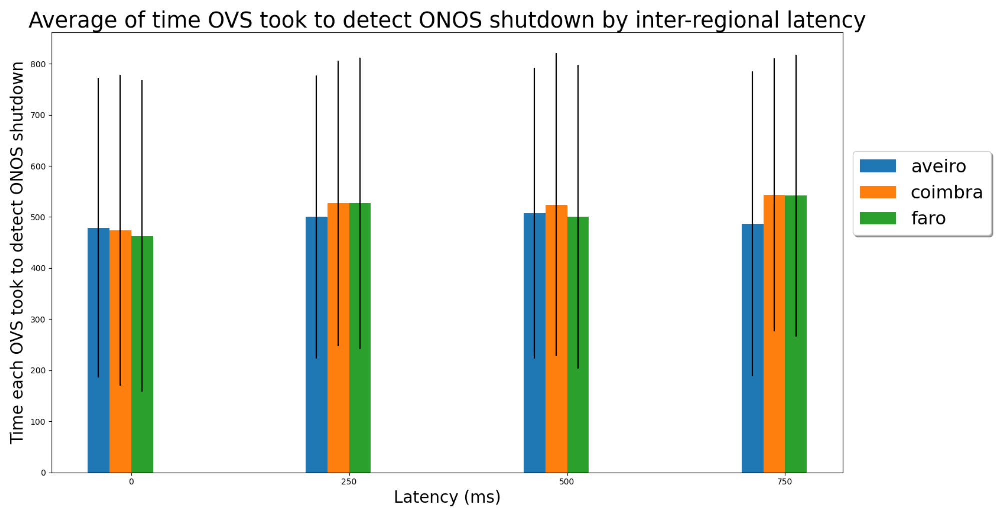


Figure 5.5: Average of the time each region's OVS instances took to detect an Aveiro's ONOS instance shutting down

Figure 5.5 presents the averages of the measured times the different regions' OVS instances took to detect the loss of connection with Aveiro's ONOS instances, as these were shut down. It is observable that at 250 and 750ms the expected behavior is observed, with Aveiro's OVS instances taking on average less time to detect the shutdowns and the other regions taking very similar times. This behavior is expected since there is no latency added within a region, and the links Aveiro-Coimbra and Aveiro-Faro have the same latency added. At 0ms, however, Aveiro's OVS instances unexpectedly take more time to detect the shutdowns than the others. Finally, at 500ms, Faro's OVS instances unexpectedly take less time to detect the shutdowns than Aveiro's instances.

5.4.3 Test 3 - Latency tolerance awareness

Figure 5.6 shows that while at higher latencies ONOS tends to set Faro's controllers as masters more often. Not only that, but the figure also shows that the OVS instances still transition between regions many times, which is unexpected. Further analysis of the log files reveals that most of these transitions are

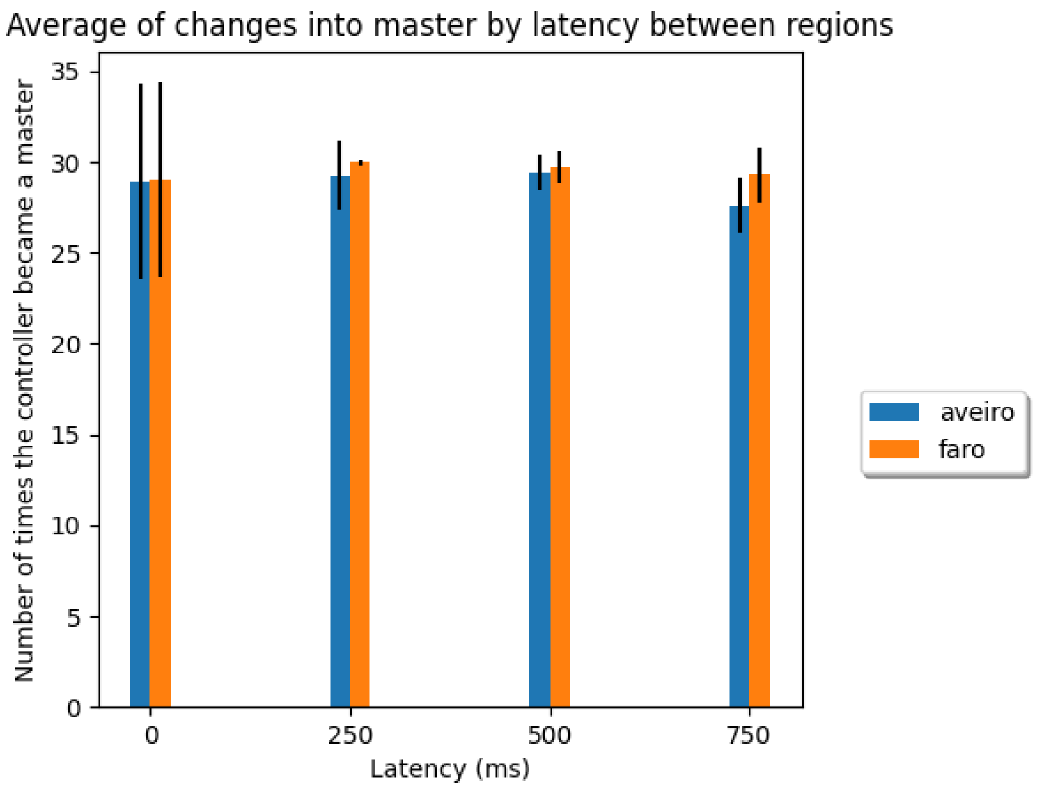


Figure 5.6: Number of times each ONOS instance became the master of one of Coimbra's OVS.

not caused by either a loss of connection with the current master or by the load balancer.

5.4.4 Test 4 - Latency tolerance awareness (cont.)

It is observable in figure 5.7 that a high number of mastership changes occur. This behavior, along with the behaviors observed in tests 2 and 3 (sections 5.4.2 and 5.4.3, respectively) might indicate that the hotfix used to make the system stop crashing when all controllers in a region went offline might be the cause the unexpected behaviors. The region-based mastership balance is expecting to have at least one controller in each region. If there is no controller in a region, the mastership associations do not stabilize and keep changing between the remaining regions. For instance, multiple instances of ONOS transitioning the OVS instances between regions for no apparent reason as the system is balanced and yet the OVS instances keep switching regions.

5.4.5 Test 5 - Shutting down two ONOS instances from a region

Multiple instances appear of the OVS' mastership being changed to one of the other regions, but later on, being changed back to the remaining Aveiro ONOS

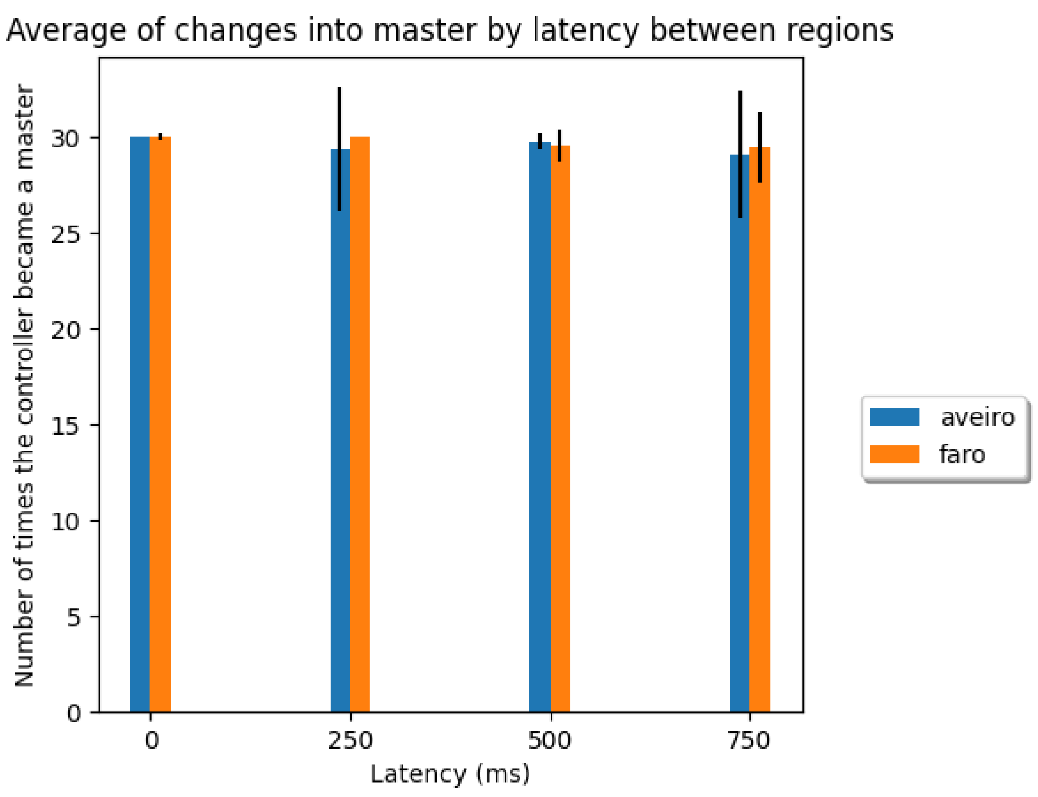


Figure 5.7: Number of times each ONOS instance became the master of one of Coimbra's OVS.

instance. While this change appears to happen because of load balancing, as the former follows the latter, the logs don't indicate the change like usual. For this reason, a new set of tests (test 6) were executed to verify if the same behavior happens when the load balancer is disabled. A tendency to change into Faro is also observed.

It should be noted that in these tests, once the system is stabilized the transitions stop, unlike in tests 2 and 3 (sections 5.4.2 and 5.4.3, respectively), which does not refute the hypothesis presented in test 4 (section 5.4.4).

Figure 5.8 shows the average number of times each ONOS instance became master of one of Aveiro's OVS instances. While the colored lines represent the number of times one of Aveiro's OVS instances had their mastership changed to the corresponding ONOS instance, the black line represents the standard deviation. It is observable that the average for aveiro3 is 2 with no standard deviation. This is the expected behavior, as while the OVS instances may have their mastership changed to one of the other regions (with the Faro region being preferred for unknown reasons) they always have their mastership changed back to Aveiro (aveiro3 more precisely, since it is the only remaining instance).

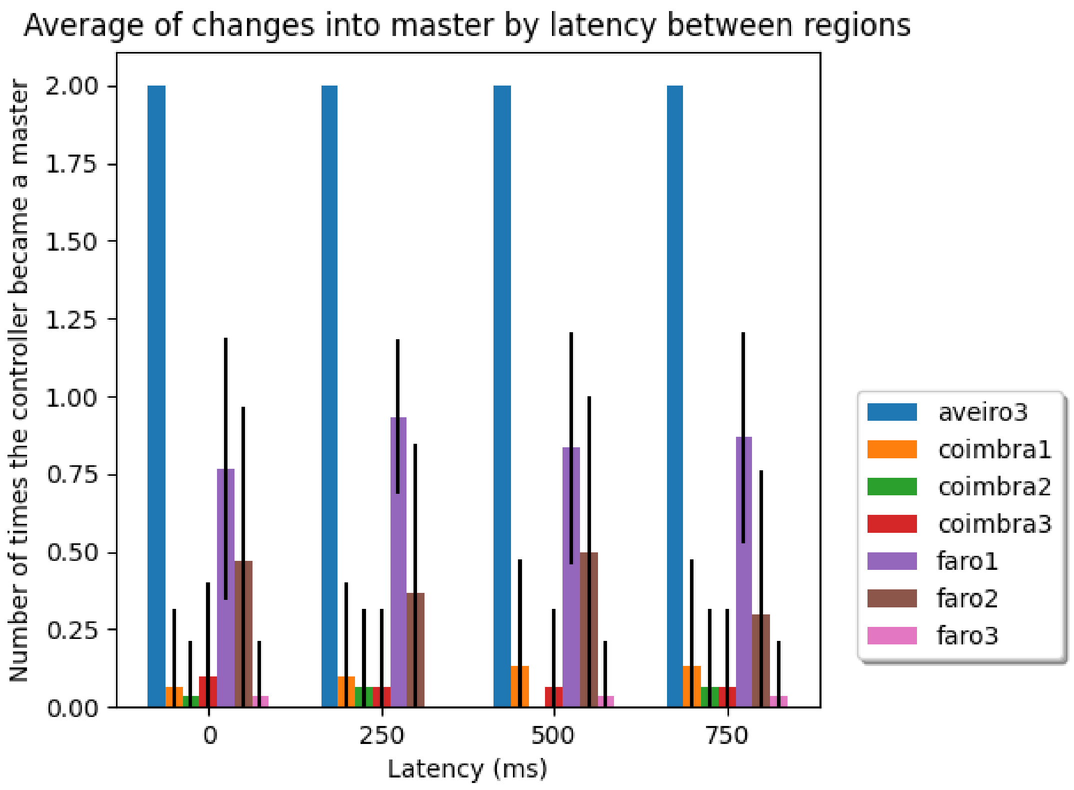


Figure 5.8: Number of times each ONOS instance became the master of one of Aveiro's OVS.

5.4.6 Test 6 - Shutting down two ONOS instances from a region (cont.)

The results of this test show that without the load balancer, the Aveiro OVS instances that lose their master don't necessarily end up in their region. By reading the logs a pattern is observed: either the OVS stay in the Aveiro region once the ONOS instances are shut down, or if they are moved to another region, it is never moved back to the Aveiro region. This leads the author to believe the *mastership load balancer* application was the cause of the desired behavior of OVS instances being moved back to their region when they were moved to another.

Figure 5.9 shows the average number of times each ONOS instance became master of one of Aveiro's OVS instances. While the colored lines represent the number of times one of Aveiro's OVS instances had their mastership changed to the corresponding ONOS instance, the black line represents the standard deviation. It is observable that unlike in figure 5.8, the average for aveiro3 (the blue bar) is lower than 2, the expected value, showing that the 2 orphans OVS instances do not always have their mastership set to aveiro3.

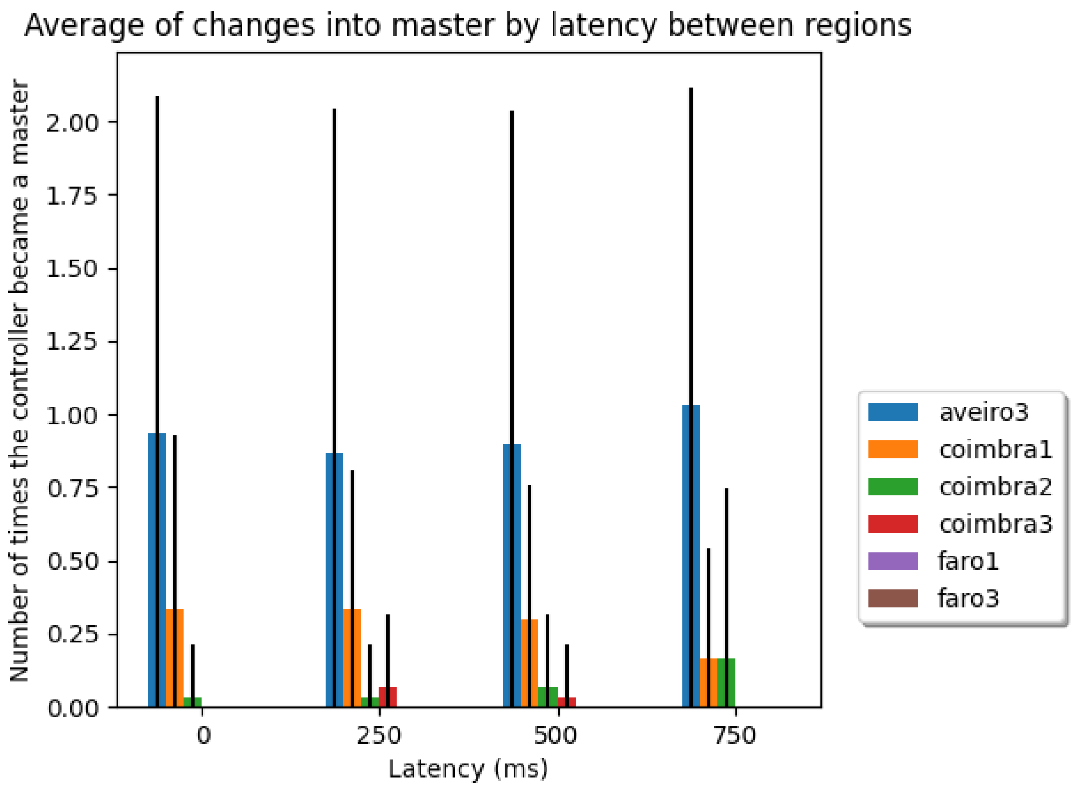


Figure 5.9: Number of times each ONOS instance became the master of one of Aveiro's OVS.

5.5 Difficulties

During the execution of this work a few difficulties were encountered, hindering progress. The following are some of the encountered difficulties:

- **Lacking ONOS documentation** - To form an ONOS cluster the use of Atomix is mandatory, but not all versions of these two tools are compatible with each other, and since no documentation on this was found, some trial-and-error was necessary.

- **OpenFlow version** - Although both ONOS and OVS support the latest version of OpenFlow (1.5 at the time of writing), some discrepancies appear to exist between the two causing errors to happen in message parsing. This issue was solved but forcing OVS to use the older 1.3 version of OpenFlow, as ONOS will change to the version the switches are using.

- **Regions implementation** - The implementation of regions in ONOS seems to have either not been finished or fully tested, as a *NullPointerException* occurs at the time of load balancing if all controllers in a region are offline, putting the whole system in an unstable state where it is unable to balance the load of the available nodes. To solve this issue, firstly ONOS was updated from version 2.5 which was being used to 2.7, the latest at the time of writing, when this update proved ineffective the only found solution was to change the source code to check for this edge case and use all online nodes for balancing if needed.

- **Upgrading ONOS version** - As stated before, an update to ONOS was assumed to be needed to fix the *NullPointerException* in case of all nodes in a region being down. This proved to be inconvenient, since not only the Atomix version used until then was not compatible with the new ONOS version, but the configuration files were not compatible with the new version either and had to be remade.

5.6 Discussion

The goal of these tests was to analyze the behavior of ONOS as a distributed SDN controller in a geographically distributed scenario and how it behaves when the quality of the links connecting the different regions degrades. With that said, the behavior observed demonstrated that ONOS is not ready for this use case. While the tool has the potential, with the desired behaviors being observed when a region has at least one master available, the bug that causes the whole cluster to stop functioning correctly when all masters in a region shutdown is not acceptable, as it not only affects the masterless region but all others too. While the author tried to fix the problem, it was nothing more than a band-aid solution, as it is hypothesized that said solution is causing the erratic behaviors seen in tests 2, 3, and 4 (see sections 5.4.2, 5.4.3 and 5.4.4). While the project is still active, a solution for this issue should not be expected in the foreseeable future as the project is, at the time of writing, being migrated to a new architecture code-named μONOS.

Chapter 6

Requirements

This chapter presents the requirements elicited for the project. Requirements are of extreme importance when designing a piece of software as they detail the desired functionality and characteristics of the software system, guiding the development process by defining the scope of the project and what needs to be accomplished. In the context of a software system, requirements can be divided as:

- Functional requirements - Define the functions the system must be able to perform to achieve the user's goals, how the system should react to the user's inputs and how the system should behave in certain situations.
- Non-Functional requirements - Also known as quality attributes, describe how the system must perform in terms of usability, security, performance, among other characteristics. These requirements define what characteristics the system needs to meet the needs of its users, as such they must be measurable.
- Design Constraints - Define limitations on the design of a software system. These can vary, from budget constraints to specific languages among other limitations. As they can have a significant impact on the design they must be carefully defined so the development team can consider them when designing the system.

As explained in section 1.2 the project's goal is to create a vBNG to be deployed in a NFV platform. To achieve this, both the requirements of the vBNG and the NFV platform must be considered.

6.1 Functional Requirements

The identified functional requirements are displayed in tables 6.1 and 6.2. Table 6.1 shows the functional requirements for the vBNG while table 6.2 details the requirements for the platform supporting it.

Each requirement is assigned a priority. This is done to define which requirements must be achieved, which requirements should be achieved but are secondary and which will not be achieved. To this end, the following priorities are used:

- **Must Have (MH)** - The highest priority, requirements with this priority are considered critical for the success of this project.
- **Should Have (SH)** - Requirements with this priority while important are deemed not critical for the success of the proof-of-concept.
- **Could Have (CH)** - Requirements with this priority are deemed out of scope of the proof-of-concept, although would be necessary in a real scenario.

Table 6.1: vBNG functional requirements

ID	**Requirement**	**Description**	**Priority**
FR-001	Packet Forwarding	The system forward packets to their destination.	MH
FR-002	PPPoE	The system allows clients to establish PPPoE connections to it.	MH
FR-003	IP Address Assignment	The system assigns unique IP addresses to the connected clients for proper communication and identification.	MH
FR-004	Multicast	The system enables one-to-many communication by sending packets to the pertinent subscribers.	CH
FR-005	Rate Limiting	The system sets limits to data transmission rate in a a per-customer basis.	SH
FR-006	User Authentication	The system authenticates users connecting to the network, verifying their identities before granting access.	MH
FR-007	Access Control	The system enforces access control policies, restricting and granting network resource access to based on predefined rules.	MH
FR-008	QoS Control	The system prioritizes specific traffic types, ensuring quality of service levels according to predefined parameters.	CH

6.2 Non-Functional Requirements

Table 6.3 summarizes the non-function requirements elicited for this project. To each requirement is assigned a priority, between *HIGH*, *MEDIUM* and *LOW*.

Table 6.2: NFV platform functional requirements

ID	**Requirement**	**Description**	**Priority**
FR-009	Ease of Deployment	The platform offers straightforward installation and setup procedures.	MH
FR-010	Templates	The platform supports the creation and management of templates for VNFs and NSs.	MH
FR-011	Simple deployment process	The platform allows users to easily select a template to deploy.	MH
FR-012	Networking	The platform allows configuration and management of networking aspects, such as virtual networks, subnets, IP addressing, and routing.	MH
FR-013	Lifecycle Management	The platform provides capabilities for managing the full lifecycle of VNFs and NSs, including creation, deployment, updating, and destruction.	MH
FR-014	Access to VNFs	The platform provides mechanisms to access VNF instances directly for troubleshooting purposes.	MH
FR-015	Web Interface	The platform provides a user-friendly web-based interface for managing VNFs, NSs and templates.	MH
FR-016	CLI	The platform offers a robust CLI to allow automation of tasks.	SH

Table 6.3: System non-functional requirements

ID	Requirement	Priority
NFR-001	The vBNG should efficiently handle high throughput demands during peak load conditions.	HIGH
NFR-002	The vBNG should effectively manage concurrent user connections.	HIGH
NFR-003	The vBNG should minimize packet loss.	LOW
NFR-004	The vBNG should contribute to low latency in network communication.	MEDIUM
NFR-004	The vBNG should manage memory usage efficiently.	MEDIUM
NFR-005	The vBNG should optimize CPU usage for efficient performance.	MEDIUM
NFR-006	The vBNG should ensure establishment of subscriber sessions.	Low

Quality attributes must be testable, as such the following tables present the quality attribute scenarios defined.

Table 6.4: Quality Attribute Scenario NFR-001

Id	NFR-001
Description	The vBNG should efficiently handle high throughput demands during peak load conditions.
Priority	HIGH
Source of stimulus	Subscribers
Stimulus	Surge in traffic demand.
Environment	Normal Operation
Artifact	vBNG
Response	The vBNG forwards user traffic to it's destination.
Response Measure	The vBNG should maintain a total throughput of atleast 2 Gbps on average.

For NFR-002, a subscriber reference model is necessary. Table 6.5 showcases the defined model. This model was based on the "High User" model from [Cruz et al., 2013], while the Voice over Internet Protocol (VoIP) bitrate was based on [Altice].

Table 6.5: Subscriber Model

Service	**Bitrate (Mbps)**
Internet	10.0
Telephony	1.0
SDTV 2 channels (MPEG-4)	3.0
HDTV 2 channels (MPEG-4)	16.0
Total	30

Table 6.6: Quality Attribute Scenario NFR-002

Id	NFR-002
Description	The vBNG should effectively manage concurrent user connections.
Priority	HIGH
Source of stimulus	Subscribers
Stimulus	Normal network usage
Environment	Normal Operation
Artifact	vBNG
Response	The vBNG forwards user traffic to it's destination.
Response Measure	The vBNG should provide an average throughput of 30 Mbps to 100 subscribers.

Table 6.7: Quality Attribute Scenario NFR-003

Id	NFR-003
Description	The vBNG should minimize packet loss.
Priority	LOW
Source of stimulus	IPTV and VoIP services
Stimulus	UDP traffic stream
Environment	Normal Operation
Artifact	vBNG
Response	The vBNG forwards UDP traffic to relevant subscribers.
Response Measure	The vBNG should provide a packet loss ratio of at most 1%.

Table 6.8: Quality Attribute Scenario NFR-004

Id	NFR-004
Description	The vBNG should manage memory usage efficiently.
Priority	MEDIUM
Source of stimulus	Network traffic processing demands.
Stimulus	Memory usage
Environment	Normal Operation
Artifact	System memory
Response	The vBNG manages available memory to prevent reliance on swap file.
Response Measure	The vBNG should not use more than 2GB of memory.

Table 6.9: Quality Attribute Scenario NFR-005

Id	NFR-005
Description	The vBNG should optimize CPU usage for efficient performance.
Priority	MEDIUM
Source of stimulus	Network traffic processing demands.
Stimulus	CPU workload
Environment	Normal Operation
Artifact	CPU
Response	The vBNG manages the CPU time efficiently.
Response Measure	The vBNG should show an average CPU usage of at most 80% while serving up to 100 subscribers

Table 6.10: Quality Attribute Scenario NFR-006

Id	NFR-006
Description	The vBNG should ensure establishment of subscriber sessions.
Priority	LOW
Source of stimulus	Subscriber
Stimulus	Attempts to establish a session
Environment	Normal Operation
Artifact	vBNG
Response	The vBNG establishes the subscriber session.
Response Measure	The average time between an authentication request and the response should not exceed 10 seconds.

6.3 Design Constraints

Table 6.11 presents the identified design constraints for the project.

Table 6.11: System design constraints

ID	Constraint
DC1	The system must integrate the concepts of SDN and NFV.
DC2	The vBNG must be compatible with the chosen NFV platform.
DC3	The vBNG should rely on an external AAA server for better scalability.
DC4	The system must be implemented using available free open-source tools.

6.4 Threshold of Success

This project will be considered a success if the following criteria are met:

- All *Must Have* functional requirements are met
- At least 70% of the *Should Have* functional requirements are met
- All the *High* non-functional requirements are met
- At least 50% of the *Medium* non-functional requirements are met

Chapter 7

System architecture

This chapter details the software architecture of the platform proposed in this thesis. Since the architecture of each component highly depends on the implementation used, a low level view of each component is not provided. Despite this, this chapter details the functions of each component in order to specify the desired functionality, which lead to the solutions used in chapter 8.

7.1 Platform Architecture

Figure 7.1 presents the high-level architecture of the system. This architecture is composed of four main components: the NFV MANO, the NFVI, the field network and the VNFs.

The Field Network, being part of the SDN subsystem, is composed of a mesh of OpenFlow-enabled Forwarding devices, as support for the protocol is required for such devices to interact with the SDN controller. These devices are responsible for forwarding data between the different hosts and VNFs, according to the flow rules installed by the SDN controller.

The NFV MANO is the central of the NFV sub-system. It is responsible for managing the NFVI, and the VNFs and setting up connections between. It is composed by three main components: NFVO, VNFM and VIM (see section 2.2). The NFV MANO is responsible for managing all aspects of the VNFs, including deployment, managing and terminating them.

The SDN controller is the central component of the SDN subsystem. Its primary role is to enable connectivity between the many hosts and VNFs. Despite being a logically centralized controller, the SDN controller should be composed of multiple controller nodes in a cluster in order to avoid single points of failure, and so meet the availability requirements.

The NFVI consists of the servers where the virtualization layer is installed and the VNFs are deployed. The virtualization layer used is completely dependent on the solution used for NFV MANO.

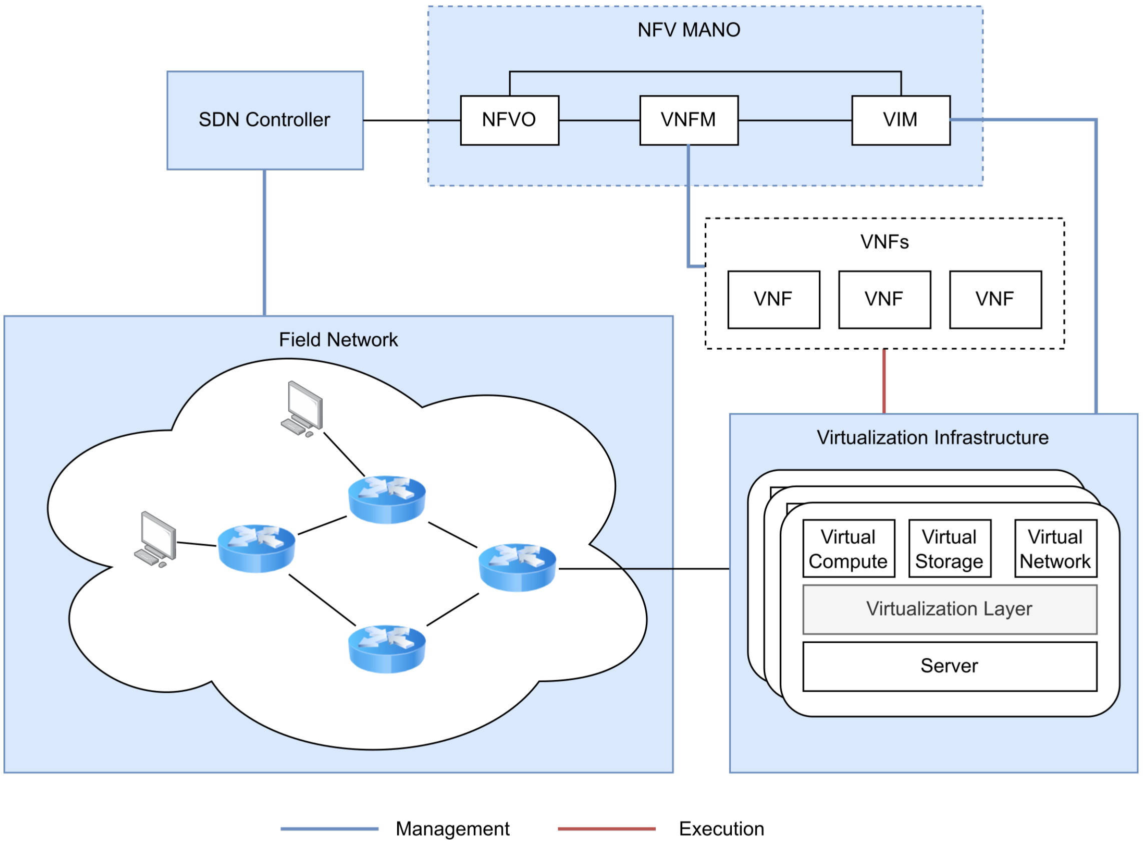


Figure 7.1: High level system architecture

7.2 vBNG Architecture

While the vBNG was originally planned to follow a disaggregated design, the research done during the exploratory phase (see section 4.2) work revealed that the tools available were at the time of writing better suited to a more traditional aggregated architecture, therefore such an architecture was designed, as seen in figure 7.2.

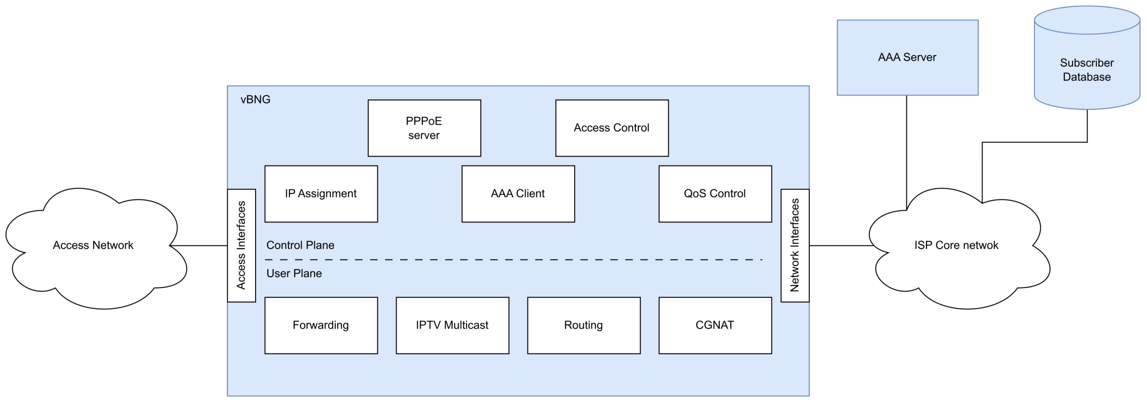


Figure 7.2: vBNG Architecture

This architecture was designed to fulfill the requirements presented in chapter 6, with a major focus on the functional requirements (see section 6.1).

The Access Interfaces component refers the connection points through which subscribers connect to the broadband network. These include Ethernet and PPPoE interfaces.

The Network Interfaces component refers to the interfaces connecting the vBNG to the internal network, it's services and the internet.

The Forwarding component illustrates the function of directing incoming and outgoing data packets between the subscribers and the ISP's network, packet encapsulation/decapsulation and QoS enforcement. Functional requirement FR-001 is satisfied by the operation of this component.

The Subscriber Database holds subscriber information, such as credentials, usage records, service plans, access control data and more. Therefore it's an essential component in achieving the functional requirements FR-002, FR-005, FR-006 and FR-007.

The AAA Server component is responsible for authenticating subscribers connecting to the vBNG, granting them access to the ISP's internal network and services. This component is also tasked with accounting for billing purposes. It does this by interfacing with the Subscriber Database component. This component plays a role in fulfilling functional requirements FR-002, FR-005, FR-006 and FR-007.

The QoS component illustrates the service that defines the policies and rules enforced by the Forwarding component, ensuring distinctive treatment for diverse data traffic types within the network, by following predefined policies. Traditionally, in the context of a BNG, the QoS control prioritizes Internet Protocol Television (IPTV) and telephony traffic, ensuring these services receive the necessary bandwidth, reduced latency, and minimal packet loss, resulting in smoother and more reliable performance and therefore a better experience. This component was introduced to address functional requirement FR-008.

The IPTV Multicast component is responsible for establishing multicast streams with sources and processing Internet Group Management Protocol (IGMP) messages from subscribers, fulfilling the functional requirement FR-004. In summary, this component establishes the multicast streams with the upstream sources, and duplicates packets which are then forwarded to the relevant subscribers. Subscribers send IGMP messages to subscribe and unsubscribe from this streams, informing the BNG that they want to receive this content. This avoids network congestion by not having a stream between the BNG and data source for each subscriber consuming the content.

The Carrier-Grade NAT (CGNAT) component is responsible for translating subscribers IP addresses to a single public IP address. This technology is commonly used by ISPs to mitigate IPv4 address exhaustion [Jiang et al., 2011]. This component was introduced to facilitate the fulfillment the functional requirements FR-001 and FR-003.

The Access Control component is responsible for defining rules and policies to manage the access that devices and subscriber have over the resources, such as restricting access to the AAA server and Subscriber Database, as defined in functional requirement FR-007.

The IP Assignment component was introduced to address functional requirement FR-003. This component is responsible for assigning IP addresses to the subscribers once the Point-to-Point (PPP) connection is established.

The PPPoE Server component play as crucial role in the proposed solution, as it provides many functions that facilitate the connection between subscribers and the ISP's network. It provides the following functions:

- Packet Encapsulation/Decapsulation - The PPPoE encapsulates packets into PPP frames and then encapsulates those frames within Ethernet frames. These PPP frames facilitates the establishment and management of point-to-point connections by adding session, control, and protocol information to network data, fulfilling the functional requirement FR-003.
- Authentication and Authorization - By interfacing with the AAA client, the PPPoE server authenticates subscribers by requiring them to provide credentials such as username and password, taking a role in achieving the functional requirement FR-006. Once authenticated, subscribers are granted access to the ISP's network and resources.
- Billing and Accounting - The PPPoE server collects usage data that can be used for billing and accounting purposes, such as session times and data transferred.
- IP Assignment - By interfacing with the AAA client, the PPPoE server assigns IP addresses to subscribers once the connection is established, making it essential in addressing the functional requirement FR-003.
- Security - Once the session is established, traffic encryption encryption may be used.
- Traffic Shaping - The PPPoE Server is able to control the flow of data for each subscriber. It allocates bandwidth based on predefined policies, preventing any single subscriber from monopolizing the available resources.

Chapter 8

Implementation

This chapter describes how the various components were configured to implement the system. Section 8.1 lists the chosen tools and their function in the context of the architecture detailed in chapter 7. Section 8.2 details how MicroStack was deployed and configured, section 8.3 details how OSM was deployed and configured, section 8.4 describes how VyOS was configured to enable the vBNG functionality and section 8.5 shows how RADIUS was configured for AAA.

8.1 Tools Used

In this section, the selection of tools used in the implementation of the proposed solution are listed and explained. Table 8.1 showcases a summarised view of the choices and their rationale.

When researching NFV platforms that could support the solution, two options were discovered, ONAP and a combination of OSM and a VIM. Both these alternatives provide the required functionality. Ultimately, it was decided that OSM would be the better choice, as ONAP has steep system requirements [ONAP], requiring a minimum of 224 gigabyte (GB), 112 vCPUs and 160 GB of storage, which the author could not meet. This problem was not observed in the case of OSM. Furthermore, the author also could not obtain access to ONAP's support channels which was not a problem in the case of OSM.

Having chosen to pursue an implementation using OSM, a VIM was necessary. The OSM documentation details a few options for a VIM deployment, although the only option fitting the design constraint DC4 was OpenStack as the other options were not free or open-source, or were deprecated. The option chosen was MicroStack, a small-scale OpenStack deployment, created to allow developers to quickly and easily obtain an working environment that while limited, still presented all the necessary functionality, although it should be noted that for production a full OpenStack environment would be necessary. MicroStack implements the functions of the VIM and Virtualization Infrastructure components. It also manages the connectivity between the various VNFs and the exterior using OVS bridges that are manage by the Neutron service, an analog to an SDN controller.

VyOS was chosen as the basis for the vBNG as it was the only solution found that met all the requirements of providing all the necessary functionality, free and open-source and compatible with the already chosen tools and environment. While not a requirement, the fact the project is under active development and having good documentation, which eases the process of getting started, made it clear that VyOS was the most viable choice.

FreeRADIUS was chosen as the AAA Server for being the standard open-source implementation of the RADIUS protocol, which is necessary to allow VyOS to use an external server. Similarly, MySQL was chosen as the Subscriber Database for its compatibility with FreeRADIUS and due to the author having some prior experience with the technology.

Table 8.1: List of tools used to implement the solution

Solution	**Components**	**Reason**
OSM	NFVO VNFM	Under active development Simple deployment process Support from community members
MicroStack	VIM Virtualization Infrastructure Field Network SDN Controller	Under active development Simple deployment process Low system requirements
VyOS	vBNG	Meets most functional requirements Good documentation Active development
FreeRADIUS	AAA Server	Free Open-Source implementation of the RADIUS protocol Compatible with VyOS
MySQL	Subscriber Database	Compatible with FreeRADIUS Author has Some prior experience with the solution

8.2 Microstack

To deploy MicroStack, a machine was setup with two network interfaces, one connecting it to the internal network, providing connection to the MANO and internet, and another providing a connection point for the the future subscribers. This machine is equipped with 8 vCPUs, 50 GB of memory and 64 GB of storage. Due to this machine being used to deploy VMs, virtualization extensions should be enabled. Finally, to allow PPPoE connections to be established, the switch connecting the clients to the machine should have both promiscuous mode and forged transmits enabled, to allow establishment of the PPPoE connections.

The guide present in MicroStack's documentation [Canonical] was followed to install it. Following this, the images necessary to deploy the VNFs are loaded. Finally, the external network "access" is create (along side a sub-network "access-subnet") and a OVS bridge that connects it to the access network is setup. The necessary commands are listed in Listing 8.1 (alternatively the web interface can be used).

Listing 8.1: Commands used to setup the MicroStack infrastructure

```
#add images to MicroStack
microstack.openstack image create --file="./bionic-server-
    cloudimg-amd64.img" --container-format=bare --disk-
    format=qcow2 ubuntu18.04
microstack.openstack image create --file="./vyos-1.3.3-
    cloudinit.qcow2" --container-format=bare --disk-format=
    qcow2 vyos-1.3.3

#create external network and subnet
microstack openstack network create --external --provider-
    physical-network physnet1 --provider-network-type flat
    access

microstack openstack subnet create --network access --
    subnet-range 192.168.0.0/24 --no-gateway access-subnet

#create bridge to external network
microstack.ovs-vsctl --retry --may-exist add-br br-ac --
    set bridge br-ac datapath_type=system protocols=
    OpenFlow13,OpenFlow15

#map bridges to external networks (br-ex is created by
    microstack at setup)
microstack.ovs-vsctl set open . external-ids:ovn-bridge-
    mappings=physnet1:br-ex,physnet2:br-ac
microstack.ovs-vsctl add-port br-ac ens192
```

8.3 OpenSource MANO

To deploy OSM the guide present in the documentation [OSM, c] was followed. A virtual machine was setup for this deployment, equipped with 8 vCPUs, 32 GB of memory and 50 GB of storage. Once OSM is installed the VIM account must be created so OSM can deploy NSs into Microstack.

To create the VIM account, OSM's installation script was adapted to add a remotely deployed MicroStack instance. The full script can be seen in Appendix D. In summary, the script does some basic setup (creating a management network and keypairs) and in the end obtains the file `/var/snap/microstack/common/etc/microstack.rc` which contains the necessarily information to create the VIM

Listing 8.2: Commands to import the descriptors

```
#add vnfd, run in directory containing descriptor packages
osm vnfpkg-create radius_sql_vnf
osm vnfpkg-create vyos_vnf

#add nsd
osm nspkg-create radius_sql_ns
osm nspkg-create vyos_ns
```

account in OSM, including the username, password and the url for authentication. To add the descriptors, the web interface can be used or alternatively the commands shown in listing 8.2. It should be noted that the VNF descriptors must be added before the network service descriptors, or an error will occur, as the latter reference the former.

8.4 VyOS

This section documents the steps taken to configure VyOS. All the relevant files are included in Appendix B.

8.4.1 Image

To configure VyOS it is necessary to obtain an image that can be deployed into a cloud infrastructure. The version used was VyOS 1.3.3, an LTS release. The guide present in [Vyos, b] was followed to obtain the iso file. Once the iso has been obtained, VyOS's official VM image building tool [Vyos, a] is used to create a VM image that can be deployed in MicroStack.

To create the image the command shown in listing 8.3 was used. This command takes a few parameters but the most relevant ones are the ones pertaining to cloud-init as these allow VyOS to be configured during deployment using the tool.

Listing 8.3: VyOS VM image build command

```
ansible-playbook qemu.yml -e iso_local=/tmp/vyos.iso -e
   grub_console=kvm -e vyos_version=1.3.3 -e cloud_init=
   true -e cloud_init_ds=OpenStack,ConfigDrive,None -e
   enable_dhcp=true -e guest_agent=qemu -e enable_ssh=true
   -e keep_user=true
```

Then, the resulting image is loaded into the VIM using the command presented in listing 8.4, it should be noted the image name must match the one defined in the descriptors, or an error occurs.

Listing 8.4: VyOS VM image build command

```
microstack.openstack image create --file="./vyos-1.3.3-
    cloudinit.qcow2" --container-format=bare --disk-format=
    qcow2 vyos-1.3.3
```

8.4.2 Descriptors

The descriptors were based on the *hackfest_vyos_vnf* and *hackfest_vyos_ns* packages hosted in OSM's demo repository [OSM, a], for the sake of speeding up development. It should be noted that OSM provides tools to ease the creation of new descriptors, both using the command line [OSM, b] and web interfaces.

The VNF Descriptor (VNFD) is composed by two files, the YAML file describing the VNF and a cloud-init file to be used for configuration at boot time. These files can be viewed in appendix B, in listings B.1 and B.2 respectively. The YAML file describes a rather simple VNF, composed of a single Virtual Deployment Unit (VDU), the VM hosting the virtual function, with three connection points: one for the internal network, one for the external network and one for management. It also details the image the VNF uses, in this case "vyos-1.3.3", the same name that was given to the image when imported into the VIM, and the file to be used for cloud-init configuration. At the bottom it defines the resources to be allocated to the VNF, two virtual CPUs, 2GB (2 gigabytes) of memory and 10GB (10 gigabytes) of storage.

The cloud-init file contains a series of commands that VyOS should execute when booting to enable the BNG functionality. According to [VyOS, a] VyOS uses two cloud-config modules: "vyos_userdata" and "write-files". The first module runs each command in the "vyos_config_commands" section while booting for the first time, while the second module is capable of writing to files in the file system. Ideally all commands would be in the block corresponding to the first module but it was not possible so the second was used as a workaround. This is due to how VyOS handles its configuration, it is handled in the form of commits, where multiple configuration changes can happen but will only take effect once the "commit" command is used. This is a generally useful feature but caused an issue with the desired configuration, as the configuration for PPPoE requires multiple commands to be committed at once, which is impossible in the first module, additionally bash variables are used to automate the process, which is equally impossible using the first module. To this end the cloud-init script details that the commands to be used should be written to VyOS's postconfig script [VyOS, b], a script called after the initial boot. Finally, the double brackets are used to signal OSM that that value is a parameter and it should be replaced with the value provided in the configuration file. This provides a bit more flexibility to the descriptor, allowing for little changes to be made to the deployment without needing to edit the descriptor.

The postconfig script can be divided into the following sections:

1. **Check user group** - It starts by checking the group of the user running the

file and if it is not "vyattacfg" runs it as said group in order to avoid the problems, as explained here [VyOS, b],

2. **Setup PPPoE** - The script sets up the PPPoE server, setting the interface it listens in, the address passed to the clients, the pool of IP addresses to assign to clients, enabling rate limiting and setting the authentication mode to use a remote RADIUS server, although it could manage users locally. The decision to use an external RADIUS server was made with the intent of laying the groundwork for a future iteration of the project where multiple VyOS instances are deployed and access the RADIUS server for authentication, for better scalability.

3. **Setup SNAT** - The script sets up a NAT with IP masquerading, to hide the client's IP address and so MicroStack stops blocking packets coming from the clients. This happens due to the internal OVS bridge lacking the necessary flow rules to allow it, and creating them manually is not feasible.

4. **Setup Firewall** - Finally, the script sets up a firewall to block clients from accessing restricted networks.

5. **Commit** - The script commits the changes, applying the configurations and enabling the services configured.

The Network Service Descriptor (NSD), which can be seen in listing B.3, is used to create the template for a NS. As this NS is composed by a single VNF it's a rather simple one, simply mapping each of the VNF's connection points to a virtual link, that will be assigned to a network withing the VIM. In a more complex NS the virtual link would define the connection path between the different VNFs.

Once the descriptors are created, the commands presented in listing 8.5 to add them to the catalog, keeping in mind the VNFD must be added first. It should be noted that in this example `vyos_vnf` and `vyos_ns` are the directorys containing the descriptors, while `vyos-vnf` and `vyos-ns` are the names assigned to the descriptors.

Listing 8.5: Commands to add the VyOS descriptors to catalog

```
#Import descriptors
osm vnfpkg-create vyos_vnf
osm nspkg-create vyos_ns

#Update descriptors
osm vnfpkg-update vyos-vnf --content vyos_vnf
osm nspkg-update vyos-ns --content vyos_ns
```

Finally the configuration file, which can be viewed in listing B.4, is a file passed to OSM when deploying a NS to specify parameters about the deployment, such as which networks each VNF should be connected to, their IP addresses, or parameters defined in the VNFD. In this instance, the file is used to specify the IP address of the RADIUS server used for authentication, as well as the name of the networks the VNF should be connected to. For more information about the parametrization of the NS, refer to [OSM, d], keeping in mind the examples given pass the parameters inline instead of using a configuration file.

8.5 RADIUS

This section documents the steps taken to configure the RADIUS to be used for AAA. All the relevant files are included in Appendix C.

8.5.1 Image

The RADIUS server uses Ubuntu 18.04 cloud image as a base, as such it must be obtained from the official ubuntu repository and loaded into the VIM, as shown in listing 8.6, keeping note of the name given to the image.

Listing 8.6: VyOS VM image build command

```
microstack.openstack image create --file="./bionic-server-
   cloudimg-amd64.img" --container-format=bare --disk-
   format=qcow2 ubuntu18.04
```

Once OSM is deployed and properly configured, the instances can simply be created using the commands shown in Listing 8.7, or using OSM's web interface.

Listing 8.7: Commands used to launch the instances

```
#launch vyos
osm ns-create --ns_name vyos-ns --nsd_name vyos-ns --
   vim_account microstack-site-vim1 --config_file
   vyos_config.yaml

#launch radius
osm ns-create --ns_name radius-sql-ns --nsd_name radius-sql
   -ns --vim_account microstack-site-vim1 --config_file
   radius_sql_config.yaml
```

The files `vyos_config.yaml` and `radius_sql_config.yaml` serve as alternatives to the commands in [OSM, d], allowing the user to pass parameters to OSM at the moment of instantiation, such as which networks the instance should be connected to.

8.5.2 Descriptors

Similarly to the previously detailed VNFD, the one for the RADIUS server is composed by two files, the YAML file and the cloud-init file. This VNF is equipped with two network interfaces, one for management and the other to provide connection to the clients, the systems connecting to RADIUS for authentication, in this instance VyOS.

To configure the RADIUS server the guide present in [Ciro, 2022] was followed. The cloud-init file can be divided into the following main sections:

1. **Install packages**- The file defines the packages that Ubuntu should install when booting.
2. **Setup MySQL Database**- The script starts by configuring the database. It applies the schema and setup files provided by freeradius (open source RADIUS server) for the basic setup and then creates the row in the clients table, allowing VyOS access to the authentication service.
3. **Setup SQL Modules**- The script configures the modules to use the MySQL database, as freeradius allows the data to be loaded from configuration files. This process is mostly done by uncommenting the correct lines, as the default values are valid, and specifying the SQL syntax used.
4. **Enable SQL Modules**- Once the modules are properly configured, the script enables them by creating a symbolic link into the *mods-enabled* directory.
5. **Restart Freeradius**- Finally, once freeradius is properly configured, it is restarted as some of the configurations are only read once when the service starts.

As the NS is composed by a single VNF, the NSD, visible in listing C.2 simply maps each connector defined in the VNF to a virtual link.

As with VyOS descriptors, it is required to import them to the catalog, with the commands shown in listing 8.8

Listing 8.8: Commands to add the VyOS descriptors to catalog

```
#Import descriptors
osm vnfpkg-create radius_vnf
osm nspkg-create radius_ns

#Update descriptors
osm vnfpkg-update radius-sql-vnf --content radius_vnf
osm nspkg-update radius-sql-ns --content radius_ns
```

Finally, the configuration file is used to define the network the RADIUS will be connected to, and defines a static IP address for the interface used to serve the clients, for better control over the addresses used.

8.6 Prototype

Figure 8.1 presents the platform supporting the vBNG implemented using the chosen tools, Microstack and OSM.

MicroStack provides a several of services, including [Page et al., 2020]:

- **Horizon** - The official implementation of OpenStack's Dashboard. It provides a user interface to interact with OpenStack's services [Horizon, 2019]. In the context of this work, it was necessary to setup the virtual networks.
- **Keystone** - OpenStack service that provides authentication and authorization mechanism [Keystone, 2019], used to authenticate OSM and allow it to manage the VNFs.
- **Glance** - OpenStack's image catalog service [Glance, 2019]. This service allows users to register and discover VM images to use for VNF deployment.
- **Nova** - OpenStack service that provides mechanisms to provision the virtual servers. It supports creating virtual machines, baremetal servers and has limited support for system containers [Nova, 2019]. It also provides the metadata service [OpenStack, 2020a] that provides instances with data for configuration via cloud-init [Esler et al.], allowing for more flexible VNF deployments.
- **Neutron** - OpenStack's networking service, tasked with providing network connectivity to the instances [OpenStack, 2020b], by managing the *br-int* OVS bridge, where ports are created and to which the instances are connected. It's also tasked with intercepting the instances requests for metadata, and redirecting them to the Nova service [OpenStack, 2020a].

Some of the services provided by OSM include [Salguero et al., 2020]:

- **NBI** - OSM's North Bound Interface, Restful server that allows external applications to interact with OSM.
- **NG-UI** - Interacts with the NBI to provide a web GUI, where users can access OSM's functions.
- **LCM** - Live Cycle Management module, responsible for managing the workflows associated with the lifecycle events of VNFs and NS, such as instantiation, termination, scaling, healing, and upgrading.
- **RO** - Resource Orchestration module, responsible for interacting with the VIM layer to orchestrate resource allocation.

Figure 8.2 presents the topology of the virtual networks within MicroStack. The figure presents the following types of network:

- **External network** - Also known as Provider network, these networks are attached to the physical network infrastructure to provide connectivity between the VNFs and networks outside the virtual environment [OpenStack, 2023b].
- **Internal network** - Virtual network within the MicroStack environment that provides connectivity between the various VNFs [OpenStack, 2022a].

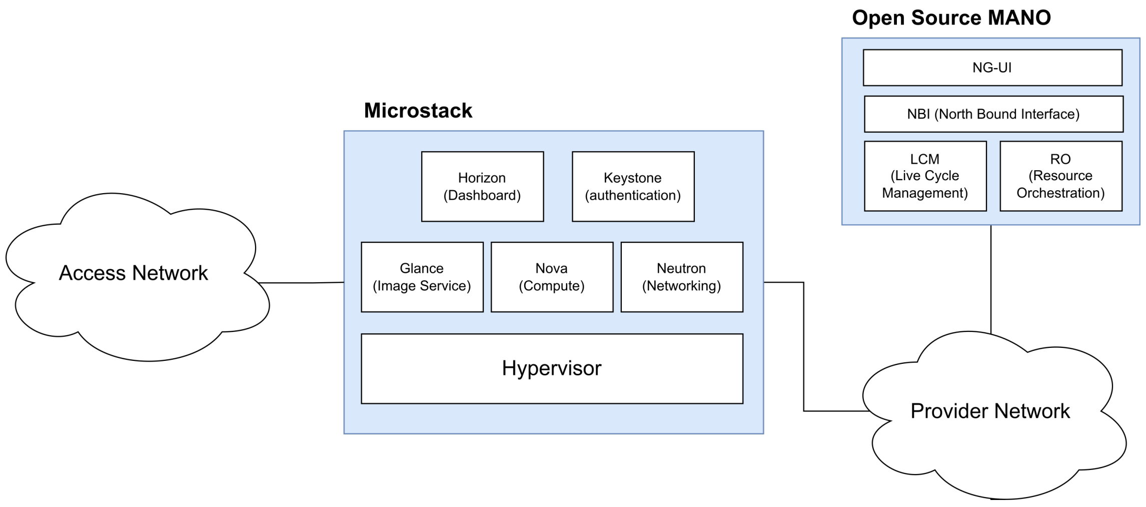


Figure 8.1: Implemented Architecture

- **Management network** - Functionally identically to the Internal Network, but needs to be accessible by OSM for VNF configuration [OSM, e].

The OVS bridges are setup to bridge the data center's network interfaces and external networks, allowing traffic to be forwarded between the two, enabling connections to be established between the subscribers and VyOS.

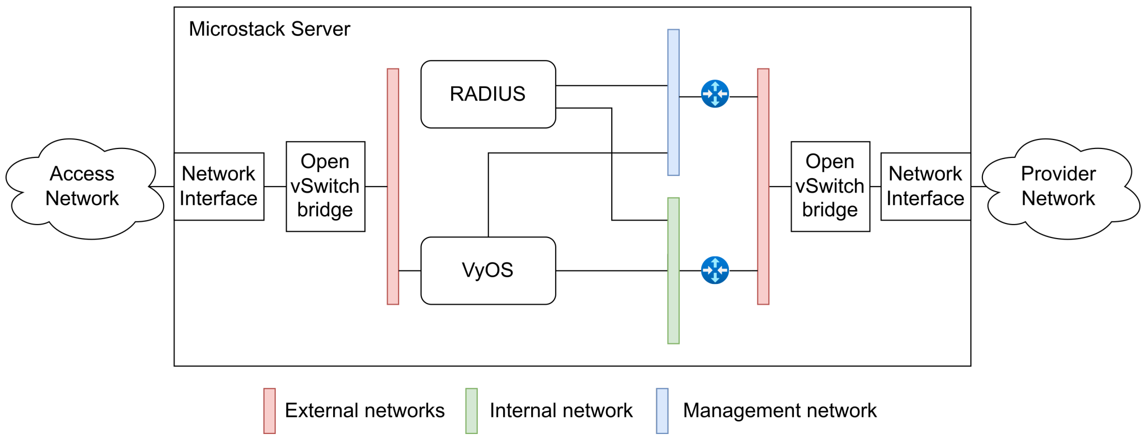


Figure 8.2: MicroStack virtual network topology

Chapter 9

Testing

This chapter presents the steps taken toward validating the prototype, and evaluating its performance. In section 9.1 the test scenario is presented, including the goals of the tests and how the testbeds were designed. Section 9.2 presents an overview of how the testbeds were setup. The detailed steps for each individual test are defined in section 9.3. The results are presented in section 9.4. Finally, section 9.6 presents the concluding thoughts based on the obtained results.

9.1 Test Scenario

The goal of these tests is to evaluate how the prototype behaves when deployed. As such, the testbed must be able to simulate subscribers that connect to the vBNG and network services that serve the subscribers. It must also provide a way to generate traffic between the subscribers and the service.

Considering the requirements, the testbed was designed to use 3 machines: one to simulate the subscribers, one to host the MicroStack cloud infrastructure and one to host the OSM deployment. For the network service it was decided that it would consist of a instance deployed into the provider network, hosting iperf servers to which the subscribers would connect. Figure 9.1 demonstrates the topology.

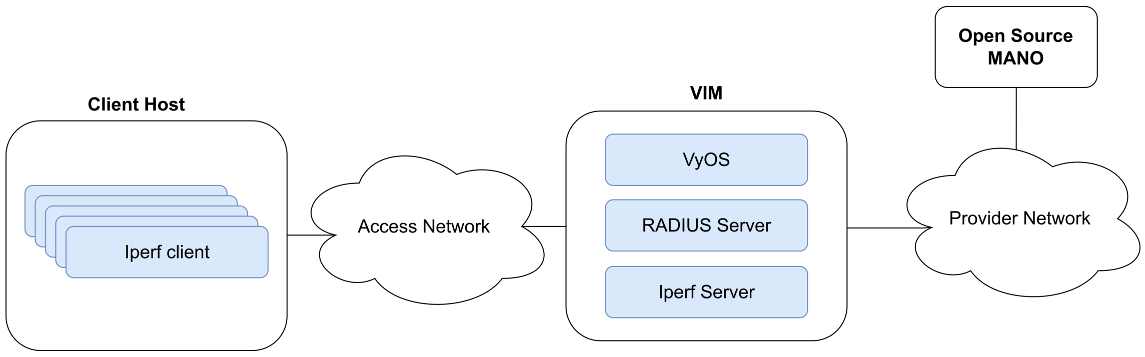


Figure 9.1: Simplified view of the testbed

Early tests revealed lower performance than expected, while the exact nature of this behaviour could not be determined, one of the factors hypothesised to cause this was the the nested virtualization happening. Due to the resources available, the author was deploying the testbed into a VMware ESXi environment, including MicroStack which itself would virtualize the functions deployed. To mitigate the issues it was decided that a second testbed would be setup, emulating the VIM's network directly in the VMware ESXi. The author understands this choice might compromise the results, but it was taken as it approximates the environment to what a real scenario would be, without the nested virtualization. Figure 9.2 demonstrates the topology.

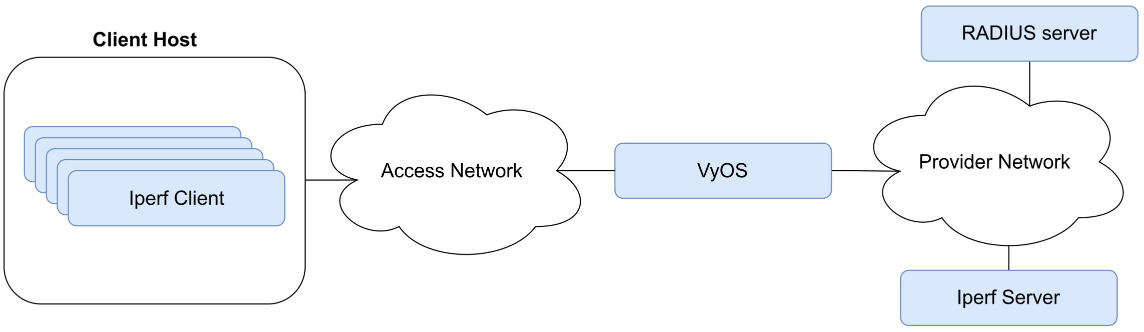


Figure 9.2: Simplified view of the second testbed

9.2 Testbed Setup

In this section presents how each of the testbeds were setup.

9.2.1 First Testbed - MicroStack

For the deployment of OSM and MicroStack please refer to sections 8.3 and 8.2 respectively.

Subscribers

To simulate the subscribers, a Ubuntu 22.04.2 LTS machine was setup. This machine will be used to simulate the subscribers establishing the PPPoE connections with the bng and to generate traffic. Therefore, it requires a connection to the same network as the prototype's access network. This machine is equipped with 8 vCPUs, 32GB of memory and 25GB of storage.

To establish PPP connections in Ubuntu, the 'ppp' is necessary as this package includes 'pppd' [Mackerras], the component that establishes and manages the connections. Additionally, the package 'pppoeconf' was used to generate a base configuration file to used in the scripts created to automate the process of establishing connections. A basic configuration file is presented in Listing 9.1, these files are typically located in the `/etc/ppp/peers/` directory.

Listing 9.1: Sample pppoed configuration file

```
hide-password
noauth
persist
plugin rp-pppoe.so
usepeerdns
```

To simulate multiple subscribers, multiple PPP connections are necessary, which isn't a common use case. So, a few extra steps are necessary to the success of the connections.

Firstly, a virtual network card is needed for each subscriber, as using the same for all of them would create MAC Address conflicts. To do this, the guide presented in [SteveYi, 2021] is adapted into a generic script that creates multiple virtual network cards. Secondly,a configuration file is needed. For that we use the one 'pppoeconf' generated, adding the missing information (network interface and username). Thirdly, the configuration file does not include the password, those are stored in the file `/etc/ppp/pap-secrets`, so the script appends the password to that file if needed (it is assumed the user credentials were previously inserted into the RADIUS database). Listing 9.2 presents the finalised script.

Listing 9.2: Script to configure the connections

```
#!/bin/bash

cd /etc/ppp/peers/

for (( i=1; i<=$1; i++ )) do
    #add config file for each connection
    cp example subscriber$i
    echo "nic-wan$i" >> subscriber$i
    echo "user \"user$i\"" >> subscriber$i

    #add password so we can initiate pppoe connection
    if ! sudo grep -q "user$i\"" "/etc/ppp/pap-secrets";
       then
        echo "\"user$i\" * \"password$i\"" >> /etc/ppp/pap-
            secrets
    fi

    #create virtual network card for user
    if ! ip link show wan$i >/dev/null 2>&1; then
        ip link add link ens192 name wan$i type macvlan
    fi
    ip link set dev wan$i up

done
```

At this stage, the PPPoE connections can be started manually or using a script as the one presented in Listing 9.3. Despite this, not all preparations are done as the routing rules are missing, due to them causing conflicts (duplicates of the same route).

Listing 9.3: Script to establish PPP connections

```
#!/bin/bash
poff -a #stop all existing pppoe clients

for (( i=1; i<=$1; i++ )) do
    pon subscriber$i
done
```

To fix the routing problem, the author used Policy-Based Routing (PBR), with network namespaces being a possible alternative, PBR essentially being a technique to select the path that network traffic takes based on predefined policies or criteria, in this case the criteria being the source.

Firstly, the routing table has to be created by defining it in the `/etc/iproute2/rt_tables` file. One way to automate the process is to grab the interface's name and check whether a table with that name exists, and create one if it doesnt. take note to prepend the number obtained as the first PPP interface is always "ppp0" and table 0 being the default routing table. Secondly, the new table must be populated, with the gateway and the default routing rule. And finally, the new routing table is assigned the desired source IP. For convenience, this process was automated in a script as seen in listing 9.4. This script is located in the `/etc/ppp/ip-up.d/` directory, such that is run when a new PPP interface comes up.

Listing 9.4: Script to setup PBR for the ppp interfaces

```
#!/bin/bash
#     Arg   Name                          Example
#     $1    Interface name                ppp0
#     $4    Local IP number               12.34.56.78
#     $5    Peer  IP number               12.34.56.99

# Get the name of the interface
interface=$1

# Extract the interface name from the full interface string
interface_name=$(basename "$interface")

# Check if the interface starts with "ppp"
if [[ "$interface_name" == ppp* ]]; then

    # Get the interface number
    interface_number=$(echo "$interface_name" | sed 's/ppp
       //')

    # Check if the routing table exists
    if ! ip route show table $interface_name &> /dev/null;
```

```
        then
         # Create the routing table
         # add a 1 to the beginning because the first ppp
            interface has number 0 and the routing table 0
            is the default one
         echo "1$interface_number        $interface_name" |
            sudo tee -a /etc/iproute2/rt_tables
    fi

    ip route flush table $1

    # Add a default route via the interface to the table
    ip route add $5 dev $1 table $1
    ip route add default via $5 dev $1 table $1

    #assign routing table
    ip rule add from $4 table $1
fi
```

9.2.2 Second Testbed - Non-nested Virtualization

The majority setup of the second testbed is simply done by deploying VMs with the configurations defined in the VNFDs (see sections 8.4.2 and 8.5.2). As such only the differences from those configurations will be detailed in this section. It should be noted, the configuration of the subscribers will not be mentioned as it is the exact same as in the previous testbed.

Network

To simulate the intended scenario in the VMware ESXi 3 virtual switches were used, one for the access network, one for the provider network and a third one for management. The virtual switch used for the access network must have both promiscuous mode and forged transmits enabled to allow the establishment of the PPPoE connections.

Prototype

The VyOS VM is created according to the specifications of the VNFD, with one interface connected to each of the switches mentioned before. It is then configured using the commands present in the cloud-init file (see Appendix B), with the distinction of the IP address of the provider network interface having to be configured manually (as DHCP was not configured). The command for this configuration is can be seen is listing 9.5.

Similarly, RADIUS was deployed by manually applying the configurations spec-

Listing 9.5: Setting a static IP address in VyOS

```
config
set interfaces ethernet eth1 address 192.168.222.1/24
commit
```

Listing 9.6: Setting a static IP address in RADIUS server

```
sudo ip addr add 192.168.222.3/24 dev ens192
```

ified in the corresponding VNFD (see Appendix C), with a interface connected to both the management and provider switches. Just as with VyOS, the distinction is that the IP address of the provider network must be manually set, as seen in listing 9.6.

9.2.3 Difficulties and Limitations

Initial tests in the first testbed revealed lower performance than expected, for example a maximum throughput of 600Mbps. While no concrete cause could be identified, during troubleshooting various aspects were noted that could be causing the issue:

- **Nested Virtualization**- As mentioned before, the testbed was deployed in a VMware ESXi environment, which lead to nested virtualization that could be impacting the performance.

- **MicroStack**- MicroStack was created to allow developers to quickly create an environment where they could develop their solutions, it was not built for performance. Ideally a full OpenStack environment would be used for such tests but it was deemed too complex.

- **Kernel-based Virtual Machine (KVM)**- Although during deployment MicroStack detected and reported the use of KVMs, hardware assisted virtualization, when deploying the VNFs it was observed the tecnology was not being used, analysing the logs revealed that libvirt, the tool used by MicroStack to manage the virtualization, reported that KVMs were not supported.

Furthermore, despite the prototype not having QoS capabilities, each subscriber uses multiple traffic streams to simulate the various services used, those being Internet, IPTV and VoIP. This may cause the traffic streams for IPTV and VoIP to suffer losses of bandwidth that may not happen in a real scenario, as these streams are typically given priority over Internet traffic.

9.3 Test Definition

In this section the tests conducted to validate the prototype are specified.

To validate the prototype, acceptance tests were conducted to verify that the functional requirements are properly met. An excerpt of the tests conducted is visible in table 9.1, with the full list being present in Appendix E.

Table 9.1: Validation tests (Excerpt)

ID	Requirement	Details	Result
Test 4	FR-002	**Description:** Establish PPPoE connection with correct credentials **Expected Behaviour:** PPPoE connection is established **Observed Behaviour:** PPPoE connection is established	Pass
Test 12	FR-008 and FR-001	**Description:** As a subscriber, attempt to access the internet **Expected Behaviour:** Subscriber can access the internet **Observed Behaviour:** Subscriber can access the internet	Pass

The following subsections will detail how each performance test was conducted. To avoid repetition, a few notes should be kept in mind:

- All tests conducted assume the users were registered beforehand.
- The tests are conducted in the second testbed, which doesn't use nested virtualization.
- Each test is executed with 15 repetitions.
- For automation purposes, *ssh* is used to remotely execute commands and collect metrics from the vBNG. The impact of which will be disregarded as it is minimal.

9.3.1 Test 1- Bandwidth

This test aims to measure the number of subscribers connected after the bandwidth available to each subscriber. To this end the following procedure was defined, for an number of subscribers N:

1. Setup the subscriber rate limits in the RADIUS' database.
2. Setup N virtual network interfaces.
3. Setup the configuration file of each PPPoE subscriber.

4. Start 2N *iperf* servers, as each subscriber connects to two servers.

5. Initiate the PPPoE clients.

6. Wait 30 seconds so that the PPPoE connections are fully configured.

7. Initiate 2N *iperf* clients, in reverse mode (the reason will be explained later), saving the outputs to files for later analysis.

Based on the reference subscriber model showcased in table 9.2, the following properties were defined for the simulated subscribers:

- A subscriber is subject to a rate limit of 100Mbps for both upstream and downstream data transmission. This is defined in RADIUS' database, specifically the *radreply* table. Based on [Altice].

- A subscriber has an *iperf*-generated User Datagram Protocol (UDP) traffic stream, limited to 20Mbps, to simulate IPTV and VoIP traffic.

- A subscriber has an *iperf*-generated Transmission Control Protocol (TCP) traffic stream, limited to 100Mbps to use the remaining bandwidth, to simulate internet traffic.

- Both *iperf* processes are set to *Reverse* mode, meaning the server sends traffic to the subscriber, as this is more representative of a real scenario.

Table 9.2: Subscriber Model

Service	**Bitrate (Mbps)**
Internet	10.0
Telephony	1.0
SDTV 2 channels (MPEG-4)	3.0
HDTV 2 channels (MPEG-4)	16.0
Total	30

The test will be conducted in ten variants, each with a different number of subscribers connecting to the vBNG. These variations are: 1, 10, 20, 25, 30, 40, 50, 100, 150, 200. Each iteration of the variations will take three minutes, with an extra five seconds added, as the first five are ignored to account for TCP slow start, that causes *iperf* to report rates beyond the limits of the connection.

9.3.2 Test 2- Resource Utilization

This test aims to measure the CPU and memory utilization of the prototype. To achieve this, these resources were monitored using the tool *top* while executing the previous tests using the function shown in listing 9.7.

Listing 9.7: Monitoring the prototype's resource utilization

```
function resource_utilization {
#wait for 5 seconds to keep consistent with iperf
sleep 5

#monitor resource for N seconds, provided by the parameter,
    for each cpu core
ssh vyos_baremetal "sudo top -1 -b -d 1 -n $1"
}
```

9.4 Test Results

This section presents the data collected from the defined tests.

9.4.1 Test 1- Bandwidth

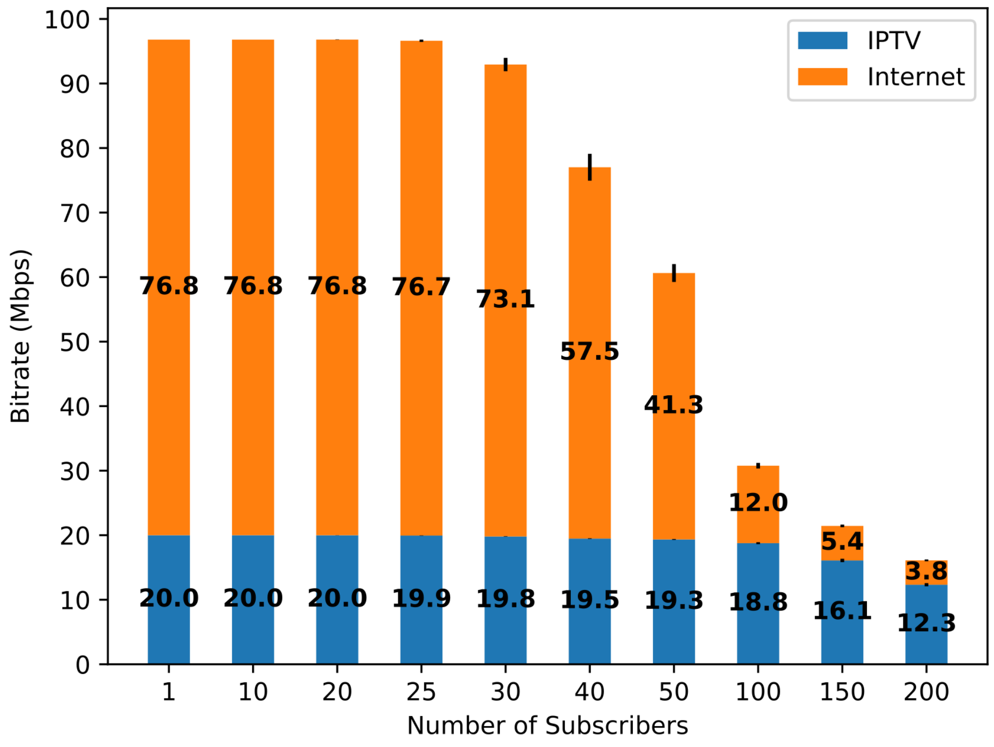


Figure 9.3: IPTV and Internet bit rate variation with changing number of subscribers

Figure 9.3 presents the average throughput of all subscribers. It is visible that the prototype can provide a stable 100Mbps of throughput to around 25 subscribers. These results also show that the prototype should support around 100 reference model subscribers. Although no QoS control is active, it is visible that the TCP stream has its bit rate lowered at a faster rate than UDP stream. This can be explained by TCP's congestion control and its bigger overhead.

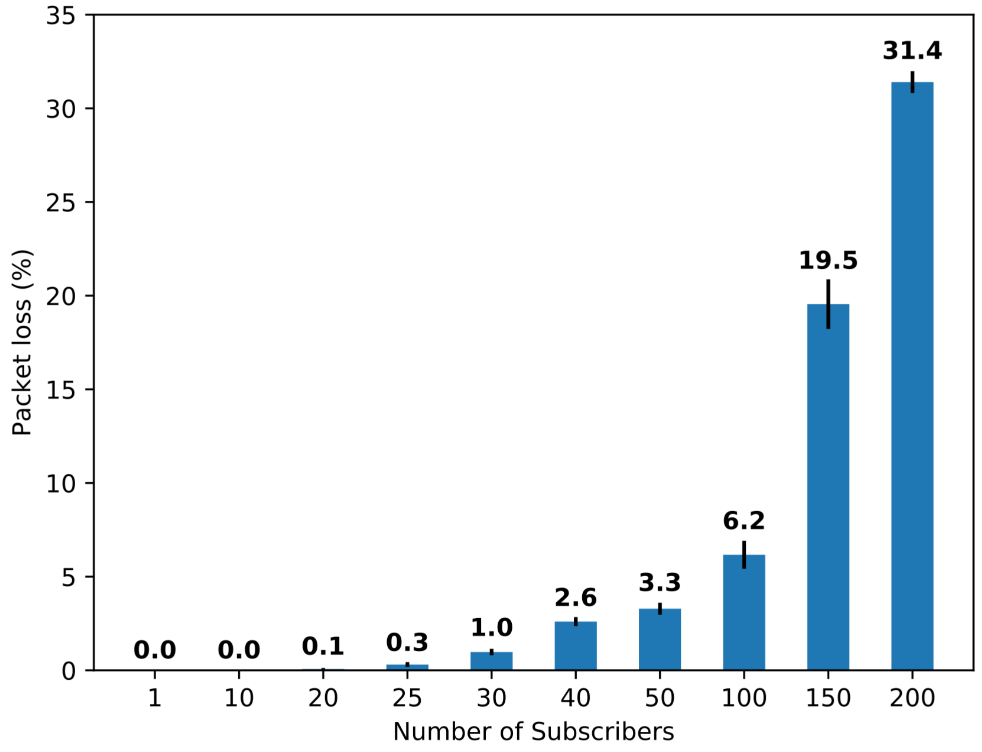


Figure 9.4: UDP packet loss with changing number of subscribers

Figure 9.4 presents the average UDP packet loss as the number of subscribers increases. When more than 30 subscribers are connected, this value becomes bigger than 1%, which is not acceptable [Biljan, 2023]. These results can be justified by the lack of QoS control functionality.

Finally, from these results it can be extrapolated that the prototype supports around 3Gbps of throughput.

9.4.2 Test 2- Resource Utilization

Figure 9.5 presents the average of total Central Processing Unit (CPU) usage as the number of subscribers changes. The results show that the CPU usage is at its peak with 25 subscribers, while also presenting a big standard deviation, an unexpected result. While the concrete reason could not be ascertained, the fact that this peak coincides with the number of subscribers where the vBNG no longer is able to maintain a full 100Mbps connection, it leads the author to believe that from this point a bottleneck in some other component of the vBNG exists, but further testing would be necessary to diagnose this behaviour.

Figure 9.6 presents the usage of individual cores as the number of subscribers

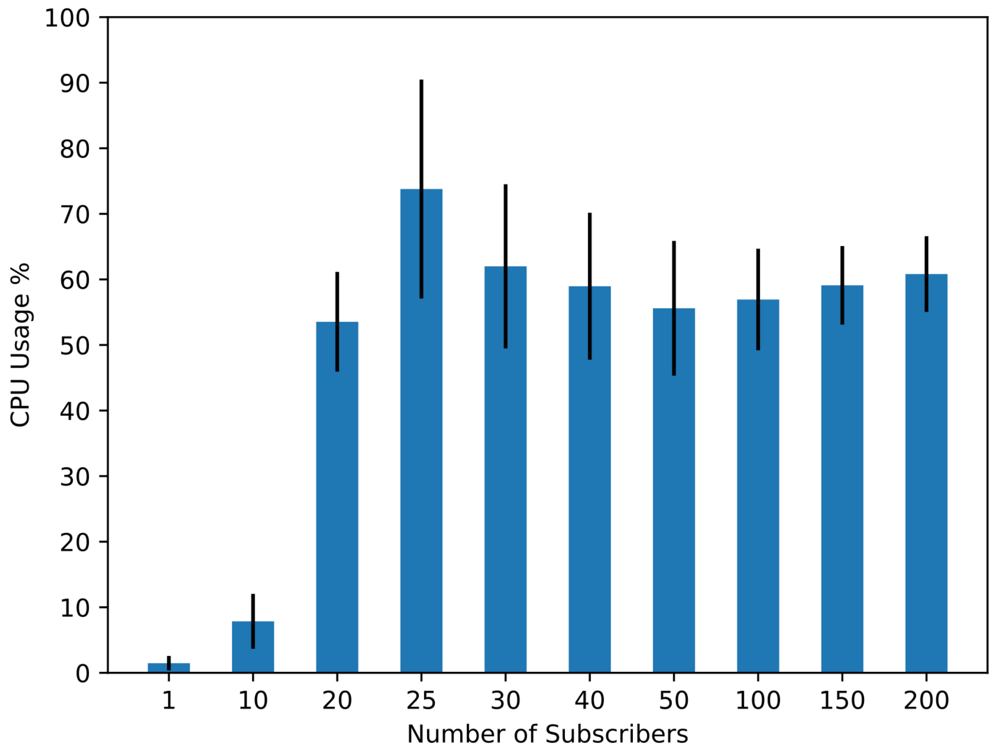


Figure 9.5: Total CPU utilization variation with changing number of subscribers

changes. It is visible that in less demanding tasks, the usage percentage is similar between both cores, meanwhile on more demanding tasks, there is a clear preference of one core over the other. Analysing the log files it is also visible that the majority of CPU time is being taken by the process *ksoftirqd*, which is a "per-cpu kernel thread that runs when the machine is under heavy soft-interrupt load" [Gohr]. This leads the author to formulate the following hypothesis: in the less demanding scenarios, the kernel scheduler balances the task of handling the interrupts between the two available CPUs, but as the the number of interrupts increases and this form of handling becomes ineffective, the scheduler assigns as many interrupts as possible to one of the CPUs to optimize performance, by reducing the need to fetch data from memory into cache.

Figure 9.7 presents the average memory used in each test. It shows a consistent use of memory resources, and as such it can be concluded that it was not a bottleneck in the system. It also shows that slightly lowering the amount of allocated memory from 2GB should not cause performance issues.

9.5 Requirement Validation Summary

In this section the results of the requirement validation are summarized, based on the validation and performance tests.

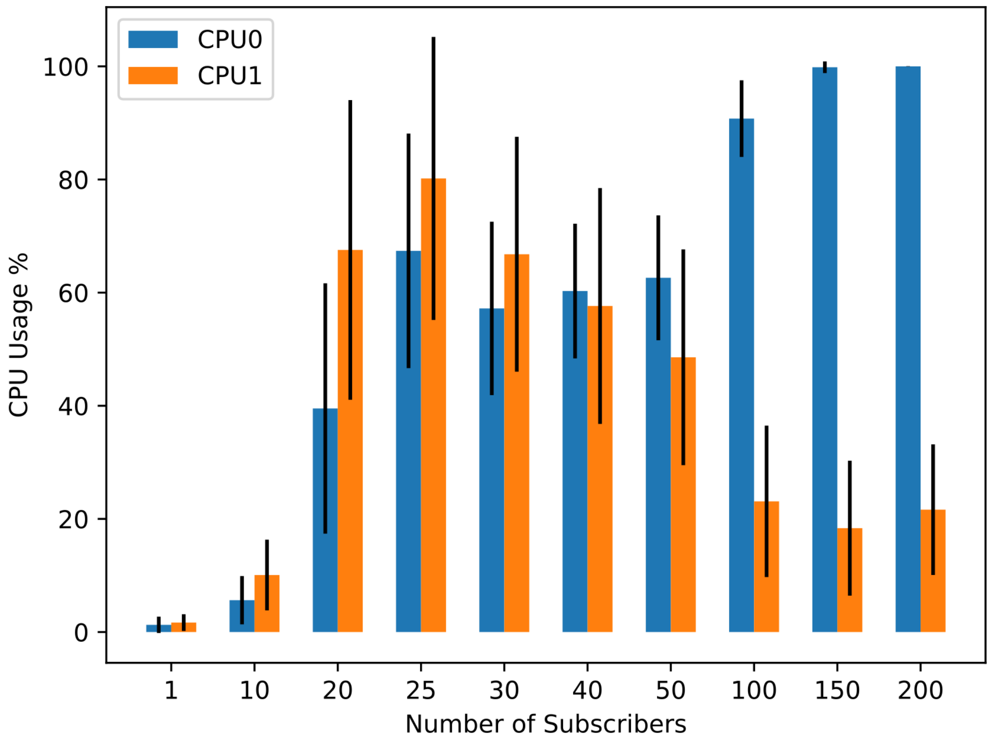


Figure 9.6: CPU utilization variation (per core) with changing number of subscribers

9.5.1 Functional Requirements

Table 9.3 summarizes the results for the functional requirements. Most requirements were met, with the exception of FR-004 and FR-008. This was due to time constraints that would not allow their validation in a timely manner, as the validation of both these requirements would require extra components for the test bed. It should be noted however, that these requirements had been assigned the *Could Have* priority, the lowest, meaning their implementation is not at all critical to the success of the project and could be implemented in future iterations.

Table 9.3: vBNG functional requirements results

ID	**Requirement**	**Result**
FR-001	Packet Forwarding	PASS
FR-002	PPPoE	PASS
FR-003	IP Address Assignment	PASS
FR-004	Multicast	FAIL
FR-005	Rate Limiting	PASS
FR-006	User Authentication	PASS
FR-007	Access Control	PASS
FR-008	QoS Control	FAIL

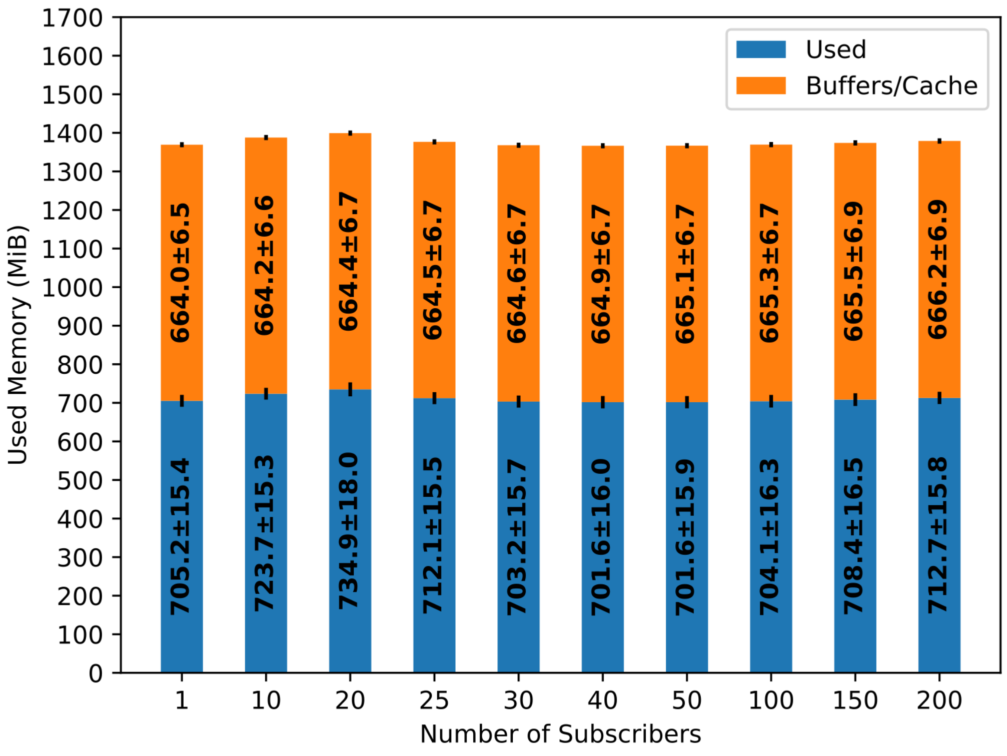


Figure 9.7: Memory utilization with changing number of subscribers

9.5.2 Non-Functional Requirements

Table 9.4 summarizes the results for the non functional requirements.

Table 9.4: vBNG non functional requirements results

ID	**Requirement**	**Result**
NFR-001	The vBNG should efficiently handle high throughput demands during peak load conditions.	PASS
NFR-002	The vBNG should effectively manage concurrent user connections.	PASS
NFR-003	The vBNG should minimize packet loss.	FAIL
NFR-004	The vBNG should contribute to low latency in network communication.	UNTESTED
NFR-004	The vBNG should manage memory usage efficiently.	PASS
NFR-005	The vBNG should optimize CPU usage for efficient performance.	PASS
NFR-006	The vBNG should ensure establishment of subscriber sessions.	UNTESTED

9.6 Conclusion

The goal of these tests was to analyse the behaviour of the proposed solution in a high fidelity scenario, one that approximates a real-world scenario as closely as feasibly possible. As the threshold of success was met it can be concluded that the thesis was a success.

Due to various circumstances more data could not be gathered, the obtained results already show potential in the proposed solution. Continuing to leverage the robust foundation laid by the VyOS project, a project that is still in active development, with NFV platforms like OpenStack should prove a viable solution for a virtualized BNG.

As a final note, the author is aware that these tests may be insufficient to analyse the solution, the following is a list of planned tests that due to constraints could not be conducted, and could be conducted on a future iteration of the work:

- Average time needed to establish a PPPoE session.
- Varying number network interfaces, for redundancy and load balancing.
- Vary number of CPUs to better understand the CPU usage balancing.
- Measure latency with varying number of concurrent subscribers.
- Diagnose the cause for CPU usage at 25 subscriber being above the remaining tests.

Chapter 10

Conclusion

In this chapter, the conclusions taken are presented, as well as a brief reflection on the progress achieved and a proposal for the work to be done to a possible future iteration of the project

10.1 Conclusions

This thesis documents the efforts in studying, designing, implementing and validating a solution for a virtualized BNG, based on the concepts of SDN and NFV. This effort started with a study of the BNG and the two technologies, as well as SFC, a complementary technology, to understand what functions the solution must provide and how each of these technologies help empower the idea of a virtualized BNG. This research was followed by a study on available solutions that could be used to create the vBNG. Meanwhile a study on ONOS was conducted in order to understand how it works and if it would be be useful to this project, while also providing the author with hand-on experience in SDN. This research and experience resulted in an architecture for the platform that would support the vBNG, concluding the first semester of this project.

To kick off the semester an exploratory phase was held to address still open questions, like the tools to be used. It started by analysing the various approaches, by analysing the documentation, attempting to access the support channels and researching user guides. This research concluded that an approach not using SFC would be more viable which was reflected on the architecture, and that a solution based on OSM, MicroStack and VyOS would be the most viable. Following this, a more practical phase started, where the author deployed and experimented with OSM and MicroStack, with the help of a member of the community to understand how these two complex systems interact and learn the basics of creating and deploying VNFs. In this stage it was decided to ultimately drop the integration of an SDN controller, due to multiple difficulties and MicroStack's Neutron service providing the necessary functionality.

Once the exploratory phase was completed, a lengthy development phase was conducted. The first step was to integrate VyOS into the platform, a task that revealed more complex than expected, as a cloud image with cloud-init support is required, a tool that the author had no prior experience in. Once VyOS had been integrated into OSM and MicroStack the configuration was done, starting with the PPPoE which also revealed itself to be an issue, as connecting the VNF with the exterior was not a straight forward process at first, requiring a new OVS bridge to connect the VIM's access interface to a virtual network within the MicroStack environment. The remaining features were then configured with no major issues, except for needing to introduce an Source Network Address Translation (SNAT) into the prototype, as otherwise the packets forwarded by the VyOS would be dropped by MicroStack due to lack of flow rules.

With all but the optional functional requirements implemented, a testing phase was conducted. In terms of functional requirements, the solutions passed all the tests with the exception of the lowest priority requirements. In terms of performance however, not all tests could be conducted due to time constraints. Despite this, the obtained results showed potential, the solutions was able to handle around 100 users with acceptable bandwidth, keeping a total throughput of about 3Gbps. It can be concluded that the thresholds of success were met and therefore the thesis was a success.

Finally, this dissertation was written with the goal of documenting the process of achieving the final proof-of-concept.

10.2 Future Work

A future iteration of the prototype would include implementing the remaining functions and refine the existing ones. In summary:

- Migration to a full OpenStack without the need for nested virtualization. This could prevent some of the issues detailed in section 9.1 and provide more accurate results.
- Implement the remaining, low priority, functional requirements such as Multicast and QoS.
- Separate the RADIUS server from the subscriber database. This would provide better scalability as multiple RADIUS servers could connect to the same database for authentication.
- Database replication to improve redundancy and high-availability.
- Deployment of backup RADIUS servers, and configuration of the vBNGs to use them, improving scalability.
- Diagnose unexpected behaviours observed section 9.4.2 through more testing.
- Take better advantage of the functions provided by both MicroStack and OSM, as deadlines and the complexity of both these systems, as well as

the remaining tools, left some options unexplored that could provide better flexibility and automation of the processes. As an example, OSM's Network Slice Templates would automate the process of configuring the VIM's virtual networks.

References

Altice. Condições Específicas De Prestação Do Serviço De Acesso À Internet Em Banda Larga. https://conteudos.meo.pt/meo/Documentos/Condicoes-Utilizacao/clausulas-alteradas-anexo-II.pdf. [Online, Accessed: 2023-7-1].

arnold. Gawk - GNU Project - Free Software Foundation (FSF). https://www.gnu.org/software/gawk/, 9 2022. [Online, Accessed: 2022-11-28].

Tim De Backer. tc.sh. https://github.com/excentis/impairment-node/blob/master/tc.sh, 11 2016. [Online, Accessed: 2022-11-30].

Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and Communication Networks*, 9:5803–5833, 12 2016. ISSN 19390114. doi: 10.1002/sec.1737.

Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. Onos: towards an open, distributed sdn os. pages 1–6. ACM, 8 2014. ISBN 9781450329897. doi: 10.1145/2620728.2620744.

Roberto Bifulco, Thomas Dietz, Felipe Huici, Mohamed Ahmed, Joao Martins, Saverio Niccolini, and Hans-Joerg Kolbe. Rethinking access networks with high performance virtual software brases. pages 7–12. IEEE, 10 2013. ISBN 978-1-4799-2433-2. doi: 10.1109/EWSDN.2013.8. URL http://ieeexplore.ieee.org/document/6680551/.

Ivan Biljan. How to Build an IPTV Core Network: A Complete Digital TV Network Architecture Design Guide. https://www.uniqcast.com/blog/iptv-core-network-tutorial#:~:text=For%20example%2C%20in%20IPTV%20service,than%2030ms%20is%20considered%20acceptable., 2023. [Online, Accessed: 2023-7-1].

Canonical. MicroStack - Single-node quickstart. https://microstack.run/docs/single-node. [Online, Accessed: 2023-3-23].

Lovy Chaudhary. Scrum vs Waterfall. https://www.educba.com/scrum-vs-waterfall/. [Online, Accessed: 2022-11-30].

Damiano Ciro. Configure FreeRADIUS to use MySql on Linux. https://medium.com/codex/configure-freeradius-to-use-mysql-on-linux-95fa546cc3a7, 2022. [Online, Accessed: 2023-7-1].

Cisco. Broadband Network Gateway Configuration Guide for Cisco ASR 9000 Series Routers, IOS XR Release 7.3.x - Broadband Network Gateway Overview [Cisco ASR 9000 Series Aggregation Services Routers]. https://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k-r7-3/bng/configuration/guide/b-bng-cg-asr9000-73x/bng-overview.html, 12 2022. [Online, Accessed: 2022-12-26].

Tiago Cruz, Paulo Simões, Nuno Reis, Edmundo Monteiro, Fernando Bastos, and Alexandre Laranjeira. An architecture for virtualized home gateways. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 520–526. IEEE, 2013.

Miguel Rosado Borges de Freitas et al. Network softwarization for iacs security applications. 2018. URL http://hdl.handle.net/10316/83548.

Thomas Dietz, Roberto Bifulco, Filipe Manco, Joao Martins, Hans-Joerg Kolbe, and Felipe Huici. Enhancing the bras through virtualization. pages 1–5. IEEE, 4 2015. ISBN 978-1-4799-7899-1. doi: 10.1109/NETSOFT.2015.7116144.

Docker. Docker: Accelerated, Containerized Application Development. https://www.docker.com/. [Online, Accessed: 2022-11-30].

Mark Esler et al. OpenStack — cloud-init 22.1 documentation. https://cloudinit.readthedocs.io/en/22.1_a/topics/datasources/openstack.html. [Online, Accessed: 2023-9-3].

ETSI. OSM. https://osm.etsi.org/. [Online, Accessed: 2022-12-1].

ETSI. Network Functions Virtualization — Introductory White Paper. https://portal.etsi.org/NFV/NFV_White_Paper.pdf, 10 2012. [Online, Accessed: 2022-11-24].

ETSI. Gs nfv 002 - v1.2.1 - network functions virtualisation (nfv); architectural framework, 12 2014a. URL https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf.

ETSI. Network Functions Virtualisation (NFV); Management and Orchestration. https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf, 12 2014b. [Online, Accessed: 2022-11-24].

Rubens Figueiredo and Andreas Kassler. Bng-hal: A unified api for disaggregated bngs. pages 116–119. IEEE, 11 2021. ISBN 978-1-6654-3983-1. doi: 10.1109/NFV-SDN53031.2021.9665122.

Broadband Forum. Control and user plane separation for a disaggregated broadband network gateway (bng). Technical report, Broadband Forum, 2020.

Martin Fowler. DSL Guide. https://martinfowler.com/dsl.html, 8 2019. [Online, Accessed: 2023-1-13].

GanttProject. GanttProject: free project management tool for Windows, macOS and Linux. https://www.ganttproject.biz/. [Online, Accessed: 2022-9-29].

Pete Vander Giessen, Gabor Meszaros, and Dmitrii Shcherbakov. microstack/setup-br-ex at master - microstack - OpenDev: Free Software Needs Free Tools. https://opendev.org/x/microstack/src/branch/master/snap-overlay/bin/setup-br-ex, 2020. [Online, Accessed: 2023-8-30].

Andreas Gohr. Linux Manpages Online - man.cx manual pages. https://man.cx/ksoftirqd. [Online, Accessed: 2023-7-31].

Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53:90–97, 2 2015. ISSN 0163-6804. doi: 10.1109/MCOM.2015.7045396.

Hassan Hawilo, Abdallah Shami, Maysam Mirahmadi, and Rasool Asal. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). *IEEE Network*, 28:18–26, 11 2014. ISSN 0890-8044. doi: 10.1109/MNET.2014.6963800.

Shujun Hu, Donald E. Eastlake 3rd, Fengwei Qin, Tee Mong Chua, and Daniel Huang. The China Mobile, Huawei, and ZTE Broadband Network Gateway (BNG) Simple Control and User Plane Separation Protocol (S-CUSP). RFC 8772, May 2020. URL https://www.rfc-editor.org/info/rfc8772.

iwaseyusuke. iwaseyusuke/mininet Tags | Docker Hub. https://hub.docker.com/r/iwaseyusuke/mininet/, 3 2022. [Online, Accessed: 2022-11-30].

Sheng Jiang, Brian E. Carpenter, and Dayong Guo. An Incremental Carrier-Grade NAT (CGN) for IPv6 Transition. RFC 6264, June 2011. URL https://www.rfc-editor.org/info/rfc6264.

Eddie Kohler. Click. https://github.com/kohler/click. [Online, Accessed: 2022-12-1].

Ayaka Koshibe and Elena Olkhovskaya. Cluster Coordination - ONOS - Wiki. https://wiki.onosproject.org/display/ONOS/Cluster+Coordination, 11 2016. [Online, Accessed: 2022-12-29].

Ralf Kundel, Leonhard Nobach, Jeremias Blendin, Hans-Joerg Kolbe, Georg Schyguda, Vladimir Gurevich, Boris Koldehofe, and Ralf Steinmetz. P4-bng: Central office network functions on programmable packet pipelines. pages 1–9. IEEE, 10 2019. ISBN 978-3-903176-24-9. doi: 10.23919/CNSM46954.2019.9012666.

Ralf Kundel, Leonhard Nobach, Jeremias Blendin, Wilfried Maas, Andreas Zimber, Hans-Joerg Kolbe, Georg Schyguda, Vladimir Gurevich, Rhaban Hark, Boris Koldehofe, and Ralf Steinmetz. Openbng: Central office network functions on programmable data plane hardware. *International Journal of Network Management*, 31(1):e2134, 2021. doi: https://doi.org/10.1002/nem.2134. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2134. e2134 nem.2134.

Bob Lantz. OpenFlow - ONOS - Wiki. https://wiki.onosproject.org/display/ONOS/OpenFlow, 1 2017a. [Online, Accessed: 2022-10-17].

Bob Lantz. Introduction to the ONOS APIs - ONOS - Wiki. https://wiki.onosproject.org/display/ONOS/Introduction+to+the+ONOS+APIs, 2 2017b. [Online, Accessed: 2022-11-21].

Pingping Lin and Jonathan Hart. Virtual BNG. https://wiki.onosproject.org/display/ONOS/Virtual+BNG, 10 2015. [Online, Accessed: 2022-12-1].

Paul Mackerras. Ubuntu Manpage: pppd - Point-to-Point Protocol Daemon. https://manpages.ubuntu.com/manpages/focal/en/man8/pppd.8.html. [Online, Accessed: 2023-7-7].

Mininet. Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet. https://mininet.org/. [Online, Accessed: 2022-11-28].

NEC Laboratories Europe. ClickOS - Systems and Machine Learning. http://sysml.neclab.eu/projects/clickos/. [Online, Accessed: 2022-12-1].

Gianfranco Nencioni, Bjarne Helvik, and Poul Heegaard. Implementing the availability model of a software-defined backbone network in möbius, 11 2017.

ONAP. Offline Installer - Installation Guide — onap master documentation. https://docs.onap.org/projects/onap-oom-offline-installer/en/latest/InstallGuide.html. [Online, Accessed: 2023-7-31].

ONF. P4 – Language Consortium. https://p4.org/. [Online, Accessed: 2023-1-13].

ONF. Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions. https://opennetworking.org/onos/, 2022. [Online, Accessed: 2022-9-30].

OpenDayLight. Service Function Chaining User Guide — ODL SFC master documentation. https://docs.opendaylight.org/projects/sfc/en/latest/user-guide.html#sfc-classifier-user-guide. [Online, Accessed: 2023-8-31].

OpenInfra. OpenStack Releases: Mitaka. https://releases.openstack.org/mitaka/index.html, a. [Online, Accessed: 2022-11-30].

OpenInfra. Open Source Cloud Computing Infrastructure - OpenStack. https://www.openstack.org/, b. [Online, Accessed: 2022-11-30].

OpenInfra. Tacker - OpenStack. https://wiki.openstack.org/wiki/Tacker, c. [Online, Accessed: 2022-11-30].

opennetworkinglab. Mastership Load Balancer. https://github.com/opennetworkinglab/onos/tree/master/apps/mlb, 10 2018. [Online, Accessed: 2022-11-21].

OpenStack. Welcome to Glance's documentation! — glance 27.0.0.0b3.dev24 documentation. `https://docs.openstack.org/keystone/latest/`, 2019a. [Online, Accessed: 2023-9-3].

OpenStack. Horizon: The OpenStack Dashboard Project — horizon 23.2.1.dev21 documentation. `https://docs.openstack.org/horizon/latest/`, 2019b. [Online, Accessed: 2023-9-3].

OpenStack. Keystone, the OpenStack Identity Service — keystone 23.1.0.dev62 documentation. `https://docs.openstack.org/keystone/latest/`, 2019c. [Online, Accessed: 2023-9-3].

OpenStack. OpenStack Docs: Hypervisors. `https://docs.openstack.org/ocata/config-reference/compute/hypervisors.html`, 8 2019. [Online, Accessed: 2022-12-18].

OpenStack. Metadata service — nova 27.1.0.dev168 documentation. `https://docs.openstack.org/nova/latest/admin/metadata-service.html`, 2020a. [Online, Accessed: 2023-9-3].

OpenStack. Welcome to Neutron's documentation! — Neutron 23.0.0.0b4.dev16 documentation. `https://docs.openstack.org/neutron/latest/`, 2020b. [Online, Accessed: 2023-9-3].

OpenStack. OpenStack Docs: Networking (neutron) concepts. `https://docs.openstack.org/neutron/rocky/install/concepts.html`, 2022a. [Online, Accessed: 2023-9-3].

OpenStack. OpenStack Docs: Service function chaining. `https://docs.openstack.org/neutron/queens/admin/config-sfc.html`, 1 2022b. [Online, Accessed: 2022-12-1].

OpenStack. OpenStack Compute (nova) — nova 27.1.0.dev168 documentation. `https://docs.openstack.org/nova/latest/`, 2023a. [Online, Accessed: 2023-9-3].

OpenStack. Provider network — Installation Guide documentation. `https://docs.openstack.org/install-guide/launch-instance-networks-provider.html`, 2023b. [Online, Accessed: 2023-9-3].

Tomasz Osiński, Mateusz Kossakowski, Mateusz Pawlik, Jan Palimąka, Michał Sala, and Halina Tarasiuk. Unleashing the performance of virtual bng by offloading data plane to a programmable asic. In *Proceedings of the 3rd P4 Workshop in Europe*, EuroP4'20, page 54–55, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381819. doi: 10.1145/3426744.3431325. URL `https://doi.org/10.1145/3426744.3431325`.

OSM. OSM Packages. `https://osm.etsi.org/gitlab/vnf-onboarding/osm-packages`, a. [Online, Accessed: 2023-7-1].

OSM. ANNEX 2: Reference of OSM Client commands and library. https://osm.etsi.org/docs/user-guide/latest/10-osm-client-commands-reference.html, b. [Online, Accessed: 2023-7-1].

OSM. Installing OSM. https://osm.etsi.org/docs/user-guide/latest/03-installing-osm.html, c. [Online, Accessed: 2023-7-7].

OSM. Advanced instantiation: using instantiation parameters. https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html#advanced-instantiation-using-instantiation-parameters, d. [Online, Accessed: 2023-7-22].

OSM. 4. How to Set Up Virtual Infrastructure Managers (VIMs) — Open Source MANO documentation. https://osm.etsi.org/docs/user-guide/latest/04-vim-setup.html, e. [Online, Accessed: 2023-7-1].

James Page, Pete Vander Giessen, Peter Matulis, Zuul, Nikolay Vinogradov, and Corey Bryant. microstack/README.md at master - microstack - OpenDev: Free Software Needs Free Tools. https://opendev.org/x/microstack/src/branch/master/README.md, 2020. [Online, Accessed: 2023-8-30].

Jorge Proença, Tiago Cruz, Paulo Simões, and Edmundo Monteiro. Virtualization of residential gateways: A comprehensive survey, 4 2019. ISSN 1553877X.

P. Quinn and T. Nadeau. Problem Statement for Service Function Chaining. RFC 7498, RFC Editor, April 2015. URL http://www.rfc-editor.org/rfc/rfc7498.txt. http://www.rfc-editor.org/rfc/rfc7498.txt.

P. Quinn, U. Elzur, and C. Pignataro. Network Service Header (NSH). RFC 8300, RFC Editor, January 2018.

Francisco-Javier Ramon Salguero et al. README.md · master · osm / NG-UI · GitLab. https://osm.etsi.org/docs/developer-guide/02-developer-how-to.html, 2020. [Online, Accessed: 2023-9-3].

Ken Schwaber and Jeff Sutherland. *Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. 11 2020. URL https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

Gaganpreet Singh. Common protocols used in Southbound APIs. https://www.telecomtrainer.com/common-protocols-used-in-southbound-apis/, 2023. [Online, Accessed: 2023-9-3].

SteveYi. Implementing Virtual WAN Multi-PPPoe on Linux. https://blog.steveyi.net/en/posts/multi-pppoe-in-linux/, 2021. [Online, Accessed: 2023-7-7].

Docker Team. registry - Official Image | Docker Hub. https://hub.docker.com/_/registry, 11 2022. [Online, Accessed: 2022-11-30].

The Linux Foundation. Home - ONAP. https://www.onap.org/. [Online, Accessed: 2022-12-1].

The Linux Foundation. networking:iproute2 [Wiki]. https://wiki.linuxfoundation.org/networking/iproute2, 1 2022. [Online, Accessed: 2022-11-30].

Mary Thengvall and Ain Indermitte. CORD - Community - Confluence. https://wiki.opennetworking.org/display/COM/CORD, 8 2021a. [Online, Accessed: 2022-12-1].

Mary Thengvall and Ain Indermitte. VyOS – Open source router and firewall platform. https://vyos.io/use-cases/bras, 8 2021b. [Online, Accessed: 2022-12-1].

Thomas Vachuska. Basic ONOS Tutorial - ONOS - Wiki. https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial, 2 2019. [Online, Accessed: 2022-11-28].

VMWare. What is Software-Defined Networking (SDN)? | VMware Glossary. https://www.vmware.com/topics/glossary/content/software-defined-networking.html, 2022. [Online, Accessed: 2022-10-17].

Vyos. vyos-vm-images. https://github.com/vyos/vyos-vm-images, a. [Online, Accessed: 2023-7-31].

Vyos. Build VyOS — VyOS 1.3.x (equuleus) documentation. https://docs.vyos.io/en/equuleus/contributing/build-vyos.html, b. [Online, Accessed: 2023-7-31].

VyOS. VyOS cloud-init — VyOS 1.3.x (equuleus) documentation. https://docs.vyos.io/en/equuleus/automation/cloud-init.html, a. [Online, Accessed: 2023-7-1].

VyOS. Command Scripting — VyOS 1.3.x (equuleus) documentation. https://docs.vyos.io/en/equuleus/automation/command-scripting.html, b. [Online, Accessed: 2023-7-1].

Xilinx. What is an FPGA? Field Programmable Gate Array. https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Online, Accessed: 2023-1-13].

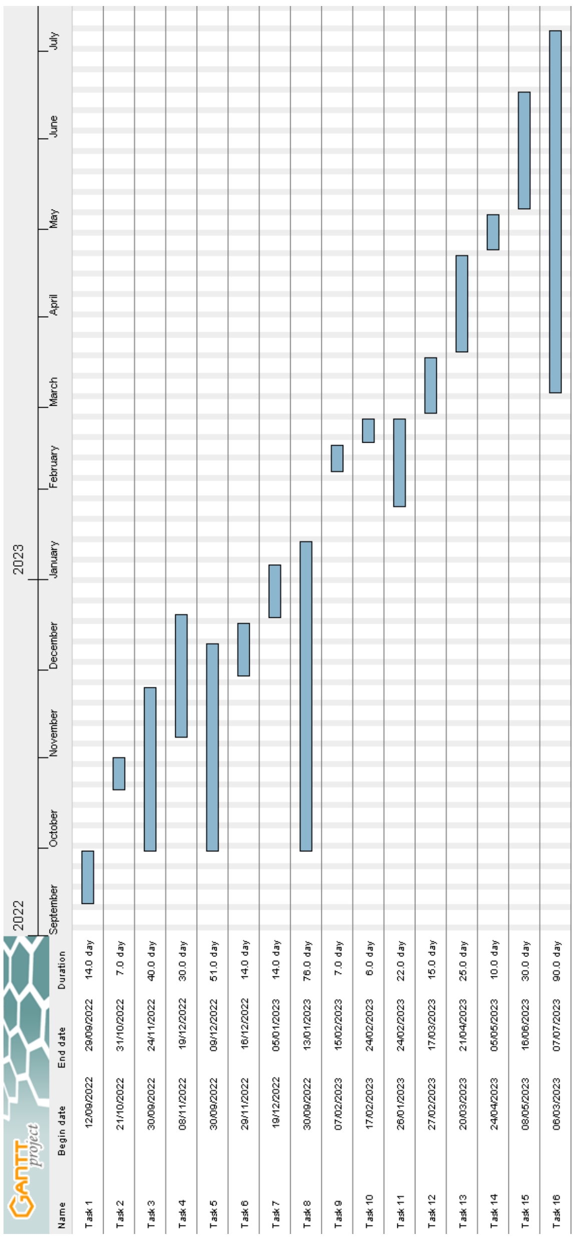
Bo Yi, Xingwei Wang, Keqin Li, Sajal k. Das, and Min Huang. A comprehensive survey of network function virtualization. *Computer Networks*, 133:212–262, 3 2018. ISSN 13891286. doi: 10.1016/j.comnet.2018.01.021.

Eder Ollora Zaballa. Automating cluster creation. https://wiki.onosproject.org/display/ONOS/Automating+cluster+creation, 10 2020. [Online, Accessed: 2022-11-30].

Appendices

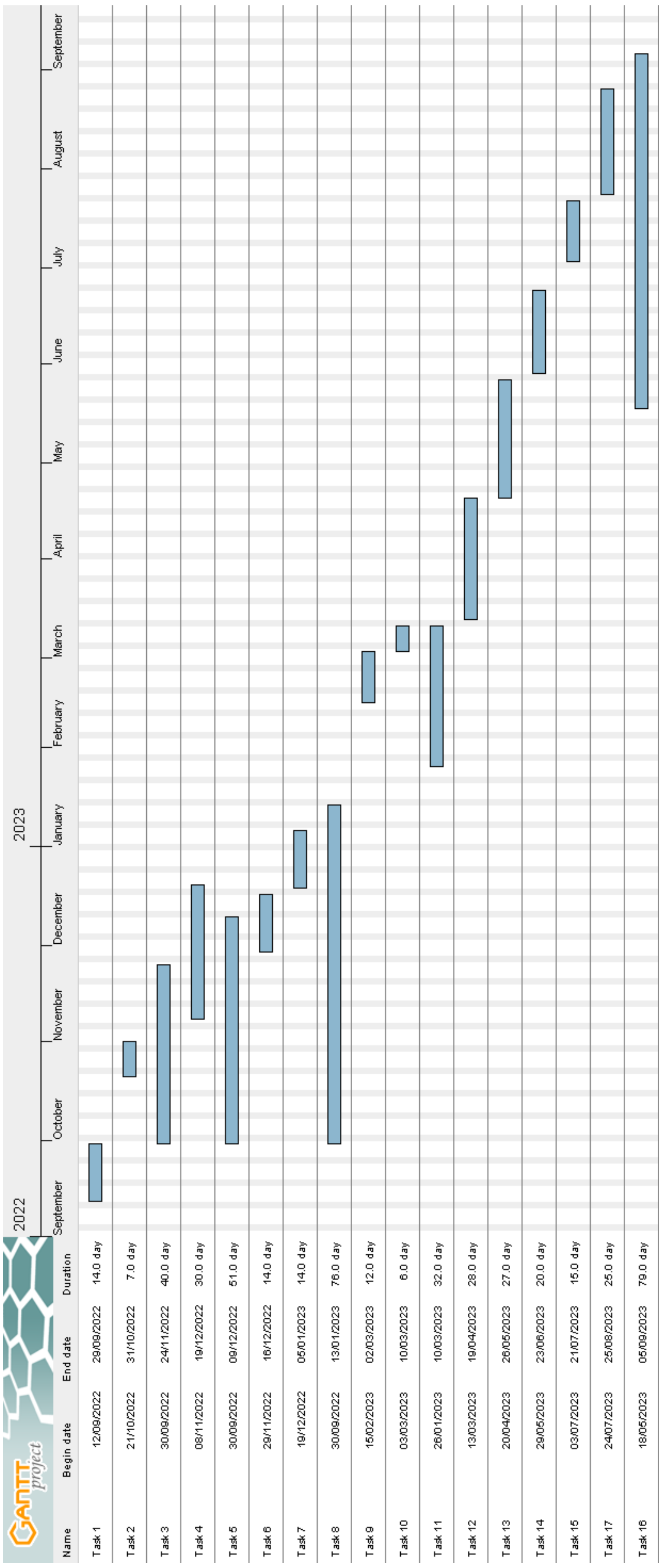
Appendix A

Work Timeline Gantt Chart



Name	Begin date	End date	Duration
Task 1	12/09/2022	29/09/2022	14.0 day
Task 2	21/10/2022	31/10/2022	7.0 day
Task 3	30/09/2022	24/11/2022	40.0 day
Task 4	08/11/2022	19/12/2022	30.0 day
Task 5	30/09/2022	09/12/2022	51.0 day
Task 6	29/11/2022	16/12/2022	14.0 day
Task 7	19/12/2022	05/01/2023	14.0 day
Task 8	30/09/2022	13/01/2023	76.0 day
Task 9	07/02/2023	15/02/2023	7.0 day
Task 10	17/02/2023	24/02/2023	6.0 day
Task 11	26/01/2023	24/02/2023	22.0 day
Task 12	27/02/2023	17/03/2023	15.0 day
Task 13	20/03/2023	21/04/2023	25.0 day
Task 14	24/04/2023	05/05/2023	10.0 day
Task 15	08/05/2023	16/06/2023	30.0 day
Task 16	06/03/2023	07/07/2023	90.0 day

Figure A.1: Planned Work Timeline



Name	Begin date	End date	Duration
Task 1	12/09/2022	29/09/2022	14.0 day
Task 2	21/10/2022	31/10/2022	7.0 day
Task 3	30/09/2022	24/11/2022	40.0 day
Task 4	08/11/2022	19/12/2022	30.0 day
Task 5	30/09/2022	09/12/2022	51.0 day
Task 6	29/11/2022	16/12/2022	14.0 day
Task 7	19/12/2022	05/01/2023	14.0 day
Task 8	30/09/2022	13/01/2023	76.0 day
Task 9	15/02/2023	02/03/2023	12.0 day
Task 10	03/03/2023	10/03/2023	6.0 day
Task 11	26/01/2023	10/03/2023	32.0 day
Task 12	13/03/2023	19/04/2023	28.0 day
Task 13	20/04/2023	26/05/2023	27.0 day
Task 14	29/05/2023	23/06/2023	20.0 day
Task 15	03/07/2023	21/07/2023	15.0 day
Task 17	24/07/2023	25/08/2023	25.0 day
Task 16	18/05/2023	05/09/2023	79.0 day

Figure A.2: Actual Work Timeline

Appendix B

VyOS Configuration Files

Listing B.1: VyOS VNF descriptor file

```
vnfd:
  description: A virtual BNG
  ext-cpd:
  - id: vnf-mgmt-ext
    int-cpd:
      cpd: vdu-eth0-int
      vdu-id: vyos-VM
  - id: vnf-internal-ext
    int-cpd:
      cpd: vdu-eth1-int
      vdu-id: vyos-VM
  - id: vnf-external-ext
    int-cpd:
      cpd: vdu-eth2-int
      vdu-id: vyos-VM
  id: vyos-vnf
  mgmt-cp: vnf-mgmt-ext
  product-name: vyos-vnf
  sw-image-desc:
  - id: vyos-1.3.3
    image: vyos-1.3.3
    name: vyos-1.3.3
  vdu:
  - cloud-init-file: vyos-userdata
    id: vyos-VM
    int-cpd:
    - id: vdu-eth0-int
      virtual-network-interface-requirement:
      - name: vdu-eth0
        position: 0
        virtual-interface:
          type: PARAVIRT
    - id: vdu-eth1-int
      virtual-network-interface-requirement:
      - name: vdu-eth1
        position: 1
        virtual-interface:
          type: PARAVIRT
    - id: vdu-eth2-int
      virtual-network-interface-requirement:
      - name: vdu-eth2
        position: 2
        virtual-interface:
          type: PARAVIRT
    name: vyos-VM
    supplemental-boot-data:
      boot-data-drive: true
    sw-image-desc: vyos-1.3.3
    virtual-compute-desc: vyos-VM-compute
    virtual-storage-desc:
    - vyos-VM-storage
  version: 1.0
  virtual-compute-desc:
  - id: vyos-VM-compute
    virtual-cpu:
      num-virtual-cpu: 2
    virtual-memory:
      size: 2.0
  virtual-storage-desc:
  - id: vyos-VM-storage
    size-of-storage: 10
```

Listing B.2: VyOS cloud-init file

```
#cloud-config
vyos_config_commands:
  - set system host-name 'vbng'
  - set interfaces ethernet eth0 ip disable-forwarding
  - set interfaces ethernet eth2 dhcp-options no-default-
     route
write_files:
  - path: /opt/vyatta/etc/config/scripts/vyos-postconfig-
     bootup.script
    owner: root:vyattacfg
    permissions: '0775'
    content: |
        #!/bin/vbash
        source /opt/vyatta/etc/functions/script-template

        #check user group to avoid configuration lockup
        if [ "$(id -g -n)" != 'vyattacfg' ] ; then
  exec sg vyattacfg -c "/bin/vbash $(readlink -f $0
     ) $@"
  exit 0
fi

configure

#setup pppoe
set service pppoe-server authentication radius
   server {{ radius_ip }} key 'openstack'
set service pppoe-server authentication mode radius
set service pppoe-server interface eth2
set service pppoe-server gateway-address
   192.168.0.1
set service pppoe-server name-server 8.8.8.8
set service pppoe-server client-ip-pool start
   192.168.0.2
set service pppoe-server client-ip-pool stop
   192.168.0.254

#enable rate limiting using radius
set service pppoe-server authentication radius rate
   -limit attribute Filter-Id
set service pppoe-server authentication radius rate
   -limit enable

#enable snat
set nat source rule 100 outbound-interface eth1
set nat source rule 100 source address
   192.168.0.2-192.168.0.254
set nat source rule 100 translation address
   masquerade

#block users from accessing restricted networks
cidr_mngt=$(ip -o -f inet addr show eth0 | awk '{
   print $4}')
cidr_provider=$(ip -o -f inet addr show eth1 | awk
   '{print $4}')
set firewall name BLOCK_CLIENTS default-action
   accept
set firewall name BLOCK_CLIENTS rule 10 action drop
set firewall name BLOCK_CLIENTS rule 10 destination
    address $cidr_provider
set firewall name BLOCK_CLIENTS rule 10 protocol
   all
set firewall name BLOCK_CLIENTS rule 10 source
   address 192.168.0.2-192.168.0.254
set firewall name BLOCK_CLIENTS rule 11 action drop
```

```
        set firewall name BLOCK_CLIENTS rule 11 destination
            address $cidr_mngt
        set firewall name BLOCK_CLIENTS rule 11 protocol
           all
        set interfaces ethernet eth1 firewall out name
           BLOCK_CLIENTS

        commit
        exit
```

Listing B.3: VyOS NS descriptor file

```
nsd:
  nsd:
  - description: Single VyOS BNG VNF
    df:
    - id: default-df
      vnf-profile:
      - id: VyOS Router
        virtual-link-connectivity:
        - constituent-cpd-id:
          - constituent-base-element-id: VyOS Router
            constituent-cpd-id: vnf-mgmt-ext
          virtual-link-profile-id: mgmtnet
        - constituent-cpd-id:
          - constituent-base-element-id: VyOS Router
            constituent-cpd-id: vnf-internal-ext
          virtual-link-profile-id: internal
        - constituent-cpd-id:
          - constituent-base-element-id: VyOS Router
            constituent-cpd-id: vnf-external-ext
          virtual-link-profile-id: external
        vnfd-id: vyos-vnf
    id: vyos-ns
    name: vyos-ns
    version: 1.0
    virtual-link-desc:
    - id: mgmtnet
      mgmt-network: true
    - id: internal
    - id: external
    vnfd-id:
    - vyos-vnf
```

Listing B.4: Configuration file for VyOS' deployment

```
{
  additionalParamsForVnf:
    [
      {
        member−vnf−index: "VyOS Router",
        additionalParams:
          {
            radius_ip: "192.168.222.3"
          },
      },
    ],
  vld:
    [
      { name: internal, vim−network−name: provider },
      { name: external, vim−network−name: access },
    ],
}
```

Appendix C

RADIUS Configuration Files

Listing C.1: RADIUS VNF descriptor file

```
vnfd:
  description: A RADIUS VNF backed by a SQL Database
  df:
    - id: default-df
      instantiation-level:
        - id: default-instantiation-level
          vdu-level:
            - number-of-instances: 1
              vdu-id: radius-VM
      vdu-profile:
        - id: radius-VM
          min-number-of-instances: 1
  ext-cpd:
    - id: vnf-mgnt-ext
      int-cpd:
        cpd: vdu-eth0-int
        vdu-id: radius-VM
    - id: vnf-cp0-ext
      int-cpd:
        cpd: radius-VM-eth_6MmH
        vdu-id: radius-VM
  id: radius-sql-vnf
  mgmt-cp: vnf-mgnt-ext
  product-name: radius-sql-vnf
  sw-image-desc:
    - id: ubuntu18.04
      image: ubuntu18.04
      name: ubuntu18.04
  vdu:
    - cloud-init-file: user-data
      id: radius-VM
      int-cpd:
        - id: vdu-eth0-int
          virtual-network-interface-requirement:
            - name: vdu-eth0
              virtual-interface:
                type: PARAVIRT
        - id: radius-VM-eth_6MmH
          virtual-network-interface-requirement:
            - name: radius-VM-eth_6MmH
              position: 1
              virtual-interface:
                type: PARAVIRT
      name: radius-VM
      supplemental-boot-data:
        boot-data-drive: true
      sw-image-desc: ubuntu18.04
      virtual-compute-desc: radius-VM-compute
      virtual-storage-desc:
        - radius-storage
  version: '1'
  virtual-compute-desc:
    - id: radius-VM-compute
      virtual-cpu:
        num-virtual-cpu: 2
      virtual-memory:
        size: 2
  virtual-storage-desc:
    - id: radius-storage
      size-of-storage: '10'
```

Listing C.2: RADIUS NS descriptor file

```
nsd:
  nsd:
    - description: radius server with sql database
      df:
        - id: default-df
          vnf-profile:
            - id: radius server
              virtual-link-connectivity:
                - constituent-cpd-id:
                    - constituent-base-element-id: radius
                       server
                      constituent-cpd-id: vnf-mgnt-ext
                  virtual-link-profile-id: mgntnet
                - constituent-cpd-id:
                    - constituent-base-element-id: radius
                       server
                      constituent-cpd-id: vnf-cp0-ext
                  virtual-link-profile-id: internal
            vnfd-id: radius-sql-vnf
    id: radius-sql-ns
    name: radius-sql-ns
    version: '1.0'
    virtual-link-desc:
      - id: mgntnet
        mgmt-network: true
      - id: internal
        mgmt-network: false
    vnfd-id:
      - radius-sql-vnf
```

Listing C.3: RADIUS cloud-init file

```
#cloud-config
hostname: radius
ssh_pwauth: true
chpasswd: { expire: False }
password: password
packages:
  - freeradius
  - freeradius-mysql
  - freeradius-utils
  - mysql-server
package_update: true
package_reboot_if_required: true
runcmd:
  - mysql -uroot -e "UPDATE mysql.user SET
     authentication_string=PASSWORD('$rootpass') WHERE User
     ='root';"
  - mysql -uroot -e "DELETE FROM mysql.user WHERE User='
     root' AND Host NOT IN ('localhost', '127.0.0.1',
     '::1');"
  - mysql -uroot -e "DELETE FROM mysql.user WHERE User='';"
  - mysql -uroot -e "DELETE FROM mysql.db WHERE Db='test'
     OR Db='test_%';"
  - mysql -uroot -e "FLUSH PRIVILEGES;"
  - mysql -uroot -e "CREATE DATABASE radius;"
  - mysql -uroot radius < /etc/freeradius/3.0/mods-config/
     sql/main/mysql/schema.sql
  - mysql -uroot radius < /etc/freeradius/3.0/mods-config/
     sql/main/mysql/setup.sql
  - mysql -uroot radius -e "INSERT INTO nas VALUES (NULL ,
     '192.168.222.0/24', 'openstack', 'other', NULL, '
     openstack', NULL, NULL,  'Provider Network');"

  - sed -i 's/\${modules\.sql\.dialect\}/mysql/g' /etc/
     freeradius/3.0/mods-available/sqlcounter
  - sed -i 's/dialect = "sqlite"/dialect = "mysql"/g' /etc/
```

```
      freeradius/3.0/mods-available/sql
  - sed -i 's/driver = "rlm_sql_null"/driver = "
     rlm_sql_mysql"/g' /etc/freeradius/3.0/mods-available/
     sql

  - sed -i '/^\s*#\s\+server = "localhost"/s/#//' /etc/
     freeradius/3.0/mods-available/sql
  - sed -i '/^\s*#\s\+port = 3306/s/#//' /etc/freeradius
     /3.0/mods-available/sql
  - sed -i '/^\s*#\s\+login = "radius"/s/#//' /etc/
     freeradius/3.0/mods-available/sql
  - sed -i '/^\s*#\s\+password = "radpass"/s/#//' /etc/
     freeradius/3.0/mods-available/sql
  - sed -i '/^\s*#\s\+read_clients/s/#//' /etc/freeradius
     /3.0/mods-available/sql

  - sed -i 's/^\#\(\s\+\)daily$/\1dailycounter/' /etc/
     freeradius/3.0/radiusd.conf

  - ln -s ../mods-available/sql /etc/freeradius/3.0/mods-
     enabled/sql
  - ln -s ../mods-available/sqlcounter /etc/freeradius/3.0/
     mods-enabled/sqlcounter

  - systemctl restart freeradius
```

Listing C.4: Configuration file for RADIUS' deployment

```
{
  vld:
    [
      {
        name: internal,
        vim-network-name: provider,
        vnfd-connection-point-ref:
          [
            {
              member-vnf-index-ref: "radius server",
              vnfd-connection-point-ref: vnf-cp0-ext,
              ip-address: "192.168.222.3",
            }
          ],
      },
    ],
}
```

Appendix D

Add MicroStack to OSM script

Listing D.1: Commands used to launch the instances

```
#! /bin/bash
hosts=("vim1")

for host in "${hosts[@]}"
do
    echo "Configuring microstack for $host"

    echo "Updating default security group in MicroStack to
       allow all access"
    for i in $(ssh $host microstack.openstack security
       group list | awk '/default/{ print $2 }'); do
        for PROTO in icmp tcp udp ; do
            echo "  $PROTO ingress"
            CHECK=$(ssh $host microstack.openstack security
                group rule create $i --protocol $PROTO --
               remote-ip 0.0.0.0/0 2>&1)
            if [ $? -ne 0 ] ; then
                if [[ $CHECK != *"409"* ]]; then
                    echo "Error creating ingress rule for
                       $PROTO"
                    echo $CHECK
                fi
            fi
        done
    done

    ssh $host microstack.openstack network show osm-ext &>/
       dev/null
    if [ $? -ne 0 ]; then
        echo "Creating osm-ext network with router to
           bridge to MicroStack external network"
        ssh $host microstack.openstack network create --
            enable --no-share osm-ext
        ssh $host microstack.openstack subnet create osm-
            ext-subnet --network osm-ext --dns-nameserver
            8.8.8.8 --subnet-range 172.30.0.0/24
        ssh $host microstack.openstack router create
            external-router
        ssh $host microstack.openstack router add subnet
            external-router osm-ext-subnet
        ssh $host microstack.openstack router set --
            external-gateway external external-router
    fi

    if ! ssh $host [ -f ~/.ssh/microstack ]; then
        echo "Generating microstack keypair"
        ssh $host 'ssh-keygen -t rsa -N "" -f ~/.ssh/
            microstack'
    fi

    ssh $host microstack.openstack keypair show microstack
        &>/dev/null
    if [ $? -eq 0 ]; then
        echo "Keypair exists, deleting..."
        ssh $host microstack.openstack keypair delete
            microstack
    fi

    echo "Generating keypair"
    ssh $host microstack.openstack keypair create --public-
        key ~/.ssh/microstack.pub microstack

    echo "Creating VIM microstack-site in OSM"
    scp $host:/var/snap/microstack/common/etc/microstack.rc
         .
    . microstack.rc

    osm vim-delete microstack-site-$host &>/dev/null
    osm vim-create \
        --name microstack-site-$host \
        --user "$OS_USERNAME" \
        --password "$OS_PASSWORD" \
        --auth_url "$OS_AUTH_URL" \
        --tenant "$OS_USERNAME" \
        --account_type openstack \
        --config='{use_floating_ip: True,
                   insecure: True,
                   keypair: microstack,
                   management_network_name: osm-ext}'
        rm microstack.rc
        echo ""
done
```

Appendix E

Validation tests

The tests conducted to validate the prototype are visible in table E.1. For clarification, the term *client* used refers to a device connecting to the RADIUS server for authentication (VyOS in this case), a user is a person registered in RADIUS' database and a subscriber refers to a device connected to the prototype via a PPPoE connection.

Table E.1: Validation tests

ID	Requirement	Details	Result
Test 1	FR-006	**Description:** Attempt to authenticate using RADIUS from a valid client, with correct secret **Expected Behaviour:** Authentication is successful **Observed Behaviour:** Authentication is successful	Pass
Test 2	FR-006	**Description:** Attempt to authenticate using RADIUS from a valid client, with incorrect secret **Expected Behaviour:** Authentication is not successful **Observed Behaviour:** Authentication is not successful	Pass
Test 3	FR-006	**Description:** Attempt to authenticate using RADIUS from an invalid client **Expected Behaviour:** Authentication is not successful **Observed Behaviour:** Authentication is not successful	Pass

Test 4	FR-002	**Description:** Establish PPPoE connection with correct credentials **Expected Behaviour:** PPPoE connection is established **Observed Behaviour:** PPPoE connection is established	Pass
Test 5	FR-002	**Description:** Establish a PPPoE connection with wrong credentials **Expected Behaviour:** PPPoE connection is not established **Observed Behaviour:** PPPoE connection is not established	Pass
Test 6	FR-003	**Description:** Establish a PPPoE connection **Expected Behaviour:** IP assigned is within the limits established **Observed Behaviour:** IP assigned is within the limits established	Pass
Test 7	FR-003	**Description:** Establish more PPPoE connections than there are available IP addresses **Expected Behaviour:** IP addresses are not repeated **Observed Behaviour:** IP addresses are not repeated	Pass
Test 8	FR-006	**Description:** Create a new user at runtime **Expected Behaviour:** The new user is able to log in without the need of restarting the vBNG **Observed Behaviour:** The new user is able to log in without the need of restarting the vBNG	Pass
Test 9	FR-008	**Description:** As a subscriber, attempt to access the RADIUS server. **Expected Behaviour:** Subscriber cannot access the server **Observed Behaviour:** Subscriber cannot access the server	Pass
Test 10	FR-008	**Description:** Attempt to access the radius server from the VyOS instance. **Expected Behaviour:** Access is allowed **Observed Behaviour:** Access is allowed	Pass
Test 11	FR-008 and FR-001	**Description:** As a subscriber, attempt to access the iperf server. **Expected Behaviour:** Subscriber can access the server **Observed Behaviour:** Subscriber can access the server	Pass

Test 12	FR-008 and FR-001	**Description:** As a subscriber, attempt to access the internet **Expected Behaviour:** Subscriber can access the internet **Observed Behaviour:** Subscriber can access the internet	Pass