MASTER THESIS

TITLE: Deployment of NFV and SFC scenarios

MASTER DEGREE: Master's Degree in Telecommunications Engineering

AUTHOR: Pau Capdevila i Pujol

ADVISOR: David Rincón Rivera

DATE: February, 17th 2017

Títol: Desplegament d'escenaris NFV i SFC

Autor: Pau Capdevila i Pujol

Director: David Rincón Rivera

Data: 17 de febrer del 2017

Resum

Els serveis de telecomunicacions s'han dissenyat tradicionalment enllaçant dispositius de maquinari i definint els mecanismes per propiciar la interoperabilitat entre aquests. Els dispositius físics sovint són exclusius per a un sol servei i estan basats en tecnologia propietària. Per altra banda, el model d'estandardització vigent parteix de la definició de protocols rigorosos per tal d'assolir els alts nivells de fiabilitat que han caracteritzat l'entorn d'operador.

L'aprovisionament de nous serveis suposa dificultats a diversos nivells ja que són necessàries modificacions en la topologia de xarxa per tal d'intercalar-hi els dispositius requerits. Això comporta ineficiències en el desplegament i l'increment dels costos operatius. Per superar els actuals impediments cal flexibilitzar la instal·lació de noves funcions de xarxa i la seva inserció en la cadena d'elements que composen un servei.

El model vigent a les operadores tradicionals s'ha vist superat per l'oferta dels proveïdors de continguts que operen a traves d'Internet (Facebook, Netflix, etc.), amb cicles de producte i de desenvolupament molt dinàmics. Això ha suposat una competència i una sobrecàrrega per la infraestructura dels les operadors i els ha forçat a cercar noves tecnologies per irrompre de nou al mercat amb serveis flexibles i rendibles.

La virtualització de funcions de xarxa (NFV) i l'encadenament de funcions de servei (SFC) formen part del seguit d'iniciatives iniciades pels proveïdors de servei per tal de recuperar el lideratge. En el present projecte s'ha experimentat amb algunes de les tecnologies, ja disponibles, que estan cridades a vertebrar els nous paradigmes de xarxa (5G, IOT) i permetre nous serveis de valor afegit sobre infraestructures eficients.

Concretament s'han desplegat escenaris SFC amb Open Platform for NFV (OPNFV), projecte de la Linux Foundation. S'han demostrat algun dels casos d'ús de la tecnologia NFV amb aplicació a laboratoris docents. Tot i que l'actual implementació no assoleix un grau de fiabilitat apte per a entorns de producció, ofereix un entorn adequat pel desenvolupament de noves millores i l'avaluació funcional i de rendiment d'infraestructures de xarxa virtualitzades.

Title: Deployment of NFV and SFC scenarios

Author: Pau Capdevila i Pujol

Advisor: David Rincón Rivera

Date: February, 17th 2017

Abstract

Telecommunications services have been traditionally designed linking hardware devices and providing mechanisms so that they can interoperate. Those devices are usually specific to a single service and are based on proprietary technology. On the other hand, the current model works by defining standards and strict protocols to achieve high levels of quality and reliability which have defined the carrier-class provider environment.

Provisioning new services represent challenges at different levels because inserting the required devices involve changes in the network topology. This leads to slow deployment times and increased operational costs. To overcome the current burdens network function installation and insertion processes into the current service topology needs to be streamlined to allow greater flexibility.

The current service provider model has been disrupted by the over-the-top Internet content providers (Facebook, Netflix, etc.), with short product cycles and fast development pace of new services. The content provider irruption has meant a competition and stress over service providers' infrastructure and has forced telco companies to research new technologies to recover market share with flexible and revenue-generating services.

Network Function Virtualization (NFV) and Service Function Chaining (SFC) are some of the initiatives led by the Communication Service Providers to regain the lost leadership. This project focuses on experimenting with some of these already available new technologies, which are expected to be the foundation of the new network paradigms (5G, IOT) and support new value-added services over cost-efficient telecommunication infrastructures.

Specifically, SFC scenarios have been deployed with Open Platform for NFV (OPNFV), a Linux Foundation project. Some use cases of the NFV technology are demonstrated applied to teaching laboratories. Although the current implementation does not achieve a production degree of reliability, it provides a suitable environment for the development of new functional improvements and evaluation of the performance of virtualized network infrastructures.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Current communication service provider´s (CSP) networks contain a growing variety of proprietary hardware appliances to support legacy and new network services. Maintaining the life cycle of existent devices is a costly procurement, integration and deployment cycle with no revenue. Developing a new service means additional planning for new rack space, power and network expansion or change to accommodate any new required devices.

In the last years, large over-the-top (OTT) content providers like Facebook, Google or Netflix are competing with CSPs and their service offering. Moreover, OTT content providers pose a high traffic demand over the communication provider's network [1] and have competitive advantages like agile and efficient delivery of highly scalable and resilient services which force the traditional telco to transform or end up relegated to a secondary role.

Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) technologies are candidates to change the CSP traditional network service delivery model from a proprietary and tightly integrated stack into an open and decoupled model where applications are hosted on commodity computing and networking hardware. NFV proposes an innovative and cost-effective network infrastructure for faster service time-to-market.

NFV takes the idea of enterprise server virtualization to the CSP world. The NFV architecture implements network nodes such as Customer Premises Equipment (CPE) [2], usually delivered via purpose-built proprietary hardware, on Virtual Machines (VMs) or containers hosted either on bare-metal servers or a cloud computing infrastructure.

A subtle difference with classical IT virtualization is that a Virtualized Network Function (VNF) can use one or more VNF Components (VNFC) to compose a given function. VNFs act as building blocks that can then be chained together to create comprehensive communication services. Chains can be modified dynamically to leverage complex service topologies via VNF orchestration.

A Service Function Chain (SFC) is the concatenation of Network Functions (NF) to provide a given service. This has been accomplished in a rather static fashion until now. The introduction of NFV and SDN technologies ease the provision of network infrastructure and simplify the creation of service chains that can be deployed at a higher scale to be used by CSPs to provide feature-rich services.

The aim of this thesis is to explore the current state of the art in standards-based service chaining use cases for end-to-end service delivery within the scope of Network Functions Virtualization. The scope of the work is mainly focused on enterprise and telco scenarios as NFV technologies play a key role in the next generation of carrier-grade networks.

The project's working methodology has been based both on research and practical implementation. First, a state of the art overview on NFV and adjacent

technologies was conducted. During this phase unit proof of concepts (POCs) were run in sandbox environments for the most relevant open source implementations to assess the maturity of each project.

After that, a selection of projects implementing the core NFV building blocks was integrated into wider POC scenarios. Stress has been put in using standards-compliant open source tools which the industry uses to steer innovation. This approach is also open to academic contribution and porting the scenarios to the EETAC labs has been an achievement in this direction.

The document has been divided into the following chapters to better describe the project development phases:

1. Introduction to NFV and SFC
2. State of the art in open source NFV
3. Deployment and testing of NFV SFC Scenarios
4. Conclusions

Some sections have been added as annexes and supporting materials:

A. NFV management and orchestration
B. NFV use case examples
C. SFC encapsulation: NSH
D. State of the art in virtualized computing
E. OPNFV projects overview
F. Linux and OpenStack networking
G. Detailed scenario setup guide
H. Detailed scenario walkthrough

During the project, several NFV SFC scenarios have been tested successfully showcasing the basic features of the proposed technologies. The learning curve with the evaluated tools has been steeper than expected. But finally, failproof procedures have been carried out in a repeatable fashion thanks to small code improvements which can be contributed back to the community.

The available open source implementations of NFV are not yet production-ready, but provide a solid playground for feature development and functional infrastructure testing and benchmarking. There is currently a lack of available VNFs, which indicates that the available open source frameworks are useful reference implementations but still have to reach maturity.

NFV relies on commodity hardware but to realize its full potential last generation equipment is required to take advantage of the cutting-edge virtualization features. Several lab environment shortcomings and hardware limitations have put the project development into struggle limiting the thesis achievements. But the potential of the technology has been proved to a great extent.

Regarding the SFC technology, the available implementation has limitations currently solved with several workarounds. This reveals that there is still work pending to provide a solid reference implementation. This may also indicate a problem in terms of the contributor's balance of commitment between the open source and the internally derived implementations.

CHAPTER 1. INTRODUCTION TO NFV AND SFC

This chapter provides an introduction to the current communication service provider (CSP) architecture limitations, which are the main drivers for the NFV paradigm. It also includes a brief overview of the different fields in the NFV standardization: the ETSI specifications and the IETF standards. The information model standards based in the OASIS projects have been enclosed in the MANO-specific Annex A due to space constraints.

1.1. Limitations of the current model

Telecommunication carriers and vendors have been ahead in the market during the pre-Internet era. However, in recent years they have clearly lagged behind content providers. The big tech companies have taken a very successful vertical integration approach where, for example, Facebook built their network and developed its switches to accommodate better their traffic patterns [3].

One reason is the lack of a virtualization strategy in the Telecom industry equivalent to the one performed in the information technology (IT) industry. In recent years some signs of change have started showing with the early adoption of SDN (Software Defined Network) [4] technologies, but this is still not enough to cope with the speed of change that the big tech firms impose.

Based on [5], several factors have led to stagnation in the access provider landscape:

- Dependency on application-specific hardware and vendor lock-in. This also has side effects on the skills required to manage telecom-specific network equipment. It all adds up to the organization's technical debt.

- Increased energy consumption due to the expanding need for hardware. The inability to consolidate workloads and lack of power awareness of current devices add to the power budget.

- Long cycles of innovation due to the lack of test facilities similar to the production infrastructure. Test and integration are more difficult increasing development costs and time to market.

- Heavy investments in hardware-based functionalities are required to create new services or add features. Economies of scale are not achieved with the hardware-based model due to a variety of equipment.

- Need to overprovision and oversubscribe due to the static capacity of the network. Services cannot be scaled up or down on-demand nor can be scaled based on geographic demand.

- Reduced operational efficiency due to the diversity of the network infrastructure and management platforms. Lack of orchestration leads to manual installation and inability to modify the capacity on-demand.

1.2. Introduction to NFV

NFV (Network Functions Virtualization) defines how network services are abstracted using virtualized technologies which decouple them from the underlying hardware. The virtualization brings the opportunity to orchestrate services over commodity hardware without the operational burdens of architectures with network functions tied to physical appliances [6].

NFV is not just about virtualizing network appliances. It also involves breaking down all network functions to its elemental building blocks to later compose and orchestrate services only with the required components. The actual potential of NFV can only be realized with Service Function Chaining (SFC). Without SFC and orchestration, NFV alone would only add complexity.

NFV could be realized relying on the protocols already in use in many multi-tenant networks[1]. However, managing multi-tenancy in current networks for customer isolation and resource partitioning is difficult and time-consuming due to its coupling with the network topology. Adding SDN can enhance and simplify operations, administration and maintenance and leverage end-to-end dynamic multi-tenancy.

NFV and SDN can be deployed independently, but implemented together they play a complimentary role as enabling technologies for future networks. NFV can provide the infrastructure for SDN. The other way around SDN can provide the data-center (DC) and wide area network (WAN) network layers for NFV with proper orchestration coordination as stated in [7]. Therefore, NFV and SDN can be thought as the sides of the same coin in some scenarios. Also, NFV aligns with the SDN goal to employ commercial off-the-shelf (COTS) hardware.

SDN plays a key role in the NFV framework as virtual network functions (VNFs) can be spread over data-centers, network nodes or end-user premises. VNFs could also belong to different domains. An SDN-decoupled control plane leverages programmability, elasticity and openness between the NFV virtualization infrastructure (NFVI) network and the WAN or the data-center interconnect (DCI) [8]. In this context, one of the most relevant SDN solutions are Network Virtualization Overlays (NVOs).

Figure 1.1 shows the comparison of the hardware appliance model and the NFV approach.

[1] Multi-tenant networks are networks where different operators share the same infratructure but are logically isolated between them. This has many advantaces, specially from the economic point of view (economies of scale), and is a common deployment model in cloud computing data-centers.

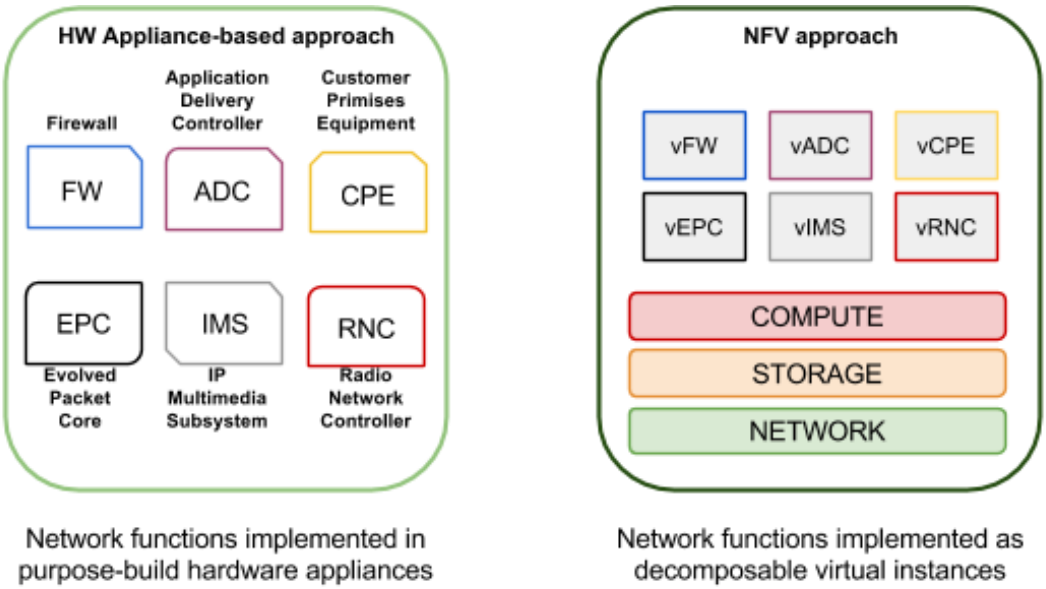

Figure 1.1: Comparison of classical and NFV model

An overlay network is a network built on top of another network (the underlay). NVOs are part of a novel paradigm were VMs and bare-metal servers are integrated into public, private or hybrid clouds where the network overlay provides the connectivity. NFV can take advantage of overlay networks to allow VMs to perform a network service function. This VMs, now VNFs, are usually transit entities, not end-points. Examples of such network functions could be stateful firewalling, application delivery controllers or deep packet inspection.

As stated in [5] NFV promises the following improvements:

- Cost efficiency: Reduce CAPEX and OPEX through lower equipment costs and improved ROI from new services.
- Complexity reduction: integrating and deploying new software appliances in a network.
- Service agility: reduced time-to-market for new network services.
- Deployment automation and operations simplification: Greater flexibility to scale up, scale down or evolve services.
- Resource optimization: Maximize server usage and minimize energy consumption and rack space.
- Openness: Opportunity of a virtual appliance market may generate pure software suppliers
- Opportunities to trial and deploy new innovative services through network abstractions

But poses the following challenges:

- Performance: network stability and service levels must be granted without degradation during appliance load and relocation.
- Reliability and resiliency: ensure proper service survival to hardware and software failures.

- Interoperability and portability: appliances should be portable between different hardware vendors and different hypervisors.
- Migration and compatibility with legacy networks: coexistence with legacy hardware-based network platforms need to be assured while enabling an efficient migration path to a fully virtualized network.
- Standardized management interfaces: should allow re-using existing OSS/BSS (Operations Support System/Business Support System).
- Automation at scale: virtualized network platforms should be simpler to operate than existing networks.
- Security: ensuring security both from attack or misconfiguration.

NFV was proposed by a group of network service providers at the SDN and OpenFlow World Congress in 2012. Later, ETSI (European Telecommunications Standards Institute) [9] was decided to be the summit of the Industry Specification Group (ISG) for NFV. The ETSI NFV ISG [10] defined the architecture that is widely accepted and used in this thesis.

ETSI also defined a set of first high-level use cases [11], spanning from the virtualization of the CPE or the network core (IMS, EPC, and RAN) to NFV clouds. ETSI NFV proves the feasibility of the use-cases through proof of concept trials. The other Standards Developing Organizations (SDO) and Open Source Foundations are expected to detail each use case further.

The Virtual Customer Premises Equipment (vCPE) is an NFV use case which can help consolidate the understanding of the NFV definition subtleties. vCPE was the top use case within the original ETSI list as per [12] due to its economic benefits. The vendors have taken different approaches which need disambiguation. For example, Annex B compares how Juniper Cloud CPE and vCPE implement a different virtualization approach [13].

1.2.1. ETSI NFV Architectural Framework

After the first call for action white paper [5], the NFV Industry Specification Group (ISG) released the first set of white papers in 2013 [14] [15] with high-level use cases, proposed terminology for virtualisation and, most important, the NFV architectural framework. It also described management and orchestration (MANO) functions which should be further developed in the scope of the MANO working group [16].

The building blocks of the NFV architecture are:

- **Network Functions Virtualisation Infrastructure** (NFVI) provides the hardware resources required to run the VNFs. It includes compute, storage, networking hardware and a hardware virtualization layer.
- **Virtualized Network Functions** (VNFs) are the software-based network elements that are executed on the NFVI.
- **Management and Orchestration** (MANO) takes care of the life-cycle administration of both the NFVI through the VIM and the VNFs themselves.

It interacts with the OSS/BSS allowing integration with the legacy network management tools. It is divided into three parts:

- **NFV Orchestrator** (NFVO) contains a catalog of network services (NS) and VNF packages and interfaces with the VIM to accomplish NS lifecycle management
- **The VNF manager** (VNFM) takes care of the instantiation, scaling, updating, upgrading and termination of VNFs.
- **The Virtualized Infrastructure Manager** (VIM) provides interfaces to manage and control the compute, storage and networking resources.

Figure 1.2 provides an overview of the NFV architecture:

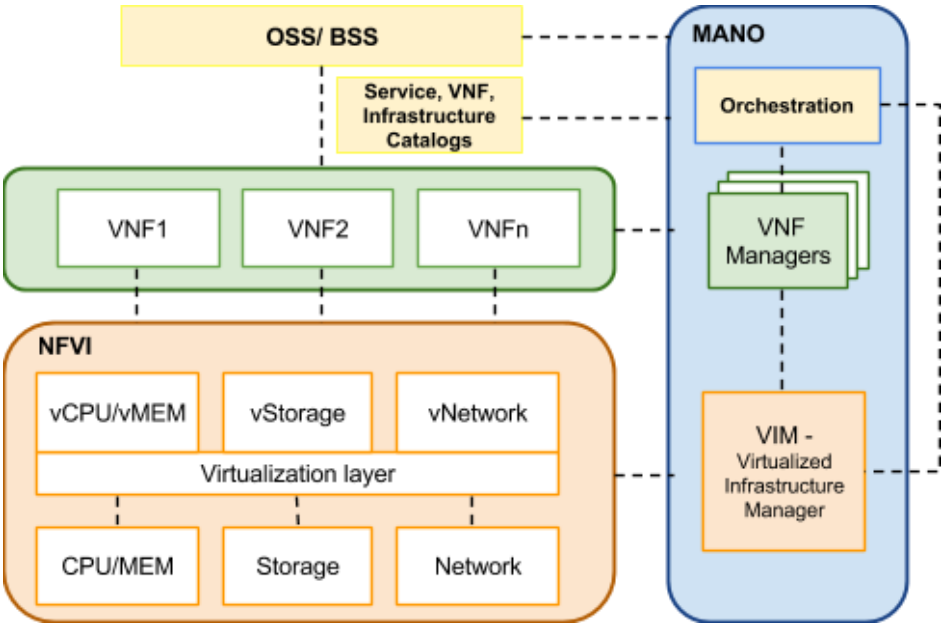

Figure 1.2: NFV ETSI architecture [8]

1.3. Integration of SDN and NFV

The modern idea behind Software-Defined Networking (SDN) was born due to the dynamic network requirements of cloud data-centers and carrier environments. SDN is a term coined by the Open Networking Foundation (ONF) [17]. The SDN definition [18] has changed over time, and it has been subject to many interpretations, but after some years of its inception it can be understood as a new network paradigm providing:

- Centralization of the control plane management functions using network controller functions decoupled from the traffic forwarding processing.
- Automation of the network lifecycle management improving both user experience and reducing operating costs.

- Abstraction of the management interfaces using APIs (Application Programming Interface) which enable direct network interaction.

The ONF aim was to provide open and standards-based implementations to this model, and the most significant achievement is the OpenFlow switch specification [19]. OpenFlow defines a communication interface between the controller and forwarding layers in the SDN architecture. In the beginnings of the ONF OpenFlow was synonym with SDN but slowly OpenFlow protocol has been losing momentum as stated in [20].

This is why ONF has started the absorption of the Open Networking Lab (ON.Lab), home of the Open Network Operating System (ONOS) [21] and Central Office Re-architected as a Datacenter (CORD) [22] projects. By merging two standards development and open source software organizations ONF is trying to regain technical relevance with OpenFlow.

Figure 1.3 shows the integration of NFV in the three SDN architecture layers:

- **Application layer** contains applications consuming SDN services through the control layer interfacing through Northbound APIs.
- **Control layer** contains the network intelligence and manages the network forwarding behavior through an open interface such as OpenFlow-based on the applications' requirements.
- **Infrastructure layer** is formed by the network elements (NE) that provide flow switching and other data-plane functions.

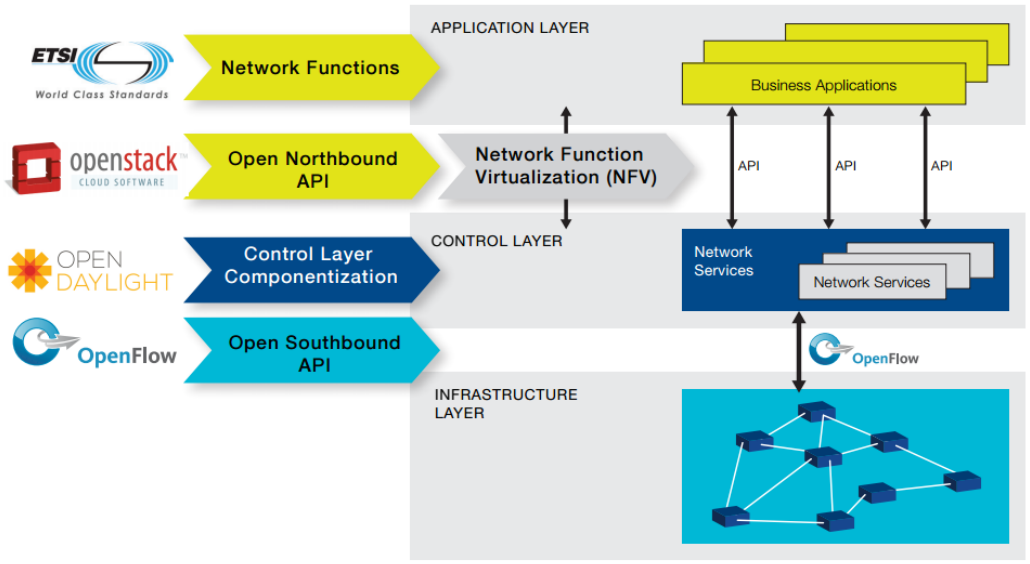

Figure 1.3: Integration of NFV and SDN [23]

Enhancing NFV with SDN networking can not only realize the common goals of each technology but multiply the benefits of both technologies. SDN can support NFV infrastructures providing scalable and on-demand networking according to the changing VNF connectivity requirements for both virtual and physical

networking infrastructures, as the OpenFlow switch specification applies both to physical and virtual switches.

1.4. Introduction to SFC

For many years, service chains have been built wiring physical appliances one after the other so that the traffic crosses them. For example, in the telco central office architecture, the several functions are linked with physical links so that traffic enters through the Optical Line Termination (OLT) and traverses an aggregation layer to traverse the Broadband Network Gateway (BNG) finally as depicted in Figure 1.4:

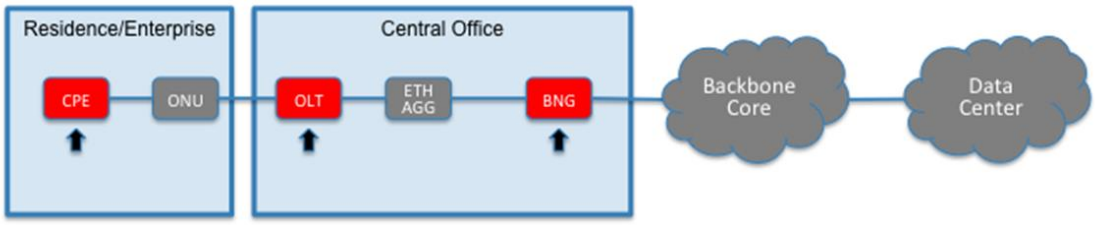

Figure 1.4: Legacy telco central office architecture [22]

This basic idea evolved in layer2 or layer3 network topologies in the same way as the traffic still needed to traverse service functions in their forwarding path. Instead of physical wiring, network functions were linked either via bridges or routers. This tight coupling between network topology and service data path poses several limitations as per [24]:

- Placement and selection of service functions.
- Setup and modification configuration complexity.
- Transport dependency.
- Lack of scalability.
- No multi-domain capability.
- Lack of end-to-end view and Service Chain OA&M mechanisms.

Several technologies have been used over time. However, they only solve some of these challenges. Some examples from [25]:

- VLAN stitching.[2]
- BGP based Routed Service Chain (BGP or SDN-based).
- VXLAN (Multi-domain) with SDN Control Plane.
- Segment Routing.
- OpenFlow-based Service Chaining.

In the Service Function Chaining (SFC) approach a classifier and mapping functions are introduced to steer matching traffic towards the service chain. This

[2] Interleaving a network function such as a transparent firewall in the middle of a layer 2 domain.

way service functions can be placed anywhere on the network and can be seen as a bump-in-the-wire. The classifying policy can be a simple match on VLAN, VRF, a flow rule match or another network-specific identifier.

For example, in the Long-Term Evolution (LTE) Evolved Packet System (EPS)[3] network a traffic flow would be classified in the SFC domain entry as illustrated in Figure 1.5. Once classified, the network would take care of dynamically route the traffic through the required functions via a separated control plane managing the service chains. The classifier may be integrated into existing devices, such as the PGW (Packet Data Network Gateway). The classification parameter could be the Access Point Name (APN).

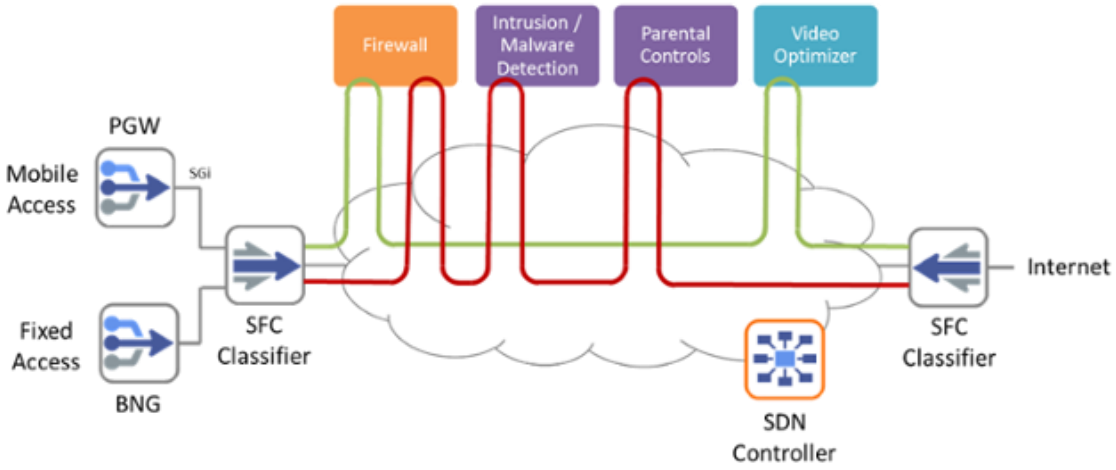

Figure 1.5: Classification in the SFC domain entry in the EPS network

The new SFC approach offers:

- Different policies applied to various types of traffic.
- The individual functions can be modified independently.
- A foundation for cross-domain use cases. The selected service functions can meet domain specific end-user requirements.

As ETSI is not a standards body organization, the IETF was designed to develop a standard covering automated end-to-end service deployments. This led to the SFC Working Group [26]. One point to stress is that the SFC Architecture also includes physical network functions, which will continue to be prevalent in the initial stages of NFV adoption. This is why the SFC nomenclature is used instead of the VNF Forwarding Graph (VNFFG) one.

[3] EPS is the network core in the LTE mobile communication system. The LTE is a 4G wireless broadband technology developed by the 3GPP partnership. Do not confuse with Evolved Packet Core (EPC), which is the framework for providing unified voice and data in 4G networks.

1.4.1. IETF SFC Architecture

The SFC WG has produced several informational RFCs and drafts, starting with the problem statement early in 2015 [24] and the SFC architecture [27]. It also defined a network transport agnostic mechanism to steer traffic through SFs based on a generic SFC encapsulation: the Network Service Header [28] for which there are some commercial implementations, such as Cisco IOS XE [29]. The IETF SFC architecture helps to create composite network services built upon an ordered set of SFs that flows must traverse if they match classification criteria. Each SF is mapped to a unique identifier within the SFC-enabled domain. It describes SF deployment mechanisms that enable topological independent dynamic ordering of SFs and metadata exchange between the different core components in the architecture, which are:

- **Service Functions** (SFs) are resources in an SFC domain that can be invoked as part of a composite service.
- **Classifiers** are logical entities delimiting the SFC domain. Any traffic entering the domain will go through an initial classification. Based on the classification decision, the traffic is encapsulated and mapped to a new or existing chain, and metadata is embedded. Classifiers are commonly implemented along with an SF or SFF.
- **Service Function Forwarders** (SFFs) take care of traffic forwarding between connected SFs based on the SFC encapsulation. They also can deliver traffic to a classifier or another SFF.
- **SFC proxies** remove and insert SFC encapsulation on behalf of an SFC-unaware SF. SFC proxies are logical elements.

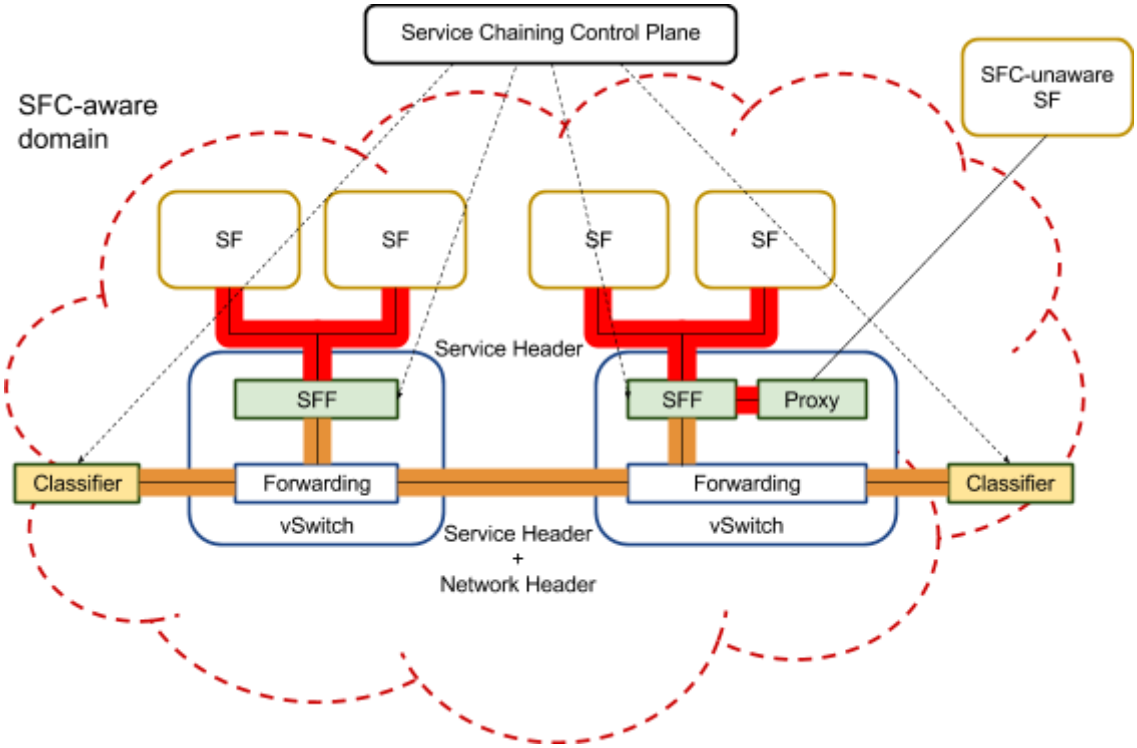

Figure 1.6: SFC IETF Architecture [27]

In the SFC architecture, packets are classified when they enter the SFC-enabled domain as shown in Figure 1.6. Then, they are mapped to service function chains and forwarded through the service function path for processing in the required service functions (SF). A service overlay is built between SFC-aware elements in the architecture thanks to the SFC encapsulation: the Network Services Header (NSH). A more in-depth explanation of SFC and the service header encapsulation can be found in Annex C.

CHAPTER 2. STATE OF THE ART IN OPEN SOURCE NFV

Some aspects of the NFV technology are actively being deployed, but others are still in an early phase of maturity. This chapter explores NFV's yet short lifespan and enumerates the key open source projects that could form a complete NFV stack. It is quite difficult to gather all state of the art in NFV in a single chapter as the recent years' evolution has been like the Big Bang expansion, as Figure 2.1 tries to resemble:

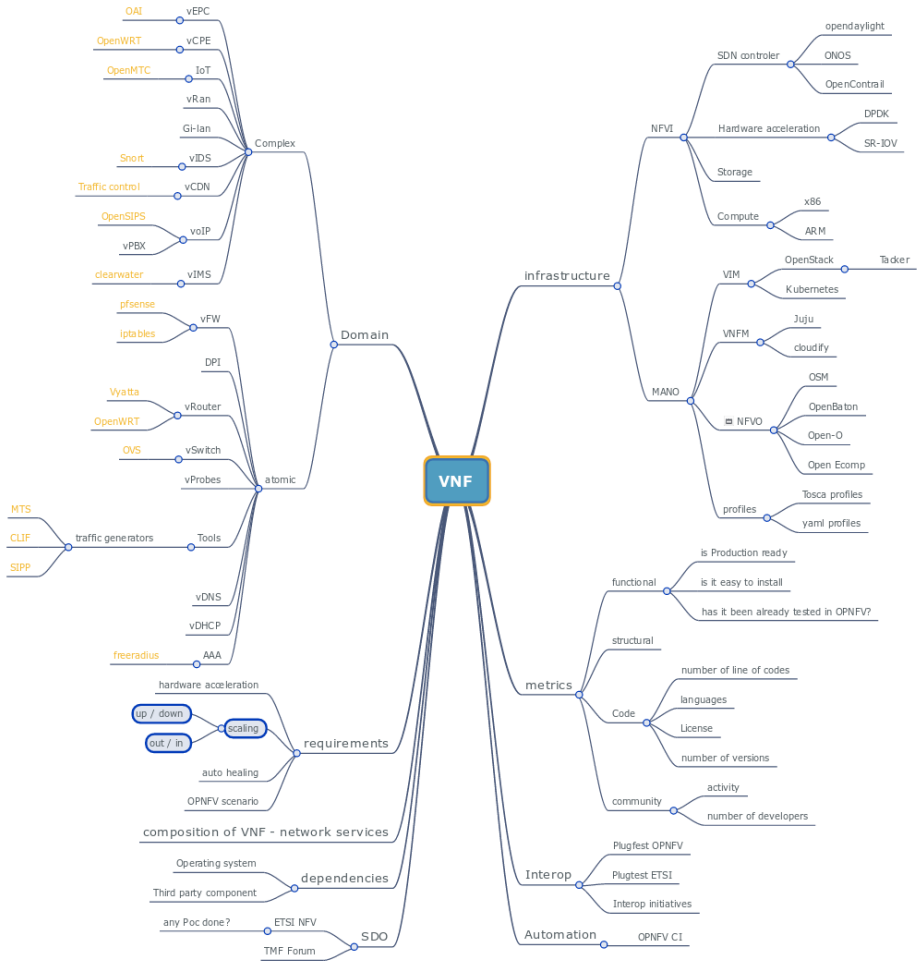

Figure 2.1: NFV mindmap by Morgan Ricchome [30]

In the mindmap, the VNF is at the center as it is the element that in the end delivers the service. The first level ramifications, in a clock-wise direction, expand on the supporting infrastructure, the technology capabilities, the current interoperability efforts, the involved standards developing organizations (SDOs), the requirements for network service composition and, finally, examples of VNFs, from the simple cases to more comprehensive ones.

2.1. NFV Historic Phases

2017 will mark the 5th anniversary of NFV. Talking about history for a 5-year span could sound bombastic but, given the speed of events, the ETSI NFV ISG releases can well be used as a timeline reference to developing this chapter content which, otherwise, could become quite disperse. In Figure 2.2 the timeline until mid-2016 is shown.

The focus of NFV Phase 1 (2012-2014) [31] was to establish the architectural framework. Phase 2 (2015-2016) focus was to promote adoption and interoperability. NFV Phase 3 (beyond 2016) aim is to foster an open ecosystem while NFV is actively deployed but, in theory, this would fall out of the ISG Charter, which was expected to end in late 2016, although it has been extended.

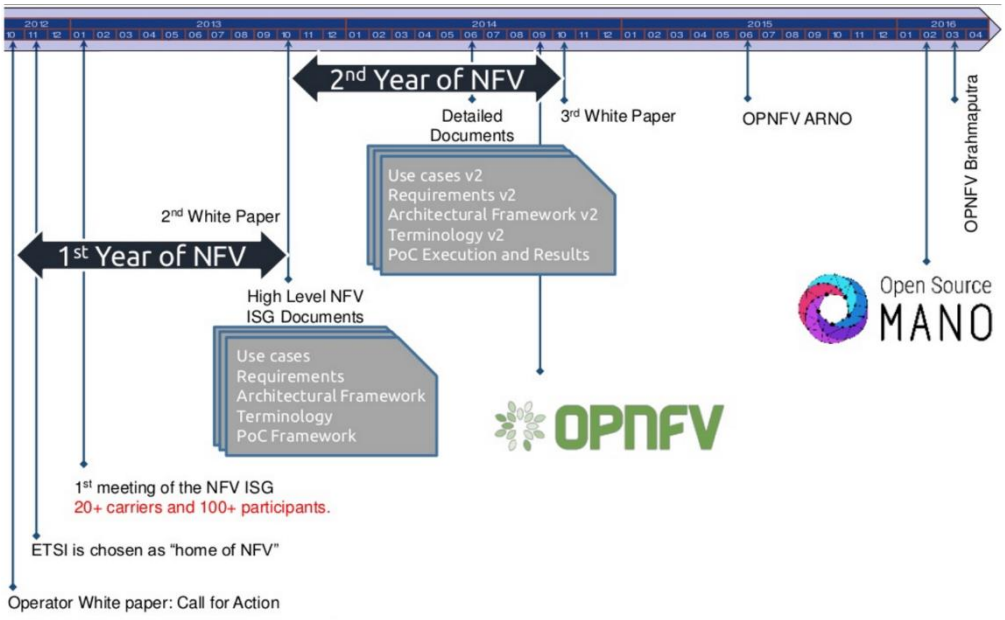

Figure 2.2: NFV timeline based on ETSI releases [32]

As stated in [6], the slowness and the political nature of ETSI has led to a migration of cooperation efforts towards OPNFV. For example, Phase 1 ended in a guidance rather than a full specification. While for Phase 2 there is still ongoing work on the interoperability field. For instance, at the time of writing (February 2017) an ETSI/OPNFV "PlugTest" took place in Madrid to align Industry and Open Source initiative against the defined standards [33].

Some NFV players endorse an open source approach to produce reference implementations of standards or by reaching de-facto standards directly. The use of Open Source Software (OSS) methodologies aims for fast-paced innovation and interoperability. This chapter reviews the open source ecosystem related to the NFV architecture. A review of MANO implementations can be found in Annex A.

2.2. An open source stack for NFV

One of the open source principles is to re-use existing tools suitable for other projects. Many open source projects can be integrated to deliver a whole NFV platform. It is no wonder that the open source community is especially active in the area of network virtualization. The next sections categorize some the related tools to implement service function chaining scenarios over an NFV infrastructure.

2.2.1 NFV Infrastructure

OpenStack [34] is a set of software projects intended for building and managing cloud computing platforms. It can control pools of computing, storage, and networking resources that can form an NFV Infrastructure. It has received many contributions from the NFV community to serve as the NFV Virtual Infrastructure Manager (VIM) component. It provides SDN integration with multiple controllers via the Neutron ML2 plugin [35].

2.2.2 Network control plane

OpenDaylight (ODL) [36] is a multi-protocol SDN controller built for deployments on heterogeneous multi-vendor networks. It provides a model-driven service abstraction layer that allows easy integration between northbound interfaces and southbound protocols. ODL SFC project [37] provides a mature implementation of the SFC architecture.

Open Network Operating System (ONOS) [21] is a carrier-grade SDN operating system. While ODL is based on a microservices architecture, ONOS follows an application-based model[4]. SFC implementation comes as a bundle in the VTN ONOS application. XOS[38] is the service orchestration layer that turns OpenStack Neutron into an ONOS network control application.

OpenContrail [39] is an extensible platform for SDN. Its main components are the controller and the vRouter. The controller provides a logically centralized control and management plane and orchestrates the vRouters, conceptually similar to the OVS but in the Contrail architecture. OpenContrail implements SFC via a high-level policy language to interconnect virtual networks through additional routing instances which steer the traffic through the overlay (L3VPN or EVPN).

2.2.3 Network data plane

Open vSwitch (OVS) [40] is a multilayer virtual switch designed to enable network automation. It can work both as a hypervisor switch or as the control plane for hardware open switches. For best performance, a Linux kernel module is

[4] ONOS is built with application policy-based directives called intents. These intents are translated to OpenFlow rules and later programmed onto the switches.

provided. It has a rich feature set, including many features in the SDN infrastructure layer. Although NSH is not officially supported, patches that provide NSH support exist for both Data-Path Development Kit (DPDK)[5] and non-DPDK data-paths.

The FD.io VPP (Fast data Input/Output Vector Packet Processing) [41] is an extensible switching/routing packet-processing platform that can run on commodity CPUs. It is based on Cisco's Vector Packet Processing [42], a modular technology built on a packet processing graph. VPP allows inserting new graph nodes without changes in the kernel. The Fd.io SFC project [43] supports NSH-based packet forwarding to allow high-performance SFC applications.

2.2.4 NFV Integrated platforms

OPNFV (Open Platform for NFV) [44] is a carrier-grade platform aimed to integrate upstream open source projects to accelerate the introduction of new NFV-based services. Several projects integrate the different components in an NFV architecture in several scenarios and a use case-based testing framework is provided to allow functional and CI testing. Several scenarios implement service chaining, including standards-based SFC.

Sonata [45] is an NFV platform part of the 5G-PPP initiative [46]. Sonata is a service programming and orchestration framework. The core of Sonata is a service platform providing a MANO implementation and catalogs that store artifacts that can be used by the Sonata system. Sonata also consists of a software development kit (SDK) that supports a programming model and a development tool-chain. The programming model focuses on service chains. Sonata is open source but follows a particular release cycle and governance model. At the moment only one version has been released.

ECOMP (Enhanced Control, Orchestration, Management & Policy) [47] is the AT&T core NFV platform. It expands the ETSI architecture focusing on Controllers and Policy. Several software subsystems cover two major architectural frameworks: a design environment to program the platform and an execution environment run the logic using policy-driven automation. Unfortunately, there is no reference implementation open for testing at the moment of writing as it will be open sourced during 2017.

After reviewing the available options, OPNFV has been selected to deploy and test NFV scenarios as it employs the most widely used tools and is the project currently covering most of the pieces in the standards-based NFV SFC architecture. The following sections describe OPNFV and its main components, which for the Colorado release are:

- OpenStack Mitaka with SFC-enabled Tacker.
- OpenDaylight Boron with SFC.
- Open vSwitch 2.5.90 with NSH patches [48].

[5] Data Plane Development Kit (DPDK) is an open-source library toolkit for fast packet software processing that supports the major processor architectures.

2.3. OpenStack

OpenStack is a software framework enabling cloud deployment and management suitable for IaaS (Infrastructure as a Service) service models. The first release only included compute and storage. Release after release it has grown to provide additional services in the software stack, such as networking, monitoring, authentication, orchestration and web user interface, amongst others. The current as of February 2017 is the 14th release, codenamed Newton [49].

The OpenStack architecture is built upon RESTful modular services. End users can interact either through the APIs or the provided CLIs and dashboards. Each 6-month release includes an updated set of core services and other accessory services under what is called the project “big tent” to complement the feature offering as shown in Figure 2.3. The core services provide:

- **Nova**: compute services such as scheduling and instantiation of VMs.
- **Neutron**: inter-networking between OpenStack components.
- **Cinder**: persistent block storage and volumes management for VMs.
- **Glance**: VM disk image and metadata storage.
- **Swift**: object (unstructured data) storage and replication.
- **Keystone**: API authentication and authorization.

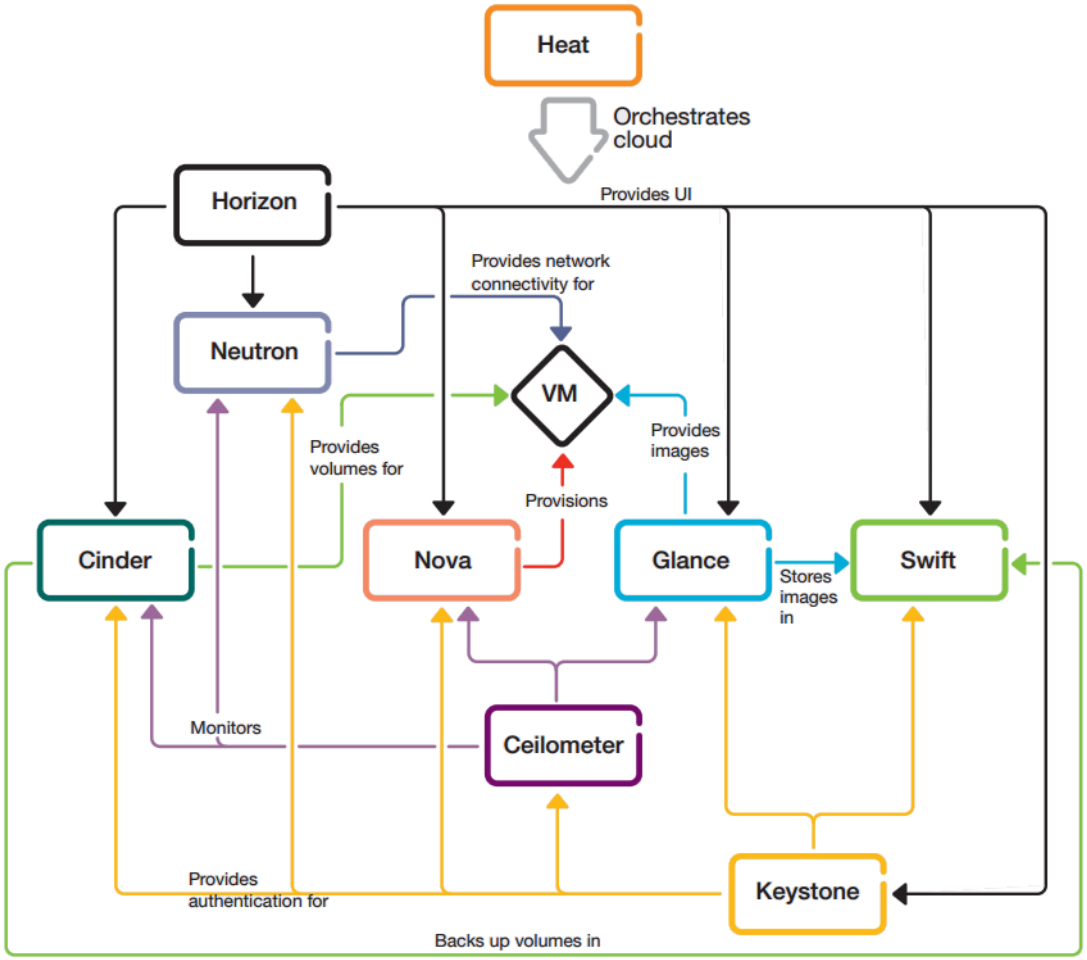

Figure 2.3: Simplified OpenStack conceptual architecture [50]

Other big tent relevant projects are:

- **Horizon**: a web portal providing a user interface for core service features.
- **Heat**: a service that provides application stack orchestration.
- **Ceilometer**: which collects event and metering data from other services.

2.3.1 OpenStack for NFV

OpenStack targeted cloud computing infrastructure as a service (IaaS) use cases initially. Over the time it is evolving to support NFV features like telemetry, orchestration and advanced network services. Before that, OpenStack already provided some of the features required in an NFV environment to render communication services over an IaaS infrastructure as shown in Figure 3.2.

OpenStack follows an open government model, and recently several CSPs have joined the Foundation's board of directors [51] to focus on NFV related requirements [52]. To prepare OpenStack for NFV, gaps in some projects must be addressed or new projects initiated.

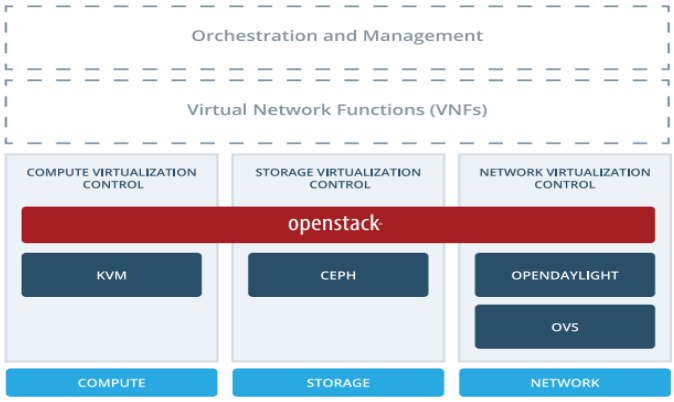

Figure 2.4: Alignment of OpenStack in NFV [52]

Table 2.1 lists some of the NFV requirements and the OpenStack projects that need improvements to fulfill them:

Telco Requirement	Gap	Projects
Distributed infrastructure	Geographically close workloads with real-time response and low latency	Heat [53]
Networking	Reliable, segmented and secure on-demand communication	Neutron [54]
Automated lifecycle management	Automatic service deployment, monitoring, scaling, healing and upgrading	Tacker [55]
NFV Infrastructure operations	Operation of distributed NFV infrastructure	Kingbird [56], Tricircle [57]
High-performance dataplane	High packet processing and input/output with current hardware (e.g. ASICS)	OVS [40] Fd.io [41]

Table 2.1: OpenStack NFV gap analysis based on [58]

2.3.2 Nova

Nova is the project that provides the compute service within OpenStack. It provides VM auto-provisioning on server hardware based on several constraints. The resources allocated to the VM are classified in flavours that define required parameters such as virtual CPUs, memory and storage space. The Nova scheduler algorithm selects the given compute node where to instantiate the VM [59].

The main components of Nova are:

- The Nova API (Application Programming Interface), a RESTful HTTP service implementing the Compute API.
- An AMPQ (Advanced Message Queuing Protocol) queue for RPC communications.
- The Nova scheduler to handle hypervisor selection based on scheduler filters.

Some recent improvements have been brought to Nova to better comply with NFV requirements, like Enhanced Platform Awareness (EPA). As further developed in Annex D, EPA enables the Nova scheduler to match a flavour taking into account finer specific hardware features:

- Encryption Acceleration
- Extended Vector Instructions
- Hardware Transcoding
- PCIe GPU Accelerator

Nova offers an API agnostic to the underlying hardware stack, which can manage on-demand compute resources. Depending on the deployment, those compute resources might be physical servers, Virtual Machines and even containers. Annex D also ellaborates on VNF packaging options. Several projects leverage containers in OpenStack:

- Kolla runs OpenStack itself on Docker containers, so it is out of scope in regards to the container use as a VNF packaging format.
- Murano offers an application catalog that allows application containers such as Kubernetes (K8s) to be installed on OpenStack.
- Magnum offers multi-tenant Container-as-a-Service APIs for several container architectures, as shown in Figure 2.5. Namely, Docker Swarm, Kubernetes and Mesos [6].

[6] Docker is a containerization platform. Docker swarm is a tool that provides clustering capabilities to a group of Docker engines. Kubernetes is a tool for automating the deployment and life-cycle management of containerized applications. Finally, Mesos is a distributed systems kernel for which a containerizer module brings support for different container formats (mesos and docker).

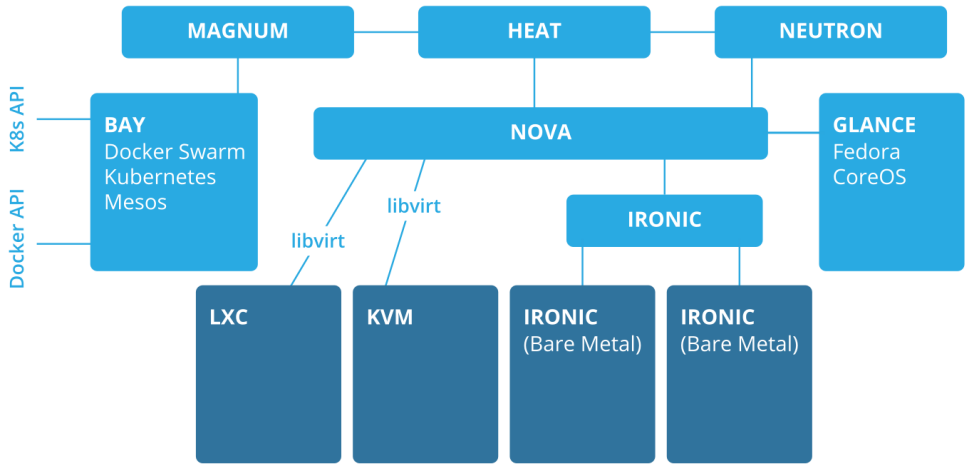

Figure 2.5: OpenStack Magnum Container-as-a-Service architecture [60]

2.3.3 Neutron

Neutron offers the networking service between OpenStack modules. It was originally included within the compute service but due to its organic growth was moved to a separate project, first called Quantum and now Neutron. The API provides basic connectivity and addressing services. Additionally, it supports on-demand services like routing, NAT, load balancing, firewalling and VPN-as-a-Service.

The main components of Neutron are:

- The Neutron server, which exposes an API to support Layer2 and basic Layer3 networking, DHCP and IPAM (IP address management).
- Plug-in and Plug-in agents that enable Neutron to interact with diverse network technologies, including routers, switches and SDN controllers.
- A message queue used in the ML2 (Multi-Layer 2) plug-in for RPC between the neutron server neutron and agents that run on each compute node hypervisor.

Neutron distinguishes between two kinds of networks:

- Tenant networks are used for connectivity within projects. By default, they are isolated and not shared between projects.
- Provider networks represent existing virtual or physical networks in the outside Neutron which can map to multiple tenants.

Neutron included initially monolithic plug-ins which were mutually exclusive. For instance, the Linux bridge and Open vSwitch plugin. Later a modular plug-in architecture was introduced to embrace the most common Layer 2 networking protocols and technologies found in modern data-centres. The ML2 (Modular Layer 2) framework offers two kinds of drivers that can be used simultaneously:

- Type drivers define the several kinds of tenant networks:

    - Local: existing on a single host.
    - Flat: implementing a regular broadcast domain.
    - VLAN: using VLAN tagging for segmentation.
    - GRE: using Generic Routing Encapsulation overlay.
    - VXLAN: using Virtual Extensible LAN overlay.

- Mechanism drivers implement the different types of networks on various open source switching solutions:
    - LinuxBridge, the plain old Linux Kernel bridge implementation.
    - Open vSwitch, a multilayer virtual switch with plenty of features.
    - MacVTap virtualizes the NIC (Network Interface Card) in vNICs.
    - SR-IOV (Single Root I/O Virtualization) vNIC at the PCIe level.
    - L2 Population limits broadcast in overlays networks.

  and also on proprietary network gear:
    - Arista EOS switches for VLAN and VXLAN provider networks.
    - Specific Cisco Nexus models support either VLAN or VXLAN.
    - Brocade NOS switches in VCS (Virtual Cluster Switching) mode.
    - Hyper-V Agent for Hyper-V compute nodes.
    - ALE Omniswitch switch range from Alcatel-Lucent Enterprise.
    - Lenovo physical switches for VLAN networks/
    - Tail-F NCS (Network Control System), which is a Multi-Vendor Network Orchestration suite.

2.3.4 Heat

Heat is a service to orchestrate cloud applications or composite services using templates. Templates are human readable formatted text files able to describe Infrastructure as Code. Heat can provision all the OpenStack Infrastructure defined in a template and can be extended to support other resources using plugins. Besides of infrastructure components and its relations, Heat includes hooks for configuration management tools like Ansible [61] or Puppet [62].

The main components of Heat are:

- Heat APIs. An OpenStack native API is available alongside with an Amazon Web Services (AWS) CloudFormation API.
- Heat engine receives RPC calls from the API components and performs the orchestration tasks.

Heat supports several kinds of templates, one for each supported API:

- HOT (Heat Orchestrator Templates) are the native template format and are usually defined in YAML.
- CFN are the CloudFormation compatible template type and are written in JSON (JavaScript Object Notation).

Heat templates have 3 sections:

- Template parameters are mainly resource IDs defined by the user.
- Resources are the objects that Heat will provision. Once resources are created, they are called stacks.
- The output is the information returned to the user/API invoking the template.

2.3.5 Tacker

The Tacker project [55] implements the VNF Manager and NFV Orchestrator functions in the ETSI MANO stack as shown in Figure 3.4. It takes care of deploying VNFs and orchestrating Network Services over an OpenStack platform. In the OpenStack Newton release, it will support SFC natively [63]. At the moment the SFC integration is implemented in a customized Tacker version for OPNFV. A more in-depth case study about Tacker is included in Annex A.

2.3.6 Fuel

Fuel [64] is a modular and extensible OpenStack deployment and management tool. It focuses on automating the installation and validation of OpenStack clusters and third-party plugins. It integrates several components. The core where the main Fuel logic is implemented is the Nailgun service, which offers a REST API, a Web UI and a CLI. Then there is Astute which interacts with the node provisioning and deployment tools, mainly Cobbler [65] and Puppet/Mcollective [66].

The Fuel architecture distinguishes between:

- The Fuel master, a server that performs preboot execution environment (PXE) booting, IP assignment, OS provisioning and initial configuration of slave nodes.
- Fuel Slaves, servers managed by the Fuel Master. Fuel slaves can become a variety of OpenStack roles based on Fuel node configuration.

2.4. OpenDaylight

OpenDaylight (ODL) [36] is a modular multi-protocol SDN platform. The current as of January 2017, is the 5th release of ODL, codenamed Boron (as the 5th element in the periodic table). It brings several enhancements to features that apply to NFV, being one of the most notable a new project, Genius (Generic Network Services). Genius improves the coexistence between apps that can program the same forwarding tables. This optimizes ODL application integration for complex implementations.

The OpenDaylight architecture includes:

- The controller platform, which contains services and applications which implement controller logic independently of the underlying network technologies below.
- Southbound interfaces and protocol plugins. OpenDaylight offers a Model-Driven Service Abstraction Layer (MD-SAL) which allows applications to control a broad range of underlying networking technologies.
- On top of the components above, there are the several north-bound APIs, the AAA (Authentication, Authorization and Accounting) layer and the DLUX Interface framework.

Figure 2.6 shows the OpenDaylight architecture for the Boron release:

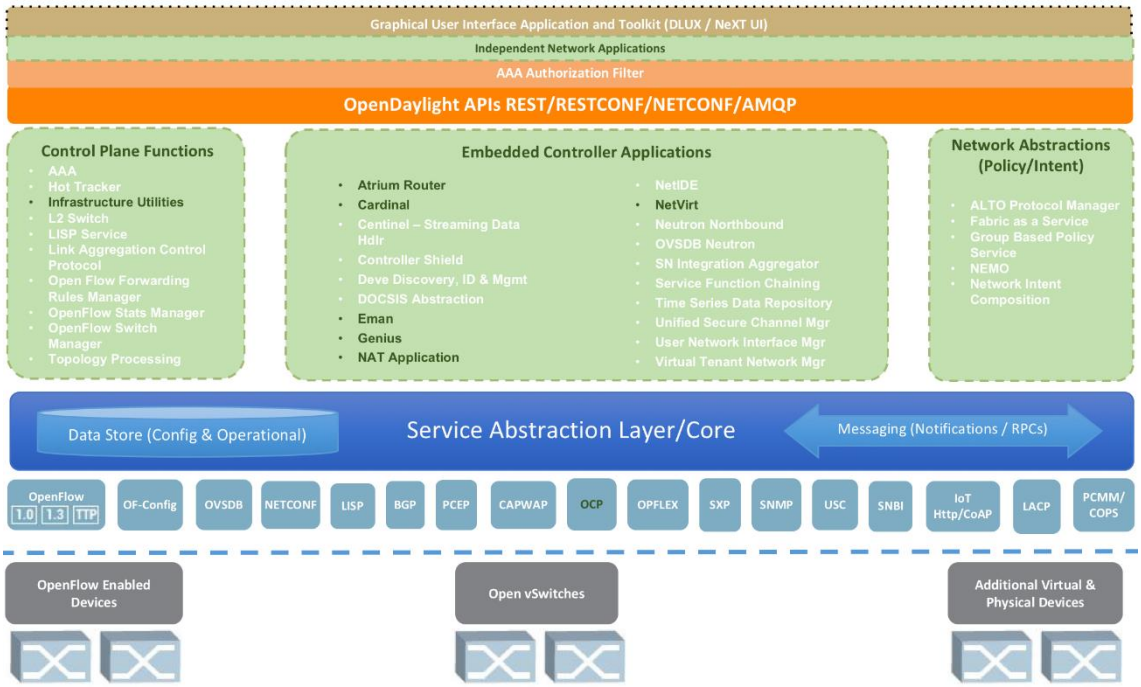

Figure 2.6: OpenDaylight high level architecture [67]

Regarding OpenStack integration, OVSDB (Open vSwitch Database Management Protocol) features have been unified to achieve parity with the Neutron Reference Implementation in OpenStack better. Several applications in ODL cooperate with OpenStack Neutron to offload network processing and provide enhanced services as showin in Figure 2.7. A more comprehensive list of features is:

- Layer 3 distributed routing with the L3 DVR (Distributed Virtual Router)
- Layer 2 distributed switching.
- Full support for clustering and HA (High Availability).
- Security Group support implemented through OpenFlow rules instead of IPTables-based OpenStack Security Groups.
- Security Group stateful connection tracking.
- Service Function Chaining.
- Network Virtualization graphical interface for DLUX.

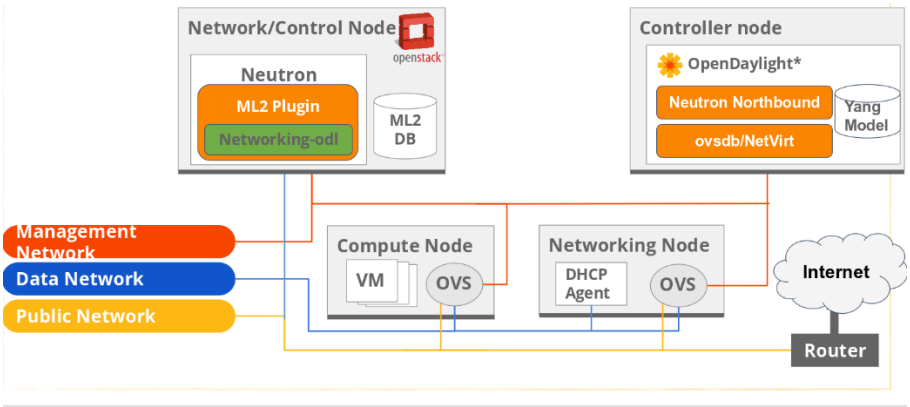

Figure 2.7: OpenStack and OpenDaylight Integration [68]

OpenDaylight original code was written in Java. Luckily, it can be run under the OpenJDK Java runtime environment (JRE) [69]. On the other hand, it is highly modular. The ODL software distribution comes in Apache Karaf format [70]. Karaf is a type of application container commonly used in the Java ecosystem. ODL applications are installed in the form of Karaf features. The features most relevant to NFV and SFC are briefly introduced in the following list:

- **Service function chaining** (SFC): Provides an API for applications to define chains and the service logic needed for ODL to provision the requested chain in the network.
- **NetVirt**: The NetVirt code was split from OVSDB. It provides OVS-based virtualization for software switches, Hardware VTEP (VXLAN Tunnel Endpoint) and Service Function Chaining. The SFC Classifier and SFF are controlled via OVSDB and Openflow.
- **Group-based policy** (GBP): Provides intent-based policy abstraction for ODL. GBP integrates with SFC in by configuring policy-based classifier rules in a declarative way.
- **VPN service**: Implements the required infrastructure to support L3 VPN services. In Boron some of the code has been moved to NetVirt and Genius, reducing VPNService to the BGP Quagga interface.

2.5. Open vSwitch

Open vSwitch (OVS) is a multilayer software switch that can bridge traffic between VMs to the physical network and vice versa. The OVS project goal is to support the common Layer2 features (tagging, tunneling, bonding, accounting, etc.) providing programmatic extensions which make OVS suitable for SDN deployments. OVS can operate in user-space or kernel space, with the aid of a kernel module.

As shown in Figure 2.8, the main components in the OVS architecture are the OVS vSwitch daemon (OVS-vSwitchd) and the OVS database server (OVSDB-server). OVS-vSwitchd is the core of the system. It communicates with ODL via OpenFlow. The OVSDB-server is a database containing the switch state and configuration. It interacts with ODL using the OVSDB protocol [71].

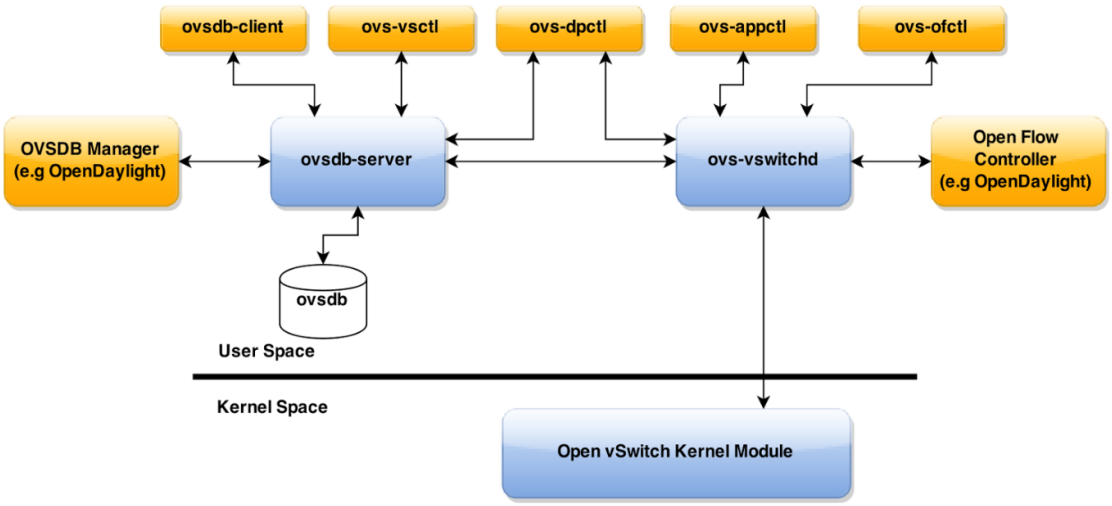

Figure 2.8: Open vSwitch architecture [68]

OVS supports several Linux-based hypervisors and, thus, competes with the Linux bridge, but it overcomes some of its limitations. For instance, support for distributed switching and off-loading to external hardware such as NICs or switches. But as shown in Figure 2.9, where OVS makes the difference to realize the NFV requirements in terms of throughput and latency with DPDK[7], which runs in user-space.

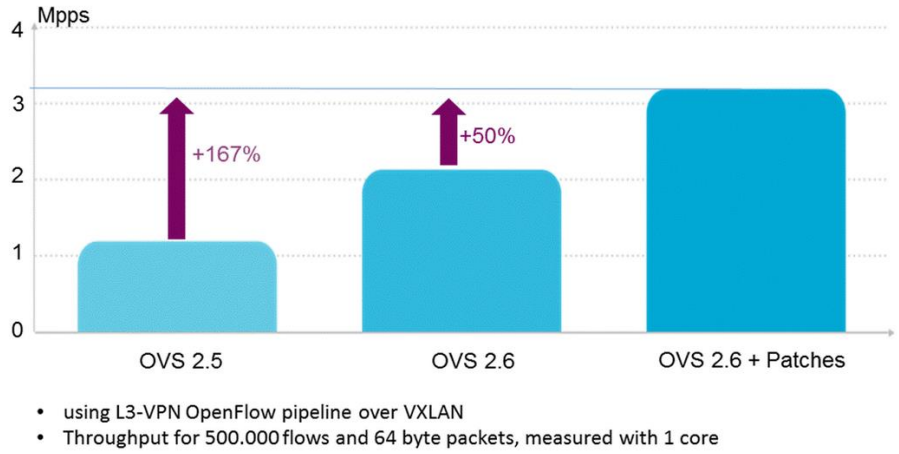

Figure 2.9: Improvements of Open vSwitch with DPDK [72]

Flow rule programming determines how inbound/outbound traffic should be treated within the virtual switches. OVS has two operating modes:

- Standalone is the default mode, and it simply acts as a learning switch.
- Secure mode relies on an external SDN controller to insert flows rules.

OVS plays a key role in the SFC architecture as the Neutron Integration OVS bridge (br-int) acts as the service function forwarder (SFF). The br-int is an

[7] DPDK is a set of multiplatform libraries and drivers for fast packet processing. The main libraries optimize packet transmission impact on the CPU and implement fast packet capture.

OpenFlow switch hosted in an OVS where all guest VMs are connected. In the OPNFV SFC implementation, there is an SFF in each compute node. The OPNFV OVS distribution includes patches to support the SFC encapsulation [48]. The patch supports the following NSH stacks:

- **ETH + NSH**, which encapsulates NSH directly over Ethernet.
- **VxLAN-GPE + ETH + NSH**, which encapsulates ETH+NSH on top of VxLAN Generic Protocol Extension[8].

The OPNFV SFC scenario is only implemented with VXLAN-GPE + ETH + NSH at the moment [37].

2.6. Open Platform for NFV

The Open Platform for NFV (OPNFV) project defines a partial reference implementation of the ETSI NFV architecture. It employs exclusively open source tools to provide a carrier-grade environment with system level integration, deployment and testing. By means of the open source collaboration philosophy brings together the standards bodies, open source communities, and vendors to become a de facto platform for the industry.

OPNFV is a rather young project. It has brought 3 releases, named alphabetically: Arno, Brahmaputra and Colorado. It follows a 6 month release cycle with follow-on releases. At the time of writing (February 2017), the current is the C release (Colorado). Colorado offers more than 40 scenarios. Taking advantage of the OPNFV continuous integration/continuous deployment (CI/CD) pipeline, the scenarios are thoroughly tested. Existing scenarios can be extended composing additional components/features.

OPNFV is primarily focused on building NFVI, VIM and, in latest releases, MANO. It builds upon open source projects such as OpenDaylight, OpenStack, Ceph Storage, KVM, Open vSwitch, and Linux. OPNFV supports OVS, fd.io and DPDK as forwarding solutions. Similarly to Openstack, it is comprised of several core projects and other approved projects. Based on the selection of components/features, several scenarios can be deployed.

The OPNFV scenarios that implement the SFC ODL feature are labelled “os-odl_l2-sfc-ha” and “os-odl_l2-sfc-noha”. Other scenarios implement interesting NFV features, like KVM tuned for NFV, OVS with DPDK or BGPVPN, that implements layer 3 MPLS-based VPNs. But they have not been considered as some features from these scenarios are not compatible with the SFC feature. The scenario naming convention follows the scheme below:

os-[controller]-[feature]-[mode]-[option]

[8] Virtual eXtensible Local Area Network (VXLAN) is a layer 2 overlay network encapsulation commonly used in OpenStack project networks as it supports a much larger tenant number than classical VLANs (24-bit segment ID versus 12 bit vlan VLAN ID). VXLAN generic protocol extension (GPE) brings support for payloads other than Ethernet to VXLAN.

Name	Short name
Accelerated Open vSwitch	os-nosdn-ovs
Layer 3 overlay using OpenDaylight	os-odl-l2-bgpvpn
FD.io based forwarding using OpenDaylight	os-odl-12-fdio-noha
High availability service function chaining	os-odl-l2-sfc-ha
Service function chaining	os-odl-l2-sfc-noha
Accelerated KVM hypervisor	os-nosdn-kvm-ha
LXD container hypervisor	os-nosdn-lxd-noha
High availability LXD container hypervisor	os-nosdn-lxd-ha

Table 2.2: Partial list of OPNFV scenarios for the Colorado release

2.6.1 OPNFV architecture

OPNFV serves the purpose of building a carrier-grade platform for predictable, repeatable and validated NFV deployment and testing of the provided scenarios. In order to do that, OPNFV integrates all the required tools to deploy end-to-end VNF-based services. Figure 2.10 shows the OPNFV architecture aligned with the project goals:

- VNF life-cycle management.
- Underlay/overlay agnostic VNF/VNFC/PNF interconnectivity.
- VNF instance scaling to meet current requirements.
- Failure detection and remediation from any element of the infrastructure.

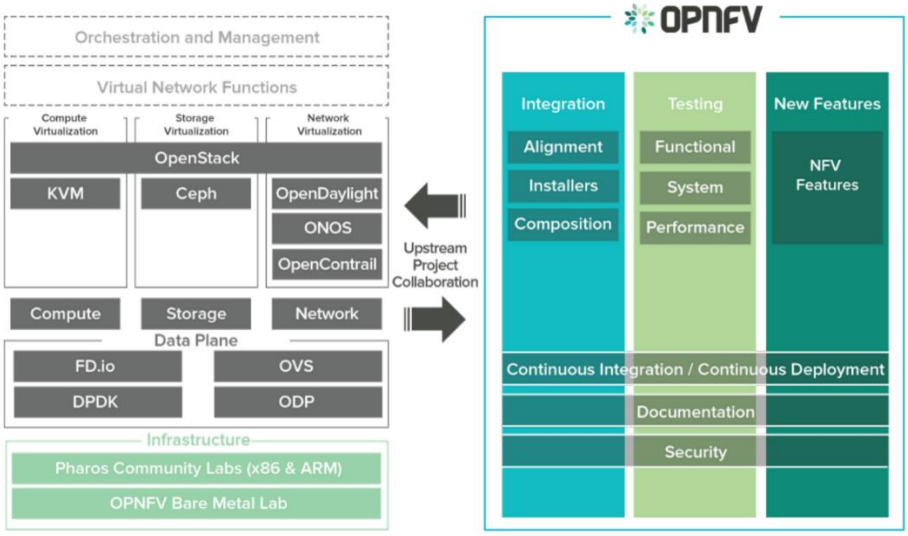

Figure 2.10: OPNFV Architecture [44]

2.6.2 Installers for OPNFV

A typical OPNFV POD consists of an OpenStack cluster and compute nodes for VNF deployment. The POD is bootstrapped using a provisioning server. After installing the provisioning server the OpenStack and additional components are pushed to the nodes. 4 OPNFV projects are focusing on installation. Each one is based on a different OpenStack installer and implements the reference scenarios to a different extent:

- Project Apex [73] relies on TripleO [74], an installer for theRed Hat Distribution of OpenStack (RDO) project [75]. TripleO stands for Openstack on OpenStack as an underlay Openstack instance is used to deploy the OPNFV OpenStack cluster.

- Compass4NFV [76] is based on Compass [77], which is an open source project that provides automated deployment and management for OpenStack. It can be considered what the LiveCD is to a single box but for a pool of servers.

- Fuel [64] is the Mirantis OpenStack (MOS) [78] distribution installer. It provides deployment, management and validation workflows for OpenStack through an intuitive web interface. The additional OPNFV components are integrated as plug-ins.

- JOID (Juju OPNFV Infrastructure Deployer) project [79] is based on Canonical Juju [80] and MAAS [81]. Juju is a service-oriented model deployment framework. MAAS (Metal as a Service) abstracts physical servers and turns a server farm into an on-demand server resource.

TripleO and Fuel are the most featureful OPNFV installers. Both toolkits support the SFC scenarios. But after testing them, Fuel was chosen as the most suitable thanks to its excellent documentation and easy learning curve. Also, TripleO required intelligent platform management interface (IPMI) for node provisioning, which is not supported on the available computers in the lab 235G at EETAC, where the tests were conducted.

2.6.3 Functional testing in OPNFV

A fundamental tool in distributed platform and scenario deployments is a functional testing framework to validate from the most basic features to the most complex scenario combinations. In OPNFV, the Functest project provides test cases developed to validate the different supported features/scenarios and other tests from upstream projects.

The Functest toolkit is provided in a Docker image [82]. This allows running Functest on any platform or operating system, but the most common place to install the container is on the jump server, as connectivity to the POD networks is required. Functest uses the different OpenStack APIs to interact with the Fuel deployment.

The Functest Docker container automates the following tasks:

- Retrieval of the OpenStack credentials.
- Preparation of the environment according to the system under test (SUT).
- Execution of the appropriate functional tests.
- Storing of the results into the test result database.

Table 2.3 show the test case categorization in Functest.

Category	**Description**
ARMband	Support for the AArch64 infrastructure.
healthcheck	Basic OpenStack commands.
smoke	vPings, Tempest and rally smoke tests.
sdn_suites	Specific SDN feature tests.
features	OPNFV feature project functional test suites.
openstack	Advanced, long duration OpenStack tests.
vnf	Complex scenario orchestration (e.g. vIMS)

Table 2.3 OPNFV Colorado functional test categories [83]

CHAPTER 3. DEPLOYMENT AND TESTING OF SFC SCENARIOS

This chapter contains the technical implementation details of the tested OPNFV scenarios. First of all, a high level overview of what constitutes an OPNFV point of delivery[9] (POD) is introduced. Then, both virtual and hardware environment deployments are described. Finally, the deployment based on Fuel and SFC scenario tests common to virtual and hardware labs are covered in detail. Only functional tests have been performed due to hardware limitations and laboratory availability.

3.1 OPNFV reference POD

An OPNFV POD supports the installation of the provided scenarios to test NFV use-cases. In order to build an OPNFV POD a virtual or physical laboratory has to be set up based on a recommended reference architecture. A Fuel-based OpenStack architecture design consists of determining how many nodes to deploy and which roles to assign to each node.

The node and role planning is a decision that will impact on the performance and high availability of the POD. Spreading the roles and workloads over many servers will maximize performance and minimize bottlenecks. For smaller hardware configurations multiple roles can be combined on fewer nodes and networks can be grouped into a limited number physical NICs.

For a basic testing POD, at least a compute node and 3 controller nodes are required, or 1 controller and 3 compute nodes. Ideally, storage nodes should be separated from the controllers to avoid resource contention, but this would need a total of 7 nodes (3 OpenStack and 3 Ceph OSD controllers). Based on a trade-off decision, as shown in Figure 3.1, 6 servers have been decided to build an OPNFV SFC scenario:

- 1 Jump server[10] acting as a provisioning node
- 3 Controller nodes to conform the OpenStack Cluster
- 2 Compute nodes for VNF deployment

On the networking side, the following networks/VLANs are required [84]:

- 2 Networks requiring routed access to the Internet:
  - Admin/PXE VLAN between the Fuel master and slaves.
  - Public VLAN for used for controllers, HA VIPs, neutron floating IPs and optionally for control nodes.
- 3 non-routed networks:
  - Private VLAN for NFV data-path such as VxLAN tunnels.

[9] In network architecture design, a point of delivery (POD) defines a repeatable building block which can be deployed in the same way multiple times to simplify network complexity.
[10] A jump server or jump host is server used to manage devices in a separate security zone.

- MGMT VLAN for OpenStack communications and Keystone endpoint URL.
- Storage VLAN for network storage communication.

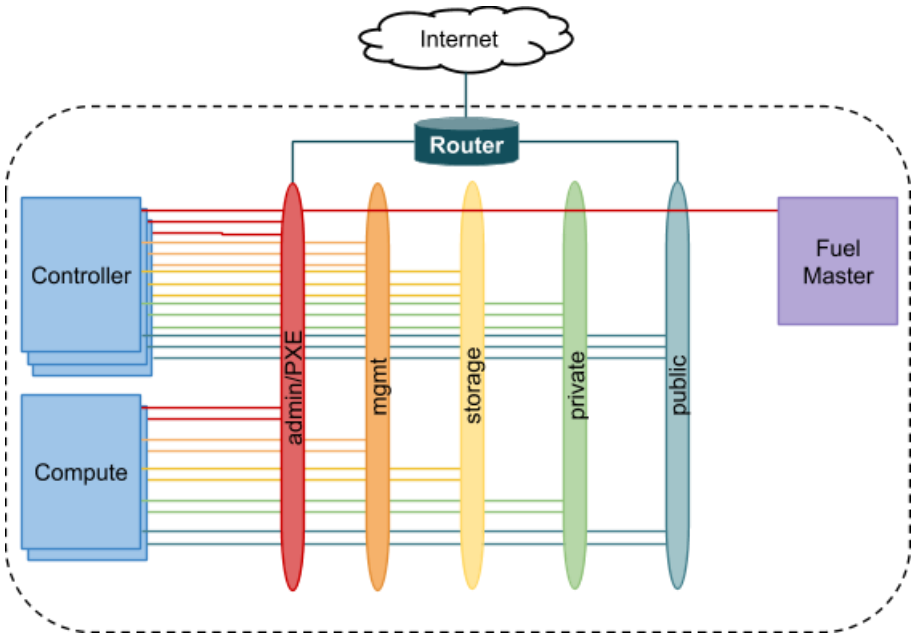

Figure 3.1**:** OPNFV generic POD architecture

3.2 Scenario Deployment Workflow

The deployment of an OPNFV POD based on Fuel consists of several phases. The following list contains a high level overview of the tasks involved:

1. Set up POD (in the case of bare-metal environment).
2. Gather parameters for the given POD topology.
3. Install Fuel master. This can be either:
   a. Bare metal server: Fuel setup is included on a CentOS ISO.
   b. Virtual Machine: Ubuntu 14.04 as base OS and Fuel VM.
4. Configure Fuel parameters before post-install.
5. Connect via SSH and install the scenario required Fuel plugins.

After these steps, the Fuel Web UI can be accessed via HTTPs and is ready for scenario creation and deployment.

6. Boot Fuel slave nodes so they are booted via Fuel PXE server.
7. Configure a deployment environment:
8. Assign node roles.
9. Configure node disk partitioning.
10. Configure node network interface mapping.
11. Configure scenario features
12. Test networking.
13. Execute the deployment.

14. Validate the deployment.

3.2.1 Automated Scenario Deployment

The SFC scenario deployment through the Fuel web UI has proven unreliable and error-prone. The most convenient way to deploy the scenario has been installing Fuel by script. The deployment script makes use of the Fuel CLI and is included as part of the continuous integration (CI) pipeline provided in the Fuel Gerrit code repository [85]. The script features:

- Automatic Fuel master installation and configuration.
- Automatic scenario configuration with validated templates.
- Automatic deployment on a Pharos-compliant lab.

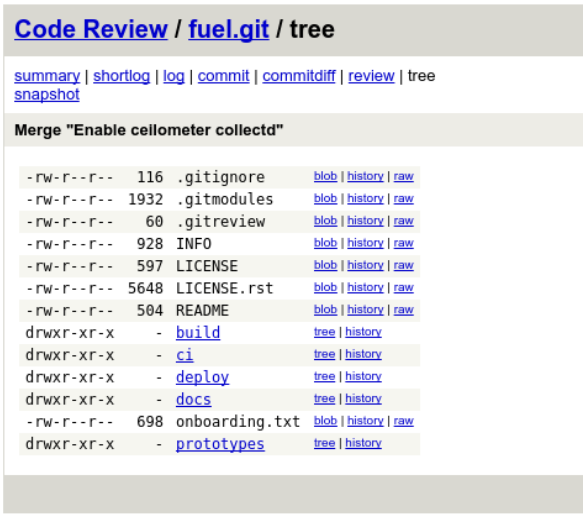

Figure 3.2: OPNFV Fuel GIT repository [85]

As shown in Figure 3.2, the CI pipeline contains a set of scripts and configuration files invoked by the OPNFV CI Infrastructure (Jenkins [86]) to automatically deploy and validate the Fuel-based supported scenarios. It was contributed by Ericsson and interacts with the Fuel CLI in an automated manner. The Fuel master is installed as VM using libvirt [87], which is the most flexible option.

The Fuel GIT repository contains three main folders:

- Build: contains the tools responsible for the Fuel ISO build.
- Deploy: contains Python scripts and templates for POD deployments.
- CI: contains Bash wrappers for the Build and Deploy tool-chains.

3.2.2 Deploy Script Overview

The CI deployment script is a shell script that takes care of the several installation phases. The bash script is a wrapper for a Python script which, once invoked, it

performs different tasks depending on the configuration files interacting with the Fuel CLI. The main deploy.py requires mainly 3 inputs:

- Fuel ISO image to deploy.
- Deployment environment adapter (DEA) configuration file, dea.yaml, contains the parametrization of a Fuel deployment (network node settings, roles, scenario features, etc.)
- Deployment Hardware Adapter (DHA) configuration file, dha.yaml, specifies the number of nodes and type/properties for the specific hardware adapter.

The dea.yaml is combined by deploy.sh from the following inputs:

- DEA POD override configuration file, dea-pod-override.yaml, containing the deployment configuration parameters that supersede the base DEA definition (dea_base.yaml).
- The scenario configuration file, which overrides the settings in the generic scenario.yaml definition. For the HA SFC deployment, this is:

ha_odl-l2_sfc_heat_ceilometer_scenario.yaml

3.2.3 Deployment hardware adapter customization

The Deployment Hardware Adapters are implementations of the DHA API for a specific hardware platform. The API implements bindings for out-of-band management such as IPMI (Intelligent Platform Management Interface) in order to power-cycle the Fuel slave nodes to boot them into PXE and make them available for the scenario deployment once they are bootstrapped by Fuel.

There are adapters for both virtual and physical servers. There is also a hybrid libvirt/IPMI adapter (virtual fuel master and bare metal fuel slaves) which is suitable for the EETAC's 235G lab. However, as the lab computers lack an IPMI, a custom adapter has been implemented. The adapter is a dummy API implementation that relies on human interaction to manually boot the computers.

3.3 Virtual lab environment

In the virtual environment, the OPNFV platform is installed on a single server using Linux Kernel-based Virtual Machines (KVM) to deploy guests acting as fuel master and slaves. The virtual lab environment has been used to test the installation and deployment procedures and familiarize with the scenarios as nesting hypervisors is not a viable option for real NFV performance tests.

The virtual POD consists of 5 KVM guests and 4 libvirt networks interconnecting them as shown in Figure 3.3. The virtual networks are Linux bridges with NAT for inbound/outbound traffic. During the deployment, a Fuel master VM and all the virtual networks are set up. Then, the Fuel slave VMs are booted in PXE mode to be bootstrapped prior to the scenario deployment launch.

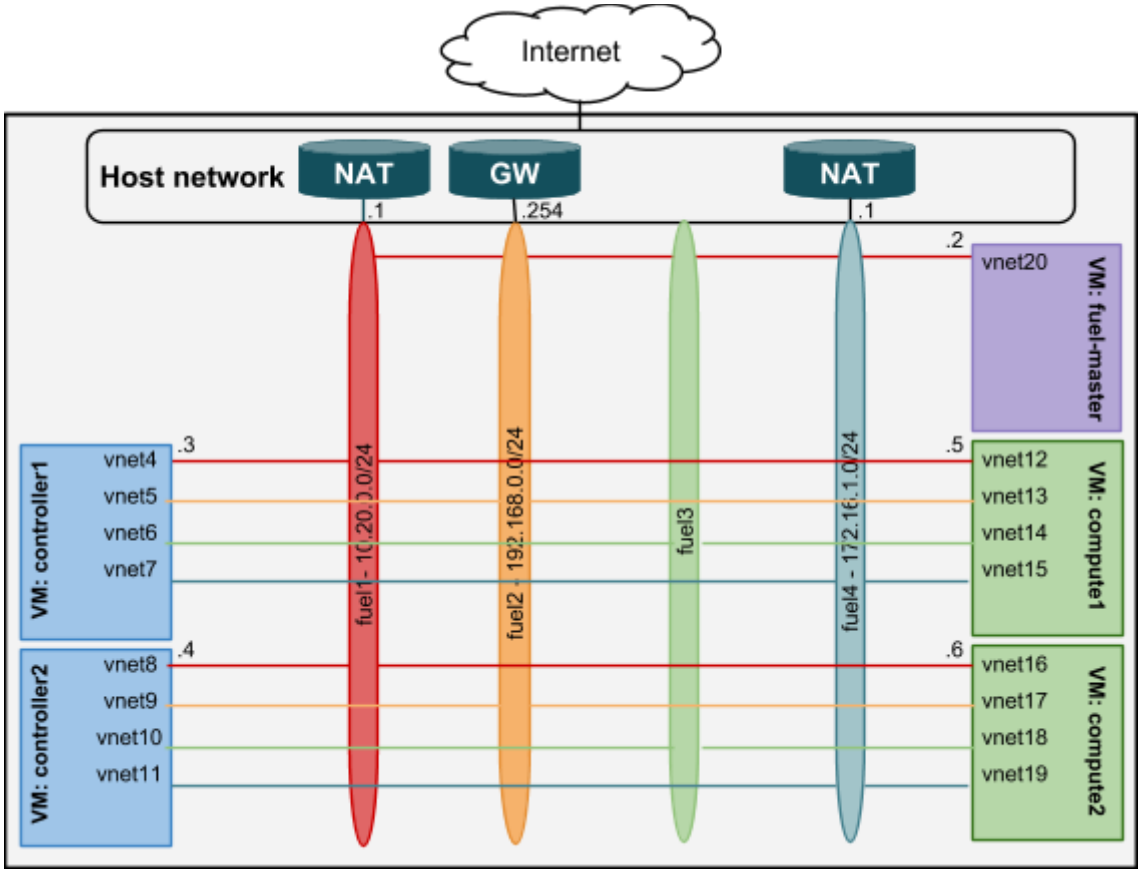

Figure 3.3**:** OPNFV virtual POD logical topology

An IBM x3550 M3 server has been available to build this POD. The following table compares the requirements against the available hardware.

Component	Required	Available
Processor	Sandy Bridge CPU with support for 5 virtual cores (5 physical cores at least)	Nehalem (previous than Sandy Bridge) CPU with 4 physical cores
Memory	54GB at least (10GBs per node and 4 for the jump server)	70GB
Disk	704GB at least (128GBs per node and 64GB for the jump server)	1.4TB

Table 3.1: OPNFV requirements for a virtual lab

3.4 Bare-metal lab environment

The OPNFV bare metal lab specification lies within the scope of the Pharos project, which defines the hardware environment that can host a fully featured OPNFV platform able to run all kind of scenarios and tests. The Pharos

specification also defines remote access to serve as a community lab and connect to the Continuous Integration Infrastructure. The specification provides:

- A secure, scalable, standard HA environment
- Full release deployment lifecycle support and CI/CD integration
- Full functional and performance testing
- Secure remote access to the lab

The 235G lab at EETAC has been made available as a bare-metal environment for the scenarios tested during this project. The lab has a total of 24 HP Compaq 8100 Elite Convertible Minitower PCs. A row of 6 computers can act as an OPNFV POD. The required networks can be implemented with a pair of Netgear unmanaged switches. Access to the BIOS is needed to modify boot settings. Table 3.2 compares the requirements against the available hardware:

Component	**Required**	**Available**
Processor	Xeon E5-2600v2 Series (Ivy Bridge or newer) 4–12 cores	Core i5-650 Series (Westmere) 2 cores
Memory	32G RAM Minimum	4/8/16GB depending on the lab row/column
Disk	2x1TB + 1x100GB SSD: -1TB: OS and tools -1TB: Ceph storage -100GB: Ceph journal	500GB
Network	2x1G Control 2x10G Data (DPDK) 2x40G Storage	5x100Mbps Intel 82557 (no DPDK)
Out-of-band Management	IPMI	Not Available
Power supplies	2 Recommended	1

Table 3.2: OPNFV requirements for a bare-metal deployment

Although these requirements are difficult to fulfill without enterprise-grade hardware, scenarios can be deployed bearing in mind the following constraints:

- Processor: number of VNFs deployed and vCPUs per VM.
- Memory: amount assigned per VM and extra RAM for the controller nodes.
- Storage: local drive/volumes per VM and object storage.
- Network: bandwidth per virtual machine and network storage.

The deployed OPNFV POD at lab 235G consisted of 6 servers and 2 Netgear switches interconnecting them. 3 NICs are used in the Fuel master and 2 in each Fuel slave as shown in Figure 3.4. VLAN tagging is used. Connectivity is achieved via the lab structured cabling. For PXE booting, the fuel slave servers use interface eth4 to bypass the lab's Rembo PXE server. Table 3.3 defines the networking details of the deployed lab:

Name	BRIDGE / VLAN	Subnet
Admin/PXE	Switch 1 / Untagged	10.20.0.0/24
Public	Switch 2 / Untagged	172.16.0.0/24
Management	Switch 2 / 101	192.168.0.0/24
Storage	Switch 2 / 102	192.168.1.0/24
Private	Switch 2 / 103	192.168.2.0/24

Table 3.3: OPNFV POD networks for a virtual deployment

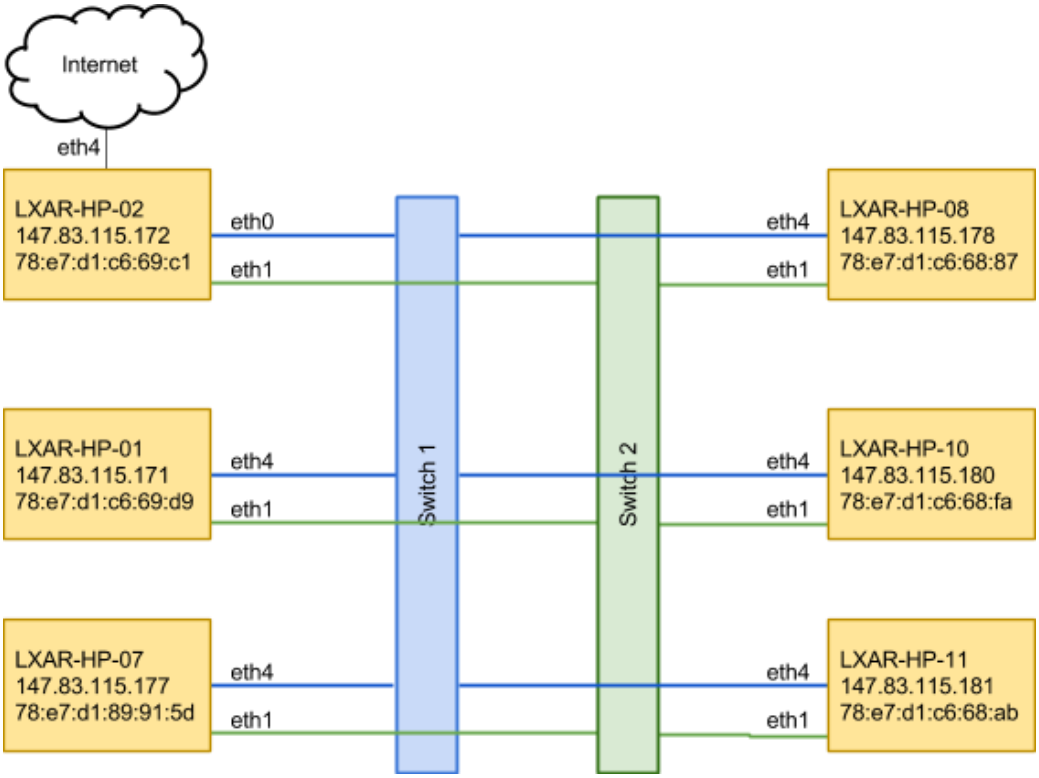

Figure 3.4**:** OPNFV bare-metal POD physical topology

As shown in Figure 3.5, three servers are used as controllers (blue boxes) and two as compute (green boxes). The remaining server is the jump server which hosts a VM for the fuel-master. The fuel-master VM is linked to the physical network with Linux bridged networking via two interfaces. The third interface provides Internet access to the jump server OS and also to the Admin/PXE and Public networks through each NAT provided by libvirt networks.

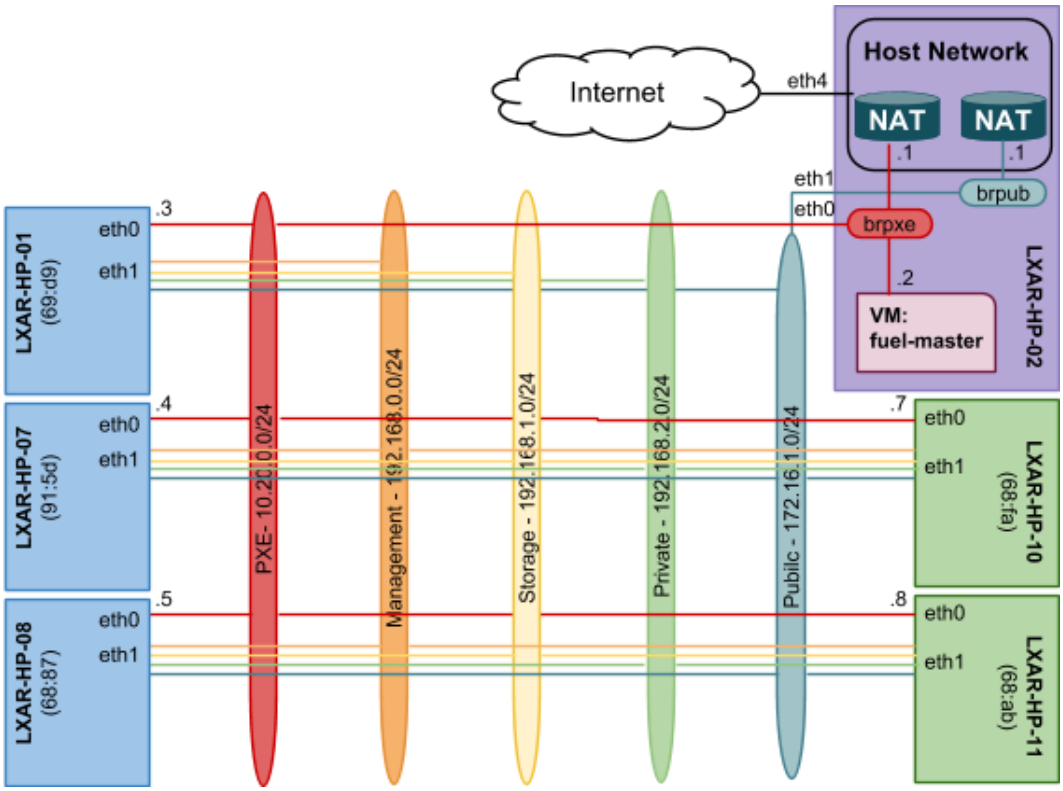

Figure 3.5**:** OPNFV bare-metal POD logical topology

Once the Fuel deployment is completed and validation succeeds, the SFC scenario is ready for VNF deployment and SFC creation. The 5 Fuel slaves contain a highly available OpenStack environment integrated with a standalone OpenDaylight SDN controller. Currently, the ODL Neutron plugin does not support ODL HA. Table 3.4 shows the node mapping for this deployment and the management IPs.

Hostname	Roles	Admin IP	MAC
LXAR-HP-01	Controller, OpenDaylight	10.20.0.3	78:e7:d1:c6:69:d9
LXAR-HP-07	Controller, Mongo, Tacker	10.20.0.4	78:e7:d1:89:91:5d
LXAR-HP-08	Controller, Ceph OSD	10.20.0.5	78:e7:d1:c6:68:87
LXAR-HP-10	Compute, Ceph OSD	10.20.0.7	78:e7:d1:c6:68:fa
LXAR-HP-11	Compute, Ceph OSD	10.20.0.8	78:e7:d1:c6:68:e2

Table 3.4: Role to server mapping in the bare-metal POD

3.5 Details of the SFC scenario

The details on the SFC scenario further developed in this section are based on the virtual OPNFV POD alone due to performance limitations in the bare-metal lab. This limitations prevented Tacker to complete the VNF deployment.

In this case, 4 Fuel slave VMs were used in a non-HA OpenStack environment integrated with an OpenDaylight SDN controller. Table 3.5 shows the node mapping for this deployment.

Hostname	Roles	Admin IP	MAC
controller1	Controller, Mongo, Tacker	10.20.0.3	52:54:00:a9:4a:fa
controller2	Ceph OSD, OpenDaylight	10.20.0.4	52:54:00:a5:23:08
compute1	Compute, Ceph OSD	10.20.0.5	52:54:00:bd:90:f8
compute2	Compute, Ceph OSD	10.20.0.6	52:54:00:91:65:47

Table 3.5: Role to server mapping in the virtual POD

Each role performs the following functions:

- The Controller provides central management for the OpenStack cluster.
- Mongo provides NoSQL database backend for Ceilometer.
- Tacker acts as a VNF Manager and SFC Orchestrator.
- The Ceph OSD (Object Storage Daemon) provides clustered storage.
- OpenDaylight controls all the OVSs deployed in the OpenStack cluster.
- Compute hosts VMs/VNFs. It Provides SFF in the “br-int” OVS bridge.

3.5.1 Tacker workflow for VNF and SFC

Tacker CLI is used as VNF manager in this scenario. The included Tacker version is a pre-Mitaka release with also support the SFC API. All the required configuration, service chains and classifiers, can be performed using Tacker installed in the second OpenStack controller. Tacker creates service chains, classification rules, creates SFs through Heat and communicates the relevant configuration to ODL.

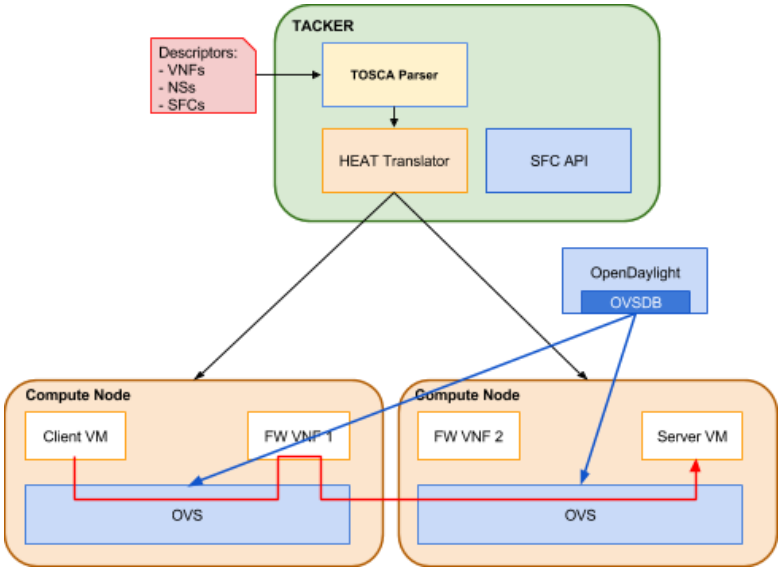

Figure 3.6: Tacker configuration flow for SFC scenario

3.5.2 Scenario high level logical topology

The OPNFV SFC scenario allows the creation of service chains, classifiers, and VNFs. Client to server traffic can be directed through the provisioned service chains. The chaining feature is showcased creating two separate Firewall Service Functions implementing a different set of filtering rules via one chain or another to prove the ability to send traffic based on classification.

The service chain creation is achieved configuring the OpenDaylight SFC feature. This configuration will, in-turn, configure service function forwarders to route traffic to the service functions. The br-int OVS in the compute nodes acts as SFF. The classifier used in this scenario is implemented with Netvirt [88]. The following are the installed ODL SFC and Netvirt Karaf features:

SFC Features	Netvirt Features
odl-sfc-model odl-sfc-provider odl-sfc-provider-rest odl-sfc-ovs odl-sfc-openflow-renderer	odl-ovsdb-openstack odl-mdsal-xsql odl-neutron-service odl-neutron-northbound-api odl-neutron-spi odl-neutron-transcriber odl-mdsal-apidocs odl-ovsdb-southbound-impl-rest odl-ovsdb-southbound-impl-ui

Table 3.6: ODL features installed in the SFC scenario

The classifiers are configured through the ODL Netvirt feature as shown in Figure 3.7. Alternatively could be configured via group based policy (GBP) [89]. Netvirt handles VM networking and can create basic classification rules to steer specific traffic to a service chain. The rules are 5-tuple based on the TCP/IP connection values: source IP address/port, destination IP address/port and protocol.

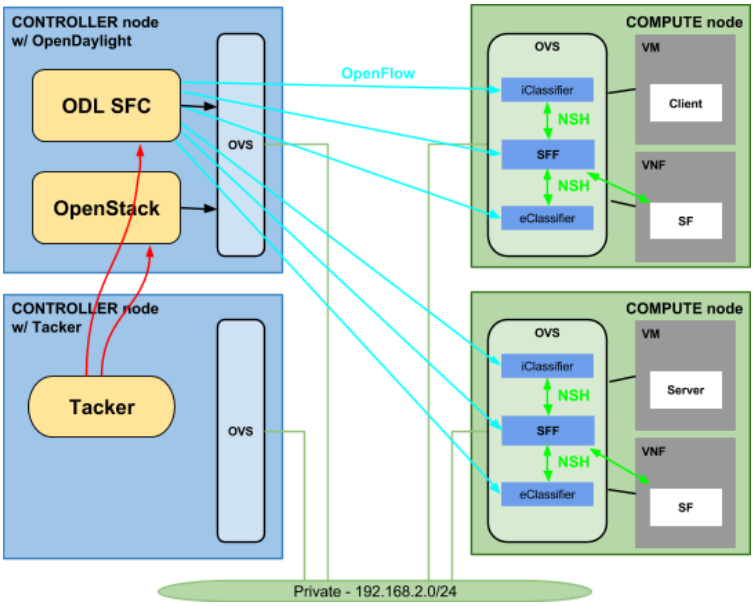

Figure 3.7: SFC scenario high level virtual network topology

3.5.3 Scenario limitations

In all SDN-based scenarios, VXLAN tunnel segmentation is a requirement for the ML2 Neutron plugin. When VXLAN segmentation is used, tenant traffic is encapsulated in tunnels. This is more flexible than using VLANs as there is no need to configure the external switches. Moreover the tenant space is much larger and subnet overlapping is supported. Other possible encapsulation mechanisms in Neutron tunneling segmentation topologies is GRE, but the number of tenants is smaller.

Currently, OpenStack terminates the VXLAN tunnels in the br-int OVS bridge instead of the service function VNF. This does not work correcty with the ODL SFC feature as the tunnel should be terminated in the VNF so the SF can access the NSH header. As per [90], a workaround was developed and it is implemented in the OPNFV SFC scenario. The workaround works by sending the packets to the SF VNF with the intact VXLAN-GPE header as shown in Figure 3.8.

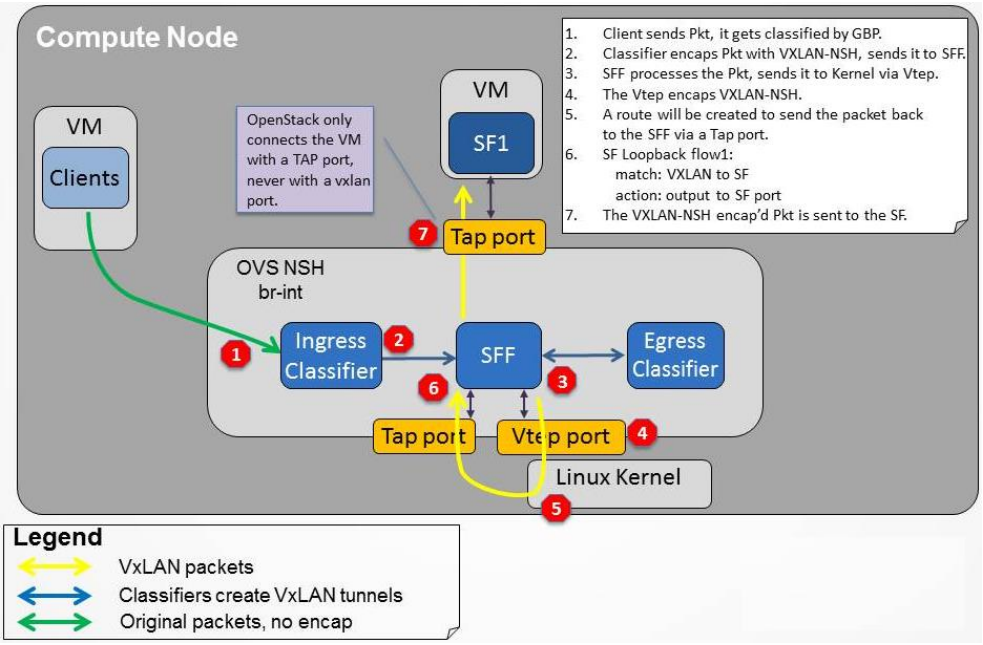

Figure 3.8: SFC scenario VXLAN workaround [91]

3.6 Tests over the SFC scenario

Due to hardware limitations, deployment on the bare-metal lab was completed successfully but tests were not able to complete due to a bug [92] in the Fuel Tacker plugin which prevented the instantiation of VNFs. In the conclusions, workarounds for these limitations are proposed. The tests conducted on the virtual lab only include functional testing as performance tests in a nested virtual environment do not make sense.

3.6.1 Scenario functional tests

Functest [93] has been used to test the deployed scenario SFC features. Functest project provides a Docker image with a set of predefined functional tests. As some glitches have been observed during subsequent POD installations it is recommended to run at least a full health check in addition to the checks already performed by Fuel. Table 3.7 shows how to run the tests.

	Command	Description
1	sudo docker pull opnfv/functest:colorado.3.0	Download Docker image
2	sudo docker run --net=docker-fuel1 --rm --privileged --name Functest -it opnfv/functest:colorado.3.0	Run Docker container
3	./repos/releng/utils/fetch_os_creds.sh -d /home/opnfv/functest/conf/openstack.creds -i fuel -a 10.20.0.2 -v	Load OpenStack credentials
4	functest env prepare	Prepare test environment
5	functest tier run healthcheck	OpenStack checks
7	functest tier run sdn_suites	ODL integration checks
8	functest tier run odl-sfc	SFC scenario checks

Table 3.7: Functional tests run on the virtual scenario

The tests consist on instantiating a pair of virtual firewalls, one filtering HTTP and the other one SSH. Two additional VMs are used to generate traffic on ports 80 and 22 alternatively. The traffic is redirected to one firewall or the other depending on the classification criteria mapped to a service function chain. Figure 3.9 shows all the VMs and service functions involved in the test topology.

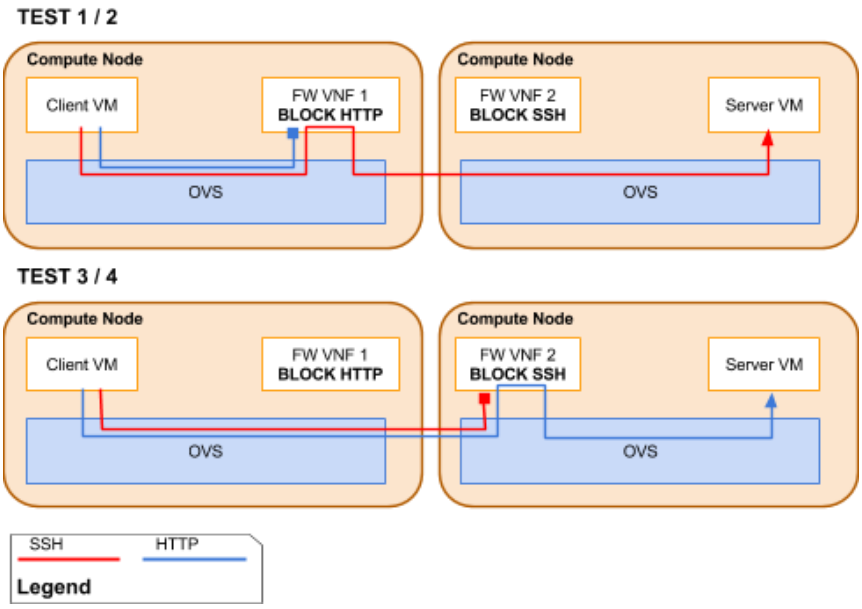

Figure 3.9: SFC scenario service chaining tests

Functional test categories are ordered by layers. If the first layer succeeds, Functest proceeds with the following. The SFC scenario with an High Availability (HA) OpenStack cluster did not pass the most basic test as shown in Table 3.8. Thus, all other tests failed. The issue was that VM instances were not getting IP via DHCP. Troubleshooting yet another failure was out of the scope of the project, so the rest of the tests were performed in the non-HA scenario.

Scenario: os-odl_l2-sfc-noha			
Test suite / Test case		**Description**	**Result**
OpenStack tests		Verify basic operation in VIM	PASS
ODL integration tests		ODL-Neutron M2L integration	PASS
ODL-SFC tests	Test 1	FW1 blocks SSH	PASS
	Test 2	FW1 allows HTTP	PASS
	Test 3	FW2 blocks HTTP	PASS
	Test 4	FW2 allows SSH	PASS
Scenario: os-odl_l2-sfc-ha			
Test suite / Test case		**Description**	**Result**
OpenStack tests		Verify basic operation in VIM	FAIL
ODL integration tests		ODL-Neutron M2L integration	FAIL
ODL-SFC tests	Test 1	FW1 blocks SSH	FAIL
	Test 2	FW1 allows HTTP	FAIL
	Test 3	FW2 blocks HTTP	FAIL
	Test 4	FW2 allows SSH	FAIL

Table 3.8: Functional test results

The tests are run creating all the VMs and VNF in the same compute node. Additional tests have been conducted to verify the VXLAN network overlay between the compute nodes by migrating VMs and VNF between nodes. Creating more complex service function chains with multiple service functions has not been successfully as it is currenly unsupported in OPNFV Colorado.

CHAPTER 4. Conclusions

This chapter concludes the main part of the document with finishing thoughts on the project achievements and gaps. Then a brief consideration on the NFV environmental effects precedes a list of proposals to further expand on related NFV topics in terms of open source platforms not covered in the experimentation phases and other trends more focused on academic research.

4.1. Conclusions

NFV SFC scenarios on both virtual and physical infrastructure have been deployed successfully. OPNFV has proven a comprehensive framework to experiment with the cutting-edge reference implementations of the selected technologies during the state-of-the art analysis initial project phase. The faced difficulties typical to open source projects have been overcome with lots of effort and a bit of help of the OPNFV community.

The installation procedures intended for regular users have been a big stopper for the project development, especially for the technical part. After doing a lot of trial and error, the continuous integration automated deployment tools have been the only way to complete deployments successfully. The CI pipeline scripts has been adapted to the Physical lab in the EETAC with a bit of development work although it was not intended to engage in software modifications in the beginning.

The OPNFV integration model allows to implement new use cases before the upstream projects implement them. This serves the purpose of filling the gaps in the technology between upstream projects. However, downstream bug tracking is not well managed and components using customized releases of the parent projects contain bugs already solved in more recent upstream releases. This is a point where OPNFV lags as an integration project.

Although the NFV approach is software-based, the server computing platform is a key element. Modern hardware is required to fulfill the performance requirements of the NFV architecture. The Pharos OPNFV lab specification [94] counts on high-spec hardware and constrained resources in the used computers can prevent successful POD deployment and testing. More processor and memory resources are undoubtedly required.

This shortage in resources can make the deployment time to be very long, which collides with the limited availability of an academic lab. And, in some cases, the installer can even fail due to race conditions[11]. As the deployments are tested in the CI/CD pipeline with cutting-edge enterprise-grade server platforms, this kind of race conditions caused by resource contemption are not detected and diagnosing them can be very difficult.

[11] A race condition in computer software is a failure that happens when the events do not happen in the order or timing they were expected due uncontrolled conditions in the environment.

The testing phase of the project has been contrained to great extent by the limited resources in the bare-metal environment at EETAC's lab 235G, the memory requirements gap almost made the controller nodes unusable due to constant disk thrashing[12], especially the Opendaylight controller. A workaround would be upgrading memory and deploying a non-HA OpenStack deployment, which would require less nodes, and install a standalone ODL controller.

In regards of the SFC tests, the limited availability of VNF images leads to just a few VNF sample images, which in addition are very heavy (more than 2GB). This heavy-duty image caused timeouts during the Tacker/Heat template deployment preventing the setup of the test scenario. A workaround could be installing a more recent Tacker version potentially breaking the integration or using a compressed image recently shared in the OPNFV mailing lists [95].

The current SFC implementation is a bit of a kludge. It contains the OVS patches, tweaks on ODL openflow tables, the VXLAN workaround for service function traffic delivery and a customized Tacker version for SFC and VNF management. Under these conditions, developing tests other than the supported out of the box has proven unfeasible due to the time constraints of the project. In fact, ongoing feature freeze in the next OPNFV release is requiring a great deal of debugging, which indicates that the implementation is not solid enough.

4.2. Environmental impact

One of the promises of NFV technology is the cost reduction by means of making a more efficient use of hardware resources. In this sense, the environmental impact would be the similar to cloud computing. In cloud computing, the highly dense data-centres can have a different environmental footprint in terms of emissions depending on the source of energy. In this area over-the-top providers also lead the race in the energy source transformation of their data-centres towards renewable energies [96].

But as stated in [97], carbon emissions are not the only contributors in the ecological balance of the premises likely to host the NFV infrastructure. For instance, batteries and other kinds of electronic waste, coolant, water, fuel and fire supressing products used in DC environmental maintenance have a high impact which is usually overlooked. In that regard, additional electronic waste due to the high replacement rate of COTS hardware can have an adverse outcome if no recycling policies are enforced.

4.3. Future lines of study

The next release of OPNFV, Danube, will not add new features relating to ODL as it will use the same ODL release. But successive versions will include new features that will allow the realization of more complex scenarios like the ones

[12] Thrashing occurs when the virtual memory sytem of a computer is paging constantly, that is querying the swap memory stored on disk, which is much slower than real memory.

allowing SFC and L3VPN interoperability. The project practical research can be further extended working on combining the current scenarios or creating new ones to validate other NFV use cases. Work can be done to integrate the MANO open source tools in OPNFV to try to orchestrate complex services. A great deal of additional testing can be implemented with the suite of testing tools already available for OPNFV.

Conducting proof of concepts on different SDN controllers can also be another line of study. Besides OpenDaylight, other controllers such as ONOS implement NSH, so comparing the two implementations can be an interesting exercise. Also other service chaining approaches exist. OpenContrail does not use NSH which can also be the object of a gap analysis. The Danube release of OPNFV includes a Kubernetes scenario. The differences that networking with containers will introduce can be a matter of study on the architectural strengths of the current implementation.

In the late phase of writing the Sonata NFV framework was discovered by chance. An evaluation of the Sonata value proposition can also be done and tests over the first release can be conducted. A more promising framework is expected to be OpenECOMP, that will deserve further attention when it is finally released jointly by AT&T, Amdocs and the Linux Foundation [98]. OpenECOM is expected to become a de facto standard changing the whole NFV landscape.

To conduct more specific practical studies, there are multiple open source implementations of the IP multimedia subsystem, for examples Project Clearwater [99], or Open IMS core [100]. Other examples that can serve as testbed for NFV are CORD [101] or OpenAirInterface [102], which aims to develop an open 5G cellular stack on COTS hardware. In the field of strictly academic research, several ideas can also be developed instance:

- Research on the evolving IETF standards with the additional drafts that are refining the SFC architecture and adding new NSH use cases like multi-domain services, an application on broadband service provider networks or network security.

- Service chaining algorithms to provide high availability mechanisms for SFC. HA in a SFC domain will be a combination of individual components and dynamic chain re-route. Interaction between VNF, SDN, MANO and NFVI will be required.

- Big data analytics for quality of experience, performance monitoring and reliability of network services will be a masterpiece in cutting-edge self-optimizing and self-healing networks. MANO will be a broad field of experimentation in this direction.

- Energy-efficient NFV architecture study is a mandatory research field if environmental impact is taken seriously. Energy optimization in terms of the ratio between power consumption and useful outcome needs to be benchmarked. The virtualization efficiency needs also to be taken into account in comparison with bare-metal equivalent resource pools.

REFERENCES

[1] P. Maillé, *Telecommunication Network Economics.* Cambridge University Press, 2014.

[2] OPNFV Project, “Virtualizing Customer Premises with Service Function Chaining.” [Online]. Available: https://www.opnfv.org/wp-content/uploads/2016/11/opnfv_odl_vcpe_sfc_brief.pdf. [Accessed: 31-Jan-2017].

[3] Z. Yao, J. Bagga, and H. Morsy, “Introducing Backpack: Our second-generation modular open switch,” *Facebook Code Engineering Blog*. [Online]. Available: https://code.facebook.com/posts/864213503715814/introducing-backpack-our-second-generation-modular-open-switch/. [Accessed: 01-Feb-2017].

[4] Lerner. Andrew, “Networking Hype Cycle 2016,” *Gartner Blog Network*, 2016. [Online]. Available: http://blogs.gartner.com/andrew-lerner/2016/07/28/networking-hype-cycle-2016/. [Accessed: 01-Feb-2017].

[5] M. Chiosi *et al.*, “Network Functions Virtualisation Introductory White Paper,” 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf. [Accessed: 24-Jan-2017].

[6] T. D. N. Ken Gray, *Network Function Virtualization*. Morgan Kaufmann, 2016.

[7] Verizon, “SDN-NFV Reference Architecture,” 2016.

[8] M.-P. Odini and A. Manzalini, “SDN in NFV Architectural Framework,” *IEEE Software Defined Networks Newsletter*, 2016. [Online]. Available: http://sdn.ieee.org/newsletter/may-2016/sdn-in-nfv-architectural-framework. [Accessed: 02-Feb-2017].

[9] “ETSI - European Telecommunications Standards Institute.” [Online]. Available: http://www.etsi.org/. [Accessed: 02-Feb-2017].

[10] “ETSI NFV ISG Team Board.” [Online]. Available: https://portal.etsi.org/tb.aspx?tbid=789&SubTB=789,795,796,801,800,798,799,797,802. [Accessed: 02-Feb-2017].

[11] ETSI, “Network Functions Virtualisation (NFV) Use Cases,” 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf. [Accessed: 02-Feb-2017].

[12] M. Cohn, “Why Operators Will Deploy Virtualized CPE,” *SDxCentral*, 2015. [Online]. Available: https://www.sdxcentral.com/articles/contributed/why-operators-deploy-virtualized-cpe/2015/10/. [Accessed: 02-Feb-2017].

[13] Juniper, “VCPE & Cloud CPE Solutions - Juniper Networks.” [Online]. Available: https://www.juniper.net/us/en/solutions/nfv/cloudcpe/. [Accessed: 02-Feb-2017].

[14] M. Chiosi *et al.*, “NFV White Paper #2.” [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper2.pdf. [Accessed: 02-Feb-2017].

[15] M. Chiosi *et al.*, “NFV White Paper #3.” [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf. [Accessed: 02-Feb-2017].

[16] “ETSI - Open Source MANO.” [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv/open-source-mano. [Accessed: 02-Feb-2017].

[17] “Open Networking Foundation.” [Online]. Available: https://www.opennetworking.org/. [Accessed: 03-Feb-2017].

[18] “Software-Defined Networking Definition.” [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition. [Accessed: 03-Feb-2017].

[19] Open Networking Foundation, “OpenFlow Switch Specification Version 1.5.1,” 2015. [Online]. Available: https://www.opennetworking.org/images//openflow-switch-v1.5.1.pdf. [Accessed: 08-Feb-2017].

[20] K. Grzegorz and A. Sanchez, *MPLS in the SDN Era*. 2015.

[21] “ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out.” [Online]. Available: http://onosproject.org/. [Accessed: 08-Feb-2017].

[22] L. Peterson, “CORD: Central Office Re-Architected as a Datacenter,” *IEEE Software Defined Networks Newsletter*, 2015. [Online]. Available: http://sdn.ieee.org/newsletter/november-2015/cord-central-office-re-architected-as-a-datacenter. [Accessed: 02-Feb-2017].

[23] “OpenFlow-enabled SDN and Network Functions Virtualization,” 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-sdn-nvf-solution.pdf. [Accessed: 03-Feb-2017].
[24] P. Quinn and T. Nadeau, “RFC7498 - Problem Statement for Service Function Chaining,” *Request for Comments (RFC) Pages - IETF*, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7498. [Accessed: 02-Feb-2017].
[25] V. Suazo and S. Dasgupta, “Network Service Chaining Solutions,” *Cisco Live Sessions*, 2015. [Online]. Available: https://www.ciscolive.com/online/connect/sessionDetail.ww?SESSION_ID=84148&tclass=popup. [Accessed: 02-Feb-2017].
[26] “IETF Service Function Chaining Working Group Charter.” [Online]. Available: https://datatracker.ietf.org/wg/sfc/charter/.
[27] J. Halpern and C. Pignataro, “RFC7665 - Service Function Chaining (SFC) Architecture,” *IETF Datatracker*, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7665.
[28] P. Quinn and U. Elzur, “Network Service Header,” 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-nsh-10.
[29] “Cisco NSH Service Chaining Configuration Guide.” [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/wan_nsh/configuration/xe-16/wan-nsh-xe-16-book.html. [Accessed: 03-Feb-2017].
[30] M. Ricchome, “VNF Mindmap.” [Online]. Available: https://framindmap.org/c/maps/295778/embed?zoom=1. [Accessed: 09-Feb-2017].
[31] “ETSI Network Function Virtualization enters Phase 2.” [Online]. Available: http://www.etsi.org/news-events/news/850-2014-12-news-etsi-network-function-virtualization-enters-phase-2?highlight=YToxOntpOjA7czo1OiJwaGFzZSI7fQ==. [Accessed: 09-Feb-2017].
[32] R. Mijumbi, “Network Functions Virtualization Conception, Present & Future.” [Online]. Available: http://www.slideshare.net/rmijumbi/nfv-tutorial-61544473. [Accessed: 09-Feb-2017].
[33] “Depurando el estándar ETSI NFV: La primera plugtest de ETSI NFV.” [Online]. Available: http://blogthinkbig.com/depurando-el-estandar-etsi-nfv/. [Accessed: 09-Feb-2017].
[34] “OpenStack - Open Source Cloud Computing Software.” [Online]. Available: https://www.openstack.org/. [Accessed: 09-Feb-2017].
[35] “OpenStack ML2 plug-in.” [Online]. Available: http://docs.openstack.org/mitaka/networking-guide/config-ml2.html. [Accessed: 09-Feb-2017].
[36] “The OpenDaylight Platform.” [Online]. Available: https://www.opendaylight.org/. [Accessed: 09-Feb-2017].
[37] “OpenDaylight Project - Service Function Chaining,” *OpenDaylight Wiki*. [Online]. Available: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main. [Accessed: 12-Feb-2017].
[38] “XOS: Service Orchestration for CORD.” .
[39] “OpenContrail.” [Online]. Available: http://www.opencontrail.org/. [Accessed: 02-Feb-2017].
[40] “Open vSwitch.” [Online]. Available: http://openvswitch.org/. [Accessed: 09-Feb-2017].
[41] “The Fast Data Project (FD.io).” [Online]. Available: https://fd.io/. [Accessed: 09-Feb-2017].
[42] “fd.io - What is VPP?” [Online]. Available: https://wiki.fd.io/view/VPP/What_is_VPP%3F. [Accessed: 09-Feb-2017].
[43] “fd.io - NSH SFC Project.” [Online]. Available: https://wiki.fd.io/view/NSH_SFC. [Accessed: 09-Feb-2017].
[44] OPNFV, “OPNFV Technical Overview.” [Online]. Available: https://www.opnfv.org/software/technical-overview. [Accessed: 10-Feb-2017].
[45] “SONATA NFV.” [Online]. Available: http://sonata-nfv.eu/content/about-sonata. [Accessed: 11-Feb-2017].
[46] “5G-PPP - The 5G Infrastructure Public Private Partnership.” [Online]. Available: https://5g-ppp.eu/. [Accessed: 11-Feb-2017].
[47] “ECOMP (Enhanced Control, Orchestration, Management & Policy) Architecture White Paper.”
[48] Yyang13, “Open vSwitch NSH patches.” .

[49] “OpenStack Releases: OpenStack Releases.” [Online]. Available: https://releases.openstack.org/. [Accessed: 14-Feb-2017].
[50] Ericsson, “OpenStack as the API framework for NFV,” 2015. .
[51] “OpenStack Foundatio Board of Directors.” [Online]. Available: https://www.openstack.org/foundation/board-of-directors/. [Accessed: 14-Feb-2017].
[52] O. Foundation, “Accelerating NFV Delivery with OpenStack.” [Online]. Available: https://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf. [Accessed: 09-Feb-2017].
[53] “Heat - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Heat. [Accessed: 12-Feb-2017].
[54] “Neutron’s developer documentation! — neutron 10.0.0.0rc2.dev43 documentation.” [Online]. Available: http://docs.openstack.org/developer/neutron/. [Accessed: 12-Feb-2017].
[55] “Tacker - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Tacker. [Accessed: 12-Feb-2017].
[56] “Kingbird - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Kingbird. [Accessed: 12-Feb-2017].
[57] “Tricircle - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Tricircle. [Accessed: 12-Feb-2017].
[58] A. Lemke, “5 areas OpenStack needs help to support NFV | Insight | Nokia.” [Online]. Available: https://insight.nokia.com/5-areas-openstack-needs-help-support-nfv. [Accessed: 10-Feb-2017].
[59] “OpenStack Docs: Scheduling.” [Online]. Available: http://docs.openstack.org/mitaka/config-reference/compute/scheduler.html. [Accessed: 12-Feb-2017].
[60] A. Otto, M. Ptl, D. Architect, R. C. Peters, and B. E. Whitaker, “Exploring Opportunities: Containers and OpenStack,” 2015. [Online]. Available: www.openstack.org. [Accessed: 10-Feb-2017].
[61] “Ansible is Simple IT Automation.” [Online]. Available: https://www.ansible.com/. [Accessed: 12-Feb-2017].
[62] “Puppet - The shortest path to better software.” [Online]. Available: https://puppet.com/. [Accessed: 12-Feb-2017].
[63] “ETSI VNFFG integration into Tacker NFVO : Blueprints : tacker.” [Online]. Available: https://blueprints.launchpad.net/tacker/+spec/tacker-vnffg. [Accessed: 12-Feb-2017].
[64] “Fuel - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Fuel. [Accessed: 12-Feb-2017].
[65] “Cobbler - Linux install and update server.” [Online]. Available: http://cobbler.github.io/. [Accessed: 12-Feb-2017].
[66] “Marionette Collective — Documentation — Puppet.” [Online]. Available: https://docs.puppet.com/mcollective/. [Accessed: 12-Feb-2017].
[67] “OpenDaylight Boron.” [Online]. Available: https://www.opendaylight.org/odlboron. [Accessed: 10-Feb-2017].
[68] I. Yamahata, “Opendaylight Summit - NetVirt Basic Tutorial.” [Online]. Available: http://schd.ws/hosted_files/opendaylightsummit2016/c9/ODL Summit 2016 NetVirt Basic Tutorial %282%29.pdf. [Accessed: 10-Feb-2017].
[69] “OpenJDK.” [Online]. Available: http://openjdk.java.net/. [Accessed: 12-Feb-2017].
[70] “Apache Karaf.” [Online]. Available: http://karaf.apache.org/. [Accessed: 12-Feb-2017].
[71] B. Pfaff and B. Davie, “The Open vSwitch Database Management Protocol,” 2013. [Online]. Available: https://tools.ietf.org/html/rfc7047. [Accessed: 14-Feb-2017].
[72] Ericsson, “Open vSwitch gets massive performance increase,” *Blog - The journey to NFV*. [Online]. Available: https://www.ericsson.com/spotlight/cloud/blog/2016/11/07/open-vswitch-gets-massive-performance-increase/. [Accessed: 18-Jan-2017].
[73] “Apex - OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/apex/Apex. [Accessed: 12-Feb-2017].
[74] “TripleO quickstart — RDO.” [Online]. Available: https://www.rdoproject.org/tripleo/. [Accessed: 12-Feb-2017].
[75] “RDO.” [Online]. Available: https://www.rdoproject.org/. [Accessed: 12-Feb-2017].
[76] “Compass4NFV - OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/compass4nfv/Compass4nfv. [Accessed: 12-Feb-2017].

[77] “Compass - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Compass. [Accessed: 12-Feb-2017].
[78] “Mirantis OpenStack.” [Online]. Available: https://www.mirantis.com/software/openstack/. [Accessed: 12-Feb-2017].
[79] “JOID - OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/joid/JOID+Home. [Accessed: 12-Feb-2017].
[80] “Canonical - Juju.” [Online]. Available: https://www.ubuntu.com/cloud/juju. [Accessed: 11-Feb-2017].
[81] “Canonical Metal as a Service.” [Online]. Available: https://maas.io/. [Accessed: 11-Feb-2017].
[82] “opnfv/functest - Docker image.” [Online]. Available: https://hub.docker.com/r/opnfv/functest/. [Accessed: 14-Feb-2017].
[83] “OPNFV Colorado Functest release notes.” [Online]. Available: http://artifacts.opnfv.org/functest/colorado/docs/release-notes/index.html. [Accessed: 14-Feb-2017].
[84] “Mirantis OpenStack: Fuel logical networks.” [Online]. Available: https://docs.mirantis.com/openstack/fuel/fuel-9.1/mos-planning-guide/network/logical_networks.html. [Accessed: 12-Feb-2017].
[85] “OPNFV Gerrit Fuel Master GIT repository.” [Online]. Available: https://gerrit.opnfv.org/gerrit/gitweb?p=fuel.git;a=tree. [Accessed: 10-Feb-2017].
[86] “OPNFV Platform CI [Jenkins].” [Online]. Available: https://build.opnfv.org/ci/. [Accessed: 14-Feb-2017].
[87] “libvirt: The virtualization API.” [Online]. Available: https://libvirt.org/. [Accessed: 12-Feb-2017].
[88] “NetVirt - OpenDaylight Project.” [Online]. Available: https://wiki.opendaylight.org/view/NetVirt. [Accessed: 12-Feb-2017].
[89] “Group Based Policy (GBP) - OpenDaylight Project.” [Online]. Available: https://wiki.opendaylight.org/view/Group_Based_Policy_(GBP). [Accessed: 12-Feb-2017].
[90] Y. Yang, “Fix VxLAN Issue in SFC Integration by Using Eth+NSH and VxLAN-gpe+NSH Hybrid Mode.”
[91] “OPNFV SFC Architecture.” [Online]. Available: http://artifacts.opnfv.org/sfc/colorado/docs/design/architecture.html. [Accessed: 17-Feb-2017].
[92] “Bug #1521323 ‘Increase default heat stack configurations to acco...’ : Bugs : tacker.” [Online]. Available: https://bugs.launchpad.net/tacker/+bug/1521323. [Accessed: 15-Feb-2017].
[93] “Functest: Opnfv Functional Testing- OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/functest/Opnfv+Functional+Testing. [Accessed: 12-Feb-2017].
[94] “OPNFV Pharos specification.” [Online]. Available: http://artifacts.opnfv.org/pharos/colorado/docs/specification/index.html. [Accessed: 17-Feb-2017].
[95] “[opnfv-tech-discuss] [SFC] Using compressed sf_nsh_colorado.qcow2 VM image.” [Online]. Available: https://lists.opnfv.org/pipermail/opnfv-tech-discuss/2017-January/014803.html. [Accessed: 17-Feb-2017].
[96] B. Walsh, “New Greenpeace Report Shows the Environmental Impact of the Internet | Time.com,” *Time Science*, 2014. [Online]. Available: http://time.com/46777/your-data-is-dirty-the-carbon-price-of-cloud-computing/. [Accessed: 13-Feb-2017].
[97] J. Kozlowicz, “8 Ways Data Center Environmental Impact Goes Beyond Emissions,” *Green House Data Blog*, 2015. [Online]. Available: https://www.greenhousedata.com/blog/data-center-environmental-impact-goes-beyond-emissions. [Accessed: 13-Feb-2017].
[98] “Amdocs Joins Forces with Linux Foundation to Accelerate OpenECOMP Adoption in Open Source.” [Online]. Available: http://www.amdocs.com/news/pages/amdocs-joins-forces-with-linux-foundation-to-accelerate-openecomp-adoption-in-open-source.aspx. [Accessed: 13-Feb-2017].
[99] “Project Clearwater.” [Online]. Available: http://www.projectclearwater.org/. [Accessed: 13-Feb-2017].
[100] “OpenIMS – The Open Source IMS Core Project.” [Online]. Available:

http://www.openimscore.org/. [Accessed: 13-Feb-2017].
[101] “CORD (Central Office Re-architected as a Datacenter): the killer app for SDN & NFV.” [Online]. Available: http://opencord.org/. [Accessed: 08-Feb-2017].
[102] “OpenAirInterface | 5G software alliance for democratising wireless innovation.” [Online]. Available: http://www.openairinterface.org/. [Accessed: 13-Feb-2017].
[103] M. Ersue, “ETSI NFV Management and Orchestration Update.” [Online]. Available: https://www.ietf.org/proceedings/89/slides/slides-89-opsawg-7.pdf. [Accessed: 03-Feb-2017].
[104] M. Bjorklund, “RFC7950 - The YANG 1.1 Data Modeling Language,” 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7950.txt.
[105] “OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC | OASIS.” [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca. [Accessed: 03-Feb-2017].
[106] “OASIS - Organization for the Advancement of Structured Information Standards.” [Online]. Available: https://www.oasis-open.org/. [Accessed: 09-Feb-2017].
[107] “Topology and Orchestration Specification for Cloud Applications Version 1.0.” [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html. [Accessed: 03-Feb-2017].
[108] “TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0.” [Online]. Available: https://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html. [Accessed: 03-Feb-2017].
[109] Heavy Reading, “NFV MANO: What’s wrong and how to fix it,” 2015. [Online]. Available: http://getcloudify.org/brochures/Heavy Reading NFV MANO Cloudify Snapshot.pdf. [Accessed: 09-Feb-2017].
[110] “Open Source NFV Part Four: Open Source MANO.” [Online]. Available: http://thenewstack.io/opensource-nfv-part-4-opensource-mano/. [Accessed: 09-Feb-2017].
[111] “OpenMANO.” [Online]. Available: http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab/openmano. [Accessed: 11-Feb-2017].
[112] “RIFT.io RIFT.ware.” [Online]. Available: https://riftio.com/riftware/. [Accessed: 09-Feb-2017].
[113] “Telefónica NFV Reference Lab.” [Online]. Available: http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab. [Accessed: 11-Feb-2017].
[114] “OpenVIM installation (Release One).” [Online]. Available: https://osm.etsi.org/wikipub/index.php/OpenVIM_installation_(Release_One). [Accessed: 11-Feb-2017].
[115] “Canonical - Jujucharms.” [Online]. Available: https://jujucharms.com/. [Accessed: 11-Feb-2017].
[116] “Open-O - Open Orchestrator.” [Online]. Available: https://www.open-o.org/. [Accessed: 09-Feb-2017].
[117] “Open Baton: an open source reference implementation of the ETSI Network Function Virtualization MANO specification.” [Online]. Available: http://openbaton.github.io/. [Accessed: 09-Feb-2017].
[118] “Cloudify - Network Function Virtualization Orchestration.” [Online]. Available: http://getcloudify.org/network-function-virtualization-vnf-nfv-orchestration-sdn-platform.html. [Accessed: 09-Feb-2017].
[119] “Intel Network Builders - RIFT.io.” [Online]. Available: https://networkbuilders.intel.com/ecosystem/rift.io. [Accessed: 09-Feb-2017].
[120] ETSI, “OSM - Open Source MANO.” [Online]. Available: https://osm.etsi.org/. [Accessed: 09-Feb-2017].
[121] “OSM Release ONE.” [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Release_ONE. [Accessed: 11-Feb-2017].
[122] A. Hoban and C. Buerger, “Open Source MANO technical overview,” 2016.
[123] “VMware vCloud Director.” [Online]. Available: https://www.vmware.com/products/vcloud-director.html. [Accessed: 11-Feb-2017].
[124] “Floodlight OpenFlow Controller.” [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed: 11-Feb-2017].
[125] “Nokia CloudBand.” [Online]. Available: https://networks.nokia.com/solutions/cloudband. [Accessed: 09-Feb-2017].
[126] “Ericsson Cloud Manager.” [Online]. Available:

http://www.ericsson.com/hyperscale/cloud-infrastructure/cloud-manager. [Accessed: 09-Feb-2017].
[127] “Oracle Communications Application Orchestrator.” [Online]. Available: https://www.oracle.com/industries/communications/service-providers/products/application-orchestrator/index.html. [Accessed: 09-Feb-2017].
[128] “HPE NFV Director.” .
[129] “Wind River Titanium Server - NFV Carrier Grade Server.” [Online]. Available: https://www.windriver.com/products/titanium-server/. [Accessed: 09-Feb-2017].
[130] “Blue Planet: a division of Ciena.” [Online]. Available: http://www.blueplanet.com/products. [Accessed: 09-Feb-2017].
[131] Amdocs, “NFV: It’s not just the network - it´s all about the service!,” 2016. .
[132] Juniper, “NFX250 Network Services Platform.” [Online]. Available: https://www.juniper.net/us/en/products-services/sdn/nfx250/. [Accessed: 02-Feb-2017].
[133] Juniper, “Understanding How MX Series Router Cloud CPE Services Virtualize Customer Premises Equipment (CPE) Services.” [Online]. Available: https://www.juniper.net/techpubs/en_US/junos13.3/topics/concept/ccpe-overview.html. [Accessed: 02-Feb-2017].
[134] Juniper, “vSRX Integrated Virtual Firewall.” [Online]. Available: http://www.juniper.net/uk/en/products-services/security/srx-series/vsrx/. [Accessed: 02-Feb-2017].
[135] Juniper, “vMX Virtual Router for Enterprise & Service Provider Networks.” [Online]. Available: http://www.juniper.net/uk/en/products-services/routing/mx-series/vmx/. [Accessed: 02-Feb-2017].
[136] T. Herbert, M. Gray, and C. Price, “NFV vSwitch Requirements,” *fd.dio*. [Online]. Available: https://wiki.fd.io/view/File:NFV_vSwitch_Requirements.pptx. [Accessed: 03-Feb-2017].
[137] J. Drake, E. Rosen, and J. Uttaro, “BGP Control Plane for NSH SFC - draft-mackie-bess-nsh-bgp-control-plane-00,” 2016. [Online]. Available: https://tools.ietf.org/html/draft-mackie-bess-nsh-bgp-control-plane-00.
[138] S. Dredge, “The Missing Link: Service Function Chaining and Its Relationship to NFV,” *Metaswitch blog*, 2015. [Online]. Available: http://www.metaswitch.com/the-switch/the-missing-link-service-function-chaining-and-its-relationship-to-nfv. [Accessed: 03-Feb-2017].
[139] E. Wang, K. Leung, and A. Ossipov, “Network Service Header (NSH) Context Header Allocation (Network Security),” *draft-wang-sfc-nsh-ns-allocation-02*, 2016. [Online]. Available: https://www.ietf.org/id/draft-wang-sfc-nsh-ns-allocation-02.txt.
[140] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An Updated Performance Comparison of Virtual Machines and Linux Containers An Updated Performance Comparison of Virtual Machines and Linux Containers,” 2014. [Online]. Available: http://domino.watson.ibm.com/library/CyberDig.nsf/home. [Accessed: 09-Feb-2017].
[141] “QEMU.” [Online]. Available: http://www.qemu-project.org/. [Accessed: 09-Feb-2017].
[142] “XenServer.” [Online]. Available: http://xenserver.org/. [Accessed: 09-Feb-2017].
[143] “Microservices Architecture in the Telco Cloud.” [Online]. Available: https://www.sdxcentral.com/nfv/definitions/microservices-architecture-telco-cloud/. [Accessed: 09-Feb-2017].
[144] Nuage Networks, “Nuage Networks VSP: Virtual Networking and SDN Infrastructure for Containers.” [Online]. Available: http://www.nuagenetworks.net/blog/containers/. [Accessed: 09-Feb-2017].
[145] “Kubernetes - Production-Grade Container Orchestration.” [Online]. Available: https://kubernetes.io/. [Accessed: 09-Feb-2017].
[146] “Mesosphere - Cloud-native Technology, Hybrid Cloud Freedom.” [Online]. Available: https://mesosphere.com/. [Accessed: 09-Feb-2017].
[147] “MirageOS.” [Online]. Available: https://mirage.io/. [Accessed: 09-Feb-2017].
[148] “Docker - Containerization platform.” [Online]. Available: https://www.docker.com/. [Accessed: 09-Feb-2017].
[149] Ericsson, “Unikernels meet NFV,” *Ericsson Research Blog*. [Online]. Available: https://www.ericsson.com/research-blog/sdn/unikernels-meet-nfv/. [Accessed: 09-Feb-2017].
[150] Unikernel.org, “NFV Platforms with MirageOS Unikernels.” [Online]. Available: http://unikernel.org/blog/2016/unikernel-nfv-platform. [Accessed: 09-Feb-2017].

[151] “Iron.io - Serverless Multi-cloud for Enteprise.” [Online]. Available: https://www.iron.io/. [Accessed: 09-Feb-2017].
[152] “Mirantis + Iron.io: Bringing Serverless Computing to OpenStack.” [Online]. Available: https://www.mirantis.com/blog/mirantis-iron-io-bringing-serverless-computing-to-openstack-2/. [Accessed: 09-Feb-2017].
[153] “Iron.io IronFunctions.” [Online]. Available: http://open.iron.io/. [Accessed: 09-Feb-2017].
[154] “AWS Lambda | Product Details.” [Online]. Available: https://aws.amazon.com/lambda/details/. [Accessed: 09-Feb-2017].
[155] “Cloud Functions - Serverless Microservices | Google Cloud Platform.” [Online]. Available: https://cloud.google.com/functions/. [Accessed: 09-Feb-2017].
[156] “Memory Deep Dive: NUMA and Data Locality - frankdenneman.nl.” [Online]. Available: http://frankdenneman.nl/2015/02/27/memory-deep-dive-numa-data-locality/. [Accessed: 09-Feb-2017].
[157] Intel, “An Introduction to SR-IOV Technology,” 2011. .
[158] “Accelerating the NFV Data Plane: SR-IOV and DPDK… in my own words.” [Online]. Available: http://www.metaswitch.com/the-switch/accelerating-the-nfv-data-plane. [Accessed: 09-Feb-2017].
[159] “Ethernity Networks Launches Flow Processing Network Interface Cards (NICs) for SDN and NFV Acceleration.” [Online]. Available: http://www.ethernitynet.com/news/ethernity-networks-launches-flow-processing-network-interface-cards-nics-for-sdn-and-nfv-acceleration/. [Accessed: 09-Feb-2017].
[160] “Napatech NT200B01 NFV NIC.” [Online]. Available: http://www.napatech.com/resources/nt200b01-nfv-nic. [Accessed: 09-Feb-2017].
[161] “Mirantis Open NFV Platform.” [Online]. Available: https://www.mirantis.com/solutions/network-functions-virtualization-nfv/open-nfv-platform/. [Accessed: 09-Feb-2017].
[162] “Mirantis OpenStack NFVI Deployment Guide.” [Online]. Available: https://content.mirantis.com/MOS-7-NFVI-Whitepaper-Landing-Page.html. [Accessed: 09-Feb-2017].
[163] “OpenStack Tacker Enhanced Placement Awareness Usage Guide.” [Online]. Available: http://docs.openstack.org/developer/tacker/devref/enhanced_placement_awareness_usage_guide.html. [Accessed: 09-Feb-2017].
[164] “OpenStack Enhanced Platform Awarenes.”
[165] “Yardstick: Inftrastructure verification - OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/yardstick/Yardstick. [Accessed: 12-Feb-2017].
[166] “VSPerf - OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/vsperf/VSperf+Home. [Accessed: 12-Feb-2017].
[167] “Project Bottlenecks - OPNFV Wiki.” [Online]. Available: https://wiki.opnfv.org/display/PROJ/Project+Proposals+Bottlenecks. [Accessed: 12-Feb-2017].
[168] P. Capdevila, “fuel-deploy git repository,” 2017. [Online]. Available: https://github.com/pcapdevila/fuel-deploy.

ACRONYMS

ADC	Application Delivery Controller
API	Application Programming Interface
APN	Access Point Name
BGP	Border Gateway Protocol
BNG	Broadband Network Gateway
BSS	Business Support System
CAPEX	Capital Expenditures
CLI	Command Line Interface
COTS	Commercial Off-the-Shelf
CORD	Central Office Re-architected as a Datacentre
CPE	Customer Premises Equipment
CSP	Communication Service Provider
DC	Data Centre
DCI	Data Centre Interconnect
DHCP	Dynamic Host Configuration Protocol
EMS	Element Management System
EPC	Evolved Packet Core
EPS	Evolved Packet System
ETSI	European Telecommunications Standards Institute
FW	Firewall
GRE	Generic Routing Encapsulation
HA	High Availability
IOT	Internet of Things
IMS	IP Multimedia Subsystem
IPMI	Intelligent Platform Management Interface
L3VPN	Layer 3 Virtual Private Network
LAN	Local Area Network
LISP	Locator/ID Separation Protocol
LSO	Lifecycle Service Orchestration
LTE	Long-Term Evolution
MANO	Management and Orchestration
MPLS	Multi-Protocol Label Switching
NAT	Network Address Translation
NF	Network Function
NFFG	NF Forwarding Graph (ETSI Terminology for chaining)
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSH	Network Services Header (Service chaining encapsulation)
NVGRE	Network Virtualization Using Generic Routing Encapsulation
NVO	Network Virtualization Overlay
OASIS	Organization for the Advancement of Structured Information Standards
ODL	OpenDaylight SDN Controller
OLT	Optical Line Terminal

ONOS	Open Network Operating System
ONU	Optical Network Unit
OVS	Open vSwitch
OVSDB	Open vSwitch Database
OPEX	Operating Expense
OSS	Open Source Software
OSS	Operations Support System
POC	Proof of Concept
POD	Point of Delivery
PNF	Physical Network Function
PNFD	Physical Network Function Descriptor
PGW	Packet Data Network Gateway
PXE	Preboot Execution Environment
RAN	Radio Access Network
REST	Representational State Transfer
ROI	Return of Investment
RSP	Rendered Service Path
SDN	Software-Defined Networking
SDO	Standards Developing Organizations
SF	Service Function
SFC	Service Function Chaining (IETF Terminology)
SFF	Service Function Forwarder
SFP	Service Function Path
SMBs	Small and Medium Business
TLV	Type Length Value
TOSCA	Topology and Orchestration Specification for Cloud Applications
UI	User Interface
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor
VNF	Virtual Network Function
VNFC	VNF Component
VNFD	Virtual Network Function Descriptor.
VNFM	Virtual Network Function Manager
VNFFG	Virtual Network Function Forwarding Graph
VNFFGD	Virtual Network Function Forwarding Graph Descriptor
VIM	Virtual Infrastructure Manager
VRF	Virtual Routing and Forwarding
VXLAN	Virtual Extensible Local Area Network
VXLAN-GPE	Virtual Extensible LAN Generic Protocol Extension
WAN	Wide Area Network
YAML	Yet Another Markup Language
YANG	Yet Another Next Generation
XML	Extensible Markup Language

ANNEX A. NFV MANAGEMENT AND ORCHESTRATION

A.1. Standardization of MANO

The Management and Orchestration (MANO) of VNFs is a part of the ETSI specification which lacked initial attention in the standardization efforts leading to the development of a variety of commercial orchestrators covered in following sections. Standard data models are required to abstract end-to-end services and achieve seamless NFV deployment and operation over diverse infrastructure.

In the MANO information model, several kind of descriptors are used to abstract the behavior of a complete NFV solution. Descriptors are organized in catalogs:

- Service catalog:
  - Network Service Descriptors
  - VNFFG/SFC Descriptors
- Element catalog:
  - VNF Descriptors
  - PNF Descriptors
  - Virtual Link Descriptors

The descriptors are templates that try to capture all the essential features of each modeled entity in a way that the orchestrator can use them to translate requests from the OSS layer to instantiate the resulting resources into a real-life NFV-based service. Figure A.1 shows the MANO descriptor files.

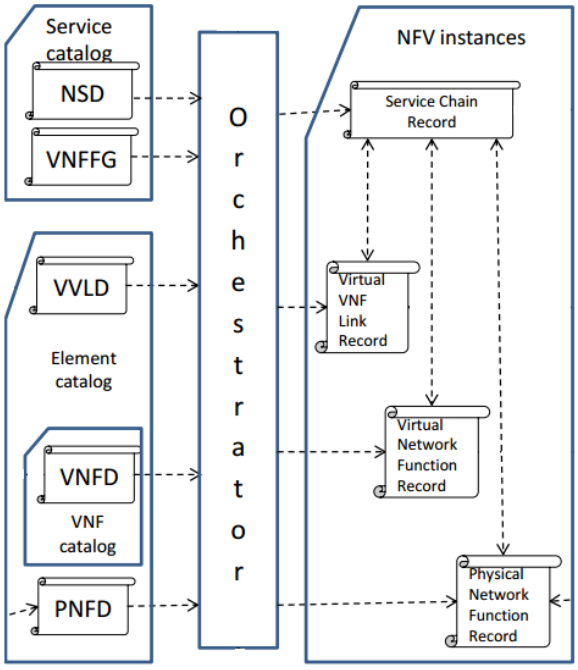

Figure A.1: Overview of MANO Descriptor Files [103]

Instead of implementing an NFV specific modeling language, two preexisting languages play a complementary role in NFV MANO:

- YANG [104] is the IETF data modeling language for the IETF NETCONF (network configuration protocol), which is a management protocol for network devices aimed to be the successor of SNMP (simple network management protocol). NETCONF is mainly used for network devices but could eventually act as element management system (EMS) in the NFV stack but is out-of-scope in the thesis.

- TOSCA (Topology and Orchestration Specification for Cloud Applications) [105] is an OASIS [106] standard for Lifecycle Service Orchestration (LSO). TOSCA is a language originally used to model cloud services but, due to its extensibility, it also supports NFV use cases. TOSCA uses templates to control workflow and describe the relationships and dependencies in an NFV infrastructure.

In the next section, the main features of the TOSCA NFV application are described to gain an understanding of the fundamentals of NFV MANO. Some basic templates will be used in the practical scenarios implementation later described in this document.

A.1.1. TOSCA

TOSCA is an OASIS (Organization for the Advancement of Structured Information Standards) standard originally intended for web services but useful also for NFV. The TOSCA specification introduces the Service Template concept to specify the structure, properties and behaviors which define the topology and orchestration of a platform. Several use cases can be fulfilled. For example, service composition, service topology modeling, inter-domain service templates and virtual images templates.

The TOSCA language defines a grammar to create Topology Templates and Plans focusing the description of services. The life-cycle management aspects are specified in the management plan models of service instances. TOSCA uses XML (eXtensible Markup Language), but simplified profiles of the TOSCA specification also exist in YAML (Yet Another Markup Language) to simplify the human writing of TOSCA service templates.

The TOSCA specification defines:

- **Service template** provides values for the build plan of the various nodes.
- **Topology template** puts together Node and Relationship Templates defining a service as a directed graph.
- **Nodes** are components of a service. A Node Type defines:
    - **Node properties** providing the definitions of the Node Template.
    - **Interfaces** Meaning the operations available to manage the node.
- **Relationship templates** specify communication between nodes.
- **Plans** describe the service instance life-cycle management processes.

Figure A.2. show the service template schema and the relations between the node, relationship and topology elements.

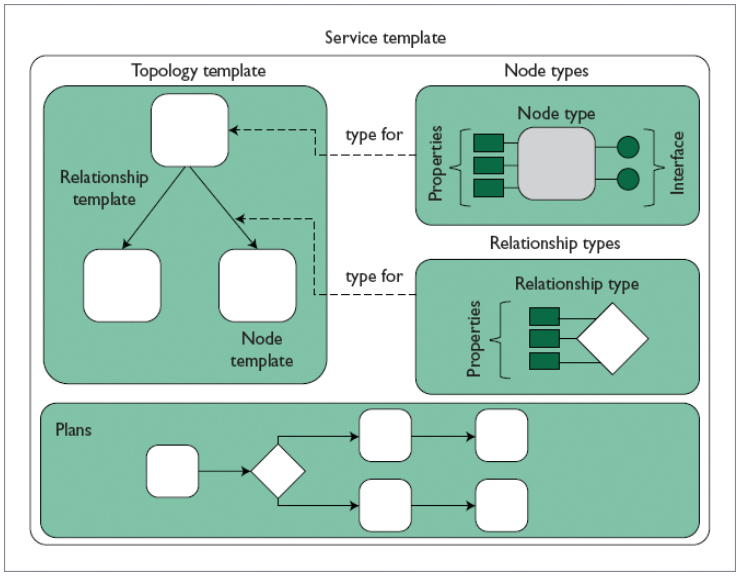

Figure A.2: TOSCA Service Template schema [107]

A.1.2. TOSCA NFV Profile

TOSCA use cases have been adapted to the NFV architecture. The TOSCA data model top level entity is the service template. The Service Template lower elements are the different node templates. Similarly, In NFV MANO, the Network Service Descriptor (NSD) is the top level element, and below the NSD there are:

- VNFD: Virtual Network Function Descriptor.
- VNFFGD: Virtual Network Function Forwarding Graph Descriptor.
- VLD: Virtual Link Descriptor.
- PNFD: Physical Network Function Descriptor.

The mapping between TOSCA and NFV takes the following approach as shown in Figure A.3:

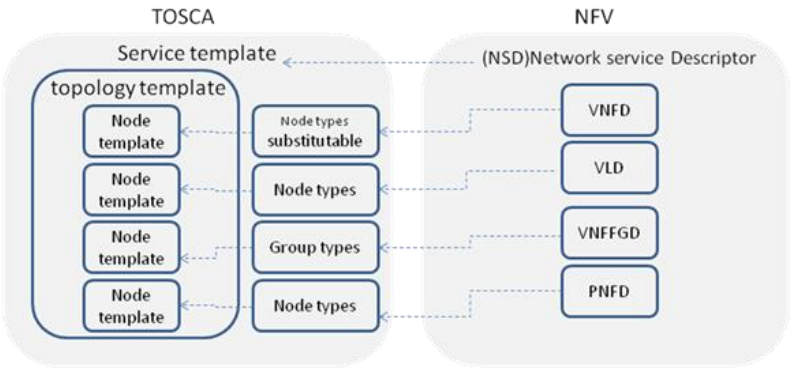

Figure A.3: Mapping between TOSCA and NFV descriptors [108]

A.2. Management software for NFV

NFV MANO is one part the ETSI specification which was initially left open to vendor interpretation leading to a profusion of incompatible and non-standard solutions. This is seen by [109] as a stopper for a wider NFV adoption. It was not until ETSI NFV Release 2 when the interfaces between each MANO module (NFVO, VNFM and VIM) started to be defined.

[110] discusses if besides of using a VIM, almost always based in OpenStack, other MANO layers should also be implemented within OpenStack, which is not the chosen option in most of the cases. Another point in current implementations is that they are not compliant to the 3 MANO layers due to the confusion splitting resource management between the VIM and the NFVO.

The introduction of the TOSCA standard will settle the existing orchestrators to more interoperable implementations and integrations between open source projects may take place as it has already occurred between OpenMANO [111] and Rift.ware [112] merging code into Open Source MANO [16].

OpenMANO was the NFV management reference implementation from the Telefonica NFV Reference Lab [113]. OpenMANO implemented the three MANO layers: NFVO, VNFM and VIM. OpenVIM [114] was just a simple virtual infrastructure manager for demo purposes. But as it now forms part of Open Source MANO it has not been taking into consideration.

A.2.1. Open source MANO implementations

The following list presents a comprehensive selection of open source MANO offerings:

- Canonical's Juju [80] is an open source modeling tool for service oriented and application oriented deployments. In the context of NFV MANO, Juju implements a generic VNF manager (VNFM), so it requires an external VNF Orchestrator. It does not employ TOSCA but implements its own language. Current Juju version is 2.0.2 and it has the following features:
    - Support for multi-cloud and multi-DC modeling, deploying and managing of VNF components and services.
    - Service descriptors take the form of Charms [115] in Juju. Charms are organized in a public marketplace acting as a service catalog.
    - Charms can be grouped in bundles, which become reusable forms of network scenario definitions.
    - Can provide VNFM services across both OpenStack and MAAS[13].

- Open-O [116] is a recent project competing with OSM under the Linux Foundation umbrella. Its aim is to provide orchestration for NFV, SDN and legacy networks. Release 1.0, codenamed Sun, features 5 projects:

[13] Metal-as-a-Service is Canonical's bare-metal provisioning and deployment platform supporting of a range of operating systems and hardware platforms to realize physical network functions.

    - Global Service Orchestrator (GS-O) provides end-to-end services.
    - SDN Orchestrator (SDN-O) provides connectivity services across SDN and legacy networks.
    - NFV Orchestrator (NFV-O) provides an ETSI-compliant orchestrator.
    - Common Services provide a Microservice bus, HA, Driver Manager, Log, Authentication and Protocol Stack to the remaining projects.
    - Common TOSCA project provides a parser, an execution engine and a model designer.

- Open Baton [117] is an interoperable and extensible ETSI and TOSCA-compliant NFVO. With three releases already it provides an extensive list of features:
    - An NFVO strictly adhering to the ETSI MANO specification.
    - Generic and Juju-based VNF Managers based on VNF descriptors.
    - A VNF autoscaling engine.
    - A runtime management system of faults at any layer**.**
    - A VIM plugin architecture independent on the orchestration logic.
    - A powerful event engine based on a pub/sub mechanism for the dispatching of lifecycle events execution.
    - Zabbix monitoring integration.
    - Libraries to build or customize the VNFM.

- Cloudify [118] is an orchestration-centric suite for cloud orchestration focusing on optimizing NFV orchestration and management. It is also a mature open source framework (Version 3.4.1). The open source version includes:
    - Cloudify Command-Line Interface (CLI)
    - Cloudify manager provides added a web interface, REST APIs, security features, blueprint catalogs, multiple blueprint deployments, concurrent workflow executions, etc.
    - TOSCA standards-based orchestration.
    - Topology-driven VNF lifecycle management and monitoring.
    - Support multiple application stacks: OpenStack, bare metal, and virtual appliance support (enabling portability to any cloud and hybrid cloud models).
    - Support for containerized workloads.
    - Designed for federated deployments.
    - Support for legacy networks.
    - Auto-healing and auto-scaling policy engine.

A.2.2. Case study: Tacker

Three main building blocks conform the Tacker architecture:

- VNF catalog contains descriptors for VNFs, NSs and SF chains.
- VNFM manages the VNF placement and life-cycle.
- NFVO deploys services based on templates using VNFs.

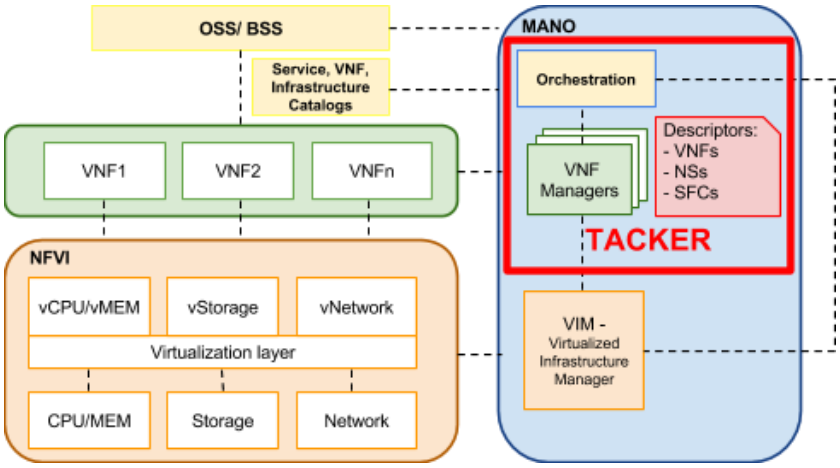

Figure A.4: OpenStack Tacker positioning in the NFV architecture

Tacker parses TOSCA templates and translates to the VIM orchestrator API, in this case OpenStack Heat. Other VIMs are supported and multiple VIMs can be orchestrated from a single Heat instance to support multi-site deployments. The VNFM takes care of policy-based VNF placement. VNF configuration, health monitoring and auto-healing according to the VNF descriptor policy. It also performs resource check for efficient VNF allocation. Figure 3.5 shows Tacker and Heat orchestrating several virtual infrastructure managers.

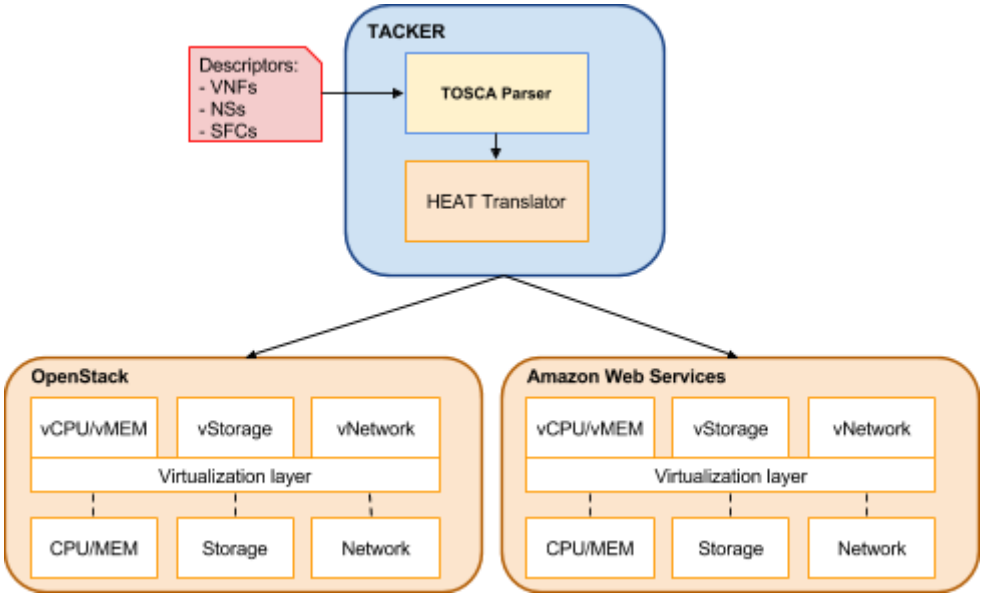

Figure A.5: OpenStack Tacker high-level overview

A.2.3. Case study: RIFT.ware

RIFT.ware [112] is an open source model-driven ETSI-compliant orchestration and automation solution. Its latest version is 4.3.3.1 at the time of writing. RIFT.ware offers a holistic approach to the NFV ecosystem where VNFs can be created, onboarded, deployed and managed at Webscale[14] on multiple Cloud Infrastructures. Figure 2.10 shows the Rift.ware architecture:

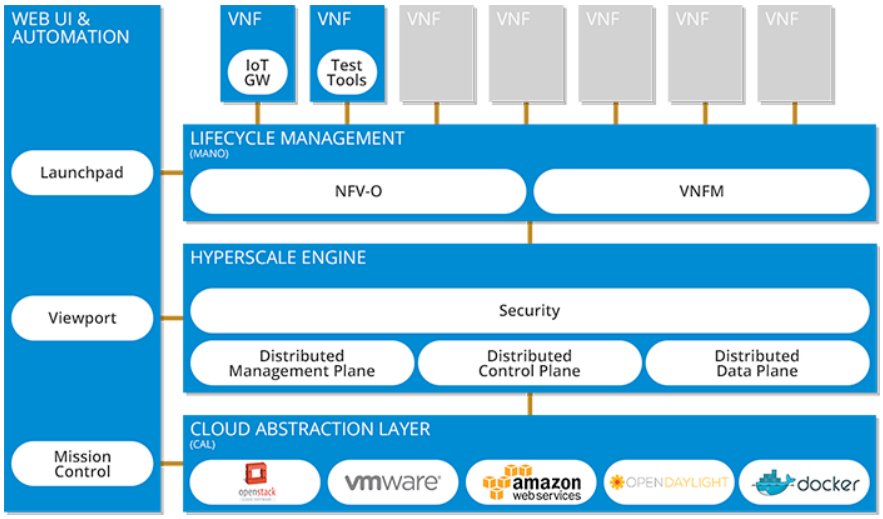

Figure A.6: Rift.ware MANO framework architecture [119]

The RIFT.ware architecture is compound of 4 main blocks:

- Lifecycle management is an ETSI MANO VNFM/NFVO with:
  - Enhanced Platform Awareness (EPA) capabilities.
  - Automated end-to-end service lifecycle management.
  - VNF and NS scaling capabilities from 1 to hundreds.
  - Carrier and enterprise-grade levels of availability.

- Hyperscale engine is a set of libraries and APIs enabling massive horizontal elasticity which offers:
  - Configuration management across multiple VMs and multi-VM VNF grouping as a single entity.
  - A programmable distributed control and forwarding planes which can present a multi-VM service with a single IP address.
  - High performance data paths such as OVS, DPDK, and SR-IOV.
  - Trusted platform for securing management, control, and data planes.

- Cloud Abstraction Layer decouples underlying cloud infrastructure and network so VNFs can be created, managed and migrated between different cloud systems/data-centers.

[14] Architectural approach used to deliver the capabilities of large cloud service achieving high levels of agility and scalability leveraging distributed software abstractions.

- WebUI and automation offer development and management interfaces to manage and monitor pools arranged in environments and organizations.

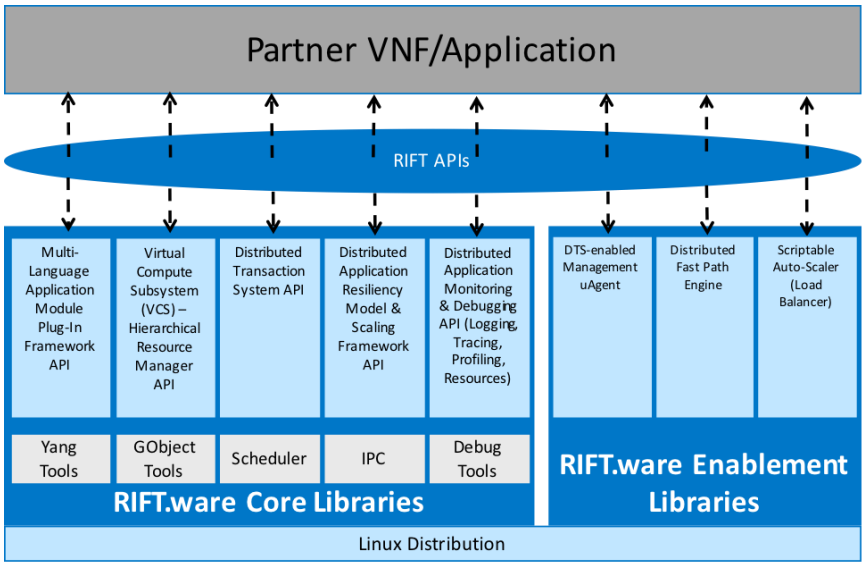

Figure A.7: Rift.ware Hyperscale engine APIs

A.2.4. Case study: Open Source MANO

Open Source MANO (OSM) [120] is a project hosted by ETSI implementing an NFV management framework. It has just seen its first release (OSM Release ONE) [121] earlier than the expected in 6-month initial cycle. The project aims to deliver a VIM-independent full MANO stack supporting open information models and suitable for all kinds of VNFs. Figure 2.12 shows the OSM architecture:

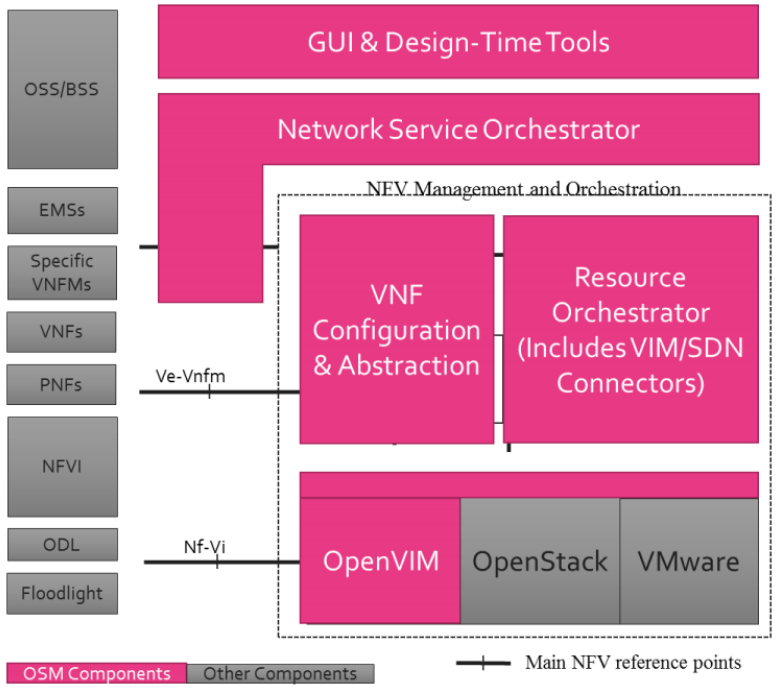

Figure A.8: Open Source MANO framework architecture [122]

The 5 main components of OSM are:

- GUI and design tools, which allow:
  - Easy VNF onboarding and packaging.
  - Simple service modeling.
  - Access to all features either through GUI, CLI or REST APIs.
- Network Service Orchestrator component, which is taken from Rift.ware.
- VNF configuration and abstraction, which relies on external VNFM such as Juju and, thus, requires a data model translator to support multiple template definition formats.
- Resource Orchestrator (RO) is an abstraction layer providing a plug-in model to support different types of VIMs, such as:
  - OpenStack [34].
  - VMware vCloud Director [123].
  - OpenVIM [114].

  and SDN Controllers:
  - OpenDaylight [36].
  - Floodlight [124].
- OpenVIM, which provides a reference virtual infrastructure with EPA support for all-in-one installations.

OSM supports EPA-based resource allocation to permit high-performance VNF to be deployed on capable hosts. It also has Multi-Site for service delivery across multiple DCs. Other interesting features are easy install and upgrade procedures.

A.2.5. Proprietary MANO implementations

For completion, a selection of commercial MANO offerings is also listed:

- CloudBand [125] groups the Nokia portfolio for integrated end-to-end MANO, which includes:
  - Infrastructure Software: Multi-purpose NFVI and VIM.
  - Application Manager: VNF management and onboarding.
  - Network Director: Network service and resource orchestration.

- Cloud Manager [126] is Ericsson's Cloud Management System (CMS) which regarding MANO implements NFVO and VNFM.

- Oracle Communications Application Orchestrator [127] provides lifecycle management of both VNFs, PNFs or a mixture of both in Composite Network Functions (CNF).

- NFV Director [128] is the Hewlett Packard Enterprise ETSI compliant NFVO. It also includes an embedded VNFM and support for external VNFMs.

- NFV Carrier Grade Server is the MANO offering from Wind River integrated on the Titanium Server offering [129], which is a comprehensive portfolio around OpenStack NFV with carrier-grade capabilities.

- Blueplanet [130] is a multi-domain service orchestrator including NFVO, analytics and health prediction and support for ONOS SDN controller.

A.2.6. State of the art in MANO summary

After reviewing all the presented options, the VNF vendor ecosystem is found to be the weakest link in the NFV supply chain. As stated in [131], NFV is all about the service. So a diverse VNF offer is required to compose first-class services. Although commercial MANO tools have some partners which provide a set of viable options, the open source approach is the one expected to succeed as it is the only one removing commercial barriers between competing companies.

It is obvious that there is a big market fragmentation in MANO, which is an entry barrier preventing the looming of a potent VNF offer. In [20], NFV is stated to be unable to scale vertically at the increasing pace of traffic demand due to the hardware architectures in use. Then, MANO is the only place where horizontal scaling can be realized.

Such is the importance of MANO, that currently there is a race between different open source projects with very fancy web pages trying to gain early adopters. But a leader in this area is yet to be decided once the solutions mature. The first one that truly leverages the virtuous circle of service life-cycle fulfillment with the aid of data analytics continuous monitoring will prevail.

ANNEX B. NFV USE CASE EXAMPLE

B.1. Virtualization of the CPE

This example is based on Juniper Networks' actual portfolio. The CPE is the edge device with sits in the customer premises connecting to the provider over a Wide-Area Network (WAN) link. Each subscriber usually has a physical appliance implementing functions like local DHCP, NAT, firewall, etc. The CPE functions can be virtualized and brought to the cloud to simplify or eliminate the CPE as shown in figure B.1. Given the number of subscribers, the savings can be huge.

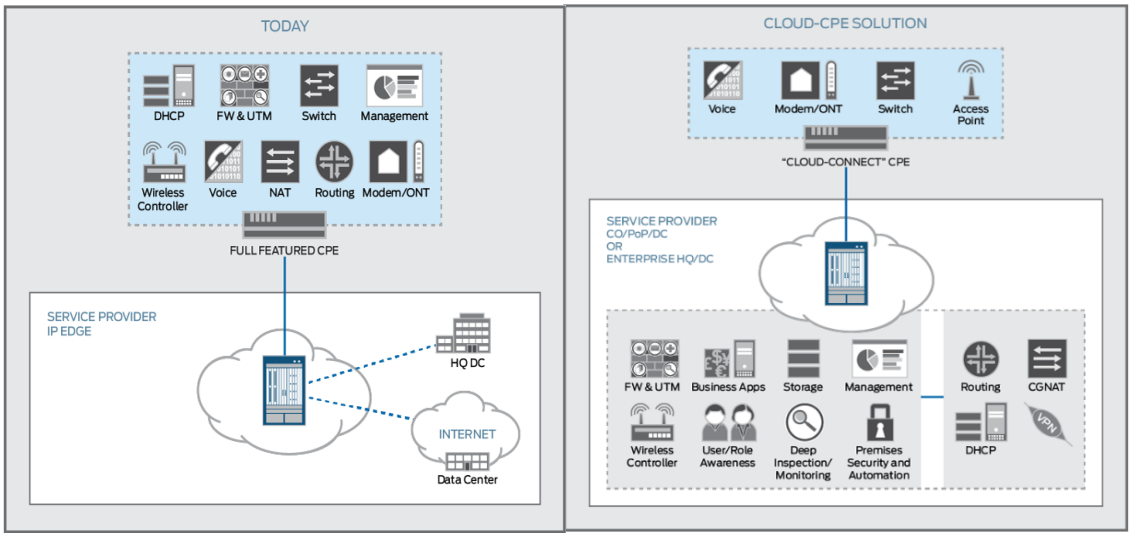

Figure B.1: Juniper Networks Cloud CPE solution [133]

Strictly speaking, this is network virtualization, not NFV, because it runs on the MX platform. This is an example of a vendor pre-NFV offering. Later, Juniper developed a truly NFV product based on the NFX250, which is a Juniper branded white-box network appliance following the COTS model. The vCPE helps reducing the number of CPE models to maintain and is orchestrated via the Contrail SDN controller. Figure B.2 shows the front chassis view of the NFX250.

Figure B.2: Juniper Networks NFX250 Network Services Platform [132]

This specific solution was designed for AT&T targeting small and medium business (SMBs). It consists of an orchestration framework based on the Contrail Virtual Network Controller [39]. The NFX250 supports up to 8 Juniper VNFs (basically vSRX [134] and vMX [135]) although it can also run Cisco or Brocade virtual appliances. It follows a decentralized model in application of distributed NFV and is the first commercially available vCPE offering.

ANNEX C. SFC ENCAPSULATION: NSH

C.1. Service Function Chains and Paths

A Service Function Chain (SFC) is a graph specifying the required SFs in the order that traffic must traverse them. Each node in the graph is a service function and can be part of zero or more SFCs. A given SF can appear one or more times in an SFC. SFs can be branching nodes in case of reclassification in one SF, leading to a new SFC.

SFCs may be unidirectional or bidirectional. An unidirectional SFC only specifies one traffic forwarding direction whereas bidirectional SFCs require symmetric path in both directions. Hybrid SFCs contain SFs which require symmetric or asymmetric treatment. SFCs can also contain cycles where traffic needs to traverse a SF more than once.

A Service Function Path (SFP) is a constrained SFC resulting from applying more granular policy to the SFC abstract requirements. Some SFPs specify exactly which SFF and SF packets must traverse, while others can be less specific on what sequence is to be used to execute the SFC. This final sequence is called the Rendered Service Path (RSP) as illustrated in Figure C.1.

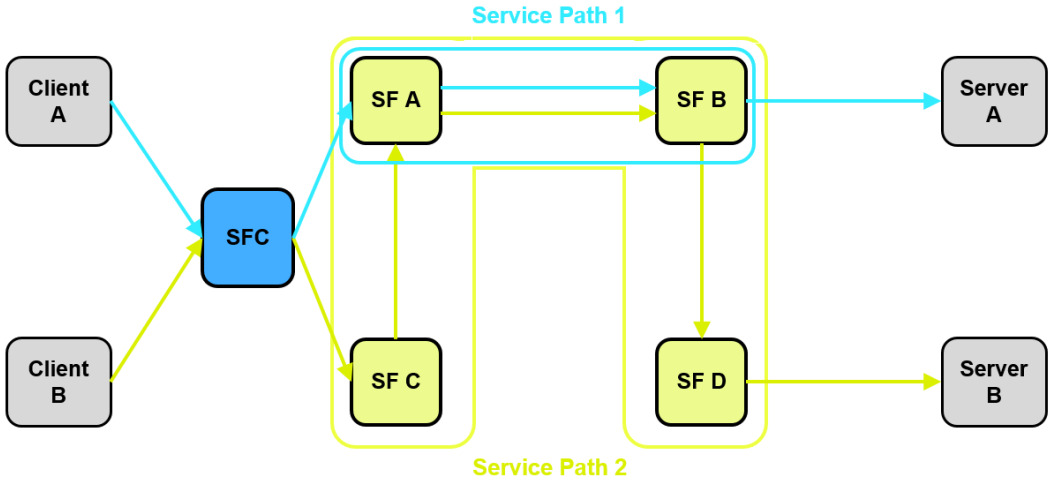

Figure C.1: SFC Service Path rendering [136]

C.2. SFC Control Plane

The SFC control plane is responsible for constructing SFPs, translating SFCs to forwarding paths, and propagating path information to participating nodes to achieve requisite forwarding behavior to construct the service overlay. The control plane manages and communicates SF capabilities, availability, and location in fashions suitable for the transport and SFC operations in use.

The control plane is also responsible for the creation of the context based on metadata so that the orchestrator can interpret them ant take action. The RFC does not specify if the control plane has to be distributed or centralized nor mandates which protocols it must use. There is a proposal for a BGP-based SFC control plane [137]. The SFC control plane should provide:

- A domain-wide view of all available SFs and their network locators.
- Policy to construct SFCs and associated SFPs.
- Static or dynamic selection of specific SFs for a given SFC.
- Provide SFC data-plane information to the SFFs.
- Provide metadata and usage information to classifiers.
- Provide policy information so the other SFC elements can interpret metadata.

C.3. NSH: the SFC encapsulation

Network Services Header (NSH) is the SFC encapsulation. It enables both SFP identification and exchange of metadata/context information between SFC-aware functions, such as SFFs, SFs and proxies. NSH carries an SFP identifier but it is not a transport encapsulation itself, because it is not used for packet forwarding. An outer encapsulation is used for forwarding ensuring transport independence.

NSH, as defined in [28], is topology-independent, allowing service insertion independently of its location in the network topology or the underlying transport protocol in use. NSH works best with VXLAN-GPE (VXLAN Generic Protocol Extension) with NSH payload. But it also supports LISP (Locator/ID Separation Protocol), GRE (Generic Routing Encapsulation), NVGRE (Network Virtualization Using GRE), amongst others.[15]

The encapsulation header contains the Base header and the Service Path Header. All other fields are optional based on the metadata (MD) field. Metadata context headers contain network platform information (VRF, Segment, etc.).

Base header carries the service header and payload protocol information. Valid values are IPv4, IPv6, Ethernet, NSH, MPLS, etc. Figure 1.11 shows de detailed base header:

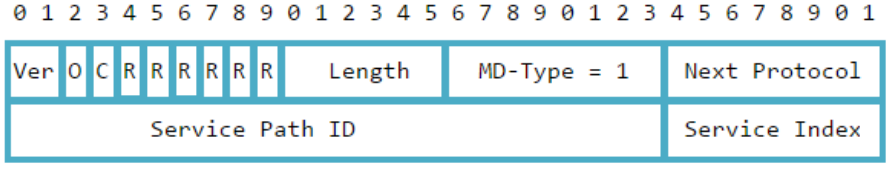

Figure C.2: NSH base header format detail

[15] VXLAN-GPE, LISP, etc. are network overlay encapsulation protocols commonly used in virtualized environments.

Service Path Header includes Service Path Identifier (SPI) and Service Index (SI). The SPI/SI combination provides the identification of a logical SF and its order within the service plane. It is used in the SFF to select the actual network locators for forwarding in the overlay.

The **SPI** is just an identifier, alone cannot be used to forward packets. It rather provides a level of indirection between the service path and the network transport. The **SI** provides service path location and can also serve as loop avoidance mechanism since each SF decrements SI along the path.

Context headers carry metadata (context data) along the service path and SFs can use this information for local decisions and policy enforcement. SFs can, in turn, inspect application information and impose additional metadata. As shown in Figure C. 3, depending on MD Type field context headers are composed of:

- **MD-Type 1**: Four fixed 32-bit context headers.
- **MD-Type 2**: A variable number of TLVs[16]

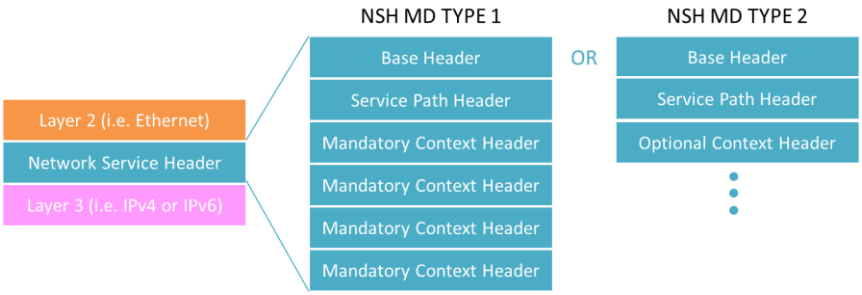

Figure C.3: NSH header format [138]

An example of context header usage can be found in [139], which defines the MD-Type 2 context header allocation for network security service functions forming a SFC. This context header provides information to support SFC operations in a generic security environment. Metadata is exchanged between Security SFs for local policy enforcement, security-based re-classification, etc. Figure C.4 shows the recommended context header allocation:

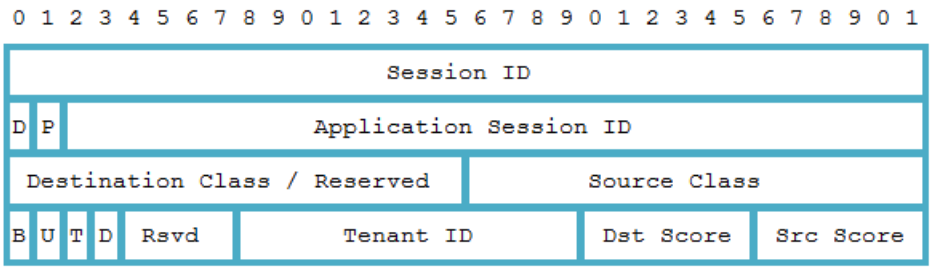

Figure C.4: Example of Type 2 context header allocation for security

[16] Type-Length-Value is a common way to encode optional information in data communication protocols. Type and length are typically fixed in size and value is of variable size.

ANNEX D. STATE OF THE ART IN VIRTUALIZED COMPUTING

D.1. Advances in virtualization and hardware architectures

The NFV technology challenges seek solutions from several fields of innovation, such as virtualization computing models or hardware architecture enhancements. While NFV proves that it can fulfill its promises, each field is evolving in a non-stop fashion. Iterative NFV implementations will have to deal with software and hardware platforms with different capabilities.

D.1.1. Virtualized computing

NFV has nourished vastly from Cloud computing regarding virtualization technologies. While the ETSI Phase 1 assumed that each VNF instance would map to one or more VMs (Virtual Machines) independently of the selected hypervisor, later discussions started considering containers, which would mean a lower resource consumption and better performance as stated in [140].

Virtual machines rely on the use a hypervisor which adds a hardware abstraction layer. Each VM runs a full guest Operating System (OS). There are two kinds of hypervisors depending if they run over a host OS such as Kernel-based Virtual Machine (KVM) or natively on bare-metal (QEMU) [141]. A further classification divides server virtualization regarding device driver emulation in Para-virtualization (XenServer) [142] or full virtualization (KVM).

Containers can be seen as a lightweight replacement for VMs. A Container is a kernel abstraction in which applications share the same kernel, libraries and binaries in a sandbox fashion. This architectural approach leads to a lower runtime overhead and shorter provisioning times. This shift in the VNF packaging aligns with the trend to implement Microservices Architectures for NFV Clouds [143].

Containers will not be a feasible replacement for virtual machines in all cases, though. VMs have some advantages in terms of security, high-availability and orchestration. For instance, VMs offer better isolation, compatibility and mobility, but container-based solutions might reach feature parity over the time with orchestration solutions like Kubernetes [145] or Mesosphere [146]. Container-based VNFs will probably co-exist with VM VNFs.

Another option for packaging VNFs is with Unikernels. Unikernels can be understood as streamlined VMs with a specialized OS instead of a general purpose one. A Unikernel is library Operating System kernel written in a high-level language. MirageOS [147] is an example of Unikernels running on top of a Xen hypervisor. Unikernels provide better isolation than Containers and can be orchestrated with the same container frontends, such as Docker [148]. Figure D.1 compares all the available VNF packaging options:

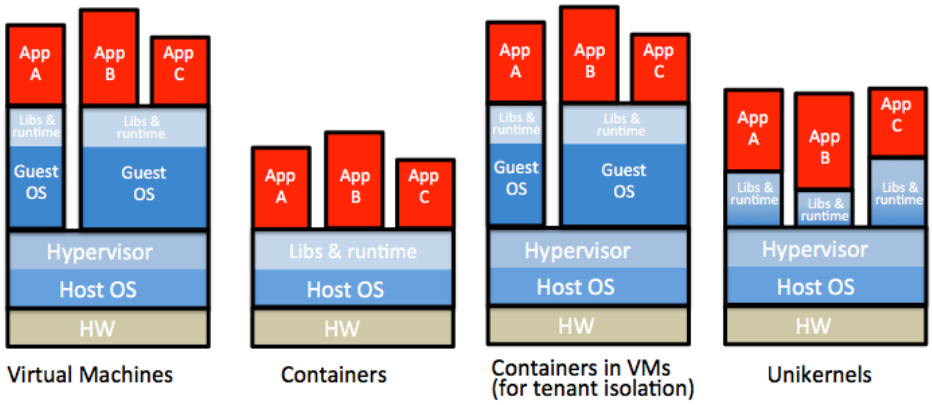

Figure D.1: Comparison of VNF packaging options [149]

Unikernels are a young technology not yet production ready. But [149] performed a NFV POC based on MirageOS early in 2016 [150]. The POC consists of a network slice orchestrator with a chaining mechanism based on shared memory. VNFs are instantiated in a just-in-time fashion to run a given service only when needed. As shown in Figure 2.5, three service Unikernels are used and a 4th one provides the network stack. The achieved memory footprints and boot times are quite impressive:

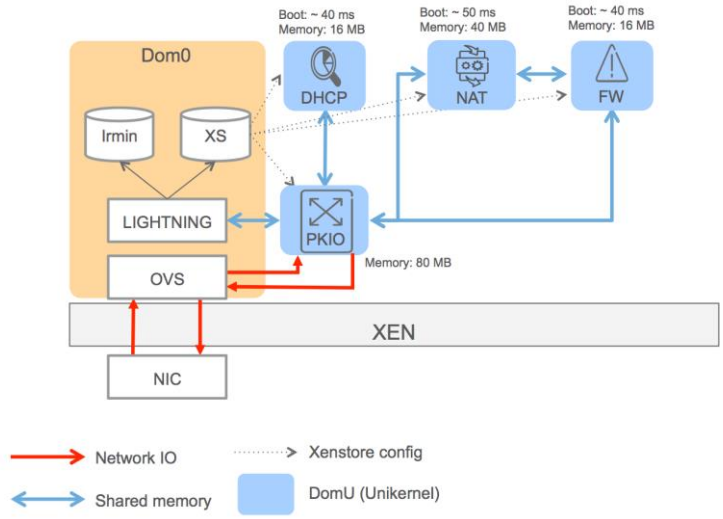

Figure D.2: Ericsson's Unikernel POC diagram

An even more cutting edge technology which could re-shape the whole NFV landscape is Serverless Computing. The idea behind Serverless Computing is invoking Application Functions[17] in the Cloud without having to instantiate a full VM. The overlapping nomenclature leaves no room for the imagination, and Mirantis showcased a POC porting Iron.io [151] Framework to OpenStack Murano in the Mirantis OpenStack Distribution [152].

[17] Functions in the context of code programming, not as virtual network functions.

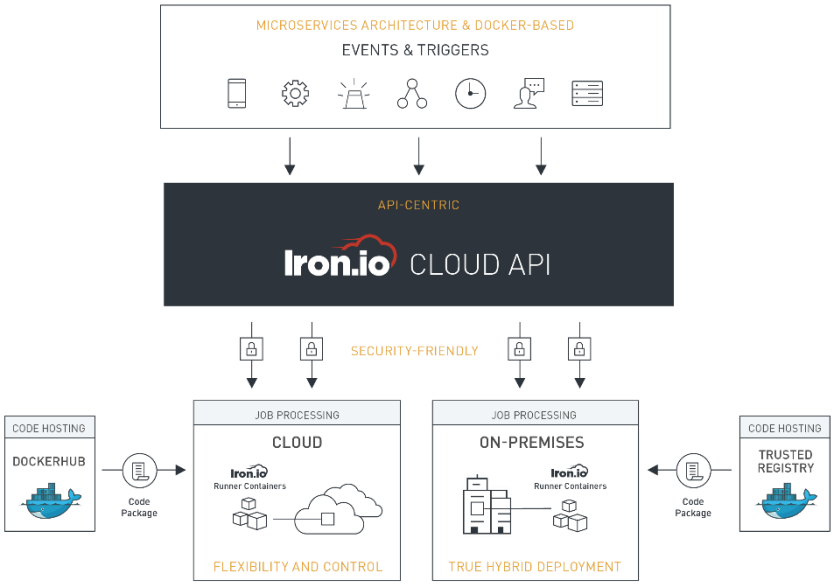

Figure D.3: Iron.io IronFunctions high-level overview

IronFunctions [153] is an open source Serverless Computing framework contender to Commercial Cloud Provider offerings like Amazon Lambda [154] or Google Cloud Functions [155]. The technology behind IronFunctions allows deploying event-triggered containerized applications and then passing jobs to them. This approach promises unprecedented speeds and scaling capabilities for several use-cases such as NFV, not only Web-Scale applications.

D.1.2. Hardware architecture enhancements

One of the main promises of NFV is the substitution of specialized network appliances by commodity computing hardware, also called COTS (Commercial off-the-shelve), which in practice refers to Intel Architecture (IA) devices. The IA, or x86 architecture provides a proven and flexible computing platform but is not optimized for networking data plane applications.

There are several techniques used in order to achieve the performance requirements of NFV, such as optimizing CPU instruction sets, introducing layers of cache memory between the processor and the CPU and increase the CPU frequency. When increasing further the frequency of a single CPU stopped scaling, CPU manufacturers introduced multiple-socket and multiple-core platforms such as Intel Sandy Bridge.

Originally, x86 systems memory access times were equal regardless the CPU in Uniform Memory Access (UMA) mode. In more recent architectures higher performance was achieved grouping CPU cores and memory into cells. This way processor systems are built upon different Non-Uniform Memory Access (NUMA) cells interconnected by a QPI (QuickPath Interconnect) bus so that all CPUs can still access the whole memory albeit at a slower speed.

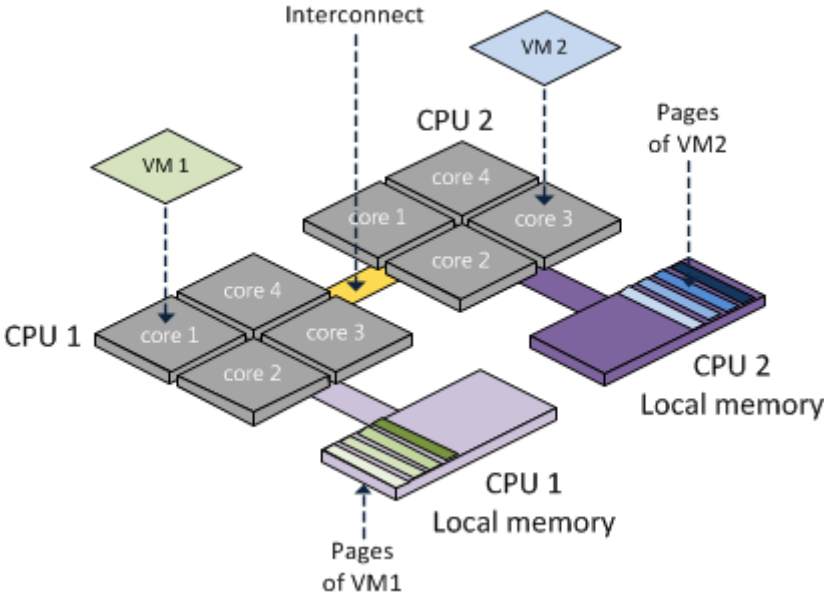

Figure D.4: Sandy Bridge-like NUMA architecture with VM pinning [156]

The CPU to memory affinity concept has also been extended to PCIe I/O channels between CPUs, with the same properties as memory. Virtual to physical CPU core pinning as represented in Figure 2.7 implements this optimization at the VM level. The performance improvements which can be achieved help realizing specialized workloads for VM-based NFV deployments.

D.1.3. Network I/O acceleration

As most of the VNFs will be more I/O-bound[18] than CPU-bound, a brief introduction on the state-of-the-art network I/O acceleration has been considered useful to understand how the high-performance requirements are technically addressed. There are several approaches to maximize throughput and minimize latency at the host level (without the use of a virtual switch): PCI pass-through and NIC offloading with specialized adapters.

PCI pass-through works by assigning a physical NIC to a VM bypassing the hypervisor. An example of this is Single Root Input/Output Virtualization (SR-IOV) [157]. SR-IOV is a PCI Express (PCIe) extension for network interface cards (NICs) that enables NIC physical port virtualization. Each virtual port, called Virtual Function (VF)[19], can be associated to a VM. The NIC hardware to offload functions from the CPU improving latency and the overall server performance.

Figure D.5 compares three network pass-through models: normal (no pass-through), operation through bottleneck eradication with Virtual Machine Device Queues (VMDq) and interrupt mitigation with Single Root Input/Output Virtualization (SR-IOV):

[18] Where data reading/writing consumes more resources than CPU processing.
[19] Again, not to be confused with the NFV virtual network functions.

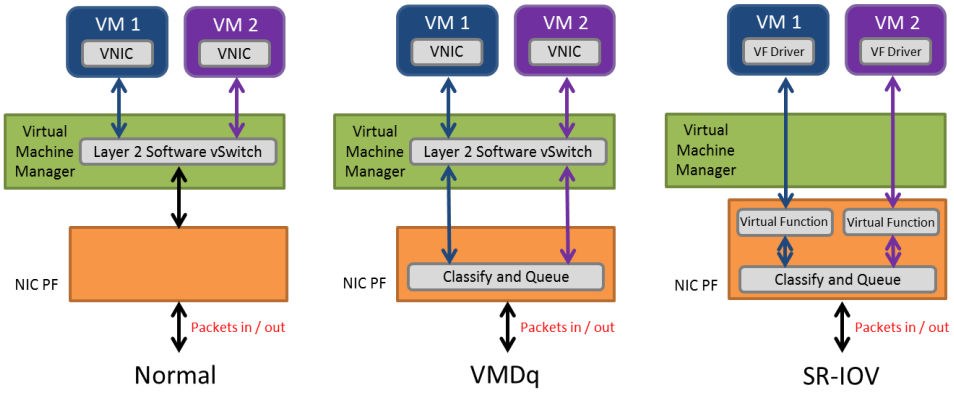

Figure D.5: Comparison of datapath acceleration options [158]

Specialized NICs can process server networking functions sparing CPU resources back to the applications. The so-called NFV NICs use either System-on-a-Chip (SOC) [159] or Field Programmable Gate Arrays (FPGAs) [160] that run as a high-speed programmable switch. They are more expensive than regular NICs used in SR-IOV mode, but they provide a higher performance useful in I/O centric deployments.

D.2. Nova NFV infrastructure platform awareness

A virtual infrastructure manager NFV solution will have to be aware of the available enhancements presented in the former sections in case a VNF instance requires special features. Mirantis Open NFV platform [161] will be used as an example on how OpenStack is leveraged for NFV use cases in terms of hardware features that help to realize the NFV requirements.

As per [162], an OpenStack platform has to provide the following features in Nova to fulfill the NFV high performance and low latency requirements:

- Guaranteed workload resources to avoid shared resource contention.
- Huge pages for faster memory lookups.
- NUMA/CPU pinning to ensure that CPU and memory proximity.
- SR-IOV for hypervisor pass-through.
- Anti-affinity groups that spread workloads to prevent resource contention.

Enhanced Platform Awareness (EPA) [163] is the OpenStack feature which takes care of intelligent VNF placement on the most appropriate compute node based on best matching capabilities. Compute nodes can be organized in Nova availability zones[20] for special NFV purposes. Using TOSCA VNFD templates that allow specifying special Virtual Deployment Unit (VDU) requirements for a VNF.

[20] Nova availability zones (AZs) are a concept linked to the host aggregates concept and not relating to other implementations of so-called availability zones. Host aggregates group hosts by means of key-value pair attributes.

Enhanced feature support is dependent on the hypervisor and the underlying platform. For example, Intel Architectures support fine-grained requirement selection for several use-cases: compute, encryption/compression acceleration, etc. Figure 2.9 shows how EPA enables the Nova filter scheduler to select a server that can deliver a flavor with specific hardware requirements:

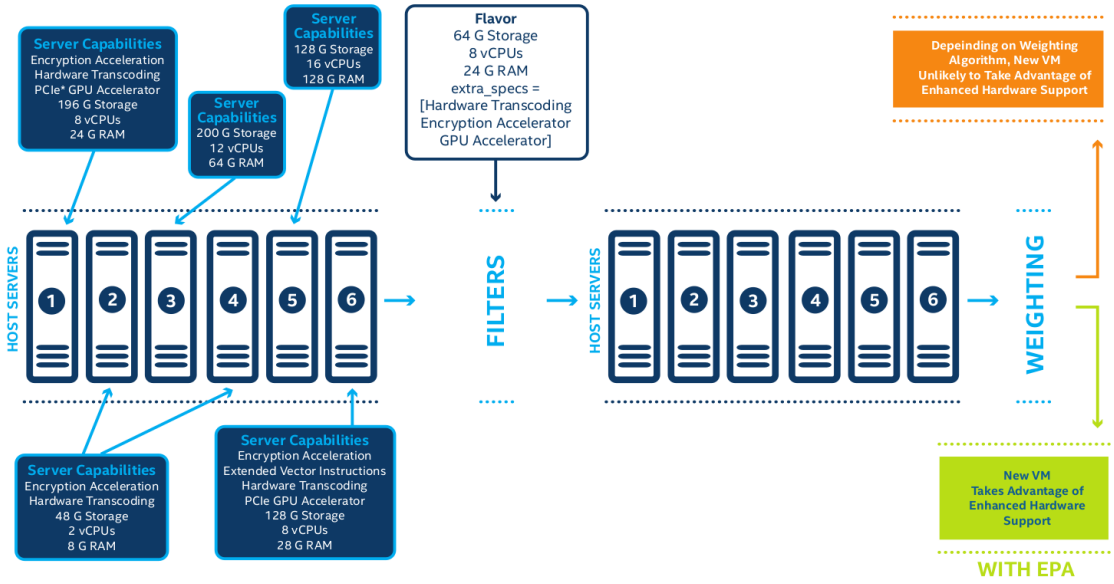

Figure D.6: Nova support for Enhanced Platform Awareness [164]

ANNEX D. OPNFV PROJECTS OVERVIEW

There are several building blocks that conform the OPNFV architecture to achieve the project goals. In the following table all the OPNFV stack components are grouped according to their function:

Building block	Function	Project/Tool
Compute	VM Control	Openstack Nova
	Hypervisor	KVM
Storage	Store VNF images	Openstack Glance
	Virtual Disks	Openstack Cinder
	Volumes	Ceph
Network	Network Control	OpenDaylight
	Network Forwading	Open vSwitch
Infrastructure	Message Bus	RabbitMQ
	Cluster Communication	Corosync
	HA and Loadbalance	PaceMaker
	Database	MySQL
MANO	Portal	Openstack Horizon
	Identity	Openstack Keystone
	Orchestration	Openstack Heat
	VNF Manager	Openstack Tacker
	Telemetry	Openstack Ceilometer
Testing	Verify	Openstack Tempest
	Benchmark	Rally
	Test VNFs	Sample VNFs

Table D.1: OPNFV project to NFV architecture mapping

E.1. OPNFV software stack

Depending on the installation toolchain, OPNFV currently supports Linux on target machines, basically Ubuntu 14.04 and CentOS 6.5. SUSE is expected for the Danube release. Virtual Infrastructure Management functionality is achieved via OpenStack. OPNFV consumes a sub-set of OpenStack projects relevant to NFV. As an network focused project it has a broad range of controllers and forwarding technologies.
Neutron provides connectivity between VNF interfaces managed by other services, namely Nova. Neutron can be integrated with several external

controllers providing a rich set of features. OPNFV extends Linux virtual networking with virtual switching and routing components like OVS or FD.io

Platform validation and measurement is a key point in a carrier-grade environment. To address this requirement efforts in OPNFV have been put on automated testing tools. Release and scenario validation is implemented thanks to the functional testing project (Functest [93]) and the performance test project (Yardstick [165]). Functest leverages OpenStack and SDN controllers' testing frameworks to make sure that the OPNFV platform is running correctly. Yardstick benchmarks performance metrics.

Additional testing tools address the validation of specific features of the OPNFV platform further extending test cases provided by Functest and Yardstick. Examples of projects focused on these areas are VSperf [166], which is a generic virtual switch testing framework and project Bottlenecks [167], a proposal intended to address the system limitations by isolating platform bottlenecks. Table E.1 lists all the approved projects for the Colorado release:

Project	**Goal**
ARMband	Support for the AArch64 infrastructure.
Copper	Add the OpenStack Congress policy framework.
Doctor	Fault management and maintenance framework HA.
Domino	TOSCA based service distribution for NS/VNF descriptors.
Fast Data Stacks	Provide a FD.io scenario for high performance networking.
IPv6	Produce an IPv6 compliant OPNFV distribution.
KVM for NFV	Framework to enhance the KVM Hypervisor for NFV.
Moon	Provide a security management framework.
Multisite	Enable the OpenStack to support multi-site NFV clouds.
NetReady	Evolve Neutron to fulfill the NFV requirements.
ONOSFW	Integrate ONOS SDN controller to OPNFV.
OVS for NFV	Improve Open vSwitch performance.
Parser	YANG to TOSCA to Heat templates translation.
Pharos	Provide an overview for setting up a Pharos lab.
Promise	Implement resource reservation and management.
SDN VPN	Extend Neutron to create BGP/MPLS based VPNs.
SFC	Integrate the OpenDaylight SFC project into OPNFV.
Bottlenecks	Automate system limitation testing and benchmarking.
Functest	Functional testing framework for OPNFV and upstream.
VSPerf	Traffic generator framework for virtual switches.
Yardstick	Deliver Framework for automated scenario testing.

Table D.2: List of OPNFV projects and their goals

Figure E.1 shows a categorized view of the projects grouped in PaaS (Platform-as-a-Service), cloud infrastructure and tools, infrastructure and continuous integration and continuous deployment (CI/CD). Installation related projects are explained in more detail in the following section.

Figure D.1: OPNFV Projects categorization [2]

ANNEX D. LINUX NETWORKING

This annex collects information relevant to virtual networking for Linux.

F.1. Linux networking

The Linux Kernel distinguishes two kinds of network devices:

- Physical interfaces represent a hardware device and are available as soon as the Kernel modules is loaded.
- Virtual interfaces are just point-to-point devices which exchange packets with user-space programs instead of physical interfaces. Besides the loopback interface, virtual interfaces are associated to a physical interface or to another virtual interface.

A non-exhaustive list of virtual interface types follows:

- Aliases are an obsolete mechanism to define secondary IPs which is now supported via iproute2, the toolkit that replaced net-tools (ifconfig, etc.)
- VLANs create virtual segments of a layer 2 networks based on the 802.1Q twelve bit tag allowing 4096 virtual LANs.
- QinQ stacks VLAN tags according to 802.1ad standard addressing provider bridging requirements.
- Bridges can be used to connect multiple ethernet segments in a transparent way.
- TUN (TUNnel) interfaces are used to encapsulate packets at network layer over tunneling protocols such as GRE, IPSec or IPv6 tunneling mechanisms..
- TAP interfaces use the same kernel driver than TUN but represent a link layer device operating at layer 2 allowing to provide virtual network adaptors to guest machines in virtualization systems.

F.1.1. Linux bridges

The Linux bridge is a kernel module that implements a virtual switch in software. It bridges two or more network segments transparently. Both physical and virtual interfaces can be associated to it. It is used with the KVM hypervisor and can be managed with the user-space tool *brctl* although it has also been superseded by the *iproute2* toolkit. Advanced processing can be done at layer 2 thanks to ebtables similar to layer 3 iptables.

The workflow with bridges is the following:

- Create a linux bridge
- Associate interfaces to the bridge
- Optionally define a virtual IP for the bridge
- Define bridge settings

- Control switch runtime parameters
- Optionally make bridge settings persistent. This is distribution dependant.

F.1.2. Linux Namespaces

Namespaces are a construct used in Neutron introduced in the Linux Kernel allowing several scopes of isolation without the need of a full virtualization layer. There are namespaces for IPC message queues, process IDs, user IDs, mount points, hostnames and network related resources. A global space of each kind exist and additional namespaces can be defined if necessary. Namespaces provide the following features:

- The same identifier can be used multiple times in different namespaces.
- Objects within a namespace are isolated from objects belonging to global or the rest of namespaces.

Similar to VRFs (Virtual Routing and Forwarding), the network namespaces provide a separate network stack. Several network resources are scoped within a namespace:

- Interfaces within a namespace cannot communicate outside the namespaces unless an external mechanism is put in place (bridge, veth pair). Overlapping Interface names and addresses can be used in different namespaces.
- Network addresses can overlap between different namespaces leveraging multi-tenancy schemes in OpenStack.
- Different routing tables can be mapped to a namespace, allowing for advanced features such as Policy Based Routing.
- Each namespace has an independent IPtables tables and chains applying to the namespace-specific netfilter hooks.

The network namespage workflow uses the iproute2 toolkit and involves:

- Create a namespace
- Associate interfaces to namespaces
- Assign address to interfaces
- Configure routes/iptables rules for the given namespace
- Optionally connect to interfaces belonging to other namespaces by means of bridge/router/veth pair
- Execute commands within the namespace

Table F.1 shows how to interact with a network namespace:

```
root@node-2:~# ip netns
qdhcp-b49acc33-dbcf-4a8f-87ba-0ae957e0221d
qrouter-0a86afb9-dae2-4c17-9d4b-59664dbd678e
qdhcp-afd571bb-0b11-445a-8898-ec44582b9088
haproxy
vrouter
root@node-2:~# ip netns exec qdhcp-b49acc33-dbcf-4a8f-87ba-0ae957e0221d ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

```
42: tap72f0bc96-97: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state
UNKNOWN group default
    link/ether fa:16:3e:fc:15:0f brd ff:ff:ff:ff:ff:ff
    inet 22.22.22.2/24 brd 22.22.22.255 scope global tap72f0bc96-97
       valid_lft forever preferred_lft forever
    inet 169.254.169.254/16 brd 169.254.255.255 scope global tap72f0bc96-97
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fefc:150f/64 scope link
       valid_lft forever preferred_lft forever
root@node-2:~# ip netns exec qdhcp-b49acc33-dbcf-4a8f-87ba-0ae957e0221d ip route
default via 22.22.22.1 dev tap72f0bc96-97
22.22.22.0/24 dev tap72f0bc96-97  proto kernel  scope link  src 22.22.22.2
169.254.0.0/16 dev tap72f0bc96-97  proto kernel  scope link  src 169.254.169.254
```

Table D.1: CLI command workflow example for a Neutron namespace

F.1.3. Veth pairs

Linux tap interfaces cannot be used to link namespaces. But veth interfaces can. Veth interfaces are software ethernet adapters connected back-to-back. That is, they always exist in pairs acting as a virtual pipe. As they support carrier detection they can be seen as a virtual cable. Deleting one end of the pair causes both interfaces to be removed.

ANNEX D. DETAILED SCENARIO SETUP GUIDE

In this section the details to deploy the scenarios presented in Chapter 4 are collected. The configuration files have been included in a Git repository [168] forked from the OPNFV Gerrit Fuel repository [85] as shown in Figure AI.1:

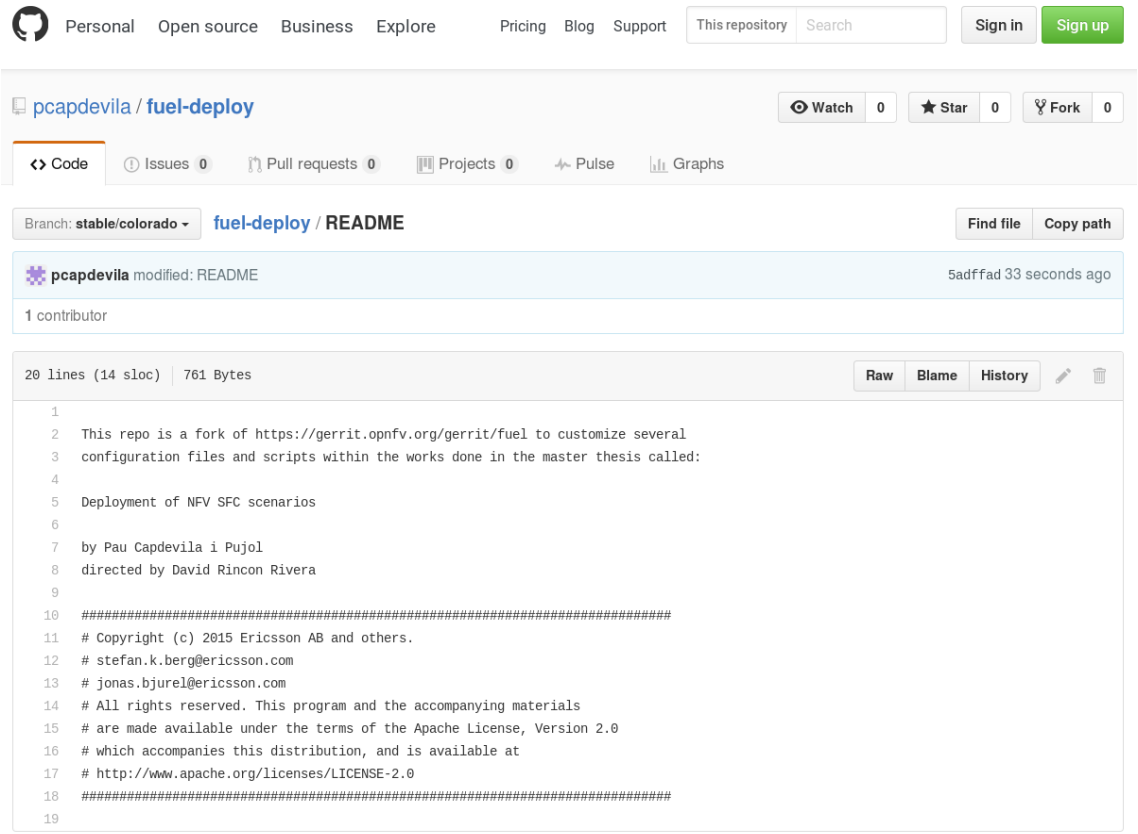

Figure D.1: fuel-deploy git repository on github.com

G.1. Common preliminary tasks

Install the Jump server OS and required packages

On the freshly installed and upgraded Ubuntu 14.04 jump server the following Ubuntu packages are needed prior to being able to run the OPNFV Fuel installation:

```
sudo apt install -y git make curl libvirt-bin libpq-dev qemu-kvm qemu-system sshpass
fuseiso genisoimage blackbox python-pip python-git python-dev python-oslo.config python-
pip python-dev libffi-dev libxml2-dev libxslt1-dev libffi-dev libxml2-dev libxslt1-dev
expect curl python-netaddr p7zip-full libvirt-bin qemu-kvm python-pip fuseiso mkisofs
genisoimage python-dev libz-dev libxml2-dev libxslt-dev libyaml-dev kvm bridge-utils
vlan libffi-dev libssl-dev screen
```

Logout and login in order to make sure the user id has been included in libvirtd group.

```
pcapdevila@eul1900636:~$ id
uid=1000(pcapdevila) gid=1000(pcapdevila) groups=1000(pcapdevila),27(sudo),111(libvirtd)
```

The following Python packages are also required:

```
sudo python -m pip install -U pip
sudo pip install -U pip setuptools
sudo pip install --upgrade GitPython pyyaml netaddr paramiko lxml scp pycrypto ecdsa
debtcollector netifaces enum cryptography certifi urllib3[secure]
```

Download the installer source code

In the jump server run:

```
git clone -b 'stable/colorado' https://github.com/pcapdevila/fuel-deploy.git
```

Configuration files are in the following folders:

```
pcapdevila@eul1900636:~$ ls -l fuel-deploy/deploy/config/labs/devel-
pipeline/kvm/fuel/config/
total 12
-rw-r--r-- 1 pcapdevila pcapdevila 5868 Jan 25 09:08 dea-pod-override.yaml
-rw-r--r-- 1 pcapdevila pcapdevila 2087 Jan 25 09:08 dha.yaml
pcapdevila@eul1900636:~$ ls -l fuel-deploy/deploy/config/labs/devel-
pipeline/lab235/fuel/config/
total 12
-rw-r--r-- 1 pcapdevila pcapdevila 5797 Jan 25 09:08 dea-pod-override.yaml
-rw-r--r-- 1 pcapdevila pcapdevila 1331 Jan 25 09:08 dha.yaml
```

The installer takes a few hours depending on the hardware specs. It is recommended to run the installer in a terminal multiplexer able to restore the login session from another connection point:

```
screen -RD
```

Enter the CI folder:

```
cd fuel-deploy/ci
```

G.2. Virtual lab deployment

Launch the deployment:

```
sudo bash ./deploy.sh -b file:///home/pcapdevila/fuel-deploy/deploy/config -l devel-
pipeline -p kvm -s no-ha_odl-l2_sfc_heat_ceilometer_scenario.yaml -i
http://artifacts.opnfv.org/fuel/colorado/opnfv-colorado.3.0.iso
```

Once the Fuel master is installed and configured, the deployment progress can be monitored also through the web UI available at https://10.20.0.2[21] (admin/admin) as shown in Figure A.2

[21] In the virtual lab a graphical user interface for Linux is recommended to be able to access the OpenStack and Opendaylight administration web portals.

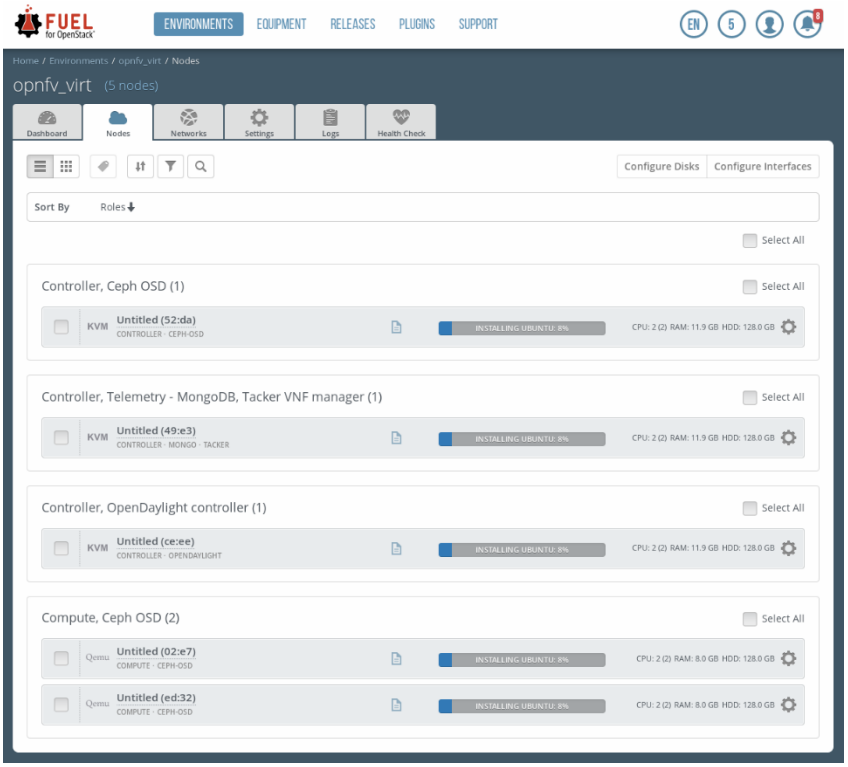

Figure D.2: Fuel environment deployment progress trough the web UI

Once deployment is completed, all health checks should pass as showin in Figure:

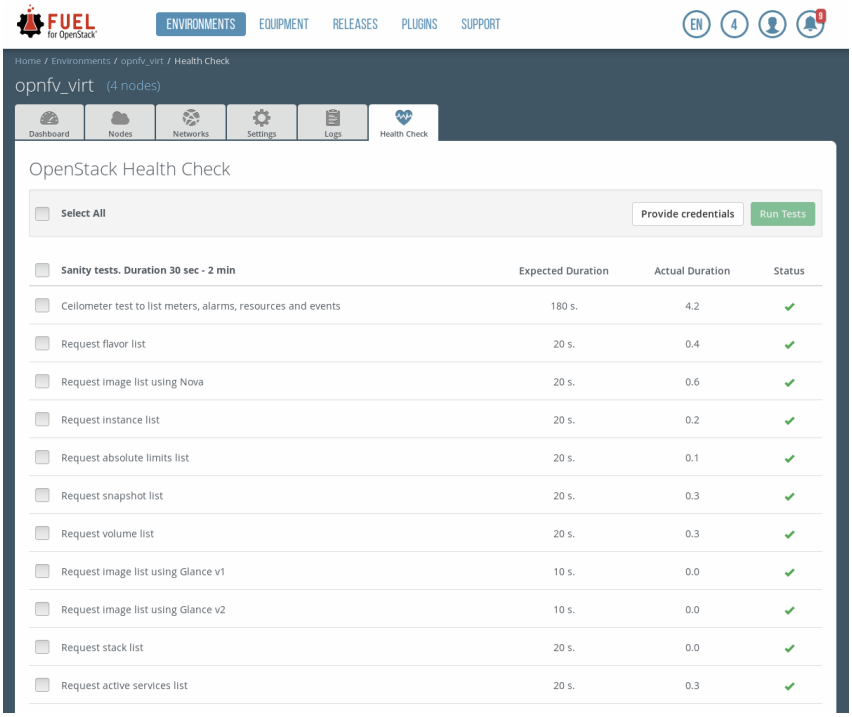

Figure D.3: Fuel environment deployment progress trough the web UI

G.3. Bare-metal lab deployment

For the bare-metal lab, the procedure is the same until this point. To launch the deployment, execute:

```
pau@LXAR-HP-02:~/fuel-deploy/ci$ sudo bash ./deploy.sh -f -b file:///home/pau/fuel-
deploy/deploy/config -l devel-pipeline -p lab235 -s ha_odl-l2_sfc_heat_ceilometer_scenario.yaml -i
file:///home/pau/opnfv
-colorado.3.0.iso
```

(Output omitted)

```
Fuel Master installed successfully !

Power OFF Node 1[22]
Power OFF Node 2
Power OFF Node 3
Power OFF Node 4
Power OFF Node 5
Set boot order ['pxe', 'disk'] on Node 1
Set boot order ['pxe', 'disk'] on Node 2
Set boot order ['pxe', 'disk'] on Node 3
Set boot order ['pxe', 'disk'] on Node 4
Set boot order ['pxe', 'disk'] on Node 5
Power ON Node 1
Power ON Node 2
Power ON Node 3
Power ON Node 4
Power ON Node 5


Check prerequisites
Check supported release: Mitaka on Ubuntu 14.04
Check previous installation
Deleting environment 2
Deleting node 7
Wait for discovered blades
Blade 4 discovered as Node 14 with MAC 78:e7:d1:c6:68:ab
Blade 3 discovered as Node 15 with MAC 78:e7:d1:c6:68:87
Blade 1 discovered as Node 16 with MAC 78:e7:d1:c6:69:d9
Blade 2 discovered as Node 18 with MAC 78:e7:d1:89:91:5d
Blade 5 discovered as Node 17 with MAC 78:e7:d1:c6:68:fa
Deleting file /home/pau/fuel-deploy/ci/config/.dea.yaml


START CLOUD DEPLOYMENT


Command: fuel release -l
id | name                         | state       | operating_system | version
---+------------------------------+-------------+------------------+-----------
2  | Mitaka on Ubuntu 14.04       | available   | Ubuntu           | mitaka-9.0
3  | Mitaka on Ubuntu+UCA 14.04   | available   | Ubuntu           | mitaka-9.0
1  | Mitaka on CentOS 6.5         | unavailable | CentOS           | mitaka-9.0
Configure environment


Deleting directory /var/lib/opnfv
Creating directory /var/lib/opnfv
Creating environment lab235 release 2 net-segment-type tun


Command: fuel env create --name lab235 --release 2 --net-segment-type tun
Environment 'lab235' with id=3 was created!
Command: fuel env --list
id | status | name   | release_id
---+--------+--------+-----------
3  | new    | lab235 | 2
Configure settings
```

(Output omitted)

```
Environment 3 successfully deployed


Command: fuel --env 3 node
id | status | name             | cluster | ip        | mac               | roles
```

[22] In the bare-metal environment these actions are expected to be performed manually once the Fuel master installation is successfully completed.

```
---+--------+------------------+---------+-----------+-------------------+--------------------------
17 | ready  | Untitled (68:fa) |       3 | 10.20.0.7 | 78:e7:d1:c6:68:fa | ceph-osd, compute
14 | ready  | Untitled (68:ab) |       3 | 10.20.0.8 | 78:e7:d1:c6:68:ab | ceph-osd, compute
16 | ready  | Untitled (69:d9) |       3 | 10.20.0.3 | 78:e7:d1:c6:69:d9 | controller, opendaylight
15 | ready  | Untitled (68:87) |       3 | 10.20.0.5 | 78:e7:d1:c6:68:87 | ceph-osd, controller
18 | ready  | Untitled (91:5d) |       3 | 10.20.0.4 | 78:e7:d1:89:91:5d | controller, mongo, tacker

Now running sanity and smoke health checks


Command: fuel health --env 3 --check sanity,smoke --force
[ 1 of 27] [success] 'Ceilometer test to list meters, alarms, resources and events' (2.485 s)
[ 2 of 27] [success] 'Create instance flavor' (3.559 s)
[ 3 of 27] [success] 'Request flavor list' (0.2009 s)
[ 4 of 27] [success] 'Request image list using Nova' (3.45 s)
[ 5 of 27] [success] 'Check create, update and delete image actions using Glance v2' (3.581 s)
[ 6 of 27] [success] 'Request instance list' (1.072 s)
[ 7 of 27] [success] 'Request absolute limits list' (0.04518 s)
[ 8 of 27] [success] 'Request snapshot list' (1.616 s)
[ 9 of 27] [success] 'Request volume list' (0.1665 s)
[10 of 27] [success] 'Request image list using Glance v1' (0.02409 s)
[11 of 27] [success] 'Request image list using Glance v2' (0.03343 s)
[12 of 27] [success] 'Request stack list' (0.02779 s)
[13 of 27] [success] 'Request active services list' (0.274 s)
[14 of 27] [success] 'Request user list' (0.1318 s)
[15 of 27] [success] 'Check that required services are running' (16.1 s)
[16 of 27] [success] 'Check internet connectivity from a compute' (0.3064 s)
[17 of 27] [success] 'Check DNS resolution on compute node' (0.5205 s)
[18 of 27] [success] 'Request list of networks' (0.4317 s)
[19 of 27] [success] 'Create volume and boot instance from it' (98.82 s)
[20 of 27] [success] 'Create volume and attach it to instance' (72.31 s)
[21 of 27] [success] 'Check network connectivity from instance via floating IP' (177.3 s)
[22 of 27] [success] 'Create keypair' (0.5829 s)
[23 of 27] [success] 'Create security group' (1.01 s)
[24 of 27] [success] 'Check network parameters' (0.1734 s)
[25 of 27] [success] 'Launch instance' (27.87 s)
[26 of 27] [success] 'Launch instance, create snapshot, launch instance from snapshot' (62.24 s)
[27 of 27] [success] 'Create user and authenticate with it.' (19.08 s)
[ 1 of 27] [success] 'Ceilometer test to list meters, alarms, resources and events' (2.485 s)
[ 2 of 27] [success] 'Create instance flavor' (3.559 s)
[ 3 of 27] [success] 'Request flavor list' (0.2009 s)
[ 4 of 27] [success] 'Request image list using Nova' (3.45 s)
[ 5 of 27] [success] 'Check create, update and delete image actions using Glance v2' (3.581 s)
[ 6 of 27] [success] 'Request instance list' (1.072 s)
[ 7 of 27] [success] 'Request absolute limits list' (0.04518 s)
[ 8 of 27] [success] 'Request snapshot list' (1.616 s)
[ 9 of 27] [success] 'Request volume list' (0.1665 s)
[10 of 27] [success] 'Request image list using Glance v1' (0.02409 s)
[11 of 27] [success] 'Request image list using Glance v2' (0.03343 s)
[12 of 27] [success] 'Request stack list' (0.02779 s)
[13 of 27] [success] 'Request active services list' (0.274 s)
[14 of 27] [success] 'Request user list' (0.1318 s)
[15 of 27] [success] 'Check that required services are running' (16.1 s)
[16 of 27] [success] 'Check internet connectivity from a compute' (0.3064 s)
[17 of 27] [success] 'Check DNS resolution on compute node' (0.5205 s)
[18 of 27] [success] 'Request list of networks' (0.4317 s)
[19 of 27] [success] 'Create volume and boot instance from it' (98.82 s)
[20 of 27] [success] 'Create volume and attach it to instance' (72.31 s)
[21 of 27] [success] 'Check network connectivity from instance via floating IP' (177.3 s)
[22 of 27] [success] 'Create keypair' (0.5829 s)
[23 of 27] [success] 'Create security group' (1.01 s)
[24 of 27] [success] 'Check network parameters' (0.1734 s)
[25 of 27] [success] 'Launch instance' (27.87 s)
[26 of 27] [success] 'Launch instance, create snapshot, launch instance from snapshot' (62.24 s)
[27 of 27] [success] 'Create user and authenticate with it.' (19.08 s)
```

ANNEX E. DETAILED SCENARIO WALKTHROUGH

Once the environment is successfully installed, access to the OpenStack Horizon and OpenDaylight is active and should be reachable from the Jump server at:

- OpenStack Horizon: http://172.16.0.3
- Opendaylight DLUX: http://172.16.0.3:8181/index.html

Figure H.1 shows the ODL DLUX UI:

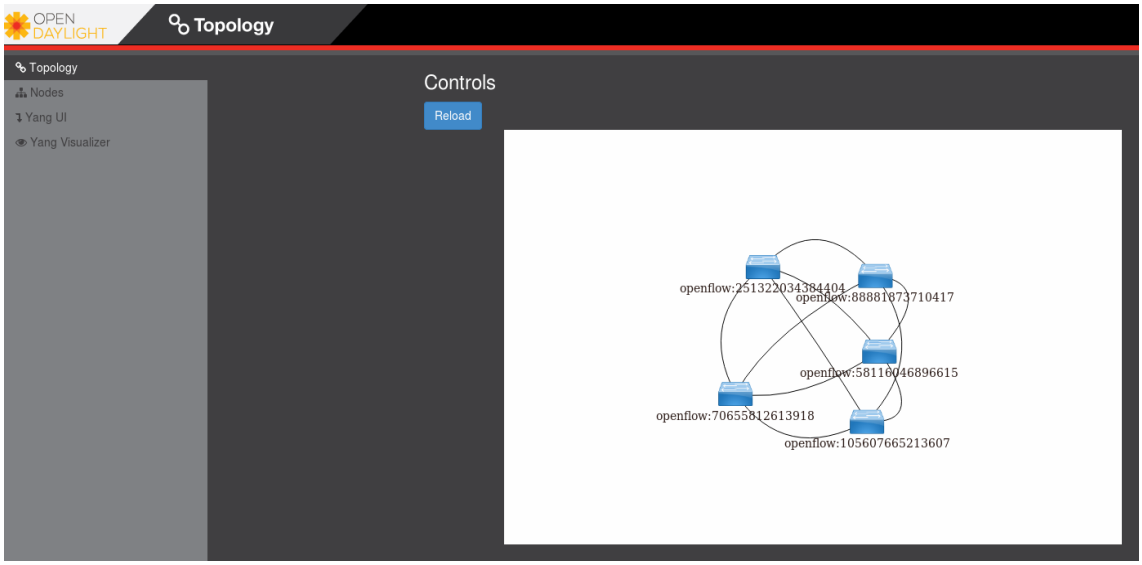

Figure E.1: Opendaylight DLUX UI showing the OpenFlow topology

Access to the node's CLI is done via the Fuel master. For example, to log into the OpenStack controller, from the jump server:

```
pcapdevila@eul1900636:~$ sshpass -p r00tme ssh root@10.20.0.2
Last login: Thu Feb 16 12:46:26 2017 from 10.20.0.1
[root@fuel ~]# fuel node
id | status | name             | cluster | ip         | mac               | roles
| pending_roles | online | group_id
---+--------+------------------+---------+------------+-------------------+--------------------------
 2 | ready  | Untitled (1d:95) |       1 | 10.20.0.6  | 52:54:00:57:1d:95 | ceph-osd, compute
 3 | ready  | Untitled (e2:b8) |       1 | 10.20.0.4  | 52:54:00:9c:e2:b8 | ceph-osd, opendaylight
 4 | ready  | Untitled (01:b0) |       1 | 10.20.0.7  | 52:54:00:eb:01:b0 | controller, mongo, tacker
 1 | ready  | Untitled (e2:56) |       1 | 10.20.0.5  | 52:54:00:53:e2:56 | ceph-osd, compute
[root@fuel ~]# ssh 10.20.0.7
Warning: Permanently added '10.20.0.7' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-103-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Thu Feb 16 12:42:47 2017 from 10.20.0.2
root@node-4:~# source openrc
root@node-4:~# source tackerc
root@node-4:~# openstack service list
+----------------------------------+----------------+----------------+
| ID                               | Name           | Type           |
+----------------------------------+----------------+----------------+
| 081102419f6f4624b4bdcffb80cd2bbb | tacker         | servicevm      |
| 0e9596c19dd24b1d8bb1b160c9c73835 | heat-cfn       | cloudformation |
| 15344b79ba0546dd849bc22c158da4df | cinderv2       | volumev2       |
| 3124631e0ee948a0a320463ff2df52d0 | compute_legacy | compute_legacy |
| 3171a17be0774fcfb519d32b0cd6ff43 | cinderv3       | volumev3       |
| 60ca82caf8e54c89b714c069b9339b4c | neutron        | network        |
| 8fbe6df3bdba4107abf37a4cab024f72 | keystone       | identity       |
| 96e8072a8d0e4622b60d82f92cc1230a | swift          | object-store   |
```

```
| 9f1e0a8176384db78a3dd3c6b5b2e2ed | heat           | orchestration  |
| b281d03f48384a55923efc99c3c05ecc | glance         | image          |
| c47e245fcb2f42b7949dbfd6e1d980fb | cinder         | volume         |
| d35a8ab7746544d58e793e5a8dc58340 | nova           | compute        |
| d6aa6d0fa22745d7a2c4eb91e94d1b76 | swift_s3       | s3             |
| dee2ccc2bb0c4f25ba5353e8ba77ed29 | aodh           | alarming       |
| f0c5c903f386475e8803e2abbe78b681 | ceilometer     | metering       |
| fd3ceaaeee0049a0b9a291a2b3f3e79e | glare          | artifact       |
+----------------------------------+----------------+----------------+
```

And to log into the Opendaylight Karaf shell to install the missing DLUX SFC UI, from the jump server:

```
pcapdevila@eul1900636:~$ sshpass -p r00tme ssh root@10.20.0.2
Last login: Thu Feb 16 12:46:26 2017 from 10.20.0.1
[root@fuel ~]# fuel node
id | status | name             | cluster | ip        | mac               | roles
| pending_roles | online | group_id
---+--------+------------------+---------+-----------+-------------------+--------------------------
 2 | ready  | Untitled (1d:95) |       1 | 10.20.0.6 | 52:54:00:57:1d:95 | ceph-osd, compute
 3 | ready  | Untitled (e2:b8) |       1 | 10.20.0.4 | 52:54:00:9c:e2:b8 | ceph-osd, opendaylight
 4 | ready  | Untitled (01:b0) |       1 | 10.20.0.7 | 52:54:00:eb:01:b0 | controller, mongo, tacker
 1 | ready  | Untitled (e2:56) |       1 | 10.20.0.5 | 52:54:00:53:e2:56 | ceph-osd, compute
[root@fuel ~]# ssh 10.20.0.4
root@node-3:~# /opt/opendaylight/bin/client
client: JAVA_HOME not set; results may vary
Logging in as karaf
19352 [sshd-SshClient[ee7d9f1]-nio2-thread-1] WARN
org.apache.sshd.client.keyverifier.AcceptAllServerKeyVerifier - Server at [/0.0.0.0:8101, RSA,
62:9e:ff:a3:b7:f7:d2:55:45:20:fb:47:73:5c:5b:ed] presented unverified {} key: {}

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.|  | |__| ____ |  |___/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|    `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/            /_____/      \/


Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>feature:install odl-sfc-ui
```

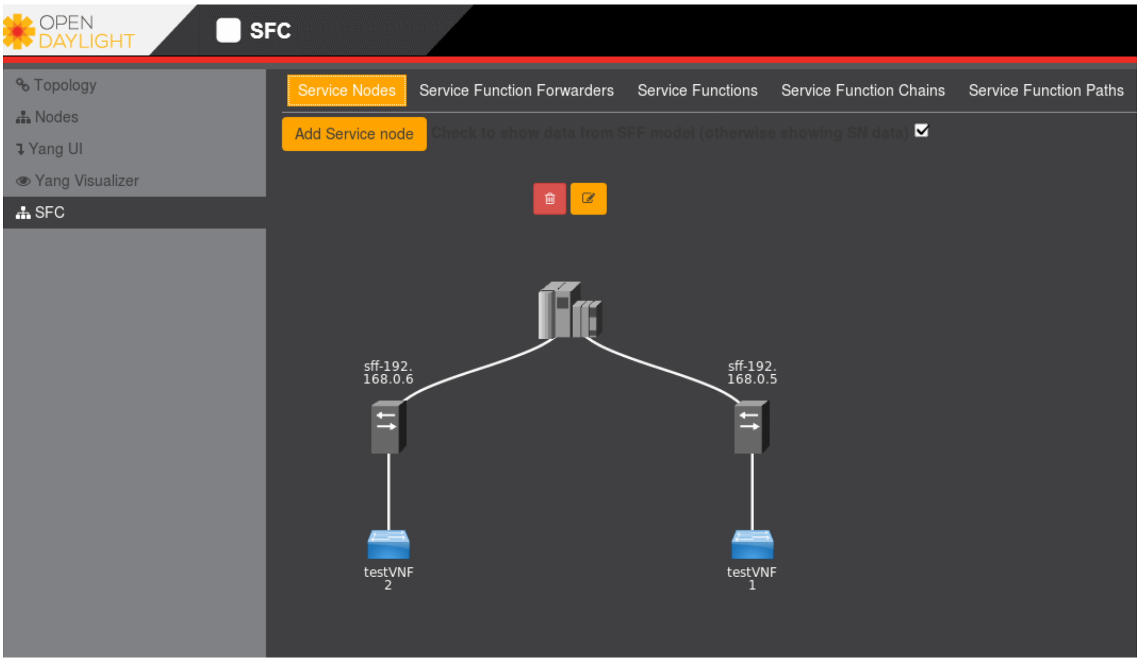

Figure E.2: Opendaylight DLUX SFC UI service node view

H.1. Functional test execution

For functional testing OPNFV Functest is used. In order to install the Functest Docker image, run:

```
pcapdevila@eul1900636:~$ sudo apt-get install docker-engine
pcapdevila@eul1900636:~$ docker pull opnfv/functest:colorado.3.0
pcapdevila@eul1900636:~$ docker run --privileged=true -id -e INSTALLER_TYPE=fuel -e
INSTALLER_IP=10.20.0.2 -e DEPLOY_SCENARIO=os-odl_l2-sfc-noha -e CI_DEBUG=true --name sfc
opnfv/functest:Colorado.3.0
```

To log into the running container and prepare the Functest environment:

```
pcapdevila@eul1900636:~$ docker exec -ti sfc bash

root@2e5bd681631e:~# functest env prepare
2017-02-16 01:18:36,149 - prepare_env - INFO - ######### Preparing Functest environment #########
2017-02-16 01:18:36,149 - prepare_env - INFO - ===============================================
2017-02-16 01:18:36,149 - prepare_env - INFO - Checking environment variables...
2017-02-16 01:18:36,149 - prepare_env - INFO -     INSTALLER_TYPE=fuel
2017-02-16 01:18:36,150 - prepare_env - INFO -     INSTALLER_IP=10.20.0.2
2017-02-16 01:18:36,150 - prepare_env - INFO -     DEPLOY_SCENARIO=os-odl_l2-sfc-noha
2017-02-16 01:18:36,150 - prepare_env - INFO -     CI_DEBUG=true
2017-02-16 01:18:36,150 - prepare_env - INFO - ===============================================
2017-02-16 01:18:36,150 - prepare_env - INFO - Creating needed directories...
2017-02-16 01:18:36,151 - prepare_env - INFO - ===============================================
2017-02-16 01:18:36,151 - prepare_env - INFO - Fetching RC file...
2017-02-16 01:18:36,151 - prepare_env - INFO - RC file not provided. Fetching it from the
installer...
fetch_os_creds.info: Verifying connectivity to 10.20.0.2...
fetch_os_creds.info: 10.20.0.2 is reachable!
fetch_os_creds.info: Fetching rc file from controller 10.20.0.7...
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.7' (ECDSA) to the list of known hosts.
fetch_os_creds.info: Exchanging keystone public IP in rc file to http://172.16.0.3:5000/v2.0
```

(Output omitted)

```
2017-02-16 01:18:45,693 - prepare_env - INFO - ===============================================
2017-02-16 01:18:45,693 - prepare_env - INFO - Verifying OpenStack services...
2017-02-16 01:18:45,700 - prepare_env - INFO - Checking OpenStack endpoints:
2017-02-16 01:18:45,707 - prepare_env - INFO - >>Verifying connectivity to the public endpoint
2017-02-16 01:18:45,716 - prepare_env - INFO -   ...OK
2017-02-16 01:18:47,764 - prepare_env - INFO - >>Verifying connectivity to the admin endpoint
2017-02-16 01:18:47,772 - prepare_env - INFO -   ...OK
2017-02-16 01:18:47,773 - prepare_env - INFO - Checking OpenStack basic services:
2017-02-16 01:18:47,777 - prepare_env - INFO - >>Checking openstack service...
2017-02-16 01:18:51,360 - prepare_env - INFO -   ...OK
2017-02-16 01:18:51,364 - prepare_env - INFO - >>Checking nova service...
2017-02-16 01:18:53,988 - prepare_env - INFO -   ...OK
2017-02-16 01:18:53,991 - prepare_env - INFO - >>Checking neutron service...
2017-02-16 01:18:55,975 - prepare_env - INFO -   ...OK
2017-02-16 01:18:55,980 - prepare_env - INFO - >>Checking glance service...
2017-02-16 01:18:57,808 - prepare_env - INFO -   ...OK
2017-02-16 01:18:57,812 - prepare_env - INFO - >>Checking cinder service...
2017-02-16 01:18:59,796 - prepare_env - INFO -   ...OK
2017-02-16 01:18:59,796 - prepare_env - INFO - OpenStack services are OK.
2017-02-16 01:18:59,796 - prepare_env - INFO - Checking External network...
2017-02-16 01:19:03,929 - prepare_env - INFO - External network found: b878795e-9095-4c0f-939d
2017-02-16 01:19:03,930 - prepare_env - INFO - ===============================================
2017-02-16 01:19:03,930 - prepare_env - INFO - Creating Rally environment...
```

(Output omitted)

```
keystone endpoints are valid and following services are available:
+-------------+----------------+-----------+
| services    | type           | status    |
+-------------+----------------+-----------+
| __unknown__ | alarming       | Available |
| __unknown__ | artifact       | Available |
| __unknown__ | compute_legacy | Available |
| __unknown__ | servicevm      | Available |
| __unknown__ | volumev2       | Available |
| __unknown__ | volumev3       | Available |
| ceilometer  | metering       | Available |
| cinder      | volume         | Available |
| cloud       | cloudformation | Available |
| glance      | image          | Available |
| heat        | orchestration  | Available |
| keystone    | identity       | Available |
```

```
| neutron     | network        | Available |
| nova        | compute        | Available |
| s3          | s3             | Available |
| swift       | object-store   | Available |
+-------------+----------------+-----------+
NOTE: '__unknown__' service name means that Keystone service catalog doesn't return name for this
service and Rally can not identify service by its type. BUT you still can use such services with
api_versions context, specifying type of service (execute `rally plugin show api_versions` for more
details).

Images for user `admin` in tenant `admin`:
+--------------------------------------+--------+-----------+
| UUID                                 | Name   | Size (B) |
+--------------------------------------+--------+-----------+
| e5cc46f1-bef2-4906-ad38-390e5d2a6516 | TestVM | 22581248 |
+--------------------------------------+--------+-----------+

Flavors for user `admin` in tenant `admin`:
+--------------------------------------+-----------+-------+----------+-----------+-----------+
| ID                                   | Name      | vCPUs | RAM (MB) | Swap (MB) | Disk (GB) |
+--------------------------------------+-----------+-------+----------+-----------+-----------+
| 1                                    | m1.tiny   | 1     | 512      | n/a       | 1         |
| 2                                    | m1.small  | 1     | 2048     | n/a       | 20        |
| 3                                    | m1.medium | 2     | 4096     | n/a       | 40        |
| 4                                    | m1.large  | 4     | 8192     | n/a       | 80        |
| 5                                    | m1.xlarge | 8     | 16384    | n/a       | 160       |
| 5693da34-4d29-4aec-85e9-5a4e70553d2c | m1.micro  | 1     | 64       | n/a       | 0         |
+--------------------------------------+-----------+-------+----------+-----------+-----------+
2017-02-16 01:19:18,860 - prepare_env - INFO - Functest environment installed.
```

Some timers in the main testscase script may need tweaking to accommodate for slower performing systems:

```
root@2e5bd681631e:~/repos/functest/testcases/features/sfc# grep -r 120 sfc.py
sfc.py:    time.sleep(120)

root@2e5bd681631e:~/repos/functest/testcases/features/sfc# vi sfc.py

root@2e5bd681631e:~/repos/functest/testcases/features/sfc# grep -r 240 sfc.py
sfc.py:    time.sleep(240)
```

The actual SFC testcase can now be invoked:

```
root@2e5bd681631e:~# functest testcase run odl-sfc
2017-02-16 01:34:09,760 - run_tests - INFO -
2017-02-16 01:34:09,760 - run_tests - INFO - ================================================
2017-02-16 01:34:09,760 - run_tests - INFO - Running test case 'odl-sfc'...
2017-02-16 01:34:09,761 - run_tests - INFO - ================================================
+=====================================================================+
| Testcase:  odl-sfc                                                  |
+=====================================================================+
| Description:                                                        |
|    Test suite for odl-sfc to test two chains and two SFs            |
| Criteria:  status == "PASS"                                         |
| Dependencies:                                                       |
|   - Installer:fuel                                                  |
|   - Scenario :odl_l2-sfc                                            |
|                                                                     |
+---------------------------------------------------------------------+

2017-02-16 01:34:09,761 - openstack_snapshot - INFO - Generating OpenStack snapshot...

Sourcing Credentials /home/opnfv/functest/conf/openstack.creds to run the test..
```

(Output omitted)

```
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.7' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
2017-02-16 01:34:40,037 - ODL_SFC - INFO - The presetup of the server worked
2017-02-16 01:34:40,205 - ODL_SFC - INFO - Executing ssh to collect the compute IPs
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.5' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.5' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.5' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.6' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.6' (ECDSA) to the list of known hosts.
```

```
Warning: Permanently added '10.20.0.2' (ECDSA) to the list of known hosts.
Warning: Permanently added '10.20.0.6' (ECDSA) to the list of known hosts.
2017-02-16 01:34:46,551 - ODL_SFC - INFO - Configuring iptables -P INPUT ACCEPT on contoller
2017-02-16 01:34:49,459 - ODL_SFC - INFO - Configuring iptables -t nat -P INPUT ACCEPT on contoller
2017-02-16 01:34:51,877 - ODL_SFC - INFO - Configuring iptables -A INPUT -m state
--state NEW,ESTABLISHED,RELATED -j ACCEPT on contoller
2017-02-16 01:34:54,386 - ODL_SFC - INFO - Downloading image
2017-02-16 01:38:35,021 - ODL_SFC - INFO - Using old image
2017-02-16 01:38:35,919 - openstack_utils - INFO - Creating flavor 'custom' with '1500' RAM, '10'
disk size, '1' vcpus...
2017-02-16 01:38:43,627 - openstack_utils - INFO - Creating image 'sf_nsh_colorado' from
'/home/opnfv/functest/data/sf_nsh_colorado.qcow2'...
2017-02-16 01:42:42,075 - openstack_utils - INFO - Creating neutron network example-net...
2017-02-16 01:42:50,058 - openstack_utils - INFO - Creating security group  'example-sg'...
```

(Output omitted)

```
2017-02-16 01:46:41,040 - ODL_SFC - INFO - Adding 'client' to security group example-sg
2017-02-16 01:46:42,629 - ODL_SFC - INFO - Creating instance 'server'...
 name=server
 flavor=custom
 image=d42664e2-e50f-4e27-9378-726093e7b303
 network=4635b1f4-09e9-44cd-8584-abbc18dbe330

2017-02-16 01:50:04,957 - ODL_SFC - INFO - Adding 'server' to security group example-sg
Created a new vnfd:
+---------------+-------------------------------------------------------------------------------+
| Field         | Value                                                                         |
+---------------+-------------------------------------------------------------------------------+
| description   | firewall1-example                                                             |
| id            | bdee6cc2-cec0-4dfa-a6fd-24445e066e9d                                          |
| infra_driver  | heat                                                                          |
| mgmt_driver   | noop                                                                          |
| name          | test-vnfd1                                                                    |
| service_types | {"service_type": "firewall1", "id": "800efd21-4ff4-41a6-b495-b93b64ff6b35"} |
|               | {"service_type": "vnfd", "id": "ba6ba48a-f52e-4b8a-a4e4-c51c6355273d"}      |
| tenant_id     | 7a619858c652488eba992c93e9f3d633                                              |
+---------------+-------------------------------------------------------------------------------+
Created a new vnfd:
+---------------+-------------------------------------------------------------------------------+
| Field         | Value                                                                         |
+---------------+-------------------------------------------------------------------------------+
| description   | firewall2-example                                                             |
| id            | b2c3c025-188f-4636-9550-ab764b0593a0                                          |
| infra_driver  | heat                                                                          |
| mgmt_driver   | noop                                                                          |
| name          | test-vnfd2                                                                    |
| service_types | {"service_type": "vnfd", "id": "672c897b-84da-49e6-ad5b-b16559c10533"}      |
|               | {"service_type": "firewall2", "id": "80a87f68-04b4-4eeb-85da-3432be0d526a"} |
| tenant_id     | 7a619858c652488eba992c93e9f3d633                                              |
+---------------+-------------------------------------------------------------------------------+
Created a new vnf:
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description | firewall1-example                    |
| id          | 1315fdac-9977-481c-8eb0-994472347798 |
| instance_id | a5f7e3e6-6b63-4ee2-b7db-bfc91df68a4d |
| mgmt_url    |                                      |
| name        | testVNF1                             |
| status      | PENDING_CREATE                       |
| tenant_id   | 7a619858c652488eba992c93e9f3d633     |
| vnfd_id     | bdee6cc2-cec0-4dfa-a6fd-24445e066e9d |
+-------------+--------------------------------------+
Created a new vnf:
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description | firewall2-example                    |
| id          | 3589bc12-4c35-4517-a20b-565367d7b799 |
| instance_id | 1115ef31-b91b-4a90-a815-0728d7e3b191 |
| mgmt_url    |                                      |
| name        | testVNF2                             |
| status      | PENDING_CREATE                       |
| tenant_id   | 7a619858c652488eba992c93e9f3d633     |
| vnfd_id     | b2c3c025-188f-4636-9550-ab764b0593a0 |
+-------------+--------------------------------------+
```

(Output omitted)

```
Created a new sfc:
+--------------+--------------------------------------+
| Field        | Value                                |
+--------------+--------------------------------------+
| attributes   | {}                                   |
| chain        | 1315fdac-9977-481c-8eb0-994472347798 |
```

```
| description  |                                      |
| id           | 8e4477da-5abc-47b8-a607-d2596216c001 |
| infra_driver | opendaylight                         |
| instance_id  | Path-red-Path-52                     |
| name         | red                                  |
| status       | PENDING_CREATE                       |
| symmetrical  | False                                |
| tenant_id    | 7a619858c652488eba992c93e9f3d633     |
+--------------+--------------------------------------+
Created a new sfc:
+--------------+--------------------------------------+
| Field        | Value                                |
+--------------+--------------------------------------+
| attributes   | {}                                   |
| chain        | 3589bc12-4c35-4517-a20b-565367d7b799 |
| description  |                                      |
| id           | c3e626b4-04a4-4549-b45e-6f3ed6621378 |
| infra_driver | opendaylight                         |
| instance_id  | Path-blue-Path-84                    |
| name         | blue                                 |
| status       | PENDING_CREATE                       |
| symmetrical  | False                                |
| tenant_id    | 7a619858c652488eba992c93e9f3d633     |
+--------------+--------------------------------------+
Created a new sfc_classifier:
+--------------------+------------------------------------------------------+
| Field              | Value                                                |
+--------------------+------------------------------------------------------+
| acl_match_criteria | {"source_port": 0, "protocol": 6, "dest_port": 80}   |
| attributes         | {}                                                   |
| chain              | 8e4477da-5abc-47b8-a607-d2596216c001                 |
| description        |                                                      |
| id                 | c84160e4-de9b-4329-93f1-405756bd5a94                 |
| infra_driver       | netvirtsfc                                           |
| instance_id        | red_http                                             |
| name               | red_http                                             |
| status             | PENDING_CREATE                                       |
| tenant_id          | 7a619858c652488eba992c93e9f3d633                     |
+--------------------+------------------------------------------------------+
Created a new sfc_classifier:
+--------------------+------------------------------------------------------+
| Field              | Value                                                |
+--------------------+------------------------------------------------------+
| acl_match_criteria | {"source_port": 0, "protocol": 6, "dest_port": 22}   |
| attributes         | {}                                                   |
| chain              | 8e4477da-5abc-47b8-a607-d2596216c001                 |
| description        |                                                      |
| id                 | 692c03ce-225e-4f2a-a8c2-0174ea93a0a0                 |
| infra_driver       | netvirtsfc                                           |
| instance_id        | red_ssh                                              |
| name               | red_ssh                                              |
| status             | PENDING_CREATE                                       |
| tenant_id          | 7a619858c652488eba992c93e9f3d633                     |
+--------------------+------------------------------------------------------+
+--------------------------------------+------+-------------+--------------+-------------+--------+
| id                                   | name | description | infra_driver | symmetrical | status |
+--------------------------------------+------+-------------+--------------+-------------+--------+
| 8e4477da-5abc-47b8-a607-d2596216c001 | red  |             | opendaylight | False       | ACTIVE |
| c3e626b4-04a4-4549-b45e-6f3ed6621378 | blue |             | opendaylight | False       | ACTIVE |
+--------------------------------------+------+-------------+--------------+-------------+--------+
+--------------------------------------+----------+-----------------------------------------+--------+
| id                                   | name     | acl_match_criteria                      | status |
+--------------------------------------+----------+-----------------------------------------+--------+
| 692c03ce-225e-4f2a-a8c2-0174ea93a0a0 | red_ssh  | { u'protocol': 6, u'dest_port': 22}     | ACTIVE |
| c84160e4-de9b-4329-93f1-405756bd5a94 | red_http | { u'protocol': 6, u'dest_port': 80}     | ACTIVE |
+--------------------------------------+----------+-----------------------------------------+--------+
2017-02-16 01:51:32,556 - ODL_SFC - INFO - Instance name and ip ta-bc12-4c35-4517-a20b-565367d7b799-
vdu1-vy3iwu7tr76w:172.16.0.134
2017-02-16 01:51:32,556 - ODL_SFC - INFO - Waiting for instance ta-bc12-4c35-4517-a20b-565367d7b799-
vdu1-vy3iwu7tr76w:172.16.0.134 to come up
2017-02-16 01:51:39,585 - ODL_SFC - INFO - SF:172.16.0.134 is reachable
2017-02-16 01:51:43,834 - ODL_SFC - INFO - Instance name and ip ta-fdac-9977-481c-8eb0-994472347798-
vdu1-rox2f3fchptu:172.16.0.135
2017-02-16 01:51:43,835 - ODL_SFC - INFO - Waiting for instance ta-fdac-9977-481c-8eb0-994472347798-
vdu1-rox2f3fchptu:172.16.0.135 to come up
2017-02-16 01:51:45,513 - ODL_SFC - INFO - classification rules updated
2017-02-16 01:51:45,514 - ODL_SFC - INFO - It took 19.2132349014 seconds
2017-02-16 01:51:47,380 - ODL_SFC - INFO - SF:172.16.0.135 is reachable
2017-02-16 01:51:50,758 - ODL_SFC - INFO - Instance name and ip server:172.16.0.136
2017-02-16 01:51:50,760 - ODL_SFC - INFO - Waiting for instance server:172.16.0.136 to come up
2017-02-16 01:51:54,564 - ODL_SFC - INFO - Server:172.16.0.136 is reachable
2017-02-16 01:51:58,693 - ODL_SFC - INFO - Instance name and ip client:172.16.0.137
2017-02-16 01:51:58,696 - ODL_SFC - INFO - Waiting for instance client:172.16.0.137 to come up
2017-02-16 01:52:03,522 - ODL_SFC - INFO - Client:172.16.0.137 is reachable
2017-02-16 01:52:03,523 - ODL_SFC - INFO - Checking SSH connectivity to the SFs with ips
[u'172.16.0.135', u'172.16.0.134']
2017-02-16 01:52:05,736 - ODL_SFC - INFO - SSH connectivity to the SFs established
2017-02-16 01:52:05,737 - ODL_SFC - INFO - Starting HTTP server on 172.16.0.136
```

```
2017-02-16 01:52:06,781 - ODL_SFC - INFO - Starting HTTP firewall on 172.16.0.134
2017-02-16 01:52:07,177 - ODL_SFC - INFO - Starting SSH firewall on 172.16.0.135
2017-02-16 01:52:07,596 - ODL_SFC - INFO - Wait for ODL to update the classification rules in OVS
2017-02-16 01:56:07,695 - ODL_SFC - INFO - Test SSH
2017-02-16 01:56:13,782 - ODL_SFC - INFO - nc: connect to 11.0.0.4 port 22 (tcp) timed out: Operation
now in progress
2017-02-16 01:56:13,784 - ODL_SFC - INFO - TEST 1 [PASSED] ==> SSH BLOCKED
2017-02-16 01:56:13,784 - ODL_SFC - INFO - Test HTTP
2017-02-16 01:56:14,182 - ODL_SFC - INFO - Connection to 11.0.0.4 80 port [tcp/http] succeeded!
2017-02-16 01:56:14,183 - ODL_SFC - INFO - TEST 2 [PASSED] ==> HTTP WORKS
2017-02-16 01:56:14,183 - ODL_SFC - INFO - Changing the classification
Deleted sfc_classifier: red_http
Deleted sfc_classifier: red_ssh
Created a new sfc_classifier:
+--------------------+------------------------------------------------------+
| Field              | Value                                                |
+--------------------+------------------------------------------------------+
| acl_match_criteria | {"source_port": 0, "protocol": 6, "dest_port": 80}   |
| attributes         | {}                                                   |
| chain              | c3e626b4-04a4-4549-b45e-6f3ed6621378                 |
| description        |                                                      |
| id                 | 3f3a631c-07af-4ca9-bdab-83e6fa8504f8                 |
| infra_driver       | netvirtsfc                                           |
| instance_id        | blue_http                                            |
| name               | blue_http                                            |
| status             | PENDING_CREATE                                       |
| tenant_id          | 7a619858c652488eba992c93e9f3d633                     |
+--------------------+------------------------------------------------------+
Created a new sfc_classifier:
+--------------------+------------------------------------------------------+
| Field              | Value                                                |
+--------------------+------------------------------------------------------+
| acl_match_criteria | {"source_port": 0, "protocol": 6, "dest_port": 22}   |
| attributes         | {}                                                   |
| chain              | c3e626b4-04a4-4549-b45e-6f3ed6621378                 |
| description        |                                                      |
| id                 | 20ee26b7-f337-4b62-93d5-9b19f99fc7f0                 |
| infra_driver       | netvirtsfc                                           |
| instance_id        | blue_ssh                                             |
| name               | blue_ssh                                             |
| status             | PENDING_CREATE                                       |
| tenant_id          | 7a619858c652488eba992c93e9f3d633                     |
+--------------------+------------------------------------------------------+
+--------------------------------------+-----------+-------------------------------------------------+
| id                                   | name      | acl_match_criteria                              | status |
+--------------------------------------+-----------+-------------------------------------------------+
| 20ee26b7-f337-4b62-93d5-9b19f99fc7f0 | blue_ssh  | { u'protocol': 6, u'dest_port': 22}   | ACTIVE |
| 3f3a631c-07af-4ca9-bdab-83e6fa8504f8 | blue_http | { u'protocol': 6, u'dest_port': 80}   | ACTIVE |
+--------------------------------------+-----------+-------------------------------------------------+
2017-02-16 01:56:26,301 - ODL_SFC - INFO - Wait for ODL to update the classification rules in OVS
2017-02-16 01:56:43,146 - ODL_SFC - INFO - classification rules updated
2017-02-16 01:56:43,146 - ODL_SFC - INFO - It took 16.8441061974 seconds
2017-02-16 01:58:06,402 - ODL_SFC - INFO - Test HTTP
2017-02-16 01:58:11,943 - ODL_SFC - INFO - nc: connect to 11.0.0.4 port 80 (tcp) timed out: Operation
now in progress
2017-02-16 01:58:11,943 - ODL_SFC - INFO - TEST 3 [PASSED] ==> HTTP Blocked
2017-02-16 01:58:11,944 - ODL_SFC - INFO - Test SSH
2017-02-16 01:58:12,455 - ODL_SFC - INFO - Connection to 11.0.0.4 22 port [tcp/ssh] succeeded!
2017-02-16 01:58:12,455 - ODL_SFC - INFO - TEST 4 [PASSED] ==> SSH Works
2017-02-16 01:58:12,474 - ODL_SFC - INFO - SFC ALL TESTS: PASS :)
```