



Mario Car

OpenStack Service Function Chaining Interface
Interface para Service Function Chaining do
OpenStack



University of Aveiro Department of Electronics,
2015. Telecommunications and Informatics

**Mario
Car**

OpenStack Service Function Chaining Interface

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor Rui Luis Andrade Aguiar, Professor catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e João Paulo Barraca, Professor Assistente Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho ao meu Pai, Mãe, irmã e amigos.

o júri / the jury

presidente / the president

Professor Doutor Anibal Manuel de Oliveira Duarte,
Professor Catedrático, Universidade de Aveiro

**voigas / examiners
commitee**

Doutor Pedro Miguel Naia Neves, Gestor Tecnológico,
Portugal Telecom Inovação, S.A.

Professor Doutor Rui Luís Andrade Aguiar,
Professor Catedrático, Universidade de Aveiro

**agradecimentos /
acknowledgments**

I would like to thank my supervisor Prof. Doutor Rui L. Aguiar and my co-supervisor Prof. Doutor João P. Barraca for their guidance, as well as Igor D. Cardoso and Vitor Cunha for all their help.

palavras-chave

cloud computing, virtualização de redes, openstack, nfv, sfc, openstack, horizon, gui, neutron

resumo

O OpenStack é uma plataforma livre e open-source de cloud computing. É visto como uma importante tecnologia no futuro das telecomunicações. O OpenStack facilita a criação de ambientes de virtualização e é visto como uma grande tecnologia para o desenvolvimento da virtualização de funções de rede (NFV). Atualmente, a fundação OpenStack está a desenvolver os casos de uso e o código para a virtualização funções de serviço, mas os aspectos das camadas mais elevadas de gestão não estão a ser considerados. Esta dissertação vai enfrentar este desafio, e vai trabalhar na criação de uma interface para um uso simples do NFV, permitindo que o operador de rede construa serviços por concatenação de elementos gráficos. As interfaces de programação de aplicações que estão actualmente a ser desenvolvidas serão analisadas e uma interface web simples para explorar potencialidades das mesmas será criada.

keywords

cloud computing, network virtualization, openstack, nfv, sfc, openstack, horizon, gui, neutron

abstract

OpenStack is a free and open-source cloud computing software platform. It is seen as a major technology enabler for the future of telecommunications. OpenStack eases the creation of virtualization environments, and is seen as a major technology for the development of network function virtualization (NFV). Currently, OpenStack is developing the use cases and the code for service function virtualization, but the higher layer management aspects are not being considered. This dissertation will address this challenge, and will work on the creation of an interface for a simple usage of the NFV functions, enabling the network manager to build services by concatenation of graphical elements. The Application Programming Interfaces that are currently being developed will be analyzed and a simple web interface to explore their potentialities will be created.

Table of content

List of figures	7
1 Introduction	11
1.1 Objectives	12
1.2 Structure.....	12
2 Problem statement	13
2.1 Motivation	13
2.1.1 IT Aveiro	14
3 State of the art.....	17
3.1 Related concepts and technologies	17
3.1.1 Virtualization.....	17
3.1.2 Network Virtualization.....	19
3.1.3 Cloud Computing	24
3.1.4 Telco Network Architecture	26
3.1.5 OpenStack Platform	29
3.1.6 OpenStack Neutron	35
3.1.7 OpenStack Horizon	38
3.2 Related work.....	42
3.2.1 Group Based Policy.....	42
3.2.2 Telco Cloud Environment	45
4 A novel GUI for SFC.....	47
4.1 Django	47
4.2 Extending the OpenStack Dashboard	48
4.2.1 Limitations	55
5 Evaluation and results.....	57
5.1 Evaluation.....	60
5.2 Overview	61
5.3 Testing	61
5.3.1 Task 1: Creating an AP and attaching it to a network.....	62
5.3.2 Task 2: Creating a steering classifier	63
5.3.3 Tasks 3 and 4: Creating a SFC	65
5.3.4 Survey.....	67
6 Conclusion.....	69
7 References	75

List of Acronyms

AMQP	Advanced Message Queue Protocol
AP	Attachment Point
API	Application Programming Interface
BNG	Boarder Network Gateway
BRAS	Broadband Remote Access Server
BSS	Business Support Systems
CapEx	Capital Expense
CLI	Command-Line Interface
CPE	Customer Premises Equipment
DETI	Departamento De Electrónica Telecomunicações e Informática
DNAT	Dynamic Network Address Translation
DNS	Domain Name Server
DPI	Deep Packet Inspection
EM	Element Manager
ETSI	European Telecommunications Standards Institute
FOOS	Free and Open-Source Software
FW	Firewall
GBP	Group Based Policy
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPsec	Internet Protocol Security
ISG	Industry Specification Group
IT	Information Technology
JS	JavaScript
L2	Layer 2
L3	Layer 3
LAN	Local Area Network
MAC	Media Access Control

ML2	Modular Layer 2
MVC	Model-View-Controller
NASA	National Aeronautics and Space Administration
NAT	Network Address Translation
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NIC	Network Interface Controller
OpEx	Operational Expense
OS	Operating System
OSS	Operations Support Systems
OVSDb	Open Virtual Switch Database
P2P	Peer-to-Peer
Paas	Platform as a Service
PC	Personal Computer
PT	Portugal Telecom
PTL	Project Team Lead
QoS	Quality of Service
R&D	Research and Development
RAM	Random Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SDN	Software Defined Networking
SF	Service Function
SFC	Service Function Chain
SNAT	Static Network Address Translation
SOA	Service Oriented Architecture
SVG	Scalable Vector Graphics
Telco	Telecommunications Company
URL	Uniform Resource Locator
VE	Virtualization Engines
vEPC	Virtual Evolved Packet Core
VIM	Virtual Infrastructure Management
VLAN	Virtual Local Area Network

VM	Virtual Machine
VNF	Virtual Network Function
vNIC	Virtual Network Interface Card
VPN	Virtual Private Network
VPS	Virtual Private Servers
VXLAN	Virtual Extensible Local Area Network
WSGI	Web Server Gateway Interface

List of figures

Figure 1 Current network architecture [2].....	13
Figure 2 New network architecture [2]	14
Figure 3 Neutron-client help	15
Figure 4 VLAN	20
Figure 5 VPN [7].....	20
Figure 6 NFV architecture model [10]	21
Figure 7 SDN architecture model [12]	22
Figure 8 Service Function Chaining [15]	23
Figure 9 Cloud computing models [19]	25
Figure 10 Classical networks vs. NFV [2]	28
Figure 11 OpenStack conceptual architecture [40]	32
Figure 12 OpenStack deployment on three nodes [41]	33
Figure 13 OpenStack contributing workflow [41]	34
Figure 14 Plug-in design [44].....	36
Figure 15 OpenStack deployment on three nodes [3]	37
Figure 16 OpenStack dashboard	39
Figure 17 Horizon UI structure [48].....	40
Figure 18 Horizon app structure.....	41
Figure 19 Dashboard class example.....	41
Figure 20 Panel class example	42
Figure 21 GBP architecture [50]	43
Figure 22 Cloud4NFV platform [16]	45
Figure 23 AP display in the Network Topology Panel	49
Figure 25 Attachment Points panel	50
Figure 24 Attachment Point creation workflow	50
Figure 26 Create AP modal	50
Figure 27 AP Attach modal.....	51
Figure 28 Create a Steering Classifier modal.....	52
Figure 29 SFC creation workflow	52
Figure 30 SFC tab	53
Figure 31 Create SFC modal	53
Figure 32 Add Port modal	54

Figure 33 SFC creation via flowchart	54
Figure 34 Flowchart Panel	55
Figure 35 Create SFC from a flowchart	55
Figure 36 Chain of chains	56
Figure 37 Demo network topology	57
Figure 38 OpenArena	58
Figure 39 Firewall blocks Player 1	59
Figure 40 OpenArena connection interrupted	60
Figure 41 Task 1: Creating and attaching an AP	63
Figure 42 Task 2: Creating a steering classifier	64
Figure 43 Task 2 comparison	64
Figure 44 Task 3 vs Task 4: Creating a SFC.....	65
Figure 45 Neutron port list	66
Figure 46 Task 4: Creating a SFC comparison	66
Figure 47 Error distribution.....	67
Figure 48 Survey results.....	67

List of tables

Table 1 Essential neutron-client commands.....	16
Table 2 Resources mapping	44
Table 3 Network service chaining resources.....	44
Table 4 Developed OpenStack Panels.....	48

1 Introduction

Network Function Virtualization (NFV) and Software Defined Networking (SDN) have emerged as one of the most promising paradigms for revolutionising the way we manage networks. Although the concept of NFV is quite recent, it is based on technologies that have proven their validity in Information Technology (IT), and it is the result of careful experimenting and evaluation by players in the industry and academy in the last years.

A majority of the telecom industrial stakeholders claim that virtualizing network functions will help their businesses by reducing management and operational costs, without affecting network performance and service provisioning workflows [1]. In fact, NFV should ease and automate the management of the network functions. Together with SDN, NFV is foreseen as the main enabler of the future network operating system. Service Function Chaining is another concept that is related to SDN and NFV. It uses the composition of network services in order to create a chain of services that provides the desired functionality.

This work proposes a solution to the problem of managing virtualized network functions (VNFs) in *OpenStack*¹. Specifically, it suggests a solution for managing Service Function Chaining. The tools that are currently available are not user friendly, and require the usage of the command line interface (CLI). Furthermore, the administrator has to be familiar with the system, know the commands and the proper sequence in order to achieve the desired configuration. Any large scale network configuration is impractical as it requires a lot of human interaction with a somewhat exhausting command line interface. However this can be simplified and somewhat automated by creating scripts that configure the whole network, but it is only helpful in the case of initial configuration of the network. Even though it is not suitable when some parts of the network have to be reconfigured it is still used because of a lack of alternatives. This approach isn't changing the paradigm, with the administrator still using the command line interface. The other approach for managing networks in OpenStack is via a graphical user interface (GUI). By taking advantage of available APIs it should be possible to create a graphical user interface for managing virtualized network functions. A GUI solution for managing networks in OpenStack already exists, but it only provides basic functionality, e.g., virtual networks and subnetworks provisioning. It does not support Service Function Chaining management. A graphical user interface for Service Function Chaining

¹ <https://www.openstack.org/>

could prove more user friendly and efficient than the command line interface thus making the job of an administrator easier and more pleasant.

1.1 Objectives

The main objective of this dissertation is to create a graphical user interface that complements the existing command-line tools, mainly the *neutron-client*, that provides functionality for extending the virtualized network segment in *OpenStack*. The *neutron-client* provides some advanced capabilities, as the creation of a Service Function Chain (SFC). This feature allows the administrator to create network functions using concatenated Virtualized Network Functions (VNFs). The main goal of this work is providing a GUI that makes the chaining of VNFs easier, more intuitive and faster. Other objectives include exploring the *OpenStack Neutron* and *Horizon* modules, their APIs and learning how they work. There is also a possibility of contributing to the *OpenStack* project.

1.2 Structure

The second chapter defines the scope of the work and proposes the solution. For the reader to be able to understand this dissertation key principles and technologies are explained in the third chapter. The third chapter also provides a review of existing solutions for extending the virtualized network segment and managing VNFs. The chapter that follows The implementation is discussed in the fourth chapter. In the fifth chapter the solution is discussed and evaluated. The last chapter contains a conclusion about the work done. Figures and tables are listed before chapter one, and citations and bibliography are placed at the end of the document.

2 Problem statement

Creating and managing SFCs in OpenStack is currently done through CLI. This work proposes a dedicated Panel in the OpenStack Dashboard for managing SFCs in order to provide a GUI to the administrator. Also a new approach to creating SFCs is proposed, namely the standard OpenStack web GUI could be enhanced by adding custom JavaScript that would enable a more user friendly workflow, i.e., creating a chain by point and click.

2.1 Motivation

The whole concept is based around the virtualization of the home environment. Currently operators provide services using backend systems and CPE (Customer Premise Equipment) devices located in the home network. These devices usually include a RGW (Residential Gateway) or HGW (Home Gateway) for Internet and VOIP services and a STB (Setup Box) for Media services (Figure 1). In the figure the home is equipped with a STB and a RGW. All services are received by the RGW and delivered inside the home. A conversion to private IP addresses is made by the RGW, which has connectivity to Internet through the BNG, also referred to as BRAS.

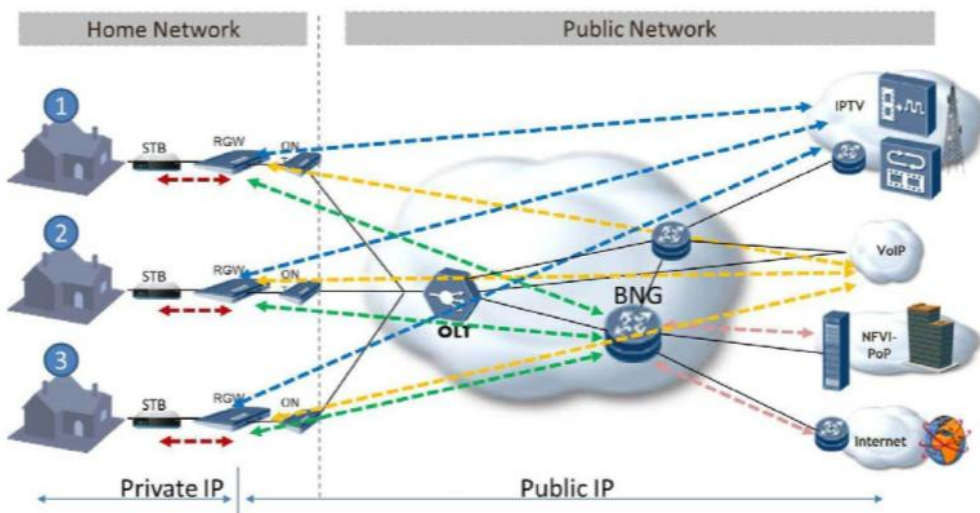


Figure 1 Current network architecture [2]

The new architecture uses virtualization of services and functionality migration from home devices to the cloud (Figure 2). RGW and STB are replaced by vRGW and vSTB.

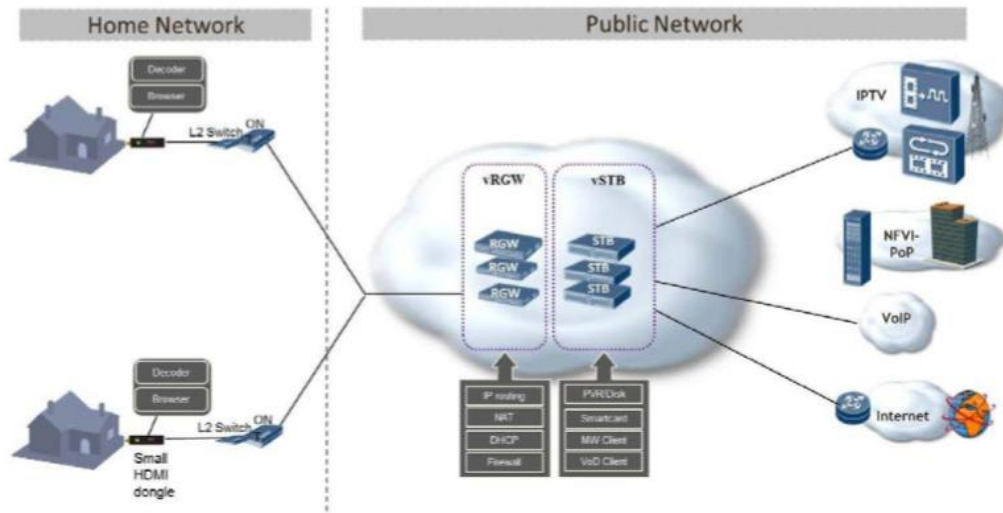


Figure 2 New network architecture [2]

There are numerous advantages both for the operator and the end customer. First of all the reduction of CapEx and OpEx for the operator by eliminating the need to deploy, maintain and upgrade the CPEs. The introduction of new services becomes easier and less dependent of the CPEs which benefits both the operator and the end customer. Also the quality of experience increases with new functionality such as multi-screen support, opposed to the current need for one STB per each TV.

2.1.1 IT Aveiro

The work done in this dissertation is strongly correlated with the work of Igor D. Cardoso, Vitor Cunha and Carlos Gonçalves. All of them were, or are currently somehow related to the *Instituto de Telecomunicações, Aveiro (IT Aveiro)*, and *DETI, Universidade de Aveiro*. In his MSc dissertation [3] Cardoso discussed the possibility of extending virtual networks with physical or virtual resources that reside outside the data centre (cloud). He developed a solution that enables a L2 tunnel between the vHGW the devices that connect to it and the VNFs running in the cloud. Currently he is working on creating VNFs, e.g., firewall. In this context VNFs are actually customized OpenStack instances (virtual machines). They can be provisioned through Horizon. In the scope of his MSc, Cunha is working on enabling dynamic port creation in OpenStack, with the goal of developing a solution that will enable automatic creation of a port for a physical device when it connects to the home device, in other words, it will enable automatic access to operators cloud and to the Internet. In neutron-client this is represented as a vHGW. Gonçalves developed the traffic steering support for the neutron-client, which consists of traffic classifiers that enable classification and port chains

that enable VNF concatenation. All three solutions are aggregated in the neutron-client. Neutron-client can be accessed through CLI with the command "*neutron*". A general pattern for the usage of neutron-client is: "*neutron command -options arguments*". There is no manual for the neutron-client, which makes the usage difficult for someone inexperienced and not previously acquainted with it. However it must be stated that a list of all commands can be obtained by executing "*neutron -h*". A help page for a specific command can be accessed either by executing the command without any arguments, e.g., "*neutron attachment-point-create*" (Figure 3) or by executing the command with the option "-h".

```
kaiser@ubuntu:~$ neutron attachment-point-create
usage: neutron attachment-point-create [-h]
                                         [-f {html,json,shell,table,value,yaml}]
                                         [-c COLUMN] [--max-width <integer>]
                                         [--prefix PREFIX]
                                         [--request-format {json,xml}]
                                         [--tenant-id TENANT_ID] [--name NAME]
                                         [--description DESCRIPTION]
                                         [--network_id NETWORK_ID]
                                         [--admin-state-down]
                                         LOCAL_IP REMOTE_IP DRIVER IDENTIFIER
                                         TECHNOLOGY
neutron attachment-point-create: error: too few arguments
kaiser@ubuntu:~$
```

Figure 3 Neutron-client help

The first approach is only useful for experienced users as it doesn't explain any of the arguments, doesn't provide examples and is only triggered after the specific command is executed. This presents a problem as some commands don't need any arguments and are normally executed even if you just want to display the help page without displaying the help, e.g., "*neutron steering-classifier-create*". However the second approach provides detail info about the specific command. One can argue that the commands should be grouped in order to provide better user experience. The commands dedicated to a certain entity (e.g., attachment point) are separated, as each command is basically independent, e.g. "*neutron attachment-point-create*" and "*neutron attachment-point-delete*". This could be reorganised so that all the commands regarding a certain entity are grouped and an action is specified as an argument or an option, e.g., "*neutron attachment-point attach other_args*". That way all the commands related would be available from the same help page, which is more practical. Essential commands for this work are displayed in Table 1. These solutions were developed in collaboration with Portugal Telecom (PT) and were confirmed multiple times both by IT

Aveiro and by PT. They provide the desired functionality, but they are not user friendly, because of their CLI nature. The user experience can be improved by introducing a GUI that will provide the same functionality. The need to know all the commands is eliminated by using GUI, also it is more intuitive to connect VNFs with a mouse in order to create a chain, than by typing commands. These services are managed by the administrator, but in the future the end customer might be able to use some of these or other similar services. It is delusional to expect that the end user will have the knowledge to use the CLI. During the 80's and 90's the first GUIs appeared and quickly became popular. New users unfamiliar with the OS prefer GUI environment, as they find it more intuitive. This is a strong argument for the GUI approach for managing SFC.

Neutron-client command	Description
attachment-point-create	Create a new attachment point.
attachment-point-delete	Delete a given attachment point.
attachment-point-attach	Attach an attachment point to a given network.
attachment-point-detach	Detach an attachment point from a network.
attachment-point-list	List attachment points that belong to a given tenant.
vhgw-network-create	Register a vHGW and associate a neutron network to its entry.
vhgw-network-delete	Delete a given vHGW.
steering-classifier-create	Create a traffic steering classifier.
steering-classifier-delete	Delete a given classifier.
steering-classifier-list	List traffic steering classifiers that belong to a given tenant.
port-chain-create	Create a port chain.
port-chain-delete	Delete a port chain.
port-chain-list	List port chains that belong to a given tenant.

Table 1 Essential neutron-client commands

3 State of the art

This chapter describes the terms, technologies and fields related with the work done in this dissertation. Virtualization and cloud computing are major trends in IT. Cloud computing is a prominent technology that is developing at a staggering rate because of the ever increasing number of users in IT. To emphasise this claim *Google Apps*² can be mentioned. It's a cloud based set of services which are available through a web browser. Services like *Gmail*, *Google Docs*, *Google Drive*, etc. are being used by millions of people every day. Another big cloud service provider is Amazon, which provides the usage of virtual machines form their cloud service platform EC2³ (IaaS). It also provides cloud data storage on *Amazon cloud drive*⁴.

3.1 Related concepts and technologies

3.1.1 Virtualization

Virtualization is a technology that uses software to create virtual entities, that can be used instead of actual hardware. The concept of virtualization has emerged during 1960s in United States, where *IBM* logically divided the resources of their mainframe computers into virtual machines to increase utilization of hardware. The motivation for creating this approach was the high price of hardware at that time. By using virtualization *IBM* managed to decrease the price of their mainframe computers. Since then the term has expanded in meaning, and different applications have emerged. Nowadays virtualization is used throughout IT systems. Sometimes we use it even without noticing it (knowing it), e.g., when we use our PCs we don't realize that our applications use virtual memory that is mapped to part of actual physical RAM modules. Virtualization applications include:

- Network virtualization
- Data virtualization
- Software virtualization
- Memory virtualization
- Computer systems virtualization

² <https://www.google.com/work/apps/business/>

³ <http://aws.amazon.com/ec2/>

⁴ <https://www.amazon.com/clouddrive/home>

Data virtualization is a method for data management. It enables the user to retrieve and manipulate the data without knowing technical details about the data, e.g., how it is formatted or where it is located [4]. It provides an abstraction layer that is used to manage the data. An example of data virtualization usage are Facebook⁵ photo albums. When you upload a photo to Facebook you don't know where it is stored, or in which format, but still you can retrieve it. Basically data virtualization provides data to an application without the concern for where the data resides, what the technical interface is, how was it implemented, what platform it uses and how much of data is available [4].

Software virtualization can be divided into multiple categories like application virtualization, workspace virtualization, operating-system-level virtualization. Application virtualization is a technology that encapsulates application software from the underlying operating system on which it is executed. A fully virtualized application is not installed in the traditional sense, although it is still executed as if it were. The application behaves at runtime like it is directly interfacing with the original operating system and all the resources managed by it, but can be isolated or sandboxed to varying degrees. Workspace virtualization is similar to application virtualization but it encapsulates an entire workspace, which consists of everything above the operating system kernel, i.e., applications, data and settings. Operating-system-level virtualization is a server virtualization method where the kernel of an operating system allows for multiple isolated user space instances, instead of just one. Such instances may look and feel like a real server from the point of view of its owners and users. They are called containers⁶, virtualization engines (VE), virtual private servers (VPS), or jails⁷. Operating-system-level virtualization is commonly used in virtual hosting environments, where it is useful for securely allocating finite hardware resources amongst a large number of mutually-distrusting users.

Memory virtualization is a technology used by operating systems in order to compensate for the shortage of physical memory (RAM). When the memory requirements can't be met the OS has to transfer data to disk storage. This process is called paging or swapping. Eventually the OS will retrieve the data from the disk storage back to the RAM. It is also used in clusters to create a pool of shared memory that all computers can access.

⁵ <https://www.facebook.com/>

⁶ https://wiki.archlinux.org/index.php/Linux_Containers

⁷ <https://www.freebsd.org/doc/handbook/jails.html>

There are three types of computer systems virtualization: Full virtualization, Partial virtualization and Paravirtualization. Full virtualization provides the almost complete simulation of the computer system hardware which allows running unmodified software (usually a guest OS) on the virtual machine (a term used to describe the simulated computer system). The difference between the full and partial virtualization lies in the fact that certain software needs to be modified to run on partially virtualized virtual machines. The original meaning of virtualization that was introduced by *IBM* is now called paravirtualization and describes a scheme where software is being run in isolated domains, as if they were running on different systems. The benefits of virtualization are: increased utilization, reduced amount of hardware, smaller demand for space and energy and inherent security. It reduces both Capital Expense (CapEx) and Operational Expense (OpEx). It also enables easier management and provides better security due to isolation from the rest of the system.

3.1.2 Network Virtualization

Network virtualization is a concept that allows multiple virtual networks to exist on a single physical network. It provides flexibility and customization through decoupling the services from the infrastructure. There are multiple technologies that embrace this approach like: VLAN, VPN, Overlay networks and Programmable networks. VLAN is an acronym for Virtual Local Area Network [5]. Using software it enables the coexistence of multiple LANs in an environment where only one LAN can exist (Figure 4), because of the hardware in use. This is done by adding tags to the header of each packet.

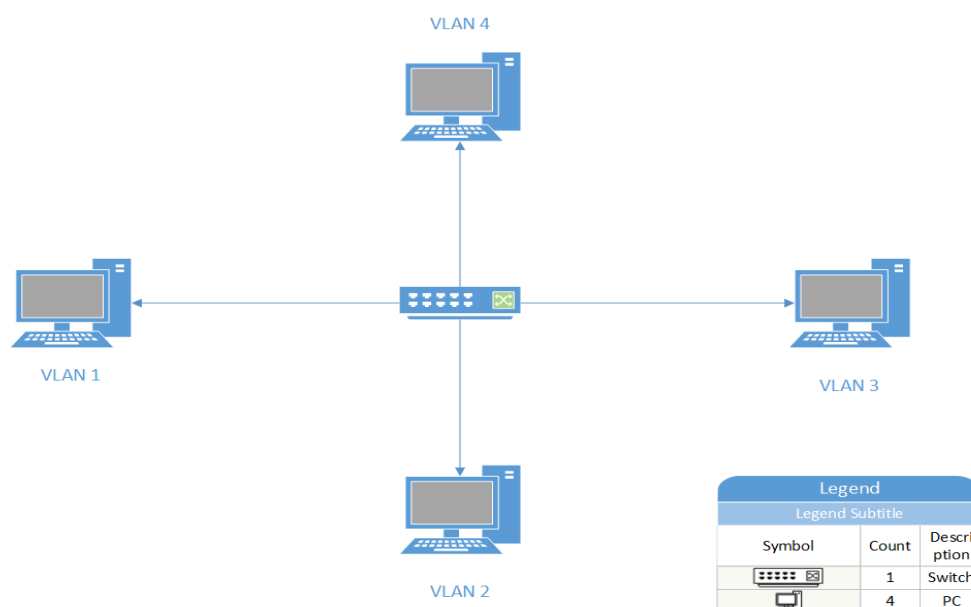


Figure 4 VLAN

A Virtual Private Network (VPN) is a private network that uses a public network (usually the Internet) to connect remote sites or users together [6]. Instead of using a dedicated, real-world connection, such as leased line, a VPN uses "virtual" connections routed through the Internet from the company's private network to the remote site or employees. This is done by creating a tunnel between the remote site and the specified private network (Figure 5). VPNs have to provide security, reliability, scalability, network management and policy management. Data confidentiality is a major issue for any VPN. Since the data is travelling over a public network it has to be encrypted in order to ensure confidentiality.

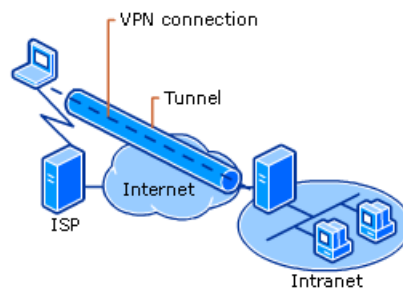


Figure 5 VPN [7]

The idea behind Overlay networks is that a virtual topology is created upon another, main topology, e.g., Peer-to-Peer (P2P) networks. These will not be discussed in detail as they are not in the focus of this dissertation. Programmable networks provide means to decouple the network infrastructure from the control software [8]. The latest trends in network virtualization are SDN and NFV.

3.1.2.1 Network Function Virtualization (NFV)

Network function virtualization is a concept that decouples network functions from dedicated hardware like routers, firewalls, load balancers, etc. and enables them to be carried out on virtual machines (VMs). This decoupling requires a new set of management and orchestration functions and creates new dependencies between them, thus requiring interoperable standardised interfaces, common information models, and the mapping of such information models to data models [9]. The problem of vendor lock-in can easily be solved by applying NFV. Using VMs instead of dedicated hardware enables quick development and shortens the time-to-market for new services. A Virtualized Network Function is a Network Function capable of running on an NFV Infrastructure (NFVI) and being orchestrated by a NFV Orchestrator (NFVO) and VNF Manager [9].

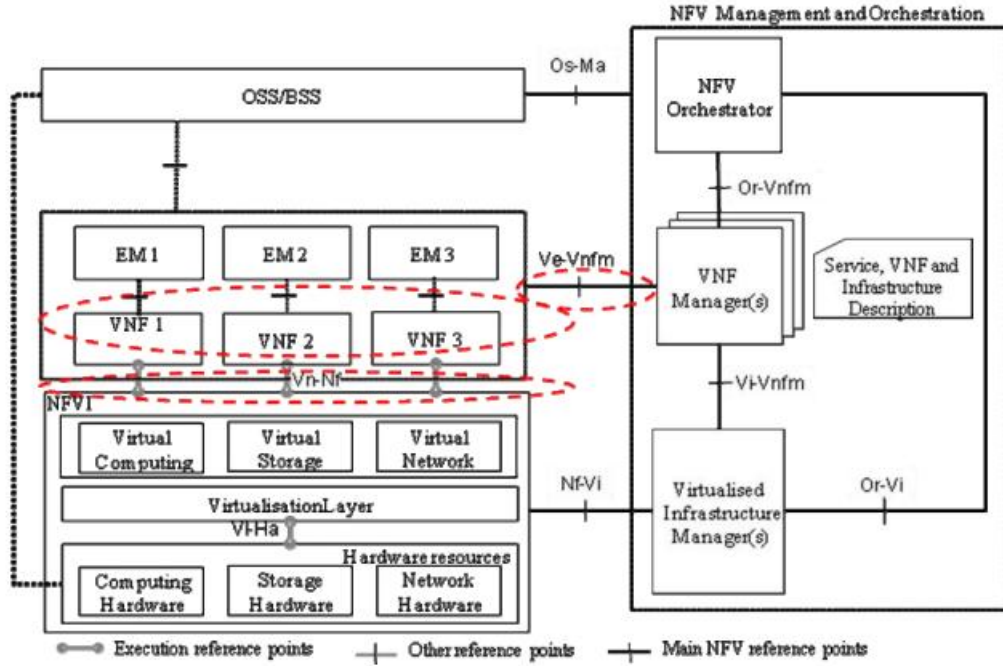


Figure 6 NFV architecture model [10]

The NFVO performs orchestration functions of NFVI resources across multiple Virtualized Infrastructure Managers (VIMs) and lifecycle management of network services. The VNFM performs orchestration and management functions of VNFs. The VIM performs orchestration and management functions of NFVI resources within a domain. The NFVO interacts with the OSS/BSS for provisioning, configuration, capacity management, and policy-based management [10]. The VNFM interacts with the Element Manager (EM) and the VNF for provisioning, configuration, and fault and alarm management [10]. The VIM interacts with the NFVI for the management and orchestration of virtualised resources.

One of the key benefits of NFV is the elasticity provided by the infrastructure for capacity expansion and the roll out of new network functions. However, to take full advantage of the elasticity benefits of NFV, higher levels of automation are needed for the provisioning, configuration, and performance testing of virtualised network functions. NFV is mostly being nurtured by the ETSI NFV ISG, a group started by nine telecom operators at the beginning of 2013 that has grown up to more than 150 participating organizations since then [11]. The group foresees the identification of relevant standards necessary for making NFV a reality.

3.1.2.2 Software Defined Networking (SDN)

Software defined networking is a concept that proposes the decoupling of data and control planes in networks. In classical networks data and control planes are tightly coupled, e.g.,

network switches forward packets based on internal decisions. The data plane (sometimes called the forwarding plane) handles the routing of data packets to its destination, while the control plane creates routing tables, and determines routes [12]. SDN concept proposes that the control plane can be logically centralized and taken away from the physical devices. The separation of the two planes provides a new approach for networking: switches/routers become simple packet forwarding devices, while a software controller manages the entire network from a logically single point [13]. The SDN model consists of: controllers, southbound APIs and northbound APIs (Figure 7). The controller is regarded as the "brains" of the network. It controls the network and handles as middleware between switches/routers and applications. It is usually composed of modules that perform different network tasks. Southbound APIs are used to relay information to switches/routers. The well known protocols used to communicate with switches/routers are OpenFlow⁸ and open virtual switch database (OVSDB). Northbound APIs are used for communication with the software and business logic above. The APIs allow quick network reconfiguration.

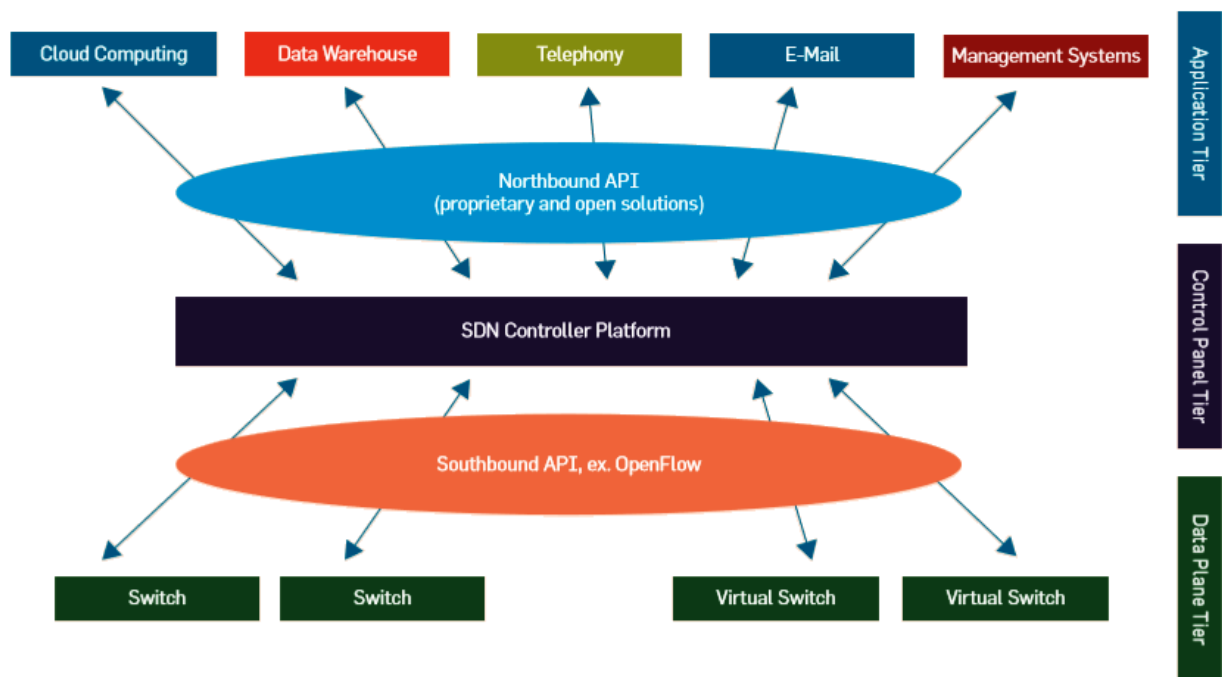


Figure 7 SDN architecture model [12]

This model provides flexibility to the network, simplifies the devices, which in return reduces their cost. Management and maintenance of the network benefit from this approach. SDN reduces both CapEx and OpEx.

⁸ <https://www.opennetworking.org/sdn-resources/openflow>

3.1.2.3 Service Function Chaining

Service function chaining is a fairly new concept that proposes a workflow defining and instantiating an ordered set of service functions and steering the traffic through them. A SFC defines an ordered set of abstract Service Functions (SFs) and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification [14]. The process of classification is rather simple, one or more classifiers, that contain certain criteria are applied to each packet/frame/flow, in order to check if they match. When identified the packet/frame/flow is then steered along the SFC. Figure 8 shows two different SFCs, classification and traffic steering.

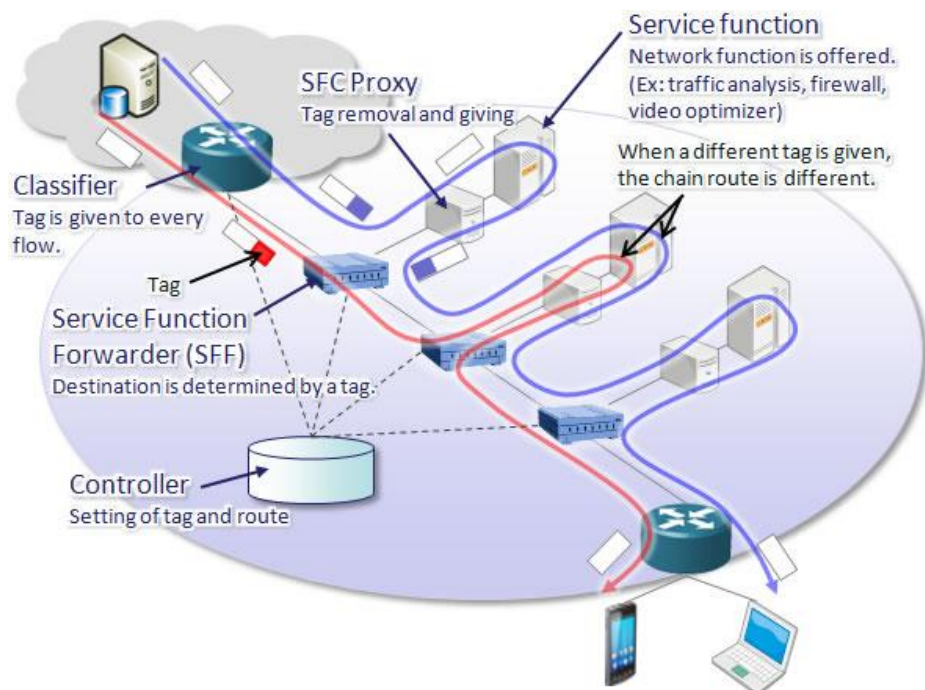


Figure 8 Service Function Chaining [15]

A SF can be realized as a virtual element or be embedded in a physical device. They can be classified as [16]: active or passive. Active SFs are a part of the main course of the chain. Furthermore there are two sub-types, the ones that drop or forward the packets, but do not change them (firewall) and functions that modify the packets (IPSec VPN server). Passive SFs are out of the main course of the chain. Traffic is considered duplicated when having to reach a passive function. They mainly inspect packets, e.g., deep packet inspection (DPI). Together with NFV and SDN service function chaining can provide flexibility and elasticity to the network. Operators can reduce their expenses and shorten the time to market for new services(network functions).

3.1.3 Cloud Computing

All these concepts and technologies have lead to the creation of cloud computing. Cloud computing is a technology that provides IT resources using virtualization. Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centres that provide those services[17]. The idea for cloud computing emerged because of the increased need for resources in IT. It is widely accepted that cloud computing has five key characteristics[18]:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

The on-demand self-service means that a user can provision resources as needed, without the human interaction with the service provider, usually through a web-based service portal. Resources are available over the network, supporting heterogeneous client platforms. They are pooled and dynamically assigned to users. Multiple users can utilize the same physical resource at the same time. They have no knowledge or control of the location of the provided resources. The resources can be elastically provisioned and released. The user has a delusion that the resources are unlimited. Resource usage can be monitored, controlled and reported which provides transparency both for the service provider and the user. This is essential for the pay-per-use model.

Cloud computing is a large scale virtualization implementation, usually consisting of a set of computers (and other hardware) combined into one large system which appears to be monolithic. The actual components of the system are invisible to the user, as if they are obscured by a cloud. The similarity of this approach with big mainframe computers from the past can't be overlooked. The services provided by cloud computing can be divided into three categories (Figure 9): Infrastructure as a service (IaaS), Platform as a service (Paas) and Software as a service (Saas). IaaS is a model where a service provider provides virtualized computing resources over the Internet. The user does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). PaaS provides a platform on top of the infrastructure, typically containing an OS, a

programming language execution environment, database and a web server is provided. The user does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. SaaS provides an application to the user, e.g., Gmail, Office365, Google, Docs, etc. The user does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.

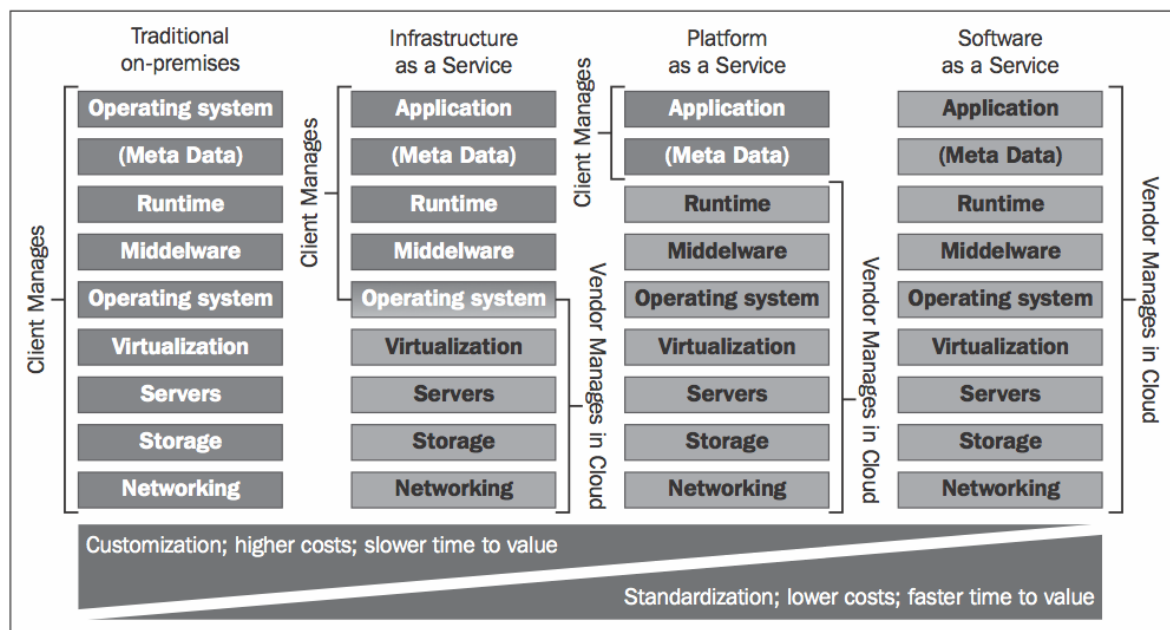


Figure 9 Cloud computing models [19]

Cloud computing aims to provide reliable, customized and quality of service (QoS) guaranteed dynamic environment [20]. The services provided are accessed by simple methods in a location independent way. The cloud can be automatically reconfigured and orchestrated before being presented to the user [20]. The main advantages of cloud computing are scalability, flexibility and on-demand provisioning. It provides a self-service model where the user is able to request and get resources, without being responsible or a need to know how the resources are provisioned or where they are located. Providing resources as services is enabled by using Service Oriented Architecture (SOA). SOA is a paradigm that puts services in focus. It guides business solutions to create, organize and reuse computing components [21]. Cloud computing is a flexible platform for building SOA solutions.

Cloud computing has a big impact on the utilization of physical resources. In the traditional client-server model the servers are usually idle, with some bursts that manifest themselves as

peaks in user activity, which means that most of the time the resources aren't used, and are basically wasted. Cloud computing uses the virtualization to run multiple VMs on one physical machine (computer), therefore increasing the utilization. The industry provides four deployment models: Private cloud, Community cloud, Public cloud and Hybrid cloud.

In the private cloud model the infrastructure is used by a single organization. It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises. A community cloud is similar to private cloud. It is a multi-tenant infrastructure that is shared among several organizations from a specific group with common computing concerns. Such concerns may be related to performance requirements, such as hosting applications that require a quick response time. The goal of a community cloud is to have participating organizations realize the benefits of a public cloud such as multi-tenancy and a pay-as-you-go billing structure but with the added level of privacy, security and policy compliance usually associated with a private cloud. The community cloud can be either on-premises or off-premises, and can be governed by the participating organizations or by a third-party. A public cloud is one based on the standard cloud computing model, in which a service provider makes resources, such as applications and storage, available to the general public over the Internet. Public cloud services may be free or offered on a pay-per-usage model. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider. Hybrid cloud is a cloud computing environment which uses a mix of on-premises, private cloud and public cloud services with orchestration between the two platforms. By allowing workloads to move between private and public clouds as computing needs and costs change. Hybrid cloud gives businesses greater flexibility and more data deployment options.

The IT industry is moving fast towards cloud computing with big steps. *Microsoft* has already invested 15 billion dollars [22] for cloud R&D (Research and Development). In September of 2014 Cisco announced that it will invest 1 billion dollars into cloud computing during the next two years [23].

3.1.4 Telco Network Architecture

Today's networks consist of a large number of different proprietary hardware (Figure 10). Because of the need for backward compatibility and support of older technologies network operators are faced with a big challenge when they want to launch a new network service.

First they have to find additional space and power for new hardware. There is also the challenge to integrate the hardware into the existing network, which requires specific skills that may not be available, or come at high price. The management and maintenance as well as the capital investment have to be taken into account. With the acceleration of technology and development of new services the life cycle of hardware is becoming shorter. In such environments it is possible that the hardware gets replaced even before it returns the investment. In other words the process of designing, integrating and deploying is repeated often and with little or no revenue.

NFV aims to address these problems by leveraging standard IT virtualization technology to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in data centres, network nodes and in the end user premises. ETSI believes network function virtualization is applicable to any data plane packet processing and control plane function in fixed and mobile network infrastructures [2]. NFV is highly complementary to Software Defined Networking (SDN). They are completely independent, but can provide each other mutual benefits. It is important to emphasise that NFV can be deployed without SDN and vice-versa.

In order not to confuse the reader a clear distinction between NFV and VNF must be made. VNF refers to a network function that is virtualized, while NFV refers to a network architecture concept. By eliminating the dependency between a network function and its hardware using VNF operators can achieve higher utilization, because multiple VNFs can share the same physical machines. This creates new business opportunities analogous to the cloud computing service models of Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), e.g., a VNF owner doesn't necessarily own the NFV Infrastructure needed for the proper functioning and operation of the VNF. To provide network services using this approach VNFs have to be organised in an ordered graph. By concatenating VNFs a SFC is created.

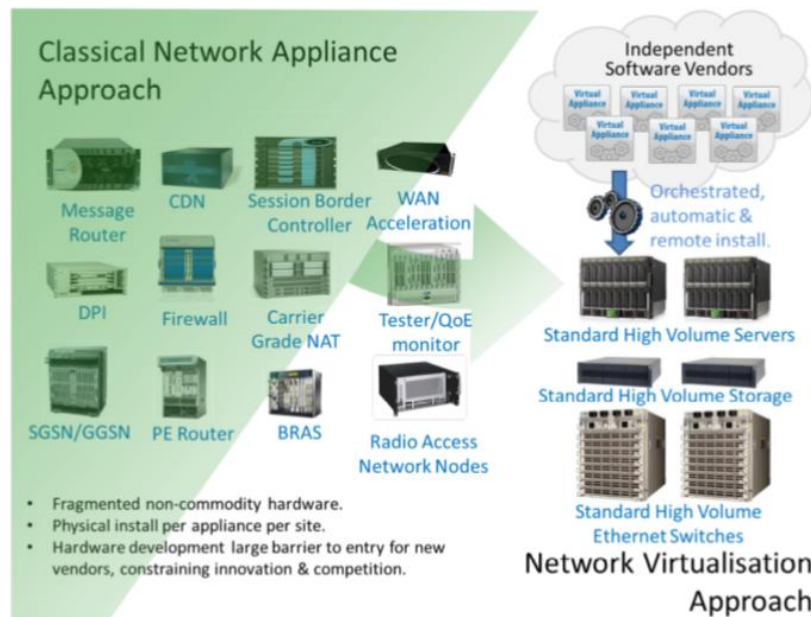


Figure 10 Classical networks vs. NFV [2]

The most common practice by telecommunication operators is resource overprovisioning. In order to cope with specific scenarios like sudden rise in traffic or network failure they are forced to provision more resources than it is actually needed, to maintain performance and availability. This could be solved by using the advantages of the cloud, where resources can be provisioned on-demand.

NFV could potentially offer many benefits including, but not limited to [2]:

- Reduced equipment costs and reduced power consumption through consolidating equipment and exploiting the economies of scale of the IT industry.
- Increased speed of Time to Market by minimising the typical network operator cycle of innovation. Economies of scale required to cover investments in hardware-based functionalities are no longer applicable for software-based development, making feasible other modes of feature evolution. NFV should enable network operators to significantly reduce the maturation cycle.
- Availability of network appliance multi-version and multi-tenancy, which allows use of a single platform for different applications, users and tenants. This allows network operators to share resources across services and across different customer bases.
- Targeted service introduction based on geography or customer sets is possible. Services can be rapidly scaled up/down as required.

- Enables a wide variety of eco-systems and encourages openness. It opens the virtual appliance market to pure software entrants, small players and academia, encouraging more innovation to bring new services and new revenue streams quickly at much lower risk.

New trends in the Telco industry suggest that future networks are going to be virtualized, thus providing service from a cloud. Both fixed line networks and mobile networks will provide services from the cloud. There are few test implementations already running around the world like the TeraStream developed by Croatian Telecom (a part of Deutsche Telecom group) [24] and Virtual Evolved Packet Core (vEPC) use case by Telenor Group in Norway [25].

3.1.5 OpenStack Platform

OpenStack is an open source software platform launched by *NASA* (National Aeronautics and Space Administration) and *Rackspace Hosting* in 2010. Currently it is being managed by *OpenStack Foundation*, a non-profit organization which promotes *OpenStack* software and its community established in 2012. In a survey conducted by Linux.com and The New Stack 8 in August 2014, OpenStack was voted the top IaaS Free and Open-Source Software (FOSS) project of 2014 and the overall best FOSS Cloud Computing project [3]. More than 200 companies [26] are involved in the development of *OpenStack*. The goal of *OpenStack* is to produce the ubiquitous open source cloud computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable [27]. This is achieved by the architecture of *OpenStack*, which consists from multiple loosely coupled modules (projects), that provide specific services and are developed separately. *OpenStack* modules (Juno release 2014.2.3):

- *Nova* (Compute)
- *Neutron* (Networking)
- *Swift* (Object storage)
- *Cinder* (Block storage)
- *Keystone* (Identity)
- *Glance* (Image service)
- *Horizon* (Dashboard)
- *Ceilometer* (Telemetry)
- *Heat* (Orchestration)
- *Trove* (Database)
- *Sahara* (Data processing)
- a few others still under development

Nova is an OpenStack project designed to provide power massively scalable, on demand, self service access to compute resources [28]. Nova was originally focused purely on providing access to virtual servers running on a variety of different hypervisors. The majority of users use Nova only to provide access to virtual servers from a single hypervisor. It's possible to have a Nova deployment include multiple different types of hypervisors, while at the same time offering containers and bare metal servers [29]. Neutron is the networking project and it will be described in a separate chapter. Swift is a highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply [30]. One can create, modify, and get objects and metadata by using the Object Storage API, which is implemented as a set of Representational State Transfer (REST) web services. HTTPS protocol is used to interact with Object Storage, and standard HTTP calls are used to perform API operations [30]. Language-specific APIs, which use the RESTful API can be used to ease the integration with third party applications. An object stores data content, such as documents, images, etc. To assert your right to access and change data in an account, you identify yourself to Object Storage by using an authentication token [30]. To get a token, you present your credentials to an authentication service. Cinder is a Block Storage service for OpenStack. It's designed to present storage resources to end users that can be consumed by the OpenStack Compute Project (Nova). The short description of Cinder is that it virtualizes pools of block storage devices and provides end users with a self service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device [31]. Cinder volumes provide persistent storage to guest virtual machines (known as instances) that are managed by OpenStack Compute software [32]. Cinder can also be used independently of other OpenStack services. Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family [33]. It implements OpenStack's Identity API. Keystone is organized as a group of internal services exposed on one or many endpoints [34]. Many of these services are used in a combined fashion by the frontend, for example an authenticate call will validate user/project credentials with the Identity service and, upon success, create and return a token with the Token service [34]. Glance project provides a service where users can upload and discover data assets that are meant to be used with other services [35]. This currently includes images and metadata definitions. Glance image services include discovering, registering, and retrieving virtual machine images. Glance has a RESTful API that allows querying of VM image metadata as

well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple filesystems to object-storage systems like the OpenStack Swift project [35]. The Horizo project is a web based user interface and it will be discussed in a separate chapter. Ceilometer project aims to deliver a unique point of contact for billing systems to acquire all of the measurements they need to establish customer billing, across all current OpenStack core components with work underway to support future OpenStack components [36]. Heat is a service to orchestrate multiple composite cloud applications using the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API [37]. Heat provides a template based orchestration for describing a cloud application by executing appropriate OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates allow creation of most OpenStack resource types (such as instances, floating IPs, volumes, security groups, users, etc), as well as some more advanced functionality such as instance high availability, instance autoscaling, and nested stacks [37]. Trove is Database as a Service for OpenStack. It's designed to run entirely on OpenStack, with the goal of allowing users to quickly and easily utilize the features of a relational database without the burden of handling complex administrative tasks [38]. Cloud users and database administrators can provision and manage multiple database instances as needed. It focuses on providing resource isolation at high performance while automating complex administrative tasks including deployment, configuration, patching, backups, restores, and monitoring [38]. Sahara project aims to provide users with simple means to provision a Hadoop⁹ cluster at OpenStack by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more [39].

The conceptual architecture can be seen on Figure 11. OpenStack is written in Python¹⁰.

⁹ <https://hadoop.apache.org/>

¹⁰ <https://www.python.org/>

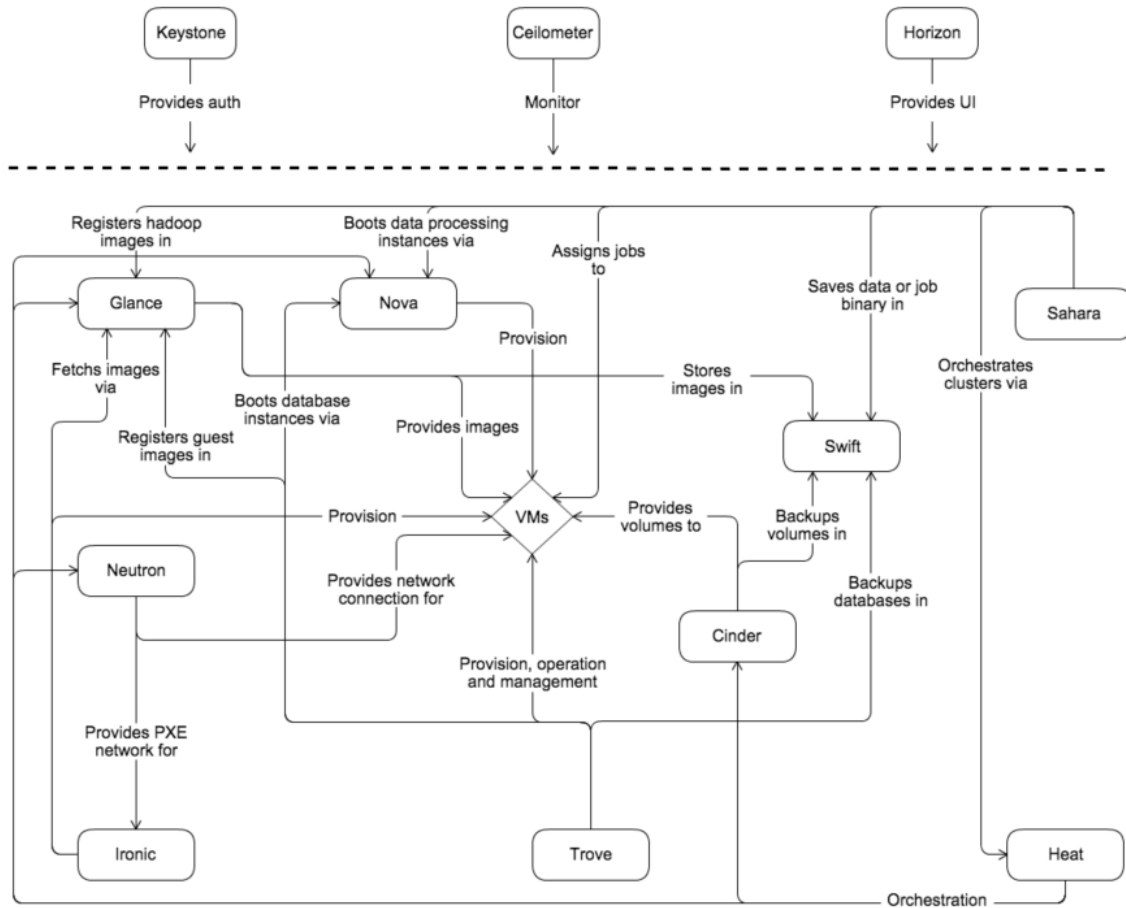


Figure 11 OpenStack conceptual architecture [40]

OpenStack enables engineers to deploy a multi-node IaaS, that provides an efficient, flexible and highly available service. The main three types of nodes are: the cloud controller node, the network node and the compute node. Figure 12 is showing a typical three node OpenStack deployment. The controller node is the central node which provides management for OpenStack. It handles the authentication and sending/receiving messages to other systems via an Remote Procedure Call (RPC) message queuing system over RabbitMQ¹¹. It can be split on several nodes to assure high availability of its services, but mostly it is deployed on a single node.

¹¹ <https://www.rabbitmq.com/>

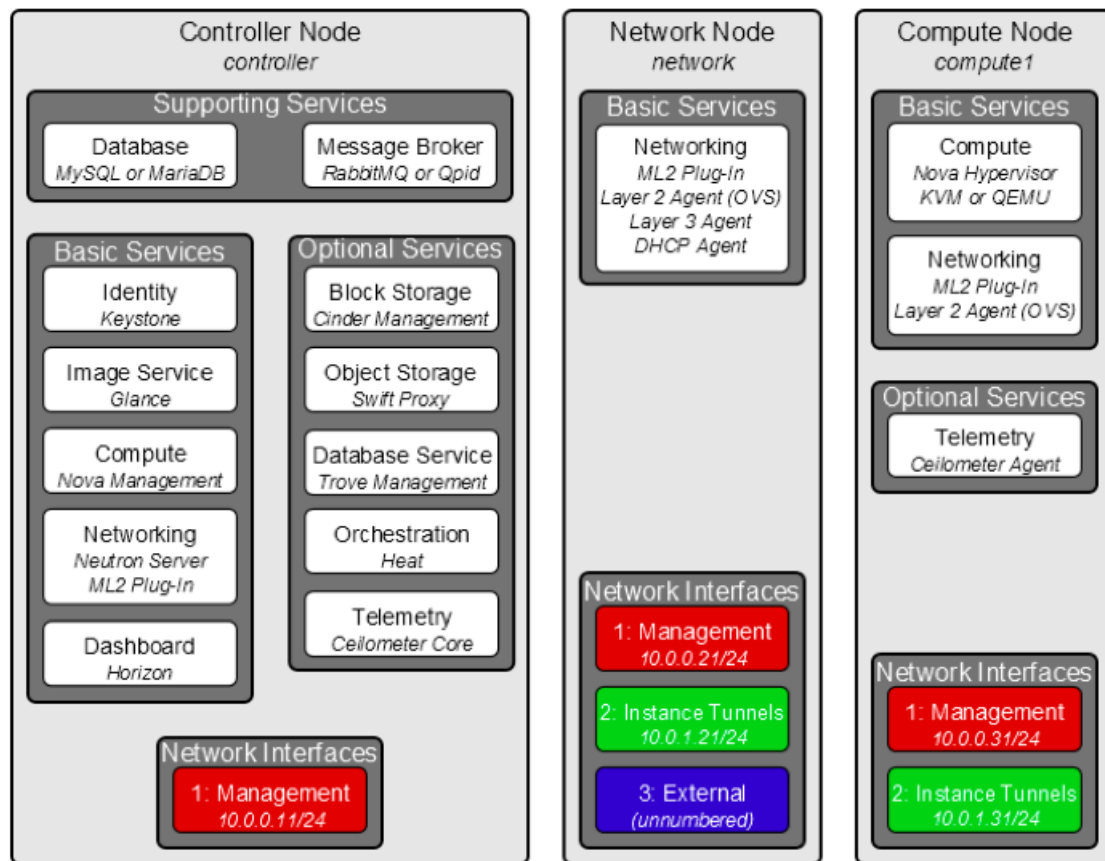


Figure 12 OpenStack deployment on three nodes [41]

The list of services that the controller node manages is shown below [42]:

- **Databases:** Track current information about users and instances, e.g., in a database, typically one database instance is managed per service.
- **Message queue services:** Advanced Message Queue Protocol (AMQP) messages for services are received and sent according to the queue broker.
- **Conductor services:** Provides a proxy service for the access to a database.
- **Authentication and authorization for identity management:** Indicates which users can do what actions on certain cloud resources.
- **Image-management services:** Stores and serves images with metadata, for launching in the cloud.
- **Scheduling services:** Indicates which resources to use first, e.g., spreading out where instances are launched based on an algorithm.
- **User dashboard:** Provides a web-based front end for users to consume OpenStack cloud services.
- **API endpoints:** Offers each service's REST API access, where the API endpoint catalog is managed by the Identity service.

The network node takes care of networking. It runs services like L3 agents that provide virtual routing entities. Typically it is directly connected to an external network in order to provide outside connectivity to VMs. Neutron services run on this node. This node can also be distributed to ensure high availability and better performance.

The compute node hosts VMs managed by Nova, and provides all the resources to run them. This kind of node is usually distributed to achieve high availability and better performance.

Every piece of code that is contributed to OpenStack passes through a system designed for reviewing and testing the code. It has to pass all automated tests before it gets reviewed by other developers. The code is accepted only if it passes all the tests, and if at least two reviews (from core reviewers) are positive. Reviewing all proposed changes is done by Core reviewers. Each project has its core team. A new member may be proposed on the openstack-dev list at any time. A proposal can come from anyone, but typically comes from the project's PTL (Project Team Lead) or an existing member of the core team. For the proposal to be accepted five existing members of the core team must vote for it. Any member of the team can put a veto on the nomination. The core team is consisted of experienced developers that can provide constructive and high quality reviews. The system uses Gerrit¹² for reviewing the code and Jenkins¹³ and Zuul¹⁴ for automated testing (Figure 13).

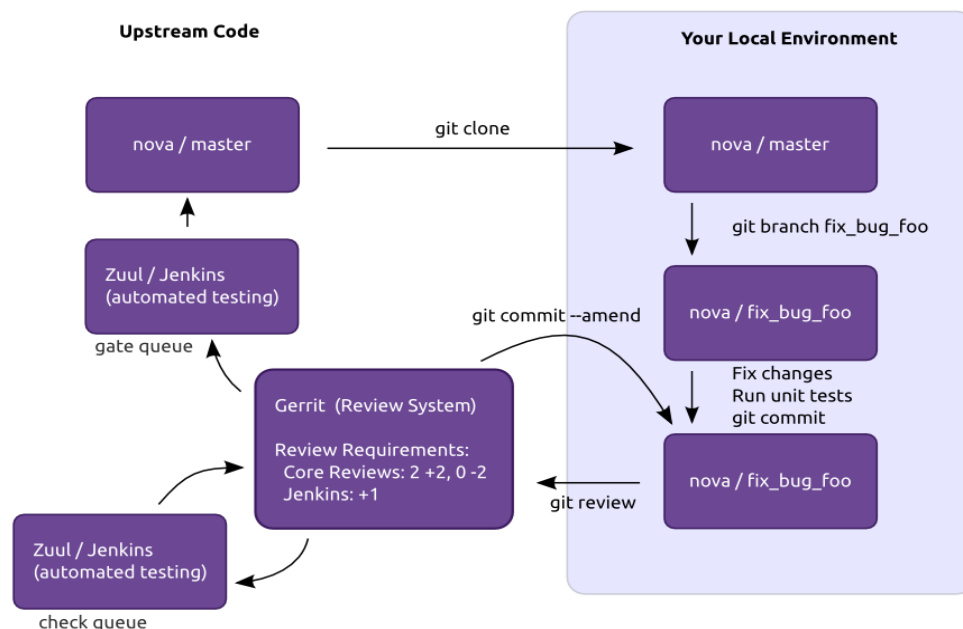


Figure 13 OpenStack contributing workflow [41]

¹² <https://code.google.com/p/gerrit/>

¹³ <http://ci.openstack.org/jenkins.html>

¹⁴ <http://ci.openstack.org/zuul.html#zuul>

3.1.6 OpenStack Neutron

This work is focused on networking, therefore the important *OpenStack* module regarding this work is *Neutron*, which provides "*networking as a service*" between interface devices (e.g., *vNICs*) managed by other OpenStack services (e.g., *Nova*). Neutron evolved from Nova, precisely from its networking part, which was used in the early days of OpenStack. It had a basic networking model supporting VLANs and Linux firewall called *iptables*. The OpenStack community realized that there was a need for a new approach. The networking part should have a separate networking project, enabling the development of new advanced networking services and features. Other benefits included the simplification of Nova and increased flexibility of OpenStack. All information about Neutron architecture applies to the Juno release 2014.2.3¹⁵. Neutron provides an API that lets you define network connectivity and addressing in the cloud. It also provides an API to configure and manage a variety of network services ranging from L3 forwarding and NAT to load balancing, edge firewalls, and IPsec VPN. Neutron API has virtual network, subnet, and port abstractions to describe networking resources:

- Network: An isolated L2 segment, i.e. a broadcast domain, analogous to VLAN in the physical networking world.
- Subnet: A block of v4 or v6 IP addresses associated with a Network. Default gateways, Domain Name Server (DNS) servers and other L3 specific attributes can be associated with a Subnet.
- Port: A virtual switch port on a logical network switch. Virtual instances attach their interfaces into ports. When IP addresses are associated to a port, this also implies the port is associated with a subnet, as the IP address was taken from the allocation pool for a specific subnet.

To make Neutron flexible it was designed to support plug-ins [43]. This is analogous to the concept of drivers for hardware devices in operating systems. There are two main types of plug-ins [41]: Core (provides L2 networking) and Service (provides additional network services). One example of core plug-ins can be vendor plug-ins, i.e., software that includes code to communicate with the specific vendor's equipment and translate the virtual resources into real, probably physical, network topologies. An example of a Service plug-in is the L3 plug-in that provides routing, NAT and floating IP functionality in Neutron networks. Other

¹⁵ <https://wiki.openstack.org/wiki/Releases>

service plug-ins include Firewall as a Service (FWaaS), Load Balancer as a Service (FWaaS) and VPN as a Service (VPNaaS). Multiple approaches to designing a plug-in are shown on Figure 14. A single plug-in (on the left) extends the core and implements L3 and firewall services. In the centre we have a core plug-in which implements the L3 service, but also has a distinct plug-in for other services. The last approach uses distinct plug-ins for each extension.

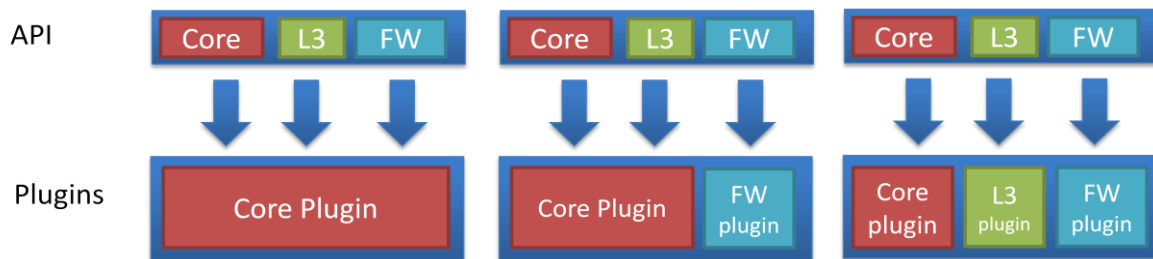


Figure 14 Plug-in design [44]

There are three different types of extensions to Neutron:

- **Resources:** New entities, like networks, subnets or ports are added to the Neutron Core Networking API.
- **Actions:** New operations that can be executed on resources. There are no core actions, but extensions can define new ones.
- **Request :** New attributes are added to existing resources. They are called that way because they extend on what API requests can provide.

The main core plug-in developed by the Neutron group is called ML2 [45], which stands for Modular Layer 2. It's open, generic and extensible, unlike vendor specific plug-ins. The ML2 plug-in is a framework allowing OpenStack Neutron to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world data centres [43]. During the development of ML2 much effort was put into reducing the code. Developers noticed that every plug-in that is being developed uses similar resources, like data structures that describe VLANs. Taking this into account they separated the code for data types from the code responsible for the functionality. This was made possible by specifying two kinds of drivers [45]. Type Drivers are responsible for handling different network types and their data types. Mechanism Drivers are responsible for taking the information established by the Type Driver and ensuring that it is properly applied given the specific networking mechanisms, that have been enabled. LinuxBridge and Open vSwitch are two examples of a Mechanism Driver. Network types that are supported by ML2 Type Drivers are [45]:

- Local: Network implemented on a single Linux host without recurring to special technologies.
- Flat: Enables the use of the underlying network on which Neutron is installed.
- VLAN: Networks that are implemented and isolated using VLAN technology, available at the OS.
- GRE: Networks isolated using GRE tunnel technology available at the OS.
- Virtual Extensible LAN (VXLAN): Networks isolated using VXLAN tunnel technology available at the OS.

Neutron is already very complex by itself but is a crucial service for the deployment of modern IaaS, including the ones that can be used by Telecommunications Operators (NFV). In order to be able to support the whole OpenStack deployment Neutron maintains multiple independent networks. An example architecture deployed on three nodes is shown on Figure 15.

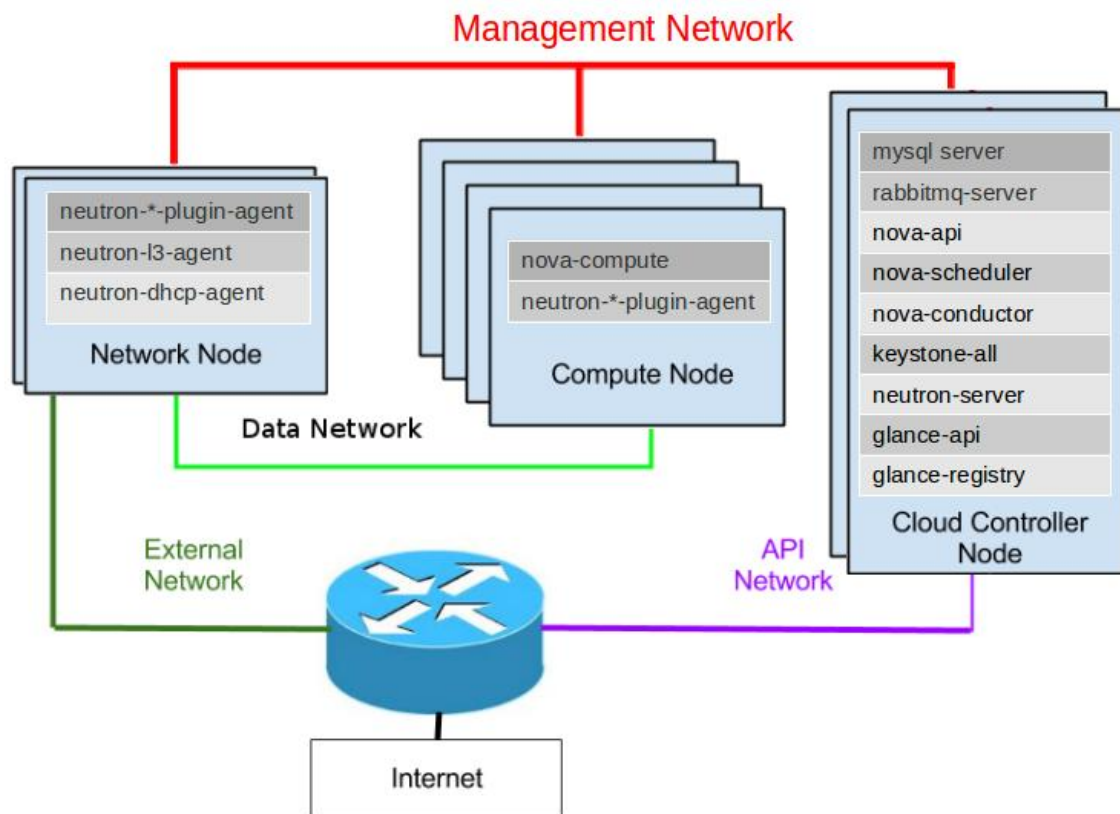


Figure 15 OpenStack deployment on three nodes [3]

The *Management Network* [46] is a private network that connects all OpenStack nodes in the same broadcast domain. This network is used for administration and monitoring of the systems. The usage of a separate, isolated network, without access to the Internet was chosen

to avoid the disruption of control traffic by tenants' traffic and because of the improved security that it provides. This network can be further split in other private networks for dedicated communication between specific internal components of OpenStack, e.g. message queues. There will usually exist a dedicated network switch to interconnect all nodes' physical NICs, which may also employ VLANs for the purpose of segregating the private network into smaller, more specific ones, as mentioned above. The *Data Network* [46], also known as the *Instance Tunnels Network*, is deployed in order to connect different Compute and Network nodes. This is the case when joint Compute and Network nodes (logical) are situated on multiple physical nodes. The traffic between VMs running on different Compute nodes passes through this network. The advantage is again the fact that it doesn't interfere with other traffic. The *External Network* [46] is directly connected to Network nodes and provides access to the Internet. This network is connected to a virtual router running inside Network Nodes, which then allow tenants' VMs to access the Internet (or just the external Network) by having these also connected to the virtual router. This router has the purpose of enabling the access to the Internet, by providing the instances with public IP addresses. Basically it does the job of a NAT server, with Dynamic NAT (DNAT) support. It is provided to Neutron as an L3 plug-in. The Network Node provides a pool of public IP address, known as floating IPs, which may be assigned to instances, thus supporting Static NAT (SNAT), enabling them to be accessed from the Internet (if the tenant so desires). Traffic from instances, having Internet as the destination, will come from the Instance Tunnels Network, as previously explained, reach the relative Network Node, cross the virtual router, and finally head out to the External Network. By having a dedicated network to access to exterior of OpenStack, security concerns in terms of malicious tenants' attacks can be alleviated. It also gives space to a dedicated API network, which may or may not reach the Internet as well, but is secure from tenants' attacks and or other disadvantages of sharing a network for tenancy and administration purposes. The *API Network* is supposed to provide a secure communication channel for outside API calls.

3.1.7 OpenStack Horizon

Horizon [47] is the project that implements the OpenStack dashboard (Figure 16). It provides a web based user interface to OpenStack services (projects). It evolved from an app that was used to manage Nova (compute service). Gradually the support for other projects and APIs was added. Horizon was designed to be extensible and easily manageable. The main goal was to support all core projects with a consistent user interface that uses reliable APIs

and provides backward compatibility. The OpenStack dashboard provides administrators and users with a graphical interface to access, provision and automate cloud-based resources. The extensible design makes it easy to plug in and expose third party products and services, such as billing, monitoring and additional management tools. The dashboard is also brandable for service providers and other commercial vendors who want to make use of it. The dashboard is just one way to interact with OpenStack resources. Developers can automate access or build tools to manage their resources using the native OpenStack API or the EC2¹⁶ compatibility API.

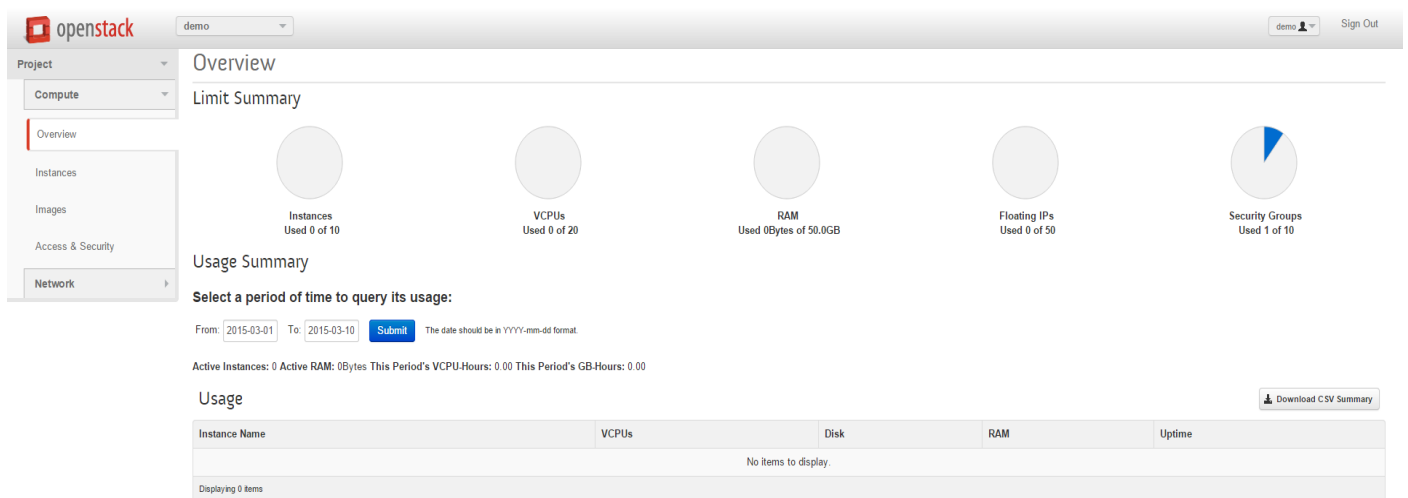


Figure 16 OpenStack dashboard

Horizon is built using Django. The dashboard application is based on the "*Dashboard*" class that provides an API for both core OpenStack apps and third party apps. The code is divided into reusable modules with logic, interaction with APIs, and presentation, to make customization in different sites easier. The Figure 17 shows the UI structure of Horizon. The Dashboard class is a top-level navigation item which contains Panel Groups, which contain different Panels. They are all displayed on the left sidebar. The sidebar functions like a menu, which displays only the Panel Groups of the currently selected dashboard. Also, only the Panels from the currently selected Panel Group are displayed to provide better user experience. Developers can either extend the functionality of existing Dashboards or create a new Dashboard. There is a simple registration pattern that enables adding a new Panel to a Panel Group, and respectively a new Panel Group to a Dashboard. Each Panel contains the

¹⁶ Amazon Elastic Compute Cloud, <http://aws.amazon.com/ec2/>

necessary logic for that interface (views, forms, tables, etc.). This principle of code organization is used in order to prevent the existence of large files that have thousands of lines of code and are therefore incomprehensible and hard to maintain. The code responsible for some functionality is easy to find by correlating it directly to navigation. Horizon provides core classes and templates that accelerate the development and provide consistency across applications.

Horizon UI Structure

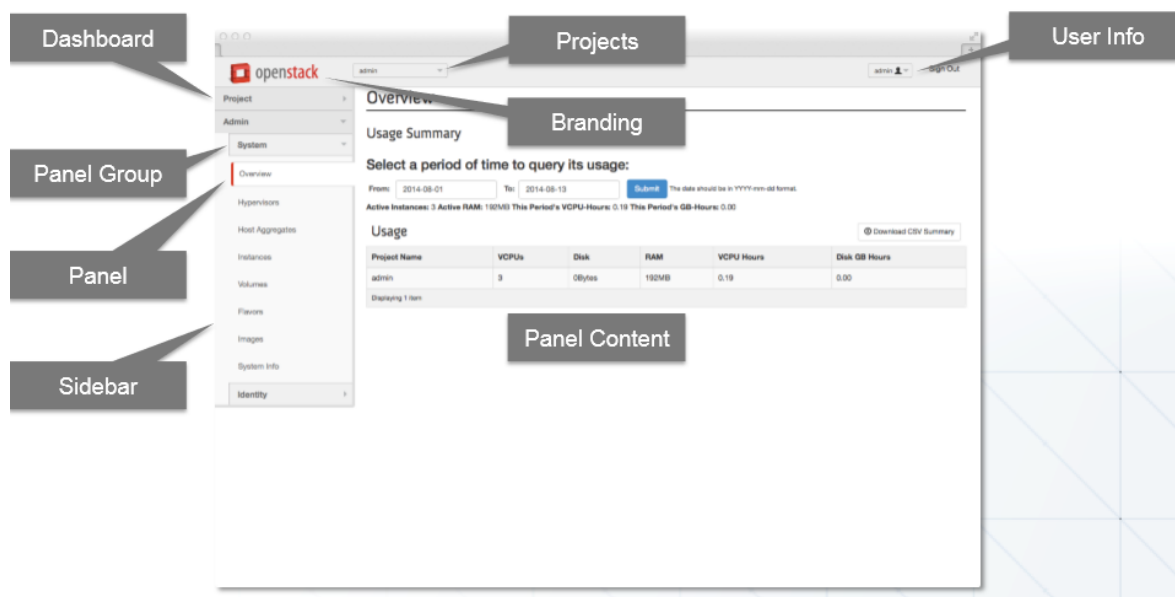


Figure 17 Horizon UI structure [48]

The structure of Horizon is a bit different than other OpenStack projects, as it has two main components: *horizon* and *openstack_dashboard*. The *horizon* directory contains generic libraries and components that can be used in any Django project. The *openstack_dashboard* directory contains a reference Django project that uses *horizon*. The structure of an application can be seen on Figure 18, where the tree of the *Project Dashboard* is shown.

```

project/
|---__init__.py
|---dashboard.py <-----Registers the app with Horizon and sets dashboard properties
|---overview/
|---images/
|   |-- images
|   |-- __init__.py
|   |-- panel.py <-----Registers the panel in the app and defines panel properties
|   |-- snapshots/
|   |-- templates/
|   |-- tests.py
|   |-- urls.py
|   |-- views.py
|   ...
...

```

Figure 18 Horizon app structure

An example of a Dashboard class is shown in Figure 19. The Dashboard consists of Panel Groups, which have to be explicitly declared in order to be displayed on the Dashboard. Following the Django conventions every class has a *name* and a *slug* attribute. The name is rendered and displayed on the web page, while the slug is used in the url.

```

import horizon

....
# ObjectStorePanels is an example for a PanelGroup
# for panel classes in general, see below
class ObjectStorePanels(horizon.PanelGroup):
    slug = "object_store"
    name = _("Object Store")
    panels = ('containers',)

class Project(horizon.Dashboard):
    name = _("Project") # Appears in navigation
    slug = "project"    # Appears in URL
    # panels may be strings or refer to classes, such as
    # ObjectStorePanels
    panels = (BasePanels, NetworkPanels, ObjectStorePanels)
    default_panel = 'overview'
    ...

horizon.register(Project)

```

Figure 19 Dashboard class example

The *panel.py* file referenced bellow (Figure 20) has a special meaning. Within a Dashboard, any module name listed in the '*panels*' attribute on the dashboard class will be auto-discovered by looking for the *panel.py* file in a corresponding directory. In order to enable the display of the Panel in a Dashboard the Panel has to be registered with that Dashboard. Panels can also require permissions for running, e.g., to be displayed only to the admin and hidden from ordinary users.

```

import horizon

from openstack_dashboard.dashboards.project import dashboard

class Images(horizon.Panel):
    name = "Images"
    slug = 'images'
    permissions = ('openstack.roles.admin', 'my.other.permission',)

# You could also register your panel with another application's dashboard
dashboard.Project.register(Images)

```

Figure 20 Panel class example

3.2 Related work

This chapter presents a short review of similar technology and existing solutions that are related to this work. They are explained in the following subchapters.

3.2.1 Group Based Policy

Group Based Policy (GBP) is an OpenStack project that delivers an intent driven model for describing application requirements in a way that is independent of underlying infrastructure. GBP introduces a generic “Group” primitive along with a policy model to describe connectivity, security, and network services between groups [49]. This replaces the network-centric constructs like Layer 2 domains. A Group is basically a collection of Neutron ports. Currently it is focused on the networking domain, with the prospect of becoming a generic framework that extends beyond networking. It was designed to overcome the void between the application developers and infrastructure teams by capturing requirements of complex applications and deploying them on OpenStack clouds. GBP introduces a policy model to describe the relationships between different logical groups or tiers of an application. The primitives have been chosen in a way that separates their semantics from underlying infrastructure capabilities. Resources may be public or local to a specific tenant. GBP was designed to separate out application security requirements (i.e. who can talk to who) from network-specific requirements (what IP address ranges to use, where to draw network boundaries, etc.). The separation of these two concerns allows application, security, and operation teams to operate independently. GBP enables users to specify the relationships between different tiers of applications. This dependency map acts as documentation for the security requirements of the application and allows different tiers of the application to evolve separately. It also makes it extremely easy to scale and automate infrastructure [50]. GBP offers an abstraction for network services and allows users to describe requirements for

chaining multiple network services as part of an application deployment. This corresponds to the SFC model. Group-Based Policy offers a new policy API through multiple OpenStack interfaces including Horizon (group-based-policy-ui), Heat (group-based-policy-automation), and CLI (group-based-policy-client) (Figure 21). It was designed to act as a layer on top of Neutron (and in the future other OpenStack services).

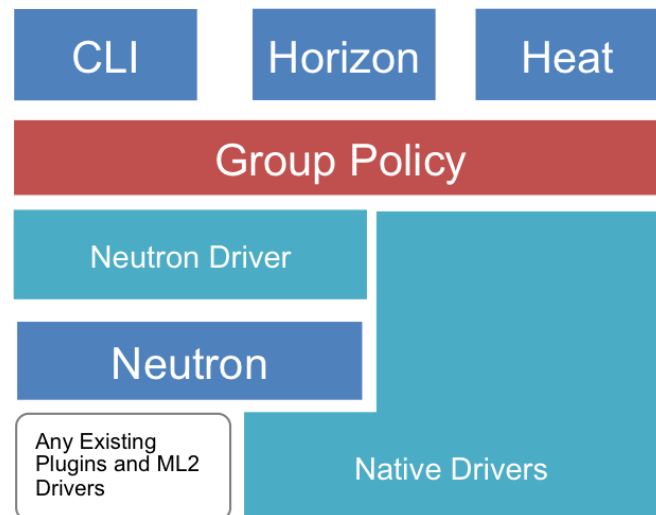


Figure 21 GBP architecture [50]

Two forms of mapping to underlying infrastructure are supported [50]:

- **Neutron Mapping Driver:** The Neutron mapping driver converts GBP resources into existing Neutron API calls. In the
- Table 2 one way to map resources is shown. The resources mapping can be customized. This allows Neutron to run any existing open source or vendor plugin, including ML2. It also allows GBP to be used in any OpenStack environment. At this time, use of both the GBP and Neutron APIs as end user facing APIs in parallel is not supported.
- **Native Drivers:** It is also possible to create drivers that directly render policy constructs through a separate SDN controller or external entity without first converting them to Neutron APIs. This is valuable as it gives the controller additional flexibility on how to interpret and enforce policy without being tied to L2/L3 behaviours. There are currently four native drivers including Cisco APIC, Nuage Networks, One Convergence, and OpenDaylight.

GBP Resource	Neutron
Policy Target	Port
Policy Target Group	Subnet
L2 Policy	Network
L3 Policy	Router

Table 2 Resources mapping

Network service chaining is a key capability of GBP. The goal is to describe the requirements for ordered chains of services by separating out network specific policies from service specific details. GBP resources related to network service chaining are displayed in Table 3.

GBP Resource	Description
Service Chain Nodes	Logical devices providing network services of a particular type (LB, firewall, etc.)
Service Chain Spec	Ordered grouping of service chain nodes. Specs may be used in the definition of a “redirect” action.
Service Chain Instance	Specific instantiation of service chain spec between Policy Groups. Instances are created automatically when a service chain is activated as part of a Rule Set.

Table 3 Network service chaining resources

GBP is an important project that has the potential to spread across the OpenStack domain and provide a solution for provisioning and management of all OpenStack components. Despite offering a different approach towards provisioning and managing networks (and other resources), GBP UI doesn't offer anything new and revolutionising that would give it extra value. Actually the UI is not intuitive, e.g., in the "*Firewalls*" panel, the tab order doesn't follow the workflow, i.e., first you have to create a firewall rule, then a firewall policy and in the end you can instantiate a firewall, but the tabs are arranged in the exactly opposite order. An addition to the GBP UI could be the implementation of the flowchart approach, which could prove to be a valuable asset.

3.2.2 Telco Cloud Environment

Service Function Chaining and its benefits are discussed in [16]. The authors present a new architecture for telecom operators based on cloud computing. They describe an approach that enables the deployment and management of Service Functions in a distributed cloud environment. The main focus of the paper are Service Functions, particularly virtual SFs. A Service Function (SF) is responsible for specific treatment of received packets. An example of a SF is the firewall, which receives packet, inspects them and acts according to the specified rules by either dropping the packets or letting them through.

The proposed architecture has four planes (Cloud4NFV platform) [16](Figure 22):

- Infrastructure plane: Handles the physical infrastructure.
- Virtual infrastructure management (VIM) plane: Handles the virtual infrastructure.
- Orchestration plane: Handles the provisioning, management and monitoring of SFs.
- Service plane: Handles the services built on Cloud4NFV.

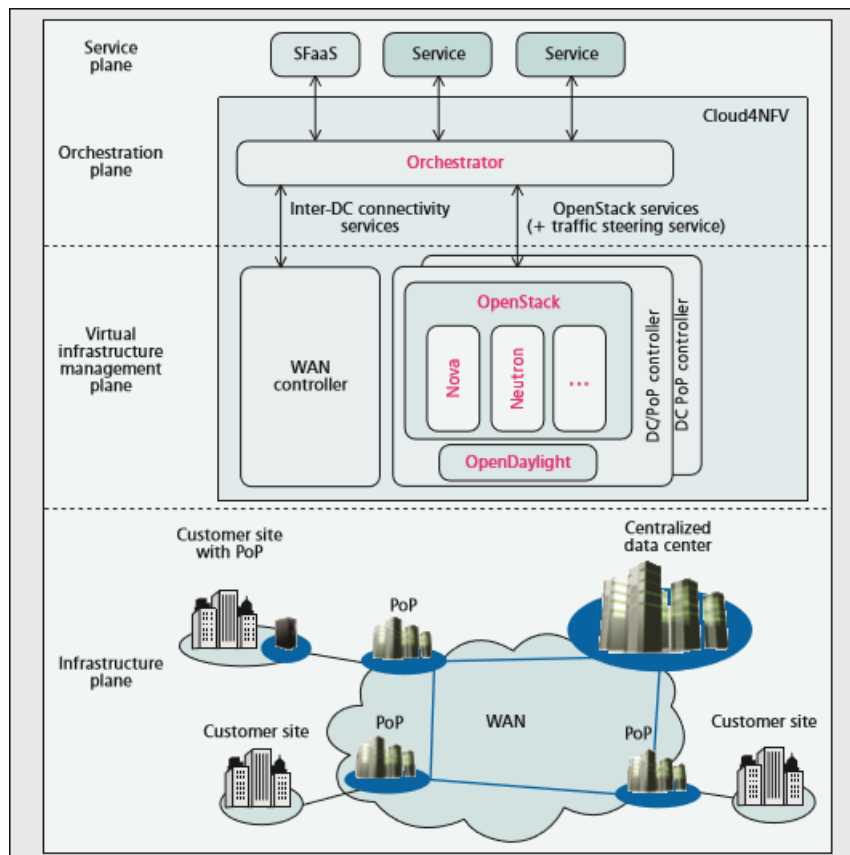


Figure 22 Cloud4NFV platform [16]

The orchestrator and the VIM plane are specially interesting. The orchestrator should provide an interface for creating, deleting and chaining SFs. The VIM plane and its components provide support for the orchestrator. OpenStack is used as a Data Centre controller, as well as the network manager. The authors argue that the OpenStack should provide only the basic networking tools, whilst advanced service should be deployed as VMs. An OpenDaylight module integrated into OpenStack Neutron enables the traffic steering between VMs. WAN controller manages the operator network. Two concepts are used to enable SFC: classification and traffic steering. Classification is a process of matching packets to a certain specification in order to correctly identify them and take the appropriate action. Traffic steering enables the redirection of traffic to a specific destination without modifying the network topology. The guidelines for a SFC solution are [16]:

- No assumption should be done on how functions are deployed, that is, whether they are deployed on physical hardware, as one or more VMs, or any combination thereof.
- An SF can be part of multiple SFCs.
- An SF can be network-transport-independent.
- An SFC allows chaining of SFs that are in the same layer 3 subnet and of those that are not.
- Traffic must be forwarded without relying on the destination address of packets.
- Classification and steering policies should not need to be done by SFs themselves.

4 A novel GUI for SFC

The objective is to create a GUI that enables Attachment Points (APs) provisioning and management as well as SFC provisioning and management. It has to be user friendly, intuitive and easier to use than the CLI. OpenStack was chosen for the implementation of this work because it is free, extensible and is driven by a very active community. This work was done on a machine running Linux, specifically Ubuntu Server 14.10¹⁷ using the Python Django web framework. All of the developed components should follow OpenStack guidelines¹⁸.

4.1 Django

Django¹⁹ is an open source Python framework for making web applications. It follows the MVC (model-view-controller) architectural pattern. MVC is a pattern of developing software so that the code for defining and accessing data (the model) is separate from request-routing logic (the controller), which in turn is separate from the user interface (the view). In every Django app there are three base files that provide the usage of the MVC architecture: *models.py*, *views.py* and *urls.py*. The *models.py* file contains a description of the database table, represented by a Python class. This class is called a *model*. Using it, you can create, retrieve, update and delete records in your database using simple Python code rather than writing repetitive SQL statements. The *views.py* file contains the logic for the page. The *urls.py* file specifies which view is called for a given URL pattern. A key advantage of such an approach is that components are loosely coupled. Each distinct piece of a Django-powered Web application has a single key purpose and can be changed independently without affecting the other pieces. For example, a developer can change the URL for a given part of the application without affecting the underlying implementation. A designer can change a page's HTML without having to touch the Python code that renders it. A database administrator can rename a database table and specify the change in a single place, rather than having to search and replace through a dozen files.

The fundamental principle embraced by Django is: "*Don't repeat yourself*". It's designed to enable rapid development and reusability of the code. It also follows the plug-in architecture and therefore enables easy integration of third party components (code). It can be run with

¹⁷ <http://www.ubuntu.com/download/server>

¹⁸ <http://docs.openstack.org/developer/horizon/contributing.html>

¹⁹ <https://www.djangoproject.com/>

Apache²⁰, NGINX²¹, Gunicorn²², Cherokee²³ or other WSGI (Web Server Gateway Interface) compliant web server. Four database backends are officially supported: MySQL²⁴, SQLite²⁵, Oracle²⁶ and PostgreSQL²⁷. There is a possibility to use Microsoft SQL Server on a Microsoft operating system with Django. NoSQL databases are supported in a fork called django-norel²⁸.

4.2 Extending the OpenStack Dashboard

In OpenStack, SFCs are created using the neutron-client. It provides means of classifying traffic via steering classifiers. In OpenStack a SFC is characterized by a port chain and one or more steering classifiers. A port chain is used to concatenate OpenStack port entities. This mechanism is used to provide connection between VNFs in OpenStack. The GUI must provide support for everything mentioned above. The mechanisms that neutron-client uses in order to achieve the desired functionality will not be explained, as they are not in the focus of this work. A detail explanation can be found in [3]. It is worth to mention that some minor additions to the neutron-client API also had to be done. Wrappers for some objects had to be added, so that the API methods return them in the proper format. The graphical user interface implementation is based on the Horizon project. Neutron-client API calls are executed through a web based graphical user interface. A table of developed components (Table 4) is shown below.

Developed Panels	Description
Attachment Points	Provisioning and management of APs
Service Function Chaining	Provisioning and management of steering classifiers, port chains and SFCs
Flowchart SFC	Drag and drop flowchart driven provisioning of SFCs

Table 4 Developed OpenStack Panels

²⁰ <https://httpd.apache.org/>

²¹ <http://nginx.org/>

²² <http://gunicorn.org/>

²³ <http://cherokee-project.com/>

²⁴ <http://www.mysql.com/>

²⁵ <http://sqlite.org/>

²⁶ <https://www.oracle.com/database/index.html>

²⁷ <http://www.postgresql.org/>

²⁸ <http://django-nonrel.org/>

In order to enable the connection to the operators network (according to the network architecture stated in previous chapter) a panel for managing Attachment Points (APs) had to be made. An AP is a logical entity that enables the extension of the virtual network segment that resides in the cloud with virtual or physical devices. Basically an AP will allow the end customer to connect his own home devices to the network in the cloud and gain access to the Internet. Some changes to the existing Network Topology panel were made in order to display the APs alongside the virtual networks that are in the cloud (Figure 23). These changes consist of modifying the JavaScript code that is used to receive and display network entities, as well as the backend Python code that delivers the data to the frontend.

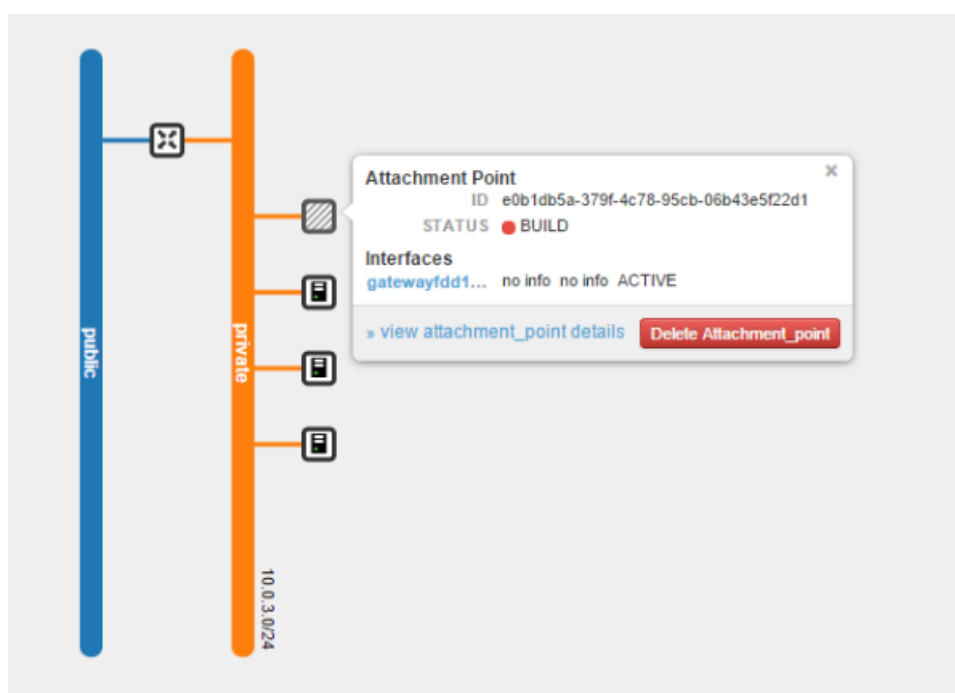


Figure 23 AP display in the Network Topology Panel

The AP panel provides the administrator with a simple GUI (Figure 24). The basic feature is the display of all APs that are shown in a table. The table is actually an instance of OpenStack DataTable²⁹ class which provides convenient, reusable API for building data-driven displays and interfaces. Actions can be defined and carried out on the entire table or only on the specific row. This provides simple means for deleting, attaching and detaching APs. The creation process is rather simple (Figure 25): the administrator has to click on the "Create Attachment Point" button which triggers a modal that contains a form, that contains fields that need to be filled (Figure 26). When all the fields are correctly filled the administrator just has to click on the "Create" button. The form is submitted to the backend where it is processed,

²⁹ <http://docs.openstack.org/developer/horizon/ref/tables.html>

followed by the neutron-client API call which creates the AP. All existing APs are displayed in the table.

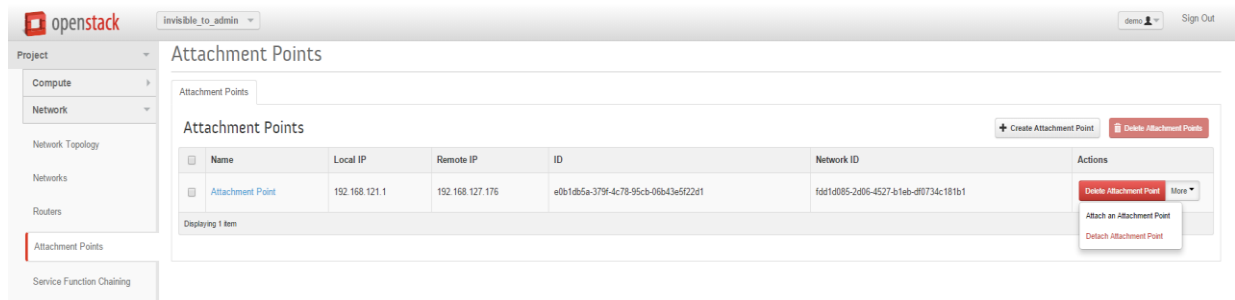


Figure 24 Attachment Points panel

Upon the creation the AP has no actual function, it needs to be associated with a network to provide the desired functionality. This process is called *attaching*. Once the AP is attached to a network, physical devices can be connected to the virtual network in the cloud. The attachment of the AP is done from the table using Horizon DataTable Actions. The workflow for attaching the AP to a network is the same as in the previous case, the administrator clicks on the action button which trigger a modal. The modal contains the AP data and a dropdown list, where the network can be chosen (Figure 27).



Figure 25 Attachment Point creation workflow

Figure 26 Create AP modal

Attach Attachment Point

Attachment Point Name: *

Attachment Point

Local IP: *

192.168.121.1

Remote IP: *

192.168.127.176

ID: *

e0b1db5a-379f-4c78-95cb-06b43e5f22d1

Network: *

private

Cancel Attach Attachment Point

Figure 27 AP Attach modal

Each end customer has a dedicated virtual network in the tenant virtual environment. A basic device that can handle L2 traffic and the connection to the operator's cloud is all what the end customer needs. Two more panels dedicated to SFC were developed. The first one was conceived to be the centre point for SFC provisioning and management, utilizing the Horizon tabs to provide multiple views to the administrator. The functionality that the second SFC panel provides was supposed to be embedded in one of the tabs in the first panel, but because of the unresolved problems caused by adding custom JavaScript (JS) scripts to the panel the decision to create a new panel had to be made. In the SFC panel there are two tabs: steering classifiers tab and port chains. In the first tab steering classifiers are managed according to the same principle as in the AP panel. The same goes for the second tab and port chains. The workflow of creating a SFC consists of two steps:

- Create one or more steering classifiers
- Create a port chain and associate it with one or more steering classifiers

Steering classifiers can be very specific, thus a number of parameters has to be taken into account while creating them. The classifier can specify a protocol, destination and/or source port, and even source and/or destination IP address. This can be useful for classifying traffic from a specific device. It is important to state that it is not necessary to fill all of the fields in the form. The ones that are obligatory are clearly marked (Figure 28).

Figure 28 Create a Steering Classifier modal

A SFC is created in the second tab (Figure 30). In order to create a SFC the administrator has to click the *"Create SFC"* button and then choose the first two members of the port chain, as well as one or more steering classifiers (Figure 29).

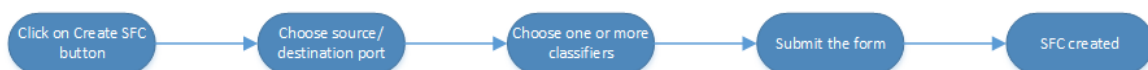


Figure 29 SFC creation workflow

If more than two ports are necessary, they can be added to the chain by the *"Add Port"* action in the table that displays SFCs. This approach was chosen because of the restrictions of Horizon. The initial approach was to dynamically add additional fields to the form in the modal (Figure 31) which is triggered by the *"Create SFC"* button. All fields in Django forms that are rendered in modals have to be specified upfront. As the number of ports in the port chain can't be predicted, the two port configuration was chosen as default. The problem with adding additional fields dynamically only becomes apparent when the form is submitted.

Namely the Python code that handles the form processing, ignores any additional data provided. It only takes into account the data from the fields that were specified before the modal was rendered. So the approach with adding port by port to the chain had to be taken (Figure 32). This approach is somewhat clumsy and impractical for longer chains.

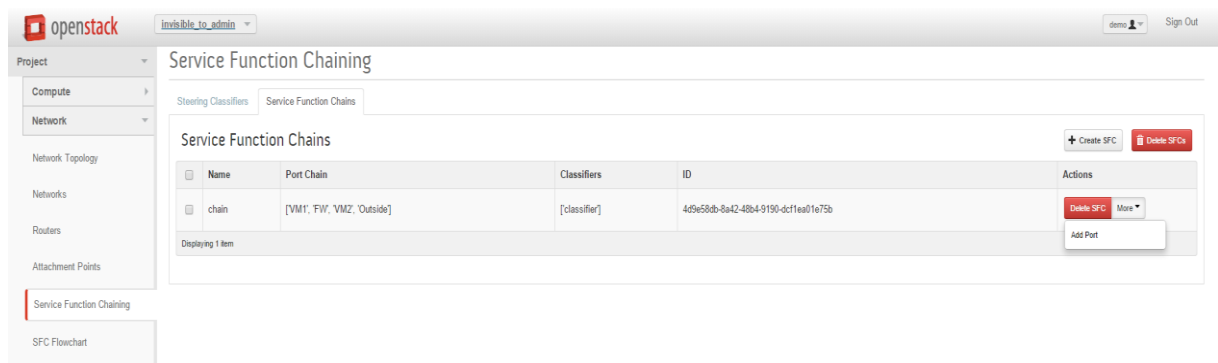
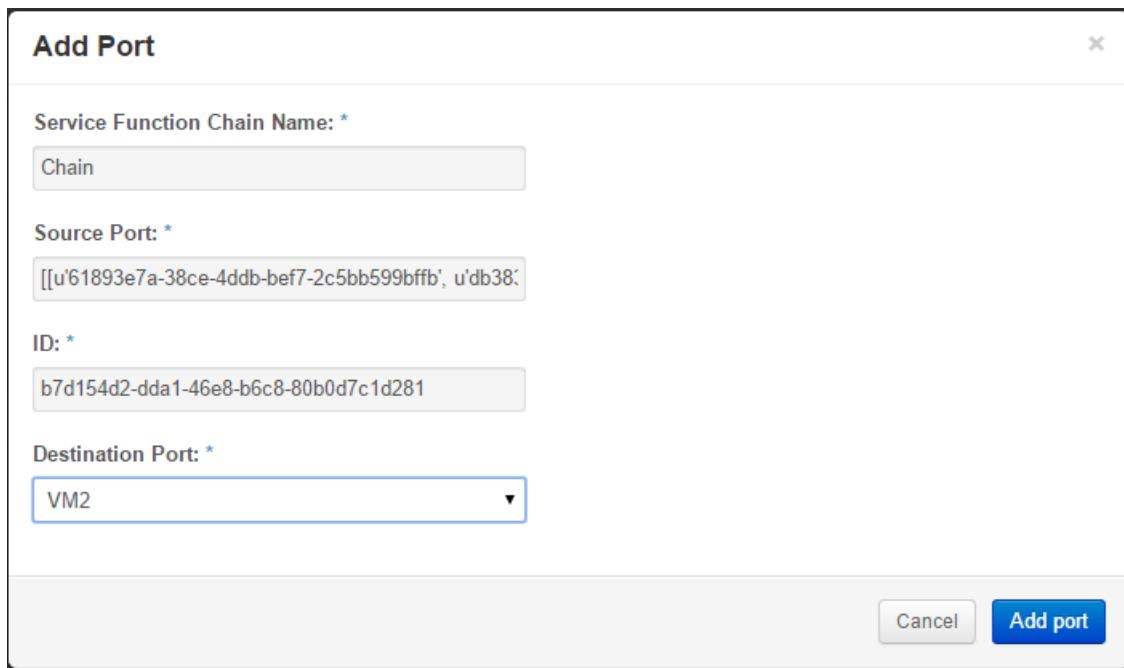


Figure 30 SFC tab

The screenshot shows a modal window titled 'Create Service Function Chain'. It contains the following fields and controls:

- Service Function Chain Name: ***: A text input field.
- Source Port: ***: A dropdown menu with 'VM1' selected.
- Destination Port: ***: A dropdown menu with 'FW' selected.
- Classifier: ***: A checkbox labeled 'classifier'.
- Buttons**: 'Cancel' and 'Create SFC' buttons at the bottom right.

Figure 31 Create SFC modal



Add Port [X]

Service Function Chain Name: *

Chain

Source Port: *

[[u'61893e7a-38ce-4ddb-bef7-2c5bb599bffb', u'db38:

ID: *

b7d154d2-dda1-46e8-b6c8-80b0d7c1d281

Destination Port: *

VM2 ▼

Cancel Add port

Figure 32 Add Port modal

The second SFC panel provides a different approach for the creation of SFCs. In order to simplify and ease the process a flowchart driven approach was taken (Figure 34). The concept consists of connecting "boxes" that represent VNFs, with the mouse and by that means creating a SFC. This could revolutionize the job of the administrator. The flowchart was implemented using the jsPlumb³⁰ library. This library provides support for connecting elements on web pages visually. It uses Scalable Vector Graphics (SVG) in modern web browsers. A button for creating a steering classifier was added to the panel to enable the execution of the whole workflow. The panel displays available VNFs as boxes that the user can drag around the screen. The workflow is displayed on Figure 33. Connection between the boxes can be made in the drag and drop fashion. The connections can be broken by clicking on the connection and choosing the option to delete it from the triggered popup. After the desired chain has been constructed the user has to click on the "*Create SFC*" button in order to specify the name of the chain and one or more steering classifiers (Figure 35).

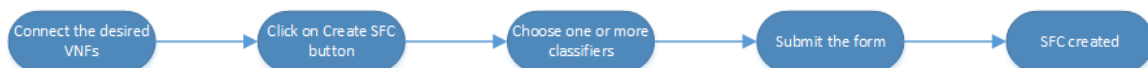


Figure 33 SFC creation via flowchart

³⁰ <http://www.jsplumb.org/demo/flowchart/dom.html>

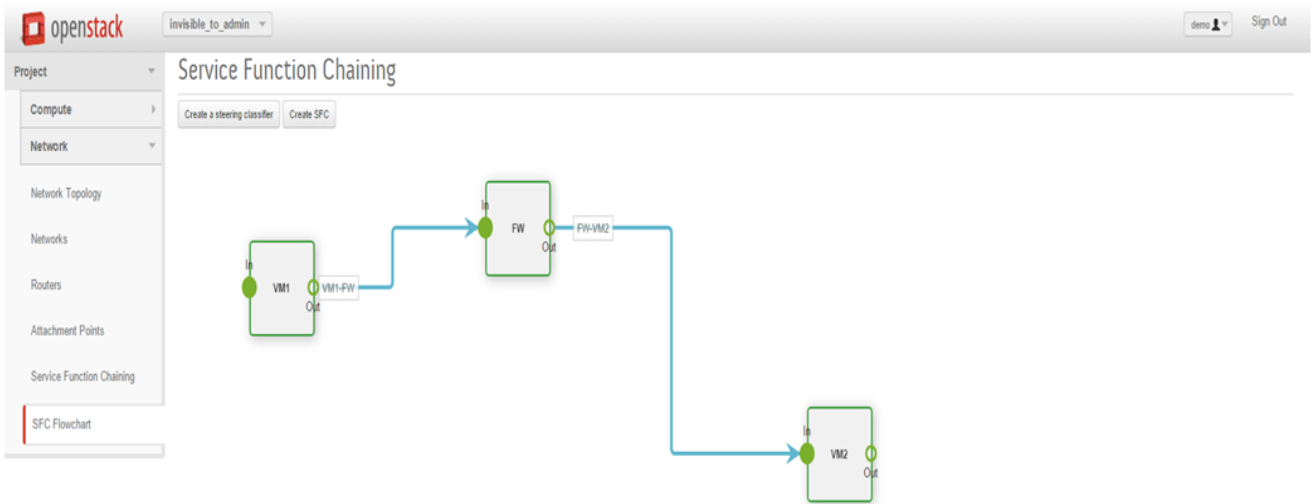


Figure 34 Flowchart Panel

Figure 35 Create SFC from a flowchart

4.2.1 Limitations

However this implementation has some limitations. VNFs in this implementations are actually OpenStack instances (VMs), that need to be provisioned before the creation of a SFC, in other words it is assumed that the instances already exist. There is also a limitation regarding the existence of an instance in a SFC. Current instances are aware of only one port, so until the support for multiple ports per instance is added, each instance can only exist in one SFC. This is a big limitation that has to be resolved in the future. Chaining of chains, i.e., adding a SFC inside another SFC is not supported (Figure 36).

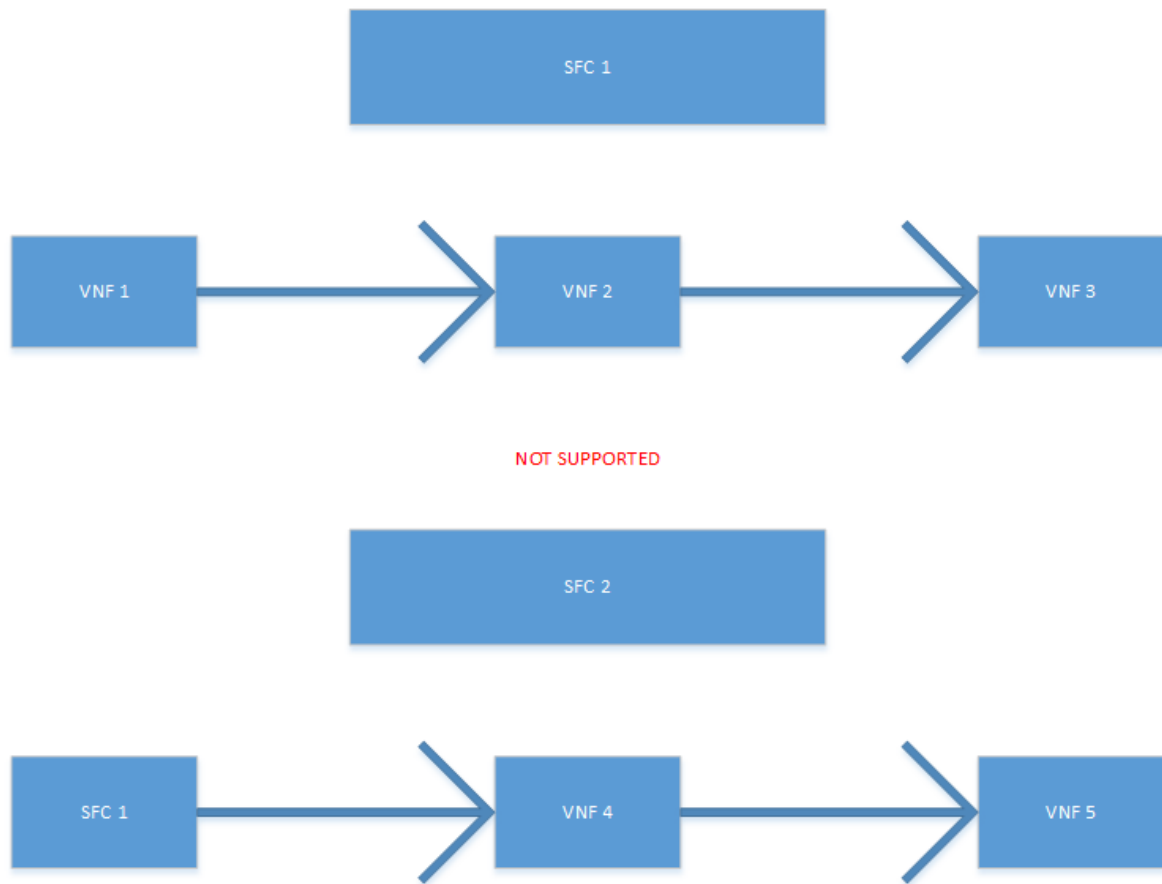


Figure 36 Chain of chains

The figure above depicts the scenario where a new chain (SFC 2) contains another chain (SFC 1). The packets that match the chosen steering classifier/s (for SFC 2) and go through SFC 1 are steered to VNF 4 and VNF 5 (SFC 2). The creation of such chains is currently not supported. The developed web interface doesn't support all functionalities of the neutron-client. These aspects need to be further developed. The problem with custom JavaScript scripts in tabs should be resolved and the flowchart approach should be implemented in the SFC panel in one of the tabs.

5 Evaluation and results

The objective is to test the developed solution and prove that it provides the desired functionality and is more user friendly, easier and intuitive than the CLI. To prove the concept an experiment that involves OpenArena³¹ was devised. Two players connected through the OpenStack network play a match (multiplayer mode) on an open server running on one of the machines (Figure 37, Figure 38). In order for the players to be able to connect to the cloud environment an Attachment Point has to be created. The Attachment Point is associated with a Raspberry PI device, that is connected to the OpenStack network and acts as a wireless access point. Both of the players are connected to the Raspberry PI. This game uses UDP packets to exchange information between all the players and the server. First a steering classifier that matches the data traffic specific for OpenArena (UDP port 27960) has to be created.

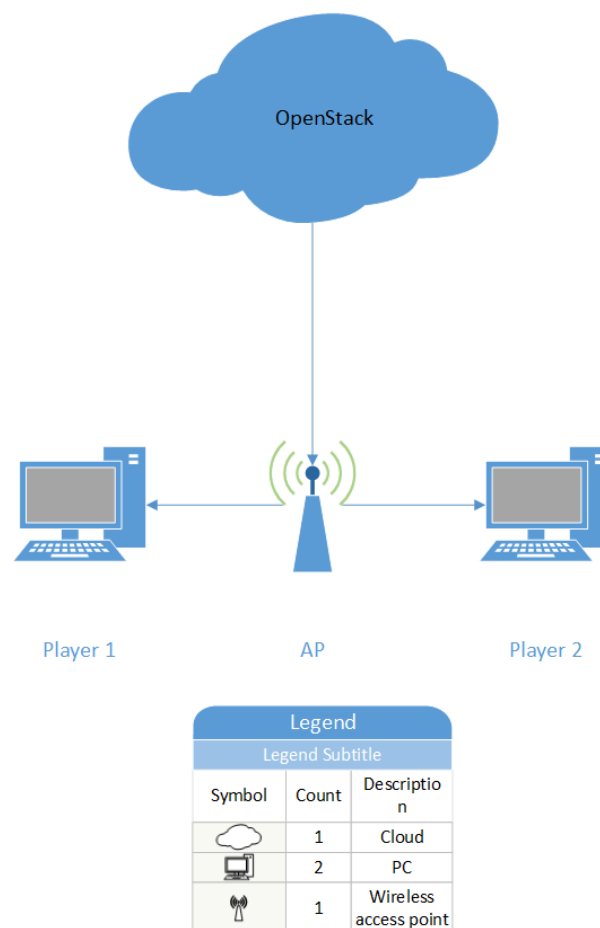


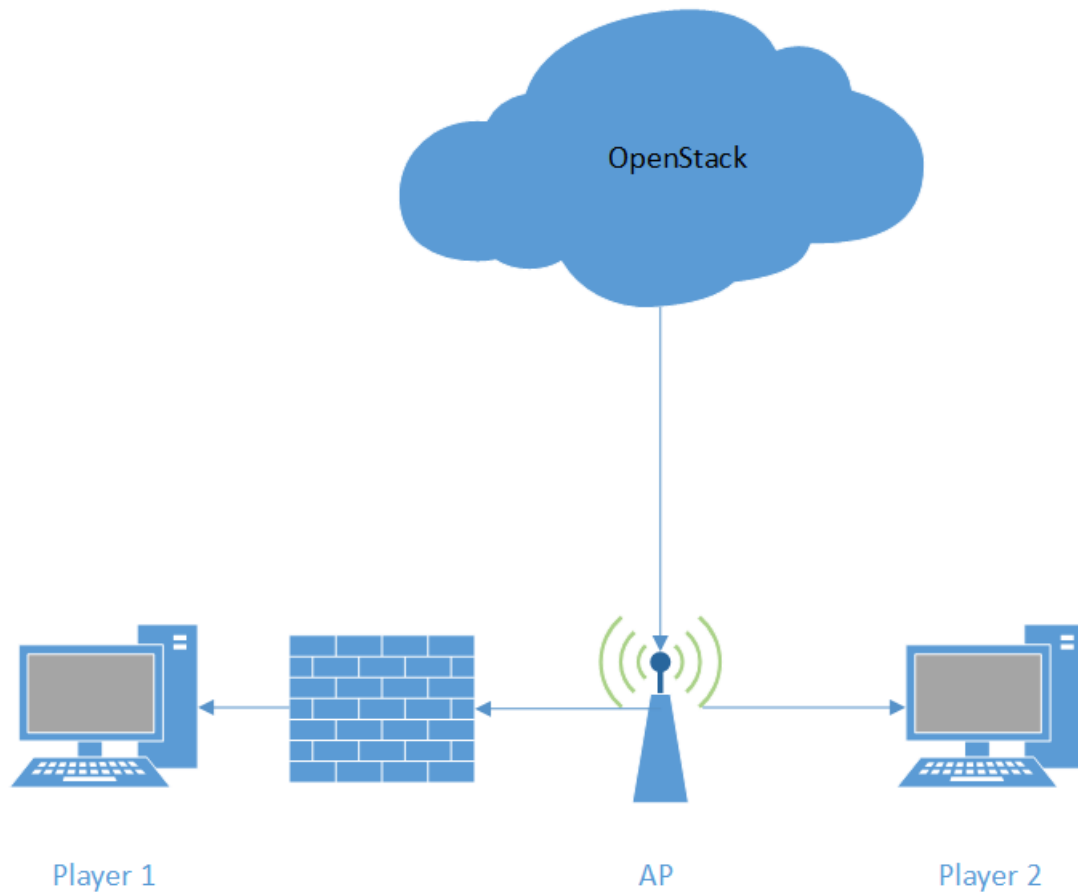
Figure 37 Demo network topology

³¹ <http://www.openarena.ws/smfnews.php>



Figure 38 OpenArena

In order to stop a player from playing the game a SFC has to be created. The traffic will be steered from the players computer through a firewall service (Figure 39). The traffic originating from a certain players computer is matched according to the classifier. As both players are connected through the same AP an IP address has to be specified in the steering classifier in order to block only the traffic from the specific player. The FW is an OpenStack instance with a predefined Linux system image. It has only the components needed to perform the function of a firewall. The FW had been preconfigured to drop UDP packets from port 27960. This was done by using the Linux iptables firewall utility. By associating the port chain with the steering classifier, a SFC is created. The player immediately loses the connection to the server and is unable to play the game (multiplayer mode) (Figure 40). But the second player can continue to play the game, despite being connected to the same network through the same Attachment Point, moreover the player that has seemingly lost the connection to the Internet can actually access the Internet.







Legend		
Legend Subtitle		
Symbol	Count	Description
	1	Cloud
	2	PC
	1	Wireless access point
	1	Firewall

Figure 39 Firewall blocks Player 1

By using other network services the player that lost the connection to the server can make sure that he still has a working connection to the Internet, and that only the OpenArena traffic is affected. The SFC has to be deleted in order to allow the first player to continue to play the game. Other experiments like blocking ICMP or HTTP protocol have also been carried out.



Figure 40 OpenArena connection interrupted

5.1 Evaluation

To evaluate the developed GUI one must ask and answer a set of questions. First of all does the GUI provide the same functionality as the CLI solution? Does it have some constraints? Is it better in some way? How does it change the workflow? Can it be modified? Can it be used by an inexperienced and untrained user? Does it follow the pattern of Horizon? How effective is it? How flexible is it? According to Nielsen [51] usability is a quality attribute that assesses how easy user interfaces are to use. He defines five quality components by which an UI can be assessed:

- Learnability : How easy is it for users to accomplish basic tasks the first time they encounter the design?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they re-establish proficiency?
- Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- Satisfaction: How pleasant is it to use the design?

Some of the well known usability problems like, design inconsistency, unclear step sequences, ambiguous menus and icons, and inadequate feedback and confirmation were considered during the evaluation of the GUI [52].

5.2 Overview

The first thing that this interface has to accomplish is to provide the administrator with the means to provision and manage APs. The interface provides the administrator the ability to create, delete, attach and detach APs. These are the most common operations done with APs, but it must be stated that not all supported neutron-client functionalities are available through GUI. That's because this implementation is supposed to explore the prospects of the GUI and show its possibilities, therefore only the most common operations were implemented. Full functionality can later be achieved by upgrading the GUI. This goes for all the developed components and not just for APs. The simplicity of working with the mouse and the visual representation of all the entities is truly revolutionary. It makes the job of the administrator more pleasant and interesting. While using the CLI it is common to use the IDs of various entities in order to properly identify them and accomplish the desired action. As the ID is a random generated string of 32 characters that presents a real problem. Also some of the commands that provide the display of data are really impractical and visually bad represented. Sometimes it is necessary to combine two or more of these in order to get the desired data. GUI has a clear advantage in the field of displaying the data, which is at most, one click of the mouse away. Also the details about the data structures and how they are connected are hidden from the user. All of the developed components follow the guidelines of Horizon. However some modifications could be made in order to further simplify the usage, e.g., the process of creating a SFC has a big requirement, as it expects that the instances (VMs) are already provisioned. One has to first provision instances in the "*Instances*" tab inside the "*Compute*" TabGroup and only after that a SFC can be created. The workflow could be modified so that one creates a SFC containing abstract VNFs and then the VNFs would be correlated with instances which would then be provisioned if they already haven't been. The most interesting part of GUI is the Panel with the flowchart. It provides means to create a SFC with the mouse, by connecting the VNFs visually. It could also display existing SFCs in a more intuitive visual way, rather than show a list of names that represent the connected VNFs.

5.3 Testing

A series of testing experiments with users were carried out in order to evaluate the developed GUI. The participants were given basic instructions describing what they should do, e.g., you have to create an attachment point and afterwards attach it to a network. In order to determine the efficiency, time that the participant takes to complete the specified task was

measured. These time is then compared with the time it takes to do the same action using the CLI. Several scenarios were taken into account:

- Creating an AP and attaching it to a network
- Creating a steering classifier
- Creating a SFC using the standard GUI
- Creating a SFC using the flowchart

The times were measured using a standard Android phone stopwatch application, with the precision of a tenth of a second. The number of mistakes the participants made was recorded. All the participant were given a short survey³² about the user experience (UX). The times measured are shown on the following figures: Figure 41, Figure 42, Figure 44 and Figure 46. All of the participants managed to accomplish all the tasks using the GUI.

5.3.1 Task 1: Creating an AP and attaching it to a network

All participants had a problem with completing the first task using the CLI. The CLI command for creating an AP requires parameters: driver, identifier and technology, that only a person with technical knowledge and previous experience with the system can know. When using the GUI these parameters are automatically specified. That was the reason why participants could not accomplish the task, therefore the measured times will not be compared. No mistakes were made during the execution of the first task, while using the GUI. The measured times are shown on Figure 41. Although the first task seems to be the simplest it proved to be unsolvable in the CLI case. The GUI case shows small variations in the time needed to solve the task. There are two results that significantly differ from others, because these participants were examining the whole GUI and not going straight to solving the task. They were also wondering how to fill the source and destination IP address parameter, and lost some time there. During this task the IP addresses were not important. They could have written any valid IP address. The same problem occurs while using the CLI. It's related to the implementation of the AP and not to the user interface.

³²Can be found in Appendix A

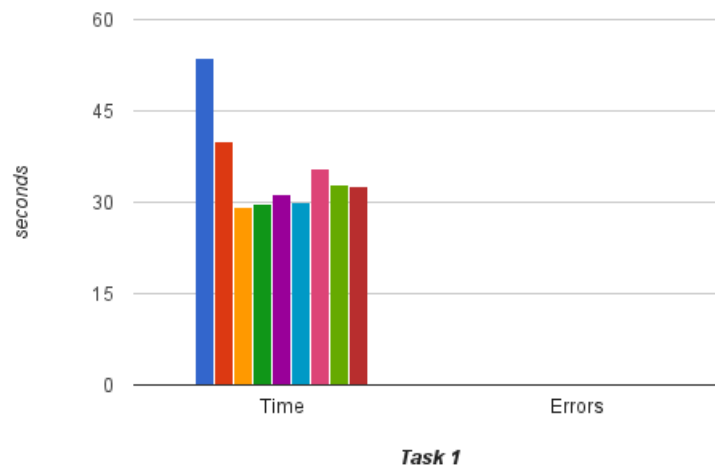


Figure 41 Task 1: Creating and attaching an AP

5.3.2 Task 2: Creating a steering classifier

The creation of a steering classifier seems to be simple, but it can easily become very complicated because of the numerous parameters, that can be specified. The results in the GUI case show big variations in time needed to solve the task (Figure 42). This phenomena can be easily explained, namely it is correlated with the number of parameters the participant tried to specify. Only one parameter is required, i.e., the name of the steering classifier. Other parameters such as: protocol, source IP address, destination IP address, source port range and destination port range are optional. If not specified the protocol parameters has a default value of 6, which is the IANA protocol number of TCP. The mistakes made by participants were related to the protocol specification, namely they didn't read the help text displayed besides the protocol field, where it is stated that the protocol is specified by the IANA protocol number. This problem can easily be eliminated by adding a dropdown menu with a list of protocols. This problem remains unsolved in the CLI case. This shows that the GUI has a clear advantage. The times on both cases can be seen on Figure 43.

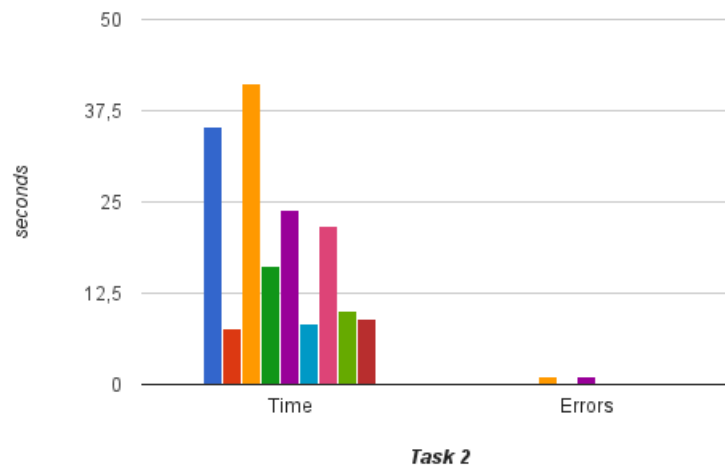


Figure 42 Task 2: Creating a steering classifier

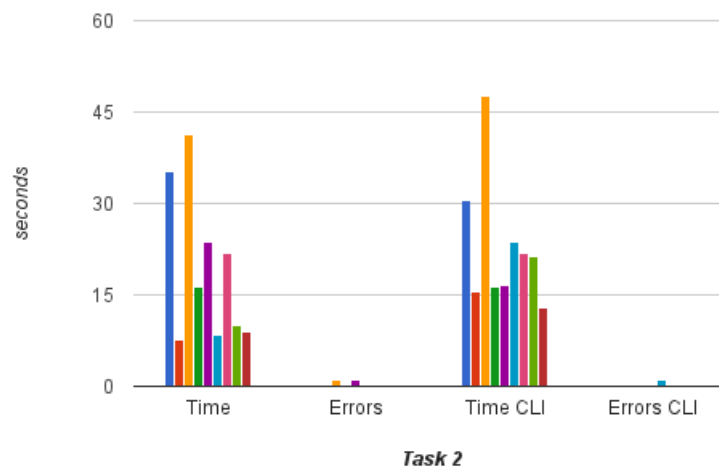


Figure 43 Task 2 comparison

As it can be seen on the previous figure the it takes more time to create a steering classifier in the CLI case. The participants first had to find the appropriate command in the list of commands, which was fairly easy, but still took some time. The ones that tried to specify more parameters took longer to accomplish the task. The main conclusion that can be drawn out of this is that an inexperienced user loses more time while figuring out how to use the commands in the CLI case, compared to the icons and buttons in GUI case. In average it took 19% more time to complete the task in the CLI case. The mistakes made in the CLI case were typos.

5.3.3 Tasks 3 and 4: Creating a SFC

Task 3 uses the standard GUI, which relies on the standard OpenStack Horizon workflow using modals, and DataTables. The last task is using the new flowchart approach, where the user connects the VNFs in a drag and drop fashion. The comparison between the two tasks is displayed on Figure 44.

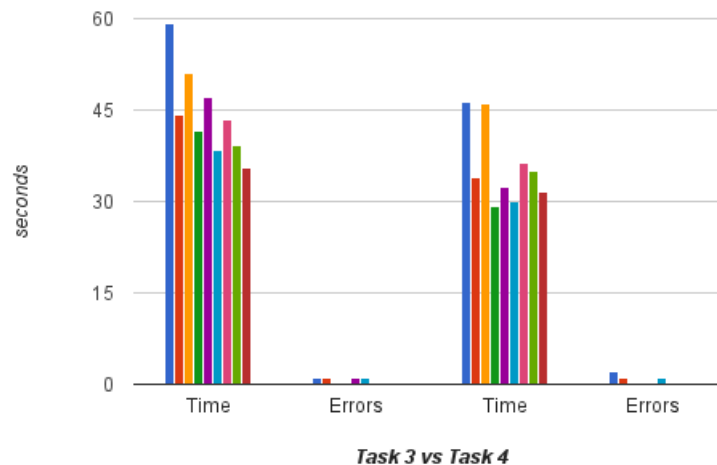


Figure 44 Task 3 vs Task 4: Creating a SFC

The results show that it takes less time to use the flowchart approach than the standard GUI approach. The flowchart approach is 24.5% quicker in average. Different type of mistakes were made while executing task 3 and task 4. The typical mistake while executing task 3 was forgetting to specify a steering classifier. Steering classifiers are shown in a list. Each one has a checkbox next to it. Some participants overlooked that the classifier is required. In the task 4 the participants were a bit confused with the two buttons: "Create a steering classifier" and "Create a SFC". Instead of going straight for the "Create a SFC" in some cases they wondered if they should create a new steering classifier. The steering classifier button is there in order to enable the whole workflow, i.e., creating a steering classifier first (if it doesn't exist) and creating a SFC (chaining VNFs and choosing a classifier). The CLI case proved to be a tough nut. The command was a bit harder to find because of the way the list of commands is displayed in the Linux terminal. Although the command and its parameters seem simple the task was hard to accomplish due to the nature of CLI. The command requires parameters: steering classifiers and ports, which have to be specified with their respective IDs. The problem is that IDs consist of random 32 characters and are therefore impractical for CLI usage. Participants first had to find the commands that will display steering classifiers and

ports in order to be able to execute the command which creates a SFC. These proved to be time costly and impractical. Also they had a problem with identifying the right ports in the list of ports. The ports are displayed with the name, ID, MAC address, subnet ID and IP address, which isn't helpful if you want to identify which port corresponds to which VNF (Figure 45). During the execution of this task the ports weren't important, but in a real world scenario it would be important to chain the ports that match the VNFs you want to chain. The GUI has a clear advantage over the CLI in this case.

id	name	mac_address	fixed_ips
3e042ea8-d05f-4fe5-bbae-f620f6411fe3		fa:16:3e:62:c4:16	{"subnet_id": "b47d8c9b-5124-4f9d-a99b-e823b0fc6bff", "ip_address": "10.0.3.4"}
61893e7a-38ce-4ddb-bef7-2c5bb599bfb		fa:16:3e:9c:e3:3c	{"subnet_id": "b47d8c9b-5124-4f9d-a99b-e823b0fc6bff", "ip_address": "10.0.3.6"}
6b522c51-10ea-4e07-9d8c-d93834c7a621		fa:16:3e:e4:db:a1	{"subnet_id": "606344b2-86dc-4ec0-839a-e4eada0ace74", "ip_address": "172.24.4.2"}
85370056-b65d-492c-8506-dc671543ed1c		fa:16:3e:fb:b7:70	{"subnet_id": "a390275f-3bb2-4ff6-866d-c0217449258e", "ip_address": "10.0.0.1"}
8f572ff5-45ef-4afe-aa1d-254caa4ad50f	myport	fa:16:3e:f5:fd:ee	{"subnet_id": "b47d8c9b-5124-4f9d-a99b-e823b0fc6bff", "ip_address": "10.0.3.5"}
9048b2bd-4d4e-4c2f-9ee2-5085780c11e9		fa:16:3e:b5:44:29	{"subnet_id": "a390275f-3bb2-4ff6-866d-c0217449258e", "ip_address": "10.0.0.3"}
99434a16-bea8-4650-a060-4106db12b008		fa:16:3e:1f:fd:ca	{"subnet_id": "b47d8c9b-5124-4f9d-a99b-e823b0fc6bff", "ip_address": "10.0.3.3"}
aa631efc-e89d-498e-99e5-0f6ba64acd29		fa:16:3e:6e:30:73	{"subnet_id": "606344b2-86dc-4ec0-839a-e4eada0ace74", "ip_address": "172.24.4.3"}
db3837ee-0e52-4f04-b1ae-f789fb19bd8c		fa:16:3e:9b:a6:1e	{"subnet_id": "b47d8c9b-5124-4f9d-a99b-e823b0fc6bff", "ip_address": "10.0.3.2"}
ff5a635f-3912-4bc0-98c6-99403b91b40e		fa:16:3e:48:7f:3d	{"subnet_id": "b47d8c9b-5124-4f9d-a99b-e823b0fc6bff", "ip_address": "10.0.3.1"}

Figure 45 Neutron port list

The CLI case is much slower than both of GUI cases (Figure 46). In average creating a SFC using the standard GUI is 362% faster than using the CLI. In the flowchart case it is 451% faster. A lot of mistakes were made in the CLI case. They were caused by the fact that a lot of commands and a lot of typing needed to be done in order to accomplish the task. The overall distribution of errors is displayed on Figure 47.

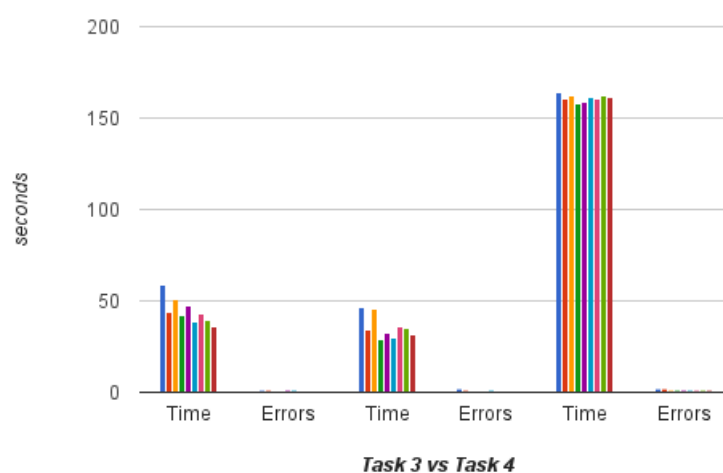


Figure 46 Task 4: Creating a SFC comparison

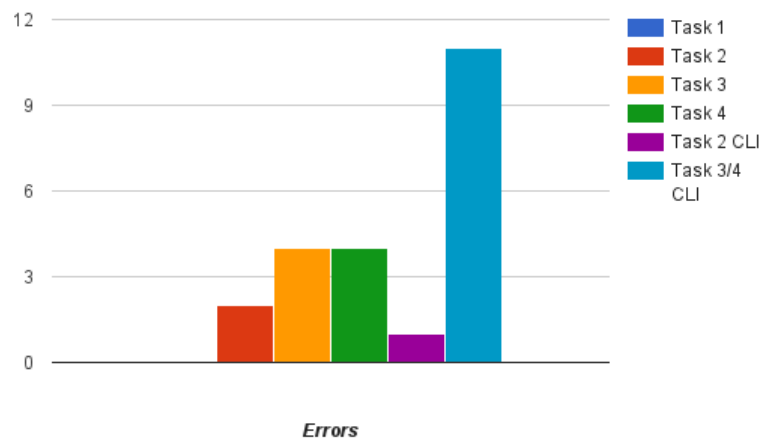


Figure 47 Error distribution

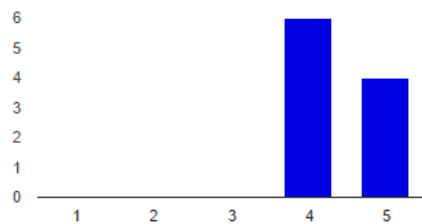
5.3.4 Survey

Are the menus ambiguous and confusing?



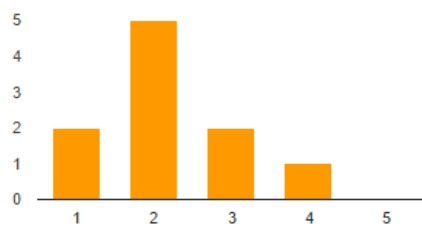
Yes	0	0%
No	10	100%

Is the Information displayed sufficient and helpful?



Insufficient/unhelpful:	1	0	0%
	2	0	0%
	3	0	0%
	4	6	60%
Completely sufficient/helpful:	5	4	40%

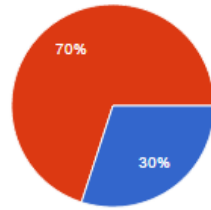
Was it difficult to memorize the workflow?



Very easy:	1	2	20%
	2	5	50%
	3	2	20%
	4	1	10%
Very difficult:	5	0	0%

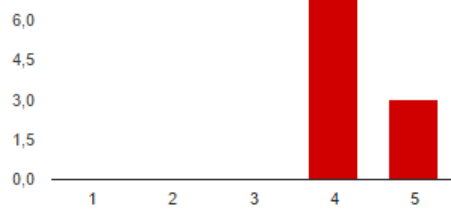
Figure 48 Survey results 1

Which chaining workflow do you prefer?



Standard GUI	3	30%
Flowchart	7	70%
CLI	0	0%

How satisfied are you with the design?



Unsatisfied:	1	0	0%
	2	0	0%
	3	0	0%
	4	7	70%
Completely satisfied :	5	3	30%

Which interface would you prefer for daily usage?



CLI	0	0%
GUI	10	100%

Figure 49 Survey results 2

The results of the UX survey are displayed on Figure 48 and Figure 49. The survey shows that participants prefer the GUI over CLI. In general they are satisfied with the design and the feedback provided by the GUI. The workflow is considered fairly easy to remember. The flowchart approach has proved to be more popular than the standard GUI.

6 Conclusion

This dissertation offers an insight into the exciting and promising field of network virtualization. In recent years this field has produced new concepts that have the prospect of changing the whole Telco industry. A lot of work still has to be done before this technology will be mature enough to replace the current network technologies, but it shows real promise. Network Function Virtualization could revolutionize the industry by reducing expenses, eliminating the need for proprietary hardware and enabling faster development and deployment of new services. This dissertation focuses on Service Function Chains. SFCs can take advantage of virtualization and be deployed in a cloud environment, instead of being embedded in a series of physical devices. This brings new opportunities as a change in the network topology, or new hardware deployment is no longer necessary in order to provide a new service. Both the operator and the customers benefit from it. Customers get flexible new services without the need for any assistance from the technicians (i.e., replacing the hardware or firmware) and the operator reduces the time-to-market and its expenses. OpenStack seems to be the platform for these future Telco cloud environments. OpenStack is open-source, flexible, scalable and easy to deploy. The OpenStack community is growing every day, with new projects that provide new functionalities. A GUI for managing and provisioning SFCs was developed in the scope of this dissertation. The developed GUI provides two approaches. The first one uses modals, data tables and follows standard OpenStack workflows. The second approach is a new idea that tries to benefit from a point and click flowchart approach. It is considered more user friendly and intuitive. It could revolutionize the everyday work of a network administrator. The flowchart model makes the chaining of NFVs and creating SFCs more intuitive than both the CLI and the GUI solutions that currently exist. This approach could in the future be used in other use cases. However this solution is far from perfect. Some features like automated VNF provisioning and creating a chain of composed of other chains, aren't supported yet. Also it implements only the mostly used functionalities of the CLI neutron-client tool. The implementation of all these functionalities is considered as important future work. A more extensive evaluation of the GUI should be made in order to improve it and the user experience. A contact with the OpenStack GBP group had been made, with the prospect of contributing to the GBP UI, i.e., the point and click flowchart approach could be implemented into the GBP UI. This would present the accomplishment of the ultimate goal of this dissertation.

Apendices

Appendix A: GUI evaluation survey

OpenStack SFC GUI Evaluation

*Obavezno

Are the menus ambiguous and confusing? *

- ☐ Yes
☐ No

Is the Information displayed sufficient and helpful? *

1 2 3 4 5

Insufficient/unhelpful ☐ ☐ ☐ ☐ ☐ Completely sufficient/helpful

Was it difficult to memorize the workflow? *

1 2 3 4 5

Very easy ☐ ☐ ☐ ☐ ☐ Very difficult

Which chaining workflow do you prefer? *

- ☐ Standard GUI
☐ Flowchart
☐ CLI

How satisfied are you with the design? *

1 2 3 4 5

Unsatisfied ☐ ☐ ☐ ☐ ☐ Completely satisfied

Which interface would you prefer for daily usage? *

- ☐ CLI
☐ GUI

Comments or suggestions

7 References

- [1] “8. NFV: Implementation and Deployment: European Future Internet Portal - the information hub for European R&D activities on the Internet of the future.” [Online]. Available: <http://www.future-internet.eu/home/future-internet-assembly/athens-mar-2014/8-nfv-implementation-and-deployment.html>. [Accessed: 02-Jun-2015].
- [2] E. G. N. 001 V1.1.1, “Network Functions Virtualisation (NFV); Use Cases,” vol. 1, pp. 1–50, 2013.
- [3] I. D. Cardoso, “Departamento de Eletrónica, Universidade de Aveiro Telecomunicações e Informática 2014,” 2014.
- [4] R. van den Lans, *Data Virtualization for Business Intelligence Systems*. 2012.
- [5] “Virtual Local Area Networks.” [Online]. Available: http://www.cse.wustl.edu/~jain/cis788-97/ftp/virtual_lans/. [Accessed: 30-Jun-2015].
- [6] Cisco, “How Virtual Private Networks Work.” [Online]. Available: <http://www.cisco.com/c/en/us/support/docs/security/vpn/ipsec-negotiation-ike-protocols/14106-how-vpn-works.html>. [Accessed: 30-Jun-2015].
- [7] “How VPN Works: Virtual Private Network (VPN).” [Online]. Available: [https://technet.microsoft.com/en-us/library/cc779919\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc779919(v=ws.10).aspx). [Accessed: 30-Jun-2015].
- [8] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN: An Intellectual History of Programmable Networks,” *ACM Sigcomm Comput. Commun.*, vol. 44, no. 2, pp. 87–98, 2014.
- [9] Etsi, “Network Functions Virtualisation (NFV); Architectural Framework,” vol. 1, no. 1, pp. 1–21, 2013.
- [10] V. Network and F. Architecture, “GS NFV-SWA 001 - V1.1.1 - Network Functions Virtualisation (NFV); Virtual Network Functions Architecture,” vol. 1, pp. 1–93, 2014.
- [11] “ETSI - NFV.” [Online]. Available: <http://www.etsi.org/technologies-clusters/technologies/nfv>. [Accessed: 02-Jun-2015].
- [12] K. Kirkpatrick, “Software-defined networking,” *Commun. ACM*, vol. 56, no. 9, p. 16, 2013.
- [13] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, 2013.

- [14] C. Pignataro and J. Halpern, “Service Function Chaining (SFC) Architecture draft-ietf-sfc-architecture-08.” [Online]. Available: <https://www.ietf.org/id/draft-ietf-sfc-architecture-08.txt>. [Accessed: 02-Jun-2015].
- [15] R. Chirgwin, “Telco heavyweights pass packets in NFV demo • The Register.” [Online]. Available: http://www.theregister.co.uk/2015/02/16/telco_heavyweights_pass_packets_in_nfv_demo/. [Accessed: 15-Jun-2015].
- [16] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, and J. P. Barraca, “Toward a Telco Cloud Environment for Service Functions,” no. February, pp. 98–106, 2015.
- [17] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” *Nist Spec. Publ.*, vol. 145, p. 7, 2011.
- [18] J. Votano, M. Parham, and L. Hall, “Essential characteristics of Cloud Computing,” *Chem. ...*, p. 6, 2004.
- [19] “Cloud Computing models.” [Online]. Available: <http://www.thoughtsoncloud.com/wp-content/uploads/2015/04/Cloud-computing-service-models.png>. [Accessed: 02-Jun-2015].
- [20] L. Wang, G. Von Laszewski, M. Kunze, and J. Tao, “Cloud computing: A Perspective study,” *Proc. Grid Comput. Environ. Work.*, vol. 28, pp. 1–11, 2008.
- [21] W.-T. T. W.-T. Tsai, X. S. X. Sun, and J. Balasooriya, “Service-Oriented Cloud Computing Architecture,” *Inf. Technol. New Gener. (ITNG), 2010 Seventh Int. Conf.*, pp. 684–689, 2010.
- [22] “Microsoft by the Numbers.” [Online]. Available: <http://news.microsoft.com/bythenumbers/index.HTML>. [Accessed: 02-Jun-2015].
- [23] “Cisco sets \$1 billion investment for global cloud computing network | Reuters.” [Online]. Available: <http://www.reuters.com/article/2014/09/29/us-cisco-systems-investment-cloud-idUSKCN0HO13T20140929>. [Accessed: 06-Jul-2015].
- [24] A. Clauberg, “Deutsche Telekom TeraStream : A Network Functions Virtualization (NFV) Using OpenStack Case Study,” pp. 4–7, 2014.
- [25] P. Grønsund, A. Gonzalez, and T. Asa, “NFV – Main Concepts , Business Perspectives and Dependability Modeling Use Case Assessment of NFV use cases Dependability modeling and assessment,” no. March, 2015.
- [26] “Companies » OpenStack Open Source Cloud Computing Software.” [Online]. Available: <http://www.openstack.org/foundation/companies/>. [Accessed: 02-Jun-2015].

- [27] “OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Main_Page. [Accessed: 02-Jun-2015].
- [28] “Welcome to Nova’s developer documentation! — nova 12.0.0.0b2.dev110 documentation.” [Online]. Available: <http://docs.openstack.org/developer/nova/#introduction>. [Accessed: 08-Jul-2015].
- [29] “Scope of the Nova project — nova 12.0.0.0b2.dev110 documentation.” [Online]. Available: http://docs.openstack.org/developer/nova/project_scope.html. [Accessed: 08-Jul-2015].
- [30] “Welcome to Swift’s documentation! — swift 2.3.1.dev127 documentation.” [Online]. Available: <http://docs.openstack.org/developer/swift/>. [Accessed: 08-Jul-2015].
- [31] “Cinder - OpenStack.” [Online]. Available: <https://wiki.openstack.org/wiki/Cinder>. [Accessed: 08-Jul-2015].
- [32] “What is Cinder (OpenStack Block Storage)? - Definition from WhatIs.com.” [Online]. Available: <http://searchstorage.techtarget.com/definition/Cinder-OpenStack-Block-Storage>. [Accessed: 08-Jul-2015].
- [33] “Welcome to Keystone, the OpenStack Identity Service! — keystone 8.0.0.0b2.dev52 documentation.” [Online]. Available: <http://docs.openstack.org/developer/keystone/>. [Accessed: 08-Jul-2015].
- [34] “Keystone Architecture — keystone 8.0.0.0b2.dev52 documentation.” [Online]. Available: <http://docs.openstack.org/developer/keystone/architecture.html>. [Accessed: 08-Jul-2015].
- [35] “Welcome to Glance’s documentation! — glance 11.0.0.0b2.dev31 documentation.” [Online]. Available: <http://docs.openstack.org/developer/glance/>. [Accessed: 08-Jul-2015].
- [36] “Welcome to the Ceilometer developer documentation! — Ceilometer 5.0.0.0b2.dev52 documentation.” [Online]. Available: <http://docs.openstack.org/developer/ceilometer/>. [Accessed: 08-Jul-2015].
- [37] “Welcome to the Heat developer documentation! — heat 5.0.0.0b2.dev172 documentation.” [Online]. Available: <http://docs.openstack.org/developer/heat/>. [Accessed: 08-Jul-2015].
- [38] “Welcome to Trove’s developer documentation! — trove 4.0.0.0b2.dev25 documentation.” [Online]. Available: <http://docs.openstack.org/developer/trove/>. [Accessed: 08-Jul-2015].
- [39] “Getting Started — Sahara.” [Online]. Available: <http://docs.openstack.org/developer/sahara/userdoc/overview.html>. [Accessed: 08-Jul-2015].

- [40] “Logical architecture - OpenStack Cloud Administrator Guide - current.” [Online]. Available: <http://docs.openstack.org/admin-guide-cloud/content/logical-architecture.html>. [Accessed: 08-Jul-2015].
- [41] “Developer Guide — neutron 2015.2.0.dev535 documentation.” [Online]. Available: <http://docs.openstack.org/developer/neutron/devref/>. [Accessed: 02-Jun-2015].
- [42] “Chapter 3. Designing for Cloud Controllers and Cloud Management - OpenStack Operations Guide.” [Online]. Available: http://docs.openstack.org/openstack-ops/content/cloud_controller_design.html. [Accessed: 08-Jul-2015].
- [43] R. (Red H. Kukura and K. (Cisco) Mestrey, “ML2-Past-Present-and-Future.” .
- [44] “How to write a Neutron Plugin - if you really need to.” [Online]. Available: http://www.slideshare.net/salv_orlando/how-to-write-a-neutron-plugin-if-you-really-need-to?qid=2960a06f-4aa7-49d6-9a7b-65f3187667b2&v=qf1&b=&from_search=1. [Accessed: 02-Jun-2015].
- [45] “Neutron/ML2 - OpenStack.” [Online]. Available: <https://wiki.openstack.org/wiki/Neutron/ML2>. [Accessed: 01-Jul-2015].
- [46] “Developer’s Guide — OpenStack Project Infrastructure Manual 0.0.1.dev143 documentation.” [Online]. Available: <http://docs.openstack.org/infra/manual/developers.html>. [Accessed: 02-Jun-2015].
- [47] “Horizon: The OpenStack Dashboard Project — horizon 8.0.0.0b2.dev132 documentation.” [Online]. Available: <http://docs.openstack.org/developer/horizon/>. [Accessed: 06-Jul-2015].
- [48] D. Lapsley, “OpenStack Horizon : Controlling the Cloud using Django,” 2014.
- [49] “GroupBasedPolicy - OpenStack.” [Online]. Available: <https://wiki.openstack.org/wiki/GroupBasedPolicy>. [Accessed: 02-Jun-2015].
- [50] “Overview — group-based-policy documentation.” [Online]. Available: <http://group-based-policy.readthedocs.org/en/latest/usage.html#what-is-group-based-policy>. [Accessed: 02-Jun-2015].
- [51] “Usability 101: Introduction to Usability.” [Online]. Available: <http://www.nngroup.com/articles/usability-101-introduction-to-usability/>. [Accessed: 28-Jun-2015].
- [52] Pforzheim Univeristy/Display Lab, “Graphical User Interface: Evaluation,” 2015. [Online]. Available: http://eitidaten.fh-pforzheim.de/daten/mitarbeiter/blankenbach/vorlesungen/GUI/IT_GUI_Evaluation.pdf. [Accessed: 15-Jun-2015].