

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

---

SCHOOL OF ENGINEERING AND ARCHITECTURE

MASTER'S DEGREE

IN

TELECOMMUNICATIONS ENGINEERING

INTEGRATION OF SDN FRAMEWORKS  
AND CLOUD COMPUTING PLATFORMS:  
AN OPEN SOURCE APPROACH

*Master Thesis*

*in*

Telecommunication Networks Laboratory M

*Supervisor*

Prof. WALTER CERRONI

*Candidate*

FRANCESCO FORESTA

*Co-supervisors*

Prof. FRANCO CALLEGATI

Prof. LUCA FOSCHINI

---

SESSION II

ACADEMIC YEAR 2016/2017



*To my family,  
for all the support*



# Contents

<b>Sommario</b>	<b>ix</b>
<b>Abstract</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 SDN, NFV and Cloud overview</b>	<b>5</b>
2.1 Computer Networks: today . . . . .	6
2.1.1 Service-oriented approach . . . . .	6
2.2 Software Defined-Networking . . . . .	9
2.2.1 The SDN controller . . . . .	11
2.2.2 NETCONF . . . . .	12
2.2.3 P4 . . . . .	13
2.2.4 OpenFlow . . . . .	13
2.3 Network Functions Virtualization . . . . .	16
2.3.1 The NVF-MANO framework . . . . .	20
2.4 Cloud computing . . . . .	21
2.4.1 Before cloud computing . . . . .	21
2.4.2 The Cloud paradigm . . . . .	22
2.5 SDN in NFV architectures . . . . .	25
2.5.1 SDN usage in NFV architectural framework . . . . .	25
2.5.2 SDN for dynamic Service Function Chaining . . . . .	30
<b>3 The OpenStack platform</b>	<b>37</b>
3.1 OpenStack logical overview . . . . .	39
3.1.1 OpenStack deployments . . . . .	41
3.1.2 OpenStack nodes . . . . .	41
3.2 OpenStack components . . . . .	43

3.2.1	Identity service: Keystone . . . . .	43
3.2.2	Computing service: Nova . . . . .	45
3.2.3	Object Storage service: Swift . . . . .	47
3.2.4	Image service: Glance . . . . .	49
3.2.5	Block Storage service: Cinder . . . . .	50
3.2.6	Networking service: Neutron . . . . .	51
3.2.7	Other OpenStack services . . . . .	53
<b>4</b>	<b>Neutron networking: details and improvements</b>	<b>57</b>
4.1	Neutron abstractions . . . . .	57
4.2	Networks and multi-tenancy . . . . .	59
4.3	OpenStack VNI . . . . .	60
4.3.1	Compute node . . . . .	63
4.3.2	Network node . . . . .	72
4.4	Remarks . . . . .	77
4.5	Performance and improvements . . . . .	78
4.5.1	OvS native firewall . . . . .	79
4.5.2	OvS learn action firewall . . . . .	87
4.5.3	OvS native interface . . . . .	89
<b>5</b>	<b>SDN controllers overview</b>	<b>91</b>
5.1	Ryu framework . . . . .	91
5.1.1	OpenStack networking with Ryu . . . . .	92
5.2	OpenDaylight . . . . .	94
5.2.1	OpenStack networking with ODL . . . . .	94
5.3	Open Networking Operating System: ONOS . . . . .	96
5.3.1	OpenStack networking with ONOS: SONA . . . . .	97
5.3.2	Security groups in SONA . . . . .	101
<b>6</b>	<b>SONA implementation and tests</b>	<b>105</b>
6.1	ONOS deployment . . . . .	106
6.2	OpenStack development infrastructure deployment . . . . .	107
6.3	L3 integration . . . . .	108
6.3.1	Gateway node, vRouter and BGP . . . . .	109
6.4	Testing SONA: walkthrough . . . . .	112
6.4.1	L2 Switching functionalities . . . . .	112
6.4.2	L3 Routing functionalities . . . . .	113
6.5	Case study: Orchestration . . . . .	114

6.5.1	Orchestration in OpenStack . . . . .	114
6.5.2	Glossary of OpenStack orchestration . . . . .	114
6.5.3	SONA approach with Heat . . . . .	115
6.5.4	Orchestration case study: Telegram bot template . . . . .	115
6.6	Case study: Gathering information . . . . .	117
6.6.1	Using ONOS to monitor OpenStack . . . . .	117
6.6.2	Case study: the sonaMonitor script . . . . .	117
6.7	Limitations and future works . . . . .	119
<b>7</b>	<b>Testbed implementation</b>	<b>121</b>
7.1	OpenStack production environment deployment . . . . .	121
7.1.1	Topology . . . . .	121
7.1.2	Prerequisites . . . . .	122
7.1.3	Identity service: Keystone . . . . .	124
7.1.4	Image service: Glance . . . . .	124
7.1.5	Computing service: Nova . . . . .	124
7.1.6	Networking service: Neutron . . . . .	125
7.1.7	Dashboard service: Horizon . . . . .	127
7.1.8	Orchestration service: Heat . . . . .	127
7.1.9	Telemetry service: Ceilometer . . . . .	127
<b>8</b>	<b>Performance evaluation</b>	<b>131</b>
8.1	In-node process throughput . . . . .	133
8.2	Latency . . . . .	133
8.3	Co-located instances measurements . . . . .	135
8.3.1	TCP throughput evaluation . . . . .	136
8.3.2	CPU evaluation . . . . .	138
8.3.3	Memory evaluation . . . . .	141
8.3.4	UDP Packet rate sustainability . . . . .	142
8.4	Externally located instances measurements . . . . .	148
8.4.1	UDP Packet rate sustainability . . . . .	148
8.5	Final remarks . . . . .	152
<b>9</b>	<b>Conclusions</b>	<b>153</b>
<b>A</b>	<b>Additional remarks</b>	<b>155</b>
A.1	SONA configuration files . . . . .	155
A.1.1	SONA network configuration . . . . .	155

A.1.2	SONA cell file . . . . .	156
A.1.3	ONOS ML2 configuration . . . . .	157
A.1.4	DevStack configuration for the testbed . . . . .	157
A.1.5	SONA vRouter configuration . . . . .	159
A.1.6	Docker BGP configuration . . . . .	160
A.2	SONA monitor code . . . . .	161
A.2.1	Monitor . . . . .	161
A.2.2	Statistics subclass . . . . .	166
A.2.3	Hosts subclass . . . . .	168
A.2.4	Flows subclass . . . . .	171
A.3	YAML template for a Telegram bot . . . . .	174
A.4	OpenStack configuration files . . . . .	177
A.4.1	Chrony . . . . .	177
A.4.2	MariaDB configuration . . . . .	178
A.4.3	Memcached . . . . .	178
A.4.4	Keystone . . . . .	178
A.4.5	HTTP server . . . . .	178
A.4.6	Glance . . . . .	180
A.4.7	Nova . . . . .	181
A.4.8	Neutron . . . . .	184
A.4.9	Load Ryu applications from Neutron . . . . .	187
A.4.10	Dashboard . . . . .	188
A.4.11	Heat . . . . .	195
A.4.12	Ceilometer . . . . .	196
	<b>Acknowledgments</b>	<b>199</b>
	<b>Bibliography</b>	<b>203</b>



# Sommario

L'aumento del numero di servizi offerti via Internet sta portando ad un incremento esponenziale del traffico nelle Reti di Telecomunicazioni. Di conseguenza, gli operatori di Telecomunicazioni e i fornitori di servizi stanno ricercando soluzioni innovative mirate ad una gestione efficiente e trasparente di questo traffico al fine di garantire la corretta fruizione dei propri servizi. La soluzione più percorribile è quella di modificare i paradigmi nel networking, l'introduzione del Software Defined Networking porterà una maggiore dinamicità in tutti gli aspetti di gestione della rete. Contemporaneamente, la Network Functions Virtualization giocherà un ruolo fondamentale consentendo di virtualizzare i nodi intermedi che implementano le funzioni di rete su di un hardware generico. L'ambiente più idoneo a sfruttare questi paradigmi è il Cloud, dove le risorse (es. potenza computazionale, ecc.) sono fornite all'utente come un servizio, su richiesta, pagando solo l'effettivo utilizzo.

In termini di requisiti però, le attuali Reti di Telecomunicazioni non sono comunque abbastanza performanti, nonostante i miglioramenti hardware: una delle cause è da imputare all'assenza di una forte integrazione tra i sopra citati paradigmi, al fine di interagire reattivamente alle necessità degli utenti. È pertanto necessario valutare soluzioni dove SDN e NFV cooperano attivamente all'interno del Cloud per fornire interessanti prestazioni.

In questo documento, dopo una descrizione dello stato dell'arte, verrà affrontato lo studio della piattaforma cloud OpenStack e di come possa essere configurata per incrementarne le prestazioni di rete. Verranno poi integrati diversi framework SDN Open Source con la suddetta piattaforma. Infine, verranno mostrate alcune misure di performance che dimostrano come questo approccio possa essere utile a gestire e migliorare la Qualità di Servizio degli utenti.



# Abstract

As a result of the explosion in the number of services offered over the Internet, network traffic has experienced a remarkable increment and is supposed to increase even more in the few next years. Therefore, Telco operators are investigating new solutions aiming at managing this traffic efficiently and transparently to guarantee the users the needed Quality of Service.

The most viable solution is to have a paradigm shift in the networking field: the old and legacy routing will be indeed replaced by something more dynamic, through the use of Software Defined Networking. In addition to it, Network Functions Virtualization will play a key role making possible to virtualize the intermediate nodes implementing network functions, also called middle-boxes, on general purpose hardware. The most suitable environment to understand their potentiality is the Cloud, where resources, as computational power, storage, development platforms, etc. are outsourced and provided to the user as a service on a pay-per-use model. All of this is done in a complete dynamic way, as a result of the presence of the implementation of the above cited paradigms.

However, whenever it comes to strict requirements, Telecommunication Networks are still underperforming: one of the cause is the weak integration among these paradigms to reactively intervene to the users' need. It is therefore remarkably important to properly evaluate solutions where SDN and NFV are cooperating actively inside the Cloud, leading to more adaptive systems.

In this document, after the description of the state of the art in networking, the deployment of an OpenStack Cloud platform on an outperforming cluster will be shown. In addition, its networking capabilities will be improved via a careful cloud firewalling configuration; moreover, this cluster will be integrated with Open Source SDN frameworks to enhance its services. Finally, some measurements showing how much this approach could be interesting will be provided.



# Chapter 1

## Introduction

In the last decade, the pervasiveness of services based on Information and Communication Technologies (ICT) has completely reshaped people's lifestyle, and nowadays the whole human society heavily relies on the distribution and use of information. Telecommunications, in particular, have evolved in different directions. The state-of-the-art technologies are indeed able to exploit more and more the physical resources in an efficient way to provide a better Quality of Service (QoS) to the user. It is the case, for instance, of the Beamforming technique, with which it is possible to experience constructive interference achieving spatial selectivity. Furthermore, another technique which is showing a noteworthy improvement in hardware performance is MIMO, able to multiply the capacity of a radio link; this is achieved using multiple antennas both in transmission and reception. Even though there are many more interesting techniques for the radio sector, it is worth to mention also the use of always more outperforming optical fibers that are able to provide very high peak transfer rates.

All of the above will be included in the physical layer of one of the key topics now under study: the 5G standard, that is going to become the fifth generation of mobile networks [1]. This will not be just the next step after the 4G: it will also present a more radical approach that is not including only radio. Indeed, fronthaul and backhaul infrastructures will change in turn, as well as the way to efficiently manage them. 5G will therefore be mainly designed taking into consideration strict service requirements that are currently not coped properly [2], e.g. the latency which is fundamental for the M2M paradigm.

On the other hand, a deep study on the architecture design must be kept

into account in order to satisfy them. Indeed, to be compliant with the requirements, some additional studies have to be performed at the upper layers of the protocol stack as a result of their actual inefficiency.

In fact, the radical change that Telecommunications are facing goes toward even other aspects: the ever-increasing merging of Computing (or Information) Technologies into the Communication Technologies provided many interesting tools that are the enablers for innovative technologies. This is for instance the case of the introduction of virtualization inside the Telco world, as proposed by the NFV paradigm. This might be considered also in addition to the Cloud computing paradigm, which states that the computing can be delivered as a service, as it is already for water, electricity, etc. Indeed, cloud computing is designed to provide a compute resource, e.g. a virtual machine, as a utility rather than having to build and maintain computing ad-hoc infrastructures. As a result, it will also implement some techniques to grant the implementation of a *Resources-as-a-Service* approach in an elastic way. Cloud will then be at the base of the 5G, for instance with the Cloud-RAN approach: a centralized, cloud computing-based architecture for radio access networks that supports the current and the future wireless communication standards. It is also noteworthy that more and more everyday devices are equipped with very powerful computing capabilities making possible to shift to new paradigms such as mobile and edge computing. This, in addition to what has been stated above and other paradigms like Internet of Things, is leading to the Fog computing.

Therefore, ICT provided some interesting tools to be used in enabling future technologies; these are required to be compliant with the new strict requirements. It is important to remark that currently the physical layer improvements are moving forward with very important results; however, the presence of protocols that may be old and legacy ruling the other layers leads to an outstanding overhead. This causes to not have a proper exploitation of the cited improvements.

Everything that has been stated requires a paradigm shift in terms of control and resource management (both computing and communication). In particular, there is the need to have more elasticity and programmability of the infrastructure, with the consequent need of orchestrating heterogeneous resources, that by now are related to different worlds (cloud and network). Accordingly, Software Defined Networking has been introduced by the Telco operators to face this issue and make possible to reduce the overhead with both proactive and reactive actions. Indeed, SDN moves the intelligence of

the network from the devices to a logically centralized entity (then physically replicated to avoid to have a single point of failure). This allows the network to dynamically behave giving the administrators the possibility to rearrange it easily and elastically. Moreover, it is possible to implement some novel resource management techniques at a minimal cost, like the intent approach [3]. Finally, the integration of SDN into different environments, including the cloud computing one, leverages the possibility to scale out accurately.

Besides that, it is important to realize that Telco operators are no longer in the lead situation in which they were a few years ago: they are now suffering from the presence of new actors, called Over-the-Top (e.g. Facebook, Google, Netflix), in the market they were leading. The last trends are showing how much the influence of OTTs is conditioning the market; in particular, their explosion let the Telco operators be aware that their traditional business (voice dominant) was now no longer valid due to the exponential increase of data services. Under such scenario, the operators' business has to be dramatically transformed. On the one hand, the enormous amount of data generated by these OTTs are causing the necessity of investments on networks, inducing an exponential increase of operational costs: these are reflecting also the necessity of resolving the users' requests to be able to access the content providers (OTTs) with a certain QoS. On the other hand, the Telco revenues are decreasing due to a substantial number of factors, for instance the competition among the operators themselves. In particular, it is easy to see how much the OTTs are pushing the operators to start designing and developing the future infrastructures, even from the scratch, to grant an improved service to their users (e.g. 5G). However, the operators are more careful in targeting smaller improvements of already implemented technologies, that require lower infrastructural costs (e.g. 4.5G). To maintain a proper amount of revenues some operators diversified their business: this is the case of British Telecom becoming also a TV content provider. Nevertheless, the new requirements have still to be addressed and, due to the cited financial situation, possibly at a low cost.

Thus, the introduction of Software Defined Networking strategies in a NFV environment (as the cloud) at apparently no cost is a win-win for the operators, becoming a plus point for the future 5G deployment, as a result of the introduction of the ability of dynamically scale out, by properly managing and replicating resources. Furthermore, this is also becoming an enabler for all those new applicative approaches introduced in the last few years: for instance, the microservices are introducing a fine-grained architecture to define

services in a lightweight way; SDN is able to take advantage of it.

However, Telcos are not the only actors interested in this joint SDN-NFV approach: indeed, this can bring several benefits and open opportunities to other players, such as infrastructure providers but also service providers and enterprise networks. In particular, it is possible to see that these players could take advantages of flexibility introduced by both SDN and NFV, somehow generating new types of business and leverage well matured virtualization technologies already in use in data centers networks [4].

In order to have such dynamic integration and orchestration of all the heterogeneous resources that are going to be present in the future networks, SDN and NFV should be strongly coordinated. However, this is by now only partially obtained but still not optimized, neither considered in a general environment but just for some specific case studies. This document will therefore cope with these open issues: in particular, the study will be mainly on the use of SDN controllers, as ONOS and Ryu, integrated with the OpenStack platform to provide an evaluation of the performance. In addition to that, the performance evaluation will be also based on the different possible OpenStack configuration, which will be analytically evaluated.

This document is structured as follows. In Chapter 2 a more detailed description about SDN, NFV and Cloud is presented, as well as the reference architectural framework proposed by ETSI. Moreover, it also presents an overview about the design of the Control Plane of SDN controllers to perform Service Function Chaining.

Chapters 3 and 4 propose a complete description of the OpenStack platform: the focus will be put on its networking mechanisms.

A wide overview of the existing SDN frameworks is presented in 5; this will also present the different ways in which these can be integrated inside the OpenStack platform. This is followed by Chapter 6, where the solution based on ONOS, SONA, is deployed and some case studies are shown.

In Chapter 7 the deployment of the testbed of OpenStack and Ryu is introduced, showing which are the criteria used to set up the different platforms in the most efficient way. The results are then shown in Chapter 8.

Finally, Chapter 9 will provide some general conclusion and hints on where and how to improve the followed approach to obtain even better performance, in particular with a reference to the case study.



# Chapter 2

## SDN, NFV and Cloud overview

As shown in Chapter 1, there is the need to solve the open research problems related to the noticeably strict service-related requirements that will be present in future Telecommunications standards. This is noteworthy in particular for Telco operators, as well as many other players, as the service providers, which are more than interested in it.

Indeed, for a service provider this approach is important from a double perspective:

1. to make sure that the network operators' subscribers are able to consume the provided service with the correct QoS;
2. to generate new business models, regarding the possibility to provide services in a not-on-premises fashion.

One important step to achieve all of this could be through the introduction of Software Defined Networking techniques and Network Functions Virtualization; therefore, to be efficient the use of Distributed Cloud computing and other approaches, like microservices, should be considered as well.

This Chapter therefore will provide a broad view of the new network paradigms that might lead to a more complete and efficient on-demand service provisioning, by respecting the users' QoS, both proactively and reactively whenever this is not compliant with their SLA.

## 2.1 Computer Networks: today

### 2.1.1 Service-oriented approach

Since the beginning of the spread of the client-server approach, the concept of *service* (i.e. what is offered from a certain provider to possibly many users) has become essential in computer networks and, in general, in the Internet. Modern distributed systems are deeply pervasive and massively used to support any business aspect and strategy, making the service requirements a crucial topic. For instance, a service requirement might be the scalability (i.e. the presence of multiple users contemporarily in the system does not have an impact on the performance) or to have a short service time, maybe renouncing to deliver a correct answer back.

In general, it is customary to refer to a “measure” of how the service is provided, which is the Quality of Service. This does not have a standard way to be specified, but in general in the Telecommunications environment some *service levels* are defined, i.e. analytical indicators such as throughput, delay, jitter, lost packets, etc. Moreover, it is important to consider that the QoS may also be subjective and therefore dependent on the user: for instance, user Lucy might complain about an online video with few Motion Vector displacements, whereas user Johnny might not.

The relationship among different subjects requesting and providing services is expressed by the Service Level Agreement (SLA). Neither the SLA has a standard form, but it can be easily tailored by the use of service levels to be defined.

Therefore, in this scenario services are extremely pervasive. As a consequence, the enabling abstract infrastructure known as the Service Oriented Architecture (SOA) [5] replaced the simple Client/Server by overcoming its limitations. In SOA all the interactions among parties are considered in terms of services. The base of the SOA is the use of the *interface*, which is the decoupling element that hides the internal technicalities of the service itself. Thus, all the providers will register themselves to a third entity called the Discovery Agency where the requestor can perform a lookup to choose dynamically the wanted service. This is possibly done in a transparent way through the use of a *middleware*, i.e. a software entity that stays in between the application and the low-level support, suitable for deeply heterogeneous organizations which are interested in providing very differentiated services. Therefore, as distributed

systems are very heterogeneous, the SOA approach is noteworthy as it is able to provide the strong abstraction that they require to properly provide their services.

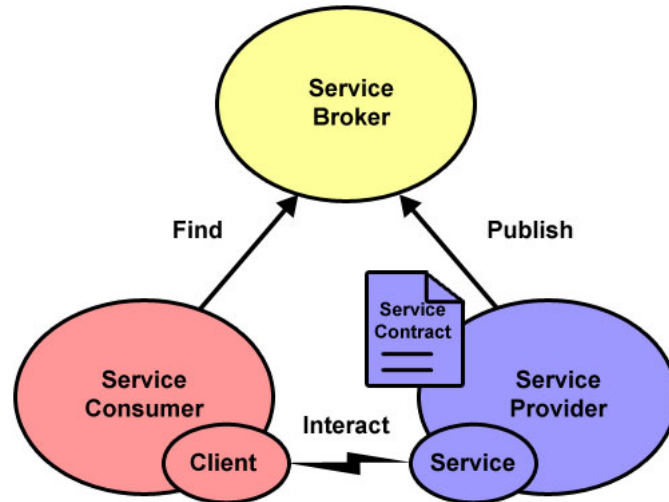


Figure 2.1: Service Oriented Architecture

Moving from a C/S approach to a less coupled one has also been considered: this approach is mandatory whenever it comes to situations where Multicast has to be implemented. In that case message exchange models have been introduced, which are nowadays widely used. In this way, it is possible to have multiple producers of services and multiple consumers of the same services, by still letting all of them to not be tied one to the others.

In addition to that, for many services it is important to mention the necessity to isolate a community of nodes that might be in different locations and networks. Thus, in order to build efficient and scalable networks the solution is the Overlay Networks approach, where networks are built at the application level rather than at the network level. An interesting characteristic is that these networks can be re-organized reactively (e.g. a node has failed, start a recovery procedure). This is convenient to grant that the QoS is always compliant with the agreed SLA. This is the case of Peer-to-Peer applications, as well as modern Distributed File Systems (e.g. Google FS - GFS [6] and Hadoop FS - HDFS [7]) and even non-relational DataBases as Apache Cassandra [8].

In general, apart from some particular cases, the requirements for an Inter-

net service are usually well defined: in particular, the service has to be highly available, but it has also to provide the required answer in a short time. The Brewer's CAP theorem [9] states that it is possible to have just two of the following three characteristic of a system together: strong consistency, high availability and partition tolerance. Therefore, as long as the systems have to scale well and be partition-tolerant while being highly available, the consistency is sacrificed. Thus, for modern services it is better to be eventual consistent and therefore to deliver a wrong answer (as an error) than to have the user waiting for a long amount of time. However, this lack of consistency means that in case of a failure, there is a certain probability for the system to fall in a non-consistent state. On the other hand, these events are kept under control by strongly reducing their happening and/or recovering easily through the use of particular strategies.

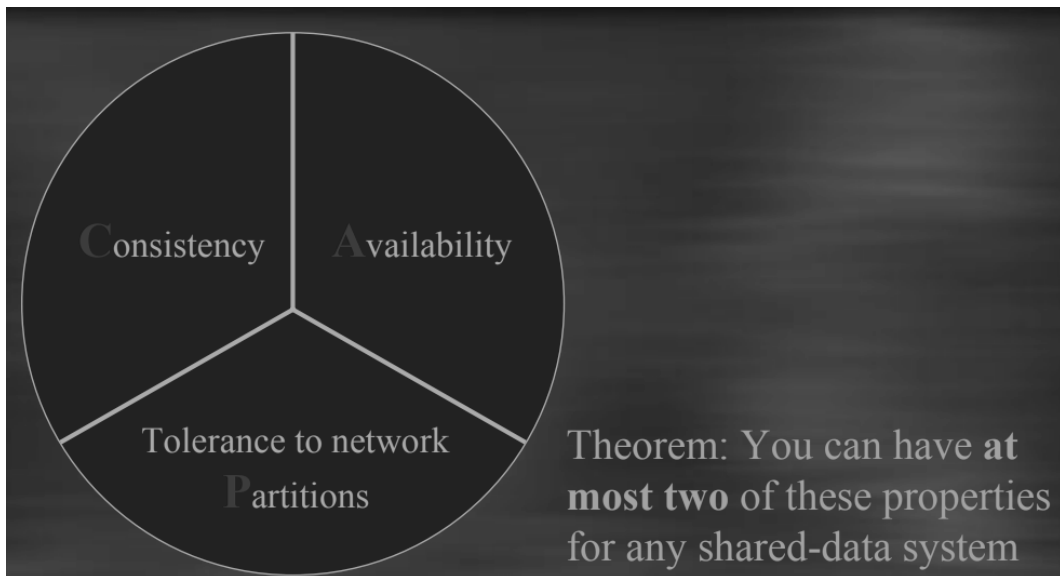


Figure 2.2: Brewer's CAP Theorem [9]

A practical example for this law is shown in non-relational databases (e.g. MongoDB [10]) and cloud computing platforms. In particular, cloud computing platforms can be seen as composed by two tiers: an external one, that is dedicated to the delivering of fast answers to the user, even though they might be wrong, and an inner one where the important data is stored in a consistent way.

Finally, to have services which are more lightweight and easier to be migrated, a new approach hit the market: the microservices approach. A microservice is a small fine-grained component capable of being hosted everywhere its containing environment (the container) is present and easy to be managed. The container already provides all the basic functionalities for the service, for instance the low-level communication details to exchange data with the other entities. As a result of this approach, the service developer has just to concentrate on its logic and not on the management, enlarging the lifecycle of the service itself. Indeed, it is easy to migrate, clone, start, stop the services as all the functions are taken by the container. A popular tool and language for microservices is the well known Docker [11], but even other solutions are present (e.g. Jolie [12]).

## 2.2 Software Defined-Networking

SDN was first standardized in 2011 by the Open Networking Foundation (ONF), “*a user-driven organization dedicated to the promotion and adoption of SDN, and implementing SDN through open standards, necessary to move the networking industry forward.*” [13]. The ONF is the entity behind the standardization of the OpenFlow protocol, one of the most used approaches to perform SDN.

The scope of the SDN paradigm is to support a more dynamic and scalable telecommunication networks environment. This is achieved through the decoupling of the control plane (i.e. the routing, which decides how to forward packets) from the data plane (i.e. the forwarding, which receives, stores and forwards the packets) [14].

Therefore, it is possible to say that SDN attempts to:

- make flow tables controllable from the outside via a common standardized API (Application Programming Interface), not tied to any vendor  $\Rightarrow$  easier *programmability* of the network;
- deal with a programming model, imposed by the particular framework used;
- concentrate the control (i.e. the network intelligence) in a logically centralized entity, eventually physically distributed to avoid to have a single

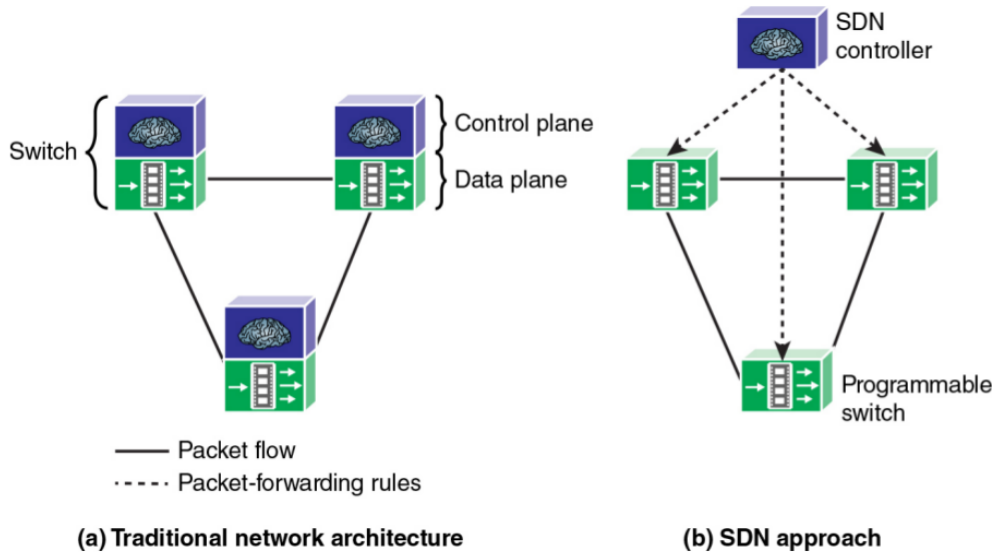


Figure 2.3: Traditional vs. SDN networking [15]

point of failure; this makes possible to maintain a global view of the network that will appear to applications as a single logical entity.

By separating the intelligence of the network from the datapath, it is possible to give to operators, developers, etc. the interface to program the network itself. Therefore, through the a-priori installation of *rules* in the switches, it is possible to implement a proactive way to implement decisions in the network; however, if the switch is not able to decide over a certain packet (in general, it is possible to say over a certain header), the centralized control entity (the *controller*) can be reactively reached to determine what to do.

The flow is actually a very general consideration: indeed, a flow may just include a certain application flow (e.g. all BitTorrent traffic), as well as all the traffic related to a certain protocol (e.g. all ICMP traffic); in a more fine-grained view, it is possible also to consider it as the traffic of a certain user.

Then, to each flow there might be a certain action to be taken: for instance, this might be the redirection of traffic over a certain node or even the tagging of a packet within a certain VLAN to isolate it.

It is remarkable to say that the SDN is revolutionary up to a certain degree:

in particular, it is possible to see similarities in terms of routing with Multi Protocol Label Switching (MPLS), even though the real plus point that SDN proposes is the standardized interface that allows an easier programmability. On the other hand, due to similarities, many operators are choosing to deploy MPLS over SDN to take the best out of the two.

Moreover, the centralized control is actually something that is not stated to be necessarily part of SDN: however, its introduction fosters a more global view of the network by the viewpoint of the network applications, which is something innovative and useful.

Finally, SDN has not to be thought as limited just to specific networks, as data centers, edge networks or cloud networks: although it is not assured that SDN will be a guaranteed market success, the aim of operators is to put SDN in production even in legacy networks.

### 2.2.1 The SDN controller

As previously stated, the SDN controller is a logically centralized, software-based entity that manages network devices operating in the data plane; it takes advantage of its global view over the network for running applications aimed at management, security and optimization of the resources it controls.

As is shown in Figure 2.4, the SDN controller can be logically placed in the *Control Plane* located between the *Data Plane*, where network devices operate the actual packet forwarding, and the *Application Plane*, where SDN applications request specific services from the underlying infrastructure, based on the network state or on specific events. However, in order to communicate with the controller, proper interfaces must be defined. The interface between Application and Control planes is usually referred to as the *Northbound Interface* (NBI), while the one between Control and Data planes, called SDN Control-Data-Plane Interface in Figure 2.4, can also be referred to as the *Southbound Interface* (SBI). Both interfaces can be specified and designed to use any compatible communication protocol.

Software Defined Networking is just a paradigm. During the past years many different solutions (proprietary or not) have come out, trying to acquire a slice of market; one of the most widely-used protocol for Controller-Device communication through the SBI is the OpenFlow protocol, but other interesting solutions are also present.

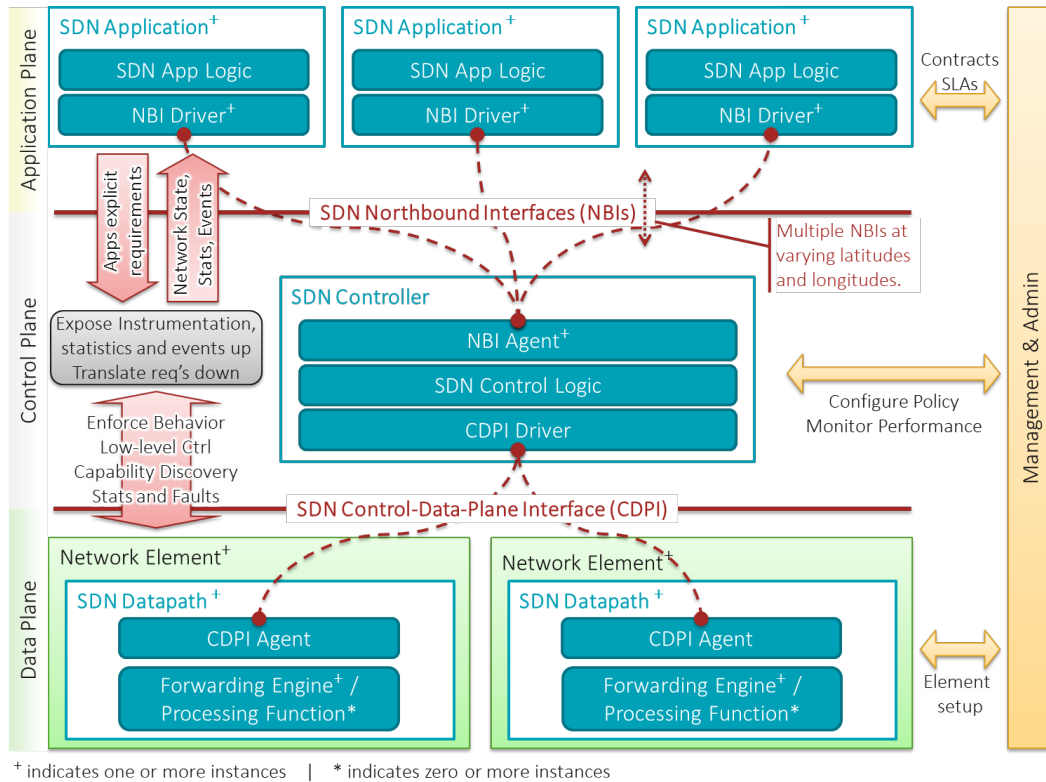


Figure 2.4: The role of the SDN controller in the SDN architecture [16]

## 2.2.2 NETCONF

The first approach to SDN came to something that was already present in the literature since years, the Network Configuration Protocol (NETCONF). This is a network management protocol developed and standardized by the IETF in two RFCs, RFC 4741 (2006) [17] and RFC 6241 (2011) [18] and thought as an incremental approach with respect to the simplicity of SNMP (Simple Network Management Protocol). NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices, by simply making use of Remote Procedure Call (RPC) and an Extensible Markup Language (XML) based data encoding for the configuration data as well as the protocol messages. These message are exchanged on top of a secure transport protocol.

During the years, an IETF workgroup has defined the YANG language [19] to let NETCONF be more human-friendly; furthermore, NETCONF has been



modified to be compatible with SNMP.

Therefore NETCONF, being the first Software-Defined approach (even before Software-Defined was something) is found in all those areas that were at the top of the research in that years: it is the case for instance of the Optical networking, where NETCONF is still massively used in optical switches. All the most important SDN frameworks still maintain the NETCONF southbound interface, as this allows them to still have a market in the optical networking area.

### 2.2.3 P4

The P4 (Programming Protocol-Independent Packet Processors) [20] is a domain-specific open-source network programming language that allows the programming of packet forwarding planes, by making use of a number of constructs optimized around network data forwarding. It is maintained by the P4 Language Consortium, a non-profit organization.

Fundamental to P4 is the concept of *match-action pipelines*. Conceptually, forwarding network packets or frames can be broken down into a series of table lookups and corresponding header manipulations. In P4 these manipulations are known as actions and generally consist of things such as copying byte fields from one location to another based on the lookup results on learned forwarding state. P4 addresses only the data plane of a packet forwarding device, it does not specify the control plane nor any exact protocol for communicating state between the control and data planes. Instead, P4 uses the concept of tables to represent forwarding plane state. An interface between the control plane and the various P4 tables must be provided to allow the control plane to inject/modify state in the program. This interface is generally referred to as the program API and fosters the use of a SDN approach.

Finally, the P4 language can be also mapped on top of the OpenFlow language, however the most used SDN frameworks already have a dedicated southbound interface for it.

### 2.2.4 OpenFlow

The most extensively adopted solution is the OpenFlow protocol. OpenFlow standardizes the communication between the network switches and the controller of the network: each time the former is not able to deal with an incoming

flow, it queries the latter to take a decision through the use of a secure TCP channel. OpenFlow is by now thought to become the standard-de-facto in SDN. In the creators' own words, OpenFlow is “*a communications protocol that provides an abstraction of the forwarding plane of a switch or router in the network*” [21].

Focusing on its main features, OpenFlow:

- takes network control functions out of switches and routers, while allowing direct access to, and manipulation of, the forwarding plane of those devices;
- specifies basic primitives that can be used by an external software application to actually program the forwarding plane of network devices;
- works on a per-flow basis to identify network traffic;
- forwards flows according to pre-defined match rules statically or dynamically programmed by the SDN control software.

OpenFlow can be used both in a reactive and in a proactive way.

Regarding the reactive way, whenever the OpenFlow-enabled device receives a data packet and does not know how to handle it, the packet is wrapped in the payload of an OpenFlow *PacketIn* message, that is sent to the network controller managing the device itself. The controller can then analyze the packet and reply to the device with an OpenFlow *FlowMod* message, containing, along with the original data packet that generated the *PacketIn* event, a set of matching rules and actions to be performed when a packet with similar header is received. After the installation of the new *flow rule* into the device's *flow table*, whenever another data packet that matches that entry (or another one already present) is received, a sequence of actions will be performed directly, without needing to query the controller again. If the device has to handle a sequence of similar packets, as in a **ping** sequence, the first packet will take a longer time to be forwarded than the following packets in the sequence. An example of this typical behavior is shown in Figure 2.5 where we have the output of a **ping** session through an OpenFlow/SDN domain. This is actually happening as a result of the forwarding to the controller of the packet that the switch is unable to deal with. Depending on the switch implementation, the packet might be kept waiting in a local queue or not.

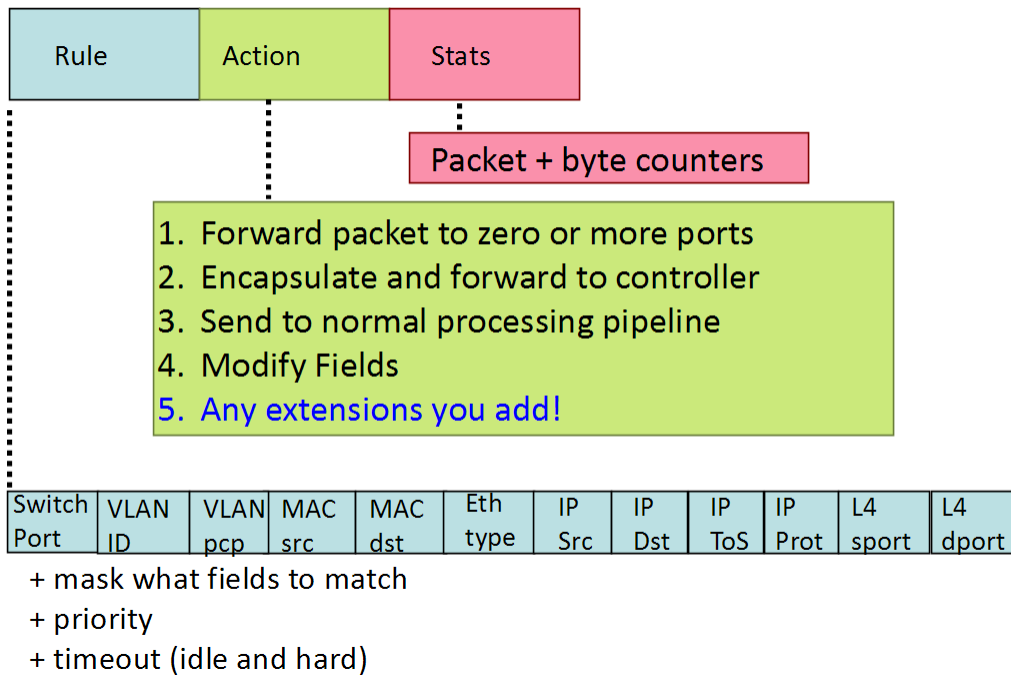


Figure 2.5: The OpenFlow table [22]

On the other hand, whenever it comes to a proactive approach, flow rules are installed before actual traffic reaches the network devices. This approach will obviously enhance data plane latency and reduces the cost of the communication to the controller, but this causes a reduced flexibility. In fact, it is often impossible to install very selective (i.e., precise) and correct flow rules by acting proactively, as in most scenarios many details regarding incoming traffic (e.g., client-side TCP port number) are not known a priori. For this reason, when acting in a proactive way, flows are defined with a coarser granularity (i.e., a lower precision) than they would be with a reactive approach.

More details about the OpenFlow protocol and its versions are available in the documents and standards produced by the ONF [23].

```
mininet> h1 ping -c 3 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=255 time=15.2 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=255 time=4.64 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=255 time=2.62 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 2.620/7.505/15.250/5.539 ms
```

Figure 2.6: ICMP message exchange through an OF switch

## 2.3 Network Functions Virtualization

In the current Internet, IP datagrams are not simply forwarded until they reach their final destination. They are indeed processed by a set of intermediate nodes, each of them implementing an additional function (e.g. address translation, packet inspection and filtering, QoS management, load balancing, etc). A network service can be defined as a sequence of one or more of these *middle-boxes*, traversed in a certain order (e.g. first a NAT, then a Load Balancer, etc.). It is therefore possible to define this sequence as the Service Function Chain (SFC); this SFC can also be reconfigured to compose a different service.

These middle-boxes, however, are typically proprietary highly-specialized systems that run on dedicated hardware, and represent a significant fraction of the network CAPEX (Capital Expenditure) and OPEX (Operational Expenditure). This causes the so-called vendor lock-in: the dependency to the vendor for an update or a new functionality in a middle-box. This is also one of the causes of the so called ossification of the Internet, that makes it difficult to innovate the network by deploying new functionalities, services, etc.

A possible solution to this issue is to virtualize the middle-boxes over a general purpose off-the-shelf server by making use of a hypervisor, rather than buying specialized hardware. This approach provides a higher degree of flexibility and reconfigurability that is desired by operators, as it provides a low-cost solution to the lock-in problem. In addition to it, a correct placement, monitoring, management and migration of these resources is necessary to be able to reconfigure proactively or reactively the service to meet the QoS requirements.

The Network Function Virtualization (NFV) initiative goes in this direction

by defining standard elementary building blocks to implement service chains capable of adapting to the rapidly changing user requirements. Therefore, the NFV is a network architecture paradigm that makes use of virtualization technologies to move toward a new way of designing, deploying and managing network services [24]. It is then possible to define the NFV as the idea to restructure the code running inside a network, moving from monolithic proprietary blocks to flexible and rearrangeable components.

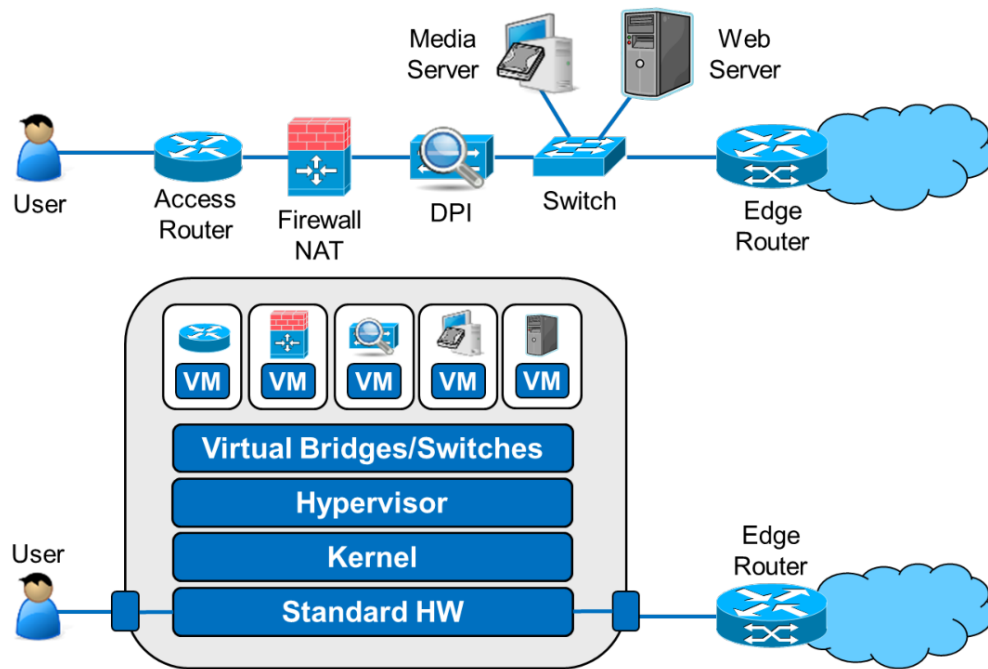


Figure 2.7: Comparison between the traditional approach and the NFV approach, simplified

The NFV introduces some advantages with respect to the traditional scenario:

- independence from (ad-hoc) hardware, allowing separate development and maintenance of software and hardware;
- flexibility of the services offered by the network, by simply rearranging the order of VNFs;

- dynamic scaling of the capabilities of the whole service, according to the actual load, allowing to reactively rearrange the SFC.

After having dealt with the problem of an efficient placement [25], it is important to think about the possible ways to set up the paths among the different virtual resources (e.g. Virtual Machines). As long as there is the need for reconfigurability, the idea for an efficient communication and management is the Software Defined Networking approach rather than a legacy solution. Therefore, whenever the Key Performance Indicators (KPIs) are not compliant with the SLA, there is the need for scaling, routing elsewhere the traffic, etc. That is the reason why when considering NFV it is customary to refer to it inside a cloud environment, where scaling techniques are considered and SDN is employed.

The NFV reference architecture is described by ETSI in [26]. A diagram of the NFV reference architectural framework is shown in Figure 2.8. The shown functional blocks are the following ones:

- Virtualized Network Functions (VNF), which are the virtualized versions of the legacy network functions (middle-boxes) and may include elements of the core network as well as elements of everyday networks (e.g., firewalls);
- the Element Management System (EMS), which performs the management functionality for one or multiple VNFs;
- NFV infrastructure, the set of all hardware and software components on top of which VNFs are deployed, managed and executed, including:
  - hardware resources, assumed to be commercial off-the-shelf physical equipment, providing processing, storage and connectivity resources to VNFs through the Virtualization Layer;
  - Virtualization Layer, such as a hypervisor, which *abstracts* the physical resources, to let the VNFs to transparently make use of the underlying infrastructure;
- Virtualized Infrastructure Manager(s) (VIM), which comprises the functionalities that are used to control and manage the interaction of a VNF with computing, storage and network resources under its authority, as well as their virtualization;
- Orchestrator, in charge of the orchestration and management of NFV infrastructure and software resources;
- VNF Manager(s), responsible for VNF life cycle management (e.g. instantiation, update, query, scaling, termination);

- Service, VNF and Infrastructure Description, which is a data set that provides information regarding the VNF deployment template, VNF Forwarding Graph, service-related information, and NFV infrastructure information models;
- Operation and Business Support Systems (OSS/BSS), which are used by an operator to support a broad range of telecommunication services.

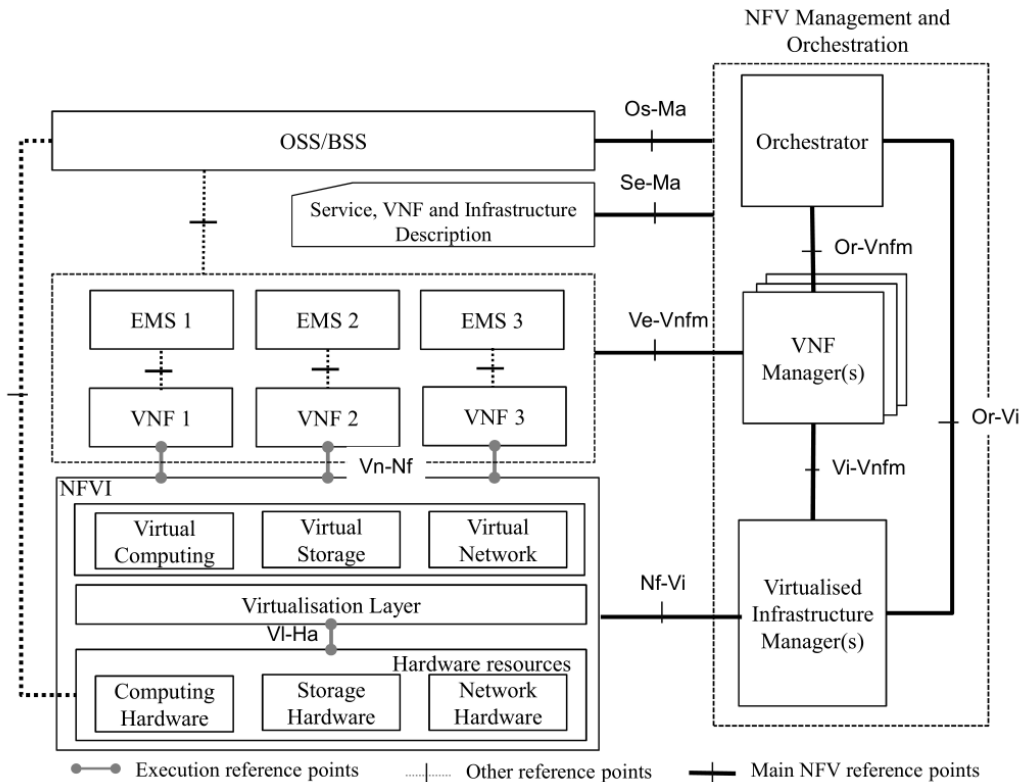


Figure 2.8: NFV reference architectural framework [26]

In Figure 2.8, the main reference points (i.e., the logical interconnections) that are in the scope of NFV are shown by solid lines. In the scope of this document, the focus is mostly on the reference points labeled as *Or-Vi* and *Nf-Vi* in the diagram. The former interconnection is used for carrying resource reservation and/or allocation requests by the Orchestrator, virtualized hardware resource configuration, and state information exchange (e.g. events). The latter is used for the specific assignment of virtualized resources in response

to resource allocation requests, forwarding of information regarding state of the the virtualized resources, and hardware resource configuration and state information exchange (e.g. events).

### 2.3.1 The NFV-MANO framework

Due to the decoupling of the Network Functions software from the NFV Infrastructure (NFVI), a strong coordination between the resources requested by the VNFs is needed. The Network Functions Virtualization Management and Orchestration (NFV-MANO) architectural framework, described by ETSI in [27], has the role of managing the NFVI while orchestrating the allocation of resources needed by the VNFs. A functional-level representation of the MANO framework is shown in Figure 2.9.

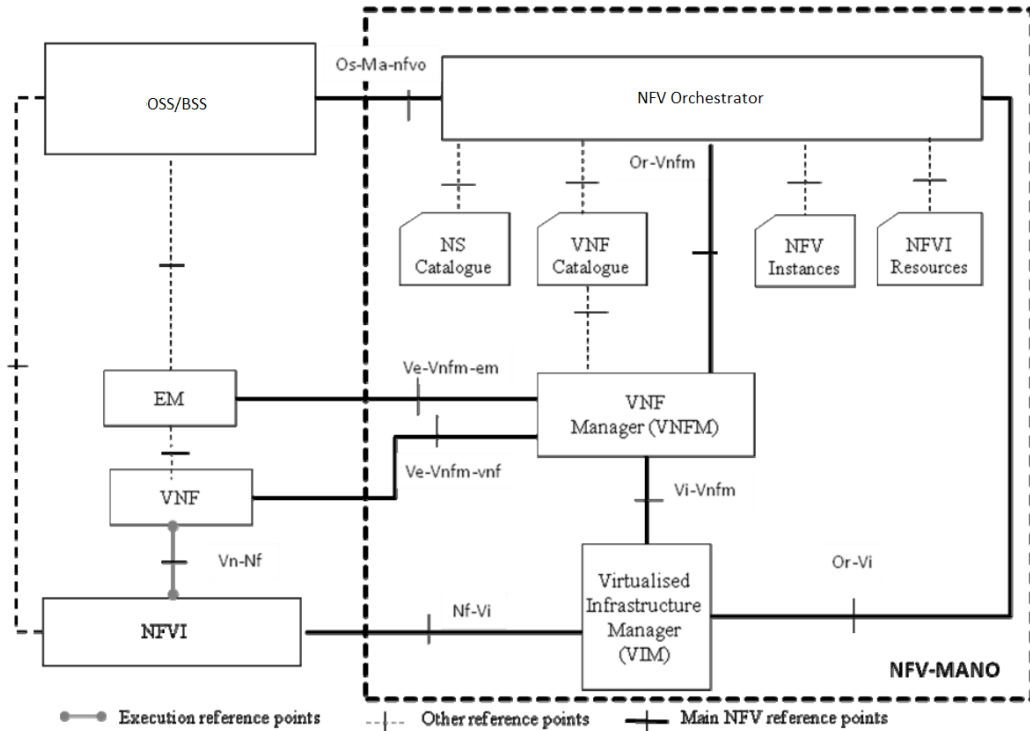


Figure 2.9: The NFV-MANO architectural framework with ref. points [27]

Moreover, it might be taken in a more general view whenever the consideration is related to a SDN/NFV deployment. A hierarchical scheme of the



architecture is shown in Figure 2.10, which highlights the role of SDN Controllers in a multi-domain scenario. For instance, the generic deployment which will be presented later as a reference architecture in Figure 2.16 is actually a slightly differently characterized version of Figure 2.10.

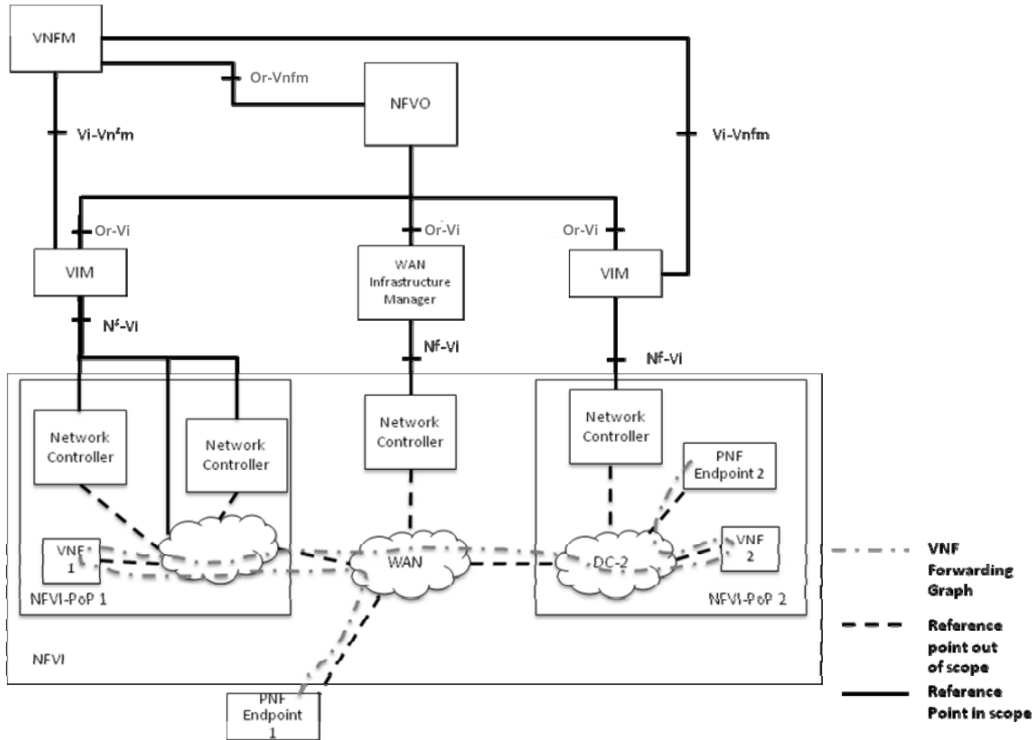


Figure 2.10: Hierarchical view in SDN multi-domain scenario [27]

## 2.4 Cloud computing

### 2.4.1 Before cloud computing

In the computer networks world, it is easy to state at least four phases that paved the road to cloud computing [28].

1. **Grid computing:** it allowed the sharing of heterogeneous resources as computational power, memory, data, etc. to be provided for very limited

and specific application areas.

2. **Virtualization:** as shown before, the need to virtualize resources has increased and therefore virtualization has become massively used, with the natural consequence of the creation of several server farms.
3. **Web 2.0:** it is possible to define it as the web after the introduction of asynchronous protocols not visible to users, asking only the required information instead of the whole web page (Asynchronous Javascript and XML - AJAX). In this way, the use of web services without the need of installing any software locally lead to a new business model (which will result then to be the Software-as-a-Service).
4. **Utility computing:** through it, it has been possible to move to a pay-per-use model for computing, networking, etc. organized as a public utility with a simple interface to be used for the client.

### 2.4.2 The Cloud paradigm

Cloud Computing is a paradigm that aims at enabling ubiquitous, on-demand access to a shared pool of configurable computing and infrastructure resources [29]. It is possible to define the Private Cloud approach as the case in which the servers are privately stored inside the company itself, as well as the Public Cloud one, where the servers are shared among all the users in the service provider data center. It is then possible to follow a hybrid Cloud approach.

As already mentioned in Chapter 1, this paradigm allows network service providers to offer their services in the same way as utility services, such as water, electric power and gas are nowadays distributed: the end users pay for what they get. In order to do so, Cloud Computing takes advantage of both hardware and software resources, which are distributed and virtualized in the network [30], and it is supported in this by the high data rates made available by broad-band connection.

End users expect the resources offered by the Cloud to be instantiated and used in a transparent, seamless way. These resources, however, may be geographically distributed all over the world, imposing high demands on the interconnecting network, in terms of configuration delay, not to mention latency and reliability in the data plane. This is where SDN comes into play, allowing the resources to quickly be configured or re-configured in order to

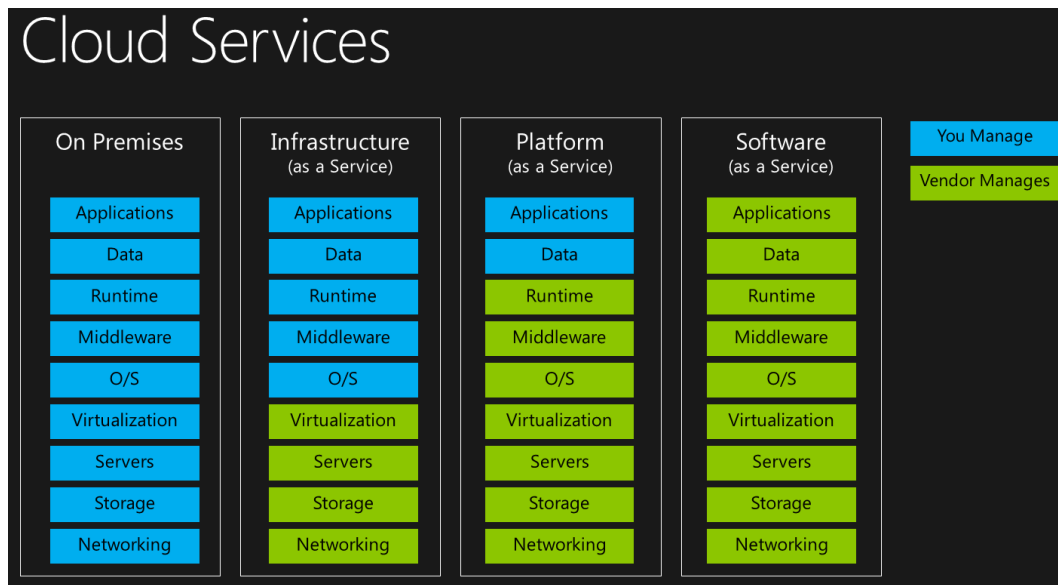


Figure 2.11: \*-as-a-Service models [33]

match the user's requests. Moreover, thanks to the centralized management approach that the SDN paradigm offers, data flows can be dynamically steered to the best path from the user to the server hosting the resources.

Depending on the service that the cloud provider is implementing, it is possible to define, as shown in Figure 2.11:

- Software-as-a-Service (SaaS): the vendor manages everything, delivering to the user the final application; this is the case for instance of Google Docs, storage services like Amazon S3 or DropBox but also CRM software like Salesforce [31];
- Platform-as-a-Service (PaaS): here the user is provided a ready environment to work on; this is the case of IBM Bluemix and Red Hat OpenShift [32];
- Infrastructure-as-a-Service (IaaS): everything that is after the virtualization is left to the user, who decides even the operating system over which the resource should run.

It is necessary to say that cloud computing is currently implemented and delivered by different providers and, as already said, at different levels: it is

significant the case of Amazon Web Services (AWS) [34], providing a platform where the internal low level details are not provided to the final user, which will transparently use its resources. Some other remarkable business solutions are Microsoft Azure [33] and IBM Bluemix [35], addressing the cited paradigm at distinct levels. On the other hand, a broad range of vendors' solution (HP, Huawei, Rackspace, etc.) are based on an Open Source project called *OpenStack* [36]: this is a well-known implementation that provides all the needed components and features for running and managing a cloud platform. One of its goals is to achieve interoperability to avoid the users' lock-in into a single solution: for instance, it is easy to export an OpenStack instance into AWS without the risk of incurring in an incompatibility issue. Unlike the previously mentioned closed solutions, one of the plus points of OpenStack is the fact that, being open source, it can then be studied deeply to understand its benefits and its bottlenecks. Its performance can be accurately evaluated in terms of internals and software and will be presented in Chapter 4. The final extent of this document is the evaluation and improvement of these performance by making an extensive use of SDN controllers and networking improvements to the platform.

However, as stated before, cloud computing is actually the follow-up of already present technologies. It is therefore possible to say that it is not a revolution, but a different, not on premises, user-friendly way to outsource resources, instead of the usual fixed data centers subscription. It enlarges the possibility to reach the scale economies and to build new services, as on-demand video or audio provisioning. For instance, Netflix is a video content provider [37] that mostly has its servers in the Cloud (specifically, it uses Amazon Web Services) and makes use of Content Delivery Networks to push the content as near as possible to the user. Similarly, also the music content provider Spotify [38] just owns few servers for the authentication phase, but all the contents are outsourced through the use of external cloud platforms (first, it used AWS [39], then it started to move over Google Cloud Platform [40]). Indeed, cloud providers grant to these companies a (possibly HA) service at a cost that is lower than what it would be whenever considering inside-company servers, including all the maintenance and workforce.

In Figure 2.12, it is possible to see the NIST view regarding the Cloud; in particular it is shown which are the deployment models cited before, the service models as well as the characteristics the Cloud should have.

As shown in Chapter 1, network operators and service providers are the

entities that mostly rely on the development and deployment of these technologies: indeed, their high-demanding requisites triggered the necessities for software tools able to easily deploy, control and rearrange new services. This is something that cloud computing is able to achieve, as a result of its virtualization capabilities as well as the software defined networking mechanisms employed. However, the performance of these components have to be considered both when in a standalone configuration, but especially when there is integration among them. This integration should be properly evaluated, configured and tuned in order to be efficient in the provisioning of the service.

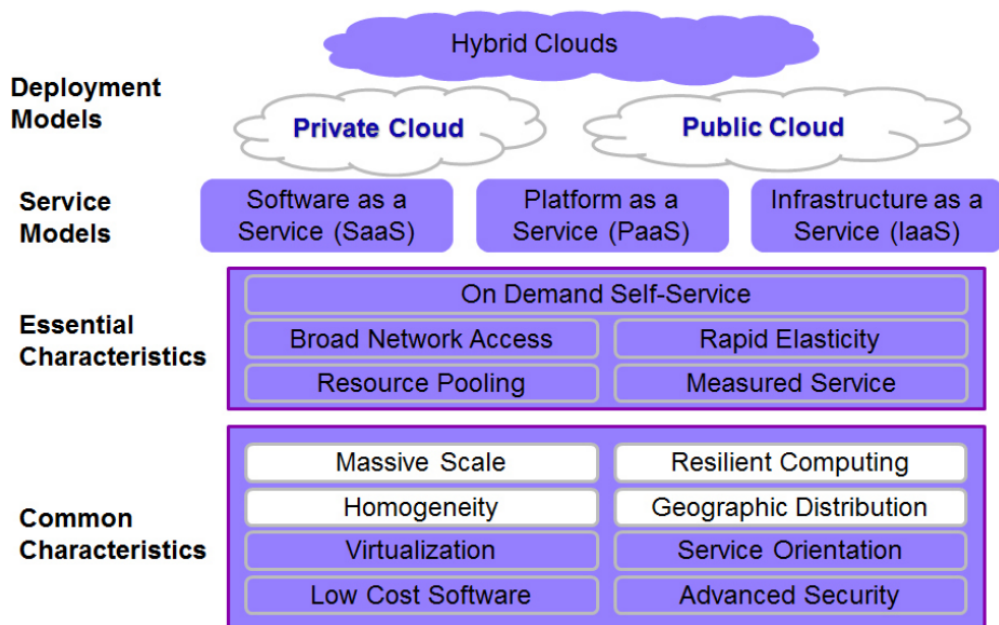


Figure 2.12: National Institute of Standards and Technology Standard Cloud definition

## 2.5 SDN in NFV architectures

### 2.5.1 SDN usage in NFV architectural framework

In December 2015, ETSI proposed the ETSI NFV-EVE [41], a report on SDN usage in NFV architectural framework. This massively complete work deals

on how and where to implement SDN in the Cloud and inherits the NFV architectural framework shown before.

What is proposed is that the SDN applications, SDN controllers and SDN resources have fixed positions in the NFV architectural framework.

Referring to Figure 2.13, for SDN resources the scenario that might be envisaged is to have a:

- (a) physical switch or router;
- (b) virtual switch or router;
- (c) e-switch, software based SDN enabled switch in a server NIC;
- (d) switch or router as a VNF.

Moreover, in case (d), the resource might be logically part of the NFVI or belong to an independent tenant's domain.

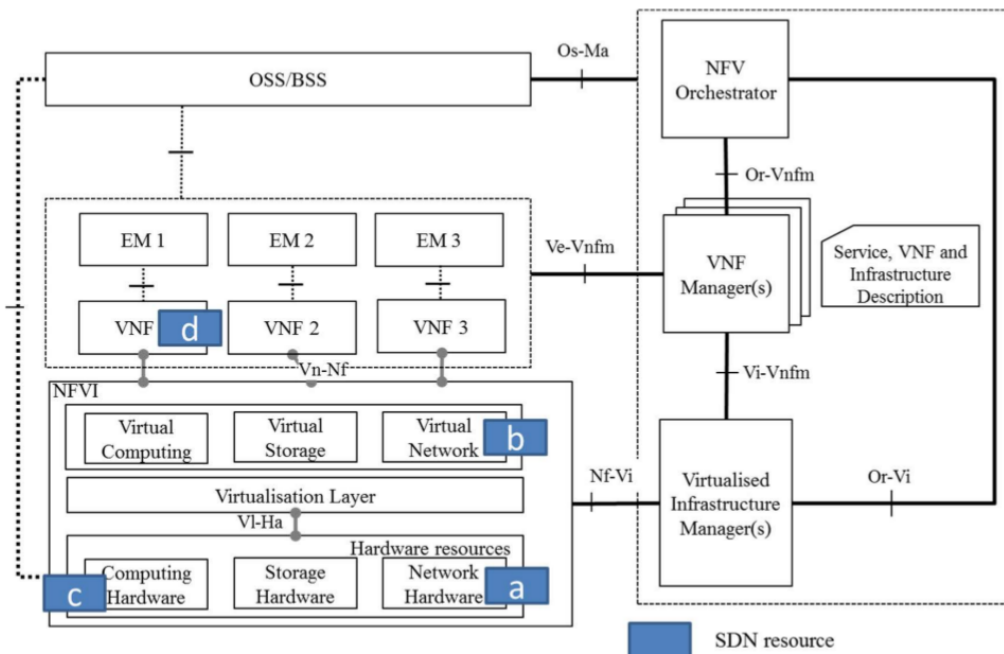


Figure 2.13: Position of SDN resources in an NFV architectural framework

Regarding the SDN controller position (Figure 2.14), the possible scenarios for the SDN controller are when it is:

1. merged with the Virtualised Infrastructure Manager functionality;
2. Virtualised as a VNF;
3. part of the NFVI and is not a VNF;
4. part of the OSS/BSS;
5. a PNF.

In Case 2), the VNF might be logically part of the NFVI and therefore belong to a special infrastructure tenant or belong to an independent tenant.

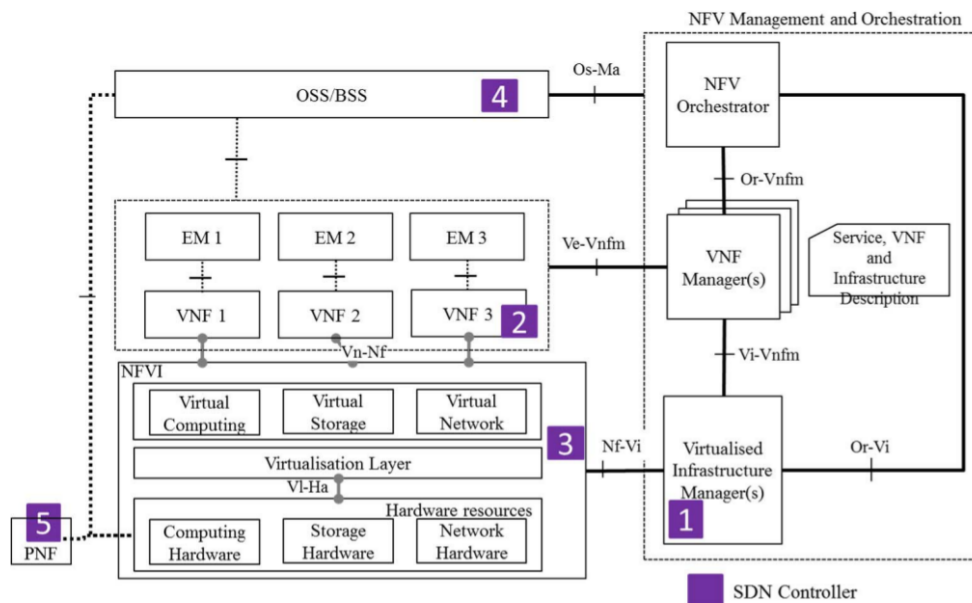


Figure 2.14: Position of SDN controllers in an NFV architectural framework

As the SDN applications may interface with multiple SDN controllers, this case study has to be properly considered in relation on where the latter are (Figure 2.15). Therefore SDN application might be:

- (i) part of a PNF;
  
- (ii) part of the VIM;
  
- (iii) Virtualised as a VNF;
  
- (iv) part of an EM;
  
- (v) part of the OSS/BSS.

In Case i), the network hardware might be a physical appliance talking to an SDN controller, or a complete solution including multiple SDN components, such as SDN controller + SDN application for instance. For Case ii), the VIM might be an application interfacing with an SDN controller in the NFVI - for instance OpenStack Neutron as a VIM interfacing with an SDN controller in the NFVI. The Case iv) considers a SDN application that might be an element manager interfacing with an SDN controller to collect some metrics or configure some parameters. Finally, in Case v), the SDN application might be an application interfacing with an SDN controller for instance in the OSS-BSS for tenant SDN service definitions.



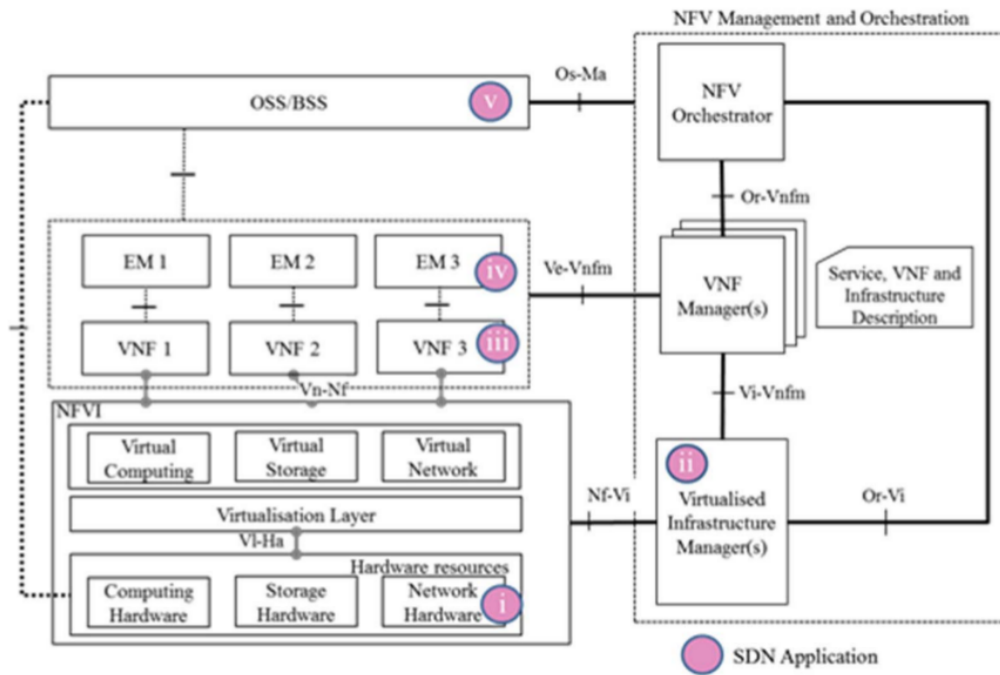


Figure 2.15: Position of SDN applications in an NFV architectural framework

## 2.5.2 SDN for dynamic Service Function Chaining

During the last few years, the University of Bologna research group in New Emerging Technologies for Networking (Net2Lab) has deeply studied the SFC problematics. In particular, the focus has been on the design of the control plane of the SDN controllers capable of performing traffic steering to achieve a fully dynamic service chaining. To show the results related to these studies, it is necessary to present the Reference NFV architecture, shown in Figure 2.16. This reference architecture considers also the possibility that SDN domains are interconnected through non-SDN domains. This assumption stems from the fact that it appears reasonable that, at least by now, a network operator will deploy SDN technologies mainly within data center infrastructures where the VNF resources will be located e.g., in the operators points of presence or central offices rather than in backbone networks. In this case, traffic flows that must traverse a number of SDN domains can be properly routed by adopting some form of tunneling or overlay network technology across the non-SDN domains, such as Generic Routing Encapsulation (GRE), Virtual eXtensible Local Area Network (VXLAN), or Network Service Header (NSH).

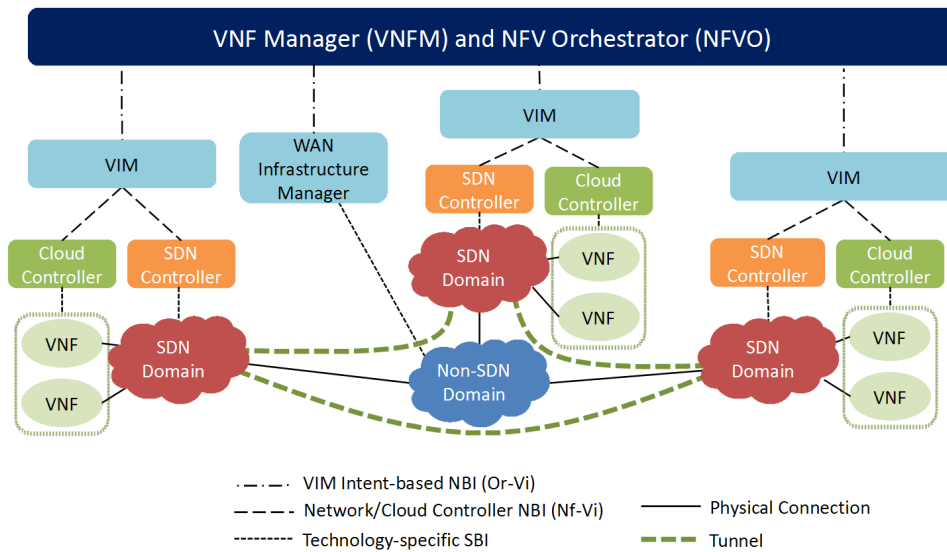


Figure 2.16: Reference multi-domain SDN/NFV architecture, in general

In [42] it is presented a proof of concept (PoC) implementation of a vendor-

independent, technology-agnostic, intent-based NBI for controlling dynamic service function chaining on top of a SDN infrastructure. The intent-based approach focuses on what should be done and not on how it should be done, and aims at decoupling the abstracted service descriptions issued by applications and orchestrators from the technology-specific control and management directives that each VIM/controller must send to its respective devices through the southbound interface (SBI). It is noteworthy to remark that this NBI is compliant with the ONF description of the intent-based NBI.

The decision has been to take advantage of SDN controller-dependent intent interfaces by choosing the Open Network Operating System (ONOS) platform for the PoC. However, the use of ONOS standalone interface does not provide the required abstraction levels for a general and technology-independent NBI. While the ONOS Intent Framework does allow users to directly specify a high-level policy, it also requires some knowledge of low-level details, such as IP addresses and switch port numbers. On the other hand, this NBI implementation used in the PoC allows the user to specify high-level service policies without the need to know any network detail: these details are grabbed and handled by our domain-specific solution, and the high-level policies are resolved and mapped into suitable ONOS intents, which will then be compiled and translated into proper flow rules.

By testing it on an emulated multi-domain SDN infrastructure, the results have shown that it was possible to successfully create, update, delete and flush the forwarding paths according to the intent specified via the NBI. The measurements of the responsiveness of the VIM implementation under increasing load proved the correct functionality of the NBI and the scalability potentials of the proposed approach.

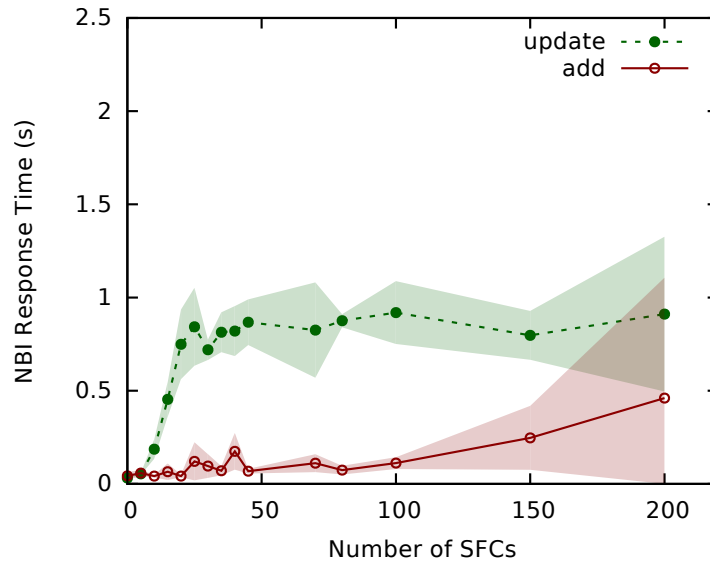


Figure 2.17: Average NBI response time and 95% confidence interval when SFC add and update actions are performed, as a function of the number of SFCs

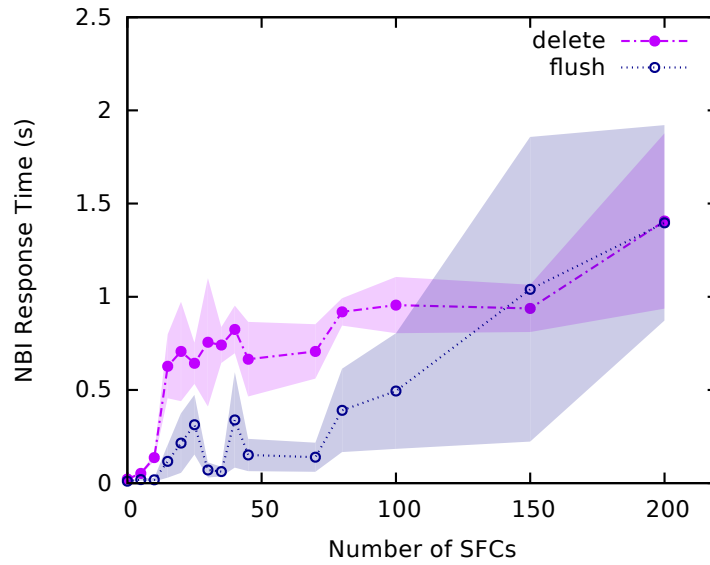


Figure 2.18: Average NBI response time and 95% confidence interval when SFC delete and flush actions are performed, as a function of the number of SFCs

Another study in the same direction is presented in [43]. Here it is presented a reference architecture inspired by the ETSI MANO framework (Figure 2.19) and an intent-based NBI for end-to-end service management and orchestration across multiple technological domains. In particular, the use case that has been considered is the one of an Internet of Things (IoT) infrastructure deployment connected, by means of an OpenFlow-based SDN domain, to the corresponding cloud-based data collection, processing, and publishing services with quality of service (QoS) differentiation. This has been validated over a heterogeneous OpenFlow/IoT SDN testbed, shown in Figure 2.20 demonstrating the feasibility of the approach and the potentials of the NBI.

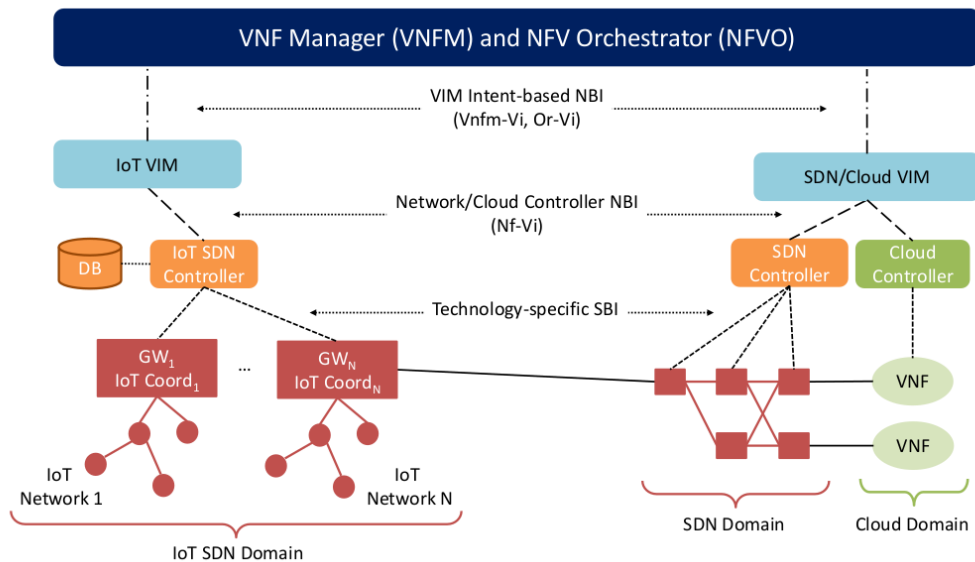


Figure 2.19: Reference multi-domain SDN/NFV architecture, specialized for the use case of IoT data collection and related cloud-based consumption.

Scalability tests on the ONOS-based VIM also gave promising results [44].

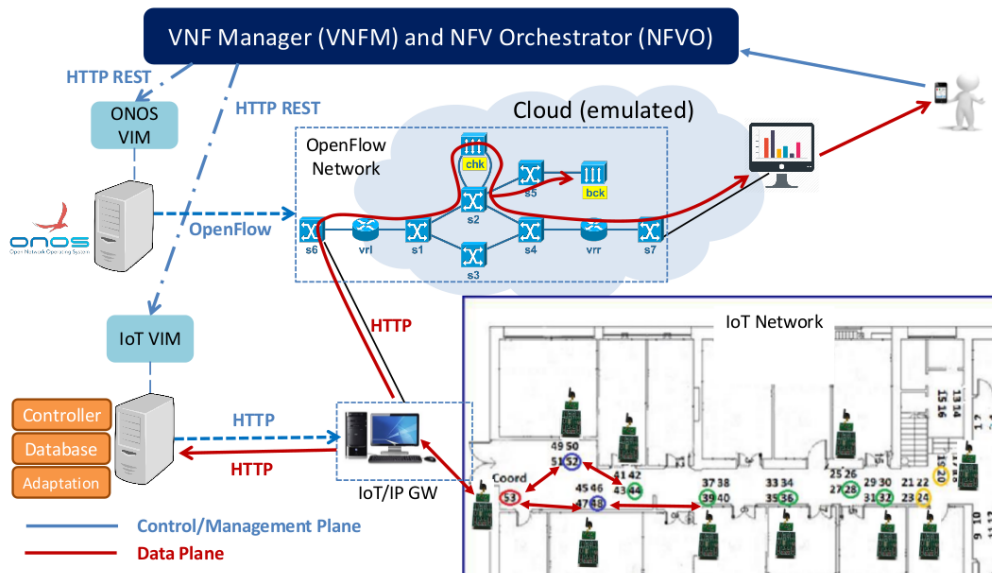


Figure 2.20: The NFV/SDN test bed setup developed to demonstrate multi-domain SDN/NFV management and orchestration

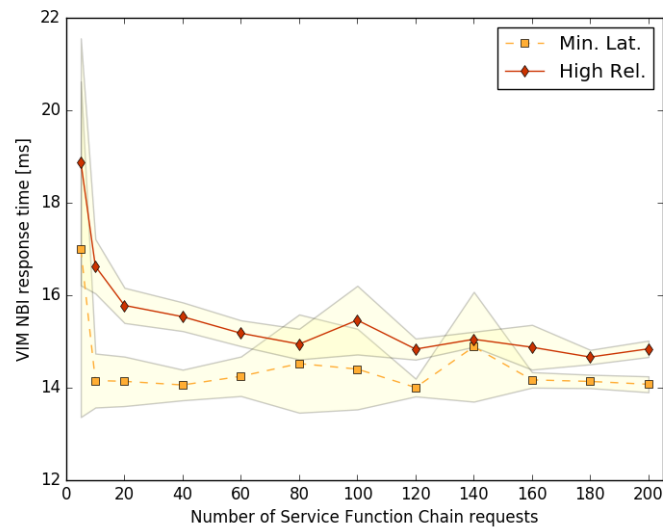


Figure 2.21: Average NBI response time and 95% confidence interval at the SDN/cloud VIM with increasing number of service chain requests.

Finally, the last work presented here describes a proof-of-concept implementation of the Service Function Chaining Control Plane, exploiting the IETF Network Service Header approach. The proposed implementation combines the OpenFlow protocol to control and configure the network nodes and the NSH method to adapt the service requirements to the transport technology. The manuscript shows that the result of this combination is a very general architecture that may be used to implement any sort of Service Function Chain with great flexibility [45].

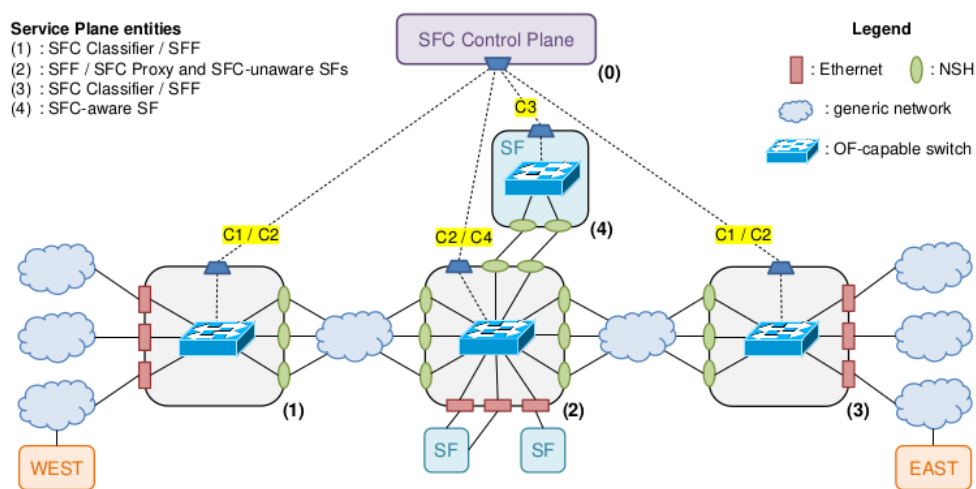


Figure 2.22: Reference NSH experiment scenario

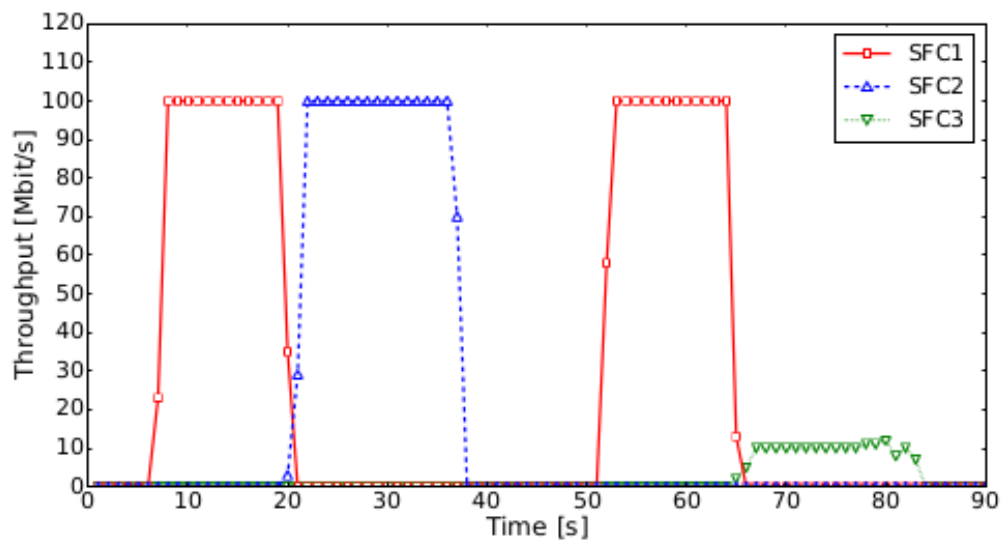


Figure 2.23: WEST-to-EAST throughput measured at the OF-S within Node (2) while applying dynamic SFC



## Chapter 3

# The OpenStack platform

OpenStack is a set of open source software tools (known as projects) for building and managing cloud computing platforms for public and private clouds.

OpenStack has been founded by NASA and Rackspace in 2010 and is now backed by more than 500 companies, among whom there are some of the biggest companies in software development and hosting, as well as vendors, Telcos and even service providers. Moreover, OpenStack is backed by thousands of individual community members and managed by the OpenStack Foundation, a “non-profit that oversees both development and community-building around the project” [46]. OpenStack has a six-month time-based release cycle. Being OpenStack an open source platform, there are copious implementations based on its software; some of them are closed versions (as Huawei FusionSphere, HP Helion, etc.). It is possible to see some of them in Figure 3.1.

As in virtualization resources such as storage, CPU, and RAM are abstracted from a variety of vendor-specific programs and split by a hypervisor before being distributed as needed, OpenStack uses a consistent set of APIs to abstract those virtual resources one step further into discrete pools used to power standard cloud computing tools that administrators and users interact with directly [48].

In order to go into the details of the main functionalities a Cloud should provide, it is possible to refer to Figure 3.2 where these details are presented in relation to the different users. First of all, it is important to state there are API to access the platform: in general, in the Cloud it is customary to isolate components by having a different API for each of it.

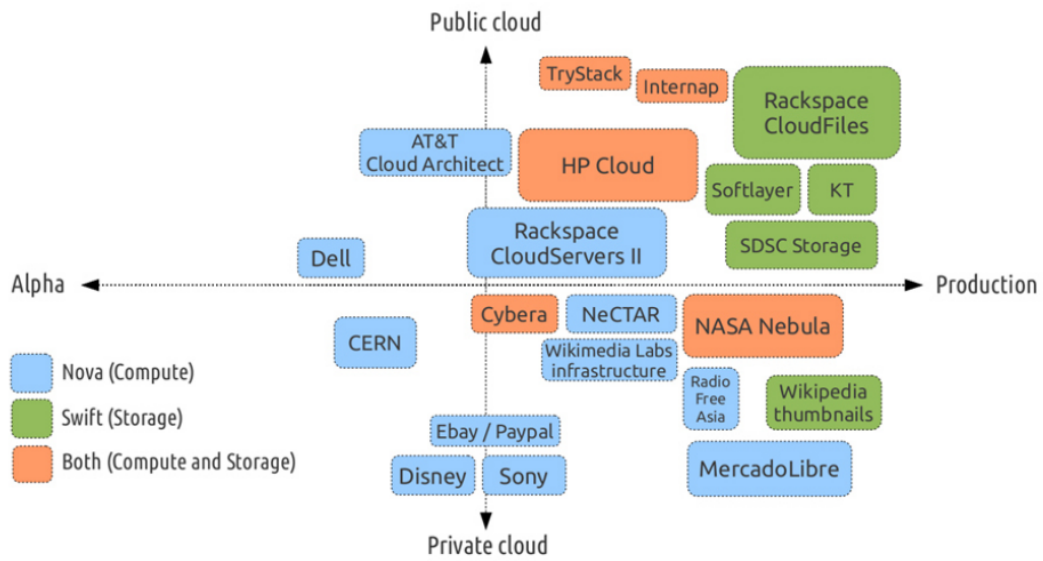


Figure 3.1: Some known deployments of OpenStack [47]

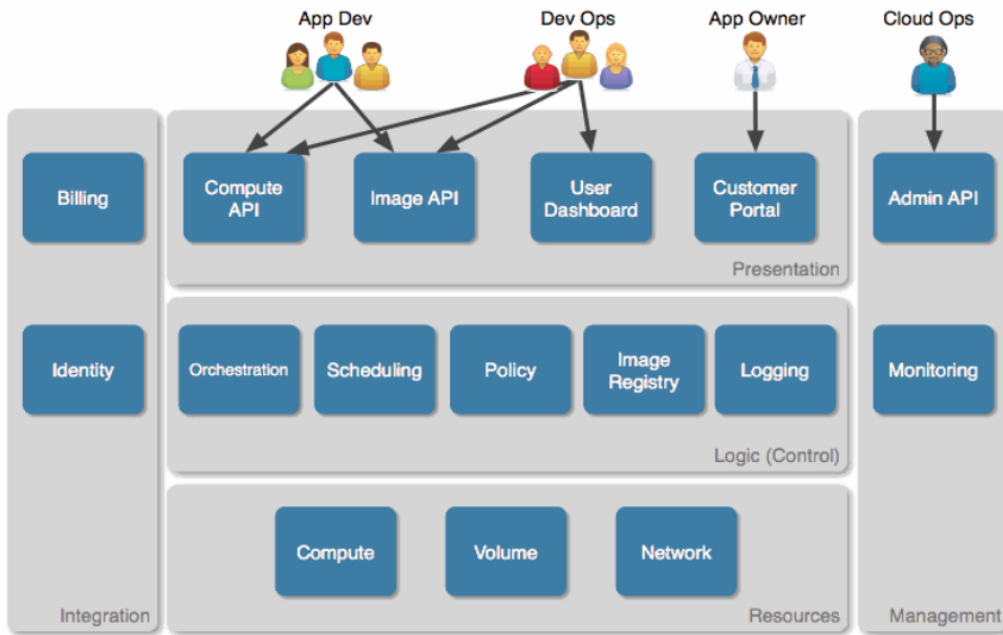


Figure 3.2: Main functions of a Cloud [49]

It is also possible to see that the Application Developers, which job is to develop applications and managing the virtual resources (a sort of System Administrator of resources) mainly interact with the Compute and Image API. On the other hand, the DevOps are also interested in the User Dashboard. Moreover, it is noticeable that there is a Management function where only the Cloud Ops (which are the cloud platform managers) may access through the Admin API.

### 3.1 OpenStack logical overview

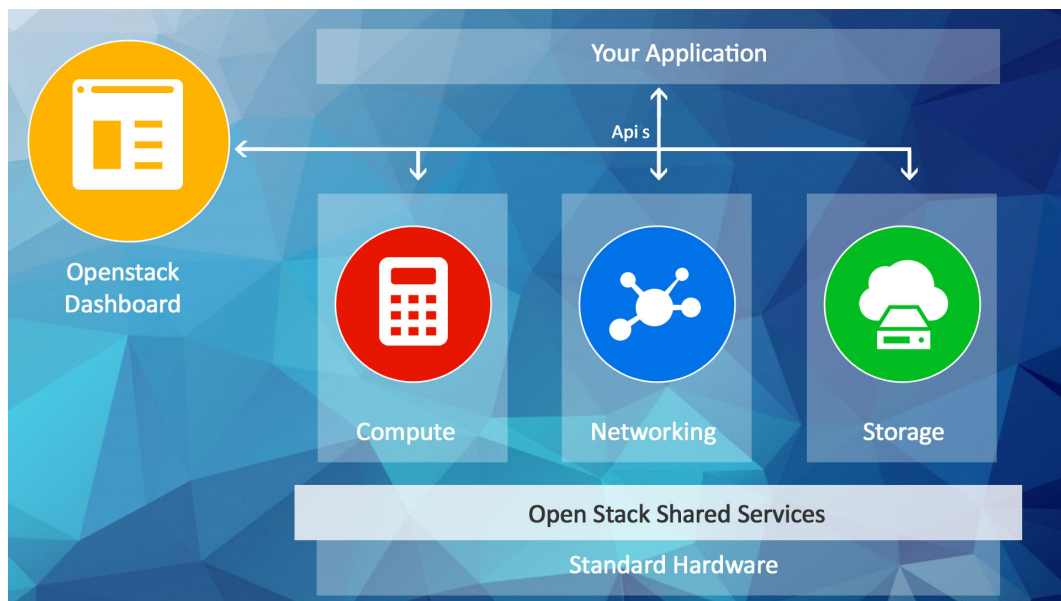


Figure 3.3: High level view of OpenStack [36]

As previously stated, OpenStack is actually a set of interrelated software components: each of these components is responsible for a certain function and exposes some REST API. Therefore, in order to provide a correct and complete service to the user, these components should cooperate together, as shown in Figure 3.4.

All the OpenStack services share the same internal architecture organization, that follows a clear design and implementation guidelines.

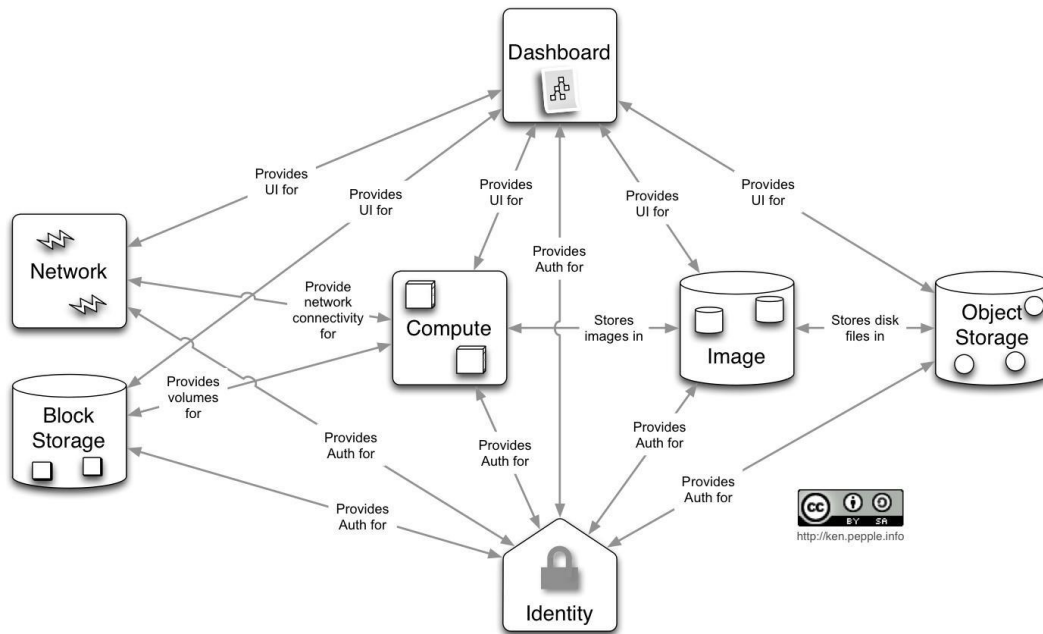


Figure 3.4: OpenStack logic overview [47]

- Scalability and elasticity: this is gained through horizontal scalability  $\Rightarrow$  the whole system is designed to add new physical hosts easily.
- Reliability: this is possible as different services have minimal dependencies among each other; moreover, core components might be replicated for robustness.
- Internal component storage: each service stores all needed information internally, which is something unusual; there is indeed a database table for each service.
- Loosely coupled asynchronous interactions: some internal interactions among components follow a complete decoupled PUB/SUB approach. This is done to decouple the requests from the responses and make it possible to use Remote Procedure Call-like operations.

Therefore, every service consists of the following core components:

- PUB/SUB messaging service, as Advanced Message Queueing Protocol (AMQP);

- one or more internal core components, realizing the service application logic;
- an API component, that acts as the service front-end to export service functionalities via interoperable RESTful APIs;
- a local database component, where to store the internal service state; it is important to mention that the service will then have its own state, stored in its own DataBase, e.g. MySQL, MongoDB, etc.

It is worth to mention that APIs are also used internally to communicate among components: this of course generates an overhead, but makes the communications more decoupled.

### 3.1.1 OpenStack deployments

In general, an OpenStack deployment may be of two types: a production environment or a development environment.

The development environment is very easy to setup, adaptive to the user's need and it is thought for an evaluation of the platform or for the testing of the functionalities. Some well known solutions are DevStack and PackStack. However, this deployment is very unstable and it is not thought for running complex experiments or to be given to a customer.

Instead, the production environment is more complex to setup, but also finer-grained. Indeed, its installation requires a, sometimes long, planning phase where it is decided on which nodes the components have to be deployed and with which configuration. Then, the actual deployment might take place in terms of writing configurations by-hand after installing the components, as the official OpenStack guide recommends for relative small environments, or by the use of automation tools as Red Hat TripleO, Ansible, Chef, Puppet, Fuel, Autopilot, etc. In both cases, the learning curve is very steep, therefore it is up to the Cloud administrator to decide which tools to use, considering also the size of the cluster to be built.

### 3.1.2 OpenStack nodes

In general, in OpenStack it is customary to say that there are different type of nodes. The actual intelligence of OpenStack is the Controller node, which is the

node that contains all the components' endpoints; therefore, whenever there is a request from the outside, this request is processed inside the controller node. This node might also be replicated for the sake of reliability, requiring then to manage the High Availability through the use of an additional node that acts as a Front End for the cluster (for instance, a HAproxy or a load balancer).

Of course, as one of the strength of the OpenStack platform is the computing, there is the necessity to have one or more Compute nodes. These are the nodes in which the hypervisor is running and are therefore configured to ease its communication with the OpenStack components.

Another important node is the one that gives the possibility to the instances to access the external networks: this node is called Network node.

In addition to them, many other nodes can be deployed, as a Block Storage node, where to store images, which is useful whenever the deployment is very wide.

However, it is noteworthy to say that depending on the network configuration of the cluster, and depending on the physical nodes in which the different components are installed, as well as how they are configured, the solution proposed by the cluster may change. In practice, it is possible to have a Controller node that is also a Network node or a Controller node that is also a Compute node, just by properly configuring the OpenStack components. A particular case is the "All-in-One" deployment, mainly proposed as the default solution by Development environments: this environment contains all the services in a unique node, which a good option only when it comes to testing.

Finally, a small remark on the networks interconnecting the nodes. In general, it is customary to have a management network, to which each node is connected: this is used by the admin to access the different nodes and for inter-service communications. Instead, the data network is that one regarding the inter-virtual instances communications; depending on the chosen tenant network virtualization mechanisms, the packets exchanged over this network might be VLAN tagged or encapsulated in a VXLAN or GRE packet. Another network is the external one, that allows the virtual instances to access the network and viceversa.

## 3.2 OpenStack components

Before entering in the details, it is important to state that some projects (e.g. Nova, Neutron) are also composed by *plug-ins* and *agents*.

On one hand, an agent is a process that runs on different hosts and communicates through RPC calls with other processes. On the other hand, a plug-in is just composed by static software files (thus, it is not a process) and implements all the APIs that are required to communicate to the corresponding agent; moreover, they usually run in the context of a process. For example, a Layer 2 plug-in runs in the context of a component (e.g. neutron-server) to communicate with the Layer 2 agent.

The concept of plug-ins and agents introduces flexibility in the system: whenever a developer wants to add some extra functionality, this makes it possible without changing the existing system source code. Indeed, it is only needed to add the plug-in to the existing system by which it is possible to talk to the agent.

### 3.2.1 Identity service: Keystone

Keystone is a framework for the authentication and authorization for all the other OpenStack services. It is in charge of the creation and management of users and groups (also called tenants) and it defines the permissions for cloud resources using a role-based access control features approach. In Figure 3.5, it is possible to see what happens when a request from a client (who might be a user or another component) is received.

The OpenStack Identity service provides a single point of integration for managing authentication, authorization, and a catalog of services. In general, the Identity service is the first service a user interacts with. After the authentication phase, it is possible for the user to access to other OpenStack services. Moreover, other OpenStack services make use of the Identity service to ensure users' identity and to discover where the other services are within the deployment through the use of the service catalog. The service catalog is a collection of available services in an OpenStack deployment.

Typically, each service might have one or many endpoints; each endpoint can be one of three types: admin, internal, or public. In addition to it, OpenStack supports multiple regions for scalability. Together, regions, services, and endpoints created within the Identity service comprise the service catalog for

a deployment. It is important to state that in a production environment, different endpoint types might reside on separate networks exposed to different types of users for security reasons. For instance, the public API network might be visible from the Internet so customers can manage their clouds. The admin API network might be restricted to operators within the organization that manages cloud infrastructure. The internal API network might be restricted to the hosts that contain OpenStack services. Each OpenStack service in the deployment needs a service entry with corresponding endpoints stored in the Identity service [50].

It is therefore possible to say that the authentication service uses a combination of domains, projects, users, and roles. A domain represents a collection of projects, groups and users that defines administrative boundaries for managing OpenStack Identity entities, whereas a project can be defined as the base unit of ownership in OpenStack; all the resources in OpenStack should be owned by a specific project. Moreover, in OpenStack Identity the users represent the individual API consumers, owned by a specific domain and associated with roles and/or projects. Finally, the role is the personality that a user assumes to perform a specific set of operations; the role includes a set of rights and privileges [51].

The Identity service contains these components:

- **Server**, to provide authentication and authorization services using a RESTful interface;
- **Drivers**, used for accessing identity information in repositories external to OpenStack (e.g. SQL databases);
- **Modules**, middleware modules to intercept service requests, extract user credentials, and send them to the centralized server for authorization.

Keystone provides four primary services implemented as backends, that is possible to see in Figure 3.6:

- **Identity**, for user authentication;
- **Token**, to replace the password authentication after the log-in;
- **Catalog**, to maintain the endpoint registry used to discover OpenStack services endpoints;



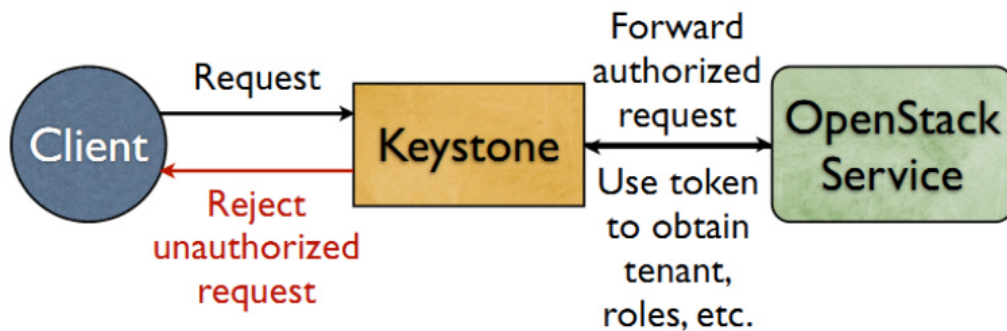


Figure 3.5: Keystone work flow [47]

- **Policy**, to provide a rule-based authorization engine.

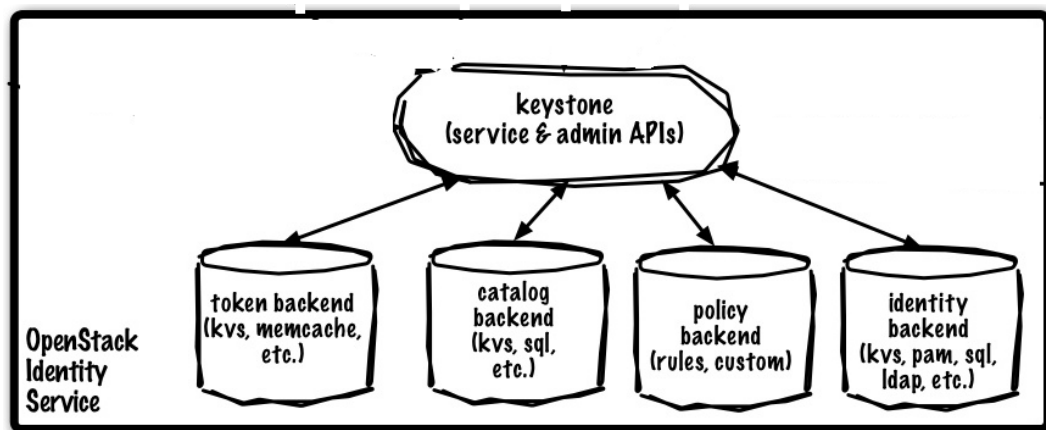


Figure 3.6: Keystone logical architecture [52]

### 3.2.2 Computing service: Nova

Nova is the service dedicated to the computing. It interacts with several hypervisors (i.e. KVM, Docker, LXC, etc.) to provide on-demand virtual servers. In the old releases of OpenStack, before Neutron and Cinder projects were designed, Nova was also in charge of networking and of persistent volumes.

Nova is designed to be modularly structured, in order to horizontally scale whenever a new node is inserted in the OpenStack cluster.

As it possible to see from Figure 3.7, Nova has multiple components. However, only some of them will be discussed in this document.

- **nova-API**: RESTful API web service.
- **nova-compute**: interacts with the hypervisor to manage the virtual instances, acting as a sort of stub; for each compute node, there is a nova-compute component related to the hypervisor of that specific node.
- **nova-scheduler**: coordinates all services and determines the placement of new requested resources; the scheduler itself decides for the node that has more available resources at the moment of the request.
- **nova database**: stores the runtime states of the infrastructure (instance types, active instances, available networks projects) in a coarse-grained way.
- **queue**: it is a middleware for inter-service communications, acting as a central hub for message passing between daemons (by default, it is RabbitMQ, but it can also be ZeroMQ or others).
- **nova-console, nova-novncproxy, nova consoleauth**: they provide user access to the consoles of virtual instances via a proxy.

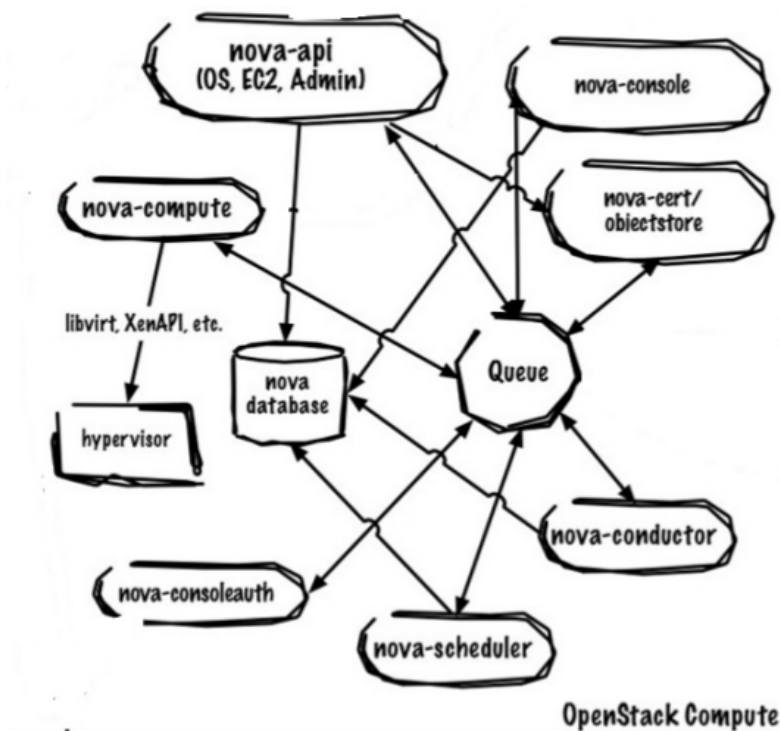


Figure 3.7: Nova logical architecture [52]

In Figure 3.8 it is possible to see what happens inside Nova when nova-API receives a request for the instantiation of a new virtual instance. Whenever a reliable installation of OpenStack (by replicating the main components) is requested, the queue, the database and the nova-scheduler are going to be replicated.

### 3.2.3 Object Storage service: Swift

Swift allows to store and recover files. It provides a completely distributed storage platform that can be accessed by APIs and integrated inside applications or used to store and backup data. It is not a traditional filesystem, but rather a distributed storage system for static data such as virtual machine images, photo storage, email storage, backups and archives. It does not have a central point of control, thus providing properties like scalability, redundancy and durability.

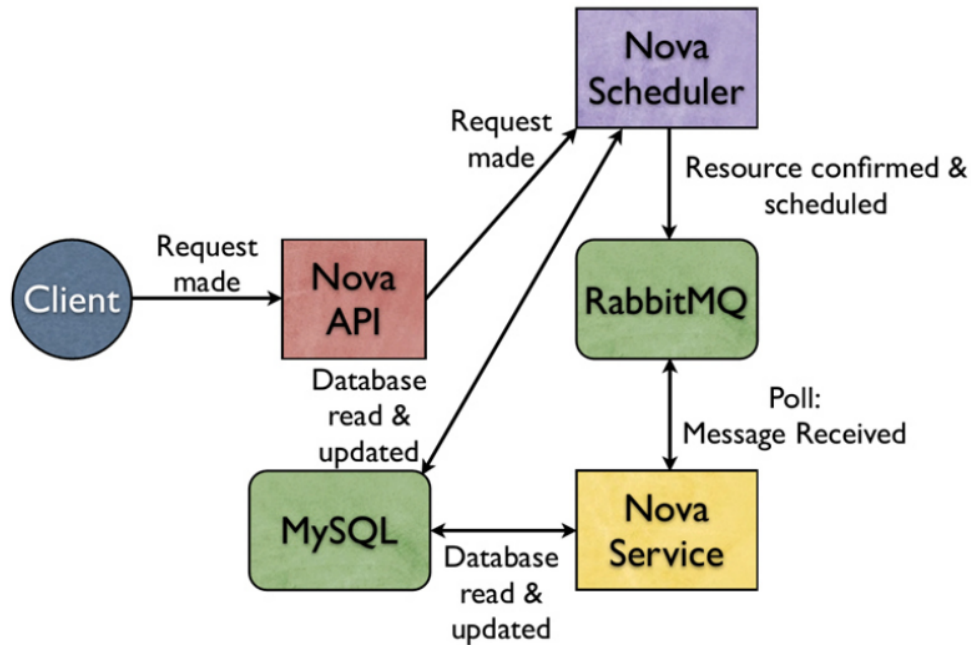


Figure 3.8: Nova work flow when a request for a new virtual resource arrives [47]

- **Proxy server:** it handles incoming requests such as files to upload, modifications to metadata or container creation.
- **Accounts server:** it manages accounts defined through the object storage service.
- **Container server:** it maps containers inside the object storage service.
- **Object server:** it manages files that are stored on various storage nodes.
- **Memcached:** it is an in-memory key-value store for small chunks of arbitrary data (e.g. strings, objects) from results of database calls, API calls, or page rendering [53].

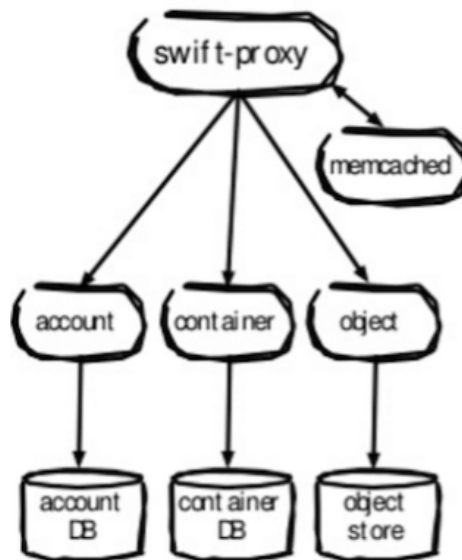


Figure 3.9: Swift logical architecture [52]

### 3.2.4 Image service: Glance

Glance handles the discovery, registration, and delivery of disk and virtual server images. It allows to store images on different storage systems, i.e., Swift, and supports several disk formats (i.e. Raw, qcow2, VMDK, etc.).

- **glance-API:** RESTful API web service to handle requests to discover, store and deliver images.
- **glance-registry:** stores, processes and retrieves image metadata, as image dimension, format, etc.
- **glance database:** database containing image metadata.
- **external repository:** stores the images. Glance supports repositories as legacy file systems, Swift, Amazon S3, and HTTP.
- **Metadata definition service:** a common API for vendors, admins, services and users to define their custom metadata.

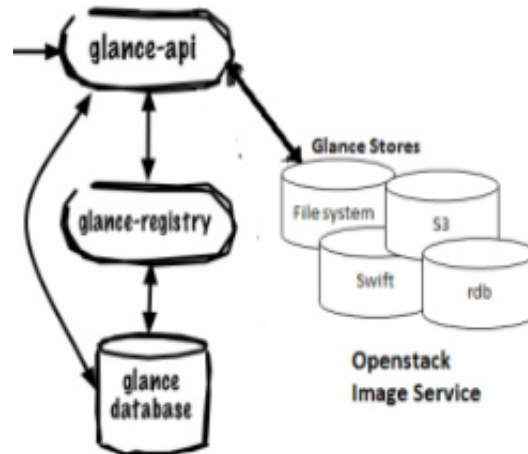


Figure 3.10: Glance logical architecture [52]

### 3.2.5 Block Storage service: Cinder

Cinder handles storage devices that can be attached to VM instances. It is indeed the handler for the creation, attachment and detachment of volumes to/from instances. It supports iSCSI, NFS, FC, RBD, GlusterFS protocols as well as several storage platforms like Ceph, NetApp, Nexenta, SolidFire, and Zadara.

It allows to create snapshots to backup data stored in volumes. Snapshots can then be restored or used to create a new volume.

- **cinder-API:** RESTful API web service that accepts user requests and redirects them to *cinder-volume* to be processed.
- **cinder-volume:** it handles the requests by reading and writing from and to the cinder database, to maintain the system in a consistent state. The interaction among cinder-volume and other components is performed through a message queue.
- **cinder-scheduler:** it selects the best storage device where to create the volume.
- **glance database:** it maintains the volumes' state.

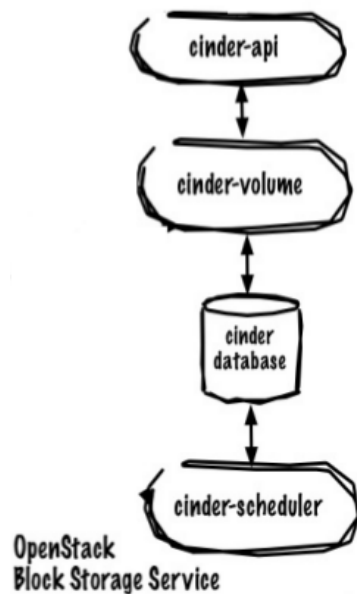


Figure 3.11: Cinder logical architecture [52]

### 3.2.6 Networking service: Neutron

In OpenStack the networking is managed by Neutron. For the purpose of this document, Neutron is extremely significant as all the measurements, the different deployments and the open research activities that will come out are related to its configuration. Therefore, it is strongly vital to know its details, as its complexity in the OpenStack environment is equal only to its influence to the measurements.

Neutron is a pluggable, scalable and API-driven support to manage networks and IP addresses. It provides Network as a Service (NaaS), as the OpenStack users are able to create their own networks where to plug the Virtual Network Interface (*vif*) of their virtual instances.

Neutron provides multi-tenancy: it is indeed possible for an user to run its private resources in some physical node that is shared with other users in a completely isolated and abstracted way. This is the result of the use of particular technologies (which will be shown in Chapter 4). The full control over virtual networks is provided to the user, that is limited on its own projects and cannot see the resources of the others.

Neutron is technology-agnostic: its APIs specify the service, while the different vendors provide their own implementation. It is therefore possible to have extensions related to vendor-specific features, without the need to modify the core source code.

Neutron is loosely coupled: it is indeed a standalone service, not exclusive to the only OpenStack project.

The Neutron components are:

- **neutron-server**: it accepts request sent through APIs and forwards them to the specific plug-in.
- **message queue**: it accepts and routes RPC requests between the neutron-server and the various neutron agents that run on each hypervisor; moreover, it acts as a DB to store the networking state for particular plug-ins.
- **neutron database**: it maintains the network state for some plug-ins.
- **plug-ins & agents**: they execute real actions, such as the connections and disconnections of the ports, the creations of networks, subnets and routers, etc. Some of these are:
  - **dhcp agent**: it provides DHCP functionalities to virtual networks.
  - **L3 agent**: it provides L3/NAT forwarding to provide an external network access for the VMs.
  - **“plug-in” agent**: it runs on each hypervisor to perform local virtual switch configuration. The agent depends on the plug-in that has been used (e.g. Linux Bridge, OpenVSwitch, Cisco, etc.).

Finally, it is important to remark the presence of the Modular Layer 2 (ML2) plug-in: this is a framework allowing OpenStack Networking to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world data centers. It currently works with the existing open-switch, linuxbridge, and hyperv L2 agents, and is intended to replace and deprecate the monolithic plug-ins associated with those L2 agents [54].



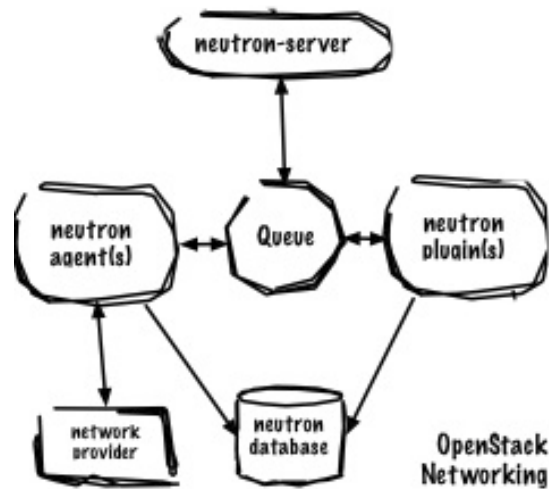


Figure 3.12: Neutron logical architecture [52]

### 3.2.7 Other OpenStack services

Other remarkable OpenStack projects are Horizon, which is a web dashboard to access the user-view of the platform, Heat, the orchestration component which will be shown in Chapter 6, Ceilometer and Monasca for telemetry service, Sahara as a way to provision data-intensive application cluster (Hadoop, Spark, etc.) as well as many others.

A complete list for the Ocata version of OpenStack can be found in [55].

Therefore, whenever an instance is requested, the OpenStack components start working together in order to allocate the needed resources to make it possible. A representation of this is shown in Figure 3.13, related to an older version of OpenStack. Even though some components have changed during the years, the handouts of this Figure is that in OpenStack there is a huge complexity behind even a simple action. All this complexity with its overhead is the other side of the coin due to the OpenStack structure developed to provide its services with a higher degree of flexibility.

Figure 3.14 shows a high-level interaction among OpenStack components. Figure 3.15 shows how OpenStack through its components is fulfilling the cloud functions shown in Figure 3.2.

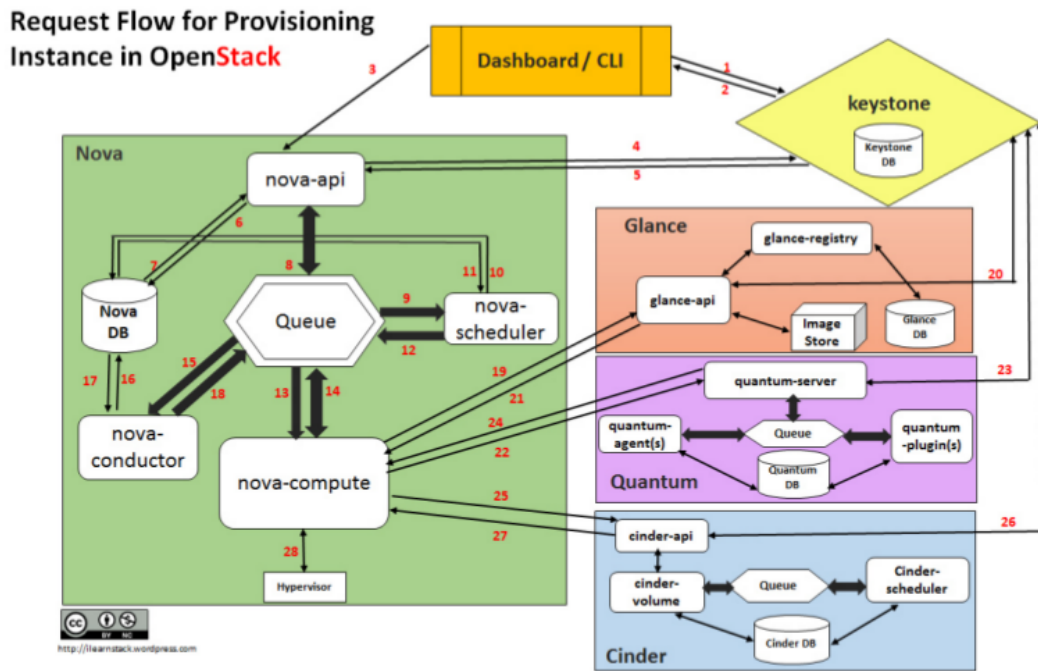


Figure 3.13: OpenStack instance provisioning workflow [56]

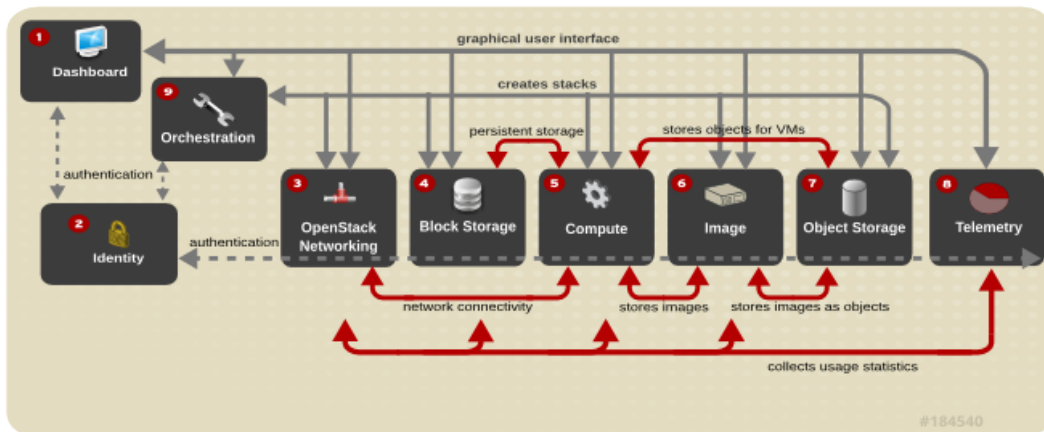


Figure 3.14: OpenStack basic services [57]

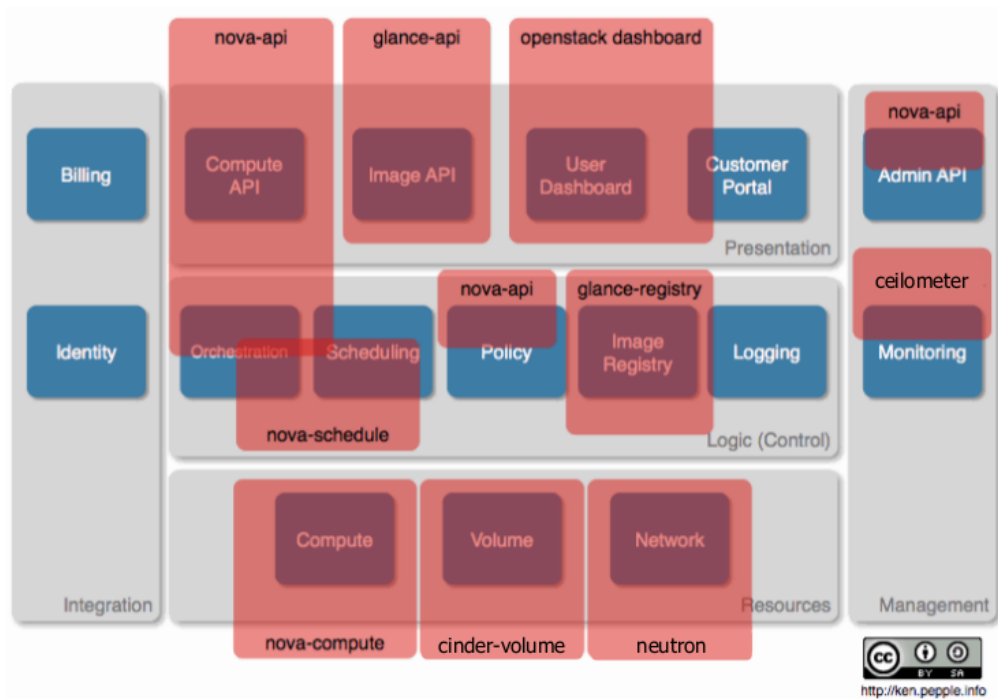


Figure 3.15: OpenStack fulfilled cloud functions [58]

Finally 3.16 shows a complete scheme of all the interactions among OpenStack components.

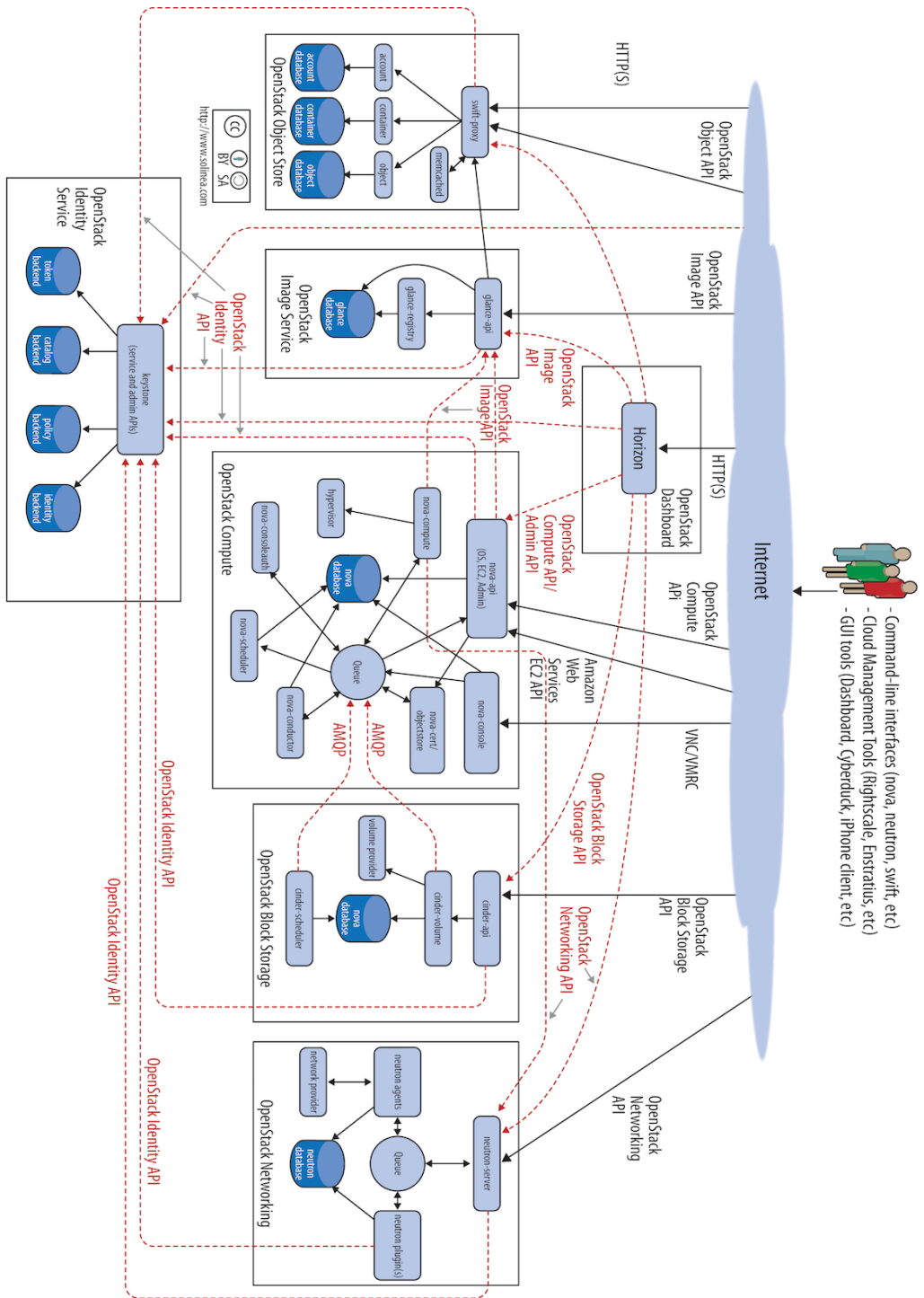


Figure 3.16: Complete OpenStack components overview [52]

# Chapter 4

## Neutron networking: details and improvements

As previously stated, the networking in OpenStack is in charge of Neutron. In its entirety, Neutron is an extremely complex project and it is the one granting to have multiple tenants that can share physical resources in a way that is transparent to the users, still maintaining the isolation. In order to do so, Neutron implies deeply complicated techniques and software.

### 4.1 Neutron abstractions

Neutron decouples the logical view of the network from the actual physical view, providing APIs to define, manage and connect virtual networks. In Figure 4.1 it is possible to observe that an user has some virtual machines connected to different networks as part of its tenant. Although these machines might be physically on different nodes, maybe with different hypervisors, abstraction mechanisms are provided by Neutron such that there is complete transparency for instances. Indeed, what the instance will see about the network is just an abstraction of a legacy network, as a result of the network abstractions that Neutron applies.

Indeed, Neutron defines some network abstractions:

- **Network:** it represents an isolated L2 virtual network segment;
- **Subnet:** it represents an IP(v4/v6) address block on a certain network, which can be assigned to VMs, routers or given networks;

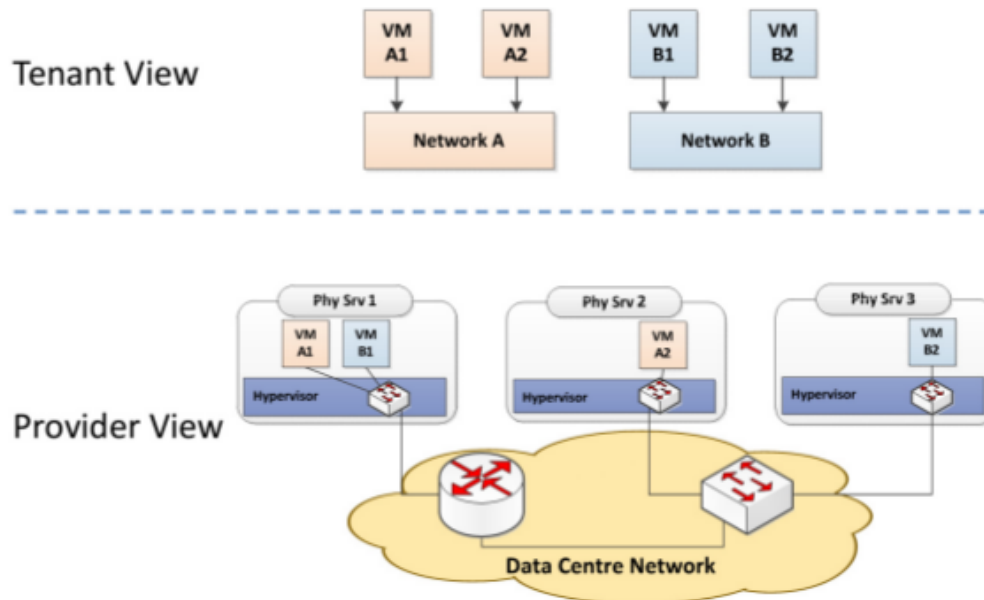


Figure 4.1: Neutron logical view vs. physical view

- **Router:** it represents a gateway between subnets.
- **Fixed IP:** it represents an IP on a tenant network.
- **Floating IP:** it represents a mapping between IP of external networks and a private fixed IP (requiring therefore a DNAT operation);
- **Port:** it represents an attachment point to a network; in practice, it represents a logical switch port on a given network that is then attached to the interface of a VM. The logical port also defines the MAC address and the IP addresses that will be assigned to the plugged interface; as long as the IP address is associated to a port, the latter will be associated to the subnet corresponding to the IP.

## 4.2 Networks and multi-tenancy

Apart from the physical networks connecting the different nodes, in OpenStack there are two types of networks: the tenant network and the provider network.

A *provider network* is a network that is external to the cluster and allows to have outside connectivity, by passing through the network node. A virtual instance can also allocate (and then deallocate, when it is no longer needed) a floating IP on this network to gain external visibility. Provider networks are created only by the OpenStack administrator.

Instead, the *tenant network* (also known as self-service network) is created by the cloud user for connectivity within projects. However, it lacks of connectivity to external networks such as the Internet, unless it is using a virtual router that has a gateway on a provider network. By default, tenant networks are fully isolated and are not shared with other projects. In the creation phase of a virtual instance, a fixed IP is taken from this network (which will not be modifiable after the creation, as a consequence of a Neutron abstraction regarding the port). Moreover, it is important to remark that the instance receives the IP from a DHCP server (which actually is a dnsmasq process generated by the DHCP agent) that is configured to provide to that instance always the same fixed IP.

Neutron supports different types of tenant networks.

- **Flat tenant network:** here there is no tenant support; every instance resides on the same network, which can also be shared with the hosts.
- **Local tenant network:** here the instances reside on the local compute host and are effectively isolated from any external networks.
- **VLAN tenant network:** here 802.1Q tags (VLAN IDs) are used, which are corresponding to the VLANs used in the physical network. This allows instances to communicate with each other across the environment.
- **VXLAN or GRE tunneling tenant network:** network overlays are employed to support private communication between instances.

Inside a node, tenant flows are always separated by *internally assigned* VLAN IDs. Then, to perform communications among physical nodes, tenant flows are separated, for instance, by *user defined* VLAN IDs, VXLAN IDs or

GRE IDs, depending on the chosen tenant isolation technique. It is important to remark that whichever it is the tenant isolation mechanism used, inside a node OpenStack will always make use of automatically internally assigned VLAN IDs. Indeed, even though a tunneling technique is used for the tenant networks, the VLAN IDs are still used to isolate the tenants' traffic inside the node.

However, although by means of this techniques (VLAN, VXLAN, GRE, etc.) the multi-tenancy is achieved, additional mechanisms are needed to let different users have overlapping networks. To do that, the virtual routers for the tenant networks, as well as the processes acting as a DHCP server, are implemented inside several *network namespaces*.

The network namespace is a technology of the Linux kernel that allows to isolate multiple network domains inside a single host, by replicating the network software stack [59]. Therefore, a process executed in a namespace sees only specific network interfaces (e.g. those of the router), their own routing and ARP tables, their own firewall and NAT rules. Namespaces are able to guarantee L3 isolation, making possible for the different users to have overlapping IP addresses.

Therefore, multi-tenancy is achieved by the use of techniques as VLAN tagging and/or tunneling (VXLAN, GRE), whereas the L3 isolation is granted by the use of several Linux kernel network namespaces. However, it is important to know that Neutron provides multiple network abstractions to let the user be transparent to all these low-level details needed to deliver and enhance multi-tenancy in this distributed system. More details about this will be provided later.

### 4.3 The journey of a packet: the OpenStack VNI

Without any loss of generality, it is possible to explain the VNI by supposing an environment composed by a controller node, a compute node and a network node.

In general, the controller node does not contribute to the VNI, unless it provides also other functionalities (e.g. a controller node with computing facilities). On the other hand, in compute and network node Neutron will (by means of agents and plug-ins) run and create, manage and destroy software



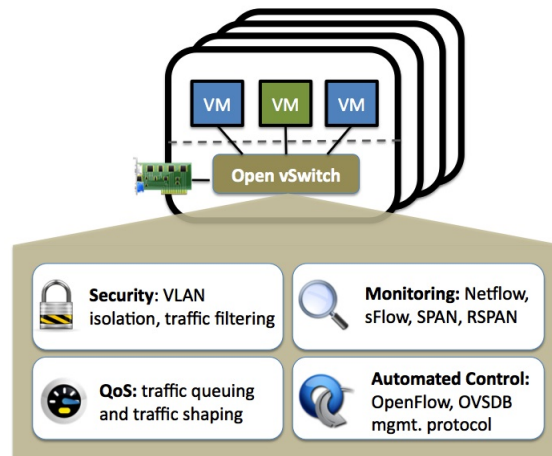


Figure 4.2: OpenVSwitch overview [61]

entities to implement its functions.

First of all, it is important to state the two possible Neutron deployments for the network hypervisor. The simplest solution implies an old component that virtualizes a bridge, called Linux Bridge [60]; the complexity increases with the OpenVSwitch [61] based solution. The OvS is an OpenFlow-enabled multilayer virtual switch, supporting standard management interfaces and protocols, as NetFlow, sFlow, etc.

For its completeness and its capacity to perform SDN mechanisms, as well as other reasons which will be stated later, here only the OvS deployment will be considered [62]. However, for any reader interested in the LB deployment, the concept will be similar but the presence of LBs instead of OvSs and, of course, related mechanisms to maintain the multi-tenancy [63].

First of all, as long as the OvS is considered as the network hypervisor, it is possible to state that for each node (compute and network) there is a Neutron agent running on it. This will initialize the node by creating, on behalf of neutron-server:

- an integration bridge *br-int*, which will act as a hub of the star network composed by the virtual instances;
- a bridge for each physical network connected to the node itself. This bridge might therefore be:

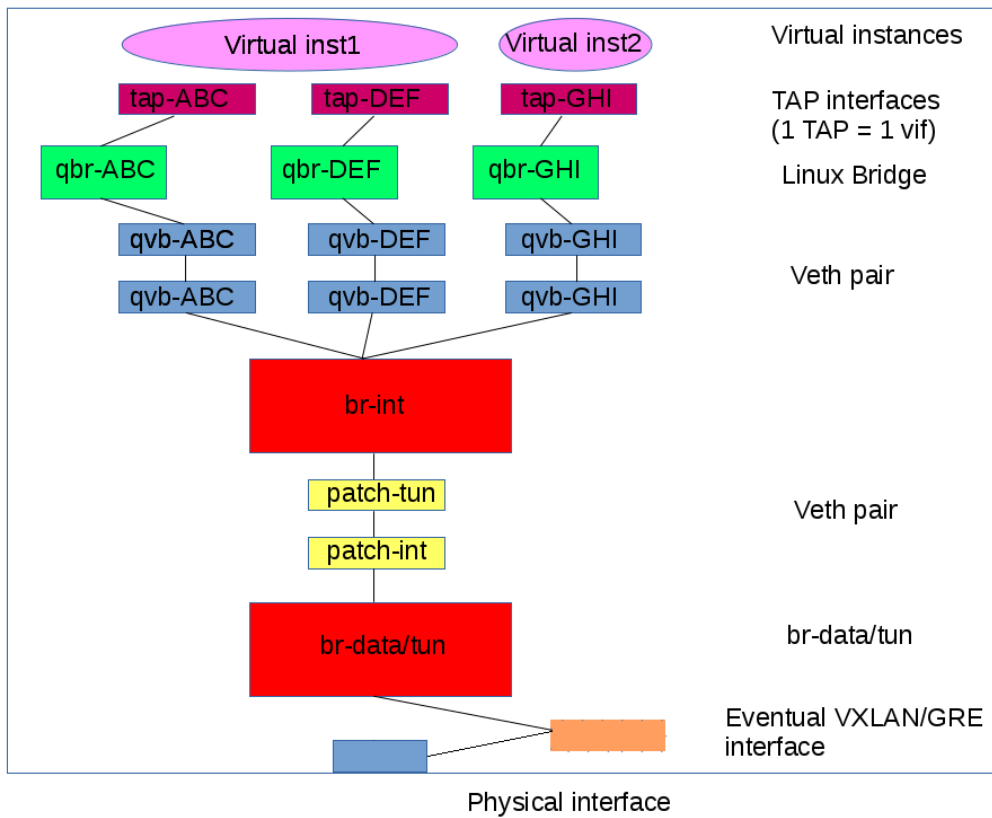


Figure 4.3: Compute node internals

- *br-ex* whenever it is related to the external network;
- *br-data*, *br-vlan* or *br-tun* whenever it is related to the data network, depending on the chosen tenant network solution.

To ease the explanation about the other internal components, two figures representing an example of default OvS-based compute node internals and an example of network node internals are shown respectively in Figure 4.3 and Figure 4.4. Moreover, the internals will be shown by starting from the compute node to reach the outside, as if the reader was following the flow of a packet from the virtual instance to the outside.

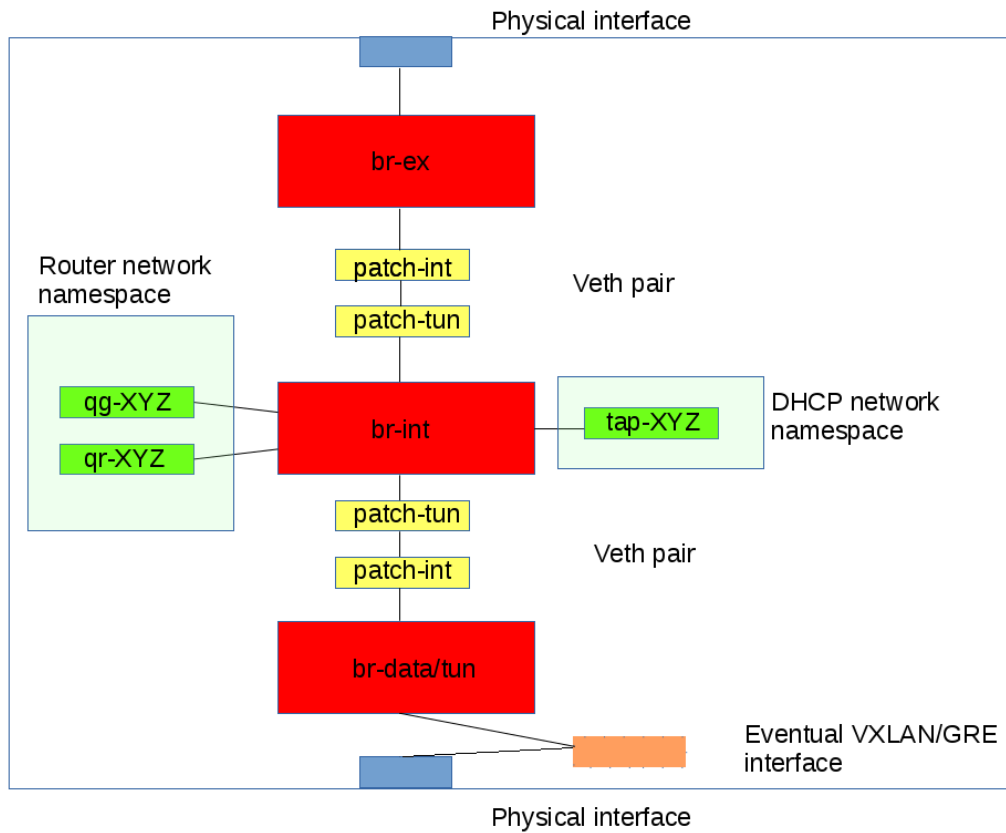


Figure 4.4: Network node internals

### 4.3.1 Compute node

In general, whenever virtualization techniques are considered, the virtual instance (e.g. the VM) has as many TAP interfaces as the number of virtual interfaces the instance has. The TAP interface indeed simulates a link layer device and operates with layer 2 packets like Ethernet frames and is the counterpart at host side of the guest's virtual interface. Ideally, the TAP should be attached directly to the integration bridge; however, this was not feasible because of how OpenStack security groups are implemented. The Security Groups are a set of rules that allow administrators and users to specify the type of traffic and direction (ingress/egress) that is allowed with respect to their instances; it is therefore a sort of firewall for instances. Indeed, OpenStack uses iptables rules on the TAP devices to implement security groups and

OvS was at the time of development not compatible with iptables rules applied directly on those TAP devices which are attached to an OvS port. This modality is called “iptables\_hybrid”; in general, also ebtables might be used if configured.

It is possible to list the firewall rules of the compute node related to a specific tap (the 11 characters after the word “tap” are the firsts 11 characters of the neutron port ID related to that interface):

```
[root@compute6 ~]# iptables -S | grep tapcb13fae5-5b
-A neutron-openvswi-FORWARD -m physdev --physdev-out tapcb13fae5-5b
  --physdev-is-bridged -m comment --comment "Direct traffic
  from the VM interface to the security group chain." -j neutron-
  openvswi-sg-chain
-A neutron-openvswi-FORWARD -m physdev --physdev-in tapcb13fae5-5b
  --physdev-is-bridged -m comment --comment "Direct traffic
  from the VM interface to the security group chain." -j neutron-
  openvswi-sg-chain
-A neutron-openvswi-INPUT -m physdev --physdev-in tapcb13fae5-5b
  --physdev-is-bridged -m comment --comment "Direct incoming
  traffic from VM to the security group chain." -j neutron-
  openvswi-ocb13fae5-5
-A neutron-openvswi-sg-chain -m physdev --physdev-out tapcb13fae5-5b
  --physdev-is-bridged -m comment --comment "Jump to the VM
  specific chain." -j neutron-openvswi-icb13fae5-5
-A neutron-openvswi-sg-chain -m physdev --physdev-in tapcb13fae5-5b
  --physdev-is-bridged -m comment --comment "Jump to the VM
  specific chain." -j neutron-openvswi-ocb13fae5-5
```

As it is possible to see from this listing and from Figure 4.5, there are several chains and rules associated to a tap. The actual iptables chain where the neutron-managed security groups are realized is the *neutron-openvswitch-sg-chain*. Instead, the chains *neutron-openvswi-ocb13fae5-5* and *neutron-openvswi-icb13fae5-5* are those controlling the traffic exiting from the instance (*neutron-openvswi-o\**, where *o* stays for output, intending the instance output) or directed to it (*neutron-openvswi-o\**, instance input).

The *neutron-openvswi-ocb13fae5-5* will be:

```
[root@compute6 ~]# iptables -S neutron-openvswi-ocb13fae5-5
-N neutron-openvswi-ocb13fae5-5
-A neutron-openvswi-ocb13fae5-5 -s 0.0.0.0/32 -d
  255.255.255.255/32 -p udp -m udp --sport 68 --dport 67 -m
  comment --comment "Allow DHCP client traffic." -j RETURN
-A neutron-openvswi-ocb13fae5-5 -j neutron-openvswi-scb13fae5-5
```

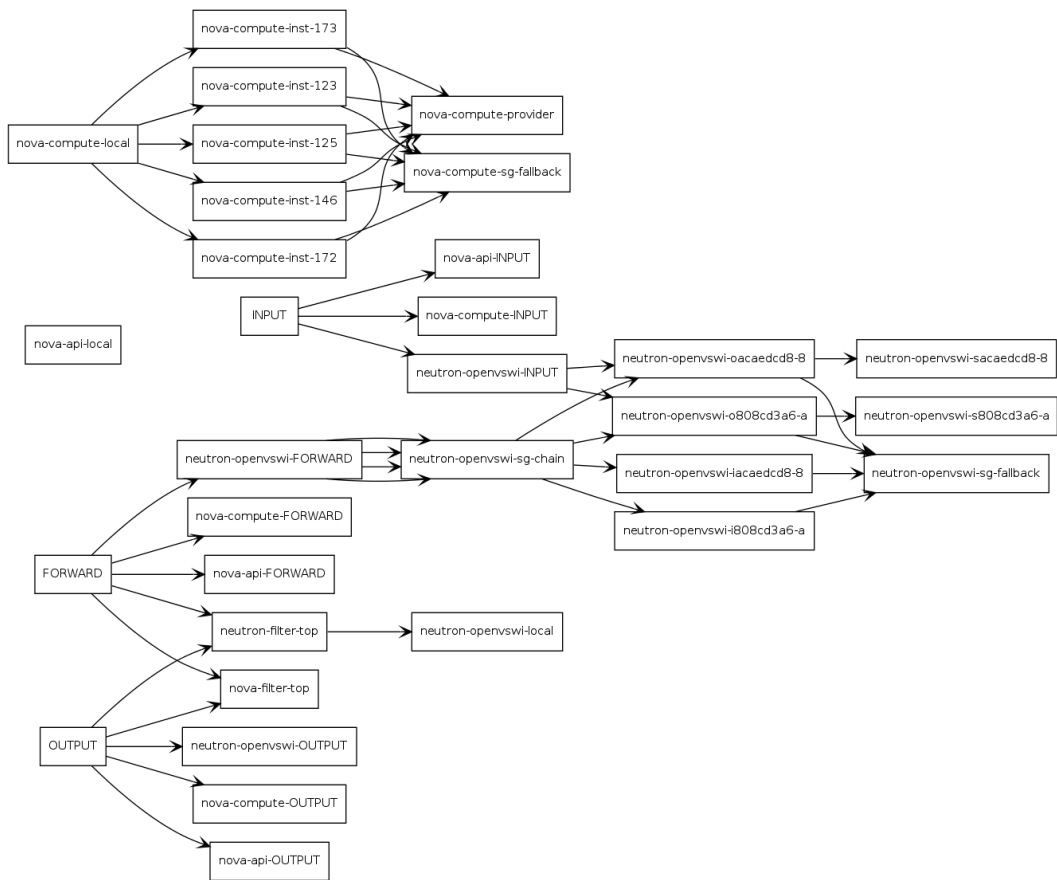


Figure 4.5: Iptables graphical representation

- ```

-A neutron-openvswi-ocb13fae5-5 -p udp -m udp --sport 68 --dport
  67 -m comment --comment "Allow DHCP client traffic." -j RETURN
-A neutron-openvswi-ocb13fae5-5 -p udp -m udp --sport 67 -m udp --
  dport 68 -m comment --comment "Prevent DHCP Spoofing by VM." -
  j DROP
-A neutron-openvswi-ocb13fae5-5 -m state --state RELATED,
  ESTABLISHED -m comment --comment "Direct packets associated
  with a known session to the RETURN chain." -j RETURN
-A neutron-openvswi-ocb13fae5-5 -j RETURN
-A neutron-openvswi-ocb13fae5-5 -m state --state INVALID -m
  comment --comment "Drop packets that appear related to an
  existing connection (e.g. TCP ACK/FIN) but do not have an entry
  in conntrack." -j DROP

```

```
-A neutron-openvswi-icb13fae5-5 -m comment --comment "Send
unmatched traffic to the fallback chain." -j neutron-openvswi-
sg-fallback
```

whereas the `neutron-openvswi-icb13fae5-5` will be:

```
[root@compute6 ~]# iptables -S neutron-openvswi-icb13fae5-5
-N neutron-openvswi-icb13fae5-5
-A neutron-openvswi-icb13fae5-5 -m state --state RELATED,
ESTABLISHED -m comment --comment "Direct packets associated
with a known session to the RETURN chain." -j RETURN
-A neutron-openvswi-icb13fae5-5 -s 192.168.10.1/32 -p udp -m udp
--sport 67 -m udp --dport 68 -j RETURN
-A neutron-openvswi-icb13fae5-5 -p icmp -j RETURN
-A neutron-openvswi-icb13fae5-5 -p tcp -m tcp --dport 22 -j RETURN
-A neutron-openvswi-icb13fae5-5 -p tcp -m tcp -m multiport --
dports 1:65535 -j RETURN
-A neutron-openvswi-icb13fae5-5 -m set --match-set NIPv49fbb4027
-302f-40b2-8e4e- src -j RETURN
-A neutron-openvswi-icb13fae5-5 -p udp -m udp -m multiport --
dports 1:65535 -j RETURN
-A neutron-openvswi-icb13fae5-5 -m state --state INVALID -m
comment --comment "Drop packets that appear related to an
existing connection (e.g. TCP ACK/FIN) but do not have an entry
in conntrack." -j DROP
-A neutron-openvswi-icb13fae5-5 -m comment --comment "Send
unmatched traffic to the fallback chain." -j neutron-openvswi-
sg-fallback
```

where it is possible to see the presence of the user-managed decision to open all the TCP and UDP ports, as well as the SSH and ICMP. If the traffic is not matched by these rules (on the ingress or egress direction, depending on the type of traffic), it will be sent to the `neutron-openvswi-sg-fallback`, which is a single DROP rule.

It is remarkable to say that a new way to perform firewalling is now present in OpenStack and will be shown later in this document, as it is one of the main points on which the measurements will focus.

Therefore, it is possible to see in Figure 4.6 that the traffic exiting from a virtual instance reaches first the TAP interface and then a Linux Bridge device. This device connects the tap to the integration bridge through a veth pair. This is a virtual Ethernet cable where the two interfaces seen by the kernel level are representing the two extremities of the cable itself: what enters on one side, exits on the other one. Therefore, the Linux Bridge is present to make possible

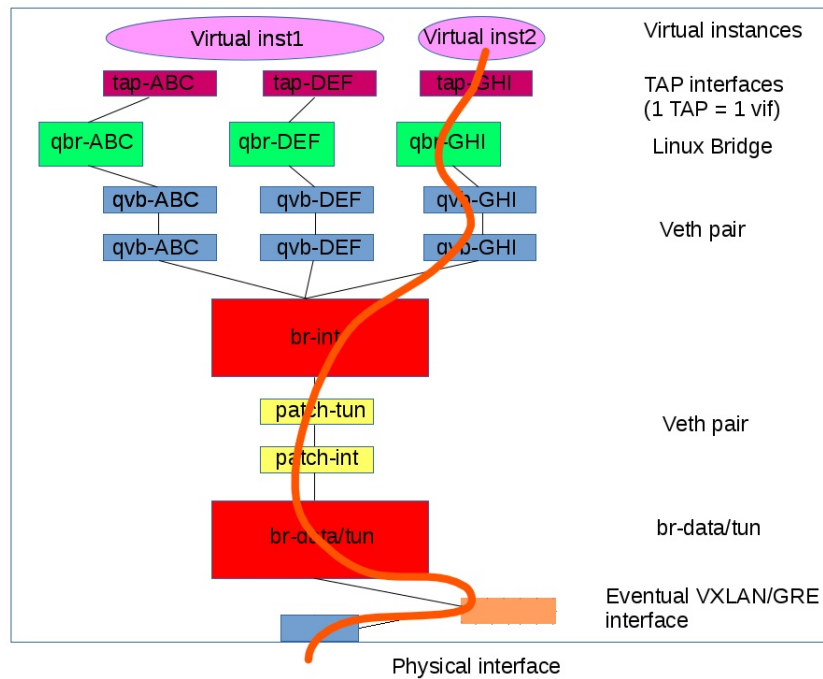


Figure 4.6: Egress instance flow in an OpenStack compute

to implement the security groups rules on the tap, whereas the role of the veth pair is just to connect the Linux Bridge to the OvS.

The port of the OvS connected to any instance's veth pair is in access mode for a certain VLAN ID, which is related to the tenant network the VM belongs to and internally decided by Neutron, in particular by the L2 agent. This is one of those actions performed by Neutron to maintain the multi-tenancy in OpenStack, as previously stated.

Therefore, the bridge br-int will receive VLAN tagged packets from the instances; this helps it to recognize to which network they belong. When the instance is created, the neutron-openvswitch-agent installs OpenFlow rules to make possible to forward traffic the traffic coming from and going to it. Indeed, the br-int performs VLAN tagging and un-tagging for traffic coming from and to the instances. It is possible to demonstrate that two virtual instances belonging to different tenant networks are not able to interact directly simply through L2 OVS switching, even though they reside in the same compute node. On the other hand, if they were on the same tenant network the

interaction would have been possible directly through the br-int, in a complete L2 switching mechanism.

```
[root@compute6 ~]# ovs-vsctl show
892c5743-ebab-4f0a-8459-0a927ffe6863
  Manager "ptcp:6640:127.0.0.1"
    is_connected: true
  Bridge br-tun
    Controller "tcp:127.0.0.1:6633"
      is_connected: true
    fail_mode: secure
    Port "vxlan-0a7d0005"
      Interface "vxlan-0a7d0005"
        type: vxlan
        options: {df_default="true", in_key=flow,
                  local_ip="10.125.0.6", out_key=flow, remote_ip
                  = "10.125.0.5"}
    Port br-tun
      Interface br-tun
        type: internal
    Port patch-int
      Interface patch-int
        type: patch
        options: {peer=patch-tun}
    Port "vxlan-0a7d0007"
      Interface "vxlan-0a7d0007"
        type: vxlan
        options: {df_default="true", in_key=flow,
                  local_ip="10.125.0.6", out_key=flow, remote_ip
                  = "10.125.0.7"}
  Bridge br-int
    Controller "tcp:127.0.0.1:6633"
      is_connected: true
    fail_mode: secure
    Port "qvocb13fae5-5b"
      tag: 1
      Interface "qvocb13fae5-5b"
    Port "qvo39215413-ba"
      tag: 2
      Interface "qvo39215413-ba"
    Port br-int
      Interface br-int
        type: internal
    Port patch-tun
      Interface patch-tun
```



```

        type: patch
        options: {peer=patch-int}
    ovs_version: "2.5.0"

```

*# Example of a br-int's rules*

```
[root@compute6 ~]# ovs-ofctl dump-flows br-int
```

```
NXSTFLOW reply (xid=0x4):
```

```

cookie=0xa7cbffe5e9447996 , duration=22970.593s, table=0,
  n_packets=0, n_bytes=0, idle_age=65534, priority=10,icmp6,
  in_port=45,icmp_type=136 actions=resubmit(,24)
cookie=0xa7cbffe5e9447996 , duration=22970.584s, table=0,
  n_packets=0, n_bytes=0, idle_age=65534, priority=10,icmp6,
  in_port=44,icmp_type=136 actions=resubmit(,24)
cookie=0xa7cbffe5e9447996 , duration=22970.591s, table=0,
  n_packets=773, n_bytes=32466, idle_age=3017, priority=10,arp,
  in_port=45 actions=resubmit(,24)
cookie=0xa7cbffe5e9447996 , duration=22970.582s, table=0,
  n_packets=1759, n_bytes=73878, idle_age=964, priority=10,arp,
  in_port=44 actions=resubmit(,24)
cookie=0xa7cbffe5e9447996 , duration=22970.595s, table=0,
  n_packets=2093585249, n_bytes=31746710699846, idle_age=3022,
  priority=9,in_port=45 actions=resubmit(,25)
cookie=0xa7cbffe5e9447996 , duration=455046.537s, table=0,
  n_packets=989591080, n_bytes=697284235563, idle_age=756,
  hard_age=65534, priority=0 actions=NORMAL
cookie=0xa7cbffe5e9447996 , duration=455046.539s, table=23,
  n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534,
  priority=0 actions=drop
cookie=0xa7cbffe5e9447996 , duration=247405.544s, table=24,
  n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534,
  priority=2,icmp6,in_port=45,icmp_type=136,nd_target=fe80::f816
:3eff:fe9b:552b actions=NORMAL
cookie=0xa7cbffe5e9447996 , duration=22970.585s, table=24,
  n_packets=0, n_bytes=0, idle_age=22970, priority=2,icmp6,
  in_port=44,icmp_type=136,nd_target=fe80::f816:3eff:fe03:92fa
actions=NORMAL
cookie=0xa7cbffe5e9447996 , duration=22970.592s, table=24,
  n_packets=6, n_bytes=252, idle_age=3017, priority=2,arp,
  in_port=45,arp_spa=192.168.10.5 actions=resubmit(,25)
cookie=0xa7cbffe5e9447996 , duration=22970.583s, table=24,
  n_packets=28, n_bytes=1176, idle_age=964, priority=2,arp,
  in_port=44,arp_spa=192.168.100.12 actions=resubmit(,25)
cookie=0xa7cbffe5e9447996 , duration=455046.536s, table=24,
  n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534,
  priority=0 actions=drop

```

```

cookie=0xa7cbffe5e9447996 , duration=22970.599s , table=25,
  n_packets=2093586020, n_bytes=31746710732172, idle_age=3017,
  priority=2,in_port=45,dl_src=fa:16:3e:9b:55:2b actions=NORMAL
cookie=0xa7cbffe5e9447996 , duration=22970.590s , table=25,
  n_packets=4031440, n_bytes=6296979873, idle_age=964, priority
=2,in_port=44,dl_src=fa:16:3e:03:92:fa actions=NORMAL

```

As it is possible to see from the above listing, br-int has two qvo interfaces attached, related to virtual interfaces belonging to different tenant networks, as shown in the tag ( $\Rightarrow$  VLAN tag) field. Therefore untagged outbound traffic related to the qvpcb13fae5-5b interface will be assigned VLAN ID 1, and inbound traffic with VLAN ID 1 will be stripped of its VLAN tag and sent out from this port.

Table 0 of the br-int considers all the ICMP6 and ARP traffic by resubmitting it to Table 24. Moreover, the traffic coming from the veth pair that connects br-int with br-tun is resubmitted to Table 25, whereas all the remaining traffic will be treated with a NORMAL action. Table 23 is also called the “canary rule table”, as it contains a DROP rule that is used only in absence of instances. Moreover, this rule is continuously checked by Neutron and, if at a certain moment it is not found, Neutron will restore all the previous rules as a matter of security. Finally Table 24 and Table 25 are related to the ARP and ICMP6 management.

As long as the instance traffic considered here is directed to the outside (e.g. the Internet), the traffic will be switched by the OvS to go through another veth pair that connects the br-int to the br-tun (or br-data, or br-vlan, when a non-tunneling tenant networks environment is deployed). On br-tun, other OpenFlow rules are implemented to translate the tagged instance egress traffic into VXLAN or GRE tunnels and viceversa: indeed, br-tun is attached to N-1 VXLAN/GRE interfaces (which is also the number of tunnels and where N is the number of nodes connected to the data tunnel network) that will encapsulate the packet. For instance, with one network node and two compute nodes, each node will have two interfaces, each one related to a tunnel shared with another node.

As it is possible to see from the next listing, br-tun makes use of multiple OvS tables. It is also shown that, in addition to the rule matching a VLAN ID and setting the tunnel it before sending to the tunnel, there are rules to match multicast traffic with a certain tunnel id to redirect to the veth pair: these are those rules related to the ingress traffic.

```

[root@compute6 ~]# ovs-ofctl dump-flows br-tun
NXST.FLOW reply (xid=0x4):
  cookie=0xb8aefdd94951cae5, duration=32025.359s, table=0,
    n_packets=1065110, n_bytes=5539393619, idle_age=3, priority=1,
    in_port=2 actions=resubmit(,2)
  cookie=0xb8aefdd94951cae5, duration=31976.629s, table=0,
    n_packets=51299, n_bytes=56395081, idle_age=3, priority=1,
    in_port=3 actions=resubmit(,4)
  cookie=0xb8aefdd94951cae5, duration=32025.358s, table=0,
    n_packets=1955, n_bytes=151833, idle_age=800, priority=0
    actions=drop
  cookie=0xb8aefdd94951cae5, duration=32025.357s, table=2,
    n_packets=1064154, n_bytes=5539287179, idle_age=3, priority=0,
    dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit
    (,20)
  cookie=0xb8aefdd94951cae5, duration=32025.355s, table=2,
    n_packets=956, n_bytes=106440, idle_age=14819, priority=0,
    dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit
    (,22)
  cookie=0xb8aefdd94951cae5, duration=32025.354s, table=3,
    n_packets=0, n_bytes=0, idle_age=65534, priority=0 actions=
    drop
  cookie=0xb8aefdd94951cae5, duration=31978.079s, table=4,
    n_packets=1740, n_bytes=173266, idle_age=1078, priority=1,
    tun_id=0x10 actions=mod_vlan_vid:1,resubmit(,10)
  cookie=0xb8aefdd94951cae5, duration=31977.962s, table=4,
    n_packets=49559, n_bytes=56221815, idle_age=3, priority=1,
    tun_id=0x2b actions=mod_vlan_vid:2,resubmit(,10)
  cookie=0xb8aefdd94951cae5, duration=32025.353s, table=4,
    n_packets=12, n_bytes=768, idle_age=38108, priority=0 actions=
    drop
  cookie=0xb8aefdd94951cae5, duration=32025.352s, table=6,
    n_packets=0, n_bytes=0, idle_age=65534, priority=0 actions=
    drop
  cookie=0xb8aefdd94951cae5, duration=32025.350s, table=10,
    n_packets=1755299, n_bytes=5723071599, idle_age=3, priority=1
    actions=learn(table=20,hard_timeout=300,priority=1,cookie=0
    xb8aefdd94951cae5,NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=
    NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID
    []->NXM_NX_TUN_ID[],output:OXM_OF_IN_PORT[]),output:2
  cookie=0xb8aefdd94951cae5, duration=31894.451s, table=20,
    n_packets=1629, n_bytes=148718, idle_age=1078, priority=2,
    dl_vlan=1,dl_dst=fa:16:3e:e6:2f:c5 actions=strip_vlan,load:0
    x10->NXM_NX_TUN_ID[],output:3
  cookie=0xb8aefdd94951cae5, duration=31892.379s, table=20,

```

```

n_packets=49415, n_bytes=8829803, idle_age=3, priority=2,
dl_vlan=2,dl_dst=fa:16:3e:6c:5a:6c actions=strip_vlan ,load:0
x2b->NXMNX_TUN_ID[] ,output:3
cookie=0xb8aefdd94951cae5 , duration=31892.377s , table=20,
n_packets=6, n_bytes=252, idle_age=31368, priority=2,dl_vlan
=2,dl_dst=fa:16:3e:85:83:e1 actions=strip_vlan ,load:0x2b->
NXMNX_TUN_ID[] ,output:3
cookie=0xb8aefdd94951cae5 , duration=1212.720s , table=20,
n_packets=0, n_bytes=0, hard_timeout=300, idle_age=1212,
hard_age=3, priority=1,vlan_tci=0x0002/0x0fff ,dl_dst=fa:16:3e
:6c:5a:6c actions=load:0->NXM_OF_VLAN_TCI[] ,load:0x2b->
NXMNX_TUN_ID[] ,output:3
cookie=0xb8aefdd94951cae5 , duration=32025.349s , table=20,
n_packets=4, n_bytes=296, idle_age=800, priority=0 actions=
resubmit( ,22)
cookie=0xb8aefdd94951cae5 , duration=31894.453s , table=22,
n_packets=135, n_bytes=16646, idle_age=800, priority=1,dl_vlan
=1 actions=strip_vlan ,load:0x10->NXMNX_TUN_ID[] ,output:3
cookie=0xb8aefdd94951cae5 , duration=31892.381s , table=22,
n_packets=75, n_bytes=9054, idle_age=31353, priority=1,dl_vlan
=2 actions=strip_vlan ,load:0x2b->NXMNX_TUN_ID[] ,output:3
cookie=0xb8aefdd94951cae5 , duration=32025.348s , table=22,
n_packets=131, n_bytes=10578, idle_age=31426, priority=0
actions=drop

```

Thus, the encapsulated packet will be sent on the tunnel data network that connects the compute node to the network node; however, if the instance traffic was directed to an instance in an another compute node, the tunnel over which send the packet would have been the one connecting the two compute nodes.

### 4.3.2 Network node

The traffic is therefore reaching the end of the tunnel by the corresponding VXLAN/GRE interface and getting at the br-tun where, in a very similar way to the compute node's br-tun, performs the VXLAN/GRE translation to VLAN and sends the traffic over the veth pair. The br-int as well will make use of OpenFlow to redirect this traffic to the network namespace of the virtual router (qr-\* interface inside the qrouter-\*) or of the dhcp (tap\* inside the qdhcp\*-). From the next listing it is important to note that the MAC address of the tap-\* interface inside the dhcp namespace is the one that is present in some rules of the compute node's br-tun. Moreover, it is also possible to see

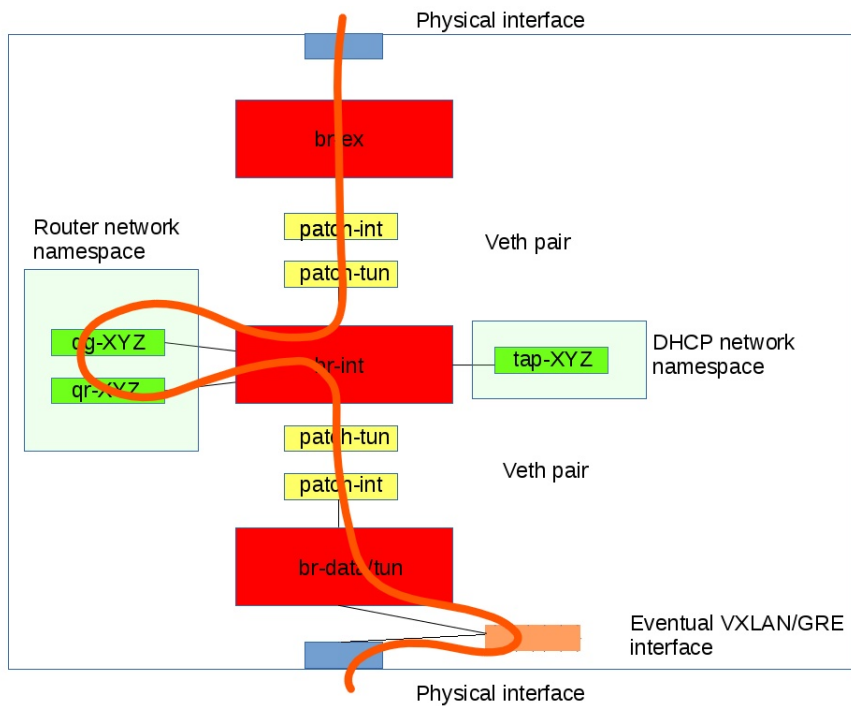


Figure 4.7: Egress instance flow in an OpenStack network node

the dnsmasq process.

```
[root@ ~]# ip netns list
qdhcp-b1fde838-71ce-40ef-abc0-46b231055fb1
qrouter-bd7895d0-4dc4-4301-92c5-42fae8569fec

[root@network01 ~]# ip netns exec qdhcp-b1fde838-71ce-40ef-abc0-46
b231055fb1 ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1 (Local Loopback)
    RX packets 154 bytes 45112 (44.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 154 bytes 45112 (44.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tap2134f8be-10: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1450
```

```

inet 192.168.10.1 netmask 255.255.255.0 broadcast
    192.168.10.255
inet6 fe80::f816:3eff:fee6:2fc5 prefixlen 64 scopeid 0
    x20<link>
ether fa:16:3e:e6:2f:c5 txqueuelen 1000 (Ethernet)
RX packets 30408 bytes 201865016 (192.5 MiB)
RX errors 0 dropped 4 overruns 0 frame 0
TX packets 16815 bytes 1474904 (1.4 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

```

[root@network01 ~]# ps -fe | grep b1fde838-71ce-40ef-abc0-46
b231055fb1
neutron    3076      1  0 11:05 ?          00:00:01 /usr/bin/python2 /
bin/neutron-ns-metadata-proxy --pid_file=/var/lib/neutron/
external/pids/b1fde838-71ce-40ef-abc0-46b231055fb1.pid --
metadata_proxy_socket=/var/lib/neutron/metadata_proxy --
network_id=b1fde838-71ce-40ef-abc0-46b231055fb1 --state_path=/
var/lib/neutron --metadata_port=80 --metadata_proxy_user=989 --
metadata_proxy_group=986 --log_file=neutron-ns-metadata-proxy-
b1fde838-71ce-40ef-abc0-46b231055fb1.log --log_dir=/var/log/
neutron
nobody    22320      1  0 13:43 ?          00:00:00 dnsmasq --no-hosts
--no-resolv --strict-order --except-interface=lo --pid_file=/
var/lib/neutron/dhcp/b1fde838-71ce-40ef-abc0-46b231055fb1/pid
--dhcp-hostsfile=/var/lib/neutron/dhcp/b1fde838-71ce-40ef-abc0
-46b231055fb1/host --addn-hosts=/var/lib/neutron/dhcp/b1fde838
-71ce-40ef-abc0-46b231055fb1/addn_hosts --dhcp-optsfile=/var/
lib/neutron/dhcp/b1fde838-71ce-40ef-abc0-46b231055fb1/opts --
dhcp-leasefile=/var/lib/neutron/dhcp/b1fde838-71ce-40ef-abc0-46
b231055fb1/leases --dhcp-match=set:ipxe,175 --bind-interfaces
--interface=tap2134f8be-10 --dhcp-range=set:tag0,192.168.10.0,
static,86400s --dhcp-option-force=option:mtu,1450 --dhcp-lease-
max=256 --conf-file= --domain=openstacklocal
root     25487 24428  0 22:40 pts/16   00:00:00 grep --color=auto
b1fde838-71ce-40ef-abc0-46b231055fb1

```

The router namespace is organized in a similar way. The `qg-*` interface connects the router to the external gateway, where as the `qr-*` connects the router to the integration-bridge. The routing tables are showing that the default gateway is the gateway on the external network that has been set during the network creation. Moreover, the netfilter `nat` table is responsible for the floating IP addresses associated to the instances as well as for the general NAT functionalities, by performing a SNAT operation.

Therefore, the traffic from `br-int` will enter into the network namespace

of the router, will go through the qr\*- and qg-\* and NAT actions will be performed on it, to be then sent again to the br-int. This bridge will now recognize that the incoming traffic is related to the external network and will forward it to the br-ex via another patch port.

Finally, the br-ex will perform other OpenFlow rules (simple switching) to let this traffic go out of the cluster. Similarly, the inverse process applies for an ingress flow.

```
[root@network01 ~]# ip netns exec qrouter-bd7895d0-4dc4-4301-92c5-42fae8569fec ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1 (Local Loopback)
    RX packets 210 bytes 21696 (21.1 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 210 bytes 21696 (21.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

qg-bc9626ae-d2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1500
    inet 10.250.0.117 netmask 255.255.255.0 broadcast
    10.250.0.255
    inet6 fe80::f816:3eff:fee9:fb20 prefixlen 64 scopeid 0
    x20<link>
    ether fa:16:3e:e9:fb:20 txqueuelen 1000 (Ethernet)
    RX packets 369371 bytes 433274239 (413.2 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 337010 bytes 58322351 (55.6 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

qr-cff634eb-5c: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu
1450
    inet 192.168.110.254 netmask 255.255.255.0 broadcast
    192.168.110.255
    inet6 fe80::f816:3eff:fed4:fad2 prefixlen 64 scopeid 0
    x20<link>
    ether fa:16:3e:d4:fa:d2 txqueuelen 1000 (Ethernet)
    RX packets 2280719 bytes 71059154 (67.7 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2279503 bytes 103108021 (98.3 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@network01 ~]# ip netns exec qrouter-bd7895d0-4dc4-4301-92c5
```

```

-42fae8569fec route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref
      Use Iface
0.0.0.0          10.250.0.252    0.0.0.0          UG    0    0
      0 qg-bc9626ae-d2
10.250.0.0      0.0.0.0         255.255.255.0    U    0    0
      0 qg-bc9626ae-d2
192.168.110.0   0.0.0.0         255.255.255.0    U    0    0
      0 qr-cff634eb-5c

[root@network01 ~]# ip netns exec qrouter-bd7895d0-4dc4-4301-92c5
-42fae8569fec iptables -t nat -S
-P PREROUTING ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT
-P POSTROUTING ACCEPT
-N neutron-l3-agent-OUTPUT
-N neutron-l3-agent-POSTROUTING
-N neutron-l3-agent-PREROUTING
-N neutron-l3-agent-float-snat
-N neutron-l3-agent-snat
-N neutron-postrouting-bottom
-A PREROUTING -j neutron-l3-agent-PREROUTING
-A OUTPUT -j neutron-l3-agent-OUTPUT
-A POSTROUTING -j neutron-l3-agent-POSTROUTING
-A POSTROUTING -j neutron-postrouting-bottom
-A neutron-l3-agent-OUTPUT -d 10.250.0.107/32 -j DNAT --to-
destination 192.168.110.12
-A neutron-l3-agent-POSTROUTING ! -i qg-bc9626ae-d2 ! -o qg-
bc9626ae-d2 -m conntrack ! --ctstate DNAT -j ACCEPT
-A neutron-l3-agent-PREROUTING -d 169.254.169.254/32 -i qr+ -p
tcp -m tcp --dport 80 -j REDIRECT --to-ports 9697
-A neutron-l3-agent-PREROUTING -d 10.250.0.107/32 -j DNAT --to-
destination 192.168.110.12
-A neutron-l3-agent-float-snat -s 192.168.110.12/32 -j SNAT --to-
source 10.250.0.107
-A neutron-l3-agent-snat -j neutron-l3-agent-float-snat
-A neutron-l3-agent-snat -o qg-bc9626ae-d2 -j SNAT --to-source
10.250.0.117
-A neutron-l3-agent-snat -m mark ! --mark 0x2/0xffff -m conntrack
--ctstate DNAT -j SNAT --to-source 10.250.0.117
-A neutron-postrouting-bottom -m comment --comment "Perform
source NAT on outgoing traffic." -j neutron-l3-agent-snat

```



## 4.4 Remarks

First of all, it is also possible to have a much more simpler deployment, where Linux Bridges are used instead of OvS; however, it lacks of SDN mechanisms and is not suitable for production environments.

In general, each time a new network is created, neutron-server will contact the neutron-dhcp-agent to create a new network namespace (the `qdhcp-*`); then, inside the namespace a virtual interface is created and a dnsmasq process is spawned by using that vif. In addition to it, whenever a router is added to the network topology, the neutron-l3-agent creates a network namespace for it, as well as a vif inside it, configures its routing tables and NAT rules. It is remarkable to say that only the network node will be the location for all the network namespaces, both routers and DHCP servers.

Instead, in the case of a new instance creation, nova-api is in charge (after having contacted the database, nova-scheduler, etc.) of contacting the nova-compute agent running on the chosen node to instruct it. Also neutron-server will be contacted by asking for a port allocation (and a fixed IP for the instance); the neutron-openvswitch-agent on the target node will therefore configure the virtual bridges via the OpenFlow protocol. In the meanwhile, in the network node neutron-dhcp will spawn the IP address decided.

An example of a VNI with VLAN tenant networks is shown in Figure 4.8. As it is possible to see, nova-compute is in charge of configuring the interaction among the hypervisor that created the virtual instances, in this case VMs, and the networking-related part, which is in turn taken into account by the Neutron L2 agent (neutron-openvswitch-agent). VM01 and VM02 have an interface on one tenant network, which the L2 agent internally assigned on the VLAN ID 1, whereas VM02 and VM03 are on another tenant network, that is assigned to VLAN ID 2. When the traffic has to exit the node, the br-int first forwards it to the port attached to the veth pair related to the br-eth (a different name for br-data or br-vlan); this switch will then perform a VLAN ID conversion with the VLAN IDs that the user decided among those defined in the ML2 plugin configuration. On the contrary, when traffic flows are entering the compute node, it is duty of the br-int to perform the conversion. Finally, in the case in which a tunneling technique is employed for the tenant network, the br-tun egress conversion (as well as the br-int ingress one) will be between a VLAN ID and a VXLAN/GRE segmentation identifier.

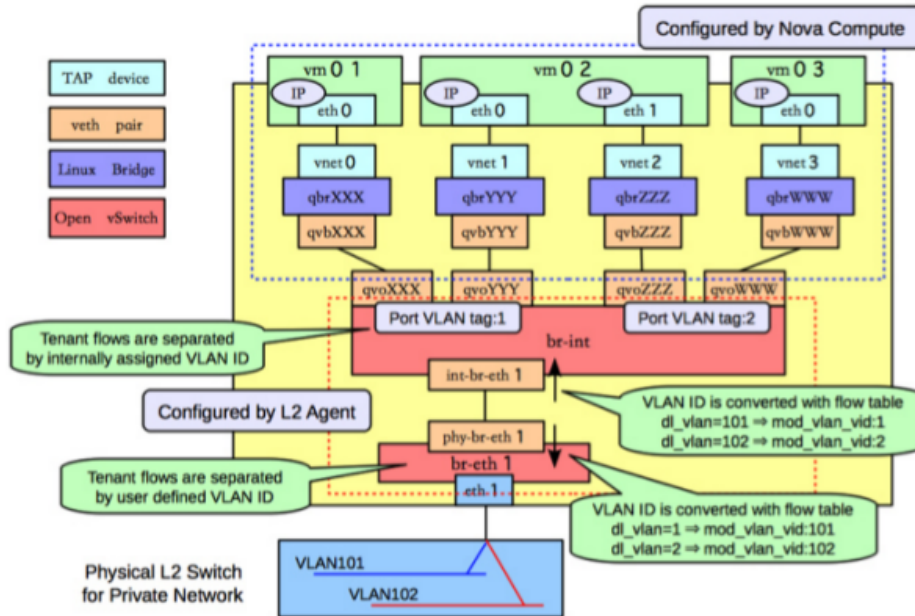


Figure 4.8: Example of a VNI with VLAN tenant network [64]

## 4.5 Performance and improvements

As shown in different works, as [65], a Cloud-based architecture poses some limitations to the network performance. In particular, this depends on the hosting hardware maximum capacity, but also to the complex components inside the VNI. For instance, it is possible to see that in OpenStack the bottleneck is represented by the Linux Bridge. In general, additional components between instances and physical network infrastructure cause scalability and performance problems.

Therefore, it is extremely vital to reconfigure the OpenStack virtual network architecture: the Linux Bridge should be deleted or at least replaced with something more performant. It is important to remark that OpenStack uses iptables rules on the TAP devices to implement security groups and the LB comes into play as OvS, historically, was not compatible with iptables rules applied directly on TAP devices connected to a latter's port. Some of the solutions to this problem are shown later [66].

### 4.5.1 Open vSwitch native firewall

To alleviate these scalability and performance problems, the OvS agent introduced an optional firewall driver that natively implements security groups as flows in OVS rather than the iptables Linux bridge solution. This is intended to increase scalability and performance, which is one of the topics that this document is intended to state [67]. The OvS driver has the same APIs as the current iptables firewall driver, keeping the state of security groups and ports inside the firewall. All the firewall APIs is controlled by security group RPC methods, which means that the firewall driver does not have any logic of which the port should be updated based on the provided changes: it only accomplishes actions when called from the controller.

First of all, every connection is split into ingress and egress processes, based respectively on the input or output port. Each port contains the initial hard-coded flows for ARP, DHCP and established connections, which are accepted by default. To detect established connections, a flow must be marked by the Linux *conntrack* module first with an `action=ct()` rule. An accepted flow means that ingress packets for the connection are directly sent to the port, and egress packets are left to be normally switched by the integration bridge. Connections that are not matched by the above rules are sent to either the ingress or egress filtering table, depending on their direction. In this modality, the OvS will make use of separate tables to store the security group rules, for an easier rule detection during removal.

Security group rules are treated differently regarding the absence or the presence of a remote group ID. A security group rule without a remote group ID is expanded into several OpenFlow rules (this is done by the method `create_flows_from_rule_and_port`), whereas a security group rule with a remote group ID is expressed by three sets of flows. The first two sets are conjunctive flows which will be described in the next paragraph. The third set matches on the conjunction IDs and does accept actions.

Whenever a security group rule with a remote group ID is considered, flows that match on `nw_src` for `remote_group_id` addresses and match on `dl_dst` for port MAC addresses are needed (for ingress rules as well as for egress rules). Without conjunction, this results in  $O(n*m)$  flows where  $n$  and  $m$  are respectively number of ports in the remote group ID and the port security group. A `conj_id` is therefore allocated for each `(remote_group_id, security_group_id, direction, ethertype)` tuple. The same `conj_id` is shared between security group

rules if multiple rules belong to the same tuple above. Conjunctive flows consist of 2 dimensions. Flows that belong to the dimension 1 of 2 are generated by the method `create_flows_for_ip_address` and are in charge of IP address based filtering specified by their remote group IDs. Flows that belong to the dimension 2 of 2 are generated by the method `create_flows_from_rule_and_port` and modified by the method `substitute_conjunction_actions`, which represents the portion of the rule other than its remote group ID [68].

### Rule explanation with an example

The following example presents two ports on the same host. They have different security groups and there is ICMP traffic allowed from first security group to the second security group. Ports have following attributes:

- Port 1
  - plugged to the port 1 in OVS bridge
  - ip address: 192.168.0.1
  - mac address: fa:16:3e:a4:22:10
  - security group 1: can send icmp packets out
  - allowed address pair: 10.0.0.1/32, fa:16:3e:8c:84:13
  
- Port 2
  - plugged to the port 2 in OVS bridge
  - ip address: 192.168.0.2
  - mac address: fa:16:3e:24:57:c7
  - security group 2: can receive icmp packets from security group 1
  - allowed address pair: 10.1.0.0/24, fa:16:3e:8c:84:14

The *table 0* contains a low priority rule to continue packet processing in *table 60*, which is called the *TRANSIENT* table. The table 0 is left for use to other features that take precedence over firewall. The only requirement is that after the feature has finished with its processing, it passes the packets for processing to the TRANSIENT table. This TRANSIENT table distinguishes the traffic as ingress or egress and loads to register 5 values identifying port

traffic. Egress flow is then determined by switch port number and ingress flow is determined by destination MAC address. The register 6 instead contains port tag to isolate connections into separate conntrack zones.

```

table=60, priority=100,in_port=1 actions=load:0x1->NXMNX_REG5[],
load:0x284->NXMNX_REG6[], resubmit(,71)
table=60, priority=100,in_port=2 actions=load:0x2->NXMNX_REG5[],
load:0x284->NXMNX_REG6[], resubmit(,71)
table=60, priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:a4:22:10
actions=load:0x1->NXMNX_REG5[], load:0x284->NXMNX_REG6[],
resubmit(,81)
table=60, priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:8c:84:13
actions=load:0x1->NXMNX_REG5[], load:0x284->NXMNX_REG6[],
resubmit(,81)
table=60, priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:24:57:c7
actions=load:0x2->NXMNX_REG5[], load:0x284->NXMNX_REG6[],
resubmit(,81)
table=60, priority=90,dl_vlan=0x284,dl_dst=fa:16:3e:8c:84:14
actions=load:0x2->NXMNX_REG5[], load:0x284->NXMNX_REG6[],
resubmit(,81)
table=60, priority=0 actions=NORMAL

```

The following table 71 implements ARP and IP spoofing protection; moreover, it allows traffic for obtaining IP addresses (this is valid for dhcp, dhcpv6, slaac, ndp) for egress traffic and allows ARP replies. It also identifies untracked connections which are processed later with information obtained from conntrack. It is important to remark the zone=NXMNX\_REG6[0..15] in the action field when obtaining information from conntrack. This shows that every port has its own conntrack zone defined by value in register 6. It is there to avoid to accept established traffic that belongs to different port with same conntrack parameters.

```

# allow ICMPv6 traffic for multicast listeners, neighbour
solicitation and neighbour advertisement
table=71, priority=95,icmp6,reg5=0x1,in_port=1,icmp_type=130
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x1,in_port=1,icmp_type=131
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x1,in_port=1,icmp_type=132
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x1,in_port=1,icmp_type=135
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x1,in_port=1,icmp_type=136
actions=NORMAL

```

```

table=71, priority=95,icmp6,reg5=0x2,in_port=2,icmp_type=130
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x2,in_port=2,icmp_type=131
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x2,in_port=2,icmp_type=132
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x2,in_port=2,icmp_type=135
actions=NORMAL
table=71, priority=95,icmp6,reg5=0x2,in_port=2,icmp_type=136
actions=NORMAL

# implement arp spoofing protection
table=71, priority=95,arp,reg5=0x1,in_port=1,dl_src=fa:16:3e:a4
:22:10,arp_spa=192.168.0.1 actions=NORMAL
table=71, priority=95,arp,reg5=0x1,in_port=1,dl_src=fa:16:3e:8c
:84:13,arp_spa=10.0.0.1 actions=NORMAL
table=71, priority=95,arp,reg5=0x2,in_port=2,dl_src=fa:16:3e
:24:57:c7,arp_spa=192.168.0.2 actions=NORMAL
table=71, priority=95,arp,reg5=0x2,in_port=2,dl_src=fa:16:3e:8c
:84:14,arp_spa=10.1.0.0/24 actions=NORMAL

# DHCP and DHCPv6 traffic is allowed to instance but DHCP servers
are blocked on instances.
table=71, priority=80,udp,reg5=0x1,in_port=1,tp_src=68,tp_dst=67
actions=resubmit(,73)
table=71, priority=80,udp6,reg5=0x1,in_port=1,tp_src=546,tp_dst
=547 actions=resubmit(,73)
table=71, priority=70,udp,reg5=0x1,in_port=1,tp_src=67,tp_dst=68
actions=drop
table=71, priority=70,udp6,reg5=0x1,in_port=1,tp_src=547,tp_dst
=546 actions=drop
table=71, priority=80,udp,reg5=0x2,in_port=2,tp_src=68,tp_dst=67
actions=resubmit(,73)
table=71, priority=80,udp6,reg5=0x2,in_port=2,tp_src=546,tp_dst
=547 actions=resubmit(,73)
table=71, priority=70,udp,reg5=0x2,in_port=2,tp_src=67,tp_dst=68
actions=drop
table=71, priority=70,udp6,reg5=0x2,in_port=2,tp_src=547,tp_dst
=546 actions=drop

# Flowing rules obtain conntrack information for valid ip and mac
address combinations. All other packets are dropped.
table=71, priority=65,ct_state=trk,ip,reg5=0x1,in_port=1,dl_src=
fa:16:3e:a4:22:10,nw_src=192.168.0.1 actions=ct(table=72,zone=
NXMLNX_REG6[0..15])

```

```

table=71, priority=65,ct_state=trk,ip,reg5=0x1,in_port=1,dl_src=
fa:16:3e:8c:84:13,nw_src=10.0.0.1 actions=ct(table=72,zone=
NXMNX.REG6[0..15])
table=71, priority=65,ct_state=trk,ip,reg5=0x2,in_port=2,dl_src=
fa:16:3e:24:57:c7,nw_src=192.168.0.2 actions=ct(table=72,zone=
NXMNX.REG6[0..15])
table=71, priority=65,ct_state=trk,ip,reg5=0x2,in_port=2,dl_src=
fa:16:3e:8c:84:14,nw_src=10.1.0.0/24 actions=ct(table=72,zone=
NXMNX.REG6[0..15])
table=71, priority=65,ct_state=trk,ipv6,reg5=0x1,in_port=1,dl_src=
fa:16:3e:a4:22:10,ipv6_src=fe80::f816:3eff:fea4:2210 actions=
ct(table=72,zone=NXMNX.REG6[0..15])
table=71, priority=65,ct_state=trk,ipv6,reg5=0x2,in_port=2,dl_src=
fa:16:3e:24:57:c7,ipv6_src=fe80::f816:3eff:fe24:57c7 actions=
ct(table=72,zone=NXMNX.REG6[0..15])
table=71, priority=10,ct_state=trk,reg5=0x1,in_port=1 actions=
drop
table=71, priority=10,ct_state=trk,reg5=0x2,in_port=2 actions=
drop
table=71, priority=0 actions=drop

```

The table 72 accepts only established or related connections, and implements rules defined by the security group. As this egress connection might also be an ingress connection for some other port, it has not switched yet but eventually processed by the ingress pipeline. All the established or new connections defined by security group rule are accepted, which will be explained later; all the invalid packets are dropped. It is important to notice that on some flows there is a `ct_mark=0x1`. Such value is related to flows that were marked as not existing anymore by an introduced later rule. Those are typically connections that were allowed by some security group rule and then the rule was removed.

```

# allow all icmp egress traffic
table=72, priority=70,ct_state=+est-rel-rpl,icmp,reg5=0x1, actions
=resubmit(,73)
table=72, priority=70,ct_state=+new-est,icmp,reg5=0x1, actions=
resubmit(,73)
table=72, priority=50,ct_state=+inv+trk actions=drop
table=72, priority=50,ct_mark=0x1,reg5=0x1 actions=drop
table=72, priority=50,ct_mark=0x1,reg5=0x2 actions=drop

# All other connections that are not marked and are established or
related are allowed.
table=72, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,
reg5=0x1 actions=NORMAL

```

```

table=72, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,
  reg5=0x2 actions=NORMAL
table=72, priority=50,ct_state=-new-est+rel-inv,ct_zone=644,
  ct_mark=0,reg5=0x1 actions=NORMAL
table=72, priority=50,ct_state=-new-est+rel-inv,ct_zone=644,
  ct_mark=0,reg5=0x2 actions=NORMAL

# marked established connections not matched in the previous flows
, they do not have accepting security group rule anymore
table=72, priority=40,ct_state=-est,reg5=0x1 actions=drop
table=72, priority=40,ct_state=+est,reg5=0x1 actions=ct(commit,
  zone=NXMLNX_REG6[0..15],exec(load:0x1->NXMLNX_CT_MARK[]))
table=72, priority=40,ct_state=-est,reg5=0x2 actions=drop
table=72, priority=40,ct_state=+est,reg5=0x2 actions=ct(commit,
  zone=NXMLNX_REG6[0..15],exec(load:0x1->NXMLNX_CT_MARK[]))
table=72, priority=0 actions=drop

```

In table 73 all detected ingress connections are sent to ingress pipeline. Since the connection was already accepted by egress pipeline, all remaining egress connections are sent to normal switching.

```

table=73, priority=100,reg6=0x284,d1_dst=fa:16:3e:a4:22:10 actions
  =load:0x1->NXMLNX_REG5[],resubmit(,81)
table=73, priority=100,reg6=0x284,d1_dst=fa:16:3e:8c:84:13 actions
  =load:0x1->NXMLNX_REG5[],resubmit(,81)
table=73, priority=100,reg6=0x284,d1_dst=fa:16:3e:24:57:c7 actions
  =load:0x2->NXMLNX_REG5[],resubmit(,81)
table=73, priority=100,reg6=0x284,d1_dst=fa:16:3e:8c:84:14 actions
  =load:0x2->NXMLNX_REG5[],resubmit(,81)
table=73, priority=90,ct_state=+new-est,reg5=0x1 actions=ct(commit
  ,zone=NXMLNX_REG6[0..15]),NORMAL
table=73, priority=90,ct_state=+new-est,reg5=0x2 actions=ct(commit
  ,zone=NXMLNX_REG6[0..15]),NORMAL
table=73, priority=80,reg5=0x1 actions=NORMAL
table=73, priority=80,reg5=0x2 actions=NORMAL
table=73, priority=0 actions=drop

```

The table 81 is similar to the table 71, allowing basic ingress traffic for obtaining IP address and ARP queries. It is important to notice that VLAN tag must be removed by adding `strip_vlan` to actions list, prior to injecting packet directly to port. All the not tracked packets are sent to obtain conntrack information.

```

table=81, priority=100,arp,reg5=0x1 actions=strip_vlan,output:1
table=81, priority=100,arp,reg5=0x2 actions=strip_vlan,output:2

```



```

table=81, priority=100,icmp6,reg5=0x1,icmp_type=130 actions=
strip_vlan,output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=131 actions=
strip_vlan,output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=132 actions=
strip_vlan,output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=135 actions=
strip_vlan,output:1
table=81, priority=100,icmp6,reg5=0x1,icmp_type=136 actions=
strip_vlan,output:1
table=81, priority=100,icmp6,reg5=0x2,icmp_type=130 actions=
strip_vlan,output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=131 actions=
strip_vlan,output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=132 actions=
strip_vlan,output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=135 actions=
strip_vlan,output:2
table=81, priority=100,icmp6,reg5=0x2,icmp_type=136 actions=
strip_vlan,output:2
table=81, priority=95,udp,reg5=0x1,tp_src=67,tp_dst=68 actions=
strip_vlan,output:1
table=81, priority=95,udp6,reg5=0x1,tp_src=547,tp_dst=546 actions=
strip_vlan,output:1
table=81, priority=95,udp,reg5=0x2,tp_src=67,tp_dst=68 actions=
strip_vlan,output:2
table=81, priority=95,udp6,reg5=0x2,tp_src=547,tp_dst=546 actions=
strip_vlan,output:2
table=81, priority=90,ct_state==trk,ip,reg5=0x1 actions=ct(table
=82,zone=NXMLNX_REG6[0..15])
table=81, priority=90,ct_state==trk,ipv6,reg5=0x1 actions=ct(table
=82,zone=NXMLNX_REG6[0..15])
table=81, priority=90,ct_state==trk,ip,reg5=0x2 actions=ct(table
=82,zone=NXMLNX_REG6[0..15])
table=81, priority=90,ct_state==trk,ipv6,reg5=0x2 actions=ct(table
=82,zone=NXMLNX_REG6[0..15])
table=81, priority=80,ct_state==+trk,reg5=0x1 actions=resubmit(,82)
table=81, priority=80,ct_state==+trk,reg5=0x2 actions=resubmit(,82)
table=81, priority=0 actions=drop

```

Similarly to table 72, table 82 accepts established and related connections. In this case all ICMP traffic coming from security group 1 (in this case only port 1) is allowed. The first two rules match on the IP packets, and the next two rules match on the ICMP protocol. These four rules define conjunction

flows.

```

table=82, priority=70,ct_state=+est-rel-rpl,ip,reg6=0x284,nw_src
=192.168.0.1 actions=conjunction(2147352552,1/2)
table=82, priority=70,ct_state=+est-rel-rpl,ip,reg6=0x284,nw_src
=10.0.0.1 actions=conjunction(2147352552,1/2)
table=82, priority=70,ct_state=+new-est,ip,reg6=0x284,nw_src
=192.168.0.1 actions=conjunction(2147352553,1/2)
table=82, priority=70,ct_state=+new-est,ip,reg6=0x284,nw_src
=10.0.0.1 actions=conjunction(2147352553,1/2)
table=82, priority=70,ct_state=+est-rel-rpl,icmp,reg5=0x2 actions=
conjunction(2147352552,2/2)
table=82, priority=70,ct_state=+new-est,icmp,reg5=0x2 actions=
conjunction(2147352553,2/2)
table=82, priority=70,conj_id=2147352552,ct_state=+est-rel-rpl,ip,
reg5=0x2 actions=strip_vlan,output:2
table=82, priority=70,conj_id=2147352553,ct_state=+new-est,ip,reg5
=0x2 actions=ct(commit,zone=NXMLNX_REG6[0..15]),strip_vlan,
output:2
table=82, priority=50,ct_state=+inv+trk actions=drop

# Same mechanism for dropping connections not allowed anymore as
that of table 72
table=82, priority=50,ct_mark=0x1,reg5=0x1 actions=drop
table=82, priority=50,ct_mark=0x1,reg5=0x2 actions=drop
table=82, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,
reg5=0x1 actions=strip_vlan,output:1
table=82, priority=50,ct_state=+est-rel+rpl,ct_zone=644,ct_mark=0,
reg5=0x2 actions=strip_vlan,output:2
table=82, priority=50,ct_state=new-est+rel-inv,ct_zone=644,
ct_mark=0,reg5=0x1 actions=strip_vlan,output:1
table=82, priority=50,ct_state=new-est+rel-inv,ct_zone=644,
ct_mark=0,reg5=0x2 actions=strip_vlan,output:2
table=82, priority=40,ct_state=-est,reg5=0x1 actions=drop
table=82, priority=40,ct_state=+est,reg5=0x1 actions=ct(commit,
zone=NXMLNX_REG6[0..15],exec(load:0x1->NXMLNX_CT_MARK[]))
table=82, priority=40,ct_state=-est,reg5=0x2 actions=drop
table=82, priority=40,ct_state=+est,reg5=0x2 actions=ct(commit,
zone=NXMLNX_REG6[0..15],exec(load:0x1->NXMLNX_CT_MARK[]))
table=82, priority=0 actions=drop

```

Note: Contrack zones on a single node are now based on network to which port is plugged in. That makes a difference between traffic on hypervisor only and east-west traffic. For example, if port has a virtual IP (vIP) that was migrated to a port on different node, then new port will not contain contrack

information about previous traffic that happened with vIP.

### 4.5.2 Open vSwitch “learn action” firewall

As long as iptables is not a perfect complement for deployments using OvS, requiring a lot of coding to perform iptables meshing with Open vSwitch, there is the need for something OvS-based. OvS already provides its own methods for implementing internal rules (using the OpenFlow protocol) and this firewall is thus based on the OpenFlow learn action, therefore based entirely on OVS rules. Indeed, this firewall creates a pure OvS model that is not dependent on functionality from the underlying platform. It uses the same public API to talk to the Neutron agent as the existing Linux Bridge firewall implementation [69].

Another OvS firewall solution is already present (Open vSwitch native firewall driver, as previously shown) and it is based on the use of the “conntrack” module from Linux. This module provides a way to implement a “stateful firewall” through tracking of connection statuses. It is expected that using conntrack will minimize the need to bring the traffic packets up to user space to be processed and should therefore yield higher performance. This firewall instead, on the other hand, is based on the concept of “learn action”. These learn actions track the traffic in one direction and set up a new flow to allow the same traffic flow in the reverse direction. This implementation is fully based on the OpenFlow standard.

When a Security Group rule is added, a “manual” OpenFlow rule is added to the OVS configuration. This new rule allows, for example, ingress TCP traffic for a specific port. When a packet matches this rule, the “manual” rule allows the packet to be delivered to its destination. However, and this is a substantial aspect of the new firewall, a new “automatic” rule is to send reverse traffic replies back to the source. Although this design could have an adverse effect on performance, due to the fact that using learn actions forces the processing of all packets in user space, the benchmark results from Intel [69] show that this design performance is better than iptables. Even more significantly, this firewall allows the usage of the DPDK [70] features of OVS, yielding performance that is more than four times higher than the performance of non-DPDK OVS without firewall.

Traffic flows can be grouped into traffic between two internal virtual machines (east-west traffic) or traffic between an internal machine and an exter-

nal host (north-south traffic). As the previously shown solutions, the Security Group rules only apply to machines controlled by Neutron and included in one or several security groups, which means that only the virtual machines inside a compute host will be affected by these rules. The firewall will only manage the rules of the br-int bridge. Moreover, the firewall rules applied in the integration bridge begin to process traffic as soon as a packet arrives on this bridge.

Table 0, known as the input table, is the default table: all traffic injected inside br-int is processed by this table. The ARP packets are processed with the highest priority. Each machine inside a VLAN must be able to populate its address among the other VLAN machines. The packets who are not ARP are not matched by table 0 rules, thus are sent to the selection table, the Table 1.

Indeed, the selection table ( $\Rightarrow$  table 1), checks if every packet is from or to a virtual machine. The rules which are added ensure that only those packets who are matching the stored port MAC address, VLAN tag and port number are allowed to pass. If a packet is a DHCP packet, the IP must be 0.0.0.0. Traffic not matching this rule is dropped.

The ingress table, which is table 2, has three kinds of rules, plus the fallback one.

- “Learn” input rules, which are those created automatically when an output rule is matched. As stated previously, creation of this output rule also invokes the creation of a reverse input rule.
- Services input rules, which are always added by the firewall to allow certain ICMP traffic and DHCP messages.
- Manual input rules, which are added by the user in the Security Group.
- All the traffic that does not match any of the former three is dropped.

Similarly, the egress table ( $\Rightarrow$  table 3) has three kinds of rules, plus the default one.

- “Learn” input rules are created automatically when an output rule is matched. As stated previously, creation of this output rule also invokes the creation of a reverse input rule.

- Services output rules are always added by the firewall to allow certain ICMP traffic and DHCP messages.
- Manual output rules are added by the user in the Security Group.
- If the traffic does not match one of the previous rules, it is sent to the egress external traffic.

Finally, the egress external traffic table processes the north-south traffic. If the traffic needs to leave the integration bridge, then it reaches this table: only external egress traffic must be managed by this table. A final check is then made: if any traffic in this table is to be sent to a virtual machine, then this traffic is dropped. The traffic not filtered by these rules is sent using the “normal” action. The packets are sent by OVS using the built-in ARP table.

By recalling [69], it seems that this approach in conjunction with the Intel DPDK OvS is the best one in terms of performance. However, this solution lacks of generality as long as it can be applied only on servers that can use the above cited library.

### 4.5.3 Open vSwitch native interface driver

Prior to OpenStack Newton, the neutron-openvswitch-agent used “ovs-ofctl” of\_interface driver by default to communicate with the Open vSwitch. From Newton on instead, the default implementation for of\_interface has become the novel “native”, that mostly eliminates spawning ovs-ofctl to slightly improve the networking performance. This is an alternative OpenFlow implementation, implemented using *Ryu SDN controller* ofproto python library from Ryu SDN Framework. This solution indeed uses Ryu to inject OpenFlow rules when requested (e.g. a new virtual instance is being created or an old one is deleted) instead of building a “ovs-ofctl add/del-flow”.

The consequences are:

- The implemented OpenFlow rules are switched to OpenFlow 1.3, rather than OpenFlow 1.0;
- **The OvS-agent acts as an OpenFlow controller perfectly integrated with the OpenStack platform;**
- The OvS of the node is configured to connect to the controller.

Among the benefits, it is also possible to state:

- Reduction of the overhead related to the invocation of the `ovs-ofctl` command (and its associated rootwrap);
- Easier dealing with a future use of OpenFlow asynchronous messages (e.g. Packet-In, Port-Status, etc.)  $\Rightarrow$  Introduction of a SDN controller inside the OpenStack platform, integration among the platform and the framework  $\Rightarrow$  Possibility to directly manage the OpenStack networking by managing Ryu;
- Simpler XenAPI integration.

More details about this approach will be presented in Chapter 7.

# Chapter 5

## SDN controllers overview

As already seen in Chapter 2 and by recalling Figure 2.4, SDN controllers are the entities managing the network, as a result of the decoupling between the control plane and the data plane. Indeed, it is possible to state that the SDN controllers are actually the “intelligence” of the network. An SDN Controller platform typically contains a collection of pluggable modules that can perform different network tasks (e.g. reactive L2 forwarding, traffic redirection, etc.) and that can be added or removed in a more or less easy way.

It is vital to underline that, in general, SDN controllers are provided to the user as ready-to-use and the user can therefore write applications that use their API to manage the network as he or she prefers. Indeed, the application instructs the SDN controller about how the network should behave, by leaving the low-level implementation details to the controller itself (e.g. contact the switch, install the flow, etc.).

On the other hand, the user might also want to extend or create ex-novo the core of the SDN controller (for instance, by creating a new module) to improve the controller intelligence.

### 5.1 Ryu framework

Ryu [71] is a *component-based* open source (freely available under Apache 2.0 license) SDN framework, entirely written in Python language. Ryu is the Japanese word for “flow” as well as “Dragon God (that rules the water world, the flows)”.

It provides software components with well defined APIs that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions.

With respect to other SDN controller solutions, Ryu is very easy to setup and provides a clean and transparent experience with OpenFlow. One of the plus points is the use of a high-level programming language like Python. It is possible to state that Ryu is stable, as it is not intended to be always cutting-edge in terms of module and features, even though it is always up to date with OpenFlow support.

A Ryu application is just a Python script: in order to make it runnable, it is important to import the Ryu `app_manager`, which is the central manager of Ryu applications. It is in charge of load Ryu applications, provide them the contexts and route messages among them. Indeed, when writing a Ryu application, a new subclass of `RyuApp` to run the Python script as a Ryu application is needed. A simple example is shown below, but more complex ones are easy to be found in the Ryu Documentation [72] [73].

```
from ryu.base import app_manager

class L2Switch(app_manager.RyuApp):
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
```

Many other components are present in Ryu. One of the most important is the `ryu.controller.controller`, which is the entity that handles the connections coming from the switches and then generates and routes events to Ryu applications. Always regarding the OpenFlow controller, Ryu provides `ryu.controller.dpset` to manage the switches, `ryu.controller.ofp_event` to deal with event definitions and `ryu.controller.ofp_handler` for handling the OF protocol.

### 5.1.1 OpenStack networking with Ryu

In the past, Ryu used to integrate to OpenStack Neutron via the OFAgent [75]. This agent was a neutron core-plugin, implemented as ML2 mechanism driver that used Ryu ofproto library. It has been removed from OpenStack since Mitaka, in favor of the OpenVSwitch mechanism driver using the “native”



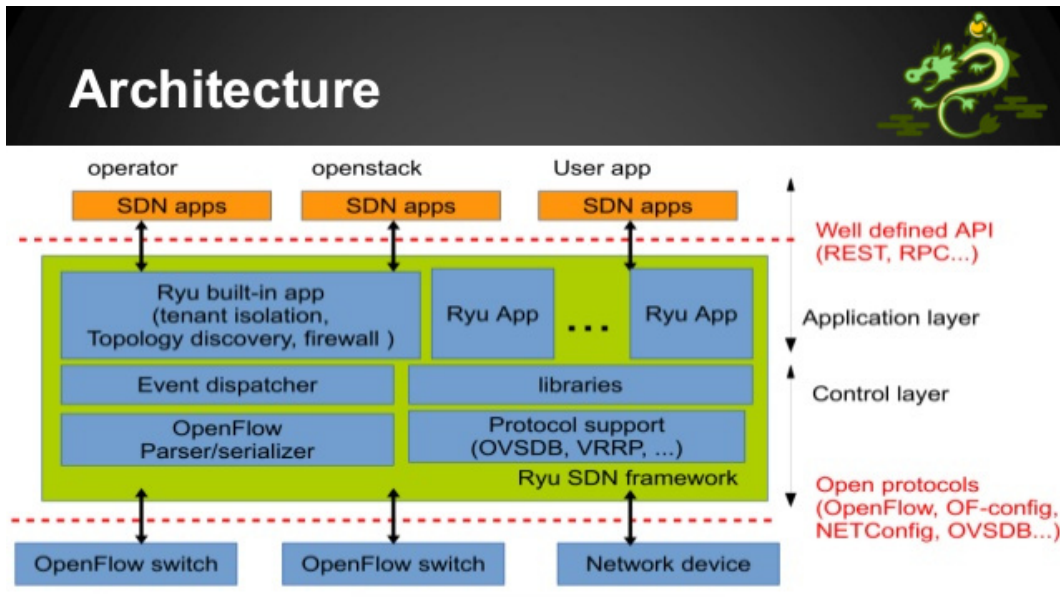


Figure 5.1: Ryu architecture [74]

of\_interface, as shown in Chapter 3.

Ryu is now therefore integrated inside OpenStack whenever the node (compute or network) is using an OvS agent with the “native” setting for the OpenFlow interface. From Newton on, by default Neutron uses the native interface of OVSDB and OpenFlow. The use of this interface allows Neutron to call `ryu.base.app_manager` during operation, and by default the native interface will have the Ryu controller listen on `127.0.0.1:6633` which can be however modified to point to another address.

The integration among Ryu and OpenStack is very basic: however, it is possible to load applications through the `app_manager` that is loaded by Neutron: therefore, the possible scenarios that can take place are infinite, because it simply needs the administrator to run the SDN application that he or she wants in order to manage the cloud platform in addition to or in a different way with respect to the default Neutron mechanisms.

Moreover, it is important to remark that the OpenStack community has chosen Ryu as its default solution to manage the OvSs in Neutron. Therefore, it is possible to say that this solution is officially supported by OpenStack itself, without the risk to have a non-compliant Ryu version with respect to

the Neutron one.

## 5.2 OpenDaylight

The OpenDaylight Project (ODL) [76] is an open source SDN project hosted by the Linux Foundation. Its aim is to offer a software-defined networking controller to an industrial environment. Indeed, the OpenDaylight controller (renamed then in OpenDaylight platform) is a community-led and industry-supported framework for the SDN, to which many IT companies (HP, Cisco, Huawei, etc.) as well as Telcos are contributing. The OpenDaylight Controller is kept within its own Java Virtual Machine (JVM) and provides a CLI and a GUI.

The OpenDaylight Controller is able to deploy in a variety of production network environments. It can support a modular controller framework, where it is possible to add or remove functionalities at runtime through the use of Apache Karaf [77]. It exposes open northbound APIs, which can be used by applications. These applications are the “users” of the Controller to collect information about the network, run some algorithms to conduct analytics, and finally to make use of the OpenDaylight Controller to create new rules to be installed in the network according to some policy.

Finally, ODL can take advantage of replication to improve its resiliency and therefore run in a cluster of three or more instances that will be coordinated and eventually consistent [78].

### 5.2.1 OpenStack networking with ODL

Before 2016, OpenDaylight provided a very basic ML2 mechanism to be run within the Neutron server. However, it was not suitable at all to be deployed in a production environment, being just a PoC of the potentialities the controller could achieve. Therefore, a second version of this integration has been completely thought from scratch, but with the “constraint” to take advantage of the already present ODL modules (for instance, the VTN one, to manage Virtual Tenant Networks of every type).

By now, the component that is installed at the OpenStack side is the OpenStack *networking-odl*, a library of drivers and plug-ins that integrates OpenStack Neutron API with OpenDaylight Backend. It provides the ML2 driver

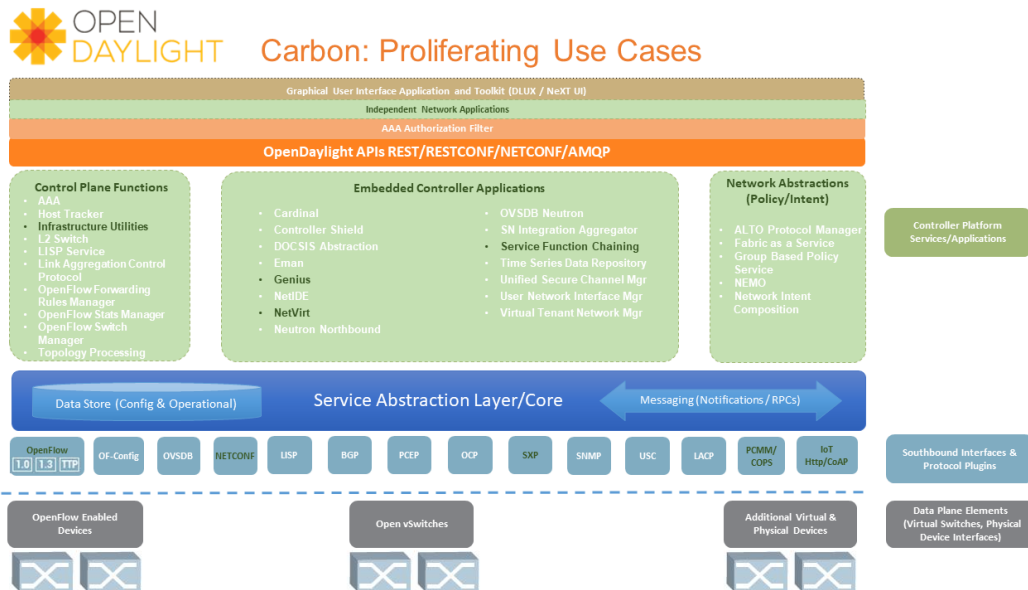


Figure 5.2: OpenDaylight Carbon modules [76]

and the L3 plug-in to enable the communication between OpenStack Neutron L2 as well as L3 resources API and the OpenDayLight Backend. However, the installation process of the whole production environment does not come simple as Ryu with Neutron packaging. Moreover, OpenStack networking-odl needs to be installed separately, either via Pip or via source.

ODL can however be installed in three different scenarios, depending on the functionalities that this SDN controller should add to the OpenStack environment in which it is intended to run. Each scenario adds some functionality that might be interesting for some companies (indeed, in ODL most components are developed by a single company that is interested in it).

- **Group Based Policy:** ODL GBP allows users to express network configuration in a declarative rather than imperative way. It is often described as asking to ODL for what you want, rather than how you can do it, Group Based Policy achieves this by implementing an Intent System. The Intent System is a process around an intent driven data model and contains no domain specifics but is capable of addressing multiple semantic definitions of intent.
- **OVSDB:** ODL OVSDB allows users to take advantage of Network Vir-

tualization using OpenDaylight SDN capabilities whilst utilizing OpenvSwitch. The stack includes a Neutron Northbound, a Network Virtualization layer, an OVSDB southbound plug-in, and an OpenFlow southbound plug-in.

- **Virtual Tenant Network:** ODL VTN is an application that provides multi-tenant virtual network on an SDN controller. VTN Manager is implemented as a plug-in to the OpenDaylight controller and provides a REST interface to create/update/delete VTN components whilst providing an implementation of Openstack L2 Network Functions API.

Therefore, `networking-odl` will act as a proxy by redirecting each request that arrives to the Neutron server to the ODL instance, where some of the above cited modules have been deployed to act as the OpenStack network intelligence. ODL will then perform each step needed for the networking (security groups management, network creation, etc.) [79].

### 5.3 Open Networking Operating System: ONOS

ONOS (Open Networking Operating System) is a widely used emerging SDN controller provided by ONLab (now Open Networking Foundation) and backed by a remarkable range of networking actors, as Cisco, Huawei, Google, Intel, Nokia, AT&T, China Unicom, etc. ONOS mission is *“to produce the Open Source Network Operating System that will enable service providers to build real Software Defined Networks”* [80].

ONOS is able to provide the control plane for a software-defined network, managing network components, such as switches and links, and running software to provide communication services to end hosts and neighboring networks [81]. It has a modular implementation: its core is composed by many modules that provide network functionalities. It is possible for the developer to deploy modules to extend its core and for the client to develop applications that are going to use the Northbound APIs that ONOS exposes.

The ONOS kernel and core services, as well as ONOS applications, are written in Java as bundles that are loaded into the *Karaf OSGi* container. OSGi is a component system for Java that allows modules to be installed and run dynamically in a single Java VM (JVM). The bundles can be activated and deactivated at runtime, through the use of Apache Karaf.

With respect to the southbound instead, ONOS provides various adapters, e.g. OpenFlow, NetConf, P4...

ONOS works in a distributed way: it is deployed in multiple copies following an *optimistic replication technique* completed by a background gossip-based protocol to ensure (eventual) consistency.

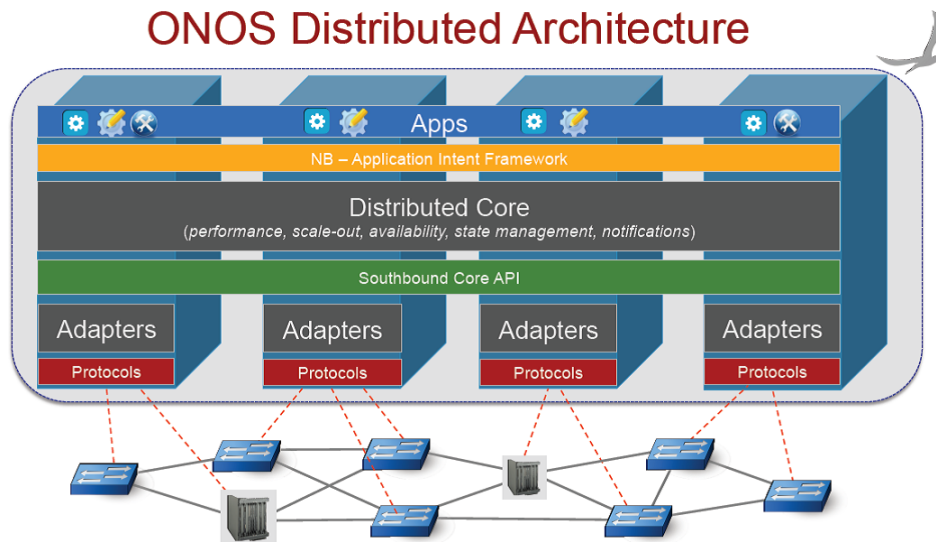


Figure 5.3: ONOS architecture

Recently, the Open Networking Foundation announced its merging with OnLab and therefore with this project [82]: this means that the entity that provides the guidelines of the SDN approach is backing a single SDN controller and this controller is ONOS. This will probably lead to have ONOS as the standard de-facto for SDN.

A chart showing the ONOS internal architecture, along with some examples of external agents towards both the Application plane and the Data plane are shown in Figure 5.4. In this Figure, there is also the UniBo contribution to extend the REST API, related to the works presented in Chapter 2.

### 5.3.1 OpenStack networking with ONOS: SONA

ONOS provides SONA, a series of modules that are able to integrate the SDN controller itself with the cloud platform. SONA is composed by a set of

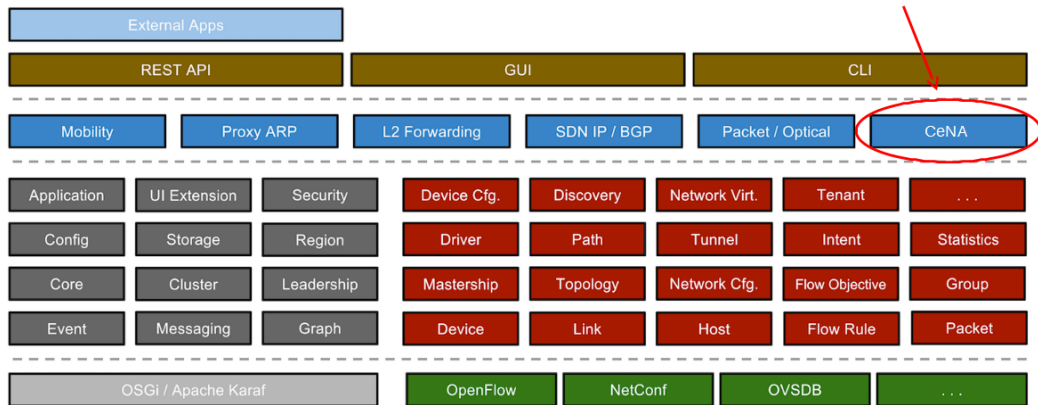


Figure 5.4: ONOS modules

modules thought for replacing *completely* Neutron. Each submodule will deal with a specific task: security groups, switching, routing, etc.

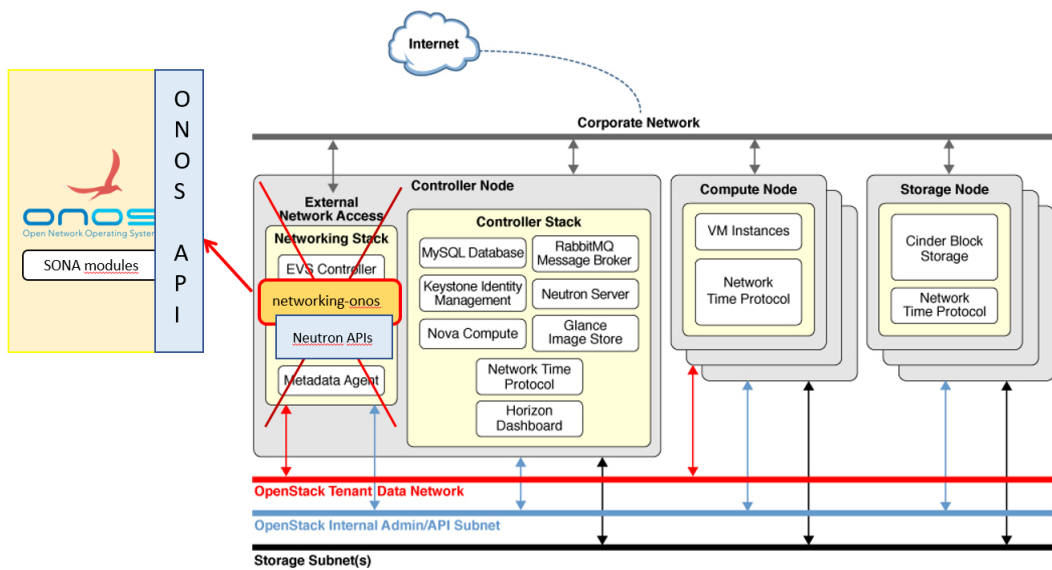


Figure 5.5: Example of implementation of ONOS in OpenStack

SONA aims are to provide:

- **Optimized and distributed logical switching:** SONA implements

multicast free VXLAN implementation where each OvS in the compute node acts as a part of a global big switch;

- **Optimized and distributed logical routing:** each OvS in the compute node takes care of all East-West routed traffic for its VMs;
- **Broadcast free experience:** SONA proxies ARP and DHCP request from a virtual instance;
- **Agent-less deployment:** there are no Neutron agents running on compute node or gateway node, everything is taken into account by SONA;
- **Scalable deployment:** SONA provides horizontal scalability of gateway nodes to connect virtual software-defined networks to the outside of the world.

The Neutron server is replaced by **networking-onos**: this is a simple entity exposing the Neutron REST APIs. The REST calls to Neutron will be therefore sent to this component, translated into the ONOS (SONA) REST APIs syntax and sent to the controller itself. In this way, OpenStack is transparent to the implementation of Neutron that ONOS provides as long all the requests are taken into account by the SDN controller. Being this controller open, any cloud administrator could extend even more the provided integration with the cloud platform by means of SDN and take advantage out of it: this was much more difficult by simply modifying the legacy OpenStack sources, apart from using the native OpenFlow interface with the Ryu controller.

SONA is composed of three ONOS applications: *openstackNode*, *openstackNetworking*, and *vRouter*. Currently, *openstackNode* and *openstackNetworking* run on a central instance of ONOS, whilst *vRouter* runs on a separate (lighter) ONOS instance at each gateway node. This is a limitation of the *vRouter* component, which does not support multiple switches. However, it is planned that all the applications could run on a single ONOS cluster whenever the developers will be able to improve the *vRouter*.

ONOS will usually provide the networking functions through SONA in a separated node. This is recommended to have a good experience: indeed, ONOS is Java-based and the hardware requirements are sufficiently high to suggest to reserve some dedicated hardware, although a generic machine is still fine with it. The *OpenStackNode* application is in charge of all the management and bootstrap of the nodes where the *OpenVSwitch* will run, therefore

computes and gateways. Its roles include all the OpenVSwitch life cycle management as well as the VxLAN support and configuration for the data network. It is possible to say that it will perform part of the actions that were delegated to the neutron-openvswitch-agent, as adding VXLAN ports to the integration bridge.

With respect to **OpenStackNetworking** module instead, this will include the needed proxies for ARP and DHCP to reply to the hosts (even by using pre-defined fake MAC address, if needed), as well as the ML2 driver and the L3 plugin backend. It will provide all the flows management to provide instances' connectivity by installing OpenFlow rules into the OVS; it also exposes the REST API that is called by *networking-onos*. Thus, whenever there is a Neutron request, this is post-committed to openstackNetworking via the networking-onos driver. Finally, through **OpenStackNetworking** all kind of East-West traffic is handled at compute nodes and only North-South traffic is forwarded to gateway nodes and NATed there to public IP before leaving the virtual world.

Finally, to provide connectivity between Software Defined Networks and legacy external IP networks, an instance of ONOS called **vRouter** is deployed in every gateway node. As a gateway node does not keep any states locally, packets for a same session does not have to be handled in a same node. This allows this horizontal scalable deployment to remove the single point of failure, to offload the traffic and to let the gateways to be independent among each other. The vRouter will use SDN-IP techniques, involving one or more quagga instances: these are providing the BGP functionalities, to let the internal SD networks to be visible from the outside. This is something that OpenStack does not provide by default. In particular, 1:1 NAT rule for floating IP is installed proactively to all gateway nodes when a floating IP is associated to a fixed IP. 1:n NAT rules are instead installed reactively when a gateway node receives unknown external packets. Gateway nodes are realized by OpenFlow select group at each compute node and all outbound packets are sent to this gateway node group. Whenever there is an outage of one gateway node, the node is automatically excluded from the group. Each gateway node establishes a peering with external router and the external router enables multi-path so that inbound traffic is distributed as well.



### 5.3.2 Security groups in SONA

As already stated, the SONA approach tends to softwarize even more all the needed networking functionalities. One of the most important ones is the implementation of security groups, a named collection of network access rules that are used to limit the types of traffic that have access to instances, i.e. the firewalling of the instances. It is interesting to see how they work in this context and how to properly integrate them within OpenStack.

#### SONA Pipeline

The security groups in SONA are mapped into OpenFlow rules written into the OVS: these rules are managed by OpenStackNetworking.

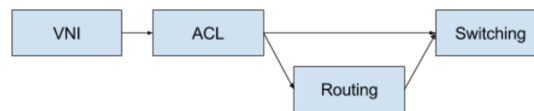


Figure 5.6: High level table design

These are the actual tables:

- **VNI (table=0)**: forwards ARP and DHCP to controller and tags the source Virtual Network Instance to packets based on the in-port of a packet.
- **ACL (table=1)**: it is actually composed by two tables:
  - **Security group**: forwards only allowed packets to the next table based on security group configurations;
  - **Connection tracking**: maintains the state of the established connections.
- **Jump (table=2)**: forwards a routing packet, identified by the destination MAC address of the gateway node, to the routing table, and a switching packet to the switching table.
- **Routing (table=3)**: forwards East-West routing packets to the switching table and North-South packets to a gateway group.

- **Switching (table=4)**: forwards packets to final destination; if the destination is not in the local machine, the packet is sent it to the vxlan port; if the destination is a subnet gateway, it is sent to the gateway node group.
- **Group**: maintains information on the gateway node group.

Examples of security groups tables:

```
# VNI Table (table=0)
priority=40000,arp actions=CONTROLLER:65535,clear_actions
priority=40000,udp,tp_src=68,tp_dst=67 actions=CONTROLLER:65535,
clear_actions
priority=30000,ip,in_port=7 actions=set_field:0x1e->tun_id,
goto_table:1
priority=0 actions=goto_table:1

# ACL Table (table=1)
priority=30000,ip,nw_src=192.168.100.3,nw_dst=192.168.101.3
actions=goto_table:2
priority=30000,ip,nw_src=192.168.101.3,nw_dst=192.168.100.3
actions=goto_table:2
priority=0 actions=drop

# Jump Table (table=2)
priority=30000,dl_dst=fe:00:00:00:00:02 actions=goto_table:3
priority=0 actions=goto_table:4

# Routing Table (table=3)
priority=28000,ip,tun_id=0x1e,nw_src=192.168.100.0/24,nw_dst
=192.168.100.0/24 actions=set_field:0x1e->tun_id,goto_table:4
priority=28000,ip,tun_id=0x22,nw_src=192.168.101.0/24,nw_dst
=192.168.101.0/24 actions=set_field:0x22->tun_id,goto_table:4
priority=28000,ip,tun_id=0x1e,nw_src=192.168.100.0/24,nw_dst
=192.168.101.0/24 actions=set_field:0x22->tun_id,goto_table:4
priority=28000,ip,tun_id=0x22,nw_src=192.168.100.0/24,nw_dst
=192.168.101.0/24 actions=set_field:0x22->tun_id,goto_table:4
priority=28000,ip,tun_id=0x22,nw_src=192.168.101.0/24,nw_dst
=192.168.100.0/24 actions=set_field:0x1e->tun_id,goto_table:4
priority=28000,ip,tun_id=0x1e,nw_src=192.168.101.0/24,nw_dst
=192.168.100.0/24 actions=set_field:0x1e->tun_id,goto_table:4
priority=25000,ip,tun_id=0x22,dl_dst=fe:00:00:00:00:02,nw_src
=192.168.101.0/24 actions=group:2901605683
priority=25000,ip,tun_id=0x1e,dl_dst=fe:00:00:00:00:02,nw_src
=192.168.100.0/24 actions=group:2901605683
```

```
# Switching Table (table=4)
priority=30000,ip,tun_id=0x1e,nw_dst=192.168.100.3 actions=
  set_field:fa:16:3e:2a:85:30->eth_dst,output:3
priority=30000,ip,tun_id=0x22,nw_dst=192.168.101.3 actions=
  set_field:10.1.1.163->tun_dst,output:1
priority=30000,ip,tun_id=0x1e,nw_dst=192.168.100.1 actions=group
:2901605683
priority=30000,ip,tun_id=0x22,nw_dst=192.168.101.1 actions=group
:2901605683

# Group table
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
  group_id=2901605683,type=select,bucket=actions=set_field
  :10.1.1.165->tun_dst,output:1
```

By now, the current testing versions are not using some of these tables: therefore, there might be some issues for OpenStack to be able to deal with the security groups of ONOS unless using a stable version of ONOS for it.



# Chapter 6

## SONA implementation and tests

This testbed has been deployed over a single server running CentOS 7: all the nodes shown in Figure 6.1 are virtualized. This could have been achieved also by using other Linux server distros like Ubuntu server, etc. where the virtualization tools are installed and configured. In this case, KVM has been used as the hypervisor in addition to nested KVM virtualization: this of course will result in a bottleneck in the computing performance whenever it will come to the OpenStack instances. However, being this a testbed implemented to understand SONA and its integration with OpenStack, the performance evaluation is not the focus point. The version of ONOS here analyzed is 1.8 (Ibis) and the version of OpenStack is Mitaka, to have a stable environment.

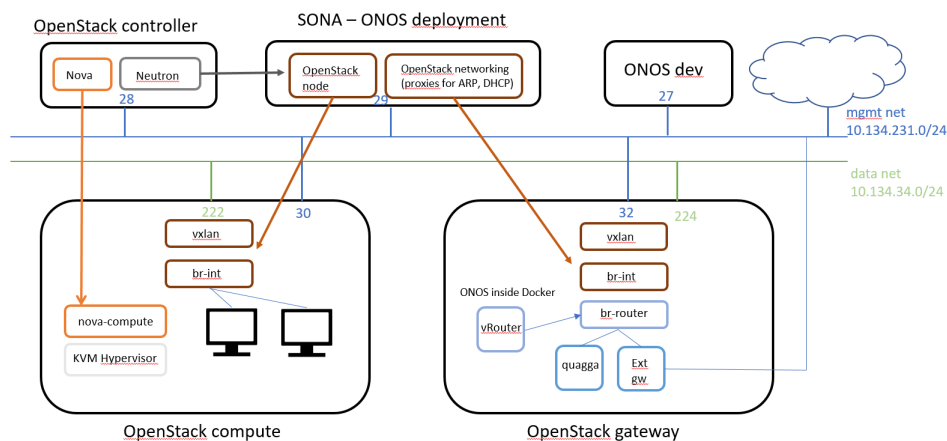


Figure 6.1: Deployed SONA-OpenStack topology

It would be possible to scale horizontally the topology by having multiple compute nodes and/or gateway nodes, to overcome the single point of failure. In addition, ONOS itself provides HA by default when there are multiple instances in the cluster: it could be then possible to add a proxy server beyond an ONOS cluster (e.g. HAproxy), and let Neutron use it as a single access point of the cluster.

## 6.1 ONOS deployment

In order to deploy ONOS and in particular SONA one development node and one deployment node are needed. It is however possible to use just one node with some tricks: here the choice went on having the two machines as there was the need to manage also other ONOS projects in parallel, thus by sharing the development node. Both the nodes are Ubuntu 14.04 machines: Karaf and Maven have been installed and configured as well as Java 8. It is then possible to install IntelliJ as an IDE on the development node as it is already configured to deal with the ONOS code.

The onos project has been then cloned with git: it comes directly with the SONA modules, that might be enabled during the deployment phase by need.

```
git clone https://gerrit.onosproject.org/onos
```

SSH is configured between the two nodes and the “*sdn*” user is created with superuser privileges and no password. Indeed, the deployment will be performed through the use of an SSH session: the login will be indeed related to the *sdn* user. To perform the deployment, some steps are needed in the development node:

- create the network configuration *onos/tools/package/config/network-cfg.json*, where important parameters of the OpenStack deployment are inserted (refer to Appendix A.1.1);
- create the cell file *onos/tools/test/cells/sona*, filled with data related to the deployment, as modules to deploy, location of the deployment node, user, etc.;
- load the cell file (Appendix A.1.1), compile ONOS through Buck (it is also possible to compile by using Maven even though it is deprecated),

create a package and deploy it; by entering into ONOS through Karaf, it is possible to see the deployed modules (Figure 6.2).

```
cell sona
onos-buck build onos
onos-package
stc setup
```

```
onos> apps -s -a
* 4 org.onosproject.optical-model 1.8.0.SNAPSHOT Optical information model
* 6 org.onosproject.ovsdb-base 1.8.0.SNAPSHOT OVSDB Provider
* 13 org.onosproject.openstackinterface 1.8.0.SNAPSHOT OpenStack Interface App
* 16 org.onosproject.drivers.ovsdb 1.8.0.SNAPSHOT OVSDB Device Drivers
* 17 org.onosproject.openstacknode 1.8.0.SNAPSHOT OpenStack Node Bootstrap App
* 18 org.onosproject.scalablegateway 1.8.0.SNAPSHOT Scalable GW App
* 19 org.onosproject.openstackrouting 1.8.0.SNAPSHOT OpenStack Routing App
* 20 org.onosproject.dhcp 1.8.0.SNAPSHOT DHCP Server App
* 21 org.onosproject.openstackswitching 1.8.0.SNAPSHOT OpenStack Switching App
* 30 org.onosproject.openflow-base 1.8.0.SNAPSHOT OpenFlow Provider
* 43 org.onosproject.drivers 1.8.0.SNAPSHOT Default device drivers
```

Figure 6.2: ONOS active modules

## 6.2 OpenStack development infrastructure deployment

As long as there are no longer neutron-openvswitch agents running in compute and gateway nodes, a prerequisite is to have in all these nodes an OvS already up and running. To do so, I preferred to add the OpenStack repository related to Mitaka, perform an update and then install all the *openvswitch* packages with their dependencies. In addition to it, the OVSDB has to be put in listening mode: a restart of the service is required.

```
Configuration 1: /usr/share/openvswitch/scripts/ovs-ctl

...
set ovsdb-server '$DB_FILE'
set '$@' --remote=ptcp:6640
...
```

The first step is to install the *networking-onos* component in the controller node. After the fulfillment of the dependencies, the component is cloned and

python is launched to perform the setup. It has then to be configured through its file `/opt/stack/networking-onos/etc/conf-onos.ini`, as shown in Appendix A.1.3.

```
sudo wget https://bootstrap.pypa.io/ez_setup.py -O - | python
git clone https://github.com/openstack/networking-onos.git
cd networking-onos/
sudo python setup.py install
```

Then it is needed to install OpenStack: in this case, for the sake of simplicity, DevStack has been used to deploy the cluster. This of course goes in the direction of having something less stable with respect to a manual package installation, but allows a faster and easier implementation of the tools. Therefore, it is completely worth the cost in this case where the focus is on the evaluation of the integration and some related case studies. Of course, if performance were considered, a stable production environment would have been requested. The definition of the *local.conf* configuration file of the controller node is presented in Appendix A.1.4.

```
git clone -b stable/mitaka https://git.openstack.org/openstack-dev
/devstack
sudo devstack/tools/create-stack-user.sh; su stack
... # Generate the local.conf file inside devstack/
./stack.sh
```

After having run the *stack.sh* script, the controller will be up: however, the installation is far from being complete.

Similarly, in the compute node the clone of the Mitaka branch of DevStack will be performed and the *local.conf* defined. Finally, the stack operation will provide the computing functionalities in the compute node.

### 6.3 L3 integration

Although the ML2 compatible module is already able to be deployed, there is the need to configure also the L3 functionalities. As shown, SONA modules will also be present in the Gateway node, with the presence of a vRouter entity, which might be nothing more than a Docker instance of ONOS running with the L3 functionalities (called *onos-router* from the OpenStack perspective).

Moreover, there will be the need for a Docker quagga instance implementing the BGP functionalities as the ONOS vRouter uses this protocol to let

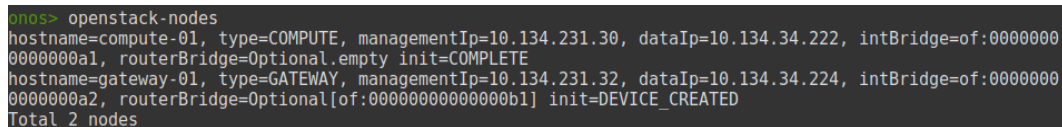


the software defined networks talk with the outside legacy world. Finally, in absence of physical external gateway, another Docker container will reproduce its functionalities.

### 6.3.1 Gateway node, vRouter and BGP

From the ONOS deployment machine, a POST REST call has to be performed: this call takes as a parameter the network configuration defined before and will try to initialise the nodes. It will be possible to see, as in Figure 6.3, that the Compute node is already ok for SONA, whereas the gateway node is seen to have been just created.

```
curl --user onos:rocks -X POST -H 'Content-Type: application/json'
  http://10.134.231.29:8181/onos/v1/network/configuration/ -d
  @network-cfg.json
```



```
onos> openstack-nodes
hostname=compute-01, type=COMPUTE, managementIp=10.134.231.30, dataIp=10.134.34.222, intBridge=of:00000000
0000000a1, routerBridge=Optional.empty init=COMPLETE
hostname=gateway-01, type=GATEWAY, managementIp=10.134.231.32, dataIp=10.134.34.224, intBridge=of:00000000
0000000a2, routerBridge=Optional[of:0000000000000b1] init=DEVICE_CREATED
Total 2 nodes
```

Figure 6.3: Intermediate configuration phase

It is therefore important to implement the other needed entities in order to implement the L3 functionalities: the vRouter, the quagga instance. In addition to them, as no external router is present in the physical topology, an external-router properly configured to deal with NAT and BGP is present. After installing Docker simply via the use of a bash script, it is possible to download a repository that already contains some useful scripts for the setup of the vRouter. A JSON configuration file has to be provided in order to be compliant with the cluster and it is shown in Appendix A.1.5.

```
wget -qO- https://get.docker.com/ | sudo sh
git clone https://github.com/hyunsun/sona-setup.git
```

This will therefore create the Docker container with an instance of ONOS inside, implementing the vRouter functionalities.

```
./vrouter.sh
```

It is then needed to modify the zebra configuration placed in *volumes/gateway/zebra.conf* and the bgpd one in *volumes/gateway/bgpd.conf* (Appendix

A.1.6): in this way a simple BGP configuration is created and the script related to quagga can be started.

```
./quagga.sh --name=gateway -01 --ip=172.18.0.254/24 --mac=fe
:00:00:00:00:01
```

In a similar way, *volumes/router/bgpd.conf* and *volumes/router/zebra.conf* have to be modified (Appendix A.1.6) and then it is possible to start the quagga script again, but related to the external-router.

```
./quagga.sh --name=router -01 --ip=172.18.0.1/24
--mac=fa:00:00:00:00:01 --external-router
```

It is then important to perform an SSH to the vRouter, through Karaf: after a quick check on the presence of the correct ports related to the interfaces of the previously deployed components (Figure 6.4), it is necessary to add the static routes in order to have connectivity to the outside as in Figure 6.5.

```
onos> ports
id=of:00000000000000b1, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2
.5.0, serial=None, driver=softrouter, channelId=172.17.0.1:47570, managementAddress=172.17.0.1, name=of:0
00000000000000b1, protocol=OF_13
port=local, state=disabled, type=copper, speed=0, portName=br-router, portMac=72:64:6b:3a:1f:4d
port=1, state=enabled, type=copper, speed=0, portName=patch-rout, portMac=02:3a:eb:05:d4:41
port=2, state=enabled, type=copper, speed=10000, portName=quagga, portMac=8e:eb:91:40:2f:44
port=3, state=enabled, type=copper, speed=10000, portName=quagga-router, portMac=62:0d:7b:17:20:ac
```

Figure 6.4: vRouter ports

Finally, it is necessary to perform again the POST curl in the ONOS deployment machine: now the result will be “*COMPLETE*“ for both nodes, as in Figure 6.6.

```

onos> hosts
id=FA:00:00:00:00:01/None, mac=FA:00:00:00:00:01, location=of:00000000000000b1/3, vlan=None, ip(s)=[172.1
8.0.1], configured=false
id=FE:00:00:00:00:01/None, mac=FE:00:00:00:00:01, location=of:00000000000000b1/2, vlan=None, ip(s)=[172.1
8.0.254], configured=false
id=FE:00:00:00:00:02/None, mac=FE:00:00:00:00:02, location=of:00000000000000b1/1, vlan=None, ip(s)=[172.2
7.0.1], name=FE:00:00:00:00:02/None, configured=true
onos> ports
id=of:00000000000000b1, available=true, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2
.5.0, serial=None, driver=softrouter, channelId=172.17.0.1:47570, managementAddress=172.17.0.1, name=of:0
0000000000000b1, protocol=OF_13
  port=local, state=disabled, type=copper, speed=0 , portName=br-router, portMac=72:64:6b:3a:1f:4d
  port=1, state=enabled, type=copper, speed=0 , portName=patch-rout, portMac=02:3a:eb:05:d4:41
  port=2, state=enabled, type=copper, speed=10000 , portName=quagga, portMac=8e:eb:91:40:2f:44
  port=3, state=enabled, type=copper, speed=10000 , portName=quagga-router, portMac=62:0d:7b:17:20:ac
onos> fpm-connections
172.17.0.3:60872 connected since 35s ago
onos> next-hops
ip=172.18.0.1, mac=FA:00:00:00:00:01, numRoutes=1
onos> routes
Table: ipv4
  Network          Next Hop
  0.0.0.0/0        172.18.0.1
  Total: 1

Table: ipv6
  Network          Next Hop
  Total: 0

onos> route-add 172.27.0.0/24 172.27.0.1
onos> routes
Table: ipv4
  Network          Next Hop
  0.0.0.0/0        172.18.0.1
  172.27.0.0/24    172.27.0.1
  Total: 2

Table: ipv6
  Network          Next Hop
  Total: 0

onos> next-hops
ip=172.27.0.1, mac=FE:00:00:00:00:02, numRoutes=1
ip=172.18.0.1, mac=FA:00:00:00:00:01, numRoutes=1

```

Figure 6.5: Internal configuration of the vRouter

```

onos> openstack-nodes
hostname=compute-01, type=COMPUTE, managementIp=10.134.231.30, dataIp=10.134.34.222, intBridge=of:0000000
0000000a1, routerBridge=Optional.empty init=COMPLETE
hostname=gateway-01, type=GATEWAY, managementIp=10.134.231.32, dataIp=10.134.34.224, intBridge=of:0000000
0000000a2, routerBridge=Optional[of:00000000000000b1] init=COMPLETE
Total 2 nodes

```

Figure 6.6: Completion of configuration

## 6.4 Testing SONA: walkthrough

This chapter is related to some tests to try the main basic functionalities of Neutron and Nova.

A possible help in performing these operation from the CLI is the cloud-init script that gives the possibility to set password of “ubuntu” user to “ubuntu” by passing it to Nova with “-user-data” option in the creation of a new VM.

### Configuration 2: cloud-config

```
#cloud-config
password: ubuntu
chpasswd: { expire: False }
ssh_pwauth: True
```

### 6.4.1 L2 Switching functionalities

Here, I created two tenant networks and the relative subnets. On the first one I created 2 VMs (net-A-01 and net-A-02), whereas on the second one 1 VM (net-B-01).

It is possible to see that net-A-01 and net-A-02 have connectivity one to the each other, but not with net-B-01: the isolation among tenant is therefore maintained.

```
neutron net-create net-A
neutron subnet-create net-A 192.168.0.0/24
neutron net-create net-B
neutron subnet-create net-B 192.168.1.0/24
```

```
nova boot --flavor 2 --image ubuntu-14.04-server-cloudimg-amd64 --
  user-data passwd.data --nic net-id=[net-A-UUID] net-A-01
nova boot --flavor 2 --image ubuntu-14.04-server-cloudimg-amd64 --
  user-data passwd.data --nic net-id=[net-A-UUID] net-A-02
nova boot --flavor 2 --image ubuntu-14.04-server-cloudimg-amd64 --
  user-data passwd.data --nic net-id=[net-B-UUID] net-B-01
```

### 6.4.2 L3 Routing functionalities

Here, first of all I created the external network, compliant with the vRouter range. Then, I created the virtual router of OpenStack and attached to both net1 and net2.

After having created the security groups to manage external traffic and updated the related ports, it is possible to see that the inter-connectivity among VMs of different networks is fulfilled. These machines are now also able to go on the Internet.

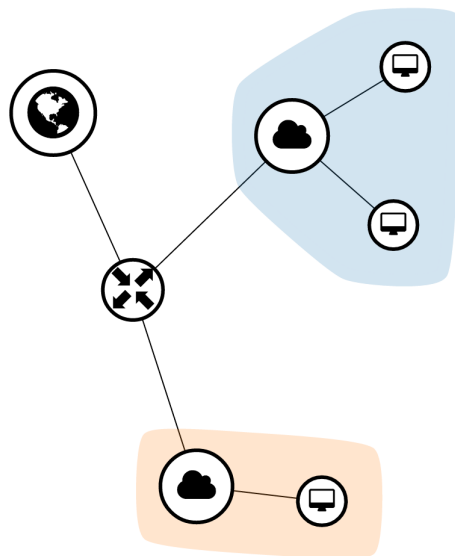


Figure 6.7: L3 topology

```
neutron net-create net-public --router:external True --provider:
  physical_network external --provider:network_type flat
neutron subnet-create net-public 172.27.0.0/24
neutron router-create router-01
neutron router-gateway-set router-01 net-public
neutron router-interface-add router-01 [net-A-subnet UUID]
neutron router-interface-add router-01 [net-B-subnet UUID]

neutron security-group-create allow-external
neutron security-group-rule-create --direction ingress --remote-ip
  -prefix 0.0.0.0/0 allow-external
```

```

neutron port-update [net-A-01 port UUID] --security-group [default
  -security-group UUID] --security-group allow-external
neutron port-update [net-B-01 port UUID] --security-group [default
  -security-group UUID] --security-group allow-external

neutron floatingip-create net-public
neutron floatingip-associate [floating-ip-id] [net-A-01 port UUID]

```

## 6.5 Case study: Orchestration

By going on and on into the cloud and the on-demand computing needs, a problem comes into consideration in deployment: the orchestration. It is indeed easy to see that whenever an instance is required, there is also the need to add it on a network, install some software, provisioning a public IP, etc. These tasks are of course very repetitive and it is therefore important to orchestrate properly the resources in order to automate the deployment and configuration of infrastructure.

### 6.5.1 Orchestration in OpenStack

Even though there are some tools already providing the orchestration functions, these are not widespread. However, Amazon Web Services provided a widely used solution by developing cloud formation templates for their users, with just a declarative script needed. OpenStack provides a module called Heat for orchestration, which is retrocompatible by need with AWS Cloud Formation template.

In OpenStack it is indeed possible to create a template that Heat understands in two ways: CloudFormation, usually written in JSON and used also in AWS or HOT (Heat Orchestration Template) [83], often written in YAML (Yet Another Markup Language) format. YAML is very easy to read and understand by non-programmers, making the HOT templates accessible to system administrators, architects, and other non-coders.

### 6.5.2 Glossary of OpenStack orchestration

- **Stack:** what it is intended to be created, a collection of VMs and their associated configuration. In Heat it also refers to a collection of resources,

which can be in turn instances, networks, security groups, and auto-scaling rules.

- **Template:** definition of the resources that will make up the stack. A template is made up of four different sections:
  - *Resources:* the objects that will be created or modified when the template runs. They could be Instances, Volumes, Security Groups, Floating IPs, or any number of objects in OpenStack.
  - *Properties:* specifics of the template.
  - *Parameters:* Properties values that must be passed when running the Heat template.
  - *Output:* what is passed back to the user.

### 6.5.3 SONA approach with Heat

The interest is in understanding what happens when an OpenStack deployment running SONA tries to perform a *stack* operation; after that, it is interesting to understand whether all the provided and needed features for orchestration are well managed by SONA.

The results of the below case study are showing what was expected: being SONA transparently called by OpenStack by means of networking-onos each time Neutron is required, the stack operation will not experience any problem as if the deployment was provided by Neutron itself.

### 6.5.4 Orchestration case study: Telegram bot template

The idea has been to create a Telegram bot. Telegram is a free and open source messaging app where users can exchange messages through the Telegram servers via HTTPS [84]. It is also possible to develop Bots via provided APIs.

Therefore, after the creation of a simple Python code dealing with the Telegram APIs to answer to the incoming messages, a template has been written in order to test the providing of all the basic network resources. The stack is composed of multiple resources: an instance, its private net and subnet, the router and a floating IP on the public net. It is coded into a Yang Model template (yaml) following the HOT reference.

Once the instance boots, it will download the bot source code, the needed dependencies and run it automatically. Moreover, I included a condition for which Heat is notified of the complete deployment only after all operations have been performed, thus only when the bot is actually up and running.

Moreover, I had to change the MTU to 1400 as a result of the overhead due to the presence of the VxLAN tunnel.

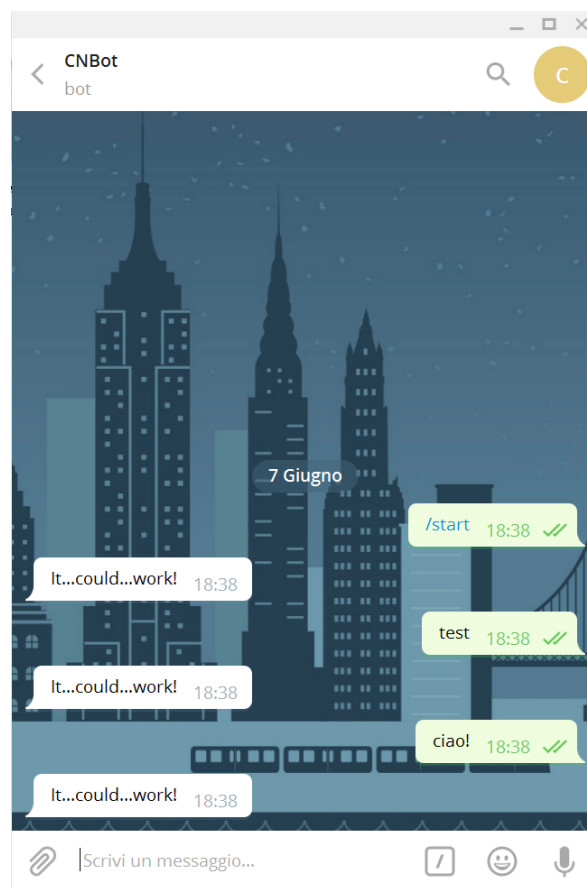


Figure 6.8: Bot working after the deployment



## 6.6 Case study: Gathering information

### 6.6.1 Using ONOS to monitor OpenStack

After the introduction of ONOS inside the OpenStack environment, all the requests which are destined to Neutron will be post-committed to SONA through the networking-onos component, which will also change the format of the request. Indeed, SONA and Neutron expose different REST APIs: the interest may therefore be to understand which are the ONOS API and which data they gather.

In particular, as ONOS has to provide the Neutron functionalities, it has to collect some data related to networks, hosts, flows, etc. In general, all these statistics may be used to produce a very coarse-grained monitor and manager.

### 6.6.2 Case study: the sonaMonitor script

The focus has been put on the monitoring phase rather than the management one, but the idea behind is to have the possibility to gather data on hosts, flows and statistics. I have written a simple Python script to make possible to ask continuously information about the OpenStack platform. This script is fully extensible and ready to add other modules related to different collected data.

The workflow is:

1. the user interacts via the CLI asking for a specific information;
2. the script performs the REST call;
3. a JSON is received back;
4. the JSON is parsed;
5. the user receives the information required;
6. repeat from 1)

These are just some of the GET calls that the script uses to fetch the data:

```
curl -u karaf:karaf -X GET --header 'Accept: application/json' 'http://10.134.231.29:8181/onos/v1/flows'
```

```

curl -u karaf:karaf -X GET --header 'Accept: application/json' '
  http://10.134.231.29:8181/onos/v1/hosts '
curl -u karaf:karaf -X GET --header 'Accept: application/json' '
  http://10.134.231.29:8181/onos/v1/network/configuration '
curl -u karaf:karaf -X GET --header 'Accept: application/json' '
  http://10.134.231.29:8181/onos/v1/statistics/ports '
curl -u karaf:karaf -X GET --header 'Accept: application/json' '
  http://10.134.231.29:8181/onos/v1/tenants '
curl -u karaf:karaf -X GET --header 'Accept: application/json' '
  http://10.134.231.29:8181/onos/v1/topology '

```

```

[francesco@workstation: ~/Dropbox/Workspace/python/sona_manager] $ python manager.py
Hi and welcome! This is a simple script showing you the possibility to exploit ONOS API to monitor OpenStack
First of all: what would you like to monitor?
Hosts, flows or stats?
hosts
Your choice is: hosts

What are you interested to see?
1) Show all the hosts
2) Show the hosts related to a particular device
3) Show the hosts related to a particular network
4) Show the hosts related to a particular tenant
1
[ { u'annotations': { u'gatewayIp': u'192.168.1.1',
                    u'networkId': u'048d242f-1f06-455d-852e-26cc27e83262',
                    u'portId': u'7438c3d2-a1c0-4904-884a-c9265fc3737b',
                    u'subnetId': u'b5541c63-b4be-4a66-b9b0-931306b589cf',
                    u'tenantId': u'b0a3c67d349840c1aa857441c6b373a2',
                    u'vlanId': u'1053'},
  u'configured': False,
  u'id': u'FA:16:3E:0E:EF:36/None',
  u'ipAddresses': [u'192.168.1.2'],
  u'location': { u'elementId': u'of:0000000000000a1', u'port': u'11'},
  u'mac': u'FA:16:3E:0E:EF:36',
  u'vlan': u'None'},
{ u'annotations': { u'gatewayIp': u'192.168.0.1',
                  u'networkId': u'28b6fbe7-525f-4564-87f0-7b8b081a85e2',
                  u'portId': u'c9e3b679-1307-41ac-98ce-b3afdbafc3c3',
                  u'subnetId': u'e2925d9d-62b0-42b1-a6db-bd645729f2ad',
                  u'tenantId': u'b0a3c67d349840c1aa857441c6b373a2',
                  u'vlanId': u'1013'},
  u'configured': False,
  u'id': u'FA:16:3E:2E:B5:2B/None',
  u'ipAddresses': [u'192.168.0.2'],
  u'location': { u'elementId': u'of:0000000000000a1', u'port': u'9'},
  u'mac': u'FA:16:3E:2E:B5:2B',
  u'vlan': u'None'},
{ u'annotations': { u'gatewayIp': u'192.168.0.1',
                  u'networkId': u'28b6fbe7-525f-4564-87f0-7b8b081a85e2',
                  u'portId': u'94e19d5a-0073-4dd2-b036-5bd79224ea7c',
                  u'subnetId': u'e2925d9d-62b0-42b1-a6db-bd645729f2ad',
                  u'tenantId': u'b0a3c67d349840c1aa857441c6b373a2',
                  u'vlanId': u'1013'},
  u'configured': False,
  u'id': u'FA:16:3E:13:3D:A3/None',
  u'ipAddresses': [u'192.168.0.3'],
  u'location': { u'elementId': u'of:0000000000000a1', u'port': u'10'},
  u'mac': u'FA:16:3E:13:3D:A3',
  u'vlan': u'None'}]

```

Figure 6.9: Simple example of a sonaMonitor execution

## 6.7 Limitations and future works

The SONA approach poses some limitations:

- A tenant cannot create more than one subnet with same IP address range even in different virtual networks; however, the same subnet can be defined across tenants. Therefore multi-tenancy is respected, whereas the isolation is maintained only up to a certain degree.
- It is not possible to change an existing security group at runtime; it is however possible to create a new security group and update the port of the instance through the neutron API to get an almost runtime-like change.
- The connection tracking feature is available only in some recent versions of OVS.
- It does not allow ingress traffic via a connected session by default, due to an OpenFlow mapping rule issue: there is the need to add an allowing rule for ingress direction with remote address “0.0.0.0/0” explicitly for the VM to be able to access the Internet.

Although the community is completely refactoring SONA to make it compatible with the newer versions of OpenStack, its development is, at the moment of writing, discontinued [85]. Therefore, it is possible to define this project as an interesting solution that, however, might just provide a PoC of its functionalities. Indeed, it was not possible to deploy SONA either on a Mitaka, Newton or Ocata production cluster, for incompatibility issues. As a consequence of this, it was not possible to take a more consistent and interesting performance evaluation of this approach.



# Chapter 7

## Testbed implementation

### 7.1 OpenStack production environment deployment

In order to proceed with some performance evaluation, a testbed has been properly setup. The aim has been to have a clean OpenStack production environment for a performance evaluation of different firewalls in different compute nodes. To manage the networking of the different nodes, the Ryu SDN framework has been chosen. This choice has been led by its complete support from the Neutron community; moreover, Ryu introduces a minimal overhead in the platform (no additional dedicated nodes, no complete backend replacement, etc.) with respect to the other solutions. Therefore, by using Ryu, the evaluation is related to an almost-out-of-the-box OpenStack approach, eventually finely tuned.

#### 7.1.1 Topology

The topology that has been setup is presented in the Figure 7.1. The OpenStack nodes are powerful servers (each compute has 40 vCPUs, 64 GB RAM) and the choice has been to not deploy an additional node for the L3 network functions (network node), as long as the tests are more related to internal OpenStack components. Another reason for it has been that the controller node is able to achieve high performance and therefore is able to keep up with the networking functions in addition to the controller ones.

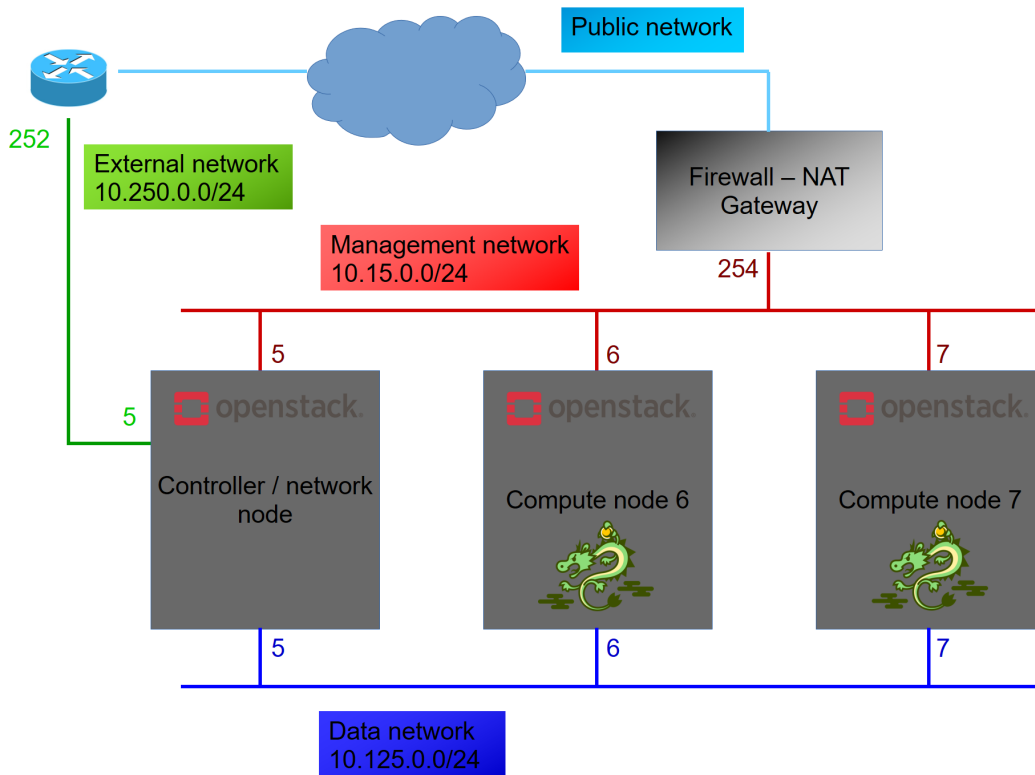


Figure 7.1: OpenStack topology

Each node is connected to both the management network and the data network, whereas the controller node, as long as it provides also network functionalities, is connected to the external network. All the nodes are running on CentOS 7 and the version of OpenStack is Newton, released in October 2016. The installed hypervisor for both nodes is KVM/QEMU.

During the next subsections, some configuration files are cited. Their final version is shown in Appendix A.4, where passwords have been modified for security reasons.

### 7.1.2 Prerequisites

The first needed thing to avoid problems while working with CentOS has been to disable the firewall: of course, this solution is employable only as long as

the whole cluster is behind a real firewall protecting the machines from the outside. Otherwise, a proper configuration of the CentOS firewall would be needed.

After the configuration of interfaces of the different nodes in the cluster, a NTP client/server mechanism has been deployed to bound all the nodes to the same time. The controller node acts as a client for an outside server to get the correct time; in the meanwhile, it is also a server for the other nodes. Indeed, the controller node is referencing more accurate lower stratum servers and other nodes are referencing the controller node. The chosen solution is called *chrony*, easily available in most Linux distros.

This NTP solution is employable as long as the topology remains a three-level topology, which is still affordable for an OpenStack cluster. The more levels will be present in this topology indeed, the less accuracy in the time synchronization will be present as a result of how the NTP protocol references.

Therefore, *chrony* is installed in controller and compute nodes, with different configurations (always refer to Appendix A.4).

After that, it is possible to add the OpenStack repositories to all nodes, to update to the latest available ones and install the python client which is the main API from the CLI that OpenStack provides. Moreover, it is recommended to install the package related to the management of SELinux to manage the policies for OpenStack services.

In order to provide a DataBase to the services, it is important to install a stable relational DB on the controller node, as MySQL. The choice has been to install MariaDB. An important thing to be performed is the deletion of all the configurations that come preinstalled with it, as the */etc/my.cnf*, otherwise the DB will not work as it loads those as the default ones.

OpenStack supports several message queue services including RabbitMQ, Qpid, and ZeroMQ. One of the most supported ones is RabbitMQ and it is the one used as a middleware for the deployment of this cluster in the controller node. It is important to create an openstack user that can access to the queues and to properly set its permissions.

The Identity service authentication mechanism for services uses Memcached to cache tokens. A combination of firewalling, authentication, and encryption to secure it would be needed in a production environment, but as long as here the deployment is running behind a real firewall, this is not needed.

### 7.1.3 Identity service: Keystone

To implement Keystone on the controller node it is important to deploy a token mechanism and a HTTP server to handle the requests. Here, Fernet tokens and Apache HTTP server are used for scalability purposes.

First of all, a database (*keystone*) has been created, and the access permissions have been granted to it. After installing the keystone component, the HTTP server and the *mod\_wsgi*, a proper configuration has been addressed (it is possible to see it in Appendix A.4). It is important to state that by proceeding in this way with the Apache HTTP server and *mod\_wsgi*, the result will be to handle Identity service requests on ports 5000 and 35357 through the Apache HTTP server, by disabling therefore the Keystone service to directly listen on them.

The Identity service DB is populated and the Fernet key repositories are initialized. Finally Keystone is bootstrapped with an admin user and the HTTP server is configured.

To deal with the authentication, OpenStack needs to be configured to deal with a combination of domains, projects, users and roles as shown in Chapter 3. Therefore, it is vital to define a *service* project that will contain a single user for each service to be added to the environment. It is then possible to create an unprivileged project and user, with the corresponding role to be added.

### 7.1.4 Image service: Glance

Glance requires as well its database and user creation, by then adding the admin role to the Glance user and service project. Therefore, it is needed to create the Glance service entity and its API endpoints (public, internal and admin). All these API endpoints are residing in the controller node, which is running behind a firewall/NAT easily managed to only forward to the controller the requests that are compliant with a security policy.

Finally, it is possible to install the Glance component on the controller, to configure it and to populate its service database.

### 7.1.5 Computing service: Nova

Before Nova installation and configuration, there is the need on the controller node to create the databases (one for nova and one for nova\_api), the service



credentials and the API endpoints. In particular, a nova user is created and the admin role is given to it; to Nova there will be 3 associated API endpoints.

Nova has been then installed and configured on the controller node. Among all the settings, there has been the need to enable the support for the Neutron networking service: this has to be done because in the past Nova provided also some network functionalities, now old and legacy.

Nova has also been installed and configured in a slightly different way on all the compute nodes.

```
[root@controllerNew ~]# nova hypervisor-list
+-----+-----+-----+-----+
| ID | Hypervisor hostname | State | Status |
+-----+-----+-----+-----+
| 1 | compute6 | up | enabled |
| 2 | compute7 | up | enabled |
+-----+-----+-----+-----+
```

Figure 7.2: Nova hypervisors

### 7.1.6 Networking service: Neutron

As for the previous components, on the controller node a DB has been created, as well as the service credentials and the API endpoints.

It has been then chosen to deploy the “self-service networks” option, where it is possible to have more functions than the “provider networks” one. Indeed, with self-service networks, the user is able to create and manage (private) networks, routers, or floating IP addresses. Basically, a self-service network is a tenant network, therefore provided with all the Neutron abstractions shown in Chapter 4. Therefore, some additional steps were considered in the configuration of the Neutron service, in particular of the ML2 plug-in. Indeed, the chosen tenant network type has been the VXLAN and the OpenVSwitch has been chosen and configured as the ML2 mechanism driver. Finally the metadata and DHCP agent, as well as the compute service have been configured.

With respect to the compute node instead, the common network component, the OvS agent and the compute service have been configured. In general,

L2 agents can be configured to use differing firewall drivers as there is no requirement that they have to be all the same. If an agent lacks a firewall driver configuration, it will default to what is configured on its server. Therefore, to deploy a suitable case study for the evaluation of performance related to different firewall approaches, compute6 has been deployed with a hybrid iptables-openvswitch mechanism, whereas compute7 has been configured to run with an OvS native firewall driver. The native OVS firewall implementation requires kernel and user space support for the *conntrack*, thus requiring minimum versions of the Linux kernel (4.3 or newer, otherwise between 3.3 and 4.3 there is the need to build it) and Open vSwitch (2.5 or newer). For instance, on CentOS 7 the *conntrack* module was already present and loaded at the time of the deployment, even though the Kernel version was 3.10.

All the nodes are running with the default Ryu native OpenFlow interface. In addition to it, it has been considered the way in which Neutron calls the Ryu `app_manager`. As a result on this, it has been possible to make Ryu one of its modules, the REST interface (actually, to make Neutron load Ryu loading its REST interface). This opens a huge number of possibilities as, by doing so, it is possible to write simple applications that instruct Ryu in order to manage the network, according to a policy, in a smart way. For example, an application might consider to retrieve from Ryu some information about the network and decide that a certain node is congested and therefore to install in the same bridge a rule with a higher priority to perform some congestion control recovery by redirecting the traffic to another node. Another example might be an application that performs service function chaining to maintain the QoS related to the accorded SLA of the user. In particular, a possible case study is the use of the REST API in each compute node and to deploy an orchestrator on top of the cluster that has a complete knowledge of the network (through the communication with OpenStack APIs) and is able to insert flow rules inside the different integration bridges, to perform SFC by taking advantage of the SDN integration.

It would be however easy to deploy the topology with a separate network node. Indeed, the controller node will only manage the common Neutron component, whereas all the other network functions that are now deployed in the controller will be placed in the network node.

```
[root@controllerNew ~]# neutron agent-list
```

| id                                   | agent_type         | host          | availability_zone | alive | admin_state_up | binary                    |
|--------------------------------------|--------------------|---------------|-------------------|-------|----------------|---------------------------|
| 6ddcce7e-37ce-42ed-8183-c3a3d9089990 | Open vSwitch agent | compute7      |                   | :)    | True           | neutron-openvswitch-agent |
| 80427e2f-cf80-4dff-bdcb-4fa1bf437f00 | Metadata agent     | controllerNew |                   | :)    | True           | neutron-metadata-agent    |
| 8b2c7772-b242-40d7-91f2-c1c76feabb84 | Open vSwitch agent | controllerNew |                   | :)    | True           | neutron-openvswitch-agent |
| 8da39c6c-9577-4068-95b4-a889f247545b | DHCP agent         | controllerNew | nova              | :)    | True           | neutron-dhcp-agent        |
| bd7fb665-ac4b-4ca7-a894-40670972e8bc | Open vSwitch agent | compute6      |                   | :)    | True           | neutron-openvswitch-agent |
| de8a1a43-fcad-40c8-9bbe-3b3718413d91 | L3 agent           | controllerNew | nova              | :)    | True           | neutron-l3-agent          |

Figure 7.3: Neutron agents

### 7.1.7 Dashboard service: Horizon

In a very easy way, also the dashboard has been installed and configured. This lets the end user to easily manage his/her tenants, by making use of a website that is able to perform REST calls to other components, as the command was launched by the CLI. Moreover, it is possible to connect to virtual instances via noVNC.

### 7.1.8 Orchestration service: Heat

To add orchestration capabilities to the cluster, Heat has been deployed and configured as well. Its installation has consisted in the creation of its DB and its user, the creation of the orchestration services and of the endpoints. In particular, the services are two as there is the usual OpenStack orchestration but also the cloudformation one, which is compatible with other cloud solutions as Amazon EC2. Finally, a stack owner is created among the roles as well as a stack user.

### 7.1.9 Telemetry service: Ceilometer

In general, when a cloud service is offered the provider have consider a way to calculate bills to its users. This usually leads to three steps: metering, rating

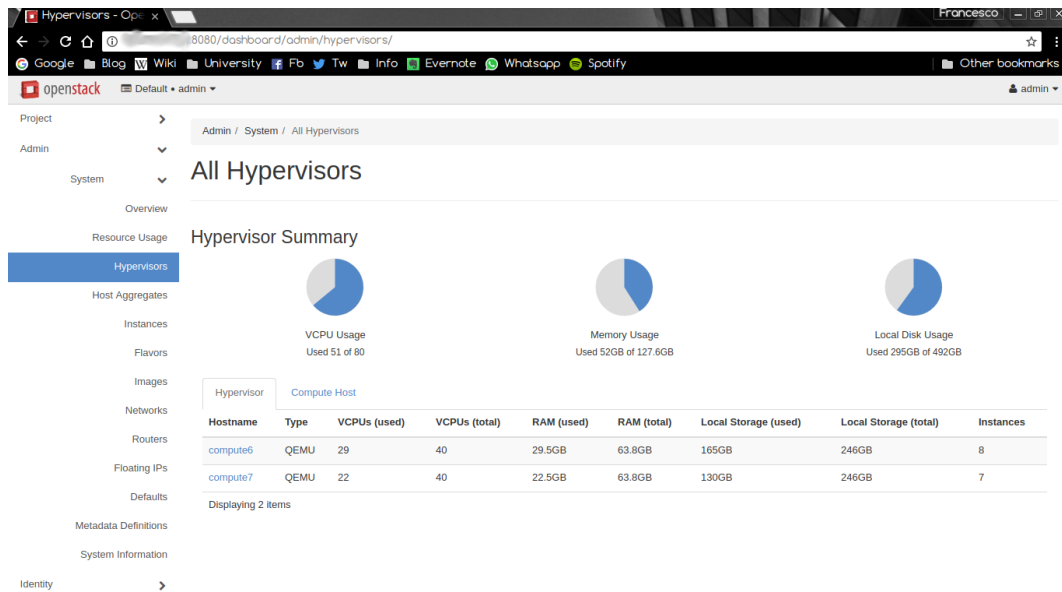


Figure 7.4: Dashboard

and billing. In OpenStack, in order to meter the infrastructure, a component called Ceilometer is used: Ceilometer creates messages every time something is measured. These messages are collected and stored. There are then two ways to collect the data that is created by Ceilometer: using a notification agent that listens on the notification bus or a polling agent that periodically contacts for the data via the related API. The data gets collected and normalized to be stored in a database. Since there are lots of pieces of information being captured and needed to be stored, the requirements for database performance are high.

To deploy the Telemetry service therefore, unlike other services, a non-relation DB is needed. In particular, a NoSQL DB has been used for this deployment: MongoDB. However, it is possible to run other DBs as the Ceilometer backend, like Gnocchi which is considered to become the default solution in the future of Ceilometer.

| Project   | Service | Meter                         | Description                                                        | Day        | Value (Avg)      | Unit      |
|-----------|---------|-------------------------------|--------------------------------------------------------------------|------------|------------------|-----------|
| admin     | Nova    | network.outgoing.packets.rate | Average rate per sec of outgoing packets on a VM network interface | 2017-09-23 | 0.0670146631387  | packet/s  |
| admin     | Nova    | disk.write.requests.rate      | Average rate of write requests                                     | 2017-09-23 | 0.0284241059518  | request/s |
| admin     | Glance  | image                         | Image existence check                                              | 2017-09-23 | 1.0              | image     |
| UNIBO-AGH | Glance  | image                         | Image existence check                                              | 2017-09-23 | 1.0              | image     |
| unibonos  | Glance  | image                         | Image existence check                                              | 2017-09-23 | 1.0              | image     |
| admin     | Nova    | disk.ephemeral.size           | Size of ephemeral disk                                             | 2017-09-23 | 0.0              | GB        |
| UNIBO-AGH | Nova    | disk.ephemeral.size           | Size of ephemeral disk                                             | 2017-09-23 | 0.0              | GB        |
| admin     | Nova    | disk.read.bytes               | Volume of reads                                                    | 2017-09-23 | 440,994,337.185  | B         |
| admin     | Nova    | network.outgoing.bytes.rate   | Average rate per sec of outgoing bytes on a VM network interface   | 2017-09-23 | 23.3305999525    | B/s       |
| admin     | Glance  | image.size                    | Uploaded image size                                                | 2017-09-23 | 428,641,689.6    | B         |
| UNIBO-AGH | Glance  | image.size                    | Uploaded image size                                                | 2017-09-23 | 722,993,152.0    | B         |
| unibonos  | Glance  | image.size                    | Uploaded image size                                                | 2017-09-23 | 4,125,032,448.0  | B         |
| admin     | Nova    | disk.write.bytes              | Volume of writes                                                   | 2017-09-23 | 2,211,456,734.81 | B         |
| admin     | Nova    | cpu_util                      | Average CPU utilization                                            | 2017-09-23 | 1.14692830546    | %         |
| admin     | Nova    | vcpus                         | Number of VCPUs                                                    | 2017-09-23 | 4.0              | vcpu      |

Figure 7.5: Ceilometer statistics



# Chapter 8

## Performance evaluation

In Chapter 7 the physical testbed has been presented. In this chapter, some measurements performed on a logical topology implemented on it will be shown and commented. To do that, a topology has been therefore deployed, with resources on the inside (virtual instances) and on the outside (physical servers) of the OpenStack domain.

This topology involves 5 virtual machines per compute node, which can be considered as 4 clients and 1 server for the sake of the measurements. Each virtual instance is deployed with 4 vCPUs, 4 GB RAM and the same 20 GB disk to let the tests show some appreciable range changes. To obtain a correct CPU and memory evaluation, only the needed clients are turned on or off for each test (i.e. if the test is thought for having only measurements among two clients and the server, just client1 and client2 will be turned on). It is important to remark that the two compute nodes have been configured precisely in the same way (apart from the Neutron firewall, as explained in Chapter 7); it is therefore possible to properly evaluate the differences between the “hybrid iptables” firewall (from now on, Linux Bridge FW) and “Open VSwitch native firewall” (OvS FW) in terms of performance.

The real implemented testbed is shown in Figure 8.2, whereas in Figure 8.1 it is possible to see the dashboard (user) view, which is totally transparent with respect to the implementation. As it is possible to see, to perform some measurements, an additional physical node is placed in the External network, where it is able to reach the virtual instances via all the Neutron mechanisms shown in Chapter 4. It is also possible to remark the presence of two VXLAN interfaces, as the number of nodes with networking functions is 3 (therefore,

each node must be able to reach the other two). Moreover, there are two network namespaces related to routers and four ones for DHCP, as there are four private networks of which only two are connected to the external network.

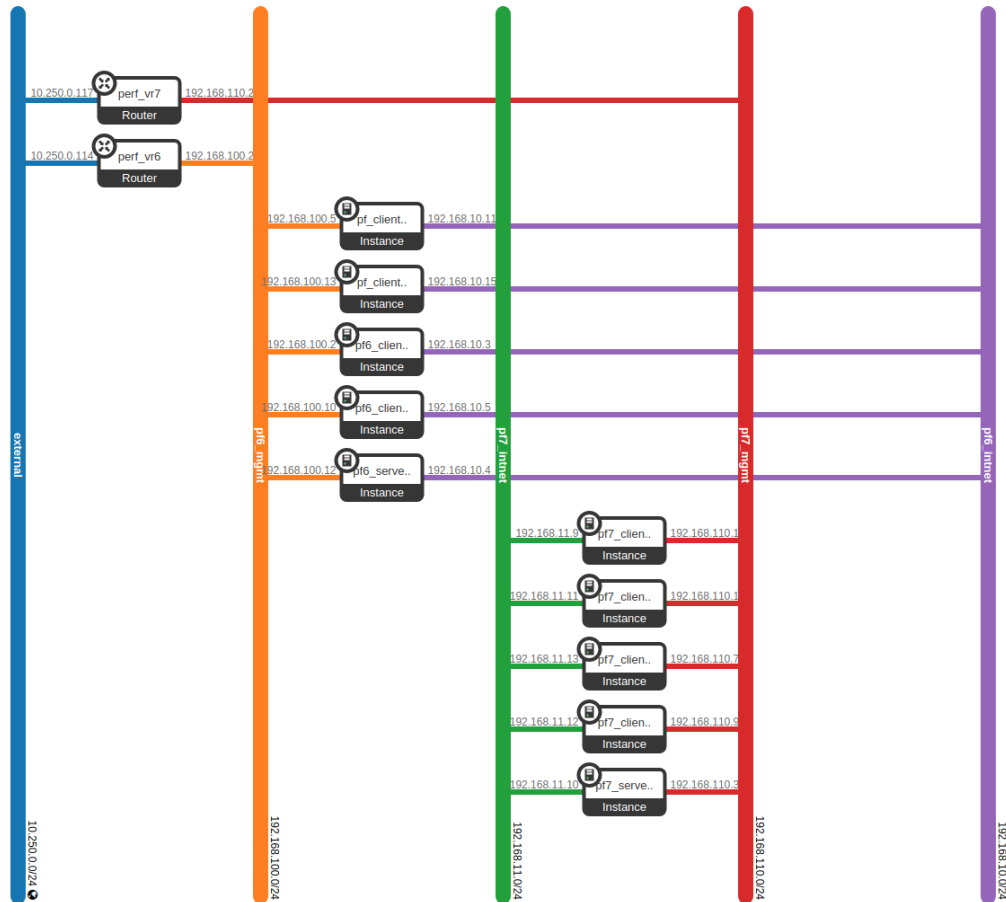


Figure 8.1: User view of the testbed



## 8.1 In-node process throughput

To have a rough idea of what could be an upper bound of the achievable throughput on a node, *iperf3* has been used. Indeed, with every VM off, on each compute node an iperf3 server has been bounded to the loopback interface, and an iperf3 client has been started for 60 seconds. The measurement has then been repeated 30 times, for each compute node. Thus, in total there has been 30 TCP measurements on compute node 6 (LB FW) as well as 30 on compute node 7 (OvS FW). UDP measurements have not been performed as, for the single client iperf limits the sending rate to the maximum achievable while filling the socket. The results are shown in Table 8.1.

| Firewall mechanism     | Mean throughput [Gbps] | Standard deviation [Gbps] |
|------------------------|------------------------|---------------------------|
| Linux Bridge FW        | 40.457                 | 2.265                     |
| Open vSwitch native FW | 49.860                 | 4.503                     |

Table 8.1: In-node process throughput

As it is possible to state from the above Table, the node employing the OvS native firewall performs better in terms of throughput. The node with the LB FW underperforms as a result of the presence of multiple iptables rules in the INPUT chain, which are created by Neutron (as shown in Chapter 4). Therefore in this compute node, each flow coming from the client has to go through all the INPUT chain until the firewall decides that it is admitted. Thus, in this case the LB firewall is adding a penalty in terms of performance.

However, it is important to remark that the measurement has been just performed to show how much the LB approach (with the iptables rules) is influencing the node performance.

## 8.2 Latency

By considering each compute node with only two virtual instances active, some evaluations on latency can be performed. In order to do that, the *netperf* tool [86] (released by HP as Open Source) has been used and the Request/Response mechanism has been used. Generally speaking, netperf request/response performance is quoted as “transactions/s” for a given request and response size.

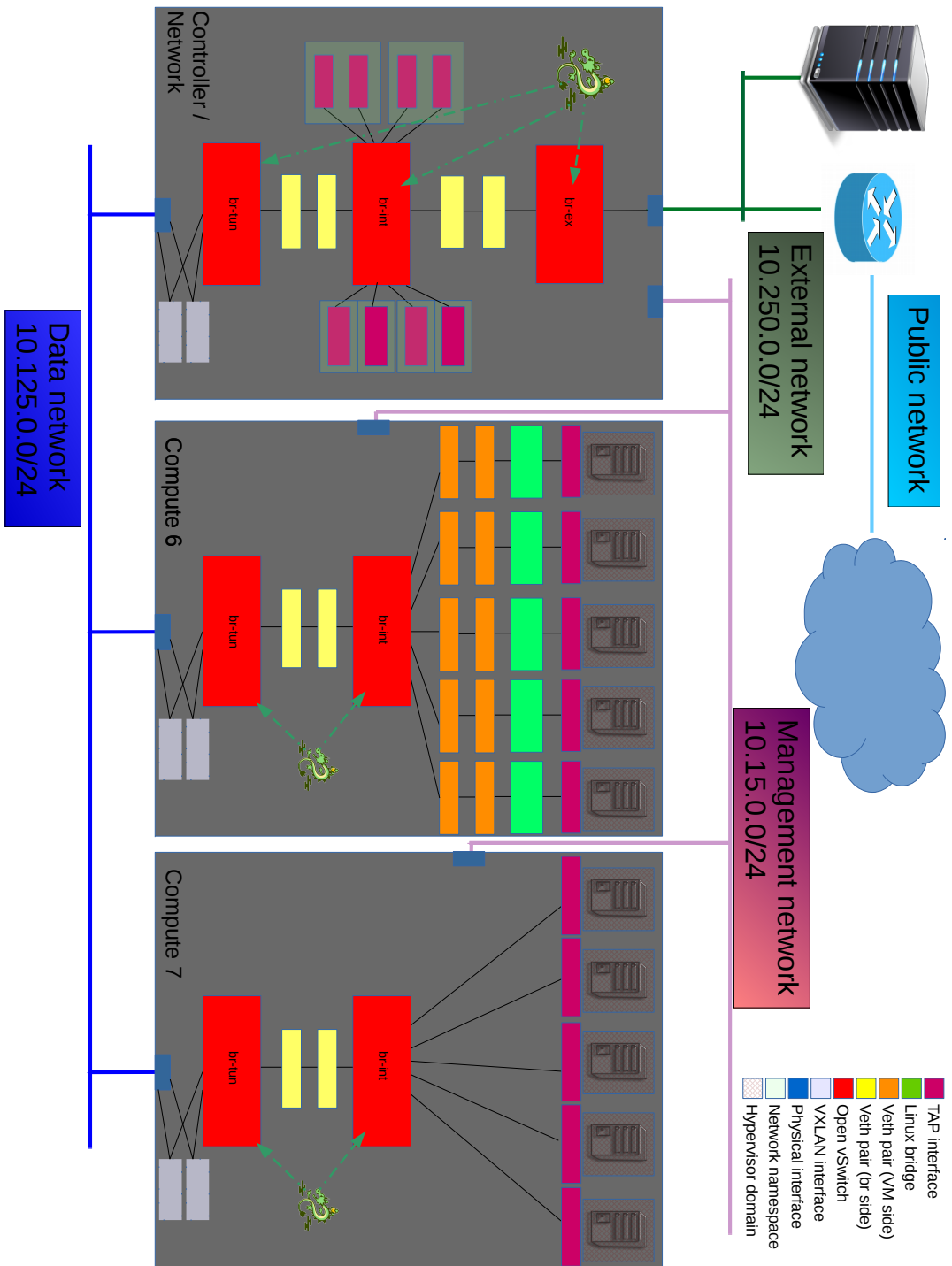


Figure 8.2: Physical implementation of the testbed

A transaction is defined as the exchange of a single request and a single response. From a transaction rate, one can infer one way and round-trip average latency.

In the single compute node, a VM acted therefore as the “netserver”, while the other one as the client. The client therefore contacts the server with a “Request” and gets back a “Response”, evaluating the time elapsing between the starting of the Request and the reception of the Response. These measurements have been performed both for TCP and UDP 30 times, for 30 seconds each, separately on each compute node. Therefore there has been 60 runs of it for the LB FW (30 TCP, 30 UDP) and 60 runs for the OvS FW (again, 30 TCP and 30 UDP).

| Firewall mechanism     | Protocol | Mean latency [ $\mu$ s] | Standard deviation [ $\mu$ s] | Maximum latency [ms] |
|------------------------|----------|-------------------------|-------------------------------|----------------------|
| Linux Bridge FW        | TCP      | 56.035                  | 3.011                         | 9.678                |
| Open vSwitch native FW | TCP      | 51.19                   | 2.066                         | 5.851                |
| Linux Bridge FW        | UDP      | 51.335                  | 2.594                         | 8.337                |
| Open vSwitch native FW | UDP      | 44.715                  | 3.251                         | 6.609                |

Table 8.2: Latency evaluation

It is remarkable to show that the OvS FW approach performs better than the LB one, as a result of the absence of internal components in between the VM and the Open vSwitch, apart from the tap device. Moreover, also the maximum latency is shown here. Although these are casual peaks, this additional evaluation has been performed to provide to the reader a rough idea of the complexity to be compliant with the 10 ms latency requirement requested by the 5G standard. This happens as a result of a lack of integration among the Communication Technologies world with these novel IT tools. However, it is shown that the OvS FW causes lower peak values with respect to the LB FW.

### 8.3 Co-located instances measurements

In order to understand the pros and the cons of the different firewall approaches, some measurements among the instances on the same compute node have been performed.

### 8.3.1 TCP throughput evaluation

This test has been performed through the use of the TCP\_STREAM mode in netperf. For this test, 30 runs of 60 seconds each have been evaluated and then computed.

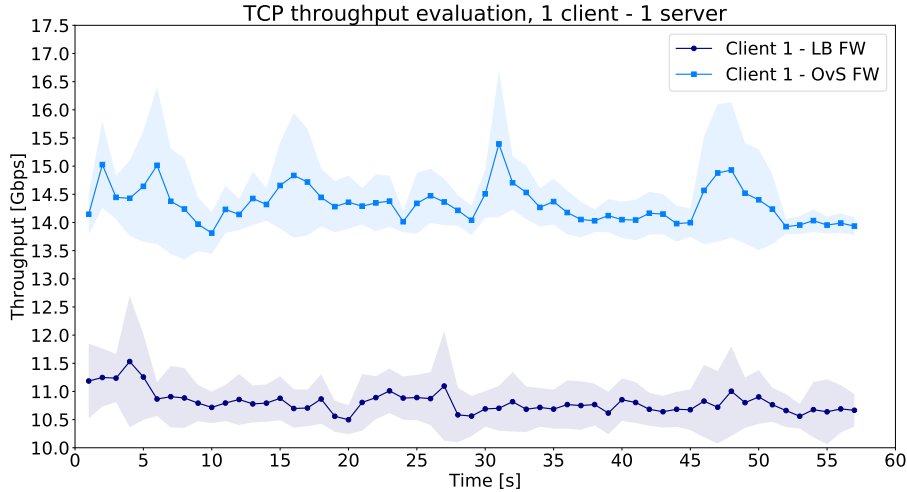


Figure 8.3: TCP throughput - 1 client 1 server

In Figure 8.3 it is shown that the use of an OvS FW approach provides a higher TCP throughput with respect to the LB FW one. This shows how much the presence of the Linux Bridge in between the virtual instance and the OvS is causing a decrease in the performance.

In Figure 8.4 it is shown a test performed by two clients and one server per node, where the first client starts sending data at  $t = 0$  s and the second one at  $t = 30$  s. This Figure shows the TCP contention phase among the 2 clients per node and it is interesting to notice that the node with the OvS FW reaches higher throughput values with respect to the LB FW one. As expected, in the long run the two clients will achieve a similar throughput. However, it is remarkable that the sum of these two flows, for instance at time  $t = 60$  s in the node with the LB FW, is larger than the throughput that the one achieved by the single TCP flow at  $t = 0$  s (and even when only 1 client was employed in the previous experiment).

$$Total_{thr} = 7.36 + 7.28[Gbps] = 14.64[Gbps] < 13.57[Gbps]$$

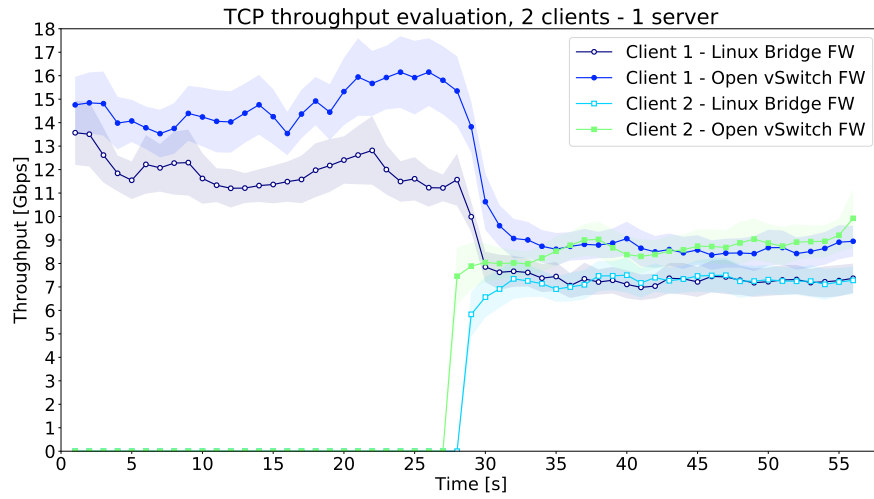


Figure 8.4: TCP throughput - 2 clients 1 server

This applies also to the other node.

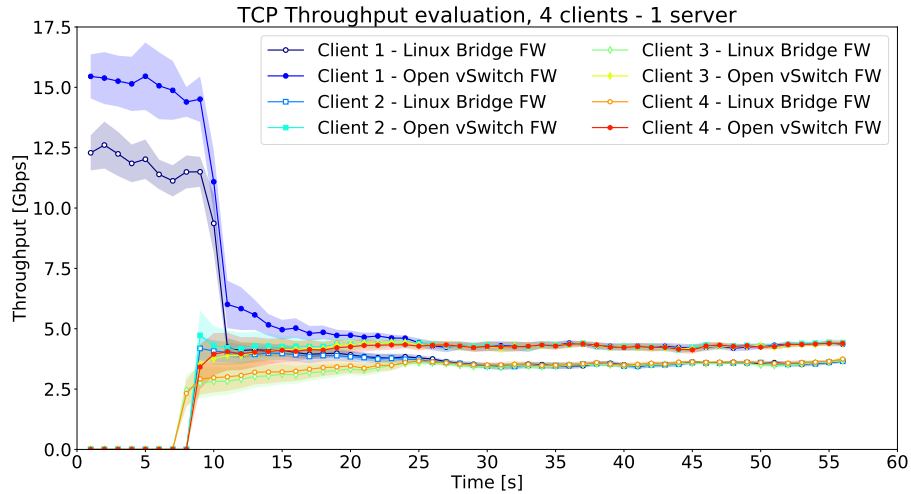


Figure 8.5: TCP throughput - 4 clients 1 server

Similarly, another proof can be seen in Figure 8.5 where 4 clients are con-

sidered. In particular, client1 starts at  $t = 0$  s, whereas the other 3 ones are generating traffic from  $t = 10$  s. The results are similar to those achieved in the case with 2 clients. Therefore, it is possible to say that this platform is able to deal in an *optimized* way with aggregate flows.

### 8.3.2 CPU evaluation

This performance has been tested through the use of the System Activity Report (sar) Unix tool [87], properly tuned to gather verbose information from the compute node about the memory and the CPU utilization (here plotted in user, system, software, guest and used -total-).

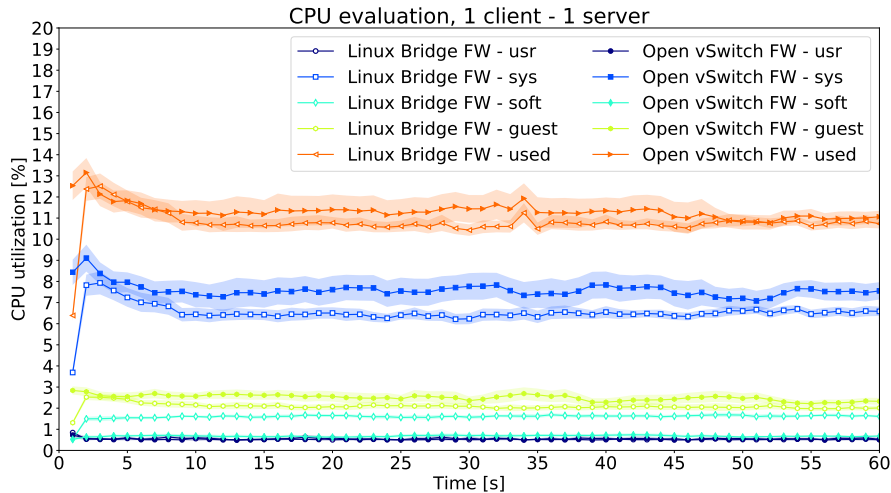


Figure 8.6: CPU usage during the TCP test with 1 client and 1 server

From Figure 8.6 it is possible to state that, on the average, the used CPU whenever a OvS FW approach is employed is slightly higher than the LB one. The total (or used) line shows that the OvS approach is more proactive than the LB one, as it is mostly constant, even at the beginning of the measurement. Instead, the LB approach has a peak between  $t = 0$  s and  $t = 1$  s, coming from the system activity and probably it is a result of the increase of communication among the user space and the kernel space, even though its average system value is still lower than the OvS one. On the other hand, it is interesting to

note a small but remarkable difference in the software activity: the OvS FW has a lower consumption with respect to the LB FW.

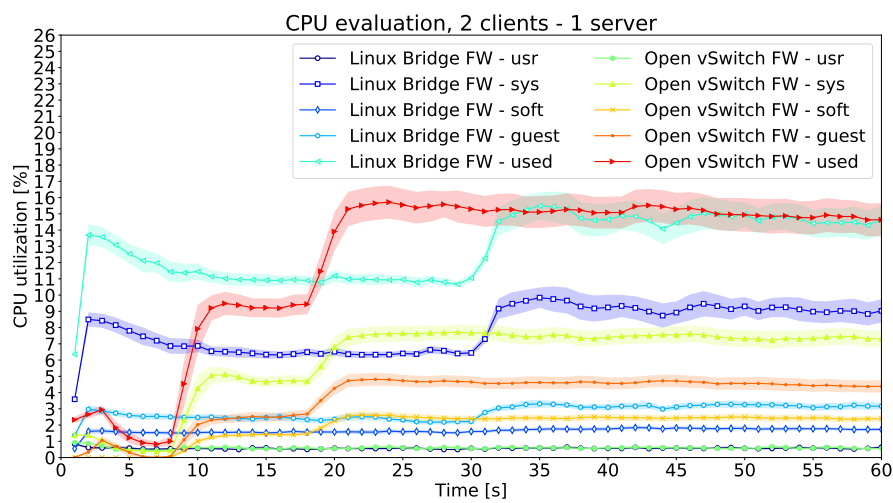


Figure 8.7: CPU usage during the TCP test with 2 clients and 1 server

In Figure 8.7 it is possible to appreciate the entire dynamic, taking into consideration the arrival of a second client at  $t = 30$  s. At that point the CPU consumption of the LB FW approach is mostly equal to the one of the OvS. Moreover, it is possible to see more in detail that the part related to the system activity increases. This however is compensated by the increment of the guest and software activities in the OvS FW-based node.

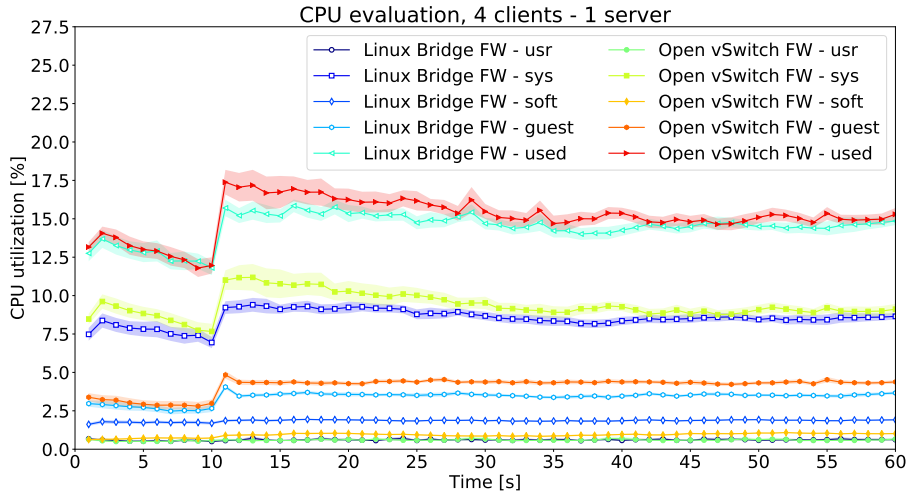


Figure 8.8: CPU usage during the TCP test with 4 clients and 1 server

Finally, in Figure 8.8 it is shown that the CPU utilization for the two different nodes is similar even when 3 clients are starting to exchange information at  $t = 10$ s, in addition to the already present client.

Therefore it is possible to say that whenever few virtual instance activities are considered, a LB FW has a lower CPU consumption, whereas the OvS FW has in general a more proactive approach and therefore might be more stable whenever multiple instances are exchanging traffic. A LB FW approach is thus more suitable for a low-traffic node in terms of CPU utilization, even though in data centers it is customary to not have idle nodes in order to increase its efficiency. In general, however, it is possible to see that whenever there is scaling among the number of instances in the physical node, the CPU consumption is mostly the same for the two approaches.



### 8.3.3 Memory evaluation

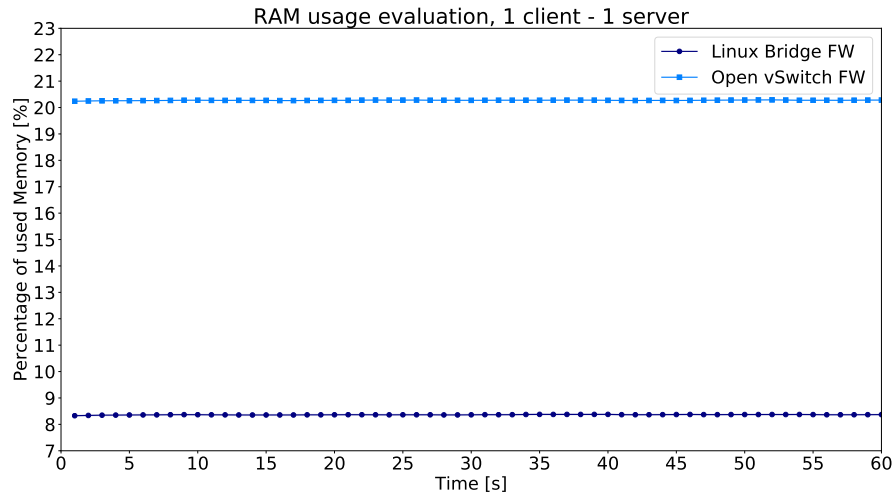


Figure 8.9: Memory usage during the TCP test with 1 client and 1 server

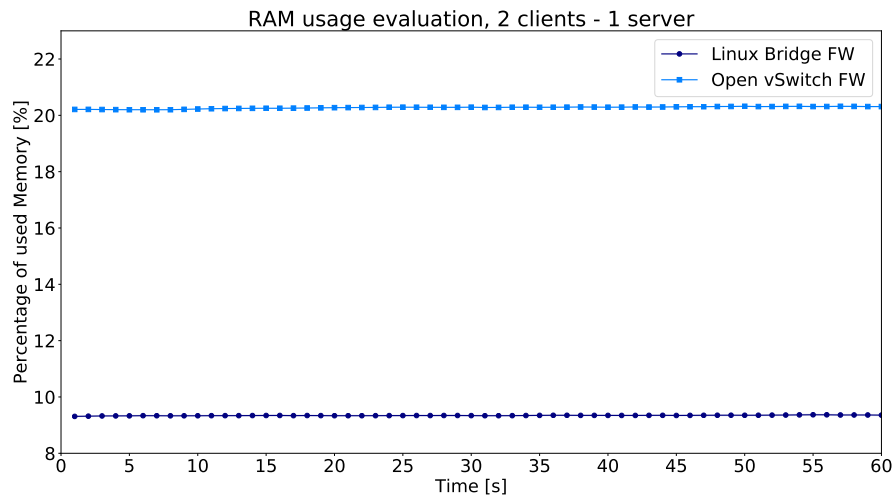


Figure 8.10: Memory usage during the TCP test with 2 clients and 1 server

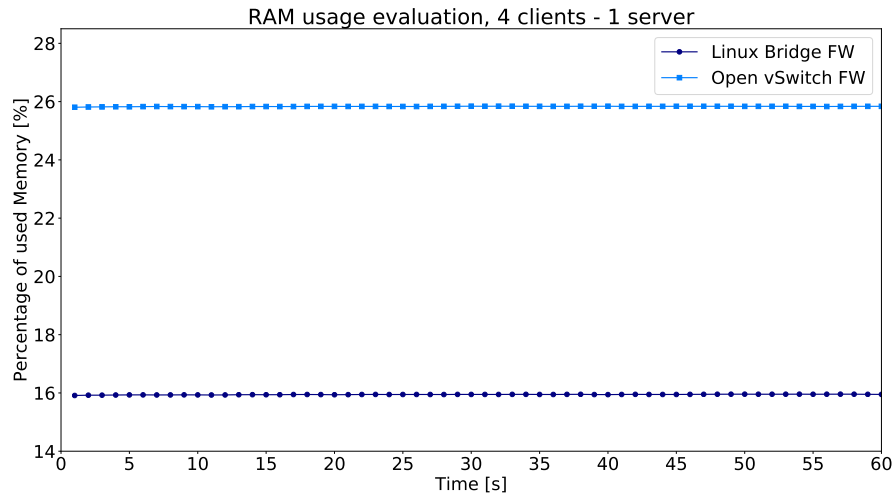


Figure 8.11: Memory usage during the TCP test with 4 clients and 1 server

With respect to the memory, it is possible to see that the results are usually stable and do not change at runtime. It is however interesting to notice in Figures 8.9, 8.10 and 8.11 that the OvS approach is consuming the double of the memory with respect to the LB one: this is probably related to the usage of multiple tables by the OvS FW with respect to the LB one. On the other hand, it is important to notice that the increase in the memory usage whenever the number of clients is larger for nodes running with a LB FW mechanism. Therefore, it is expected that in the case of a large number of instances, the OvS approach is still reasonable in terms of memory, as it will not introduce such an overhead as the LB mechanism does.

### 8.3.4 UDP Packet rate sustainability

Another interesting test that has been performed is the one related to the packet rate sustainability. This test employs the use of the RUDE & CRUDE [88] tool and therefore of UDP packets. The size of the packet has been chosen as 64 B or 1500 B at the network layer, which are the minimum and maximum size for UDP packets (actually, the UDP generated packet is 1472 B, that has to be added to 8 B for the UDP header and 20 B for the IP header). These tests are performed by considering a server where the crude processes

are started and a number of rude clients (or in general, of flows) on the same node, running in the clients virtual instances. This number is different for each test and will lead to a complete comprehension of the networking limitations the cloud platform as well as the chosen firewall have.

Figures 8.12, 8.13, 8.14, 8.15 represent the cases with 1 flow and 2 flows for 64B and 1500B. As it is possible to see, for all the generated traffic there corresponds an equal traffic received. Therefore it is possible to say that the system behaves properly.



Figure 8.12: Packet rate sustainability, 1 internal flow - 64 B UDP packets

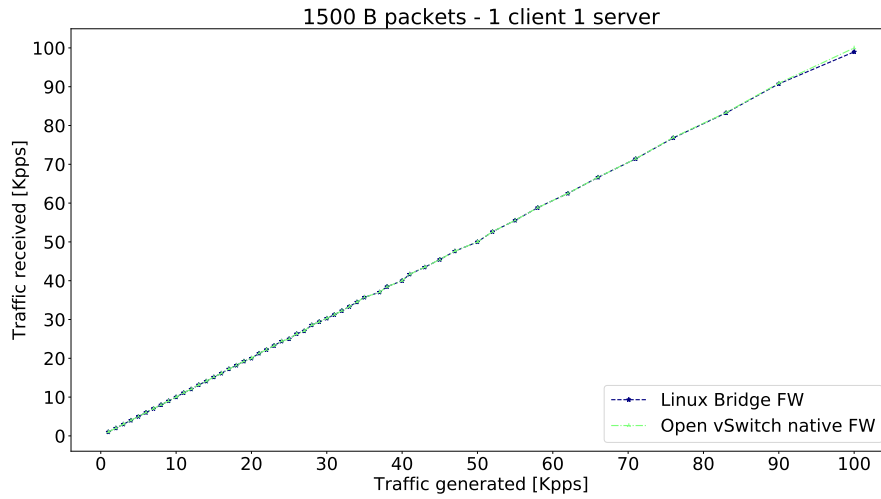


Figure 8.13: Packet rate sustainability, 1 internal flow - 1500 B UDP packets

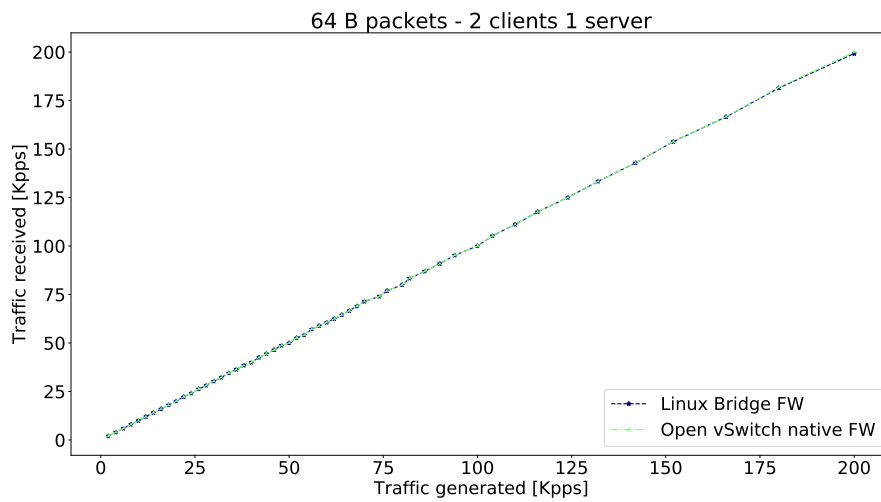


Figure 8.14: Packet rate sustainability, 2 internal flows - 64 B UDP packets

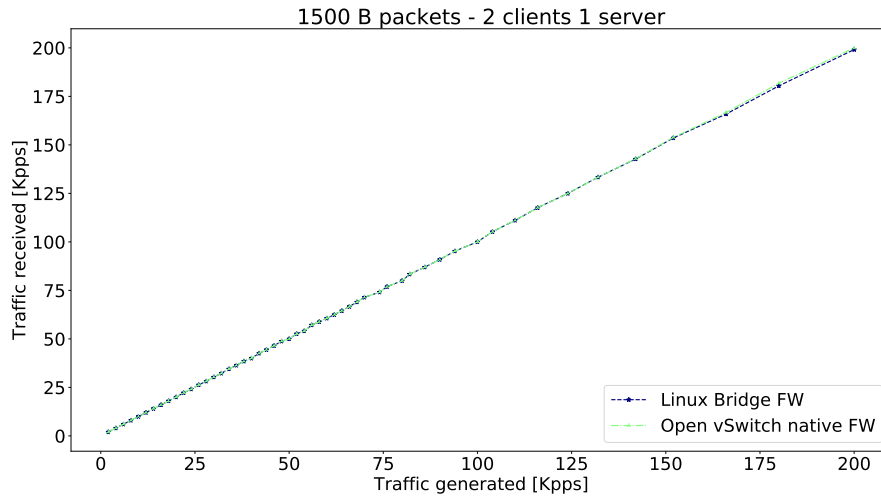


Figure 8.15: Packet rate sustainability, 2 internal flows - 1500 B UDP packets

Similarly, Figure 8.16 represents the case of 4 contemporaneously flows and is still not showing anything remarkable, whereas in Figure 8.17 it is possible to see some performance degradation, even though it is not possible to state anything as it regards just a point. However, in Figure 8.18 it is shown that 8 flows are able to saturate, in both systems, the capacity to handle a quantity of traffic higher than 400000 packets/s.

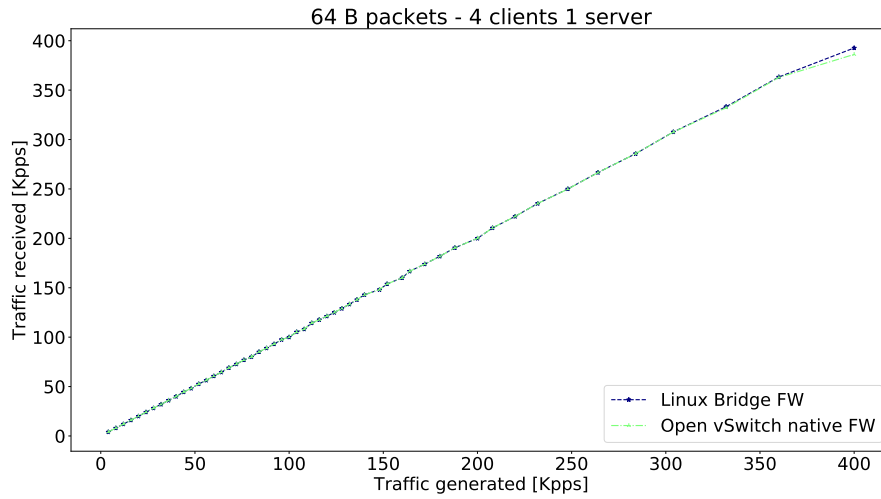


Figure 8.16: Packet rate sustainability, 4 internal flows - 64 B UDP packets

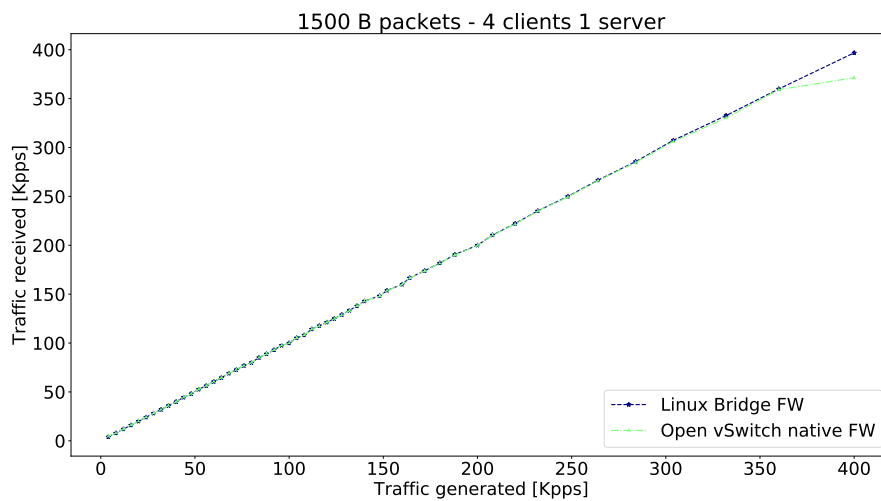


Figure 8.17: Packet rate sustainability, 4 internal flows - 1500 B UDP packets

Finally, in Figure 8.19 it is remarkable the difference of behavior among the two different FW mechanisms. In particular, it is important to see that

the LB-based node is able to handle at maximum 375000 packets/s before reaching a saturation phase, whereas the OvS-based one is able to achieve up to 460000 packets/s. In general, these results are higher than those achieved in [65] as a result of software improvements of the OpenStack platform and its internals, and of the use of the new integrated SDN approach. It is still important to outline that the used servers are still off-the-shelf and thought for general purpose.

However, this result shows that the performance of an OvS-based FW is remarkably higher with respect to the one of a LB-based one.

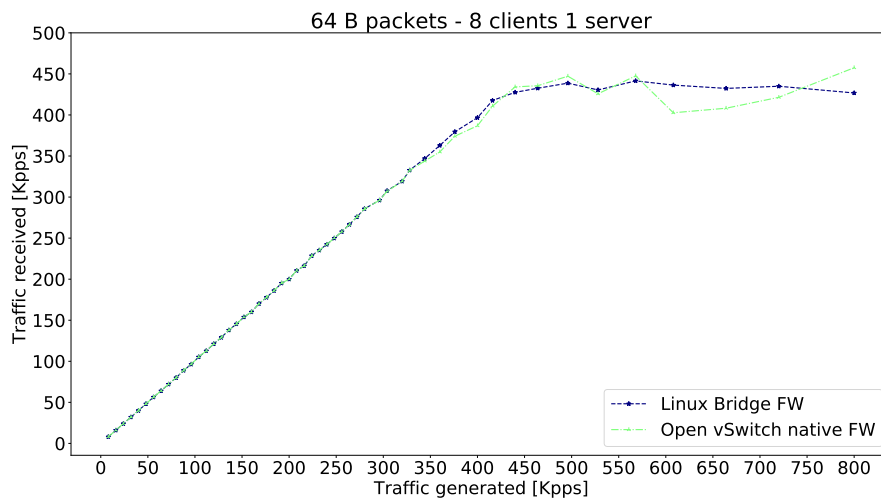


Figure 8.18: Packet rate sustainability, 8 internal flows - 64 B UDP packets

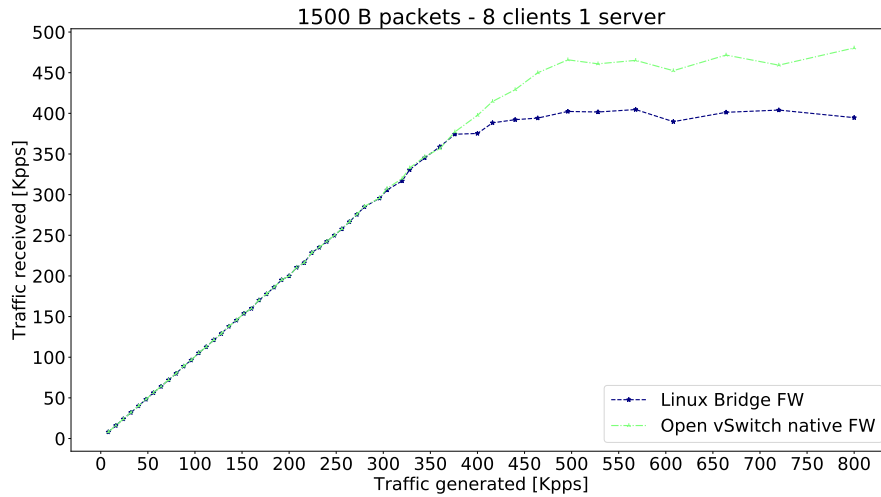


Figure 8.19: Packet rate sustainability, 8 internal flow - 1500 B UDP packets

## 8.4 Externally located instances measurements

Other tests have also been performed, by using a remote externally located client. This client resides in the physical node which is directly connected to the OpenStack cluster via the external network. Therefore, this analysis involves also the traversing of the network node, which in this testbed corresponds also to the controller node.

### 8.4.1 UDP Packet rate sustainability

Similarly to the previous case, the evaluation has been performed by considering a server instance in each compute node. However, here the client is remotely placed on the external network.

As it is possible to see from Figure 8.20 the system mostly behaves properly. On the other hand, Figure 8.21 and 8.22 are showing that a limit is reached at approximately 100000 packets/s.

Moreover, Figure 8.23, 8.24 and 8.25 are showing that the system is not able to handle traffic greater than 65000 packets/s. This means that network throughputs higher than  $65000 \text{ pps} * (1500 * 8) \text{ b} = 780 \text{ Mbps}$  are not handled by



the platform. This is probably caused by the presence of an additional node and its constraints (even the network node indeed has some firewall rules), therefore the theoretical limit of 1 Gbps is not reached. Moreover, the behavior of the system when the traffic generated is increased is interesting as it is not stable to the same saturation value, but it decreases down to another value. This however would need further studies as it is out of the scope of this document. In both cases, there is not such a difference among the different FW approaches, probably due to the same reason here considered.

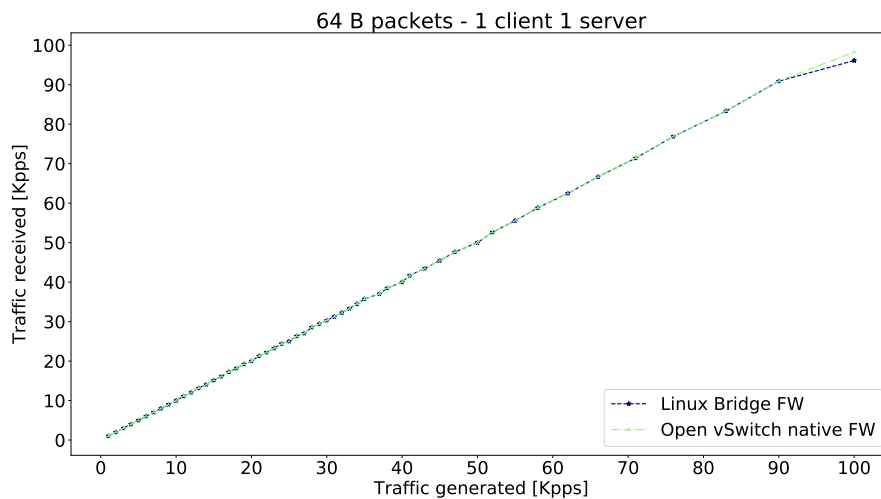


Figure 8.20: Packet rate sustainability, 1 external flow - 64 B UDP packets

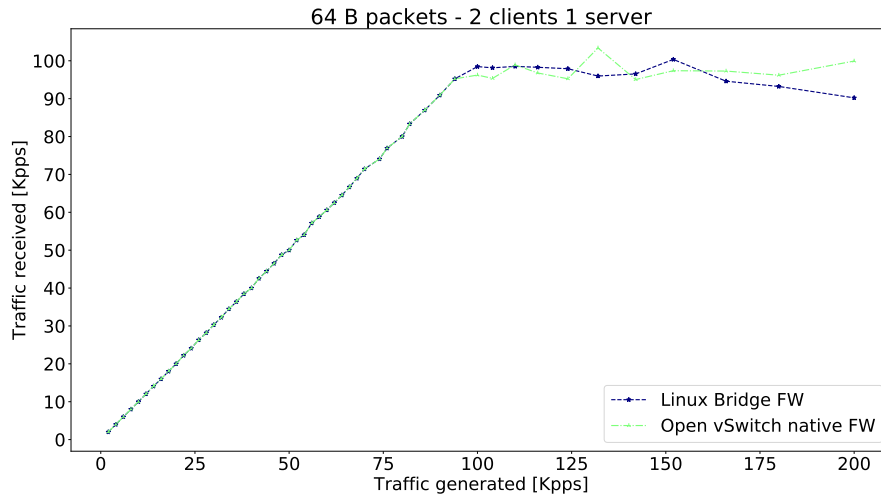


Figure 8.21: Packet rate sustainability, 2 external flows - 64 B UDP packets

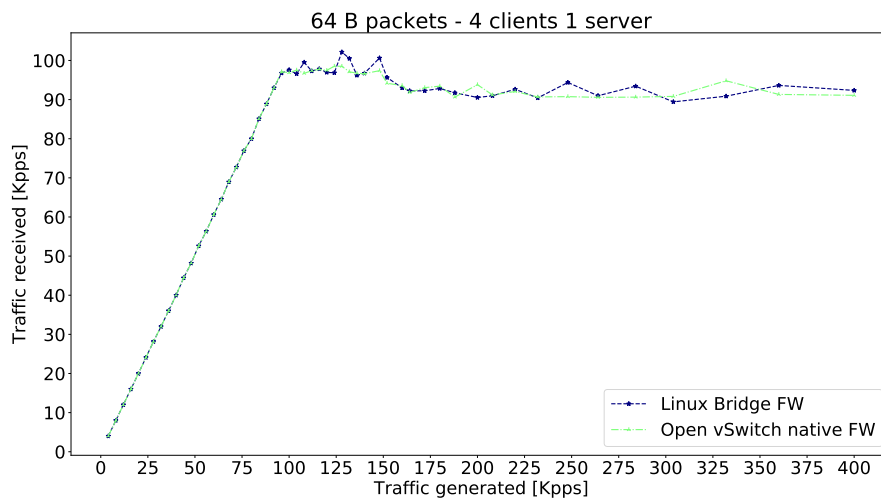


Figure 8.22: Packet rate sustainability, 4 external flows - 64 B UDP packets

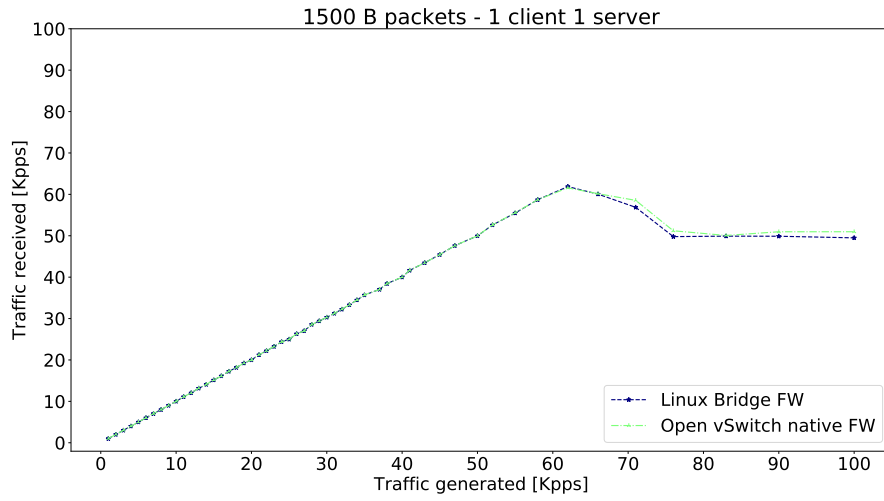


Figure 8.23: Packet rate sustainability, 1 external flow - 1500 B UDP packets

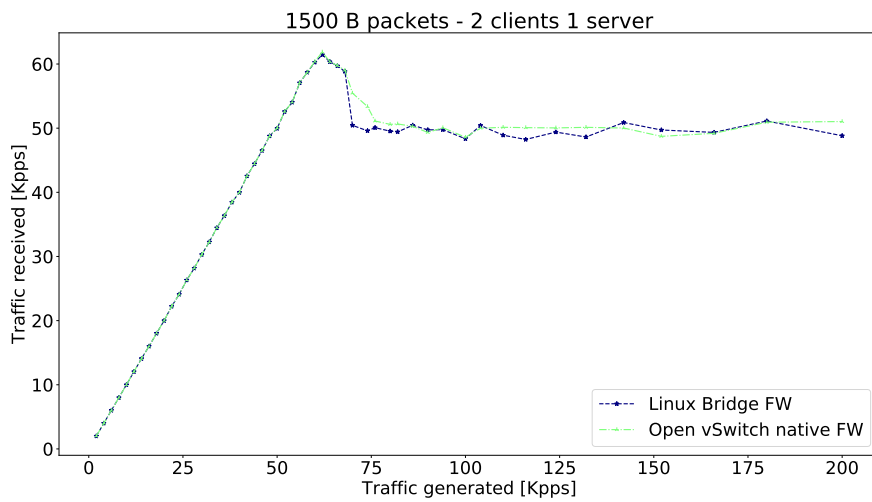


Figure 8.24: Packet rate sustainability, 2 external flows - 1500 B UDP packets

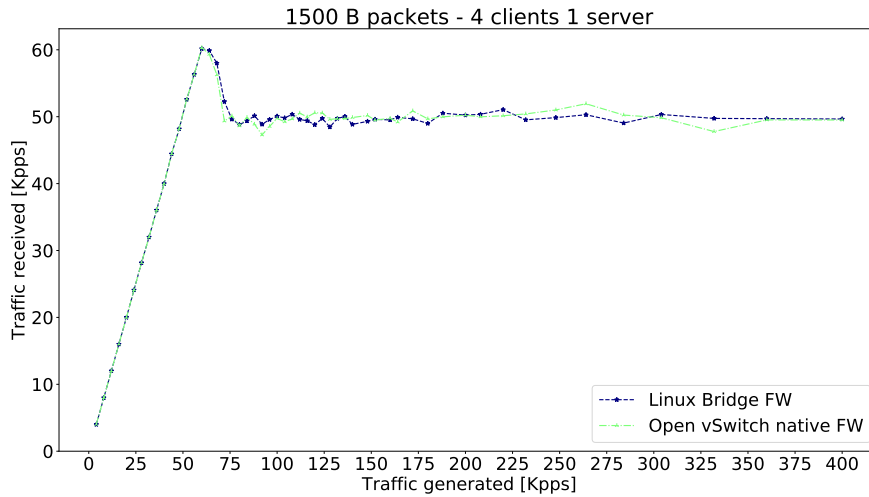


Figure 8.25: Packet rate sustainability, 4 external flows - 1500 B UDP packets

## 8.5 Final remarks

It is therefore remarkable that the trade-off when using the two FW approaches (LB and OvS) is mostly, at the moment of writing, between Telecommunications KPIs (Throughput, latency, etc.) and Computer Science KPIs (CPU, memory, etc.). When choosing an OvS FW it is possible to state that the CPU and the memory are more used than what a LB FW would. However, this results in an improved performance whenever it comes to throughput and latency. Therefore, this approach is very interesting and, by an optimization of software, in particular of the OvS component itself, it would be possible to achieve even more interesting results.

# Chapter 9

## Conclusions

This document has considered the state of the art in terms of networking, where new paradigms as Software Defined Networking, Network Functions Virtualization and Cloud Computing are entering the market. These paradigms are interesting for multiple actors, from the service providers themselves, as well as the infrastructure providers as they are considered the possible enablers to deploy innovative services.

An overview of a well-known Open Source cloud platform, named OpenStack, has been provided. In particular, its networking aspects have been shown. The main focus has been the integration among this platform and Open Source SDN frameworks, as ONOS (SONA) and the native Ryu framework solution. In addition to it, different firewalling mechanisms have been configured for the OpenStack cluster, as previous studies remarked that they were the main bottleneck of the cloud platform.

The experimental results have shown that the integration among SDN and Cloud is leading to interesting results, both from a qualitative point of view (delivering the possibility to introduce more network intelligence, e.g. performing Service Function Chaining, orchestration of resources, etc.) and a quantitative point of view. Indeed, by properly configuring the cluster and by following an integrated approach with a SDN controller, some performance evaluations are provided. These have shown that the Open vSwitch native Firewall achieves a better performance, as a result of its disruptive way of considering the firewalling with respect to the Linux Bridge Firewall. This is achieved at the cost of a higher CPU and memory utilization that, however, is remarkable only when few instances are employed.

Therefore, it is possible to state that the SDN-cloud integration approach, followed by an outstanding network tuning of the cloud platform, is leading to significant results. However, this document is also intended to underline that additional work is requested, especially on the cloud internals, to improve even more the performance and be compliant with the strict requirements that the future standards will demand.

# Appendix A

## Additional remarks

### A.1 SONA configuration files

#### A.1.1 SONA network configuration

```
{
  "apps" : {
    "org.onosproject.openstackinterface" : {
      "openstackinterface" : {
        "neutronServer" : "http://10.134.231.28:9696/v2.0/"
        "keystoneServer" : "http://10.134.231.28:5000/v2.0/"
        "userName" : "admin"
        "password" : "nova"
      }
    },
    "org.onosproject.openstacknode" : {
      "openstacknode" : {
        "nodes" : [
          {
            "hostname" : "compute-01",
            "type" : "COMPUTE",
            "managementIp" : "10.134.231.30",
            "dataIp" : "10.134.34.222",
            "integrationBridge" : "of:0000000000000000a1"
          }
        ]
      }
    }
  }
}
```

```

    {
        'hostname' : 'gateway-01',
        'type' : 'GATEWAY',
        'managementIp' :
            '10.134.231.32',
        'dataIp' : '10.134.34.224',
        'integrationBridge' : 'of
            :00000000000000a2',
        'routerBridge' : 'of
            :00000000000000b1',
        'uplinkPort' : 'quagga-router
            ',
        'routerController' :
            '172.17.0.2'
    }
    ]
}
};
'devices' : {
    'of:00000000000000a1' : {
        'basic' : {
            'driver' : 'sona'
        }
    },
    'of:00000000000000a2' : {
        'basic' : {
            'driver' : 'ovs'
        }
    }
}
}
}

```

### A.1.2 SONA cell file

```

export OCI=10.134.231.29
export OC1=10.134.231.29
export ONOS_APPS=drivers , openflow-base , openstackswitching ,
    openstackrouting
export ONOS_GROUP=sdn
export ONOS_SCENARIOS=/home/developer/onos/tools/test/scenarios
export ONOS_TOPO=default
export ONOS_USER=sdn
export ONOS_WEB_PASS=rocks
export ONOS_WEB_USER=onos

```



### A.1.3 ONOS ML2 configuration

```
# Configuration options for ONOS ML2 Mechanism driver
[onos]
# (StrOpt) ONOS ReST interface URL. This is a mandatory field.
url_path = http://10.134.231.29:8181/onos/openstacknetworking
# (StrOpt) Username for authentication. This is a mandatory field.
username = onos
# (StrOpt) Password for authentication. This is a mandatory field.
password = rocks
```

### A.1.4 DevStack configuration for the testbed

#### Controller node

```
[[ local | localrc ]]
HOST_IP=10.134.231.28
SERVICE_HOST=10.134.231.28
RABBIT_HOST=10.134.231.28
DATABASE_HOST=10.134.231.28
Q_HOST=10.134.231.28

ADMIN_PASSWORD=HiddenOpenStackPassword
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD

DATABASE_TYPE=mysql

# Log
SCREEN_LOGDIR=/opt/stack/logs/screen

# Images
IMAGE_URLS='http://cloud-images.ubuntu.com/releases/14.04/release
/ubunt-14.04-server-cloudimg-amd64.tar.gz,http://www.planet-
lab.org/cord/trusty-server-multi-nic.img'
FORCE_CONFIG_DRIVE=True

# Networks
Q_ML2_TENANT_NETWORK_TYPE=vxlan
Q_ML2_PLUGIN_MECHANISM_DRIVERS=onos_ml2
Q_PLUGIN_EXTRA_CONF_PATH=/opt/stack/networking-onos/etc
Q_PLUGIN_EXTRA_CONF_FILES=(conf-onos.ini)
```

```
ML2_L3_PLUGIN=networking_onos.plugins.l3.driver.ONOSL3Plugin
NEUTRON.CREATE_INITIAL_NETWORKS=False
```

```
# Services
```

```
enable_service q-svc
enable_service h-eng h-api h-api-cfn h-api-cw
disable_service n-net
disable_service n-cpu
disable_service tempest
disable_service c-sch
disable_service c-api
disable_service c-vol
```

```
# Branches
```

```
GLANCE_BRANCH=stable/mitaka
HORIZON_BRANCH=stable/mitaka
KEYSTONE_BRANCH=stable/mitaka
NEUTRON_BRANCH=stable/mitaka
NOVA_BRANCH=stable/mitaka
HEAT_BRANCH=stable/mitaka
```

### Compute node

```
[[ local | localrc ]]
HOST_IP=10.134.231.30
SERVICE_HOST=10.134.231.28
RABBIT_HOST=10.134.231.28
DATABASE_HOST=10.134.231.28

ADMIN_PASSWORD=HiddenOpenStackPassword
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD

DATABASE_TYPE=mysql

NOVA_VNC_ENABLED=True
VNC_SERVER_PROXYCLIENT_ADDRESS=$HOST_IP
VNC_SERVER_LISTEN=$HOST_IP

LIBVIRT_TYPE=kvm
# Log
SCREEN_LOGDIR=/opt/stack/logs/screen
```

```
# Services
ENABLED_SERVICES=n-cpu,neutron
```

```
# Branches
NOVA_BRANCH=stable/mitaka
KEYSTONE_BRANCH=stable/mitaka
NEUTRON_BRANCH=stable/mitaka
```

### A.1.5 SONA vRouter configuration

```
{
  "devices" : {
    "of:000000000000000b1" : {
      "basic" : {
        "driver" : "softrouter"
      }
    }
  },
  "apps" : {
    "org.onosproject.router" : {
      "router" : {
        "controlPlaneConnectPoint" : "of:000000000000000b1/2",
        "ospfEnabled" : "true",
        "interfaces" : [ "b1-1", "b1-2" ]
      }
    }
  },
  "ports" : {
    "of:000000000000000b1/3" : {
      "interfaces" : [
        {
          "name" : "b1-1",
          "ips" : [ "172.18.0.254/24" ],
          "mac" : "fe:00:00:00:00:01"
        }
      ]
    }
  },
  "of:000000000000000b1/1" : {
    "interfaces" : [
      {
        "name" : "b1-2",
        "ips" : [ "172.27.0.254/24" ],
        "mac" : "fe:00:00:00:00:01"
      }
    ]
  }
}
```

```

    ]
  }
},
“hosts” : {
  “fe:00:00:00:00:02/-1” : {
    “basic” : {
      “ips” : [“172.27.0.1”],
      “location” : “of:000000000000000b1/1”
    }
  }
}
}

```

### A.1.6 Docker BGP configuration

Configuration 3: volumes/gateway/bgpd.conf

```

! -- bgp --
!
! BGPd sample configuration file
!
!
hostname gateway-01
password zebra
!
router bgp 65101
  bgp router-id 172.18.0.254
  timers bgp 3 9
  neighbor 172.18.0.1 remote-as 65100
  neighbor 172.18.0.1 ebgp-multihop
  neighbor 172.18.0.1 timers connect 5
  neighbor 172.18.0.1 advertisement-interval 5
  network 172.27.0.0/24
!
log file /var/log/quagga/bgpd.log

```

**Configuration 4: volumes/router/bgpd.conf**

```
! -- bgp --
!
! BGPd sample configuration file
!
!
hostname router-01
password zebra
!
router bgp 65100
  bgp router-id 172.18.0.1
  timers bgp 3 9
  neighbor 172.18.0.254 remote-as 65101
  neighbor 172.18.0.254 ebgp-multihop
  neighbor 172.18.0.254 timers connect 5
  neighbor 172.18.0.254 advertisement-interval 5
  neighbor 172.18.0.254 default-originate
!
log file /var/log/quagga/bgpd.log
```

**Configuration 5: volumes/gateway/zebra.conf & volumes/router/zebra.conf**

```
!
hostname gateway-01
password zebra
!
fpm connection ip 172.17.0.2 port 2620
```

## A.2 SONA monitor code

### A.2.1 Monitor

```
import logging
import flows
import hosts
import stats
import pprint
```

```

logging.basicConfig(level=logging.DEBUG)

def hosts_handler():
    host_loop = True
    print 'I am the handler 1'
    while host_loop:

        print 'What are you interested to see?'
        interest = raw_input('1) Show all the hosts \n2) Show the
            hosts related to a particular device \n3) Show the '
            'hosts related to a particular
            network \n4) Show the hosts
            related to a particular '
            'tenant\n')

        if interest == '1':
            hosts.show_all_hosts()

        elif interest == '2':
            list_dev = hosts.show_dev()
            pp = pprint.PrettyPrinter(indent=2)
            print 'Which one?'
            dev_loop = True
            while dev_loop:
                pp.pprint(list_dev)
                device = raw_input('(Please copy and paste to
                    avoid errors)\n')
                if device not in list_dev:
                    print 'Not found, try again'
                else:
                    dev_loop = False
            hosts.hosts_per_dev(device)

        elif interest == '3':
            list_net = hosts.show_net()
            pp = pprint.PrettyPrinter(indent=2)
            print 'Which one?'
            net_loop = True
            while (net_loop):
                pp.pprint(list_net)
                net = raw_input('(Please copy and paste to avoid
                    errors)\n')
                if net not in list_net:

```

```

        print ‘‘Not found, try again’’
    else:
        net_loop = False
        hosts.hosts_per_net(net)

elif interest == ‘‘4’’:
    list_tenant = hosts.show_tenant()
    pp = pprint.PrettyPrinter(indent=2)
    print ‘‘Which one?’’
    tenant_loop = True
    while (tenant_loop):
        pp.pprint(list_tenant)
        tenant = raw_input(‘‘(Please copy and paste to
            avoid errors)\n’’)
        if tenant not in list_tenant:
            print ‘‘Not found, try again’’
        else:
            tenant_loop = False
            hosts.hosts_per_tenant(tenant)

else:
    print ‘‘Come on, you are not funny’’
    host_loop = False

def flows_handler():
    print ‘‘I am the handler 2’’
    flow_loop = True
    while (flow_loop):

        print ‘‘What are you interested to see?’’
        interest = raw_input(‘‘1) Show all the flows \n2) Show the
            flows related to a particular device \n3) Show the ’’
            ‘‘flows related to a particular app \
                n’’)

        if interest == ‘‘1’’:
            flows.print_all_flows()

        elif interest == ‘‘2’’:
            list_dev = flows.dev_flows()
            pp = pprint.PrettyPrinter(indent=2)
            print ‘‘Which one?’’
            dev_loop = True
            while (dev_loop):

```

```

        pp.pprint(list_dev)
        device = raw_input('(Please copy and paste to
            avoid errors)\n')
        if device not in list_dev:
            print 'Not found, try again'
        else:
            dev_loop = False
            flows.flows_per_dev(device)

elif interest == '3':
    list_app = flows.which_flows()
    pp = pprint.PrettyPrinter(indent=2)
    print 'Which one?'
    app_loop = True
    while app_loop:
        pp.pprint(list_app)
        app = raw_input('(Please copy and paste to avoid
            errors)\n')
        if app not in list_app:
            print 'Not found, try again'
        else:
            app_loop = False
            flows.flows_per_app(app)

else:
    print 'Come on, you are not funny'

flow_loop = False

def stats_handler():
    global port
    print 'I am the handler 3'
    stats_loop = True
    while stats_loop:

        print 'What are you interested to see?'
        interest = raw_input('1) Show all the statistics \n2)
            Show the ports related to a particular device \n'
            '3) Show the statistics related to a
            particular port of a device \n')

    if interest == '1':
        stats.show_all_stats()

```



```

elif interest == '2':
    list_dev = stats.show_devices()
    pp = pprint.PrettyPrinter(indent=2)
    print 'Which one?'
    stats_in_loop = True
    while stats_in_loop:
        pp.pprint(list_dev)
        device = raw_input('(Please copy and paste to
            avoid errors)\n')
        if device not in list_dev:
            print 'Not found, try again'
        else:
            stats_in_loop = False
    stats.ports_per_device(device)

elif interest == '3':
    list_dev = stats.show_devices()
    pp = pprint.PrettyPrinter(indent=2)
    print 'Which one?'
    stats_in_loop = True
    while stats_in_loop:
        pp.pprint(list_dev)
        device = raw_input('(Please copy and paste to
            avoid errors)\n')
        if device not in list_dev:
            print 'Not found, try again'
        else:
            print 'Which port?'
            port_loop = True
            list_port = stats.ports_per_device(device)
            while port_loop:
                pp.pprint(list_port)
                port = raw_input()
                port = int(port)
                if port not in list_port:
                    print 'Not found, try again'
                else:
                    port_loop = False
            stats_in_loop = False

    stats.stats_per_port(device, port)

else:
    print 'Come on, you are not funny'

```

```

        stats_loop = False

print ‘‘Hi and welcome! This is a simple script showing you the
    possibility to exploit ONOS API to monitor OpenStack’’

print ‘‘First of all: what would you like to monitor?’’

loop = True

while loop:

    choice = raw_input(‘‘Hosts, flows or stats?\n’’)
    in_choice = str(choice).lower()
    print ‘‘Your choice is: ’’ + str(in_choice)
    # logging.debug(in_choice)
    # logging.debug(type(choice))
    # logging.debug(type(in_choice))

    if in_choice == ‘‘hosts’’:
        hosts_handler()
    elif in_choice == ‘‘flows’’:
        flows_handler()
    elif in_choice == ‘‘stats’’:
        stats_handler()
    else:
        print ‘‘Incorrect choice’’

    interest = raw_input(‘‘Are you interested into looking to
        something else? [y/n]’’)
    interest.lower()
    if interest == ‘‘n’’ or interest == ‘‘no’’:
        loop = False
    elif interest == ‘‘y’’ or interest == ‘‘yes’’:
        print ‘‘Ok, about what?’’
    else:
        print ‘‘Very funny, I am closing...’’
        loop = False

```

### A.2.2 Statistics subclass

```

import logging
import json
import pprint
import os

```

```

logging.basicConfig(level=logging.DEBUG)

def send_the_curl():
    os.system('mkdir -p /tmp/json')
    os.system('curl -u karaf:karaf -X GET --header 'Accept:
              application/json' 'http://10.134.231.29:8181/onos/v1/
              statistics/ports > /tmp/json/statisticsport.json')

    with open('/tmp/json/statisticsport.json', 'r') as
        stats_file:
            stats = json.load(stats_file)
    return stats

def show_devices():
    stats = send_the_curl()

    list_dev = []
    for j in stats['statistics']:
        if j['device'] not in list_dev:
            list_dev.append(j['device'])
    logging.debug('There are ' + str(len(list_dev)) + ' devices
                  : ' + str(list_dev))
    return list_dev

def ports_per_device(device):
    stats = send_the_curl()

    list_dev = show_devices()

    if device not in list_dev:
        print 'The device you have requested is not present'
        print 'The available devices are: ' + list_dev
    else:
        for j in stats['statistics']:
            if j['device'] == device:
                ports = []
                for k in j['ports']:
                    ports.append(k['port'])

```

```

        print ‘‘The ports of device ’’ + device + ’’ are
              ’’ + str(len(ports)) + ’’: ’’ + str(ports)
    return ports

def stats_per_port(device, port):

    stats = send_the_curl()

    ports = ports_per_device(device)

    if port not in ports:
        print ‘‘The asked port is not present, try again’’
    else:
        for j in stats[‘‘statistics’’]:
            if j[‘‘device’’] == device:
                for k in j[‘‘ports’’]:
                    if k[‘‘port’’] == port:
                        bytes_received = k[‘‘bytesReceived’’]
                        break

        print ‘‘The received bytes on the port ’’ + str(port) + ’’
              are: ’’ + str(bytes_received)

def show_all_stats():

    stats = send_the_curl()

    pp = pprint.PrettyPrinter(indent=2)
    pp.pprint(stats[‘‘statistics’’])

```

### A.2.3 Hosts subclass

```

import logging
import json
import pprint
import os

logging.basicConfig(level=logging.DEBUG)

def send_the_curl():

    os.system(‘‘mkdir -p /tmp/json’’)

```

```

os.system('curl -u karaf:karaf -X GET --header 'Accept:
application/json' 'http://10.134.231.29:8181/onos/v1/hosts'
> /tmp/json/hosts.json')

with open('/tmp/json/hosts.json', 'r') as host_file:
    hosts = json.load(host_file)
return hosts

def num_hosts():

    hosts = send_the_curl()

    print "There are " + str(len(hosts['hosts'])) + " hosts"

def show_net():

    hosts = send_the_curl()

    list_net = []
    for i in hosts['hosts']:
        if i['annotations']['networkId'] not in list_net:
            list_net.append(i['annotations']['networkId'])
    return list_net

def hosts_per_net(net_id):

    hosts = send_the_curl()

    machines = []
    for i in hosts['hosts']:
        if i['annotations']['networkId'] == net_id:
            machines.append(i)

    if not machines:
        print "There is no host related to network id " + str(
            net_id)
        list_net = show_net()
        print "The available network ids are: " + str(list_net)
    else:
        print "There are " + str(len(machines)) + " hosts
            related to network id " + net_id
        print machines

```

```

def hosts_per_tenant(tenant_id):

    hosts = send_the_curl()

    machines = []
    for i in hosts['hosts']:
        if i['annotations']['tenantId'] == tenant_id:
            machines.append(i)

    if not machines:
        print "There is no host related to tenant id " + str(
            tenant_id)
        list_tenants = show_tenant()
        print "This is a list of available tenants: " + str(
            list_tenants)
    else:
        print "There are " + str(len(machines)) + " hosts
            related to tenant id " + tenant_id
        print machines

def show_dev():

    hosts = send_the_curl()

    list_dev = []
    for i in hosts['hosts']:
        if i['location']['elementId'] not in list_dev:
            list_dev.append(i['location']['elementId'])
    logging.debug("The list of devices: " + str(list_dev))
    return list_dev

def show_tenant():

    hosts = send_the_curl()

    list_tenant = []
    for i in hosts['hosts']:
        if i['annotations']['tenantId'] not in list_tenant:
            list_tenant.append(i['annotations']['tenantId'])
    logging.debug("The list of tenants: " + str(list_tenant))
    return list_tenant

```

```

def hosts_per_dev(device):

    hosts = send_the_curl()

    list_dev = show_dev()

    if device not in list_dev:
        print 'The requested device ' + device + ' is not
            present '
        print 'This is a list of available devices: ' + str(
            list_dev)
    else:
        machines = []
        for i in hosts['hosts']:
            if i['location']['elementId'] == device:
                machines.append(i)
        print 'The hosts attached to device ' + device + ' are:
            ' + str(machines)

def show_all_hosts():

    hosts = send_the_curl()

    pp = pprint.PrettyPrinter(indent=2)
    pp.pprint(hosts['hosts'])

```

#### A.2.4 Flows subclass

```

import logging
import json
import pprint
import os

logging.basicConfig(level=logging.DEBUG)

def send_the_curl():

    os.system('mkdir -p /tmp/json')
    os.system('curl -u karaf:karaf -X GET --header 'Accept:
        application/json' 'http://10.134.231.29:8181/onos/v1/flows'
        > /tmp/json/flows.json')

```

```

with open('json/flows.json', 'r') as flow_file:
    flows = json.load(flow_file)
return flows

def flow_num():

    flows = send_the_curl()

    elem = len(flows['flows'])
    logging.debug('There are ' + str(elem) + ' flows in the
        OpenStack cluster')
    return elem

def which_flows():

    flows = send_the_curl()
    elem = flow_num()

    # check the app name
    list_app = []
    for i in range(0, elem, 1):
        if flows['flows'][i]['appId'] not in list_app:
            list_app.append(flows['flows'][i]['appId'])
    return list_app

def flows_per_app(name_app):

    count = 0

    flows = send_the_curl()
    elem = flow_num()
    list_app = which_flows()

    if name_app not in list_app:
        print 'The app you requested (' + name_app + ') is not
            present.'
        print 'This is the list of available apps: ' + str(
            list_app)
        return 1

    for i in range(0, elem, 1):
        if flows['flows'][i]['appId'] == name_app:

```



```

        count += 1
    print 'The app ' + name_app + ' has generated ' + str(
        count) + ' flows'

def dev_flows():

    flows = send_the_curl()

    list_dev = []
    for j in flows['flows']:
        if j['deviceId'] not in list_dev:
            list_dev.append(j['deviceId'])
    logging.debug('There are ' + str(len(list_dev)) + ' devices
        on which flows are currently stored: ' + str(list_dev))
    return list_dev

def flows_per_dev(device):

    flows = send_the_curl()
    list_dev = dev_flows()

    if device not in list_dev:
        print 'The device you have requested is not present'
        print 'The available devices are: ' + list_dev
    else:
        list_flows_dev = []
        for i in flows['flows']:
            if i['deviceId'] == device:
                list_flows_dev.append(i)
        print 'There are ' + str(len(list_flows_dev)) + ' flows
            in the device ' + str(device)
        print list_flows_dev

def print_all_flows():

    flows = send_the_curl()

    pp = pprint.PrettyPrinter(indent=2)
    pp.pprint(flows['flows'])

```

### A.3 YAML template for a Telegram bot

```
heat_template_version: 2013-05-23

description: This template deploys a single Telegram bot.

parameters:
  image:
    type: string
    label: Image name or ID
    description: Image to be used for the server. Please use an
      Ubuntu based image.
    default: trusty-server-multi-nic
  flavor:
    type: string
    label: Flavor
    description: Type of instance (flavor) to be used on the
      compute instance.
    default: m1.small
  key:
    type: string
    label: Key name
    description: Name of key-pair to be installed on the compute
      instance.
    default: mykey-heat
  public_network:
    type: string
    label: Public network name or ID
    description: Public network with floating IP addresses.
    default: net-public
  security_group:
    type: string
    label: Security groups
    description: Chosen security group
    default: allow-external
  root_pw:
    type: string
    label: Root PW
    description: Root password
    default: ubuntu

resources:

  private_network:
```

```
    type: OS::Neutron::Net

private_subnet:
  type: OS::Neutron::Subnet
  properties:
    network_id: { get_resource: private_network }
    cidr: 10.10.10.0/24
    dns_nameservers:
      - 137.204.57.1
      - 137.204.58.10
      - 8.8.8.8

router:
  type: OS::Neutron::Router
  properties:
    external_gateway_info:
      network: { get_param: public_network }

router-interface:
  type: OS::Neutron::RouterInterface
  properties:
    router_id: { get_resource: router }
    subnet: { get_resource: private_subnet }

bot_port:
  type: OS::Neutron::Port
  properties:
    network: { get_resource: private_network }
    security_groups:
      - { get_param: security_group }

wait_condition:
  type: OS::Heat::WaitCondition
  properties:
    handle: { get_resource: wait_handle }
    count: 1
    timeout: 4000

wait_handle:
  type: OS::Heat::WaitConditionHandle

bot_instance:
  type: OS::Nova::Server
  properties:
    image: { get_param: image }
```

```

flavor: { get_param: flavor }
key_name: { get_param: key }
networks:
  - port: { get_resource: bot_port }
user_data_format: RAW
user_data:
  str_replace:
    params:
      '@ROOTPW@': {get_param: root_pw}
      wc_notify: { get_attr: ['wait_handle', 'curl_cli'] }
    template: |
      #!/bin/sh -ex
      echo 'Hello, World!'

      ip link set eth0 mtu 1400

      ping -c 10 8.8.8.8
      ping -c 10 www.google.com

      ROOTPW='@ROOTPW@'
      echo 'ubuntu:$ROOTPW' | chpasswd

      wget https://dl.dropboxusercontent.com/s/
        HIDDEN_CONTENT/bot.txt # repositories

      mv /etc/apt/sources.list /etc/apt/sources.list.old
      mv s.txt /etc/apt/sources.list

      apt-get update -y
      apt-get install software-properties-common -y
      apt-get install python-pip -y

      pip install telepot
      wget https://dl.dropboxusercontent.com/s/
        HIDDEN_CONTENT/bot.py # bot code

      wc_notify --data-binary '{"status": "SUCCESS"}'

      python bot.py &

```

```

floating_ip:
  type: OS::Neutron::FloatingIP
  properties:
    floating_network: { get_param: public_network }

```

```

floating_ip_assoc:
  type: OS::Neutron::FloatingIPAssociation
  properties:
    floatingip_id: { get_resource: floating_ip }
    port_id: { get_resource: bot_port }

outputs:
  instance_name:
    description: Name of the instance
    value: { get_attr: [bot_instance, name] }
  instance_ip:
    description: The IP address of the deployed instance
    value: { get_attr: [floating_ip, floating_ip_address] }

```

## A.4 OpenStack configuration files

### A.4.1 Chrony

#### Controller node

```

# /etc/chrony.conf
server time.ien.it iburst
server ntp1.ien.it iburst
server ntp2.ien.it iburst
stratumweight 0
driftfile /var/lib/chrony/drift
rtcsync
makestep 10 3
allow 10.15.0.0/24
bindcmdaddress 127.0.0.1
bindcmdaddress ::1
keyfile /etc/chrony.keys
commandkey 1
generatecommandkey
noclientlog
logchange 0.5
logdir /var/log/chrony

```

#### Compute node

```

# /etc/chrony.conf
server controllerNew iburst
stratumweight 0
driftfile /var/lib/chrony/drift

```

```

rtcsync
makestep 10 3
bindcmdaddress 127.0.0.1
bindcmdaddress ::1
keyfile /etc/chrony.keys
commandkey 1
generatecommandkey
noclientlog
logchange 0.5
logdir /var/log/chrony

```

### A.4.2 MariaDB configuration

```

/etc/my.cnf.d/openstack.cnf
[mysqld]
bind-address = 10.15.0.5

default-storage-engine = innodb
innodb_file_per_table
max_connections = 4096
collation-server = utf8_general_ci
character-set-server = utf8

```

### A.4.3 Memcached

```

# /etc/sysconfig/memcached
PORT='11211'
USER='memcached'
MAXCONN='1024'
CACHE_SIZE='64'
OPTIONS='-l 10.15.0.5'

```

### A.4.4 Keystone

```

# /etc/keystone/keystone.conf
[DEFAULT]
[database]
connection = mysql+pymysql://keystone:
    keystonepassword@controllerNew/keystone
[token]
provider = fernet

```

### A.4.5 HTTP server

```

# /etc/httpd/conf/httpd.conf
ServerRoot "/etc/httpd"
Listen 80
Include conf.modules.d/*.conf
User apache
Group apache
ServerAdmin root@localhost
ServerName controllerNew
<Directory />
    AllowOverride none
    Require all denied
</Directory>
DocumentRoot "/var/www/html"
<Directory "/var/www">
    AllowOverride None
    Require all granted
</Directory>
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
<Files ".ht*">
    Require all denied
</Files>
ErrorLog "logs/error_log"
LogLevel warn
<IfModule log_config_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" " combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
<IfModule logio_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio
</IfModule>
    CustomLog "logs/access_log" combined
</IfModule>
<IfModule alias_module>
    ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
</IfModule>
<Directory "/var/www/cgi-bin">
    AllowOverride None

```

```

    Options None
    Require all granted
</Directory>
<IfModule mime_module>
    TypesConfig /etc/mime.types
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
    AddType text/html .shtml
    AddOutputFilter INCLUDES .shtml
</IfModule>
AddDefaultCharset UTF-8
<IfModule mime_magic_module>
    MIMEMagicFile conf/magic
</IfModule>
EnableSendfile on
IncludeOptional conf.d/*.conf

```

## A.4.6 Glance

### Glance-API

```

# /etc/glance/glance-api.conf
[DEFAULT]
rpc_backend = rabbit
[database]
connection = mysql+pymysql://glance:glancepassword@controllerNew/
    glance
[glance_store]
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
[image_format]
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = glancepassword
[oslo_messaging_notifications]
driver = messagingv2
[oslo_messaging_rabbit]

```



```

rabbit_host = controllerNew
rabbit_userid = openstack
rabbit_password = rabbitpassword
[paste_deploy]
flavor = keystone

# /etc/glance/glance-registry.conf
[DEFAULT]
rpc_backend = rabbit
[database]
connection = mysql+pymysql://glance:glancepassword@controllerNew/
    glance
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = glancepassword
[oslo_messaging_notifications]
driver = messagingv2
[oslo_messaging_rabbit]
rabbit_host = controllerNew
rabbit_userid = openstack
rabbit_password = rabbitpassword
[paste_deploy]
flavor = keystone

```

### A.4.7 Nova

#### Controller node

```

# /etc/nova/nova.conf
[DEFAULT]
enabled_apis = osapi_compute,metadata
transport_url = rabbit://openstack:rabbitpassword@controllerNew
auth_strategy = keystone
my_ip = 10.15.0.5
use_neutron = True
firewall_driver = nova.virt.firewall.NoopFirewallDriver
linuxnet_interface_driver = nova.network.linux_net.
    LinuxOVSIfaceDriver

```

```

vif_plugging_is_fatal=False
vif_plugging_timeout=10
firewall_driver=nova.virt.firewall.NoopFirewallDriver
[api_database]
connection = mysql+pymysql://nova:novapassword@controllerNew/
    nova_api
[database]
connection = mysql+pymysql://nova:novapassword@controllerNew/nova
[ephemeral_storage_encryption]
[glance]
api_servers = http://controllerNew:9292
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = novapassword
region_name=RegionOne
[neutron]
url=http://controllerNew:9696
region_name=RegionOne
service_metadata_proxy=True
metadata_proxy_shared_secret = sharedsecret
auth_type=password
auth_url=http://controllerNew:35357
project_name=service
project_domain_name=default
default_domain_name=default
username=neutron
user_domain_name=default
password=neutronpassword
[oslo_concurrency]
lock_path = /var/lib/nova/tmp
[vnc]
vncserver_listen = $my_ip
vncserver_proxyclient_address = $my_ip

```

### Compute node

```

# /etc/nova/nova.conf
[DEFAULT]

```

```
enabled_apis = osapi_compute,metadata
transport_url = rabbit://openstack:rabbitpassword@controllerNew
auth_strategy = keystone
my_ip = 10.15.0.6
use_neutron = True
firewall_driver = nova.virt.firewall.NoopFirewallDriver
instance_usage_audit_period=hour
instance_usage_audit=True
notify_on_state_change=vm_and_task_state
[glance]
api_servers = http://controllerNew:9292
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = novapassword
[neutron]
url=http://controllerNew:9696
region_name=RegionOne
auth_type=password
auth_url=http://controllerNew:35357
project_name=service
project_domain_name=default
default_domain_name=default
username=neutron
user_domain_name=default
password=neutronpassword
[oslo_concurrency]
lock_path = /var/lib/nova/tmp
[oslo_messaging_notifications]
driver = messagingv2
[vnc]
enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = $my_ip
novncproxy_base_url = http://firewallnat:6080/vnc_auto.html
```

## A.4.8 Neutron

### Controller node

```
# /etc/neutron/neutron.conf
[DEFAULT]
transport_url = rabbit://openstack:rabbitpassword@controllerNew
auth_strategy = keystone
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
core_plugin = ml2
state_path = /var/lib/neutron
service_plugins = router
dhcp_agent_notification = true
allow_overlapping_ips = True
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true
rpc_backend = rabbit
[agent]
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
[database]
connection = mysql+pymysql://neutron:neutronpassword@controllerNew/neutron
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = neutron
password = neutronpassword
[nova]
auth_url = http://controllerNew:35357
auth_type = password
project_domain_name = Default
user_domain_name = Default
region_name = RegionOne
project_name = service
username = nova
password = novapassword
[oslo_concurrency]
```

```
lock_path = /var/lib/neutron/tmp
[oslo_messaging_notifications]
driver = messagingv2
[oslo_messaging_rabbit]
rabbit_host = controllerNew
rabbit_port = 5672
rabbit_userid = openstack
rabbit_password = rabbitpassword

# /etc/neutron/plugins/ml2/ml2_conf.ini
[DEFAULT]
[ml2]
type_drivers = flat ,vlan ,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch ,l2population
extension_drivers = port_security
[ml2_type_flat]
flat_networks = public
[ml2_type_vxlan]
vni_ranges = 1:1000
[securitygroup]
enable_ipset = True
firewall_driver = iptables_hybrid
enable_security_group = true

# /etc/neutron/plugins/ml2/openvswitch_agent.ini
[DEFAULT]
[agent]
tunnel_types = vxlan
l2_population = True
[ovs]
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = 10.125.0.5
bridge_mappings = public:br-ex
of_interface = ovs-ofctl
[securitygroup]
firewall_driver = iptables_hybrid
enable_ipset = true

# /etc/neutron/l3_agent.ini
[DEFAULT]
interface_driver = neutron.agent.linux.interface.
    OVSInterfaceDriver
external_network_bridge =
```

```
[AGENT]
```

```
# /etc/neutron/dhcp_agent.ini
[DEFAULT]
interface_driver = neutron.agent.linux.interface.
    OVSIInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = True
force_metadata = True
[AGENT]
```

```
# /etc/neutron/metadata_agent.ini
[DEFAULT]
nova_metadata_ip = controllerNew
nova_metadata_port = 8775
metadata_proxy_shared_secret = sharedsecret
[AGENT]
```

## Compute node 6

```
# /etc/neutron/plugins/ml2/ml2_conf.ini
[DEFAULT]
auth_strategy = keystone
transport_url = rabbit://openstack:rabbitpassword@controllerNew
[agent]
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = neutron
password = neutronpassword
[oslo_concurrency]
lock_path = /var/lib/neutron/tmp
```

```
# /etc/neutron/plugins/ml2/openvswitch_agent.ini
[DEFAULT]
[agent]
tunnel_types = vxlan
l2_population = True
[ovs]
integration_bridge = br-int
```

```
tunnel_bridge = br-tun
local_ip = 10.125.0.6
[securitygroup]
firewall_driver = iptables_hybrid
enable_security_group = true
enable_ipset = true
```

### Compute node 7

```
# /etc/neutron/plugins/ml2/ml2_conf.ini
[DEFAULT]
auth_strategy = keystone
transport_url = rabbit://openstack:rabbitpassword@controllerNew
[agent]
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = neutron
password = neutronpassword
[oslo_concurrency]
lock_path = /var/lib/neutron/tmp

# /etc/neutron/plugins/ml2/openvswitch_agent.ini
[DEFAULT]
[agent]
tunnel_types = vxlan
l2_population = True
[ovs]
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = 10.125.0.6
[securitygroup]
firewall_driver = openvswitch
enable_security_group = true
enable_ipset = true
```

### A.4.9 Load Ryu applications from Neutron

```
# /usr/lib/python2.7/site-packages/neutron/plugins/ml2/drivers/
  openvswitch/agent/openflow/native/main.py
```

```

from oslo_config import cfg
from ryu.base import app_manager
from ryu import cfg as ryu_cfg

cfg.CONF.import_group(
    'OVS',
    'neutron.plugins.ml2.drivers.openvswitch.agent.common.config')

def init_config():
    ryu_cfg.CONF(project='ryu', args=[])
    ryu_cfg.CONF.ofp_listen_host = cfg.CONF.OVS.of_listen_address
    ryu_cfg.CONF.ofp_tcp_listen_port = cfg.CONF.OVS.of_listen_port

def main():
    app_manager.AppManager.run_apps([
        'neutron.plugins.ml2.drivers.openvswitch.agent.',
        'openflow.native.ovs_ryuapp',
        'ryu.app.ofctl_rest',

```

#### A.4.10 Dashboard

```

# /etc/openstack-dashboard/local_settings
import os
from django.utils.translation import ugettext_lazy as _
from openstack_dashboard import exceptions
from openstack_dashboard.settings import HORIZON_CONFIG
DEBUG = False
WEBROOT = '/dashboard/'
ALLOWED_HOSTS = ['*', ]
OPENSTACK_API_VERSIONS = {
    'data-processing': 1.1,
    'identity': 3,
    'image': 2,
    'volume': 2,
    'compute': 2,
}
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = 'default'
LOCALPATH = '/tmp'
SECRET_KEY='cce32bc18cb36720a18e'
CACHES = {

```



```

    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.
            MemcachedCache',
        'LOCATION': 'controllerNew:11211',
    },
}
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
OPENSTACK_HOST = 'controllerNew'
OPENSTACK_KEYSTONE_URL = 'http://%s:5000/v3' % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = 'user'
OPENSTACK_KEYSTONE_BACKEND = {
    'name': 'native',
    'can_edit_user': True,
    'can_edit_group': True,
    'can_edit_project': True,
    'can_edit_domain': True,
    'can_edit_role': True,
}
OPENSTACK_HYPERVERSOR_FEATURES = {
    'can_set_mount_point': False,
    'can_set_password': False,
    'requires_keypair': False,
    'enable_quotas': True
}
OPENSTACK_CINDER_FEATURES = {
    'enable_backup': False,
}
OPENSTACK_NEUTRON_NETWORK = {
    'enable_router': True,
    'enable_quotas': True,
    'enable_ipv6': True,
    'enable_distributed_router': False,
    'enable_ha_router': False,
    'enable_lb': True,
    'enable_firewall': True,
    'enable_vpn': True,
    'enable_fip_topology_check': True,
    'profile_support': None,
    'supported_vnic_types': ['*'],
}
OPENSTACK_HEAT_STACK = {
    'enable_user_pass': True,
}
IMAGE_CUSTOM_PROPERTY_TITLES = {
    'architecture': _('Architecture'),

```

```

    'kernel_id': _('Kernel ID'),
    'ramdisk_id': _('Ramdisk ID'),
    'image_state': _('Euca2ools state'),
    'project_id': _('Project ID'),
    'image_type': _('Image Type'),
}
IMAGE_RESERVED_CUSTOM_PROPERTIES = []
API_RESULT_LIMIT = 1000
API_RESULT_PAGE_SIZE = 20
SWIFT_FILE_TRANSFER_CHUNK_SIZE = 512 * 1024
INSTANCE_LOG_LENGTH = 35
DROPDOWN_MAX_ITEMS = 30
TIME_ZONE = 'Europe/Rome'
POLICY_FILES_PATH = '/etc/openstack-dashboard'
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'operation': {
            'format': '%(asctime)s %(message)s'
        },
    },
    'handlers': {
        'null': {
            'level': 'DEBUG',
            'class': 'logging.NullHandler',
        },
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
        },
        'operation': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'operation',
        },
    },
    'loggers': {
        'django.db.backends': {
            'handlers': ['null'],
            'propagate': False,
        },
        'requests': {
            'handlers': ['null'],
            'propagate': False,
        },
    },
}

```

```
    },
    'horizon ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'horizon.operation_log ': {
        'handlers ': [ 'operation ' ],
        'level ': 'INFO',
        'propagate ': False ,
    },
    'openstack_dashboard ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'novaclient ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'cinderclient ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'keystoneclient ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'glanceclient ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'neutronclient ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
    'heatclient ': {
        'handlers ': [ 'console ' ],
        'level ': 'DEBUG',
        'propagate ': False ,
    },
```

```

    },
    'ceilometerclient': {
        'handlers': ['console'],
        'level': 'DEBUG',
        'propagate': False,
    },
    'swiftclient': {
        'handlers': ['console'],
        'level': 'DEBUG',
        'propagate': False,
    },
    'openstack_auth': {
        'handlers': ['console'],
        'level': 'DEBUG',
        'propagate': False,
    },
    'nose.plugins.manager': {
        'handlers': ['console'],
        'level': 'DEBUG',
        'propagate': False,
    },
    'django': {
        'handlers': ['console'],
        'level': 'DEBUG',
        'propagate': False,
    },
    'iso8601': {
        'handlers': ['null'],
        'propagate': False,
    },
    'scss': {
        'handlers': ['null'],
        'propagate': False,
    },
},
}
SECURITY_GROUP_RULES = {
    'all_tcp': {
        'name': _('All TCP'),
        'ip_protocol': 'tcp',
        'from_port': '1',
        'to_port': '65535',
    },
    'all_udp': {
        'name': _('All UDP'),

```

```
        'ip_protocol': 'udp',
        'from_port': '1',
        'to_port': '65535',
    },
    'all_icmp': {
        'name': _('All ICMP'),
        'ip_protocol': 'icmp',
        'from_port': '-1',
        'to_port': '-1',
    },
    'ssh': {
        'name': 'SSH',
        'ip_protocol': 'tcp',
        'from_port': '22',
        'to_port': '22',
    },
    'smtp': {
        'name': 'SMTP',
        'ip_protocol': 'tcp',
        'from_port': '25',
        'to_port': '25',
    },
    'dns': {
        'name': 'DNS',
        'ip_protocol': 'tcp',
        'from_port': '53',
        'to_port': '53',
    },
    'http': {
        'name': 'HTTP',
        'ip_protocol': 'tcp',
        'from_port': '80',
        'to_port': '80',
    },
    'pop3': {
        'name': 'POP3',
        'ip_protocol': 'tcp',
        'from_port': '110',
        'to_port': '110',
    },
    'imap': {
        'name': 'IMAP',
        'ip_protocol': 'tcp',
        'from_port': '143',
        'to_port': '143',
    },
```

```
},
'ldap': {
  'name': 'LDAP',
  'ip_protocol': 'tcp',
  'from_port': '389',
  'to_port': '389',
},
'https': {
  'name': 'HTTPS',
  'ip_protocol': 'tcp',
  'from_port': '443',
  'to_port': '443',
},
'smtps': {
  'name': 'SMTPS',
  'ip_protocol': 'tcp',
  'from_port': '465',
  'to_port': '465',
},
'imaps': {
  'name': 'IMAPS',
  'ip_protocol': 'tcp',
  'from_port': '993',
  'to_port': '993',
},
'pop3s': {
  'name': 'POP3S',
  'ip_protocol': 'tcp',
  'from_port': '995',
  'to_port': '995',
},
'ms_sql': {
  'name': 'MS SQL',
  'ip_protocol': 'tcp',
  'from_port': '1433',
  'to_port': '1433',
},
'mysql': {
  'name': 'MYSQL',
  'ip_protocol': 'tcp',
  'from_port': '3306',
  'to_port': '3306',
},
'rdp': {
  'name': 'RDP',
```

```

        'ip_protocol': 'tcp',
        'from_port': '3389',
        'to_port': '3389',
    },
}
REST_API_REQUIRED_SETTINGS = ['OPENSTACK_HYPERVISOR_FEATURES',
                              'LAUNCH_INSTANCE_DEFAULTS',
                              'OPENSTACK_IMAGE_FORMATS']
ALLOWED_PRIVATE_SUBNET_CIDR = {'ipv4': [], 'ipv6': []}

```

### A.4.11 Heat

```

# /etc/heat/heat.conf
[DEFAULT]
heat_metadata_server_url = http://controllerNew:8000
heat_waitcondition_server_url = http://controllerNew:8000/v1/
    waitcondition
stack_user_domain_name = heat
stack_domain_admin = heat_domain_admin
stack_domain_admin_password = heatpassword
rpc_backend = rabbit
[clients_keystone]
auth_uri = http://controllerNew:35357
[database]
connection = mysql+pymysql://heat:heatpassword@controllerNew/heat
[ec2auth_token]
auth_uri = http://controllerNew:5000
[oslo_messaging_notifications]
driver = messagingv2
[oslo_messaging_rabbit]
rabbit_host = controllerNew
rabbit_userid = openstack
rabbit_password = rabbitpassword
[trustee]
auth_type = password
auth_url = http://controllerNew:35357
username = heat
user_domain_name = default
password = heatpassword
[keystone_auth_token]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = default

```

```

user_domain_name = default
project_name = service
username = heat
password = heatpassword

```

### A.4.12 Ceilometer

```

# /etc/ceilometer/ceilometer.conf
[DEFAULT]
auth_strategy = keystone
rpc_backend = rabbit
[keystone_authtoken]
auth_uri = http://controllerNew:5000
auth_url = http://controllerNew:35357
memcached_servers = controllerNew:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = ceilometer
password = ceilometerpassword
[oslo_messaging_rabbit]
rabbit_host = controllerNew
rabbit_userid = openstack
rabbit_password = rabbitpassword
[service_credentials]
auth_url = http://controller:5000
project_domain_id = default
user_domain_id = default
auth_type = password
username = ceilometer
project_name = service
password = ceilometerpassword
interface = internalURL
region_name = RegionOne

# /etc/httpd/wsgi-ceilometer.conf
Listen 8777
<VirtualHost *:8777>
    WSGIDaemonProcess ceilometer-api processes=2 threads=10 user=
        ceilometer group=ceilometer display-name=%{GROUP}
    WSGIProcessGroup ceilometer-api
    WSGIScriptAlias / ‘‘/var/www/cgi-bin/ceilometer/app’’
    WSGIApplicationGroup %{GLOBAL}
    ErrorLog /var/log/httpd/ceilometer_error.log

```



```
    CustomLog /var/log/httpd/ceilometer_access.log combined
</VirtualHost>
```

```
WSGISocketPrefix /var/run/httpd
```



# Acknowledgments

Reaching this goal means much to me. However, this has been possible as a result of the presence of wonderful people around me.

First of all, I would like to thank the Net2Lab research group I have been part of for these years. I have met admirable people, that brought me competences, friendship, values and life experiences. It is not easy to say how much grateful I am and will always be with Walter and Franco, for their support during these years: their door was always open for me for an advice, a help or a simple chat. I do not know if all advisors are like them, but I am sure to have found those who really fit me. Moreover, my gratitude goes also to Carla Raffaelli, Flavio Esposito, Antonio Corradi and Luca Foschini, for their availability and for what they have transmitted me.

Of course, our research Lab means also students, actually, friends. It is therefore important to thank Chiara “tap”, my support for years and mate of incredible adventures on conferences, as well as Andrea “The Senior”, one of the funniest and nicest guy ever met, the real soul of the Lab. It is obviously straightforward that Federico has a big role here, for his truly friendship, the talks about everything, the shared beers. Finally, Gianluca too has a huge part in these acknowledgments for his continuous support, starting from the first week of Bologna Engineering School up to the help with data when time was running out. The competences that he now teaches me are what make me say that I have done something good in my time there. Finally, thanks Bahare for all the funny moments, like every morning arrival in the Lab, with the fairy tales about how much my train was delayed or the continuous AC turn on/off.

I would like to thank my classmates Gianluca R., Michele M., Pasquale, Karol, Giacomo, Marco, Alberto, Michele P., Lorenzo. It has been fun to share these two years of classes with you, with strange English accents and particular people. I would also like to thank Silvia M., Vincenzo, Serena, Chiara G. and

Giovanni for the moments together, inside or outside the faculty. A special mention goes to Betty, a girl with a big heart that I did not expect that I would have ever met. I am very happy that my path has been shared with a true friend like her. Moreover, I am in her Spotify family, therefore no additional words are needed.

Regarding the research community, it is mandatory to thank two wonderful Sicilian people (even though, Catania vs. Palermo...) with which it has always been a pleasure to talk and share some time together, among one open source project failure, fairy adventure of packets and “this software is running faster than hardware” (including our contemporary “what the heck is he saying?” faces when listening). Thank you “Maestro” Giuseppe, thank you “(hONOrary) ambaSSador” Silvia. In the research field it is usually easier to meet people, but I am pleased to have established a nice friendship with you.

I would also like to thank Giulia and Elena, who I have had the pleasure to meet just a moment before departing to China together to work in Huawei. Among all the strange things, the different habits, the medicine pills, the “yi, er, san”, the “Shenzhen” metro announcement, we have had a great time together. You are the part of China that I am missing.

A whole chapter should be dedicated to Federica, the girl with which I have been happy to share my days from 5 years on. Your presence, your silence when needed, your voice when unneeded: I would love to thank you for everything you did during this time. Among every ups and downs, there is something more that keeps our hands tight together. Therefore, thank you for the support, for the time, for the patience. You are a pure drop of water in a polluted world, thank you for being like this, thank you for being “my drop”. I love you so much.

I would love to conclude with a very complete paragraph for my family, but words might not come as wanted. First of all, I would like to thank my grandparents for the support, the presence and the love they have provided to me. From them, indirectly or directly, I have learned many lessons that made me stronger and vulnerable at the same time. Maybe the open wounds will heal one day, but I am going on as a result of your love, regardless where you are now. Thank you for everything you did and do for me, Francesco and Antonio, Dorotea and Immacolata.

A special mention to my sister Fabiana and her husband Matteo is of course important to be mentioned for all the time together since ever, the hospitality, the pieces of cakes that sometimes are able to reach home to be tasted. In

addition to it, in depth, thank you, from the bottom of the heart, for all the affection that you feel for me.

Finally, it is important to include my mother and my father, incredible parents who made one sacrifice over the other to let me and my sister grow well, reach our satisfaction and be able to take off (reasonable) whims. Everything that you did for me, for us, is too much to even try to coarsely be put on a list. But this thesis is for you, it is a small piece of joy for me and a result of all your sacrifices. I hope that this acknowledgment will let you see that this is the result of every single car ride to the station, of every public mean of transport subscription, of every day-after lunch prepared at 11 pm, of all the distance from the place where your heart belonged. But also of all those days-at-work with dad and its stamps, of the first computer, of the first dial-up 64K Internet connection with the marvelous orange modem and its sounds, of all the floppy disks with unlicensed games. After everything that you have taught to me in this years, you seamlessly have helped me in finding my way one step after the other. And I really wanted to dedicate you this document, as a reminder of those few long-term benefits of the sacrifices that you did. These benefits are not always in our sight, but they still somehow exist and I am happy to provide you one of these.

Thank you, everyone, for your support, the time, the friendship. I am ready to have my future: how will it be?

Faenza, 09/30/17  
Francesco



# Bibliography

- [1] 3GPP. *3GPP system standards heading into the 5G era*. [http://www.3gpp.org/news-events/3gpp-news/1614-sa\\_5g](http://www.3gpp.org/news-events/3gpp-news/1614-sa_5g). (Accessed on 09/24/2017).
- [2] 3GPP. *5G service requirements*. [http://www.3gpp.org/news-events/3gpp-news/1831-sa1\\_5g](http://www.3gpp.org/news-events/3gpp-news/1831-sa1_5g). (Accessed on 09/24/2017).
- [3] *Intent NBI Definition and Principles*. [https://www.opennetworking.org/wp-content/uploads/2014/10/TR-523\\_Intent\\_Definition\\_Principles.pdf](https://www.opennetworking.org/wp-content/uploads/2014/10/TR-523_Intent_Definition_Principles.pdf). (Accessed on 09/24/2017).
- [4] Chiara Contoli. *Virtualized Network Infrastructures: Performance Analysis, Design and Implementation*. 2017. URL: <http://amsdottorato.unibo.it/8100/>.
- [5] *SOA Reference Architecture Introduction*. [http://www.opengroup.org/soa/source-book/soa\\_refarch/p1.htm](http://www.opengroup.org/soa/source-book/soa_refarch/p1.htm). (Accessed on 09/24/2017).
- [6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Bolton Landing, NY, USA: ACM, 2003, pp. 29–43. ISBN: 1-58113-757-5. DOI: 10.1145/945445.945450. URL: <http://doi.acm.org/10.1145/945445.945450>.
- [7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. “The Hadoop Distributed File System”. In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 2010, pp. 1–10. DOI: 10.1109/MSST.2010.5496972.
- [8] Avinash Lakshman and Prashant Malik. “Cassandra: A Decentralized Structured Storage System”. In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), pp. 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: <http://doi.acm.org/10.1145/1773912.1773922>.

- [9] E. Brewer. *Towards Robust Distributed Systems*. [http://awoc.wolski.fi/dlib/big-data/Brewer\\_podc\\_keynote\\_2000.pdf](http://awoc.wolski.fi/dlib/big-data/Brewer_podc_keynote_2000.pdf). (Accessed on 09/24/2017). July 2000.
- [10] *MongoDB for GIANT Ideas — MongoDB*. <https://www.mongodb.com/>. (Accessed on 09/24/2017).
- [11] Dirk Merkel. “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. In: *Linux J*. 2014.239 (Mar. 2014). ISSN: 1075-3583. URL: <http://dl.acm.org/citation.cfm?id=2600239.2600241>.
- [12] *Jolie Programming Language*. <http://www.jolie-lang.org/>. (Accessed on 09/24/2017).
- [13] Open Networking Foundation. In: *ONF White Paper* (Apr. 2012).
- [14] F. Hu, Q. Hao, and K. Bao. “A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation”. In: *IEEE Communications Surveys Tutorials* 16.4 (2014), pp. 2181–2206. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2326417.
- [15] William Stallings. *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional, 2016.
- [16] *SDN Architecture Overview*. Tech. Rec. The Open Networking Foundation (ONF), 2013. URL: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf>.
- [17] R Enns. *RFC 4741: NETCONF Configuration Protocol, 2006*.
- [18] R Enns, M Bjorklund, J Schoenwaelder, and A Bierman. “NETCONF Configuration Protocol (IETF RFC 6241)”. In: (2011).
- [19] M Bjorklund. *Rfc 6020: Yang-a data modeling language for the network configuration protocol*. 2010.
- [20] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. “P4: Programming protocol-independent packet processors”. In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.



- [21] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.
- [22] S. Seetharaman. *OpenFlow/SDN tutorial*. OFC/NFOEC. 2012. URL: <http://www.slideshare.net/openflow/openflow-tutorial>.
- [23] *OpenFlow*. URL: <https://www.opennetworking.org/sdn-resources/openflow>.
- [24] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. “Network Function Virtualization: State-of-the-Art and Research Challenges”. In: *IEEE Communications Surveys and Tutorials, Vol 18, No. 1* (2016).
- [25] Hao F., Kodialam M., Lakshman T. V., and Mukherjee. “Online allocation of virtual machines in a distributed cloud”. In: *IEEE/ACM Transactions on Networking* 25(1) (2017).
- [26] *Network Functions Virtualisation (NFV); Architectural Framework*. The European Telecommunications Standards Institute (ETSI). 2013. URL: <http://www.etsi.org/technologies-clusters/technologies/nfv>.
- [27] *Network Functions Virtualisation (NFV); Management and Orchestration*. The European Telecommunications Standards Institute (ETSI), 2014. URL: <http://www.etsi.org/technologies-clusters/technologies/nfv>.
- [28] Foschini L. Corradi A. *Middleware and Cloud models*. <http://lia.deis.unibo.it/Courses/CompNetworksM/1617/slides/MiddlewareCloudx2.pdf>. (Accessed on 09/24/2017).
- [29] Q. Hassan. “Demistifying Cloud Computing”. In: *The Journal of Defense Software Engineering* (2011).
- [30] Francesco Foresta. *Composizione dinamica di funzioni di rete virtuali in ambienti cloud*. URL: <http://amslaurea.unibo.it/8381/>.
- [31] *CRM Software & Cloud Computing Solutions - Salesforce*. <https://www.salesforce.com/>. (Accessed on 09/24/2017).
- [32] *OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes*. <https://www.openshift.com/>. (Accessed on 09/24/2017).

- [33] *Microsoft Azure: piattaforma e servizi di cloud computing*. <https://azure.microsoft.com/it-it/>. (Accessed on 09/24/2017).
- [34] *Amazon Web Services (AWS) Servizi di cloud computing*. <https://aws.amazon.com/it/>. (Accessed on 09/24/2017).
- [35] *Piattaforma cloud: infrastruttura cloud - IBM Bluemix*. <https://www.ibm.com/cloud-computing/bluemix/it>. (Accessed on 09/24/2017).
- [36] *Home - OpenStack is open source software for creating private and public clouds*. <https://www.openstack.org/>. (Accessed on 09/24/2017).
- [37] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. “Unreeling netflix: Understanding and improving multi-cdn movie delivery”. In: *INFOCOM, 2012 Proceedings IEEE*. IEEE. 2012, pp. 1620–1628.
- [38] Gunnar Kreitz and Fredrik Niemela. “Spotify—large scale, low latency, P2P music-on-demand streaming”. In: *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*. IEEE. 2010, pp. 1–10.
- [39] *Spotify Case Study Amazon Web Services (AWS)*. <https://aws.amazon.com/it/solutions/case-studies/spotify/>. (Accessed on 09/24/2017).
- [40] *Announcing Spotify Infrastructures Googley Future — News*. <https://news.spotify.com/us/2016/02/23/announcing-spotify-infrastructures-googley-future/>. (Accessed on 09/24/2017).
- [41] DGS ETSI. *NFV-EVE 005. Network Function Virtualization (NFV) Ecosystem: Report on SDN Usage in NFV Architectural Framework*. 2015.
- [42] Franco Callegati, Walter Cerroni, Chiara Contoli, and Francesco Foresta. “Performance of Intent-based Virtualized Network Infrastructure Management”. In: *IEEE International Conference on Communications (ICC) (2017)*.
- [43] Walter Cerroni, Chiara Buratti, Simone Cerboni, Gianluca Davoli, Chiara Contoli, Francesco Foresta, Franco Callegati, and Roberto Verdone. “Intent-based Management and Orchestration of Heterogeneous OpenFlow/IoT SDN Domains”. In: *IEEE Conference on Network Softwarization (Netsoft) (2017)*.
- [44] G. Davoli. “Intent-based approach to virtualized infrastructure management in SDN/NFV deployments”. 2017.

- [45] G. Davoli, W. Cerroni, C. Contoli, F. Foresta, and F. Callegati. “Implementation of Service Function Chaining Control Plane through OpenFlow”. In: *IEEE Conference on Network Function Virtualization and Software Defined Networks*. 2017.
- [46] *Foundation - OpenStack is open source software for creating private and public clouds*. <https://www.openstack.org/foundation/>. (Accessed on 09/24/2017).
- [47] Tsung-Cheng Jason. *OpenStack Framework Introduction*. <https://www.slideshare.net/jasonhoutw/openstack-introduction>. (Accessed on 09/24/2017).
- [48] *Understanding OpenStack*. <https://www.redhat.com/en/topics/openstack>. (Accessed on 09/24/2017).
- [49] Ken Pepple. *OpenStack Nova Architecture*. <http://ken.pepple.info/openstack/2011/04/22/openstack-nova-architecture/>. (Accessed on 09/24/2017).
- [50] *OpenStack Docs: Identity service overview*. <https://docs.openstack.org/newton/install-guide-rdo/common/get-started-identity.html>. (Accessed on 09/24/2017).
- [51] *OpenStack Docs: Glossary*. <https://docs.openstack.org/newton/install-guide-rdo/common/glossary.html>. (Accessed on 09/24/2017).
- [52] *Solinea - Open Infrastructure Experts*. <https://solinea.com/>. (Accessed on 09/24/2017).
- [53] *memcached - a distributed memory object caching system*. <https://memcached.org/>. (Accessed on 09/24/2017).
- [54] *Neutron/ML2 - OpenStack*. <https://wiki.openstack.org/wiki/Neutron/ML2>. (Accessed on 09/24/2017).
- [55] *OpenStack Releases: Ocata*. <https://releases.openstack.org/ocata/>. (Accessed on 09/24/2017).
- [56] iLearnStack. *Request Flow for Provisioning Instance in Openstack — iLearnStack*. <https://ilearnstack.com/2013/04/26/request-flow-for-provisioning-instance-in-openstack/>. (Accessed on 09/24/2017).

- [57] Red Hat Linux Enterprise. *Component Overview*. [https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux\\_OpenStack\\_Platform/6/html-single/Component\\_Overview/index.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/6/html-single/Component_Overview/index.html). (Accessed on 09/24/2017).
- [58] Foschini L. Corradi A. *OpenStack and more*. <http://lia.deis.unibo.it/Courses/CompNetworksM/1617/slides/Openstackx2.pdf>. (Accessed on 09/24/2017).
- [59] *ip-netns, process network namespace management*. <https://www.systutorials.com/docs/linux/man/8-ip-netns/>. (Accessed on 09/24/2017).
- [60] *Linux Bridge, Linux Foundation Wiki*. <https://wiki.linuxfoundation.org/networking/bridge>. (Accessed on 09/24/2017).
- [61] *Open vSwitch*. <http://openvswitch.org/>. (Accessed on 09/24/2017).
- [62] *OpenStack Docs: Open vSwitch mechanism driver*. <https://docs.openstack.org/ocata/networking-guide/deploy-ovs.html>. (Accessed on 09/24/2017).
- [63] *OpenStack Docs: Linux bridge: Self-service networks*. <https://docs.openstack.org/ocata/networking-guide/deploy-lb-selfservice.html>. (Accessed on 09/24/2017).
- [64] *Open vSwitch-OpenStack Networking Guide -current*. [http://docs.ocselected.org/openstack-manuals/kilo/networking-guide/content/under\\_the\\_hood\\_openvswitch.html](http://docs.ocselected.org/openstack-manuals/kilo/networking-guide/content/under_the_hood_openvswitch.html). (Accessed on 09/24/2017).
- [65] Callegati F., Cerroni W., Contoli C., and Santandrea G. "Performance of Network Virtualization in Cloud Computing Infrastructures: The OpenStack Case". In: *Proc. of 3rd IEEE International Conference on Cloud Networking (CloudNet 2014)*. Luxemburg, 2014.
- [66] Jakub Libosvar and Rodolfo Alonso. *Tired of iptables based security groups? Here's how to gain tremendous speed with Open vSwitch instead!* <https://www.openstack.org/assets/presentation-media/Austin-Summit-SG-firewall-Presentation-v2.3.pdf>. (Accessed on 09/24/2017).
- [67] *OpenStack Docs: Native Open vSwitch firewall driver*. <https://docs.openstack.org/ocata/networking-guide/config-ovsfdriver.html>. (Accessed on 09/24/2017).

- [68] *OpenStack Docs: Open vSwitch Firewall Driver*. [https://docs.openstack.org/neutron/latest/contributor/internals/openvswitch\\_firewall.html](https://docs.openstack.org/neutron/latest/contributor/internals/openvswitch_firewall.html). (Accessed on 09/24/2017).
- [69] *Implementing an OpenStack Security Group Firewall Driver Using OVS Learn Actions — Intel Software*. <https://software.intel.com/en-us/articles/implementing-an-openstack-security-group-firewall-driver-using-ovs-learn-actions>. (Accessed on 09/24/2017).
- [70] *DPDK*. <http://dpdk.org/>. (Accessed on 09/24/2017).
- [71] *Ryu SDN Framework*. <https://osrg.github.io/ryu/>. (Accessed on 09/24/2017).
- [72] *Ryu 4.17 documentation*. [https://ryu.readthedocs.io/en/latest/getting\\_started.html](https://ryu.readthedocs.io/en/latest/getting_started.html). (Accessed on 09/24/2017).
- [73] *Ryubook*. <https://osrg.github.io/ryu-book/en/Ryubook.pdf>. (Accessed on 09/24/2017).
- [74] Cheng Li. *Ryu Learning Guide*. <https://www.slideshare.net/ssuser6888ad/ryu-48823041>. (Accessed on 09/24/2017).
- [75] *Neutron/OFAgent/ComparisonWithOVS - OpenStack*. <https://wiki.openstack.org/wiki/Neutron/OFAgent/ComparisonWithOVS>. (Accessed on 09/24/2017).
- [76] *OpenDaylight*. <https://www.opendaylight.org/>. (Accessed on 09/24/2017).
- [77] *Apache Karaf*. <http://karaf.apache.org/>. (Accessed on 09/24/2017).
- [78] *What is an OpenDaylight Controller?* <https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/>. (Accessed on 09/24/2017).
- [79] *networking-odl*. <http://events.linuxfoundation.org/sites/events/files/slides/networking-odl.pdf>. (Accessed on 09/24/2017).
- [80] *ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out*. <http://onosproject.org/>. (Accessed on 09/24/2017).
- [81] *ONOS Wiki*. <https://wiki.onosproject.org/>. (Accessed on 09/24/2017).

- [82] *Open Networking Foundation and ON.Lab to Merge to Accelerate Adoption of SDN Open CORD*. <http://opencord.org/open-networking-foundation-and-on-lab-to-merge-to-accelerate-adoption-of-sdn/>. (Accessed on 09/24/2017).
- [83] *Heat Orchestration Template specifications*. Accessed: 2017-06-07. URL: [https://docs.openstack.org/developer/heat/template\\_guide/hot\\_spec.html](https://docs.openstack.org/developer/heat/template_guide/hot_spec.html).
- [84] *Telegram*. Accessed: 2017-06-07. URL: <https://telegram.org/>.
- [85] Francesco Foresta. *Is SONA discontinued?* <http://bit.ly/2htqDvQ>. (Accessed on 09/24/2017).
- [86] *The Netperf Homepage*. <https://hewlettpackard.github.io/netperf/>. (Accessed on 09/29/2017).
- [87] *sar (Unix) - Wikipedia*. [https://en.wikipedia.org/wiki/Sar\\_\(Unix\)](https://en.wikipedia.org/wiki/Sar_(Unix)). (Accessed on 09/29/2017).
- [88] *RUDE and CRUDE download — SourceForge.net*. <http://rude.sourceforge.net/>. (Accessed on 09/29/2017).

# List of Figures

2.1	Service Oriented Architecture . . . . .	7
2.2	Brewer’s CAP Theorem [9] . . . . .	8
2.3	Traditional vs. SDN networking [15] . . . . .	10
2.4	The role of the SDN controller in the SDN architecture [16] . . . . .	12
2.5	The OpenFlow table [22] . . . . .	15
2.6	ICMP message exchange through an OF switch . . . . .	16
2.7	Comparison between the traditional approach and the NFV approach, simplified . . . . .	17
2.8	NFV reference architectural framework [26] . . . . .	19
2.9	The NFV-MANO architectural framework with ref. points [27] . . . . .	20
2.10	Hierarchical view in SDN multi-domain scenario [27] . . . . .	21
2.11	*-as-a-Service models [33] . . . . .	23
2.12	National Institute of Standards and Technology Standard Cloud definition . . . . .	25
2.13	Position of SDN resources in an NFV architectural framework . . . . .	26
2.14	Position of SDN controllers in an NFV architectural framework . . . . .	27
2.15	Position of SDN applications in an NFV architectural framework . . . . .	29
2.16	Reference multi-domain SDN/NFV architecture, in general . . . . .	30
2.17	Average NBI response time and 95% confidence interval when SFC add and update actions are performed, as a function of the number of SFCs . . . . .	32
2.18	Average NBI response time and 95% confidence interval when SFC delete and flush actions are performed, as a function of the number of SFCs . . . . .	32
2.19	Reference multi-domain SDN/NFV architecture, specialized for the use case of IoT data collection and related cloud-based consumption. . . . .	33

2.20	The NFV/SDN test bed setup developed to demonstrate multi-domain SDN/NFV management and orchestration . . . . .	34
2.21	Average NBI response time and 95% confidence interval at the SDN/cloud VIM with increasing number of service chain requests.	34
2.22	Reference NSH experiment scenario . . . . .	35
2.23	WEST-to-EAST throughput measured at the OF-S within Node (2) while applying dynamic SFC . . . . .	36
3.1	Some known deployments of OpenStack [47] . . . . .	38
3.2	Main functions of a Cloud [49] . . . . .	38
3.3	High level view of OpenStack [36] . . . . .	39
3.4	OpenStack logic overview [47] . . . . .	40
3.5	Keystone work flow [47] . . . . .	45
3.6	Keystone logical architecture [52] . . . . .	45
3.7	Nova logical architecture [52] . . . . .	47
3.8	Nova work flow when a request for a new virtual resource arrives [47] . . . . .	48
3.9	Swift logical architecture [52] . . . . .	49
3.10	Glance logical architecture [52] . . . . .	50
3.11	Cinder logical architecture [52] . . . . .	51
3.12	Neutron logical architecture [52] . . . . .	53
3.13	OpenStack instance provisioning workflow [56] . . . . .	54
3.14	OpenStack basic services [57] . . . . .	54
3.15	OpenStack fulfilled cloud functions [58] . . . . .	55
3.16	Complete OpenStack components overview [52] . . . . .	56
4.1	Neutron logical view vs. physical view . . . . .	58
4.2	OpenVSwitch overview [61] . . . . .	61
4.3	Compute node internals . . . . .	62
4.4	Network node internals . . . . .	63
4.5	Iptables graphical representation . . . . .	65
4.6	Egress instance flow in an OpenStack compute . . . . .	67
4.7	Egress instance flow in an OpenStack network node . . . . .	73
4.8	Example of a VNI with VLAN tenant network [64] . . . . .	78
5.1	Ryu architecture [74] . . . . .	93
5.2	OpenDaylight Carbon modules [76] . . . . .	95
5.3	ONOS architecture . . . . .	97



5.4	ONOS modules . . . . .	98
5.5	Example of implementation of ONOS in OpenStack . . . . .	98
5.6	High level table design . . . . .	101
6.1	Deployed SONA-OpenStack topology . . . . .	105
6.2	ONOS active modules . . . . .	107
6.3	Intermediate configuration phase . . . . .	109
6.4	vRouter ports . . . . .	110
6.5	Internal configuration of the vRouter . . . . .	111
6.6	Completion of configuration . . . . .	111
6.7	L3 topology . . . . .	113
6.8	Bot working after the deployment . . . . .	116
6.9	Simple example of a sonaMonitor execution . . . . .	118
7.1	OpenStack topology . . . . .	122
7.2	Nova hypervisors . . . . .	125
7.3	Neutron agents . . . . .	127
7.4	Dashboard . . . . .	128
7.5	Ceilometer statistics . . . . .	129
8.1	User view of the testbed . . . . .	132
8.2	Physical implementation of the testbed . . . . .	134
8.3	TCP throughput - 1 client 1 server . . . . .	136
8.4	TCP throughput - 2 clients 1 server . . . . .	137
8.5	TCP throughput - 4 clients 1 server . . . . .	137
8.6	CPU usage during the TCP test with 1 client and 1 server . . .	138
8.7	CPU usage during the TCP test with 2 clients and 1 server . . .	139
8.8	CPU usage during the TCP test with 4 clients and 1 server . . .	140
8.9	Memory usage during the TCP test with 1 client and 1 server .	141
8.10	Memory usage during the TCP test with 2 clients and 1 server .	141
8.11	Memory usage during the TCP test with 4 clients and 1 server .	142
8.12	Packet rate sustainability, 1 internal flow - 64 B UDP packets .	143
8.13	Packet rate sustainability, 1 internal flow - 1500 B UDP packets	144
8.14	Packet rate sustainability, 2 internal flows - 64 B UDP packets .	144
8.15	Packet rate sustainability, 2 internal flows - 1500 B UDP packets	145
8.16	Packet rate sustainability, 4 internal flows - 64 B UDP packets .	146
8.17	Packet rate sustainability, 4 internal flows - 1500 B UDP packets	146
8.18	Packet rate sustainability, 8 internal flows - 64 B UDP packets .	147

- 8.19 Packet rate sustainability, 8 internal flow - 1500 B UDP packets 148
- 8.20 Packet rate sustainability, 1 external flow - 64 B UDP packets . 149
- 8.21 Packet rate sustainability, 2 external flows - 64 B UDP packets . 150
- 8.22 Packet rate sustainability, 4 external flows - 64 B UDP packets . 150
- 8.23 Packet rate sustainability, 1 external flow - 1500 B UDP packets 151
- 8.24 Packet rate sustainability, 2 external flows - 1500 B UDP packets 151
- 8.25 Packet rate sustainability, 4 external flows - 1500 B UDP packets 152