



# **UNIVERSIDAD DE EXTREMADURA**

## **Escuela Politécnica**

**MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN**

**Trabajo Fin de Máster**

**TÍTULO**

**Despliegue de un testbed de redes definidas por software para la  
gestión de recursos de red en un CPD**

Laura Amarilla Cardoso

Diciembre 2015



# **UNIVERSIDAD DE EXTREMADURA**

## **Escuela Politécnica**

### **MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN**

#### **Trabajo Fin de Máster**

#### **TÍTULO**

### **Despliegue de un testbed de redes definidas por software para la gestión de recursos de red en un CPD**

Autor: Laura Amarilla Cardoso

Directores: José Luis González Sánchez

Javier Carmona Murillo

#### **Tribunal Calificador**

Presidente: Pablo García Rodríguez

Fdo:

Secretario: María Luisa Durán Martín-Merás

Fdo:

Vocal: Francisco Javier Rodríguez Pérez

Fdo:

CALIFICACIÓN:

FECHA:

# General index

	pag.
Chapter 1. Introduction	8
Chapter 2. State of the art	13
2.1 SDN concept	13
2.2 SDN architecture	15
2.3 Control and data plane decoupling	17
2.4 Network virtualization	34
2.5 Programming in SDN	40
2.6 SDN development to network status lecture	42
Chapter 3. Cloud Computing and SDN	44
3.1 Cloud computing definition	44
3.2 Cloud computing types	46
3.3 Service models	48
3.4 Cloud computing platforms	50
Chapter 4. Testbed Design	56
4.1 SDN in data centers	56
4.2 Infrastructure used	58
4.3 Cloud platforms evaluation	62
4.4 OpenStack integration with SDN controllers	66
4.5 Testbed design	78

Capítulo 5. Resultados	82
5.1 Herramientas y metodología usada	83
5.2 Rendimiento de Ryu	91
5.3 Implementación de una red SDN con Mininet y Ryu	100
5.4 Pruebas en CPD	121
5.5 Comparativa de los resultados	135
5.6 QoS en SDN	136
Chapter 6. Conclusions and Future work	141
Anexes	143
References	151
Agradecimientos	153

# Figures index

	pag
Figure 1. Traditional architecture vs SDN	14
Figure 2. SDN architecture	16
Figure 3. Switch OpenFlow and components	19
Figure 4. Flowchart	21
Figure 5. Centralized vs Distributed	24
Figure 6. Data plane	29
Figure 7. Click example	31
Figure 8. Common architecture	33
Figure 9. Network virtualization	35
Figure 10. Northbound API and Southbound API	41
Figure 11. Programming in SDN	42
Figure 12. OpenStack	50
Figure 13. OpenStack component	52
Figure 14. OpenNebula services	54
Figure 15. OpenNebula components	55
Figure 16. Lusitania topology	57
Figure 17. Opendaylight architecture	69
Figure 18 .Opendaylight graphical interface	71
Figure 19. OpenStack Dashboard	72
Figure 20. Ryu architecture	73
Figure 21. Integration ryu-plugin with OpenStack	75
Figure 22. Network topology	78
Figure 23. VNC graphical interface	81
Figure 24. Ryu controllerr startup	93
Figure 25. OpenFlow messages with cbench tool	99
Figure 26 . Topology used in Mininet	100
Figure 27. Connection establishment between Mininet and Ryu	101
Figure 28. Connection establishment between Mininet and Ryu with Wireshark	102
Figure 29. OF_HELLO message	102
Figure 30. Of_features_request messages	102

Figure 31. Of_features_reply message	103
Figure 32. PACKET_IN message in the controller	104
Figure 33. PACKET_OUT message	104
Figure 34. Connectivity test between h1 y h2	105
Figure 35. Ping command results in the controller	106
Figure 36 . Ping command results between h1 and h2	106
Figure 37. PACKET_IN messages in command ping	107
Figure 38. Command dpctl-show results	107
Figure 39. Examination of flow table using dpctl	108
Figure 40. Jperf graphical interface	110
Figure 41. Command ping results	110
Figure 42. VLC graphical interface	116
Figure 43. Video transmission configuration in the server	116
Figure 44 . Video transmission configuration in the client	117
Figure 45. Openflow messages in the controller	117
Figure 46. Wireshark capture of video transmission HTTP	118
Figure 47. Transmitted video	120
Figure 48. Ping command in the controloller	121
Figure 49. Ping between u1 and u2	122
Figure 50. ovs-dpctl show command	123
Figure 51. ovs-dpctl dump-flows command	123
Figure 52. ovs-ofctl dump-flows br-int command	124
Figure 53. brctl show command	125
Figure 54. ps -ea   grep ovs command	125
Figure 55. Ping between u1 and u2	126
Figure 56. Video transmission in the controller	132
Figure 57. HTTP video transmission	133
Figure 58. Image at the beginning of the transmission	134
Figure 59. Image during the transmission	134
Figure 60. Ryu startup with QoS	137
Figure 61. QoS transmission in h1	140
Figure 62. QoS transmission in h2	140

## Table index

	pag
Table 1. Number of headers in Openflow specification	32
Table 2. Comparative OpenStack and OpenNebula	65
Table 3. SDN controllers comparative	68
Table 4. Ryu OpenFlow protocol messages	74
Table 5. IP addresses of OpenStack instances	80
Table 6. Latency tests results	95
Tabla 7. Latency estimations	95
Table 8. Throughput test	97
Table 9. Flow table	108
Table 10. Performance comparative	136
Table 11. Flow entry in the switch	138

## Graphs index

Graph 1. Ryu controller latency	96
Graph 2. Throughput test	98
Graph 3. Server bandwidth	111
Graph 4. Client bandwidth	112
Graph 5. Server bandwidth	113
Graph 6. Client bandwidth	113
Graph 7. Jitter	114
Graph 8. HTTP Transmisión	118
Graph 9. HTTP packets	119
Graph 10. Bandwidth in the server	127
Graph 11. Bandwidth in the client	128
Graph 12. UDP transmission in the server	129
Graph 13. UDP transmission in the client	130
Graph 14. Jitter	131
Graph 15. Video transmission with HTTP	133

# Chapter 1. Introduction

Designing and managing networks has become more innovative over the past few years with the aid of Software Defined Networking (SDN). Although this technology seems to have appeared recently, it is actually part of a long history of trying to make computer networks more programmable.

Traditional networks involve many kinds of equipment such as routers, switches, firewalls, networks address translators, server load balancers and intrusion-detection systems. Routers and switches run complex, distributed control software which implements networks protocols that are subjected to standardization and interoperability testing. Networks administrators typically configure individual network devices using configuration interfaces that vary between vendors and even between products from the same vendor.

These types of networks are ill-suited to meet requirements of today's enterprises, carriers and end users. Network designers are constrained by the limitations of current networks, which include high complexity, inconsistent policies, inability to scale and vendor dependence. This mismatch between market requirements and network capabilities has brought the industry to a tipping point. In response, the industry has developed the Software-Defined Networking architecture, which is an emerging architecture that is dynamic, manageable, cost-effective and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications.

As organizations are planning to roll out new virtualized applications, they will need to consider how to make the transition to software-defined networking and the impact it will have on the way they build their networks. The number of applications that need to be managed is growing fast, so it is imperative to connect disparate resources, some of them virtualized and some on legacy infrastructure, with the SDN environment.

Nowadays, data centers have reached a turning point due to enterprises deploying new applications are looking to increase business agility and gain management insights. The deployment of these applications to realize this objective requires configuring each device of the network, an incredibly



time-consuming task that increases administrative overhead and make the applications deployment slower.

Server virtualization and cloud computing are changing the form in which data centers are used nowadays. For example, many data centers are totally virtualized in such a way that it is possible to provide all the information in a easy way. Both approaches provide a more efficient usage of IT resources using flexible, on-demand, and rapidly scalable models. These technologies working together enables organizations to better meet organizational demands and provide greater agility for the data centers. Despite of the rapid development of this two technologies, networking technologies still lack consistency to support an infrastructure based on virtualization or cloud computing. At this point, SDN can help in solving many hard problems of the networks.

The concept of SDN is not new and completely innovative but appears as the result of contributions, ideas and progress in network investigations. The idea of a network that can be scheduled and not simply configured, has already been implemented by many manufactures but with the limitation that all efforts had been very proprietary.

The Open Networking Foundation (ONF) [1] is an user-driven organization focused on promoting the adoption of software-defined networking through open standards development. It was founded by prominent companies such as Deutsche Telekom, Google, Microsoft, Verizon and Yahoo!. The ONF determines three important issues: active networks, data and control plane separation and OpenFlow API and NOS.

Active networking emerges at early 90's. These networks, oriented to the network control, suggest a programmable interface or network API that opens to the user the individual resources of each node such as processing, memory resources and packets processing, and they allow to include custom functionalities to the packets circulating through the node. It is necessary in this type of networks the usage of different programming models at the nodes, leading to the first step in network virtualization investigation and in the development in frameworks or platforms to the improvement in applications in the node. The Architectural Framework for Active Networks v1.0 model [2], used in PlanetLab Platform [3], in which researchers conducted experiments in virtual execution environments and packages that were demultiplexed in each virtual environment based on its

header, contains a Node Operating System (NodeOS) which share a group of Execution Environments (EEs) and active applications (AAs). The technology push that encouraged active networks allows to reduce the computational cost, progressing in programming languages and in virtual machines technologies. An important catalyst in the ecosystem was the increase in the interest from agencies, which resulted in the creation of programs such as DARPA, from mid-90s to early 2000s.

At early 2000s, the significant growth of traffic volume that circulated in the network and the necessity of improving the management and using administrative functions as, for example, traffic engineering, traffic prediction and fast recovery of falls in the network; made researchers looked for new approaches to improve this characteristics, since the resources and methods used until now were limited.

Until this moment a close relationship between data and control planes exists. This relation was a barrier in the development of these new technologies. In addition, the speed in the links (backbones) was bigger at each time and this make the packet forwarding mechanisms to be concentrated in the hardware, separating the control or administration of the network to a software application. In this area, the project FortCES (Forwarding and Control Element Separation) [4], which was standardized by the IETF (RFC 3746), established an interface between the data and control plane at network nodes. Another innovation was the settlement of a centralized logic control of the network, proposed by the Routing Control Platform (RCP) project [5], whereby the administration is facilitated and the capacity of innovation and network programming was gave. RCP had an immediate application since it took advantage of the existence of the Border Gateway Protocol (BGP) to install entries in the forwarding tables of the routers.

With data and control plane separation, “clean-slate” architectures were developed, among which, the 4D project [6] and the Ethane project [7] must be mentioned. 4D project propose a 4 layer architecture based on its functionality: data plane, discovery plane, dissemination plane and decision plane. The proposal of Ethane consists on a centralized control system of network links to business networks. With the operational deployment of this project at Standford University, the Openflow creation stage began and, particularly, the simple design of the switch of Ethane project was converted in the base of the OpenFlow API.

During the first decade of the 21st century, several investigation groups and enterprises started to be interested in the experimentation in wide networks, due to the success of experimental infrastructures like PlanetLab and Emulab, and the availability of more funds from government institutions to invest in the sector. Thanks to this work, one of the most pioneering project in the area of virtual networks called GENI (Global Environment for Networking Innovations) [8] appeared. This project provides a virtual lab to experiment with new network architectures and distributed systems and where the programmers can use the allocated resources to try out, among other issues, new network configurations at universities, industrial labs, sensor networks and wireless networks. At the same time, at Stanford University, a group of researchers set up the Clean Slate Program [9], whose mission was to “reinvent the internet” for the purposes of overcoming fundamental architectural limitations of Internet, incorporating new technologies and enabling new class of applications and services.

Around 2010, there was a massive response from the industry to create fully open interfaces, including OpenFlow [1], defining a single, open and common way for all manufacturers in extending their equipment and exposing them to pieces of software that enable programming.

Since then, the term SDN has attracted considerable interest in the industry of computer networks. Either as a threat or an opportunity, most of the manufactures of equipment, many of the creators of software, dedicated companies or virtualization solutions, cloud computing, etc., have developed their own solutions or adopted Openflow standard. However, the reactions of those companies is not homogeneous: nearly all of them implement the OpenFlow solution, but, some place it as a “low segmet” of its product range, and set apart advanced features for his own implementation. In this manner SDN would become a proprietary solution and even more tied to a single manufacturer than traditional networks.

According to the analysis published by Extreme Networks [10] about the development of SDN in 2015, the market evolution of these networks will be guided by a growing virtualization of network functions in data centers, “overlay” architectures” dominating big data centers, SDN completely developed in campus market, innovations in vertical market and the first deployment based in OpenDaylight.

SDN by its own does not resolve any concrete problem, but offers a more flexible tool to manage networks in the best way possible. As a consequence of the significant traction that Software-Defined Networking has gained, a full study of this new approach is carried out in this project, emphasizing the application of this new networking concept in a data center. This project is the result of a collaboration grant between Fundación Cénits-COMPUTAEX and University of Extremadura. Its structure consists; firstly, on a state of the art about the evolution of this new type of networks and an explanation of its characteristics. Secondly, the relation between cloud computing platforms and SDN networks is mentioned. The project finishes with the application of a testbed in both, an emulated environment and a real one. Having taking into consideration all the issues mentioned before, the main target of software defined networking is no other but automating more and more arduous tasks of the network administrator in a data center.

During the development of this project, the general objective was the implementation and evaluation of a SDN network. But several specific points were developed:

- Making a deep study of the basic principles of SDN.
- Knowing different SDN controllers in order to make a decision about which one is the most appropriate to use in the development of this project.
- Evaluating the functionality and the performance of the controller.
- Developing a SDN implementation in a virtual environment and in a real one.
- Making an evaluation of the implementation named before.

# Chapter 2. State of the art

Setting up new services and online applications, both fixed and mobile terminals, has transformed communications networks in a strategic point at business world and homes. The infrastructure in charge of the transmission of information from IoT devices (routers, switches, 3G-4G networks, QoS, access points, etc) have to adapt to new post-PC services (VoIP, Virtualization, QoS, Cloud Computing, IoT applications) and, at the same time, provide security, scalability, speed and availability among others. Because of this, in recent years the idea of customized the behaviour of the network has emerged and also giving flexibility to the users to use the network resources according to their own needs.

SDN is a new network architecture which takes out the current rigidity in traditional networks, enabling a more flexible behaviour of the network and more adaptable to the needs of each organization or user. His centralized design allows to collect information about the network and use it to improve and adapt their policies dynamically.

## 2.1 SDN concept

According to the Open Networking Foundation, “Software Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services” [1]

In traditional networks, control and data plane are running at the same switch or router. These devices support two fundamental functions:

- A transport function, which consists of sending data packets to the routes previously calculated,
- A control function, which consists of exchanging information about connectivity with other routers/switches and calculate routes based on the information obtained.

Conventional networks are not able to cover the new requirements of end users, service providers and companies. As a consequence, new approaches are sought in networking. SDN is a network model that, on one hand, clearly separates the transport and control functions and, on the other hand, implements the control function with software (instead of doing it in hardware). In addition, SDN centralizes in a single element (the controller) this control function which means that the network can be programmed using applications, providing enormous flexibility and ease in the deployment of network functions. In figure 1, a comparison between both models is shown.

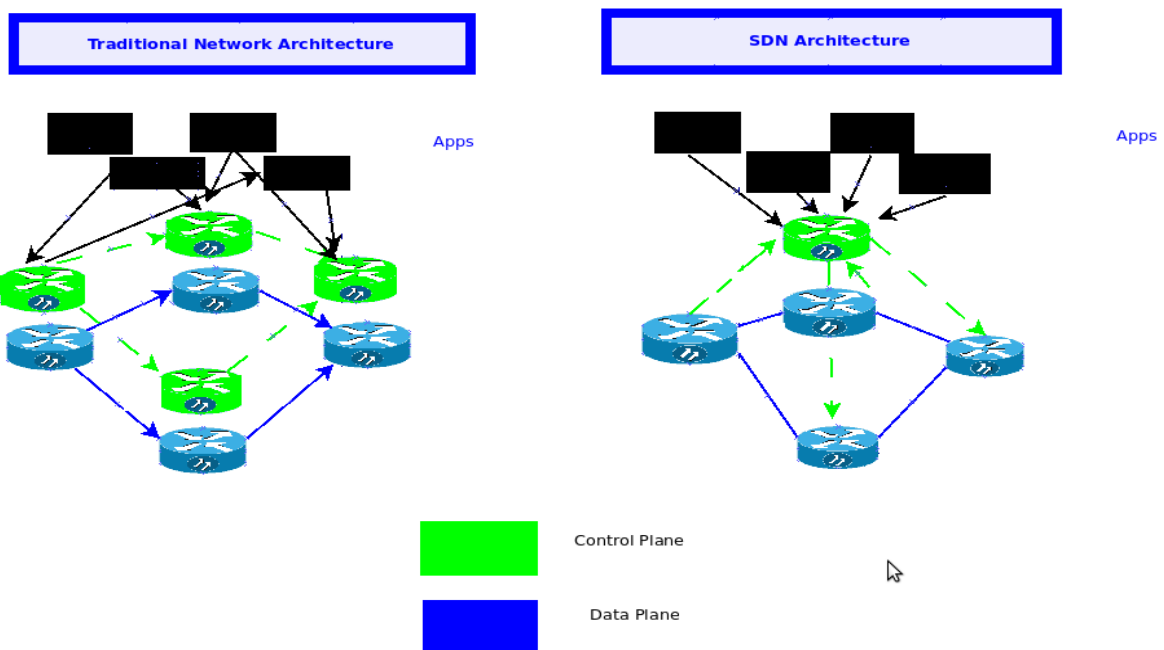


Figure 1. Traditional architecture vs SDN

The concept underpinning SDN is simple. If the data and control plane are decoupled, the formerly static network can become intelligent, responsive, programmable and centrally controlled.

## 2.2 SDN architecture

The aim of SDN is to provide open interfaces that enable the development of software which can control the connectivity provided by a set of network resources, and network traffic travelling through them, combined with possible inspection and modification of traffic that may be performed in the network.

SDN architecture is based on the transmission of “flows”. A flow can be defined as a sequence of packets travelling through the network and sharing fields in its packet header.

The communication process begins when the user makes a request of a service. The switch received that request and communicates with the SDN controller, that provides the instructions to be followed for the specified service. The communication between the switch and the controller is made through OpenFlow protocol through a secure channel. An analogy with a three layer architecture can be made, shown in figure 2:

- **Application layer.** It is related with the applications required for the business and customized by the user. Applications use SDN communication services through northbound API in the control layer, like REST, JSON, XML, etc. This layer allows services and applications simplify and automate configuration tasks, manage new network services, offering operators new sources of income, differentiation and innovation.
- **Control layer.** It will be in charge of defining the treatment of flows (in terms of data) via the control software of SDN (controller). The transmission of the instructions to the switch is done through a standardized protocol, commonly OpenFlow.
- **Infrastructure layer.** Formed by communication physical devices. It is made up with network nodes which perform switching and routing of packets. It provides an open programmable access through southboubd API, like OpenFlow.

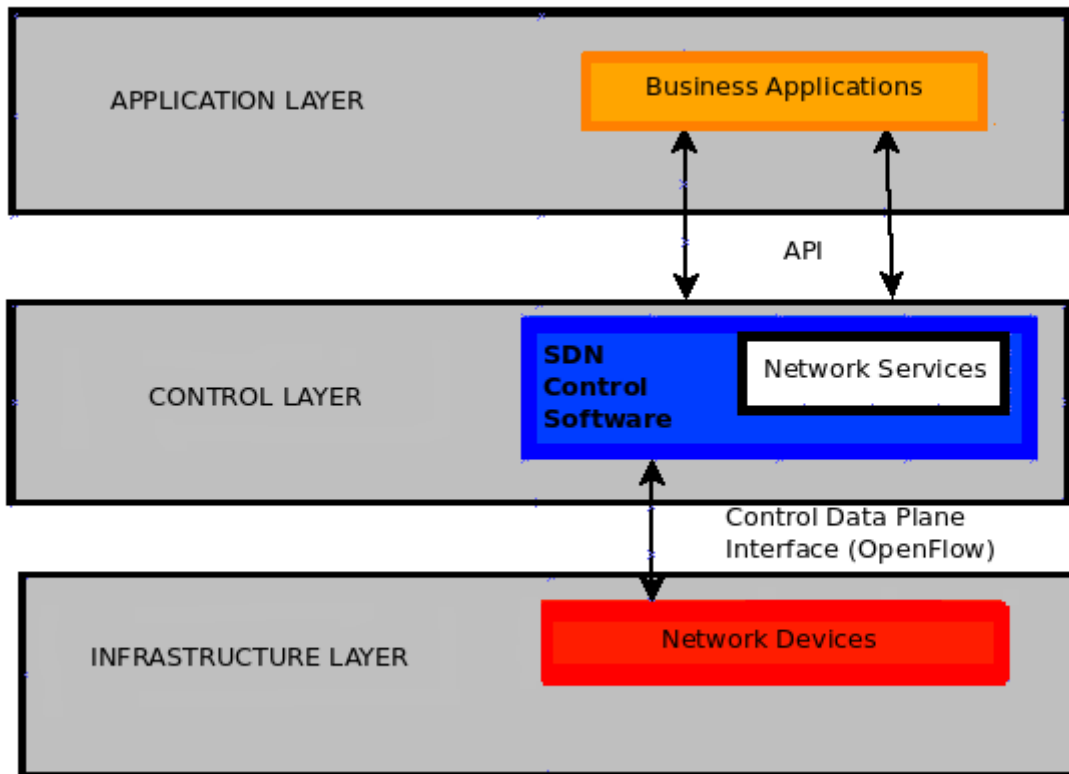


Figure 2 . SDN architecture

The main components in the SDN architecture are:

- **SDN controller**, a software entity that has exclusive control over an abstract set of data plane resources and where treatment information policies are generated. It may also offer an abstracted information model instance to at least one client.
- A **secure communication channel**, like Secure Sockets layer (SSL), which connects the control software and the switch through OpenFlow protocol.
- A **switch** which supports OpenFlow.



Taking into consideration all points mentioned, it can be concluded that the SDN architecture is:

- **Directly programmable.** Network control is directly programmable because it is decoupled from forwarding functions.
- **Agile.** Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
- **Centrally managed.** Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- **Programmatically configured.** SDN lets network managers configure, manage, secure and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
- **Open standards-based and vendor-neutral.** When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

The term SDN makes it possible to modify the behaviour of the whole network remotely. To this end, such networks are based on three main features: data and control plane separation, network virtualization and programmability of a network.

## 2.3 Control and data plane decoupling

One of the principal approaches of SDN networks is that it separates data and control plane.

The control plane is the logic which controls the behaviour of network forwarding. It decides how, when and where the flows must be sent. Examples: routing protocols, transport network configuration (firewalls configuration, load balancing configuration).

The data plane makes the transfer of data effective, namely, forwards the traffic according to the logic of the control plane. This plane is normally implemented in hardware but it is becoming increasingly implemented in software-based routers.

SDN technology eliminates network intelligence of hardware in traditional networks delegating decision-making capabilities on the server. The technology decouples the data layer from the control layer and each of this layer is automated. This makes it possible the independent control and evolution of each of those planes. It also permits manage the network with a single high level software program so its whole behaviour can be controlled.

## 2.3.1 Control plane

The control plane has two main different aspects:

- OpenFlow
- SDN controller

### OpenFlow

OpenFlow [1] is the protocol most widely used in Software-Defined Network. It is the first protocol designed for SDN networks, which facilitates the programmability of the network, by configuring, management and control the flow from a centralized software. OpenFlow allows traffic partition, choose the best route and improve the ways in which packets are processed. It has the focus on new forms of routing, security, traffic control and addressing schemes among others.

This protocol propose a way for researchers to experiment with protocols in the networks at the same time that vendors do not have to expose their internal process. The proposal of OpenFlow is very clear, it allows researchers to evaluate their ideas in a real working environment.

Its development is based on Ethernet switches. It is focused on the fact that the most of these devices contains flow tables which runs to implements firewalls, QoS (Quality of Service) and collect statistics. Although flow tables of each provider are different, there are a set of functions which are common to many of switches and routers. The main objective of OpenFlow is to exploit these common features.

OpenFlow is an open communication standard between a controller (main element in SDN networks) and switches. Its structure is designed with messages that set a communication and generate the appropriate actions.

Using OpenFlow allows centralizing migration packet decisions, so the network can be programmed independently from individual switches and equipment in the data center. This lets having a greater control of the network and, therefore, greater efficiency. The dissociation between the control and data plane supposed to delegate this function to a external controller which means that it is possible to program out of the device the way to process flow packets. These packets will be listed on a flow table, who contains the informations about the flows of packets in a way that will be possible know what sources and destinations have passed through the device.

The controller communicates with the OpenFlow switch using a secure channel and the protocol orders in an effective form the switch OpenFlow to update the flow entries to realize different actions over some flows of traffic that passed through the switch. All the intelligence of the network lives in the controller and the work of switches is reduced to forward the traffic according to the flow tables entries.

Therefore, it can be said that OpenFlow is the medium whereby the device operates as configuration done in the controller. In figure 3, it is shown the OpenFlow switch and its components.

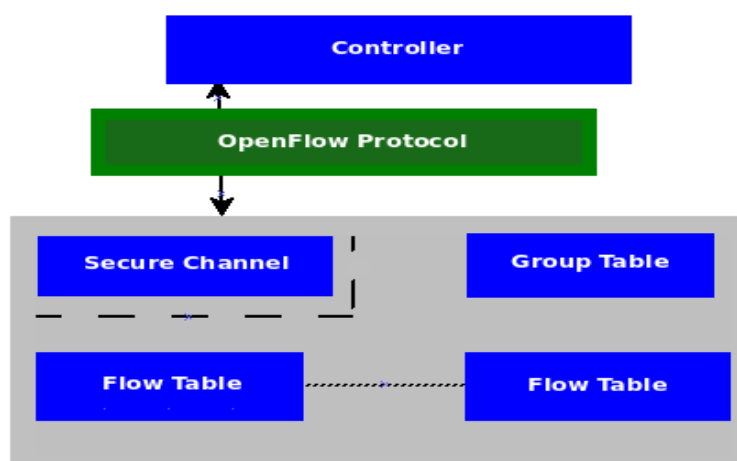


Figure 3. Switch OpenFlow and components

The OpenFlow switch is the basis device in the packet routing. It is a connectivity device created to connect with the controller in order to receive the rules that will define data flow patterns using OpenFlow. It could be a software program or a hardware device. Its idea is to exploit the fact that most Ethernet switches have flow tables to implement firewalls, QoS or collect network statistics. OpenFlow takes advantage from that and offers an open protocol to program the flow-tables in different switches and routers. One example of an OpenFlow switch is OpenvSwitch [11]. It is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It enable massive network automation and supports standard management interfaces and protocols. Openvswitch is used in multiple products and runs in many large environments. It has also been integrated into many virtual management system including Openstack, OpenNebula and oVirt.

In an SDN architecture, a switch performs the following functions:

- Encapsulating and forwarding the first packet of a flow to an SDN controller, enabling to decide whether the flow should be added to the flow table.
- Forwarding of incoming packets out the appropriate port based on the flow table. The flow table may include priority information dictated by the controller.
- Packet dropping on a particular flow, temporarily or permanently, as dictated by the controller.

An OpenFlow switch consists of one or more flow tables, a group table and an OpenFlow channel. These three components are necessary to the correct operation of the switch and realized the following actions:

- **Flow tables.** It performs a search function that compares the fields in each package with a flow table that is in the switch and find a match. In other words, this table looks if the packet headers flowing in the switch matches any of the entries in the flow table. The actions carried out by the switch depend on whether there is agreement or not. If there is no match traffic came back to the controller.
- **Group table.** It contains group entries. The ability of a flow to point a group let OpenFlow represents additional transmission methods.
- **Control channel.** The OpenFlow channel is the interface which connects each OpenFlow

switch to a controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch. The communication between the controller and the switch is established by a secure connection. An OpenFlow switch may have several OpenFlow channels, with a different controller each.

Using OpenFlow, the controller can add, update and remove the flow entries at the flow table. Next it will be seen how OpenFlow realize the concordance making use of the flow entries in the flow table in a specific switch looking at the flowchart shown at figure 4.

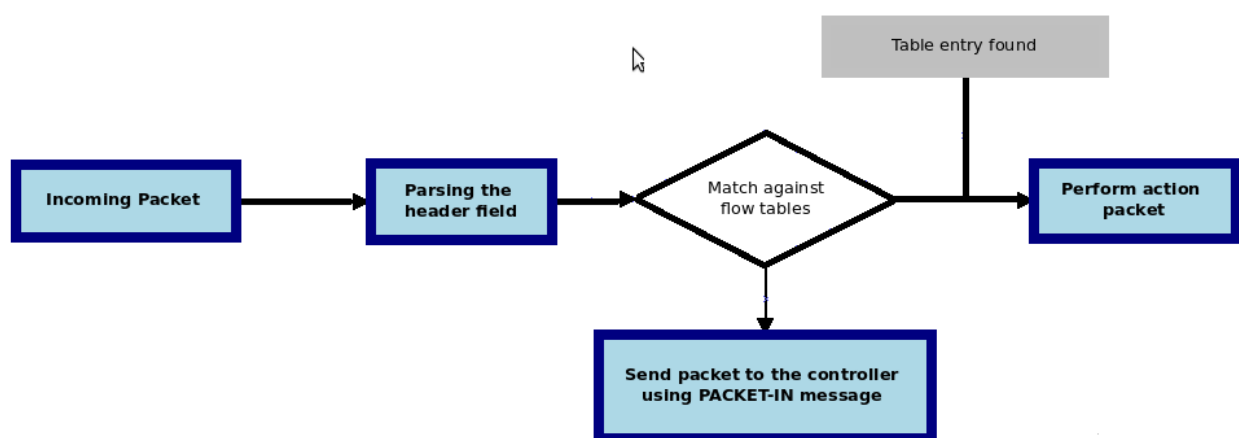


Figure 4. Flowchart

On the receipt of a packet, the switch starts analysing the packet headers and based on the values of these fields of the headers, it will try to pair this packet with the entries in the flow table.

If there is no match, the output will depend on the configuration of the miss table, like for example, the packet arrives to the controller, will be drop or continue to the next flow table.

If, on the contrary, a match exists, the actions specified in the flow table will be performed. For scalability reasons, it makes sense to match the maximum number of packets possible to avoid having too wide traffic to the controller.

The instructions associated with each flow table can contain both, actions or changes in processing. Actions can describe packet routing instructions, packet modification (whereby the switch should change several values in packet headers) and processing of the group table. Modifications in processing enable packets to be sent to following tables for further processing and allow the information, as metadata, to be communicated between tables. This processing does not specify which is the next table. At this point, the package normally is modified and forwarded.

Flow entries should be forwarded to a port. Normally, it is a physical port, although it could also be a logic port defined by the switch or by the specification (this type of port should specify actions of concrete routing).

The actions associated with flow entries can direct packets to a group, which specifies additional processing. It is essentially a set of actions list. It permits the switch referring to a common set of actions which could be done in multiple groups of matched flows. These groups represent the serie of actions related with saturation, more complex semantics forwarding (for example, multipath or link aggregation).

The OpenFlow protocol provides reliable message delivery and processing, but does not automatically provide acknowledgements or ensure ordered message processing. The OpenFlow message handling behaviour is provided on the main connection and auxiliary connections using reliable transport.

Message delivery is guaranteed unless the whole OpenFlow channel fails. Switches must process each message received from a controller in full, generating a reply. In case a switch can not process a message received from a controller, it must send an error message back.

Message ordering can be ensured through barrier messages. In the absence of these messages, switches will arbitrarily reorder messages to maximize performance so controllers will not depend on a specific processing order.

As it has been mentioned above, the OpenFlow channel is used to exchange messages between the switch and the controller. A characteristic controller manages several OpenFlow channels, each one

assigned to a different switch.

The switch should be able to establish the communication with the controller making use of an IP address and a port configured by the user. When a communication is set, both communication endpoints may send an OFPT\_HELLO message, which include the version field set to the highest OpenFlow protocol version supported by the sender. When receiving this message, the receiver may calculate the OpenFlow version to be used.

If the negotiated version is supported by the receiver, then the connection proceeds. Otherwise, the receiver must reply with an OFPT\_ERROR message with a type field of OFPET\_HELLO\_FAILED, indicating there is no compatibility between versions.

In the case that a switch loses contact with all controllers, it immediately enters either “fail secure mode” or “fail standalone mode”, depending upon the switch implementation and configuration. In “fail secure mode”, packets and messages destined to the controllers are dropped and flow entries should continue to expire. In “fail standalone mode”, the switch processes all packets like if it was a legacy Ethernet switch or router. Upon connecting to a controller again, the controller has the option of removing all the entries.

## **SDN controllers**

The controller is the core of the SDN architecture as it centralizes all the communication passing through network devices. It works as a network operating system (NOS) that has a global vision of all the network. It is responsible for manipulating the flow table in the switch and the communications between applications and network devices using OpenFlow protocol.

Generic functionality as network state and network topology information, device discovery, and distribution of network configuration can be provided as services of the NOS. Using NOS, the administrator no longer needs to take care about the low-level details of data distribution among routing elements, for instance. This fact enables a faster network configuration as well as application and services deployment.

There is a very diverse set of controllers and control platforms with different design and architectural choices. Existing controllers can be categorized based on many aspects. Taking the point of view of architectural choice, they can be classified in two main categories:

- Centralized. A single entity that manages all forwarding devices of the network. Naturally, it represents a single point of failure and may have scaling limitations. A single controller may not be enough to manage a network with a large number of data plane elements.
- Distributed. A distributed network operating system can be scaled up to meet the requirements of potentially any environment, from small to large-scale networks. A distributed controller can be a centralized cluster of nodes or a physically distributed set of elements.

While the first alternative can offer high throughput for very dense data center, the latter can be more resilient to different kinds of logical and physical failures. A cloud provider that spans multiple data centers interconnected by a wide area network may require a hybrid approach, with clusters of controllers inside each data center and distributed controller nodes in the different sites. In this architectural point of view, independent controllers can be spread across the network, each of them managing a network segment, reducing the impact of a single controller failure. A distributed controller can improve the control plane resilience, scalability and reduce the impact of problems caused by network partitions. In figure 5, both kinds of architectural views are presented.

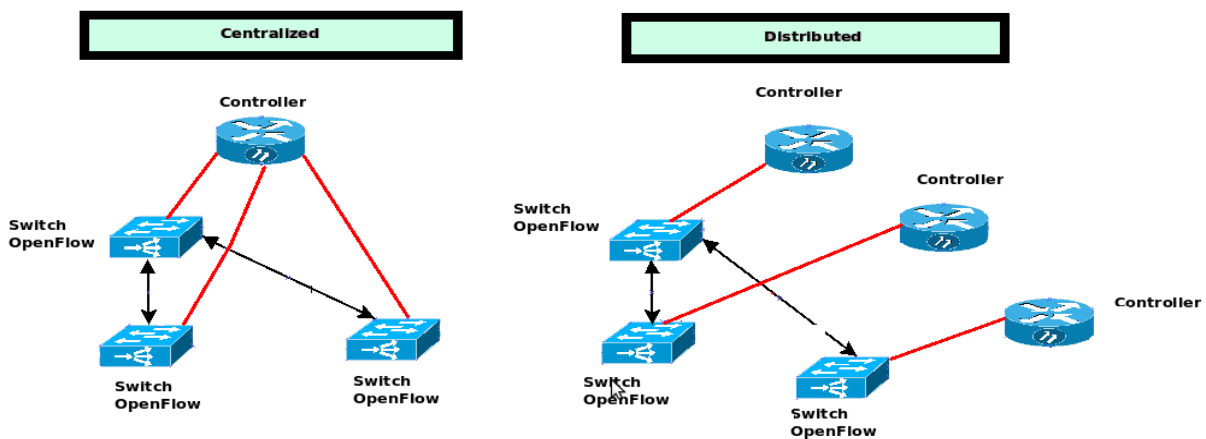


Figure 5. Centralized vs Distributed



There are several considerations when choosing which controller to use:

- Programming language, which can affect to the performance of the controller. The most used are C/C++, Java and Python.
- Learning curve, that make reference to how long it takes to learn how to operate the controller.
- Support user-band and community behind the controller
- Different controllers have different approaches in terms of
- APIs they can support
- Openstack support
- If its purpose is focused to use in education, research or production

There are several Open source OpenFlow controllers available for research and development whose major distinction is the programming language they are written in. They can be a simple software, where the administrator can control all OpenFlow switches and is responsible for the process with all flows, or an implementation with various administrators which allows to manage in a independent way different set of flows. The most prominent controllers are described below.

## **NOX**

NOX [12] is an OpenFlow controller open source that provide a platform to write network control software in C++ or Python. It was produced by Nicira Network and developed at the same time than the OpenFlow protocol.

Control device is done through the OpenFlow protocol. NOX provides low-level methods to interact with the network in its most basic form. High level functions and events are created with network applications called components. These components can be classified in two groups: core components, that provide some network applications and web services functionalities, and network applications, which offers network functionalities such as routing or monitoring.

There are two available versions of NOX:

- NOX *Classic*. The old version with support for Python and C++. It does not exist at present time, so it is not recommended to be implemented although it contains a wider range of modules than the new one.
- NOX. New version of this software. It offers support only for C++. In comparison with the classic version, it has less number of applications. Its performance is faster and a more debugged code. It supports OpenFlow 1.0 and a NOX application called CPqD supports versions 1.1, 1.2 y 1.3. It is under permanent development.

## **POX**

POX [12] is a very similar tool to NOX, but with the difference that POX only use Python to program the flow rules to the controller while NOX can use both, Python and C++. It also supports OpenFlow 1.0. It is widely used, maintained and supported and it is relatively easy read and write code in it. The main disadvantage is its performance. It is mainly used in research, demonstration and to learn concept. Principally, it is a platform for the fast development and prototyping to the software network control using Python. POX is under constant development being the developers target to give a stable API.

## **Beacon**

Beacon [13] is a controller implemented in Java which support operations based in events. It is software used to program the flow rules in SDN controllers. It has been tested in several environments without an interruption in his services during long periods of time. The software is on the market under GPLv2 and University of Standford FOSS licenses. Beacon has fast performance due to the usage of several threads of running. It allows manipulating libraries dynamically, meanly, this controller enables to stop, start or refresh the execution of those libraries in time of execution without stopping the behaviour of another libraries in an independent way.

## **Floodlight**

Floodlight [14] is a Java based open source controller that support OpenFlow 1.0. Some of its advantages include a good documentation and integration with REST API, and the production level and support for OpenStack and multiuser cloud. The main disadvantage is its steep learning curve. It is an enterprise-class controller available with Apache license to any purpose. Floodlight is designed to work with a large number of switches, routers, virtual switches and access point which can support OpenFlow protocol.

## **Opendaylight**

Open-source platform which enables to create programmable networks, like software-defined networks, and virtualization network functions. It is a controller implemented in software contained into its own Java virtual machine. The target of Opendaylight project [15] is to provide a common and open industrial platform as support for a source code robust and extensible and common abstractions for northbound capacity. Thus, Opendaylight pretends to include areas such as the controller platform or network applications protocols, virtual switches or physical interfaces of the device.

Its main advantage resides in providing independence with regards to providers, since it allows organizations to opt for different technologies which will be interoperable. The Opendaylight software is a combination of components among are included controller, interfaces, protocols plugins and applications.

It has a great number of advantages like the wide acceptance of the industry and the integration with OpenStack. However, it is extremely complex, the documentation is very few and its learning curve is very steep.

## Ryu

Component-based framework used in the development of software-defined networks. Ryu [16] offers components with well defined APIs which simplifies the creation process of SDN controllers. It is an open source SDN controller based on Python with layer 2 (L2) and 3 (L3) functionalities, although the messaging service supports components written in other languages. It supports all OpenFlow versions from 1.0 to 1.4, and some Nicira extensions. All the code of Ryu controller is free and available under Apache 2.0 license.

The main goal of Ryu is to develop an SDN operating system that has high quality enough to be used in large networks. To this end, this controller contains event management, in-memory state management, application management and series of reusable libraries (for example, NetCOONF library, sFlow/NetFlow library and OF-Config). Additionally, it supports applications such as OpenStack Quantum, layer-2 switch, Generic Routing Encapsulation tunnel interface (GRE) and tunnel abstraction. It also provides services about topology and statistics.

Its main advantages reside in the all versions of OpenFlow support and its OpenStack integration. Its main disadvantage is the performance.

## 2.3.2 Data plane

Data plane is the battle horse of switching devices in networks. It is the part of the network that transports the traffic. It has the responsibility of analysing the headers packets and manage QoS actions, filtering, encapsulation, etc. to maintain the performance needed in each type of network.

Data plane enables data to be transferred from and to the clients, managing them through multiple protocols. Traffic in data plane travels through routers. Next, it will be seen how a traditional data plane works having a look at the image shown in figure 6.

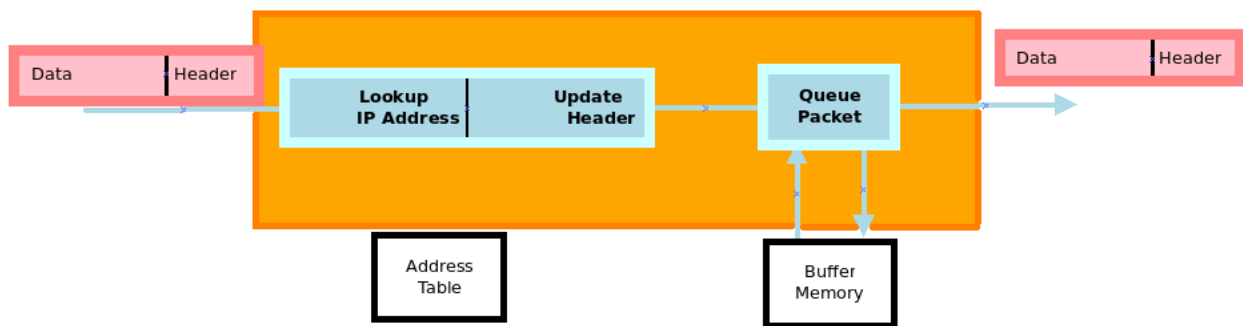


Figure 6. Data plane

First, the router receives the packet and examine its header, looking for its destination. Then, it consults its local forwarding table to determine the next step of the output interface. It modifies then the header and sends the packet to the proper output interface.

The architecture of data plane does not realize completely the forwarding tasks, but it simply sends data packets so its functions are named as “switching”.

If the relation with SDN is pointed, it is necessary to define how an OpenFlow switch works. OpenFlow switches achieve a union action of the operations of data plane.

If it is wanted a set of operations at data plane by a simple searching or operation, this requires additional customizations at data plane. To customize data plane it should principally to recognize

that data plane it is not only more than transmission algorithms with operate over the packets. Some bits of these algorithms match with the packet and realize an action typically under the mandate of control plane.

Data plane is able to make a wide range of functions, including: forwarding, access control, manipulation of headers packets, monitoring traffic, buffering, marking packets, traffic configuration and profound inspection of packets. The realization of these operations requires the power of customized the data plane in various forms. Software represents unique opportunities to do that.

Network devices are very different, so it is very difficult to add or to modify its functions since these functions are normally implemented in hardware. Particularly, even OpenFlow switches are able to do the paradigm of a particular union action. But this is only a specific type of data plane, so it so it will be seen if capacities of existing data planes can be extended, potentially in software.

The objectives when designing a programmable data plane are:

- Flexibility. It must be able to realize a great variety of data plane operations.
- It has to be extensible. It must allow to add new operations of data plane in an easy way.
- Providing interfaces between different data plane operations to be easier composing operations such as forwarding, traffic configuration, etc.

The modular router is one of these devices in a programmable data plane and implemented in software. An example of software that can be used in this case is Click [17] where each one of the elements provides a unique function like, for example, switching, searching and classification, dropping, etc.

The idea is that a developer of the data plane will take this Click elements and match them in a processing packets canalization. In figure 7 an example is shown in which it takes the packets from eth0 interface, count them and discard them by shambling construction blocks of the developers who can implement a great variety of complex functions.



*Figure 7. Click example*

The execution of a Click configuration is quite simple, it takes the packets from an input interface, shows them and then it discards them. In the configuration it will be taken as source element something that captures the packets, sends them to an element which shows them and then, to an element of discard. When executing Click, just simply invoke the configuration file and shows each packet which can be seen at the interface. It is possible to group Click elements to build a classic IP router.

Extremes between elements can be possible routes for the packets. An ascendant elements sends a packet to a descendant element. It exists an arrival packet element where data are delivered to the next processing unit. The descendant elements rescue data from ascendant elements. There are elements where transmission ports requires a packet from previous elements.

Packet storage is done by queues. It is necessary data storage until they are asked. Queues are implemented as elements in a way that its insertion or discard could be more configurable.

A lot of the new protocols requires changes in data plane and those protocols should do the forwarding of packets with acceptable speeds. They also need to be executed in parallel with existing protocols. Thus, it is necessary a programmable data plane platform to develop network protocols that can forward packets with high speed and are able to execute multiple data plane protocols in parallel.

During last 5 years, the number of headers in OpenFlow specification has evolved, like table 1 shows.

Version	Date	Headers
OF 1.0	December 2009	12
OF 1.1	February 2011	15
OF 1.2	December 2011	36
OF 1.3	June 2012	40
OF 1.4	October 2013	41

*Table 1. Number of headers in Openflow specification*

Forwarding protocols designers pointed that data and control plane are not separately enough and there is no easy way to modify formats of packets. Moreover, the addition of new features require to change in both forwarding elements and the controller.

All of this makes particularly difficult that SDN switches and control protocols evolve independently of hardware. Ideally, it is wanted to be capable of making SDN switches evolve and defining the control protocols according to the requirements of control programs and not being limited by current hardware capacities.

Current generation of ASCIs switches begins to make this possible. It can be mentioned Intel FlexPipe, RMT and CiscoDoppler architectures, which start this type of programmable packet processing, although they are difficult to program. The design of a high-level language have three goals: independence from protocol, “target independence” and field reconfiguration.

The forwarding protocol contemplate an alternative future where the hardware can be defined independently of control protocol. This new paradigm also provides reconfigurability of the switches. Thus, it would be necessary a network assembler language, which is a low-level programming language for devices of programmable networks that provides a 1-1 correspondence with the underlying hardware and use well-defined constructions to define low-level packets operations. The idea that exists behind the network assembler language is that the assembler can permit writing network programs highly optimized, and can facilitate the compiler an optimum design.



A common architecture can be seen to the one shown in the figure 8.

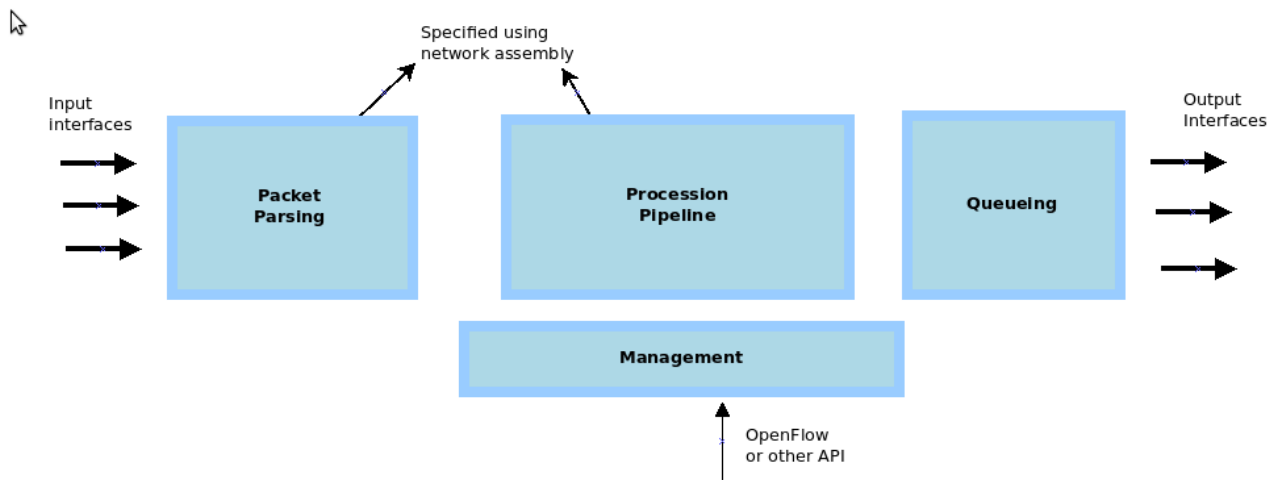


Figure 8 . Common architecture

There are input interfaces which drives to process of packets, followed of a channelling of packet processing and then it is queue at the output. The programming in network assembly implies the explicit description of the channelling of packet processing as a set of instructions or an operation sequence. The programming of a hardware channel also provides the opportunity of implement what is called “locally contained applications” where hardware can locally update the states, like records and tables, without sending packets back to the controller. The switch can update all its states locally according to a predefined specification.

Given a high-level protocol specification, the compiler can optimize the design in hardware, using the assembler to compose or decomposed tables, reordering operations or making specific operations based on traffic profiles.

## 2.4 Network virtualization

Network virtualization is a technique which pretends to simulate hardware and network elements using software. It abstracts real hardware from a non-real machine implemented in software (which is named the virtual machine), but with memory assigned, space in disk, etc, like a real machine. Thus, virtual network is defined as a framework that decouples and isolates virtual networks from the underlying physical network hardware. Because of that, applications see a machine adapted to its necessities but in some way independently of real hardware that supports it. To create and manage that virtual machines it is used a software called hypervisor.

Therefore, Network virtualization (VN) creates virtual logical networks which are decoupled from underlying real hardware to ensure the network could be better integrated in virtual environments. In the last decade, organizations have adopted virtual technologies to get a better efficiency in his resources. In general terms, it consists of emulating a hardware with a service or network installed creating, doing this, a virtual machine which is capable to operate like traditional hardware.

Thus, virtualization consists of the creation of virtual instances of any type of resources (operating system, storage, hardware) inside a common physical network infrastructure. Virtualization allows the coexistence of several virtual instances on the same hardware and independent one from each other. The most known example is the creation of an operating system inside other named host.

A virtualized network abstracted from physical hardware must still provide similar features and guarantees of a physical network, only with greater flexibility and agility than before, including more operational efficiency and hardware independence.

Although in networking area, SDN networks and virtualization are different concepts, they are related between each other. This relation is done to the sense that centralized control functions are usually implemented as virtual switches (namely, simulation of a switch in software) running in virtual machines located in physical servers and managed by an hypervisor.

It can be concluded then that the relationship between SDN and network virtualization is that SDN is a mechanism, and network virtualization is a concept, but SDN can be used to achieve network virtualization by abstracting the configuration parameters for the networking devices into logical centralized layer. That layer can be represented as a network hypervisor, where all the network applications can sit on top of the hypervisor, and implement the needed logic for all the underlying network devices independently. With this approach, changes in network can be done in minutes, by implementing new network applications that address a certain demands in the network.

To describe network virtualization the figure 9 is shown.

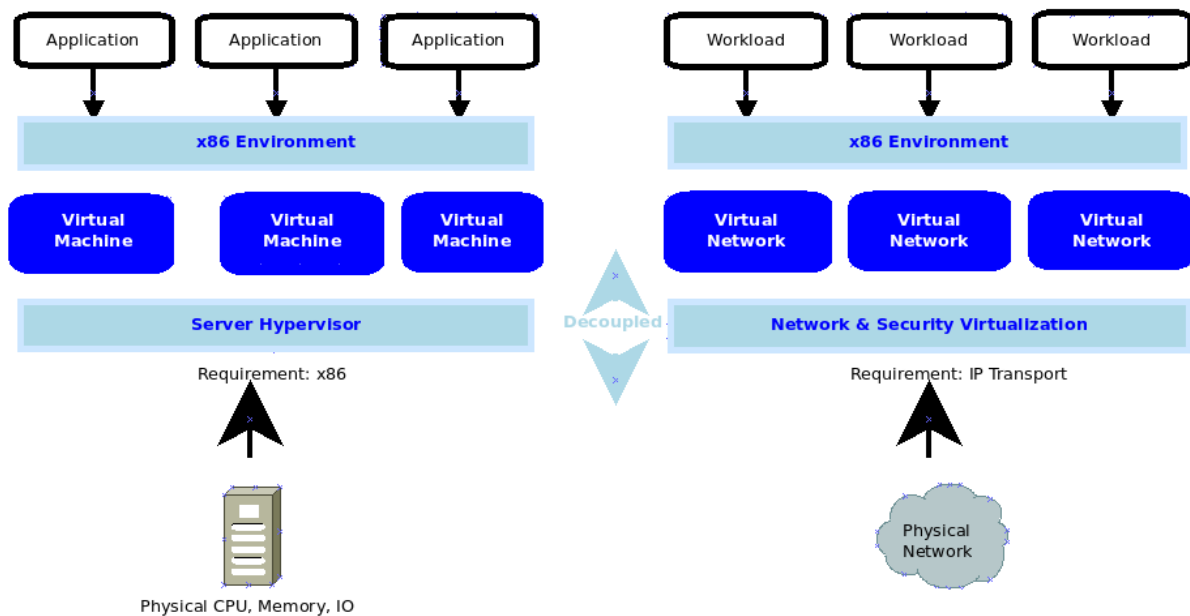


Figure 9. Network virtualization

On the left part, there is a more traditional method of virtualization, which is the virtual machine environment. In this environment, CPUs, memories and physical IO are essentially abstracted thanks to an hypervisor that execute the virtual machines. Each one of those virtual machines provide the appearance of an execution in a dedicated platform, when in fact, the hypervisor is responsible to ensure the isolation and resources control.

On the same way that a machine can be virtualized, a physical network can also be virtualized, so, similarly to the hypervisor, a virtualization layer can provide virtual networks which can coexist on top of a single shared physical network. Additionally, this virtualization layer is responsible to ensure isolation and a good resource to share among several virtual network.

The main benefits that network virtualization provides are:

- Fast innovation. Services can be developed in software speeds.
- New form of control network. Being capable of manage several newtwork groups allow to explore new forms of control.
- Provider election.
- Programming and operation simplification.

The design objectives of virtual networks are:

- **Flexibility.** Virtual networks must support different types of topologies, forwarding and routing architectures and they should be able to be configured in an independent form.
- **Maneuverability.** Politics and mechanism should be separated.
- **Scalability.** They must maximize the number of virtual networks that exists in an architecture.
- **Security and isolation.** Virtual networks must isolate both logic networks and resources to ensure any virtual network can use more physical resources than necessary or interfere with the assigned resources to other virtual network.

The concept of multiple networks coexisting can be seen from different perspectives:

- **Virtual local area network (VLAN).** A VLAN is a local area network which group a set of devices in a logical way and not physically. The communication between the different devices in a local area network is directed by the physical architecture (geographic and direction limitations). Thanks to virtual networks it is possible to eliminate this limitations, because it is defined a logical segmentation based on grouping elements according to some criteria such as MAC addresses, number of ports, protocols, etc.

There are some types of VLAN:

- Level 1 VLAN. They defined a virtual network according to the connection of the switch.
- Level 2 VLAN. A virtual network is defined according MAC addresses of the hosts. This type of VLAN is more flexible than previous one because of the network is independent from the location of the stations.
- Level 3 VLAN. Inside this type of networks, it can be defined:
  - VLAN based on network address. It connects subnetworks taking into account the source IP address. It provides a great flexibility in the way that the configuration of the switches change automatically when a station moves. As disadvantage, it could exist a light decrease in the performance, since the packet information can be analyse carefully.
  - VLAN based on protocols. It allows to create a virtual network according to the protocol (for example, TCP/IP, IPX, etc). It lets grouping all the devices that use the same protocol in the same network.

These networks have the advantage of providing a greater flexibility in the administration and the changes in the network, since the architecture could be modified using the parameters of the switches. There is an increase in the security, since the information is encapsulated and possibly, analysed. VLAN perform a reduction in the traffic transmission in the network.

- **Private virtual networks (VPN).** Local area networks (LAN) are the internal networks of organizations, namely, the connections between devices of a particular company. These networks are connected most frequently to the Internet using an interconnection device. Often, enterprises need to communicate with branch offices, clients or even the staff that can be geographically far away by Internet.

However, transmitted data through the Internet are much more vulnerable than when they travel through an internal network of the organization, because the path taken is not previously defined, which means that data should go over a public network infrastructure that allows to several entities. Because of this reason, it is possible that throughout the line,

an intrusive user, listen to the network or even kidnapped the signal. Therefore, confidential information of an organization or company should not be sent under such conditions.

The first solution to satisfy this need of secure communication imply connecting remote networks using dedicated lines. However, as most companies are not able to connect two remote local area networks with a dedicated line, sometimes it is necessary to use Internet as transmission medium.

A good solution consists of using the Internet as transmission medium using a tunnel protocol, which means that data are encapsulated before being sent in an encrypted way. The term Private Virtual Network (VPN) is used to make reference to the network created artificially in this way. It is considered to be a virtual network because it connects two “physical” networks (local area network) through a not much reliable connection (Internet) and private because only the devices that allow to one of this two networks of the VPN can “see” the data.

Therefore, a VPN system gives a secure connection with a low cost, because it is only necessary the hardware of both sides. However, it does not guarantee a quality of service comparable with a dedicated line, since the physical network is public.

A virtual private network it is based on a protocol called tunnel protocol, namely, a protocol with encrypt data that are transmitted from one side of the VPN to the other.

The word “tunnel” is used to symbolise the fact that data are encrypted from the moment they arrive to the VPN until they go out from it and, therefore, anyone that is out of one of the extremes of the VPN is not able to understand them. In a two-device VPN, the client is the part which encrypt and dis-encrypt data in the user side and the server (commonly called remote access server) is the element that dis-encrypt data from the size of the organization.

So, when a user needs to access to the private virtual network, his request is transmitted without being encrypted to the gateway system; that it is connected with the remote network using as intermediate the public network infrastructure; then, it transmit this request in an

encrypted form. The remote device provides data to the VPN server in his network and this server sends the encrypted response. When the VPN client of the user receive the data, it decodes them and finally send to the user.

- **Active networks.** The idea beyond this type of networks is to decouple hardware and programming of services offered. On the contrary to traditional networks, active networks allow the injection of programs at real time over hardware components of itself. To do this, they are based on the concept of “active packet”, that is a datagram which carries data and code or references to the code. In this manner, it can be generalised and accelerated the installation of a new service only programming this packets and migrating from a hardware component to another inside the network.

Ported data in the packets could be modified at real time. Both data and programs which will process them, or make reference to them, are encapsulated in active packets. Software/hardware components that run the code to transform ported data in active packets are named actives nodes. There are two types of active networks:

- Discrete active networks. Active packets are architecturally separated to the code. The administrator of each active nodes statically load the code that process active packets.
  - Integrated active networks. The packet distribution and the code use the same architecture. The code is loaded dynamically as needed.
- Programmable networks. This type of networks is distinguished from other types of networks by the fact that it can be programmed starting from a minimal set of APIs, from which, ideally, can be composed with an infinite spectrum of high level services. Programming of network services is obtained by the integration of computational capacity, beyond the scope of the carried out in existing routers and switches.

It is should be recalled that OpenFlow is an emergent network virtualization technology that provides flexibility and control to the user to configure its network environment according to specific specifications.

## 2.5 Programming in SDN

It should be recognised that while OpenFlow makes programmable networks to be possible, it does not mean they are easy. It is difficult to perform several independent tasks using a SDN controller.

Additionally, OpenFlow offers a low level of abstraction. It provides a control channel to manage the flow entries and switches. Operating in this level of abstraction to makes high level tasks, like implementing security policies or traffic load balancing, is particularly difficult. A higher level of abstraction is needed to carrying out more complicated functions.

Programming helps in the collection of information from several network devices, and automatically gets new configurations as a response to the network dynamic conditions. Therefore, programming is very useful in the automation of the complex management of the network. To put it simply, a programmable network allows users to access to the network intelligence to be able to adapt easily and quick to the necessities of the demand of the several applications and types of traffic in a centralized and open way.

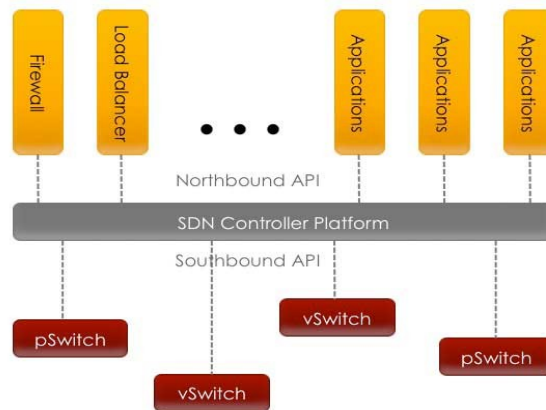
To do this, APIs are provided to the network devices in form of “software development kit” (SDK), which expose the functionality of network infrastructure to the developers. This is the most adequate model for users who wants to use its infrastructure and desire to program and manage this infrastructure with its own applications. Using the APIs, users can obtain the information needed about its network devices and then, modify the behaviour of the network.

In addition, users can create new on demand services, automate tasks, manipulate the forwarding and modify security policies. To sum up, APIs are used to get a customization, automation or extension of routers and switches functionalities giving as result a great interaction of the network.

With network programming, developers not only can interact directly with hardware but they can use agents to work with other protocols like OpenFlow.



Two types of interfaces used in SDN will be mentioned [18]: northbound API (which is used to communicate the controllers and the services and applications that are executed in the network) and southbound API (which provides communication and management between the SDN controller, nodes, physical or virtual switches and the routers). In figure 10, the location of both interfaces can be seen.



Source: *sdxcntral*

Figure 10. Northbound API and Southbound API

## 2.6 SDN development to network status

### lecture

Current software defined networks provides a control platform which offers global visibility and control of the network, as well as a control channel of the behaviour of the network using an open interface, like OpenFlow. However, this interface is relatively linked to existing hardware, fact that makes difficult to develop control applications at correct abstraction levels which are independent from the switch hardware and from protocols that control them. The development a better abstraction for programming in SDN involve the recognition of this programming implies a control loop that has three steps, as shown in figure 11.

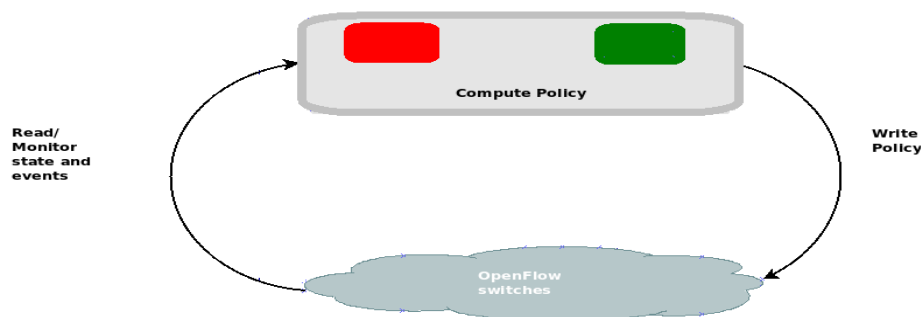


Figure 11. Programming in SDN

- The controller must be able to read and monitor the network status. It should also monitor events that change the conditions of the network (for example, changes in load traffic, security, etc)
- Calculating the forwarding behaviour of switches in the network based on policies specified by network operators. It is possible that the controller can have multiple policies in which there may be conflicts or should be developed.
- The controller have to write again a policy toward the OpenFlow switches.

Reading the status of the network typically involves several processing rules. It is assumed to have traffic counters, where each rule has bytes and packets associated with a particular flow. If several

rules exist, a way to express priorities is needed, so rule 1 takes precedence over rule number 2. The solution to this problem is to introduce predicates. Of course, it is necessary to have a delivery system which also translates those predicates in appropriate switch templates and the flow entries in the flow table. Another challenge is the limited number of rules that can be installed in the switch. For example, on the assumption that traffic should be monitored in a way that allows to produce a histogram of the traffic volume according to the IP address, clearly it makes no sense to create a rule for each IP address, so it is desired that the switch constitutes rules for new packets which arrives from different source IP address.

The solution to this problem is a primitive named “dynamic unfolding”, where the programmer can specify a GroupBy predicate (GroupBy(srcip)) and the runtime system will add rules dynamically to the switch for arrival packets with different source IP address.

A third challenge for the status network lecture is the common OpenFlow programming language. If a switch does not know how to manage a packet with the entries in the existing flow tables, it will send that packet to the controller. Then, the controller will install the rules in the switch, according to that policy. But, when more packets arrives to the controller before the rule is installed in the switch, the controller have to guess the action to be made with the packets. To fix this problem extra events are removed, letting the programmer to specify a limit where the runtime blocks the controller to identify extra packets as incoming events.

This goals have been joined in a SDN programming language named Frenetic [19]. It is a query language that allows the programmer to read the status of the network. It offers a way for the programmer to be able to query to know the exact information he is looking for.

The runtime of Frenetic is designed to minimize the controller overhead. It filters the traffic making use of high level patterns, limiting the number of values given back, and, it aggregates queries according to the number and size of the packets. The next step is to develop mechanism that enables the programmer to write control programs which estimate networks policies. Computational policies is particularly complicated because each one of the modules that observe the status of the network, can specify partially how to manage the traffic, but those modules can have conflicts among them.

# Chapter 3. Cloud computing and SDN

## 3.1 Cloud computing definition

*Cloud computing* [20] is a new form of treatment services delivery, applies to both a company or a particular user and Public Administration. This term refers to both, a technological conception or a business model that brings a great variety of ideas, such as information storage, communications among computers, service provision or applications development methodologies; all under the concept named the cloud.

Having tried to fit all this issues to define in a precise way cloud computing concept, it will be defined first the concept of Internet. Internet is known as “network of networks”, since it is a decentralized system of computer networks interconnected via different modes of connection which offers information spaces to any user who has access. This access to the information is done in a “transparent” way, that is, the physical location is not relevant to the user. This fact makes Internet an universal tool, like a cloud which can be accessed from any part of the world looking for information and services. Internet allows the user two important issues: public disclosure and service offering. The first of this matters is not considered a part of cloud computing model, so the focal point will be on services.

Among the great variety of services Internet can provide, it can be distinguished two types of services: those which use the network as a channel and those that offer its own resources. Hosting and mail services examples of the most popular PaaS options are Google App Engine, Window Azure, Red Hat OpenShift and Heroku.ing.

On the basis of foregoing considerations, cloud computing can be defined as a technological

understanding and a business model which offers storage services, access and usage of technological resources located on the network and where the channel concept is merely an instrument. Cloud computing aims to have all user's files and information on the Internet without relying on having enough capacity to store information. Business applications requires a large quantity and variety of hardware and software, which makes them very complex and expensive. Taking into account that big companies need a high number (among tends to hundreds) of applications; it is comprehensible that these companies do not get the applications they need and that small enterprises do not have any chance.

These difficulties will be able to be overcome thanks to cloud computing and will not need to manage hardware or software; leaving that responsibility to an experienced provider. Thus, cloud computing represents a new way of using information and communications technologies. The shared infrastructure makes it work as a utility: users only pay for they need, upgrades are automatic and enlargement or reduction of services includes a simple process. This means that users do not need to make investments on infrastructure but they use the one the service provider has. It can be concluded then that a cloud computing solution allows the user to optimize the assignation and cost of the resources associated to its needs of information processing. Last innovations about Cloud Computing are achieving that business applications to be more moving and have more collaboration capacity. In addition to this, it is starting to be got that users of this types of applications have real-time information.

As it can be seen, the term cloud computing is a fairly broad term and in some occasions can be confusing and, because of that, its essential features according to the NIST [21] (National Institute of Standards and Technology) are announced next:

- Available service in a automatic way and on-demand. Cloud resources can be started without the action of the operator of the company that offers the service.
- Access through the network. Resources are available through the network.
- Resources are grouping in pools in a multi-user model. In general, resources are shared by several clients, who can use them on-demand and should have privacy and security ensured.
- Elasticity. User resources can grow or decrease quickly depending on its needs.
- Per-use basis. The payment for cloud services is realized in correlation to the real use of the resources the user does.

## 3.2 Cloud computing types

In a short period of time, cloud computing have become in the most request technology by enterprises thanks to the advantages that offers over other types of technologies. Among these advantages flexibility, scalability and cost saving must be mentioned.

Not all cloud computing services and providers are equal, nor are the possibles relationships between clients and vendors. Based on those features, it is possible to classified clouds as public, private or hybrid.

### **Public cloud**

This kind of cloud are the standard type of cloud computing. On them, the cloud services vendor provides its resources in an open way to heterogeneous entities, with no other relation between them of just closing a contract with the service provider. Resources offered are located in external servers from the user, being able to access to applications for free or payment. Works from many clients can be mixed on servers, storage systems and other cloud infrastructure; although end-users have no knowledge of other clients works located in the same server.

Applications, storage and other resources are publicly available through the service provider which is the owner of the infrastructure located in its data center. Service access is only offered in a remote way, normally through Internet.

Its main advantage is that it is not necessary to install machines locally to obtain processing and storage capacity, which means not having to make an initial invest and maintenance costs, but paid for the usage. Services provider manages the operational charge and data security (backup, accessibility, etc) which makes the risk in adopting new technologies is low.

As disadvantages it has access to all the information to third parties and dependance on online services (through Internet). It can also be difficult to integrate those services with other proprietary systems. It is very important when it comes to bet for a cloud public service, to ensure that it is

possible to get all data of it, for free and in the shortest time possible.

## **Private cloud**

Management and administration of services is done by an entity, without the participation of external entities in the cloud and maintaining its control. These clouds are located in a local infrastructure managed by a single client who controls what applications want to run and where execute them. The client is the proprietary of the server, the network and disk and can decide what users are authorized to use the network infrastructure. Usually, a private cloud is a platform designed to obtain hardware only, that is, machines, storage and network infrastructure although it is also possible to have a private cloud which allows to deploy applications.

It provides a greater data security, since the location of them lies within the company itself. Furthermore, it will be easier to integrate the services provided with other own systems. Its major drawback is the initial investment in physical infrastructure, virtualization systems, bandwidth and security, leading to a loss of scalability of the platform, without forgetting maintenance expenses required.

Normally, complex entities, which need to centralized its computer resources while offering flexibility in the availability of data, opt for this type of clouds. Private clouds are a good option for companies that need a high data protection and service level issues.

## **Hybrid cloud**

This is an intermediate model between the models mentioned above. They combine local resources of a private cloud with the public one, namely, they allow a company to maintain the control of its principal applications while takes advantage of cloud computing where it is more convenient, making this company to be proprietary of some parts and share another, although in a controlled way. Thus, private infrastructure is increased with the cloud computing services of public infrastructure.

Hybrid cloud can be a previous step before relocating most part of the applications to the cloud, as it is less risky. It has the advantage of a more moderate investment and, at the same time, it counts with several modalities of on-demand service.

This type of cloud is getting a good acceptance in the company with a forward-looking, since they are developing cloud management software in order to handle the private cloud and acquire resources from large public providers.

## 3.3 Service models

Cloud providers give access to computer resources through the Internet and offers a serie of additional value-added services that can be classified depending on the degree of use of software and hardware components and its administration by third parties or its own administration. Its uses depends on the necessity a client/user has. There are three services models:

### **Software as a service (SaaS)**

At the topmost point of the current classifications of components in computing world are the final applications, final products that offers certain services for which they were created. In the service proposed as SaaS the final user employ the services offered by these final tools to directly implement processes in his company (for example, accounting applications, e-mail, workflow, document management program for the company, etc). That applications are located on the cloud provider services, with a per-use billing mechanism (in case of not being a free service) more or less simple.

The client will use the system hosted by that company, which kept the client information in its systems and provides the necessary resources to exploit that information.

In this model the software is directly offered as a service and in multiuser form where the application running in the infrastructure is used by several companies in such a way that data are



logically isolated or independent one from each other. The client will not have to be worried about the maintenance, the support and software availability, since those are managed by the provider.

Works as SaaS examples: MS Office, Sharepoint, CRM applications of Salesforce, e-mail, storage, games, office and collaboration applications, social networks, Google services and Microsoft Office 365.

## **Infrastructure as a Service (IaaS)**

This type of service offers to the client a storage space or processing capacity in his servers. In this case, the vendor provides storage capacities and raw process, over which the user should build the applications the company needs practically from scratch. Thus, the user must have “a hard disk of unlimited capacity” available and an almost infinite performance processor, only restricted to their financial capacity of hired service. IaaS can be defined, therefore, as a computational infrastructure distribution model as a service, normally using a virtualization platform.

Infrastructure as a Service (IaaS) provides options to the companies which need to adapt his servers and storage resources quickly and on-demand. The most prime example would be to convert all physical infrastructure of servers and storage in a IaaS infrastructure on the cloud, paying only for the use that will be done of it.

Most well-known cloud IaaS services are Amazon Web Services, RackSpace Cloud, Joyent or Windows Azure.

## **Platform as a Service (PaaS)**

It is an intermediate service model that directly offers an operating system and an environment where a service can be developed by a third party. PaaS is usually restricted to applications development companies that need tool to locate and develop its own applications. Among these tools are database, applications servers, sources managers, teamwork software, etc. The user can

directly use a load of services that allows to host and develop their own applications (own or acquired licenses) on a single platform.

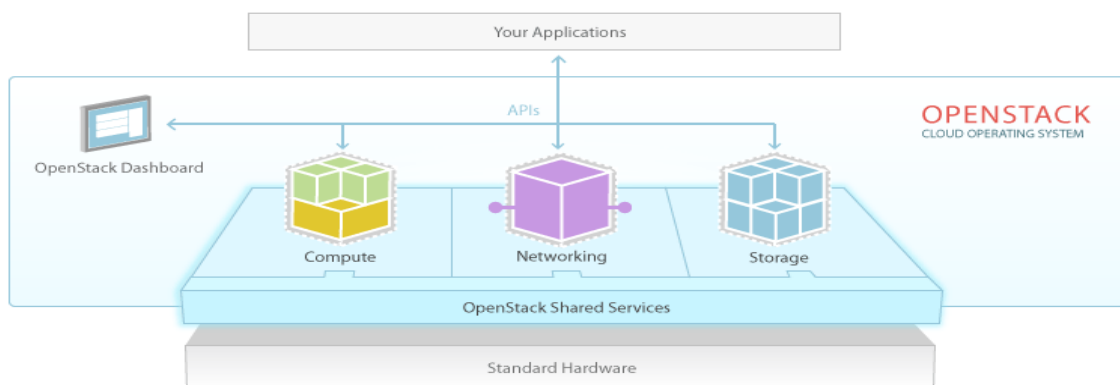
Some of the most popular PaaS options are Google App Engine, Window Azure, Red Hat OpenShift and Heroku.

## 3.4 Cloud computing platforms

One of the most important parts of cloud computing is the advent of cloud platforms. As its name suggests, this kind of platform lets developers write applications that run in the cloud, or use services provided from the cloud, or both. In this sections, two main open-source cloud computing platforms, Openstack and OpenNebula, are explained.

### 3.4.1 OpenStack

OpenStack [22] is defined as “a cloud operating system that controls large quantities of computing, storage and network resources through a data centre, all managed with a dashboard that allows administrators to control and, at the same time, boosting the resource provisioning to the users through a web interface”. Figure 15 show its general idea about it.



Source: [openstack.org](http://openstack.org)

Figure 12 . OpenStack

Its purpose is to provide an open source solution for the deployment of private and public clouds in order to obtain the maximum scalability and elasticity possible, with complete independence from any company, so whatever user or organization can collaborate in the project.

OpenStack project is a global collaboration between developers and cloud computing technologists producing an open standard cloud computing platform for both public and private clouds. The goal of the project is developing solutions for all types of clouds since it is easy to implement, massively scalable and feature rich. The technology consists of a series of programs interrelated, building several components for a cloud infrastructure solution.

OpenStack is the strategic option of different types of Organizations (OpenStack Foundation), since service providers, which want to offer computing services in the cloud in a standard hardware , to companies that look for implement private clouds initiatives, and big enterprises which implement a worldwide cloud-based solution.

## **Openstack components**

OpenStack consists of a set of applications which can be combined in terms of the features and needs in every particular case. Each component is completely autonomous and functional and use a queuing protocol, AMQP, to communicate with the other components and an API web to communicate with external processes or user. The main components, shown in figure 13, are the following:

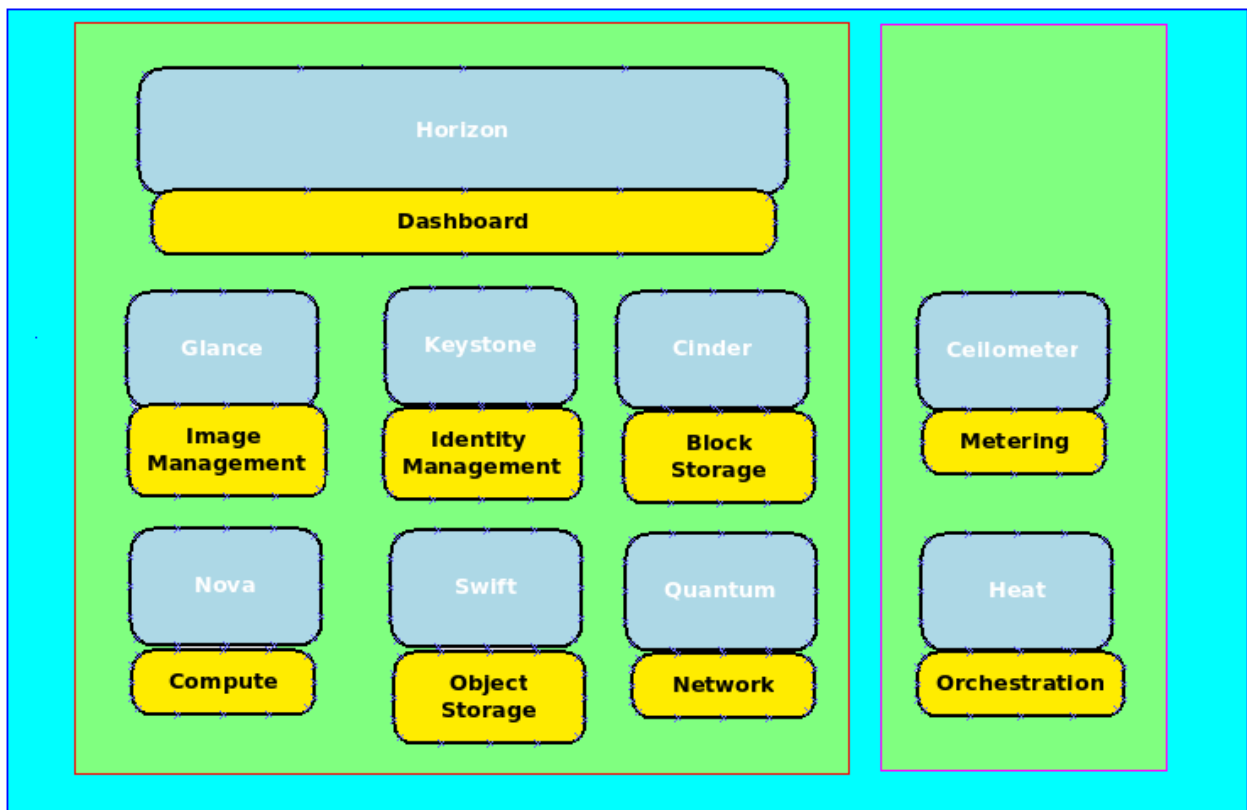


Figure 13. OpenStack component

- Ceilometer. It monitors the usage of each user in the infrastructure, as well as invoicing individually for those use.
- Cinder. It focuses on storage. It facilitates access to content hosted on disk drives that are in our cloud.
- Glance. Image management service, meaning a complete image backup hard drives available.
- Heat. It stores the requirements of an application that is offered from the cloud into a file that defines the necessary resources for its implementation. Therefore, it is responsible to establish an infrastructure requirements depending on the applications that are staying there.
- Horizon. It is responsible for displaying all through a graphical interface management OpenStack.

- Keystone. Controls different user ID connecting to the infrastructure.
- Neutron. It deals with interrelated OpenStack modules and allow these modules communicate with each other.
- Nova. It is used to deploy and manage the amount of virtual machines and other services needed.
- Swift. This is the module responsible for storing the system files to ensure their integrity and replicated by different disks found in the infrastructure, so that they are always available and accessible as quickly as possible.

## 3.4.2 OpenNebula

OpenNebula [23] is an open source management tool that helps virtualized data centers supervising all types of clouds; public, private and hybrid. It combines existing virtualization technologies with advanced aspects for multi-tenancy, automated provisioning and elasticity, offering a simple but feature-rich and flexible solution to build and manage enterprise clouds and virtualized data centers.

It integrates with existing technologies whenever possible. It works with MySQL, Ceph, LVM, GlusterFS, Open vSwitch, LDAP, etc; which allows to deliver a light, flexible and robust cloud manager.

The OpenNebula technology was Fully open-source software released under Apache licensedesigned to address the requirements of business use cases from leading companies and across multiple industries, such as Hosting, Telecom, eGovernment, Utility Computing ....

In terms of high-availability, OpenNebula consists in three different basic services, shown in figure 14:

- **OpenNebula Core:** It is the main orchestration component that supervises the life-cycle of each resources (e.g. hosts, VMs or networks) and operates on the physical infrastructure to

deploy and manage virtualized resources.

- **Scheduler:** The scheduler performs a matching between the virtual requests and the available resources by assigning a physical host, and a storage area to each VM.
- **Sunstone:** The GUI for advanced and cloud users as well as system administrators. The GUI is accessed through a well-known end-point (IP/URL). Sunstone has been architected as a scalable web application supporting multiple application servers or processes.

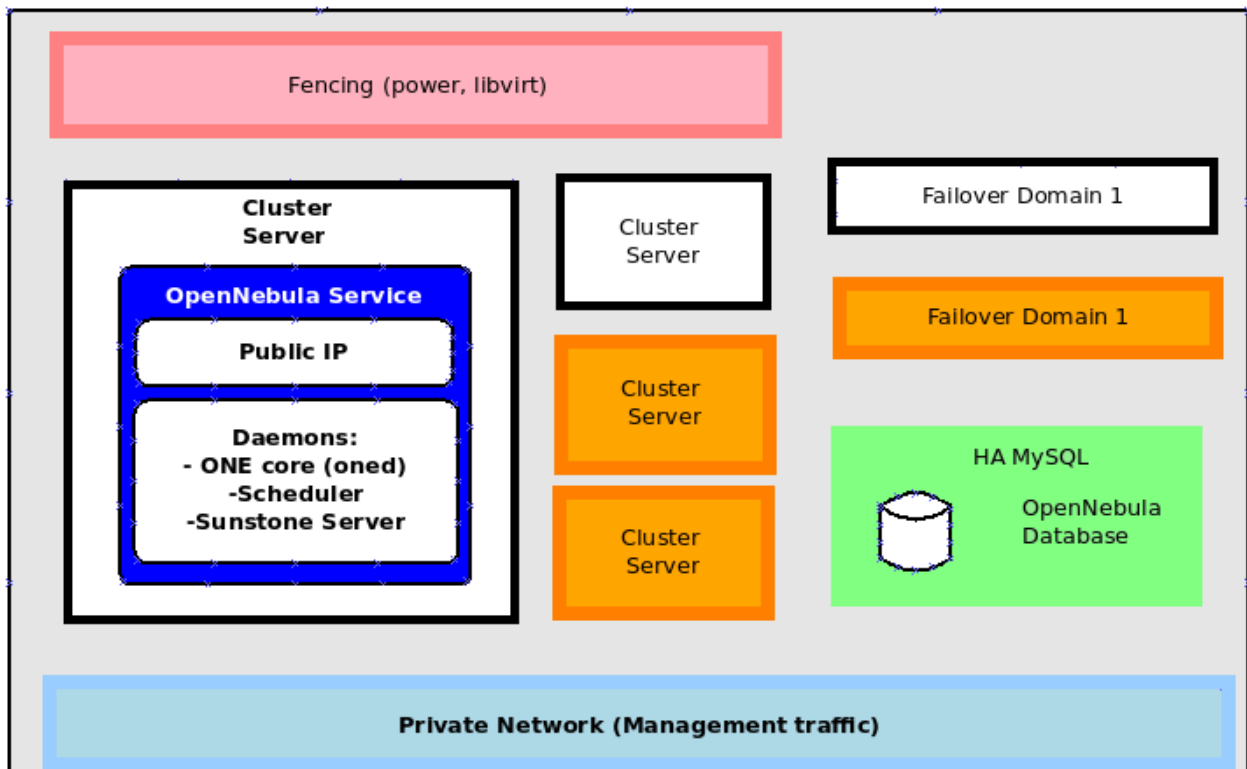


Figure 14. OpenNebula services

## OpenNebula components

OpenNebula is composed of several components, which can be observed in figure 15. The main components are the ones listed below:

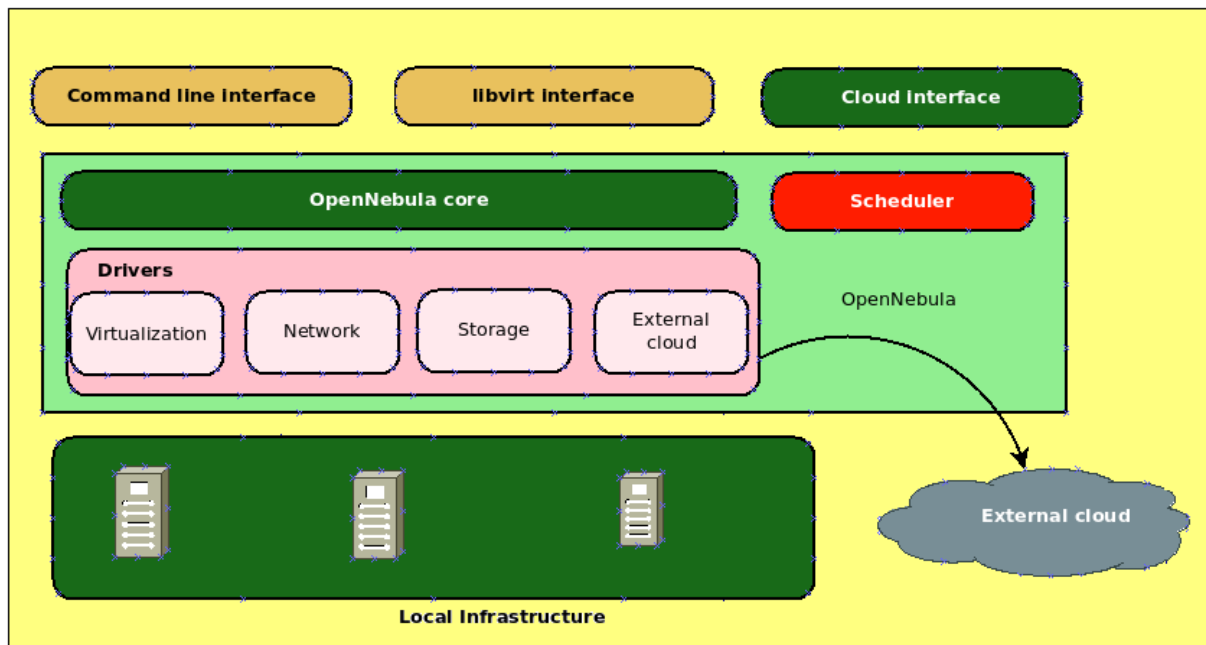


Figure 15. OpenNebula components

- Interfaces and APIs. They manage physical and virtual resources. There are two main interfaces for interaction with OpenNebula: the command-line interface and the Graphical User Interface, also known as “Sunstone”.
- Networking. The Networking interface allows almost any integration in existing data center. It supports VLANs and OpenvSwitch.
- Storage. Several storage systems are supported, such as file system storage (non-shared and shared), distributed network storage or block storage.
- Virtualization. OpenNebula supports the following hypervisors: Xen, KVM and Vmware on the hosts.
- Clusters. Clusters are a pool of hosts that share networking and data stores.

# Chapter 4. Testbed Design

## 4.1 SDN in data centers

As mentioned before, SDN has brought the promise of agility, flexibility and scalability to communication networks, but it also helps to firm up the operation in data centers to face the connectivity demand of users. Providers says that Software-Defined networking can solve lots of the problems that networks have traditionally posed for virtualized environments. SDN applied in data centers can help to bring this environments closer to the cloud, bringing the network towards parity with other aspects of virtualized infrastructure.

Nowadays, the underlying elements in the infrastructure (computing, storage and network) must respond dynamically to the required changes. Computing and storage have reacted to the needs of the companies in a proper way, but not the network. Because of that, SDN is focused on make the network to respond dynamically to the requirements of the companies.

A software-defined data center (SDDC) is an IT installation where infrastructure elements (networks, storage, CPU and security) are virtualized and dedicated as a service. Expectations for IT to deliver applications and services at speed and scale are greater than ever. As an example, the VMware SDDC [21] enables companies to evolve beyond outdated, hardware-centric architectures and to create an automated, easily managed platform that embraces all applications, for fast deployment across data centers and clouds. With an SDDC, users see a predefined list of services available to them and can have these services created instantly upon request while still enabling IT to control and secure these services. This reference architecture delivers a working configuration that provides users with the on-demand access they need while ensuring that IT maintains the control and security it requires.

One of the most significant cases of implantation of SDN is the case with Google [24]. This enterprises has long been a pioneer in distributed computing and data processing. A great computing infrastructure like this requires an advanced data center networking technology. The services of



Google are provided through a data processing center scattered across the five continents, so as to provide redundancy and high availability.

The motivations of Google to implement SDN were the fast growth of Internet bandwidth which led the company to think that they could not keep its scalability, fault tolerance, control and costs with traditional WAN technologies.

Finally, it must be included the SDN architecture for data centers with Huawei called Cloud Fabric 3.0. This architecture is designed to help enterprises to build elastic and scalable data centers without ignoring business needs. Cloud Fabric 3.0 allows a fast implementation of the services and the efficient integration of TIC resources.

In this chapter, it will be described the steps done to design the Software Defined Network. In order to apply all points mentioned in previous chapters, several components have been evaluated and tested before the complete infrastructure was built. First, a comparison among Openstack and OpenNebula have been done, in order to decide which platforms suits best to reach the goals of the project. Secondly, various controllers have been studied, fact that have allowed us to decide which one has better characteristics when developing the project.

## 4.2 Infrastructure used

In order to deploy the testbed design, the infrastructure provided by the Foundation Cénits-COMPUTAEX have been used. Cénits offers its infrastructure and resources to carry out this project since infrastructure and service settings are required. It houses Lousitania Supercomputer [25], a huge shared-memory supercomputer in Spain and Europe. Its specifications are:

### Compute nodes

- 2 HP Integrity SuperDome SX2000: 2x (64 processors/128 cores) = 128 processors/256 cores.
- Itanium2 Montvale @ 1.6 GHz, 18 MB cache.
- 2x 1TB memory per node = 2TB de memory.
- 2x 40x146GB SAS disks = 3,6TB to scratch.
- Suse Linux "SLES 10" operative system.
- High availability: N+1 OLR blowers, N+1 OLR energy suppliers, double energy suppliers, cells OLAR, OLAR I/O cards, ECC in CPUs, memory and all data ways, Dynamic Processor Resilience, Dynamic Memory Resilience (Chip Kill doble) and two ways between switches and cell driver, memory and CPUs.
- Up to 16 physical partitions and 64 virtual partitions.

### Service Nodes

- Login/Development/Managment HPC services.
- 4 x HP Integrity rx2660: 4 cores "Intel Itanium-2 dual-core Montvale (1,6Ghz/18MB "cache built in chip")" ; 16GB DDR-2 memory; 6 x 146 GB SAS disks.

## Management Nodes

- 2 x HP ProLiant DL380-G5: 8 cores “Intel Xeon Quad-Core E5450 (3.0 GHz, 1333 FSB, 80W)”; 8 GB FBD DDR-2 memory; 2 x 146 GB SAS disks.

## Cloud Computing Nodes

- 2 x HP ProLiant DL-380-G7: 2 cores "Intel Xeon Quad Core E5630", 32 GB and 64 GB memory; 2 x 146 GB SAS disks.
- 1 x HP ProLiant DL-380-G5: 2 cores “Intel Xeon Quad Core E5450”, 16 GB memory; 2 x 146 GB SAS disks.
- 2 x HP ProLiant BL465c Gen8 Server Blade, with HP BL465c Gen8 AMD Opteron 6276 (2.3GHz/16-core/16MB) cores, 256 GB RAM/server y 4 x 300GB SAS disks.
- 2 x HP ProLiant BL465c Gen8 Server Blade, with HP BL465c Gen8 AMD Opteron 6276 (2.3GHz/16-core/16MB) cores, 256 GB RAM/server and 4 x 300 GB SAS disks.
- 4 X HP ProLiant BL460c Gen6 Server Blade, with Intel Xeon 5600 (2.93 Ghz), 128 GB RAM/sever and 8 x 300 GB SAS.
- 4 x HP ProLiant BL465c Gen8 Server Blade with 2 x AMD Opteron 6366 HE (1.8GHz/16-core/32MB), which are low power consumption, 128 GB RAM and 2 x 300GB SAS/server.
- 4 x HP ProLiant BL465c Gen8 Server Blade with AMD Opteron 6376 (2.3GHz/16-core/16MB), 128 GB RAM/server and 8 x 300GB SAS.
- 1 x Fujitsu Server PRIMERGY RX350 S8, with Intel Xeon E5 2620v2(2,10GHz/12cores/15MB), 256 GB RAM and 2 x 300GB SAS.

## Computing accelerating units

- 2 x HP ProLiant WS460c G6 Workstation Blade with HP BL460c G7 Intel Xeon E5645 (2.40GHz/6-core/12MB), 96 GB RAM/server, 4 x 300GB SAS and 2 x NVIDIA Tesla M2070Q (448 cuda cores and 6 GB GDR5).
- 1 x Intel Xeon Phi Co-Processor 3120P

## Storage

- Fiberchannel network with multipathing active-active(8 ports x 4 drivers).
- 2EVAs 8100x [(208 FC disks x 450GB) + (128 FATA disks x 1TB)] = 265,6TB.
- 2 DL380-G5 NAS servers running distributed filesystem HP StorageWorks PolyServe.
- HP 3PAR StoreServ 7200 x (14 SAS 10k x 900GB) = 12,5TB

## Backup

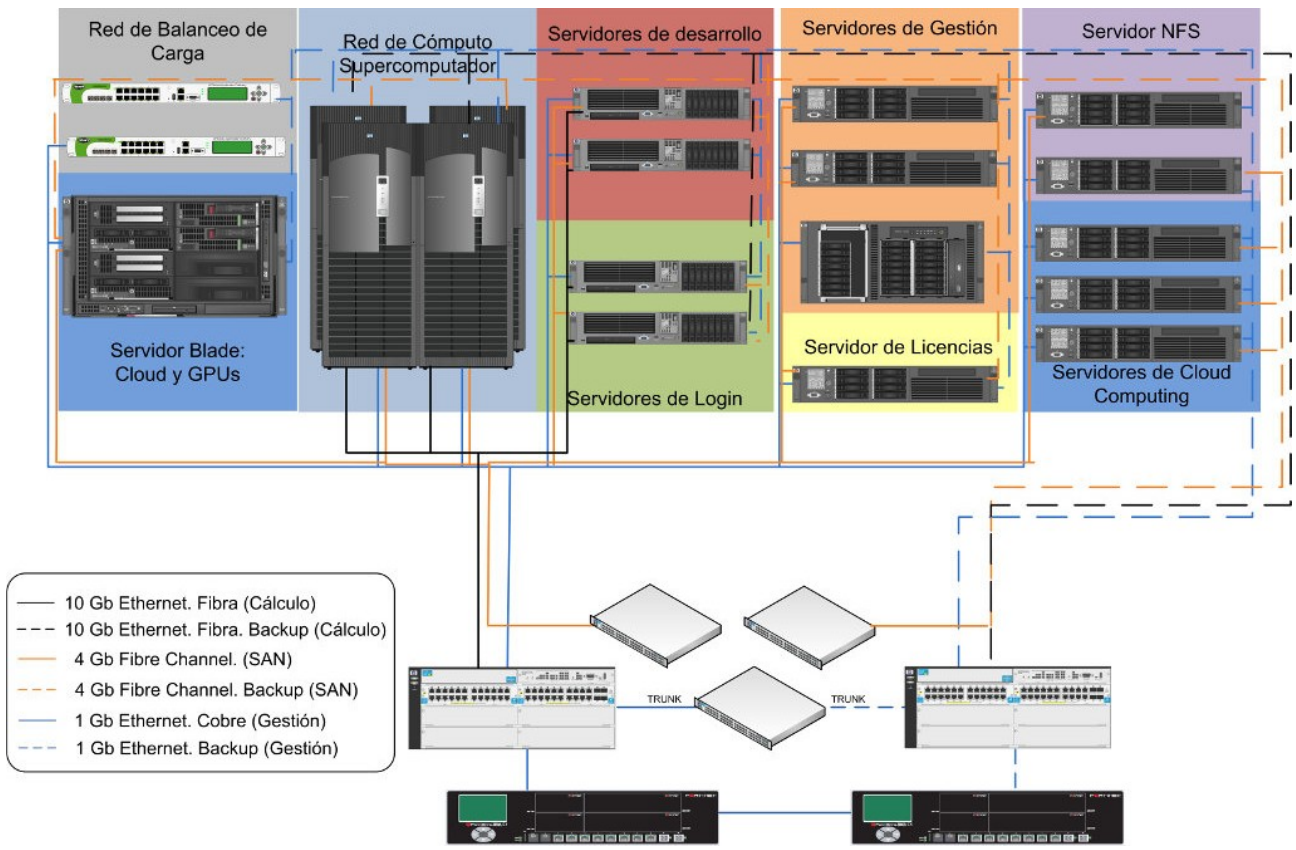
- Backup system over tape library HP Storageworks EML 245e.
- Capacity for 245 tapes of LTO-4 Ultrium 1840 technology, offering capacity of 392 TB in 2:1 compression.
- Managment and unattended planning backup HP StorageWorks DataProtector software.

## Network topology

Center connectivity with outside is solved through a 10Gbps connection with Extremadura Scientific Technology Network, connecting major cities and technology centers in the region.

Service and computation infrastructure:

- Two Fortinet Fortigate 1000C firewall, with 10Gbps connections for perimeter security, VPN, antivirus, intrusion detection and management of bandwidth per connection, configured as a redundant active-passive cluster for high performance.
- Two AppDirector Radware load balancers with load balancing capabilities in applications and remote management capabilities in redundant configuration cluster active-passive.
- Two core switches HP ProCurve 5406zl with 10Gbps switching capacity for computer network and 1Gbps management interfaces for networks and users. They have 48 ethernet ports and 4 fiber channel each.
- Two switches to interconnect network users and servers, one HP ProCurve 2626 and one HP ProCurve 2810-24G, to allow other servers and users connect to the service networks.



Source: [www.cenits.es](http://www.cenits.es)

Figure 16. Lusitania topology

## 4.3 Cloud platforms evaluation

As it have been mentioned in the previous chapter, two cloud platforms (Openstack and OpenNebula) have been studied in order to be able to use it in the infrastructure selected. Down below, the relation between these platforms and Software Defined Networking are described.

### 4.3.1 OpenStack and SDN

When implementing a Software Defined Network with OpenStack, it is of vital importance the Neutron component mentioned before. This component is a key element to manage the network in OpenStack. It constitute an open and free alternative to implement SDN in a cloud environment.

To handle OpenStack Neutron an administration console, named OpenStack Horizon, is available. The behaviour of that console is similar to a hypervisor.

Integration with SDN is expected to support the dynamic nature Neutron high scale, high density and multi-user cloud environments.

OpenStack Neutron, with its plugin-based architecture, provides the capacity of integrating SDN controller in Openstack. This integration provides a centralized management and makes the programmability of OpenStack network easier using APIs.

SDN controllers such as Opendaylight, Floodlight and Ryu can use specific plugins or the ML2 plugins with the appropriate *mechanism drivers*, to allow the communication between Neutron and the SDN controller.

Network Operating Systems, such as Opendaylight and Ryu, are responsible for providing a wide vision of the network (topology) and its current status. The controller is in charge of managing the necessary changes in the network by configuring and monitoring the network elements (physical and virtual). Generally, that changes in the network (and in network devices) come from network applications using northbound APIs.

Using this kind of OpenStack integration, Neutron and SDN controllers, changes in the network and in network devices, are triggered from OpenStack user, which are transferred in Neutron APIs and are managed via neutron plugins and its corresponding agents, running in SDN controllers. For example, OpenDaylight interact with Neutron using the ML2 plugin located in Neutron network node through a REST API using northbound communication.

## 4.3.2 OpenNebula and SDN

OpenFlow controller is a network controller that OVS switches can use to enforce network rules. This enables the OpenNebula server to send commands to the OpenFlow controller to construct the appropriate OpenFlow rules and install them on the networking hardware in the cloud.

As it has been mentioned before, the networking component of OpenNebula supports OpenvSwitch. Taking in consideration this support, it has been given the possibility for a programmable network and a merge of OpenNebula networking paradigm with the area of software defined networking.

Although OpenvSwitch can be managed by many SDN controllers, the network component in OpenNebula does not allow the integration among them, fact that limits the opportunity to develop more customized Software Defined Networks in order to allow a greater management of the network.



### 4.3.3 OpenNebula vs OpenStack

Having studied both cloud computing platforms, a comparative can be made in order to take the final decision about which one is the most appropriate in the development of the project. Therefore, table 3 is shown.

	<b>OpenStack</b>	<b>OpenNebula</b>
<b>Source</b>	Fully open-source	Fully open-source
<b>Code</b>	Apache v2.0	Apache v2.0
<b>Development model</b>	Public	Public
<b>Developer Engagement</b>	Contributor license agreement	Contributor license agreement
<b>Governance Model</b>	Foundation	Benevolent dictator
<b>API</b>	OpenStack API	Amazon API
<b>Production Readness</b>	Only available through any of the several vendor specific “stacks”	Enterprise-ready and direct support from developers
<b>Integration with SDN controller</b>	Yes	No

Table 2 . Comparative OpenStack and OpenNebula

Once both platforms have been evaluated, it was concluded that the best option when developing a software defined network is OpenStack. The decision was held based on the flexibility of the controller (main element of the network), since it allows to realize changes in the configuration of the controller according to the needs of the administrator.

During the development of the project, several ways to install OpenStack have been tested:

- **Packstack.** A command line utility that uses modules to support rapid deployment of OpenStack on existing servers over an SSH connection. Packstack is suitable for deploying both single node and more complex multi-node installations.
- **Devstack.** It is a shell script used to deploy a complete OpenStack development environment. It installs all the dependencies required to run OpenStack.

- **Ubuntu Cloud.** Ubuntu offers all software infrastructure, tools and services needed to build our own cloud OpenStack. Ubuntu OpenStack is a totally integrated combination of Ubuntu Server and OpenStack. It makes possible to deploy a cloud structure based on a reference architecture.

Having assessed all the options mentioned before, it has been reached to the conclusion that the best options to install OpenStack are DevStack and PackStack. Ubuntu Cloud was ruled out due to its difficulty during the installation since it is necessary to be able to use more number of nodes than the one which are available to realize the project.

Packstack has been tested over the operating system Centos and Devstack over Ubuntu 14.04. Both ways reached to the objective of having the platform ready to be used. Having OpenStack installed, the next step done was the evaluation of SDN controllers which can be integrated with this platform.

## 4.4 OpenStack integration with SDN controllers

SDN controllers such as OpenDaylight, Floodlight and Ryu can use specific plugins or the ML2 plugins with the appropriate *mechanism drivers*, to allow the communication between Neutron and the SDN controller.

Network Operating Systems, such as OpenDaylight and Ryu, are responsible for providing a wide vision of the network (topology) and its current status. The controller is in charge of managing the necessary changes in the network by configuring and monitoring the network elements (physical and virtual). Generally, that changes in the network (and in network devices) come from network applications using northbound APIs.

Using this kind of OpenStack integration, Neutron and SDN controllers, changes in the network and in network devices, are triggered from OpenStack user, which are transferred in Neutron APIs and are managed via neutron plugins and its corresponding agents, running in SDN controllers. For example, OpenDaylight interact with Neutron using the ML2 plugin located in Neutron network node through a REST API using northbound communication.

There are several integration options between OpenStack with SDN controllers, being possible to deploy different permutations and combinations of the settings listed below:

- Not virtualized. Complete controller instance running in a single system (physical machine).
- Virtualized. Controller instance running in a virtualized environment (such as VM).
- Integrated. All SDN controller functions running under a single instance.
- Distributed. SDN controller functions are distributed.
- Simple/redundant. A redundant controller for the network.
- Hierarchical. Controller hierarchy, possibly with client-server relationships among them.

## 4.4.1 Evaluation of SDN controllers

Once Openstack have been chosen as the cloud platform to be used in the project, the election of a controller is necessary to complete the network. Two open source controllers in terms of their usage and its integration with OpenStack have been analysed here: Opendaylight and Ryu. In table 4, a comparison of the three controllers is shown.

	<b>Ryu</b>	<b>Opendaylight</b>
<b>OpenFlow support</b>	OF v1.0, v1.2, v1.3	OF v1.0
<b>Virtualization</b>	Mininet and OpenvSwitch	Mininet and OpenvSwitch
<b>Language</b>	Python	Java
<b>REST API</b>	Yes	Yes
<b>Graphical interface</b>	Web	Web
<b>Operating sytem support</b>	Linux	Linux, MAC OS and Windows
<b>Open-source</b>	Yes	Yes
<b>Documentation</b>	Medium	Medium
<b>Integration with OpenStack</b>	Yes	Yes

*Table 3. SDN controllers comparative*

### OpenDaylight

As it can be seen in the previous table, both controllers provides good features to be used. The key point in the selection of the proper controller is the integration with OpenStack. During the development of this project, integration with both controllers have been studied.

This controller provides a robust array of services to enable a wide breadth of applications. OpenDaylight can deliver the benefits of SDN in order to control Ethernet switches using OpenFlow protocol. As shown in figure 17, it is composed of a number of different modules that can be combined as needed to meet the requirements of a given scenario.

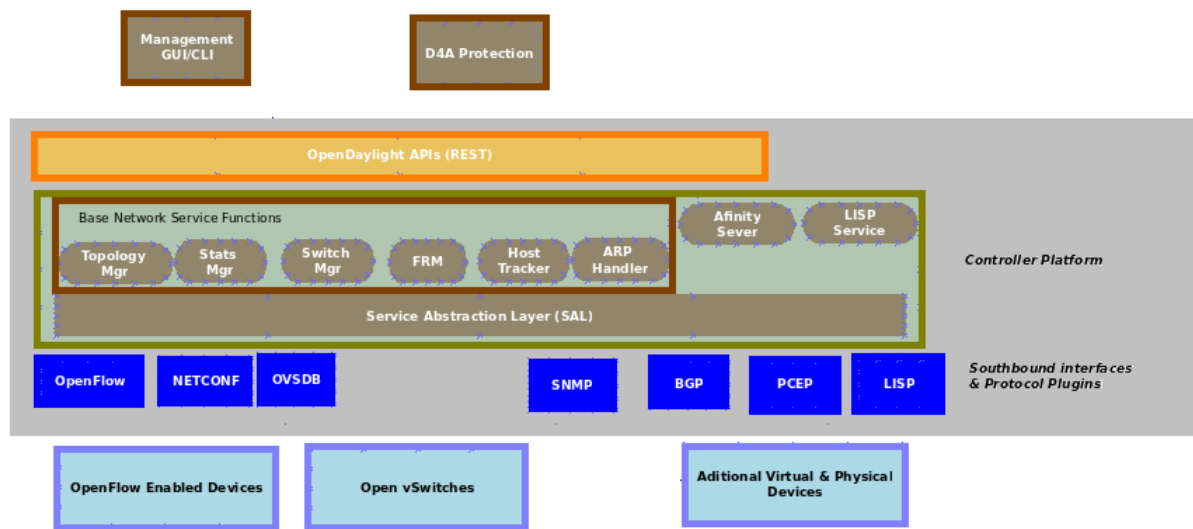


Figure 17. Opendaylight architecture

When installing OpenDaylight, it is important to select a proper distribution of the controller. There are two main distributions: basis distribution and Helium. The first one offers the essential features to run the controller. It works with OSGi (Open Service Gateway initiative), which lets Java programming and install, launch and stop Java packets in an easy way. Furthermore, Helium distribution is developed with a set of complements whose objective is to interact with the controller in a transparent form. It works with Karaf, a generic platform that provides high-level functions and services.

Both distributions are easy to install, since the code is available in the Internet. It is just needed to download a .zip file and unzip it in the desired directory.

Having used both distributions, it has come to the conclusion that basis distribution provides better results due to its simplicity when running an application. Although Helium opens up more complex functions, it is difficult to manage them once the controller is installed. Sensing this, during the project the basis distribution has been used.

When the controller is installed, it can be run using Mininet. A topology have been started indicating the use of a remote controller (OpenDaylight). The command used is the following:

```
$sudo mn --topo single, 3 -mac -switch
ovsk,protocols=OpenFlow13 -controller remote
```

A mininet terminal appears. If a ping between two of the hosts is realized, it fails because there is no intelligence in the switch to learn MAC addresses of each host and forward traffic to the correct switch ports. Because of this, intelligence must be added; so the application Hub/Learning switch is built using the following commands.

```
$ cd SDNHub_OpenDaylight
SDNHub_OpenDaylight_Tutorial$ mvn install -nsu
```

These commands use Apache Maven to build the code of the tutorial, which is based in pom.xml. The packet tutorial\_L2\_forwarding depends on the OpenDaylight controller. Once the code is built, the sample learning-switch application had been executed using the following commands:

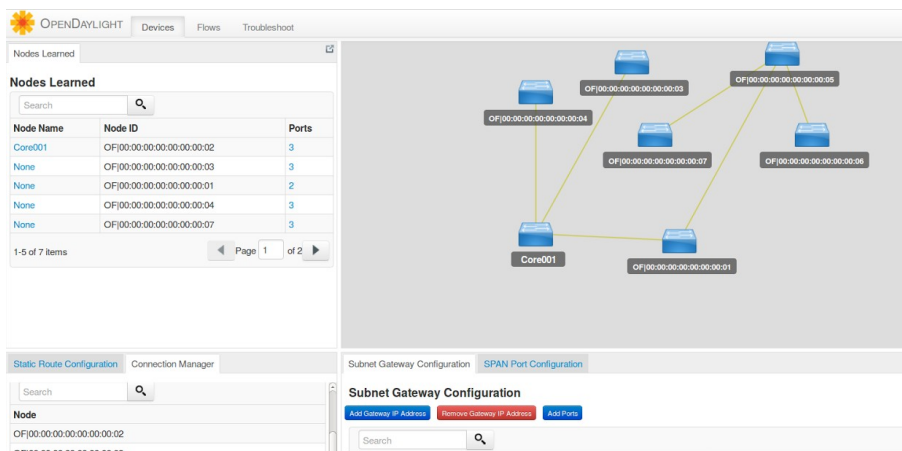
```
SDNHub_OpenDaylight_Tutorial$ mvn install -pl
distribution/.opendaylight- osgi-mdsal -also-make
-DskipTests -DskipIT -nsu
```

```
SDNHub_OpenDaylight_Tutorial$ cd
distribution/.opendaylight- osgi-
mdsal/target/distribution- osgi-mdsal-1.1.0- SANPSHOT-
osgipackage/.opendaylight
```

```
SDNHub_OpenDaylight_Tutorial/distribution/.opendaylight-
osgi-mdsal/target/distribution- osgi-mdsal-1.1.0-
SANPSHOT- osgipackage/.opendaylight$ ./run.sh
```

Once the above commands are executed, OpenDaylight controller starts. As it has been mentioned before, OpenDaylight has a web graphical interface, which can be seen in figure 18, by typing the following:

`http://localhost:8080`



*Figure 18 .Opendaylight graphical interface*

Is because of the modular nature of OpenDaylight, it makes use of the OSGi framework that allows to write a bundle which can run and interact with other bundles in OpenDaylight. There are a few bundles which are relevant to the OpenStack integration efforts: NeutronAPIService, OVSDB and OpenFlow. Each of those bundles provides a necessary component in the OpenStack integration. The NeutronAPIService provides an abstraction of the Neutron APIs into OpenDaylight. The OVSDB and OpenFlow bundles permit the creation and deletion of tunnel ports, flow programming for ports and bridge creation.

As it have been mentioned before, two ways of OpenStack installation have been made. In order to make the integration among Opendaylight and Openstack, Packstack was discarded because it was necessary to use Helium distribution, which has not been used to install OpenDaylight. As a consequence of this fact, Devstack was used to integrate the controller with OpenStack. In order to make the integration, the configuration file (local.conf) of Devstack had to be modified. The following data had been set in the local.conf file:

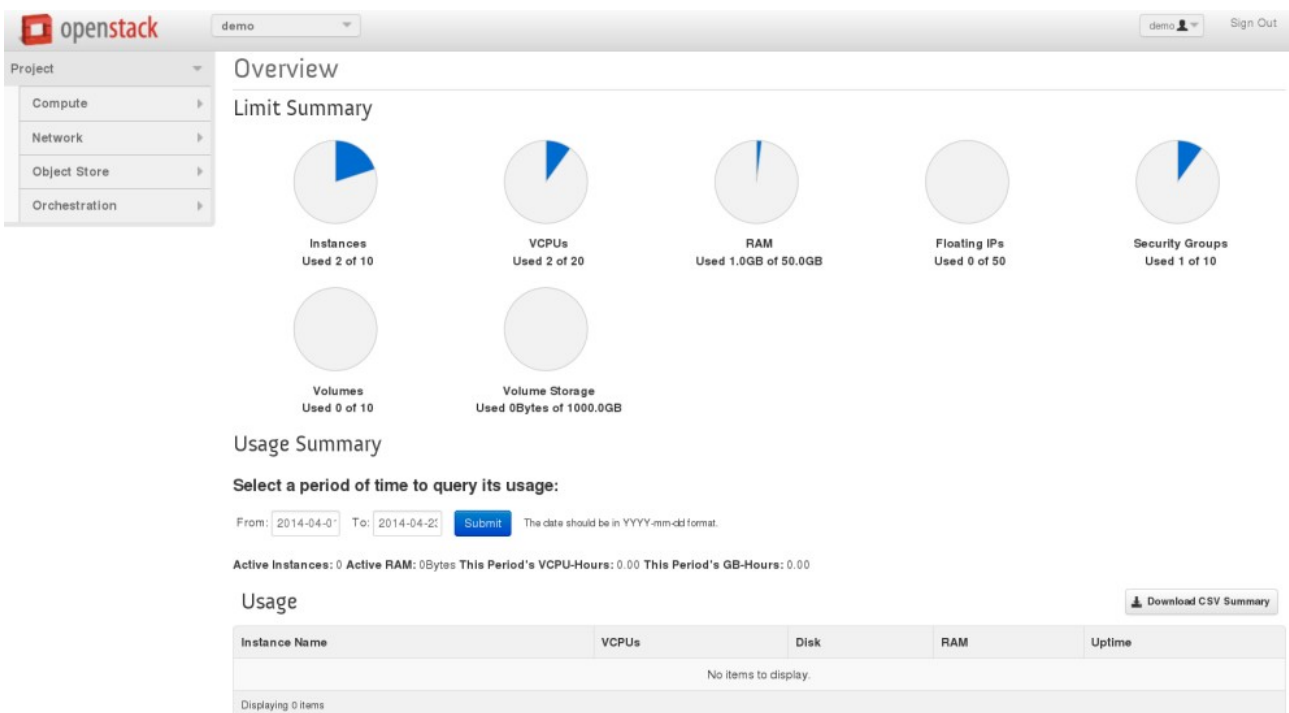
- **HOST\_IP.** IP of the host in which the controller is installed. In this case is 192.168.60.24
- **ADMIN\_PASSWORD.** This password is used for the admin and demo accounts set up as Openstack users. This credentials are necessary to login to the Openstack GUI.
- **MYSQL\_PASSWORD.** It is used in case the user needs to look at the database directly.
- **RABBIT\_PASSWORD.** It is used by messaging services used by the node.

- SERVICE\_PASSWORD. OpenStack services (Nova, Glance, etc) use this password to authenticate with KeyStone.

Once the configuration file had been modified, Devstack had been run using the command shown below. This process can take a few minutes.

```
$ ./stack.sh
```

When this process finishes, it is only necessary to write the IP address of the host where the installation have been made and the OpenStack Dashboard, shown in figure 19, appears.



Source: OpenStack.org

Figure 19. OpenStack Dashboard

Although OpenStack worked correctly, the integration with Opendaylight controller had not been completely correct. It is this worrying fact, which makes that the integration with other SDN controller had been tested.



## Ryu

Ryu controller is a software-based component defined within the framework of Software Defined Networking. It gives software components (API) that helps to create control and management network applications. Ryu supports several management network devices protocols like, for example OpenFlow, Netconf, OF-config, etc. It is implemented in Python. Like any other controller, Ryu can create and send OpenFlow messages, listen to asynchronous events and handles incoming packets. Its architecture is shown in figure 20.

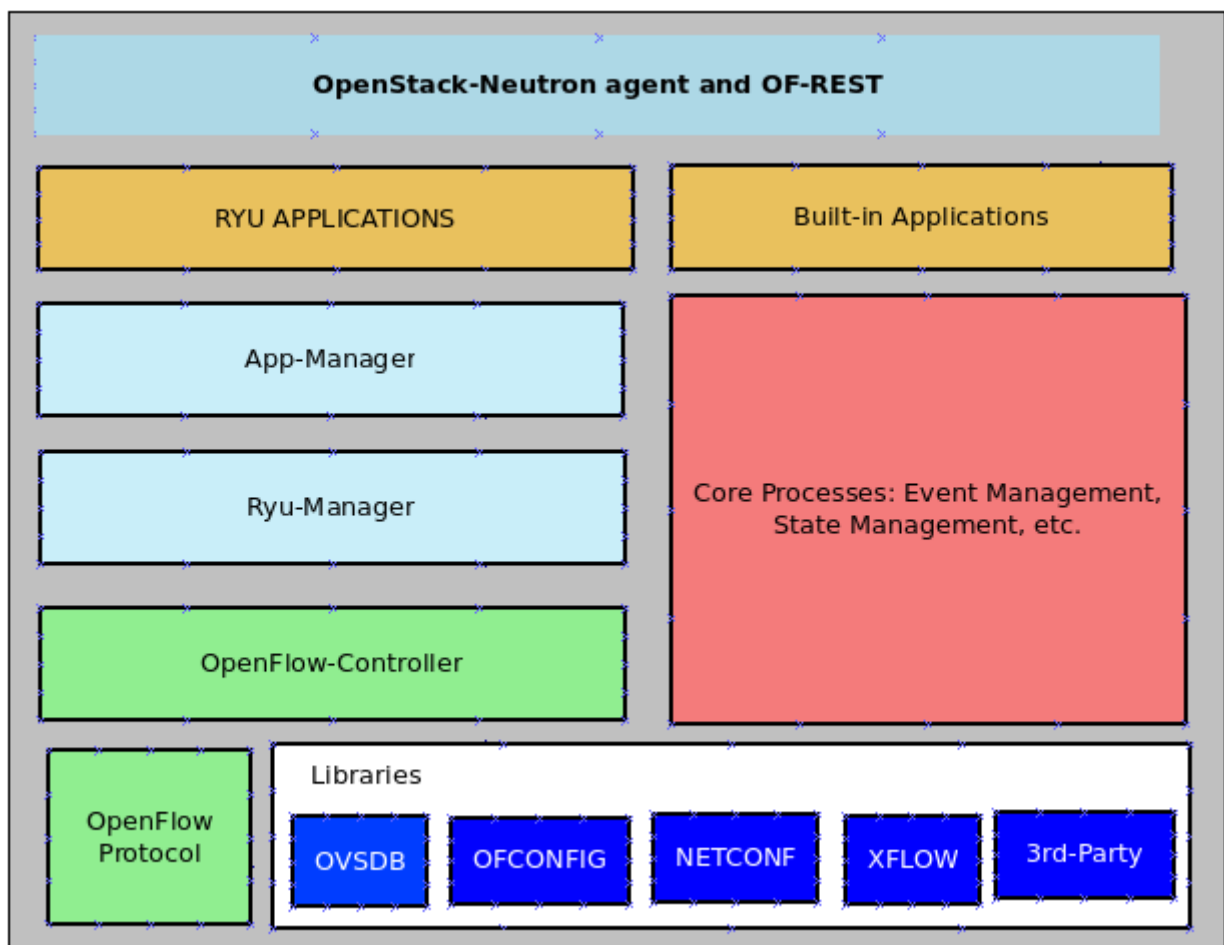


Figure 20. Ryu architecture

As it can be seen in previous figure, some of the important components of the architecture are:

- Libraries. Ryu has an impressive collection of libraries, ranging from support for multiple southbound protocols (OF-Config, OpenvSwitch Database Management Protocol, NETCONF, Xflow and other third party protocols) to various network packet processing operations.
- OpenFlow controller. One of the key components of the Ryu architecture is the OpenFlow controller, which is responsible for managing the OpenFlow switches to configure flows, events, etc. Table 4 shown below summarizes the Ryu OpenFlow protocol messages, structure and corresponding API.

<b>Controller to Switch Messages</b>	<b>Asynchronous Messages</b>	<b>Symmetric messages</b>	<b>Structures</b>
Handshake, switch-config, flow-table-config, modify/read state, queue-config, packet-out, barrier, role-reques	Packet-in, flow-removed, port-status, and Error.	Hello, Echo-Request & Reply, Error, experimenter	Flow-match
send_msg API and packet builder APIs	set_ev_cls API and packet parser APIs	Send and Event APIs	

Table 4. Ryu OpenFlow protocol messages

- Ryu-Manager. The Ryu manager is the main executable. It is the main component of Ryu applications. When it is run, it listens to an IP address and the specified port (6633 by default). Thus, any OpenFlow switch is able to connect with Ryu manager.
- Core processes. They include the components which include event management, state management, etc.
- Ryu-applications. Ryu is distributed with multiple applications that are single-thread entities, which implement various functionalities. Among these applications is particularly worthy of mention the simple\_switch application, which can be used up to the latest version 1.4 of

OpenFlow protocol. Its usability will be explained in next chapter. Ryu applications can be run and configured by passing a configuration file to the Ryu manager.

- OpenStack-neutron agent and OF-REST. Ryu includes an OpenStack Neutron plugin which supports both GRE-based overlay and VLAN configurations. It also supports a REST interface to its OpenFlow operations.

During the development of this project, the integration between OpenStack and Ryu have been made. The process was done using Ryu plugin with Packstack configuration. As it is shown in figure 21, this plugin act as SDN local driver and configure Open vSwitch using the protocol stored in the Open vSwitch DataBase.

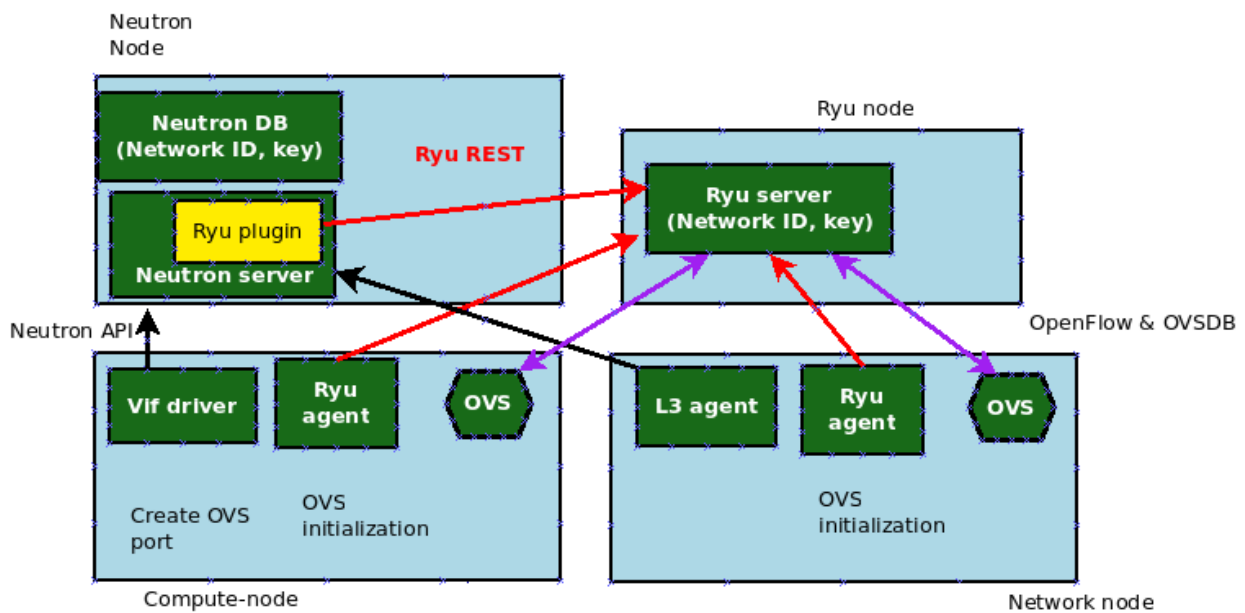


Figure 21. Integration ryu-plugin with OpenStack

Before making the integration, the steps that must be followed to install Ryu are mentioned. Its is only necessary to enter the following commands:

```
sudo pip install ryu
git clone git://github.com/osrg/ryu.git
cd ryu
sudo python ./setup.py install
```

Integrating Ryu with OpenStack is relatively easy, simply setting the OFAgent agent with use Ryu as a OpenFlow library, with OpenStack. This agent is not present in Neutron component files, so it will be necessary to install it separately from OpenStack. To this end, the following command has to be entered:

```
sudo pip install networking-ofagent
```

Once downloaded this agent, it will be needed to change configuration files of Neutron component of OpenStack. Changes will have to be done in ml2\_conf.ini and ml2\_conf\_ofa.ini.

In ml2\_conf.ini mechanism drivers of ofagent and l2population will be specified:

```
[ml2]
mechanism_drivers = ofagent,l2population
```

In ml2\_conf\_ofa.ini file will have the following configuration when making the integration:

```
[ovs]
enable_tunneling=True
local_ip=<physical-net-ip>

[agent]
tunnel_types = gre
```

Once the new configuration is done, the next step will be to configure the integration bridge, named br-int, which allows connections to virtual machines using the commands:

```
ovs-vsctl add-br br-int  
  
ovs-vsctl set-controller <br-int> tcp:<ip addr>[:<port:  
default 6633>], where
```

- ip addr: IP address of the device where the controller is hosted
- port: listen port, the default port is 6633

Lastly, the plugin OpenvSwitch (ovs\_neutron\_plugin.ini) is configured:

```
ovs_neutron_plugin.ini  
  
tunnel_types =gre  
l2_population = True  
arp_responder = True
```

Comparing both methods of integration, it has been concluded that the best solution is using Ryu with the Packstack installation of OpenStack. The reason for choosing this option is based on the fact that the integration with OpenDaylight is not reliable due to the controller does not work properly and the integration is very complex.

The integration with Ryu was relatively easy and more reliable than the one with OpenDaylight. Opendaylight required a complex process to be integrated with OpenStack and does not always work properly. On the contrary, Ryu plugins provides an easy way to use Ryu as the controller of the platform and, although there were some problems at first time when monitoring the OpenFlow traffic, once those issues were solved, its performance was suitable.

## 4.5 Testbed design

The design of the testbed have been done over the cloud computing node HP ProLiant BL465c Gen8 Server Blade with HP BL465c Gen8 AMD Opteron 6276 processors. The operating system used is CentOS 7.

Taking into consideration the key points mentioned before, the OpenStack installation chosen was PackStack using Ryu as controller of the platform.

The network topology used consists of two virtual network linked by a router and nine virtual machines using Ubuntu 14.04 as its operating system, as it can be seen in figure 22.

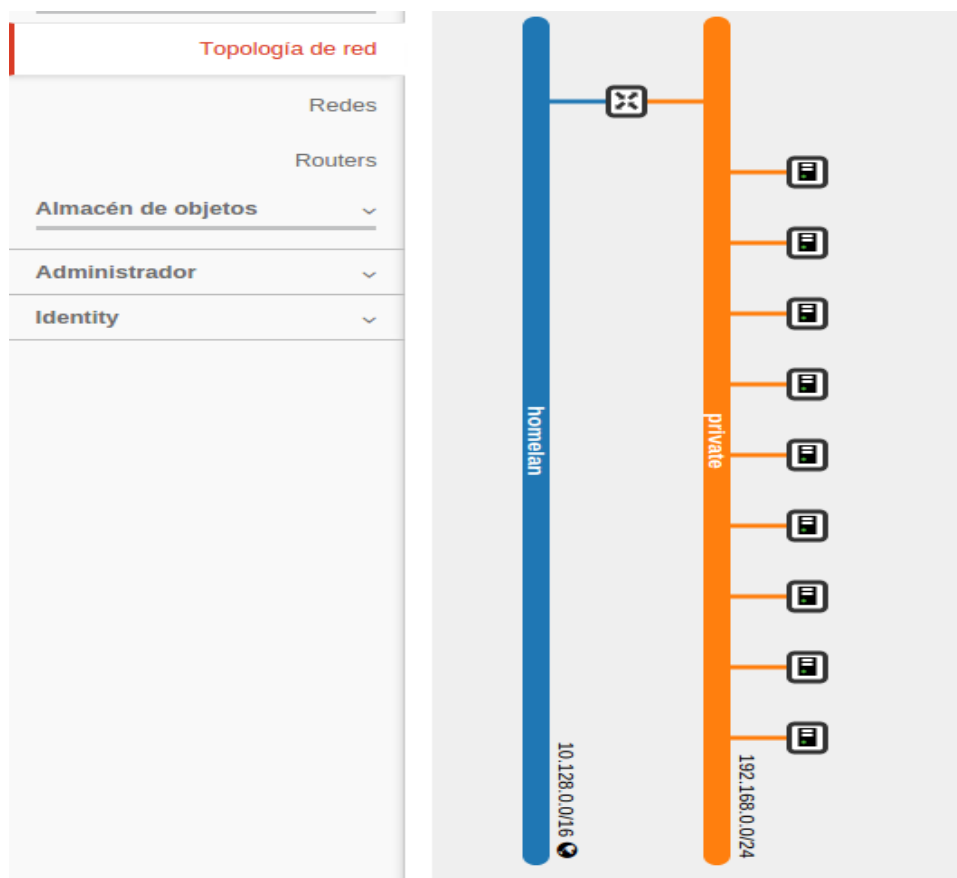


Figure 22. Network topology

In order to create the virtual machines used in the project, it was necessary first to upload an Ubuntu 14.04 image to the cloud. It is a preconfigured system image that it was used to create the instances. Instances in OpenStack are clones of images which are built according to the user requirements. Virtual machines has been configured with an Ubuntu 14.04 image. The creation of the virtual machine is quite simple. It is only necessary to upload the Ubuntu image and then, create the number of instances desired.

The process of creating the instances, have been done using OpenStack Dashboard. Instances must have internet access, but the DNS server had to be changed in order to resolve the hosts names. Initially, the user and password to access to the instance is not known, so a key pair had been used. The Keypairs section of Access & Security in OpenStack Dashboard allows importing a key pair, the public key is stored and the private one is downloaded. Once this steps have been done, it is possible to access to the instances; using the following command:

```
ssh -i clave.pem ubuntu@<virtual_machine_ip>
```

When launching an instance the following data must be set:

- Name of the instance.
- Flavor. It can be tiny, small,etc
- Keypair.
- Group security.
- Number of instances.

Once these data are set, the instances started to be launched. When instances are completely operative, private IP addresses are assigned. These IP addresses are used in the internal communication in the network. In order to access to the instances from external equipment, floating IP addresses are assigned. While private IP address are assigned automatically, floatings IP

addresses had to be assigned manually. In the section Access & Security these type of addresses can be assigned. As a consequence, each instance has a private and a floating IP address assigned, which are shown in table 5.

<b>Instance</b>	<b>Private IP address</b>	<b>Floating IP address</b>
u1	192.168.0.4	10.128.200.202
u2	192.168.0.5	10.128.200.203
u3	192.168.0.6	10.128.200.204
u4	192.168.0.7	10.128.200.205
u5	192.168.0.8	10.128.200.206
u6	192.168.0.9	10.128.200.207
u7	192.168.0.10	10.128.200.208
u8	192.168.0.11	10.128.200.209

*Table 5. IP addresses of OpenStack instances*

Although there are nine virtual machines available, only u1 and u2 had been used. As it can be observed, the virtual machines does not have a graphical interface. In order to make the analysis of the different tests to be realized over this infrastructure easier, the VNC tool was installed in each one of the virtual machines. This tool provides a graphical interface for each one of the virtual machines.

VNC execution will be done using the following commands:

For u1:

In a u1 terminal, in the /etc/init.d directory, the command `vncserver:1` is executed.

In a terminal in the computer, the command `vncviewer 10.128.200.202:5901` is executed.

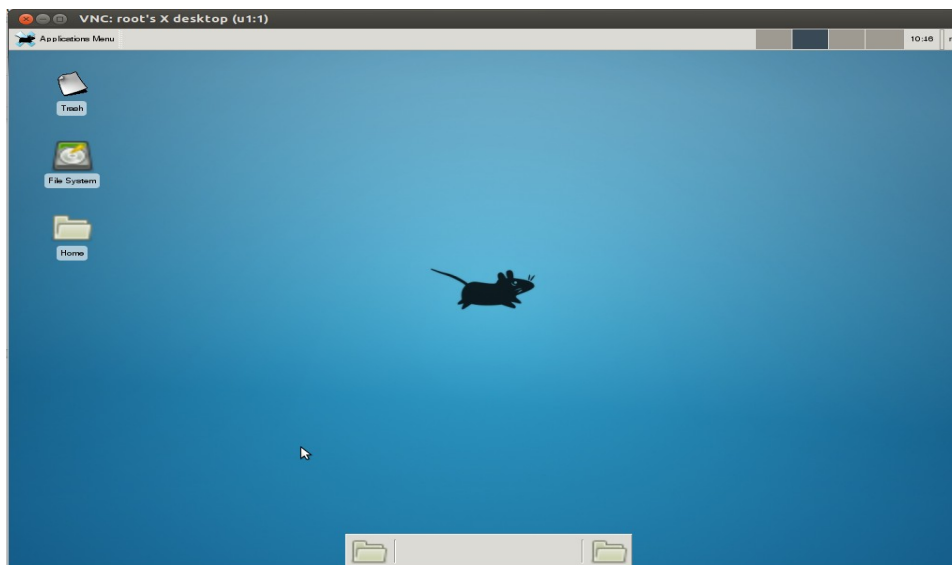


For u2:

In a u2 terminal, in the /etc/init.d directory, the command `vncserver:2` is executed.

In a terminal in the computer, the command `vncviewer 10.128.200.203:5902` is executed.

In figure 23, the graphical interface provides by VNC is shown.



*Figure 23. VNC graphical interface*

Once the process explained before was done, it is possible to accomplish the tests to evaluate the performance of a Software Defined Network. The tested performed, will be explained in the following chapter.

# Capítulo 5: Resultados

En este capítulo se realizarán pruebas que permitan comprobar el rendimiento de la red, como, por ejemplo, la medida del ancho de banda y pruebas de conectividad. Además, se incidirá en el estudio de los distintos mensajes OpenFlow existentes.

Debido a que el controlador elegido fue Ryu, en primer lugar se ha llevado a cabo un benchmark para estudiar su rendimiento. A continuación, se ha utilizado la herramienta Mininet con el objetivo de construir una red emulada y poder hacer pruebas con tráfico sintético. Por último, se han aplicado los conocimientos y pruebas usando un entorno real, usando la plataforma cloud OpenStack con el controlador Ryu integrado, lo que nos permitirá hacer pruebas con tráfico real.

Tanto el estudio del rendimiento y la utilización de Mininet se han realizado con un ordenador personal con las siguientes especificaciones:

- Procesador: Intel Core i3-4005U(1.7GHz, 3MB)
- Memoria RAM: 4GB DDR3L SODIMM (1x4GB) Max 8 GB
- Disco Duro: 500GB (5400 rpm S-ATA)
- Sistema Operativo: Ubuntu 12.04 LTS

Las pruebas con tráfico real se han hecho en el testbed diseñado en el capítulo anterior.

# 5.1 Herramientas y metodología usada

Se describen a continuación las principales herramientas utilizadas, tanto en el desarrollo de la red como a la hora de realizar pruebas para medir el rendimiento de la misma.

## **Mininet**

Mininet [26] se define como una plataforma que permite emular redes definidas por software de forma virtual y puede ser programada de forma externa, características que ofrecen escalabilidad, interactividad y flexibilidad. Es un emulador de red de código abierto que permite realizar prototipos de red con componentes virtuales (hosts, switches, controladores y enlaces) de forma rápida. Estos componentes se crean utilizando software en lugar de hardware.

Esta herramienta utiliza virtualización ligera para hacer que un solo sistema tenga el aspecto de una red completa. Este tipo de virtualización es la que nos permite tener múltiples procesos en diferentes espacios de usuario aislados unos de otros, cada uno con su configuración, pudiendo tener así escalabilidad de hasta cientos de nodos.

Esta herramienta permite llevar a cabo tareas de investigación, desarrollo, aprendizaje, prototipado, pruebas, eliminación de fallos y otras tareas para beneficiarse para tener una red experimental completa en un ordenador con comportamiento muy similar al de una red real.

### *Mininet y Redes Definidas por Software*

Mininet es la herramienta de simulación y testeo por excelencia de redes SDN ya que permite crear fácilmente redes definidas por software virtuales que consisten en un controlador OpenFlow, una red Ethernet con múltiples switches Ethernet compatibles con OpenFlow y múltiples hosts conectados a los switches.

La herramienta puede simular los controladores POX y NOX y además, permite configurar

controladores remotos como Floodlight, OpenDaylight o Ryu.

Esta posibilidad de hacer pruebas con diferentes controladores hace que esta herramienta esté pensada para evaluar la capacidad de los distintos controladores SDN, así como testear en entornos de prueba el comportamiento de funciones virtualizadas de red (NFVs).

### *Uso de mininet*

Como se ha mencionado anteriormente, Mininet permite crear, interaccionar, personalizar y compartir prototipos de redes basadas en software y proporcionan un camino fácil para ejecutarlas sobre hardware.

Los comandos básicos para usar mininet son:

- *sudo mn*: permite entrar en el entorno mininet. El prompt cambiará a mininet>. Este comando admite una gran variedad de parámetros por lo que es habitual ingresar al entorno mininet utilizando el siguiente comando:

```
sudo mn [parámetros]
```

- *exit*: salir de mininet. Tras salir de mininet es conveniente realizar un borrado del entorno, de modo que está listo para volver a ser utilizado, sin residuos de anteriores usos. Para realizar el borrado se utiliza el comando:

```
sudo mn -c
```

Para mostrar una pequeña ayuda de los diferentes comandos y opciones que soporta mn se utiliza el comando

```
sudo mn -h
```

Generalmente se invoca la topología completa que queremos implementar. Existen una serie de topologías que se pueden especificar directamente en la llamada a `mininet.problema`

- `sudo mn` ó `sudo --topo minimal`. Esta topología consta de un controlador, un switch OpenFlow y dos hosts conectados al switch.
- `sudo mn --topo single, [n]`; donde `n` es el número de hosts. Se trata de una topología con `n` hosts conectados a un switch.
- `sudo mn --topo linear, [n]`; donde `n` es el número de switches. Consta de `n` switches conectados entre sí. En cada switch hay un host conectado.
- `sudo mn --topo tree, depth=n`. Implementa una topología en árbol, con una profundidad de `n` niveles. El nivel superior es de un switch, y cada switch conecta con dos en el nivel inferior. En el nivel `n`-ésimo, cada switch conecta dos hosts.

Además de las topologías mencionadas, se pueden implementar topologías personalizadas que han de implementarse mediante scripts en Python.

Una vez que se tiene una topología a simular, existen una serie de comandos para mostrar información de la red.

- `mininet>nodes`. Muestra información sobre los nodos disponibles.
- `mininet>net`. Muestra información sobre la red, nodos y enlaces disponibles. Es posible ejecutar comandos en cada nodo, como por ejemplo:
- `mininet> dump`. Muestra información sobre las direcciones IP de cada nodo.

Es posible ejecutar comandos en cada nodo, como por ejemplo:

```
mininet>h1 ifconfig -a, que ejecutará el comando ifconfig -a desde el host h1.
```

También es importante destacar que el CLI de Mininet admite una serie de tests básicos, con el objetivo de probar la conectividad entre los hosts virtuales:

- `mininet>h1 ping -c2 h2`. Este comando envía dos ICMP-request del host 1 al host 2.
- `mininet>pingall`. Se lleva a cabo una prueba de conectividad completa. Se enviarán mensajes ICMP (ping) entre todas las parejas de hosts de la red.

Además de las mencionadas anteriormente, existen otras aplicaciones que se pueden ejecutar desde los hosts. De hecho, cualquier aplicación presente en Linux y que exista en la máquina virtual puede ser ejecutada. Por ejemplo, se puede instalar un servidor web en un hosts y hacer peticiones al puerto 80 desde otro. Además, se incorpora la herramienta `iperf`, que, como se explicará más adelante, establece sesiones TCP o UDP para calcular el ancho de banda entre dos hosts. Su sintaxis es:

```
mininet>iperf src dst,
```

si se obvian `src` y `dst` se medirá entre el primer y último host.

Otra característica importante es la posibilidad de acceder a un terminal para componentes de la red. Mininet permite abrir un emulador de terminal Xterm. Esto se consigue añadiendo el parámetro `-x` a la línea `$sudo mn ...`. Otra forma de ejecutar el comando es:

```
mininet> xterm [nombre del componente]
```

# Iperf

Iperf [27] es una herramienta de código abierto que se utiliza para hacer pruebas en redes informáticas y medir el rendimiento de una red. Sirve para medir el ancho de banda y el rendimiento de una conexión entre dos host. La calidad del enlace puede ser examinado teniendo en cuenta lo siguiente:

- Latencia (tiempo de respuesta). Puede medirse mediante el comando ping.
- Jitter (variación en la latencia). Puede medirse ejecutando una prueba con TCP.
- Pérdida de datagramas. Puede medirse con Iperf ejecutando una prueba con UDP.

El funcionamiento de Iperf consiste en crear flujos de datos TCP y UDP para medir el rendimiento de la red. Permite ajustar varios parámetros para personalizar las pruebas realizadas y así poder optimizar y ajustar la red en función de nuestros intereses. Puede funcionar bien como cliente o bien como servidor. Si utilizamos UDP, Iperf permite al usuario especificar el tamaño de los datagramas y proporciona resultados de rendimiento y de paquetes perdidos. Si, en cambio, se utiliza TCP, mide el rendimiento de la carga útil. El ancho de banda se mide mediante pruebas con TCP.

Como se ha mencionado, podemos configurar tanto el cliente como el servidor. A parte de la opción -s que deja a Iperf a la escucha, se puede usar los siguientes parámetros:

- -D: como servicio.
- -R: eliminar servicio
- -u: para crear flujos UDP
- -P: número de conexiones simultáneas
- -m: muestra la MTU
- -w: especifica el tamaño de ventana. Es muy útil a la hora de calcular el tamaño de ventana más óptimo según las medidas de ancho de banda.
- -f [bkmBKB]: muestra resultados en bits/s, kilobits/s, megabytes/s, Bytes/s, KiloBytes/s, MegaBytes/s (s=segundos).

Como cliente, el comando más básico es -c IP pero podemos establecer otras opciones como:

- -f [bkmBKB]: muestra resultados en bits/s, kilobits/s, megabytes/s, Bytes/s, KiloBytes/s, MegaBytes/s (s=segundos).
- -w: especifica el tamaño de ventana. Es muy útil a la hora de calcular el tamaño de ventana más óptimo según las medidas de ancho de banda.
- -m: muestra la MTU
- -T: especifica el tiempo de vida (ttl)
- -i: especifica el intervalo (medido en segundos), en el cual se volverá a realizar la medición.
- -t: tiempo de duración de la transmisión (segundos)
- -p: puerto en el que escucha el servidor
- -u: envío de UDP. Podemos medir también pérdida de paquetes.

Iperf muestra como salida un informe con marcas de tiempo con la cantidad de datos transmitidos y el rendimiento medido. Iperf es una herramienta multiplataforma, lo que permite que funcione en cualquier red y devolver medidas de rendimiento estandarizadas.

Existe también una interfaz gráfica para Iperf, llamada Jperf, que permite medir y representar de forma gráfica el rendimiento. Permite las mismas operaciones que Iperf, pero sin necesidad de usar comandos, todo se hace a través de la interfaz.

## **Cbench**

Herramienta que trata de cuantificar el rendimiento de un controlador. Cbench [28] emula varios switch OpenFlow conectados al controlador. Cada uno de los switch envía un número determinado de peticiones de flujo al controlador y computa el tiempo de establecimiento de dicho flujo. Existen dos modos de operación:

- Throughput, donde cada switch mantiene el número de peticiones pendientes que le permiten sus buffers. Este modo sirve para evaluar el rango máximo de flujos que el controlador puede manejar
- Latencia, cada switch permanece con la petición de flujo, esperando la respuesta antes de



emitir. Permite medir el tiempo de proceso del controlador en condiciones de bajo tráfico.

## **VLC**

VLC [29] es un reproductor multimedia y framework multimedia libre y de código abierto desarrollado por el proyecto VideoLAN. Soporta un gran número de formatos de audio y vídeo y, además, tiene la característica de poder utilizarse como servidor de streaming en una red local o de banda ancha. Esta última función va a ser de gran utilidad en este proyecto, puesto que, como veremos más adelante, permitirá realizar transmisiones de video entre dos hosts.

## **Wireshark**

Software de análisis de protocolos basado en librerías Pcap. Se trata de una herramienta muy útil para realizar análisis de redes y solucionar posibles problemas de red. Wireshark [30] permite capturar paquetes de red y mostrarlo a través de una interfaz gráfica. Su funcionalidad es similar a la de tcpdump (herramienta que explicaremos a continuación), pero añadiendo una interfaz gráfica y muchas opciones de organización y filtrado de información. Permite examinar datos de una red o de un archivo de captura y analizar la información, por medio de detalles y resúmenes por cada paquete.

## **Tcpdump**

Herramienta en línea de comandos cuya utilidad principal es analizar el tráfico que circula por la red. Permite al usuario capturar y mostrar a tiempo real los paquetes transmitidos y recibidos en la red a la cual el ordenador está conectado.

Tcpdump [31] funciona en la mayoría de los sistemas operativos, donde hace uso de la biblioteca libpcap para capturar los paquetes que circulan por la red. Es posible utilizar filtros para permitir

seleccionar los paquetes que queramos utilizar.

## **Comando dpctl**

El comando dpctl [32] es una utilidad que permite la visibilidad y control de una tabla de flujos, aunque esta última función sea también propia del controlador. Es especialmente útil para “depurar” (debugging), al visualizar estados y contadores de flujos. La mayoría de switches OpenFlow pueden iniciarse con un puerto pasivo de escucha, desde el cual se puede sondear el switch, sin tener que añadir un código de depuración al controlador.

## **VNC**

VNC (Virtual Network Computing) [33] es un sistema de conexión que permite usar un ordenador para interactuar con un entorno gráfico de escritorio en un servidor remoto haciendo su uso más fácil en caso de que el usuario no esté familiarizado con la línea de comandos.

Para configurar VNC en un servidor Ubuntu 14.04 y conectarlo de modo seguro a través de un túnel SSH, el servidor VNC usará TightVNC, un paquete de control remoto rápido y de poco peso, que asegurará que la conexión VNC sea suave y estable en las conexiones a Internet más lentas.

La instalación VNC se hará sobre las máquinas virtuales ejecutadas en OpenStack. Estas máquinas tienen Ubuntu 14.04 instalado y están configuradas con un usuario no root con privilegios de superusuario.

## 5.2 Rendimiento del controlador Ryu

En este apartado se ha realizado un estudio de rendimiento del controlador Ryu. Para ello usado en este proyecto utilizando la herramienta Cbench con el objetivo de seleccionar uno de ellos para la posterior implementación de la red SDN. El rendimiento de un controlador SDN está definido por dos características: throughput y latencia. El objetivo es obtener el máximo throughput (número de paquetes pendientes, flujos/s) y la mínima latencia (tiempo de respuesta, ms). Los cambios en estos parámetros cuando se añaden más switches y hosts a la red o más núcleos CPU al servidor donde se ejecuta el controlador muestra la escalabilidad del controlador.

Para medir el throughput, se tienen en cuenta los siguientes parámetros:

- el número de switches conectados al controlador (1, 4, 8, 16, 64, 128, 256), teniendo un número fijo de hosts por switch (  $10^5$  )
- el número de hosts fijo por switch (  $10^3, 10^4, 10^5, 10^6, 10^7$  ), teniendo un número fijo de switches (32)

En el modo throughput, los switches emulados envían tantos mensajes PACKET\_IN como sea posible al controlador, asegurándose que el controlador siempre tiene mensajes que procesar. La latencia se mide con un switch que envía peticiones y espera cada vez a la respuesta del controlador antes de enviar la siguiente petición.

El test de latencia usa Cbench para emular switches que envían un solo paquete al controlador, esperan a la respuesta; repitiendo este proceso tan rápido como sea posible. El número total de respuestas recibidas desde el controlador al final del periodo de tiempo se puede usar para calcular el tiempo promedio que tarda el controlador para procesar cada PACKET\_IN iniciado desde el switch.

En la prueba de evaluación del controlador, conectamos diferentes números de switches virtuales y cada switch está conectado a un cierto número de MAC únicas (Hosts) y en consecuencia estos

hosts están conectados al controlador.

El entorno de prueba se llevó a cabo con el software de emulación OpenFlow. Tanto el controlador como la herramienta de benchmarking se instalaron en el mismo host que es un 64-bit Ubuntu 12.04 LTS. Durante la prueba se ejecutan varias instancias paralelas del controlador y de la herramienta de benchmarking con el fin de utilizar todos los recursos disponibles. Todas las pruebas se hacen con la herramienta cbench. Cada ejecución cbench consiste en 10 bucles de prueba con la duración de 10 segundos. Se ejecuta cada experimento tres veces y tomamos el valor promedio como resultado. Cada test emulará 100000 conexiones con los hosts.

El comando usado para ejecutar cbench sigue el siguiente patrón:

```
cbench -c IP -p port -s X -M -m -l -t
```

donde:

- X: número de switches
- M: MACs (o hosts diferentes)
- m: duración del test
- l: número de test
- t: modo throughput. Si t no aparece, el modo latencia se activa.
- p: puerto de escucha.
- IP: dirección IP de la máquina física donde se está ejecutando el controlador.

Antes de comprobar el rendimiento de Ryu, se puede iniciar Ryu mediante el comando:

```
PYTHONPATH=. ./bin/ryu-manager ryu/app/simple_switch.py
```

El inicio de Ryu se muestra en la figura 24.

```
laura@pedro-TravelMate-5742:~/ryu$ PYTHONPATH=. ./bin/ryu-manager --verbose ryu/
app/simple_switch.py
loading app ryu/app/simple_switch.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/simple_switch.py of SimpleSwitch
BRICK SimpleSwitch
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPacketIn
BRICK ofp_event
  PROVIDES EventOFPPortStatus TO {'SimpleSwitch': set(['main'])}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch': set(['main'])}
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPErrormsg
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEchoRequest
```

Figure 24. Ryu controller startup

Como podemos observar hemos iniciado el controlador con la aplicación `simple_switch.py`. Esta aplicación es una de las aplicaciones más utilizadas de Ryu y consta de las siguientes funciones:

- Aprender las direcciones MAC de los hosts conectados a un puerto y guardarlos en la tabla de direcciones MAC.
- Cuando recibe los paquetes direccionados a un host conocido, los transfiere al puerto conectado al host.
- Cuando recibe los paquetes direccionados a un host desconocido, gestiona la saturación.

En el desarrollo de este proyecto interesa que los switches manejen el protocolo Openflow. Los switches OpenFlow pueden llevar a cabo las siguientes instrucciones Ryu:

- Reescribir las direcciones de los paquetes recibidos o transferir los paquetes desde un puerto específico.
- Transferir los paquetes recibidos al controlador (Packet-In)
- Transferir los paquetes encaminados del controlador desde un puerto específico (Packet-Out).

La aplicación `simple_switch.py` puede realizar estas funciones mediante el uso de la función de los paquetes `Packet-In` para que aprenda las direcciones MAC. El controlador puede usar la función de los `Packet-In` para recibir los paquetes del switch. El switch analiza los paquetes recibidos para aprender las direcciones MAC del host y la información sobre el puerto conectado.

Después de esto, el switch manda los paquetes recibidos según la dirección MAC aprendida anteriormente. Puede entonces realizar los siguientes procesados:

- Si el host es conocido, usa la función `Packet-Out` para transferir los paquetes desde el puerto conectado.
- Si el host no es conocido, usa la función `Packet-Out` para gestionar la saturación.

Cabe mencionar también que la aplicación `simple_switch` está disponible para las distintas versiones de OpenFlow.

Existe un gran número de aplicaciones para el controlador Ryu. Estas se pueden encontrar en los ficheros del mismo dentro del directorio `ryu/app`.

## **Rendimiento de Ryu**

El rendimiento de Ryu se analizará usando la herramienta `Cbench`. Esta herramienta se usará tanto en modo `throughput` como en modo `latencia`. La funcionalidad del controlador se prueba usando la aplicación `simple_switch`.

En primer lugar se estudiará la latencia. El comando que se usa es el siguiente:

```
cbench -c 192.168.60.24 -p 6633 -s X -M 100000 -m 10000 -l 10
```

El test se ejecutará variando el número de switches conectados al controlador. En cada ejecución del anterior comando, se envían paquetes `PACKET_IN`, se espera a que exista un flujo coincidente y este vuelva. Este proceso se repetirá tantas veces como se le indique.

Los resultados obtenidos se muestran en la tabla 6.

Switches	Respuestas/s
1	3967,93
4	4011,95
8	3844,62
16	3189,27
32	3208,39
64	3060,30
128	3423,71
256	3052,38

Table 6– Latency tests results

Como podemos observar, la herramienta cbench nos da los resultados en forma de flujos por segundo. Para calcular la latencia de la red, se utilizará la siguiente ecuación:

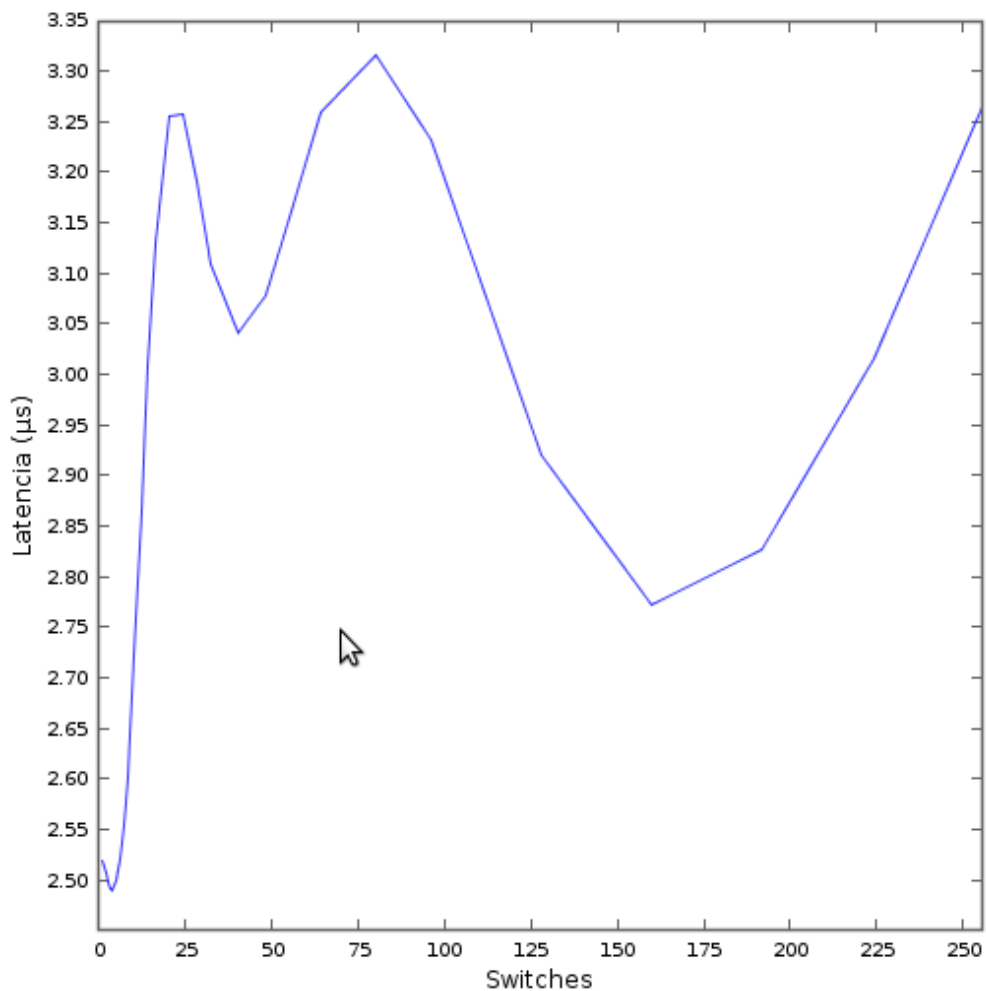
Latencia = Duración de las iteraciones del test (milisegundos) / número de Respuestas OpenFlow Recibidas del Controlador

En este caso, el número de respuestas del controlador son los valores proporcionados en la tabla 3 y la duración del test vendrá dada por el parámetro m de la llamada, que en este caso es 10000 ms. En la tabla 7 y gráfica 1 se muestran los cálculos obtenidos.

Switches	Latencia (µs)
1	2,52
4	2,49
8	2,60
16	3,13
32	3,11
64	3,26
128	2,92
256	3,27

Tabla 7. Latency estimations

Observando la gráfica 1, se puede concluir que la latencia tiene una tendencia ascendente a medida que aumenta el número de switches. Este comportamiento se debe a que el controlador no tiene la misma facilidad para establecer los flujos a la misma velocidad que se hacen las peticiones, pudiendo existir pérdidas de paquetes. Los valores más altos de latencia se dan a partir de 16 switches utilizados con un controlador, por lo que se puede concluir que el controlador Ryu funciona correctamente con un número de hasta 8 switches aproximadamente, por lo que si se quiere utilizar más de 8 switches es recomendable usar más de un controlador de forma que la latencia de la red no sea muy alta.



Graph 1- Ryu controller latency



Estudiaremos ahora el modo throughput usando el siguiente comando:

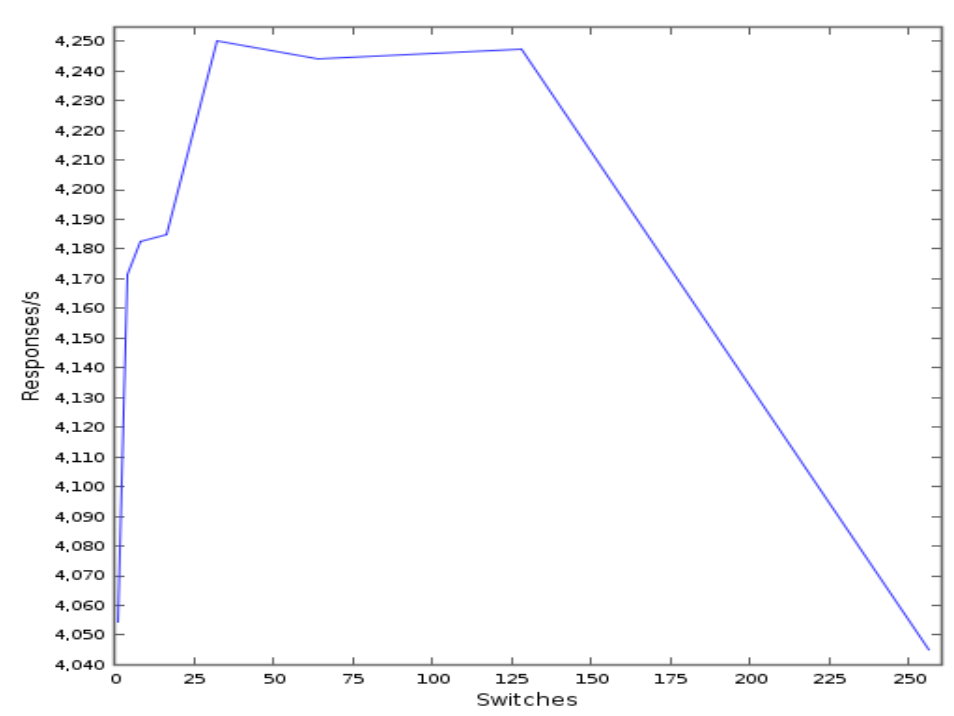
```
cbench -c 192.168.60.24 -p 6633 -s X -M 100000 -m 10000 -l 10 -t
```

Los resultados obtenidos se muestran en la tabla 8 y la gráfica 2.

Switches	Respuestas/s
1	4054,34
4	4171,57
8	4182,37
16	4184,77
32	4250,29
64	4244,18
128	4247,26
256	4045,16

*Table 8. Throughput test*

Se puede observar partiendo desde los resultados mostrados en la gráfica 2 que el número de flujos permanece prácticamente constante hasta que el número de switches es mayor a 128, donde se experimenta una caída más abrupta. Este comportamiento se debe a que el controlador puede llegar a saturarse, puesto que al aumentar el número de switches el número de flujos también aumenta y los buffers del controlador pueden saturarse.



Gráfica 2- Prueba throughput

Teniendo en cuenta que el comportamiento ideal del controlador vendrá dado en el punto en el que el throughput sea máximo y la latencia tenga el valor mínimo posible, para elegir un número de switches idóneo se buscará un equilibrio entre los dos parámetros. Viendo los resultados anteriores, se puede concluir que el número adecuado de switches es 8, puesto que proporciona un throughput considerablemente alto con un valor de latencia relativamente bajo.

## Mensajes Openflow

Como hemos mencionado anteriormente, al utilizar la herramienta cbench, se establecen una serie de flujos entre los switches emulados y el controlador. Si realizamos una captura con wireshark, mostrada en la figura 25, vemos que aparecen una serie de paquetes openflow correspondientes a estos flujos. La captura se ha realizado para la prueba con 1 switch.

4	0.000194	192.168.60.24	192.168.60.24	OF 1.0	74 of hello
6	0.002345	192.168.60.24	192.168.60.24	OF 1.0	74 of hello
8	0.002449	192.168.60.24	192.168.60.24	OF 1.0	74 of features request
10	0.002494	80:00:00:00:00:01	Broadcast	ARP + OF	374 Gratuitous ARP for 192.168.1.40 (Reply) + of_packet_in
11	0.004949	192.168.60.24	192.168.60.24	OF 1.0	78 of set config
12	0.004967	192.168.0.40	192.168.1.40	OF 1.0	148 of packet in
13	0.006077	80:00:00:00:00:01	Broadcast	OF 1.0	154 of packet out
14	0.006099	192.168.0.40	192.168.1.40	OF 1.0	148 of packet in
15	0.006803	192.168.60.24	192.168.60.24	OF 1.0	146 of flow add
16	0.006824	192.168.0.40	192.168.1.40	OF 1.0	148 of packet in
17	0.006837	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out
18	0.006851	192.168.0.40	192.168.1.40	OF 1.0	148 of packet in
19	0.007764	192.168.60.24	192.168.60.24	OF 1.0	146 of flow add

Figure 25 – OpenFlow messages with cbench tool

A partir de los flujos introducidos por la herramienta Cbench, podemos mencionar los mensajes OpenFlow más destacados.

- OF\_HELLO. Mensajes que se intercambian entre el controlador y el switch en el inicio de la conexión.
- OF\_FEATURE\_REQUEST. El controlador pide la indentificación y las capacidades del switch mediante el envío de este tipo de mensajes.
- ARP + OPENFLOW. Estos mensajes son las peticiones/respuestas ARP de los hosts. Al no coincidir con ninguna entrada de la tabla de flujos del switch, son encapsulados en paquetes openflow y enviados al controlador.
- OF\_SET\_CONFIG. El controlador establece los parámetros de configuración del switch.
- PACKET\_IN. Con este tipo de mensajes, el control del paquete pasa al controlador.
- PACKET\_OUT. Estos mensajes se usan cuando un paquete se almacena en el buffer y expira después de un tiempo.

Se puede observar que los primeros paquetes son del tipo `of_hello`, lo que quiere decir que el establecimiento de la conexión con el switch se ha realizado satisfactoriamente. Además se utilizará el protocolo ARP junto con mensajes de tipo `Packet_In` para descubrir la dirección IP del switch emulado. A continuación una serie de mensajes OpenFlow del tipo `Packet_In` y `Packet_Out` son intercambiados entre el switch y el controlador. Cada nuevo flujo está introducido por un tipo de paquetes OpenFlow denominado `of_flow_add`.

## 5.3 Implementación de una red SDN con Mininet y Ryu

Como se ha explicado anteriormente en este capítulo, Mininet es una herramienta muy útil a la hora de emular una red SDN y así poder estudiar su comportamiento. Se utilizará una topología simple, mostrada en la figura 26, con dos hosts conectados a un switch.

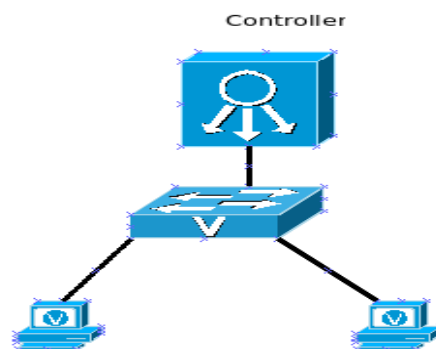


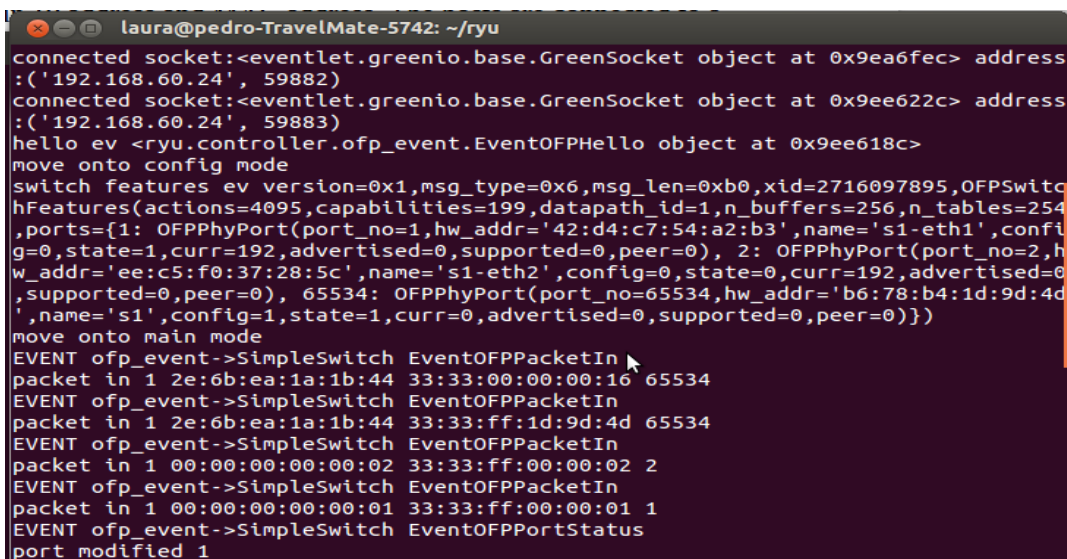
Figure 26 . Topology used in Mininet

A continuación, desde el terminal ejecutaremos mininet con esta topología e invocando el controlador Ryu. Primero iniciaremos el controlador e invocaremos mininet con la topología que queremos utilizar mediante el comando:

```
sudo mn --topo single,2 --mac --controller
remote,ip=192.168.60.24:6633 --switch ovsk
```

Se crea una topología virtual con un switch ovsk (Openvswitch) con dos puertos virtuales y dos host con una dirección IP y dirección MAC determinada. Los hosts están conectados a un puerto del switch mediante una conexión virtual Ethernet. El controlador, en este caso, es un controlador remoto (Ryu) cuya dirección IP es la 192.168.60.24 y el puerto de escucha es el 6633.

Como se puede observar en la figura 27, al iniciar Mininet y Ryu aparecerán una serie de mensajes en el controlador que indican que la conexión está establecida.



```
laura@pedro-TravelMate-5742: ~/ryu
connected socket:<eventlet.greenio.base.GreenSocket object at 0x9ea6fec> address
:('192.168.60.24', 59882)
connected socket:<eventlet.greenio.base.GreenSocket object at 0x9ee622c> address
:('192.168.60.24', 59883)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x9ee618c>
move onto config mode
switch features ev version=0x1,msg_type=0x6,msg_len=0xb0,xid=2716097895,OFPSwitc
hFeatures(actions=4095,capabilities=199,datapath_id=1,n_buffers=256,n_tables=254
,ports={1: OFPPhyPort(port_no=1,hw_addr='42:d4:c7:54:a2:b3',name='s1-eth1',confi
g=0,state=1,curr=192,advertised=0,supported=0,peer=0), 2: OFPPhyPort(port_no=2,h
w_addr='ee:c5:f0:37:28:5c',name='s1-eth2',config=0,state=0,curr=192,advertised=0
,supported=0,peer=0), 65534: OFPPhyPort(port_no=65534,hw_addr='b6:78:b4:1d:9d:4d
',name='s1',config=1,state=1,curr=0,advertised=0,supported=0,peer=0)})
move onto main mode
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 2e:6b:ea:1a:1b:44 33:33:00:00:00:16 65534
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 2e:6b:ea:1a:1b:44 33:33:ff:1d:9d:4d 65534
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:ff:00:00:02 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:ff:00:00:01 1
EVENT ofp_event->SimpleSwitch EventOFPPortStatus
port modified 1
```

Figure 27. Connection establishment between Mininet and Ryu

En la figura anterior vemos el establecimiento de la conexión en el controlador, pero estos mensajes se pueden ver más claramente haciendo una captura con la herramienta Wireshark. Esta captura se puede ver en la figura 28.

13	10.505661	192.168.60.24	192.168.60.24	OF 1.0	74 of hello
15	10.506784	192.168.60.24	192.168.60.24	OF 1.0	74 of hello
17	10.507603	192.168.60.24	192.168.60.24	OF 1.0	74 of features request
18	10.507711	192.168.60.24	192.168.60.24	OF 1.0	242 of features reply
19	10.508688	192.168.60.24	192.168.60.24	OF 1.0	78 of set config
20	10.509628	::	ff02::16	OF 1.0	174 of packet in
21	10.512140	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out
23	10.628217	::	ff02::1:ff00:1	OF 1.0	162 of packet in
24	10.629675	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out
26	10.884219	::	ff02::1:ffd4:f3b	OF 1.0	162 of packet in
27	10.886925	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out

Figure 28. Connection establishment between Mininet and Ryu with Wireshark

Al analizar la figura anterior, se puede ver como la conexión entre el controlador y el switch se lleva a cabo mediante el intercambio de mensajes HELLO. Si analizamos el contenido del paquete OF\_HELLO mostrado en la figura 29, vemos que en él podemos encontrar campos que establecen la versión del protocolo OpenFlow utilizado, el tipo de mensaje OpenFlow y su longitud.

```

▶ Frame 13: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 192.168.60.24 (192.168.60.24), Dst: 192.168.60.24 (192.168.60.24)
▶ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 60034 (60034), Seq: 1, Ack: 1, Len: 8
▼ OpenFlow
  version: 1
  type: OFPT_HELLO (0)
  length: 8
  xid: 4099365213

```

Figure 29. OF\_HELLO message

A continuación de los mensajes OF\_HELLO aparecen dos mensajes de tipo OF\_Request. Uno de ellos es de petición y otro de respuesta. Con estos mensajes, el controlador solicita que el switch se identifique y este responde a esta petición. En las figuras 30 y 31 se muestran estos paquetes.

```

▶ Frame 17: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▼ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
  ▶ Destination: 00:00:00_00:00:00 (00:00:00:00:00:00)
  ▶ Source: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Type: IP (0x0800)
▶ Internet Protocol Version 4, Src: 192.168.60.24 (192.168.60.24), Dst: 192.168.60.24 (192.168.60.24)
▶ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 60034 (60034), Seq: 9, Ack: 9, Len: 8
▼ OpenFlow
  version: 1
  type: OFPT_FEATURES_REQUEST (5)
  length: 8
  xid: 4099365214

```

Figure 30. Of\_features\_request messages

Con el mensaje mostrado en la figura 29, se establece la petición por parte del controlador sobre la identificación y las capacidades del switch. En la figura 30 mostrada a continuación, el switch manda al controlador su indentificación y sus características.

```
▼ OpenFlow
  version: 1
  type: OFPT_FEATURES_REPLY (6)
  length: 176
  xid: 4099365214
  datapath_id: 1
  n_buffers: 256
  n_tables: 254
  capabilities: Unknown (0x000000c7)
  actions: 4095
  ▼ of_port_desc list
    ▼ of_port_desc
      port_no: 1
      hw_addr: c2:35:c2:d3:aa:50 (c2:35:c2:d3:aa:50)
      name: s1-eth1
      config: Unknown (0x00000000)
      state: OFPPS_LINK_DOWN (0x00000001)
      curr: Unknown (0x000000c0)
      advertised: Unknown (0x00000000)
      supported: Unknown (0x00000000)
      peer: Unknown (0x00000000)
    ▼ of_port_desc
      port_no: 2
      hw_addr: 16:41:77:9c:10:df (16:41:77:9c:10:df)
      name: s1-eth2
      config: Unknown (0x00000000)
      state: OFPPS_STP_LISTEN (0x00000000)
      curr: Unknown (0x000000c0)
      advertised: Unknown (0x00000000)
      supported: Unknown (0x00000000)
      peer: Unknown (0x00000000)
```

Figure 31. *Of\_features\_reply message*

A continuación, se establece el flujo de paquetes del tipo PACKET\_IN y PACKET\_OUT. En la figura 32 mostrada a continuación se muestra el contenido de los paquetes del tipo PACKET\_IN.

```

▼ OpenFlow
  version: 1
  type: OFPT_PACKET_IN (10)
  length: 108
  xid: 0
  buffer_id: 256
  total_len: 90
  in_port: 65534
  reason: OFPR_NO_MATCH (0)
  ▼ Ethernet packet
    ▼ Ethernet II, Src: 0e:12:bf:71:44:4a (0e:12:bf:71:44:4a), Dst: IPv6mcast_00:00:00:16 (33:33:00:00:00:16)
      ▶ Destination: IPv6mcast_00:00:00:16 (33:33:00:00:00:16)
      ▶ Source: 0e:12:bf:71:44:4a (0e:12:bf:71:44:4a)
      Type: IPv6 (0x86dd)
    ▼ Internet Protocol Version 6, Src: :: (::), Dst: ff02::16 (ff02::16)
      ▶ 0110 .... = Version: 6
      ▶ .... 0000 0000 .... = Traffic class: 0x00000000
      ▶ .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
      Payload length: 36
      Next header: IPv6 hop-by-hop option (0x00)
      Hop limit: 1
      Source: :: (::)
      Destination: ff02::16 (ff02::16)
      ▶ Hop-by-Hop Option
    ▶ Internet Control Message Protocol v6

```

Figure 32. PACKET\_IN message in the controller

Como se observa en la figura anterior, en el mensaje OpenFlow aparece la versión del protocolo utilizada, además de un paquete de tipo Ethernet con los campos de origen y destino del paquete.

En la figura 33, se muestra el mensaje OpenFlow del tipo PACKET\_OUT. La información que se muestra en la captura es la acción asociada a este paquete. En este caso la acción asociada es la de descartar un paquete almacenado en el buffer.

```

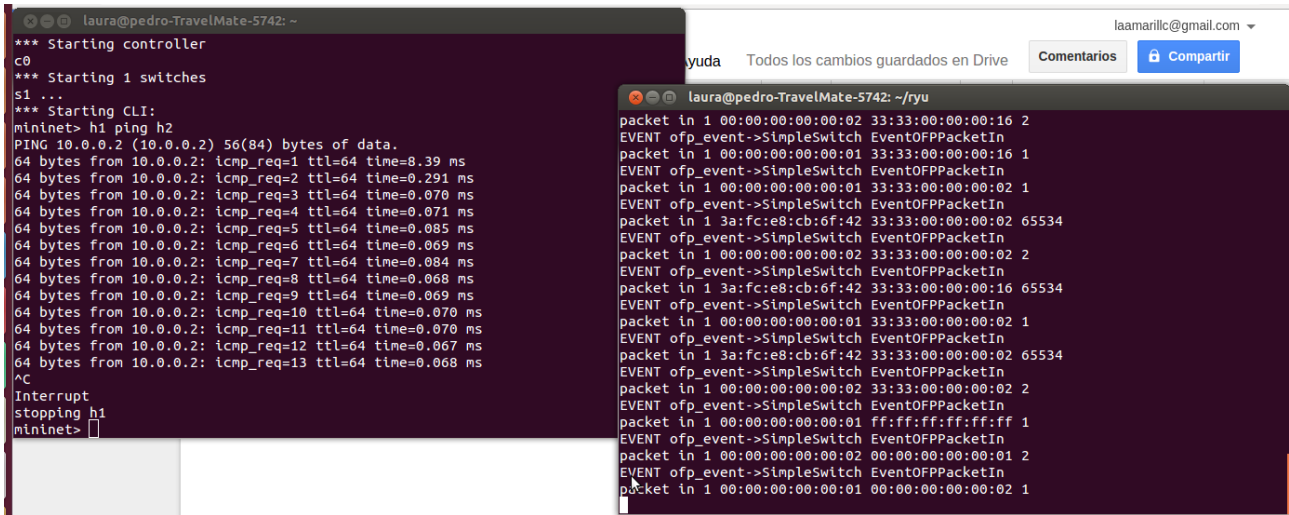
▼ OpenFlow
  version: 1
  type: OFPT_PACKET_OUT (13)
  length: 24
  xid: 4099365216
  buffer_id: 256
  in_port: 65534
  actions_len: 8
  ▼ of_action list
    ▼ of_action_output
      type: OFPAT_OUTPUT (0)
      len: 8
      port: 65531
      max_len: 65509

```

Figure 33. PACKET\_OUT message



Una vez que tanto mininet como Ryu están iniciados pasaremos a realizar las pruebas. Comenzaremos por realizar una prueba de conectividad entre los dos hosts de la topología. Esta prueba se hará mediante el uso del comando ping, cuyo resultado se muestra en la figura 34.



```
laura@pedro-TravelMate-5742: ~
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=8.39 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.291 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_req=6 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_req=7 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_req=8 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_req=9 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_req=10 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_req=11 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_req=12 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_req=13 ttl=64 time=0.068 ms
^C
Interrupt
stopping h1
mininet>

laura@pedro-TravelMate-5742: ~/ryu
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 3a:fc:e8:cb:6f:42 33:33:00:00:00:02 65534
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 3a:fc:e8:cb:6f:42 33:33:00:00:00:16 65534
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 3a:fc:e8:cb:6f:42 33:33:00:00:00:02 65534
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
```

Figure 34 . Connectivity test between h1 y h2

Podemos ver como existe conectividad entre los hosts de la topología. Se comprueba que es posible enviar paquetes ICMP entre los hosts de la red emulada. Además es posible ver en el controlador una serie de paquetes de tipo OpenFlow.

Además de comprobar la conectividad, veremos cuales son los paquetes que circulan por la red a la hora de ejecutar esta prueba. Para ello, a la vez que se lleva a cabo el comando ping, utilizaremos wireshark para escuchar el tráfico de la interfaz lo, como se muestra en la figura 35. En el controlador aparecerán paquetes de tipo ECHO.

15	4.999995	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
16	5.000505	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
18	9.999972	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
19	10.000588	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
21	15.000035	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
22	15.000570	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
24	19.999957	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
25	20.000634	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply

Figure 35. Ping command results in the controller

Los mensajes ECHO son mensajes de tipo petición/respuesta que pueden ser enviados bien desde el controlador o bien desde el switch. Se enviará primero un paquete del tipo of\_echo\_request (petición) y este debe ser respondido con un mensaje of\_echo\_reply.

Además de en la interfaz anterior, también se ha capturado el tráfico existente entre los dos hosts (h1 y h2) de la topología. Este tráfico se muestra en la figura 36.

1	0.000000	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
2	0.000606	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
4	0.652916	00:00:00 00:00:01	Broadcast	OF 1.0	126 of packet in
5	0.655539	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out
7	0.656088	00:00:00 00:00:02	00:00:00 00:00:01	OF 1.0	126 of packet in
8	0.657903	192.168.60.24	192.168.60.24	OF 1.0	146 of flow add
9	0.657989	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out
11	0.658578	10.0.0.1	10.0.0.2	OF 1.0	182 of packet in
12	0.660378	192.168.60.24	192.168.60.24	OF 1.0	146 of flow add
13	0.660461	192.168.60.24	192.168.60.24	OF 1.0	90 of packet out
15	4.999995	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
16	5.000505	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
18	9.999972	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
19	10.000588	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
21	15.000035	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
22	15.000570	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply
24	19.999957	192.168.60.24	192.168.60.24	OF 1.0	74 of echo request
25	20.000634	192.168.60.24	192.168.60.24	OF 1.0	74 of echo reply

Figure 36 . Ping command results between h1 and h2

Analizando la figura 36 mostrada anteriormente, se puede observar que además de los mensajes de petición respuesta, aparecen mensajes de tipo PACKET\_IN. En estos mensajes se proporciona la información de las direcciones de origen y destino del ping encapsulada dentro del mensaje OpenFlow. La dirección IP de origen es la 10.0.0.1 y la de destino es la 10.0.0.2, correspondientes a los hosts h1 y h2 respectivamente. También se puede ver en la figura 37, la información correspondiente al paquete ICMP, que, en este caso es de petición.

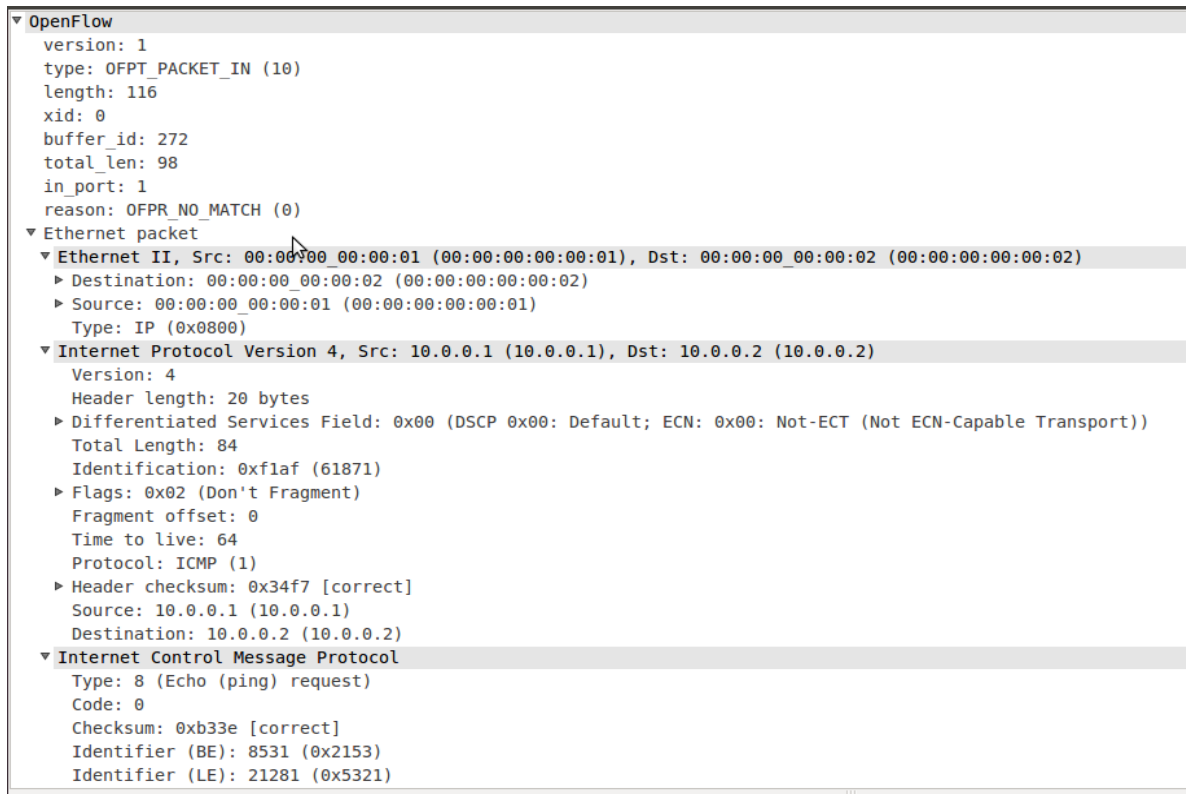


Figure 37. PACKET\_IN messages in command ping

Después de esta prueba, es lógico concluir que el switch contiene información relacionada con el tráfico que ha circulado por la red implementada. Mediante la utilización del comando `dpctl show` mostrado en la figura 38 podemos ver el estado del switch y sus capacidades. El comando utilizado será:

```
dpctl show tcp:192.168.60.24:6634
```

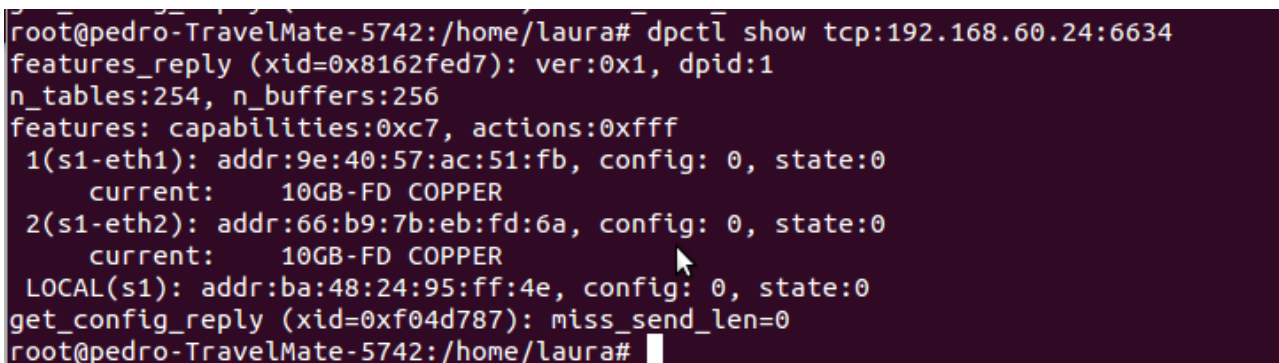


Figure 38. Command dpctl-show results

En la figura anterior es posible observar la información contenida en el switch. En esta información destacan, además de información sobre la versión OpenFlow utilizada, el número de tablas y el número de buffers; información sobre los enlaces establecidos a la hora de emular la topología. Estos enlaces emulados son los pertenecientes a la unión entre el switch y los dos hosts. Ambos tienen una dirección MAC asociada y son de tipo cobre de 10 GB, lo que hace que proporcionen una gran capacidad de encaminamiento de tráfico.

La herramienta `dpctl` también posibilita realizar la verificación de la tabla de flujos. Para ello se utiliza el comando:

```
dpctl dump-flows tcp:192.168.60.24:6634.
```

Este comando es muy útil para obtener información de los flujos instalados, mediante las reglas definidas para cada flujo, de los contadores de tiempo que informan cuánto ha durado el flujo en la tabla de flujos y de las acciones a realizar con cada flujo. El resultado obtenido se muestra en la figura 39.

```
root@pedro-TravelMate-5742:~# dpctl dump-flows tcp:192.168.60.24:6634
stats_reply (xid=0x16cff71): flags=none type=1(flow)
  cookie=0, duration_sec=659s, duration_nsec=599000000s, table_id=0, priority=32
768, n_packets=8, n_bytes=728, idle_timeout=0,hard_timeout=0,in_port=2,d1_dst=00:00:00:00:01,actions=output:1
  cookie=0, duration_sec=659s, duration_nsec=597000000s, table_id=0, priority=32
768, n_packets=7, n_bytes=630, idle_timeout=0,hard_timeout=0,in_port=1,d1_dst=00:00:00:00:02,actions=output:2
root@pedro-TravelMate-5742:~#
```

Figure 39. Examination of flow table using `dpctl`

Las tablas de flujo como la que se muestra en la tabla 9 permiten relacionar los paquetes entrantes con un flujo y un conjunto de acciones que se deben realizar para indicar al switch como manejar los flujos. Está compuesta por un conjunto de entradas de flujo.

Match Fields		Priority	Counters	Instructions	Timeouts		Cookie
in_port	PacketHeaders	32	8	output:1	idle	hard	0
2	728				0	0	

Table 9. Flow table

Como se puede observar tanto en la tabla como en la figura 36, las tablas de flujo están compuestas por las siguientes entradas:

- Match field. Contiene los puertos de entrada y las cabeceras de los paquetes.
- Priority. Proporciona el orden de precedencia de la entrada de flujo.
- Counters. Actualiza los paquetes coincidentes
- Instructions. Modifica la acción establecida.
- Timeouts. Máxima cantidad de tiempo antes de que el flujo expire.
- Cookie. Esta entrada es usada por el controlador para filtrar las estadísticas de los flujos, modificarlos o borrarlos. No se usa cuando los paquetes se están procesando.

Una vez comprobada la conectividad, se han realizado pruebas de rendimiento en la red emulada. Esta prueba se realizó mediante la herramienta Jperf. Con esta herramienta podemos medir el ancho de banda entre dos hosts. Jperf es una interfaz gráfica de Iperf (figura 40), por lo que tiene sus mismas funcionalidades. Al tratarse de una herramienta cliente-servidor, ejecutaremos Jperf en las dos máquinas (h1 y h2). En este caso h1 tendrá la función servidor y h2 la función cliente. Esta herramienta toma los parámetros de rendimiento de ancho de banda, jitter y paquetes perdidos de una transmisión TCP ó UDP en función del parámetro que se quiere obtener. Las pruebas consisten en enviar desde el servidor a un host específico, un flujo de datos a una tasa de transmisión constante por un determinado puerto. El cliente captura los datagramas en ventanas de 1 segundo y a medida que recibe el flujo de datos publica los parámetros de rendimiento. Se analizarán a continuación los resultados obtenidos.

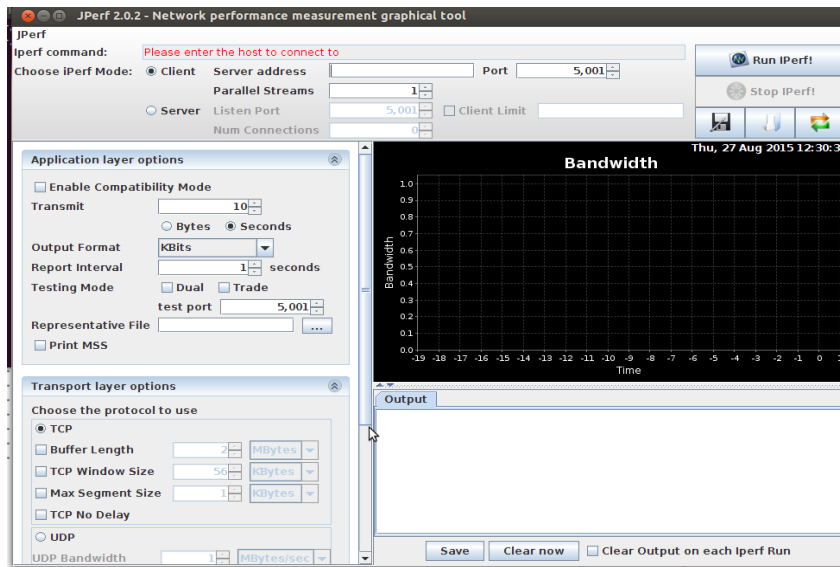


Figure 40. Jperf graphical interface

Comenzaremos midiendo la latencia de la red mediante el comando ping. Para ello abrimos el terminal del host h1 (también podría hacerse con h2). Al finalizar el proceso este comando mostrará una serie de estadísticas que servirán para obtener información sobre el rendimiento de la red. Estos resultados se muestran en la figura 41.

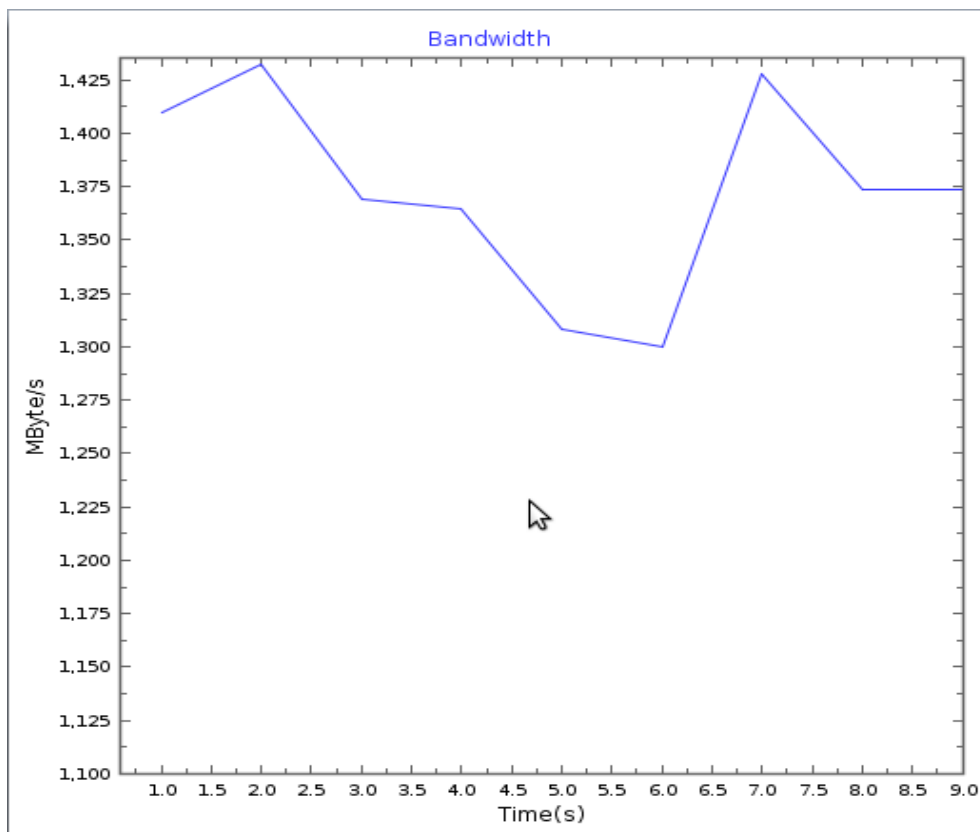
```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=7.82 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.398 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_req=6 ttl=64 time=0.068 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.050/1.419/7.828/2.868 ms
```

Figure 41. Command ping results

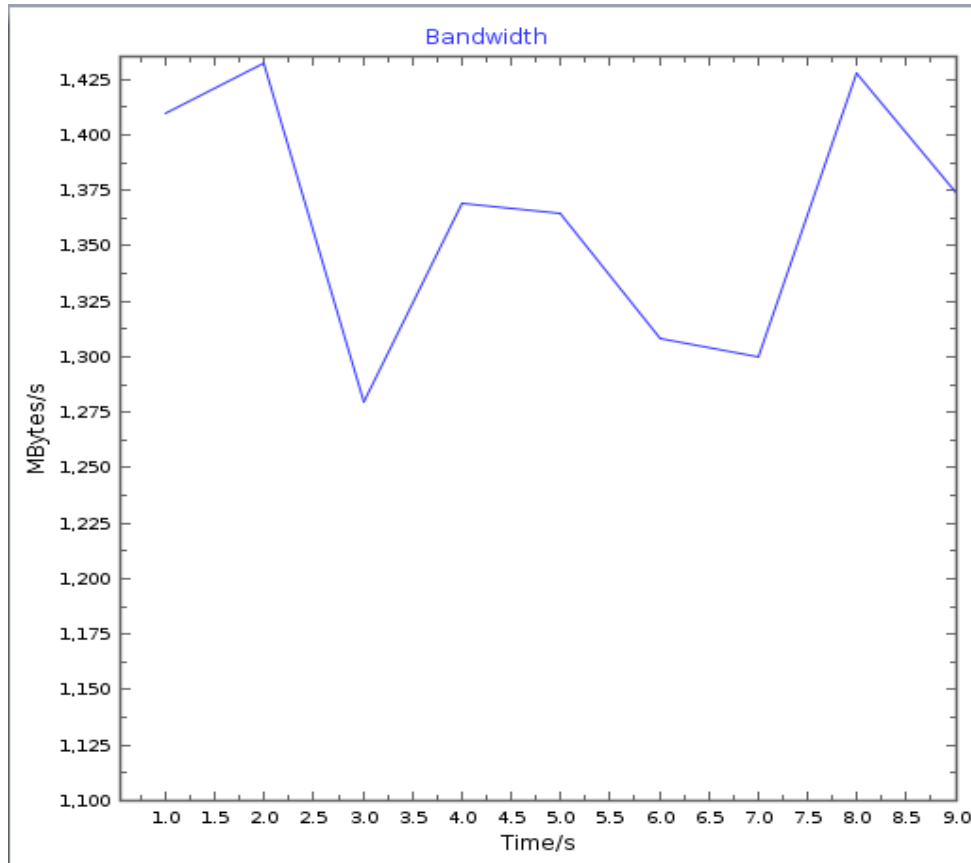
De la figura anterior podemos concluir que no existe pérdida de paquetes ICMP por lo que la conectividad de la red está asegurada. Para ver la latencia, deberemos fijarnos en el parámetro rtt (Round-trip delay). Este parámetro proporciona el tiempo que tarda un paquete de datos enviado desde el host h1 en volver a este host habiendo pasado por el host h2. En este caso vemos que el

valor medio de rtt es 1.419 ms.

A continuación se llevaron a cabo transmisiones con el protocolo TCP durante un intervalo de 10 segundos. Para el tamaño de ventana se ha escogido el valor por defecto, que es de 0.08 Mbytes. Estas transmisiones se utilizarán para medir el throughput. Como se ha explicado anteriormente, h1 se ejecuta como servidor y h2 como cliente. Los resultados obtenidos se muestran en las gráficas 3, donde se muestra el resultado del servidor, y 4, donde se muestran los del cliente.



Graph 3. Server bandwidth



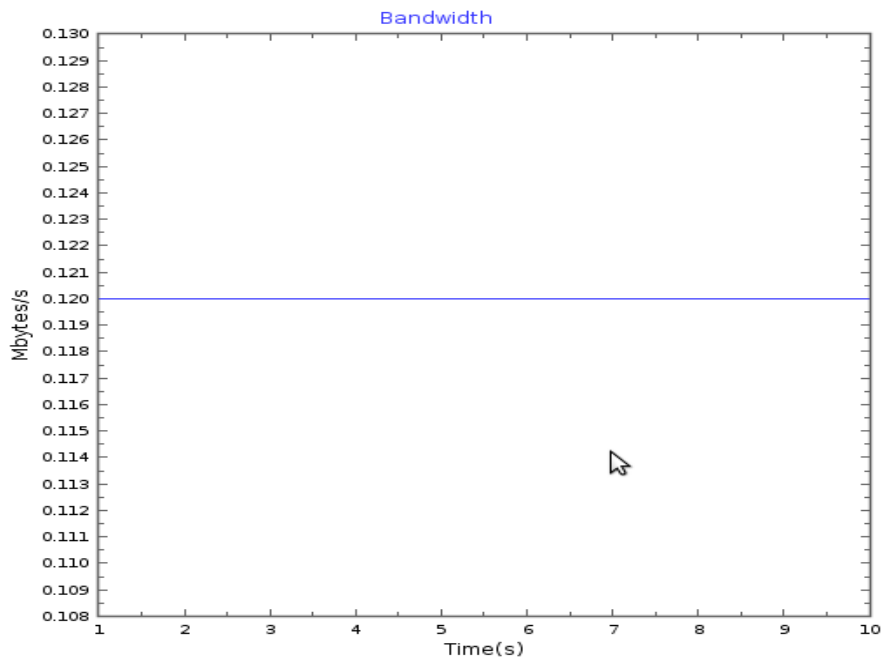
Graph 4. Client bandwidth

Se puede observar como existe una variación en el ancho de banda durante la transmisión. Esta variación puede deberse al modo de funcionamiento del protocolo TCP. Después de iniciar la sesión TCP, y del control de flujo, cada cierto número de paquetes existe un paquete ack, que deberá ser procesado, lo que implica que existirán tiempos en los que el enlace no se esté utilizando. Esto provoca que el ancho de banda en el cliente sea menor.

Realizamos a continuación una transmisión UDP, que servirá para medir la pérdida de paquetes. La duración de la transmisión es de 10 segundos. Los resultados se muestran en las gráficas 5 y 6. Las transmisiones UDP son útiles para ver si existen pérdidas de paquetes en la transmisión. Para ello, nos fijamos en las estadísticas obtenidas en la transmisión y que se muestran a continuación:

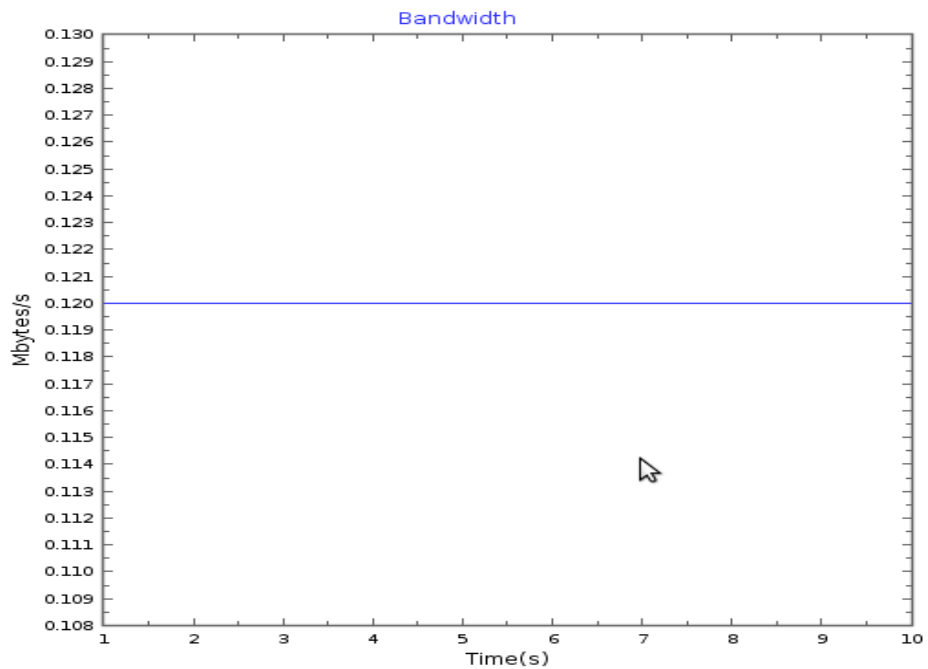
[ ID]	Interval	Transfer	Bandwidth	Jitter	Lost/Total Datagrams
[ 3]	0.0-10.0 sec	1.19 MBytes	0.12 MBytes/sec	0.022 ms	0/ 852 (0%)





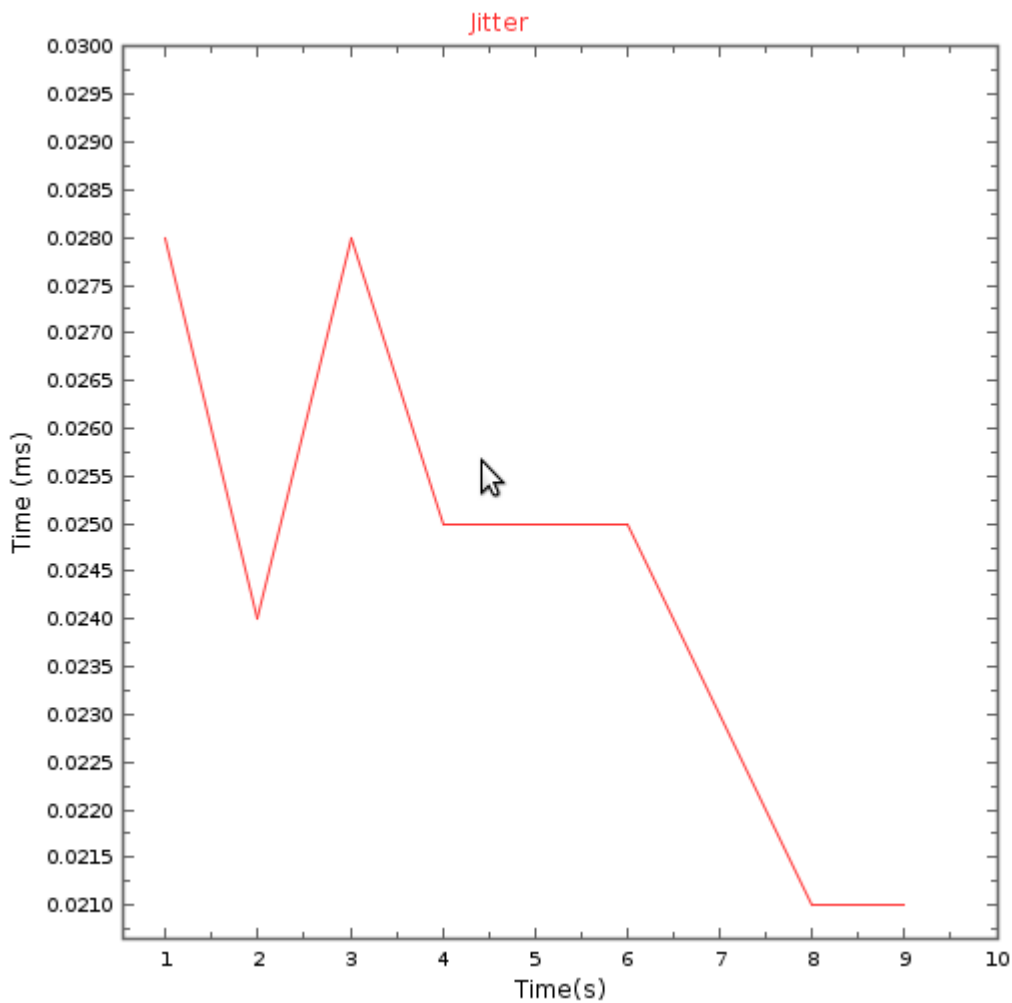
Graph 5. Server bandwidth

Se puede observar que no existe pérdida de paquetes en la transmisión. Teóricamente, no se debería tener una pérdida de paquetes mayor del 1%. Como en este caso la pérdida es del 0%, se tiene una transmisión fiable.



Graph 6. Client bandwidth

En el servidor, tal y como podemos ver en la figura 43, se tiene un ancho de banda en la transmisión de 0,12 Mbytes/s durante toda la transmisión. En el cliente se puede observar que existe un ancho de banda de 0,12 Mbytes/s como muestran las estadísticas mostradas anteriormente. Este ancho de banda se mantiene constante durante toda la transmisión, debido a que el protocolo UDP es un protocolo no orientado a conexión y tampoco utiliza mecanismos de confirmación de entrega, por lo que el enlace se estará utilizando continuamente.



Graph 7. Jitter

En la gráfica 7 mostrada anteriormente se puede observar el jitter obtenido durante la transmisión. El jitter se define técnicamente como la variación en el tiempo en la llegada de los paquetes, causada por congestión de red, pérdida de sincronización o por las diferentes rutas seguidas por los paquetes para llegar al destino. Dicho con otras palabras, el jitter es la variación de la latencia en la

transmisión, aunque no depende de ella. Es posible tener un valor de respuesta elevado y un jitter bajo. Este parámetro nos permite saber la calidad del enlace. Un jitter alto en el enlace significaría que el enlace es de mala calidad, puesto que existiría una variación en latencia importante, lo que provocaría retrasos en las transmisiones. Por ejemplo, en enlaces que se utilizan para VoIP, un jitter alto podría hacer que se cortase la llamada. Por el contrario, un jitter bajo garantiza una buena calidad del enlace. Este valor es importante para transmisiones en las que se necesita que el envío de paquetes sea constante, como por ejemplo, transmisiones de video. Prueba que realizaremos a continuación.

Un valor de jitter superior a 100 ms sería considerado alto. Si el valor es menor a esa cifra, podría compensarse de forma apropiado. En este caso el valor medio del jitter es de 0,022 ms, por lo que la calidad del enlace es buena.

## **Transmisiones de video streaming**

En este apartado haremos una prueba de transmisión de un archivo de video utilizando la herramienta VLC. El archivo de vídeo utilizado tiene un tamaño de 220,5 MB y una duración de aproximadamente 10 minutos. Las dimensiones del vídeo son de 854 x 480 con una tasa de fotogramas de 24 fotogramas por segundo.

Se hará una transmisión con una conexión del tipo cliente-servidor. Se tomará como servidor el host h1 (10.0.0.1) y como cliente el host h2 (10.0.0.2). Se hará una transmisión HTTP.

Para comenzar a hacer la prueba se abrirán los terminales de h1 y h2. Abrimos a continuación vlc desde los terminales mediante el comando

```
vlc-wrapper
```

Una vez hecho esto, es posible comenzar la emisión del vídeo. La interfaz gráfica de VLC se puede ver en la figura 42.

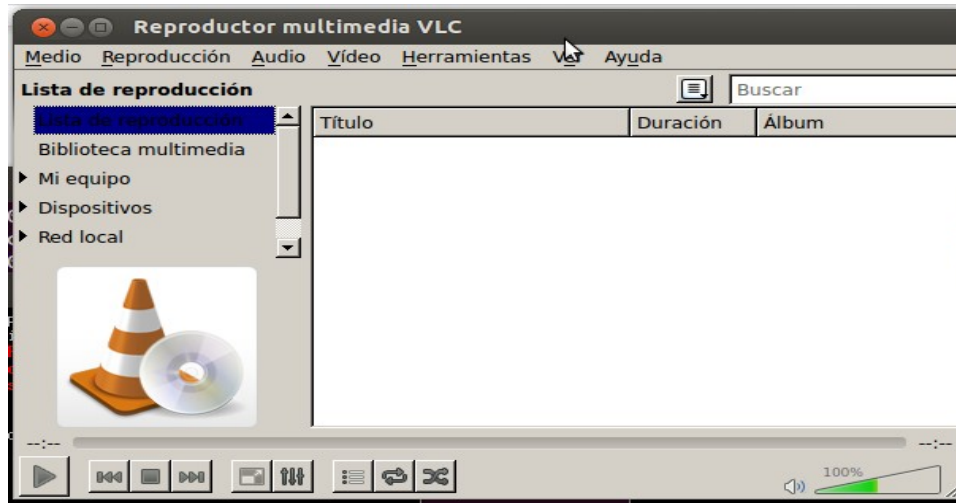


Figure 42. VLC graphical interface

A continuación se procederá a realizar una emisión de video mediante el protocolo HTTP. Estas transmisiones de video se llevan a cabo mediante el puerto 8080. Por tanto, se configuró en el servidor y en cliente este parámetro. El servidor se ha configurado tal y como se muestra en las figura 43.

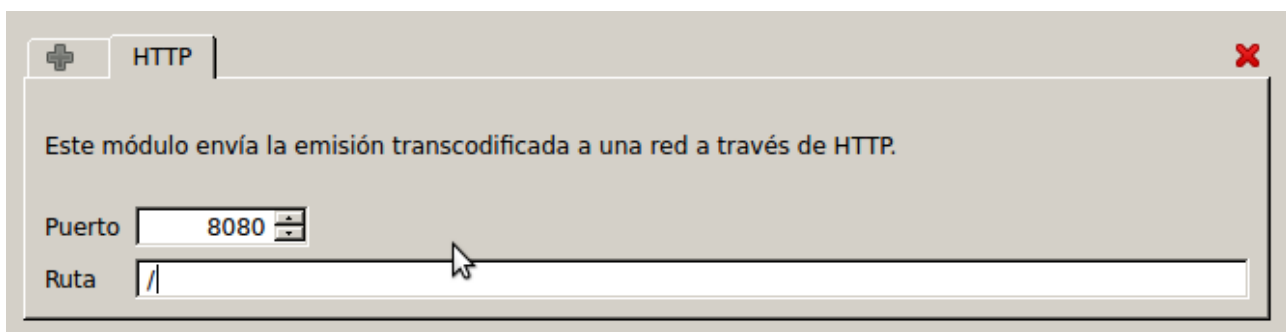


Figure 43. Video transmission configuration in the server

A continuación será necesario configurar las opciones de transcodificación. En este caso se ha utilizado un perfil de Video – H.264 + MP3 (MP4). H.264 es una norma que define un códec de vídeo de alta compresión. Un codec basado en el estándar H.264 comprime los archivos de video ocupando solo la mitad del espacio que el estándar MPEG-2. El sonido utiliza MP3 como formato de compresión. MP3 utiliza una compresión con pérdida de calidad que elimina las porciones del sonido que no son audibles para el oído humano. Suele utilizar tasas de compresión entre 64 y 128 kbps.

En el cliente, además de especificar el puerto, se indicará también la dirección ip del servidor. Que en este caso será la dirección IP del host h1, tal y como se muestra en la figura 44.

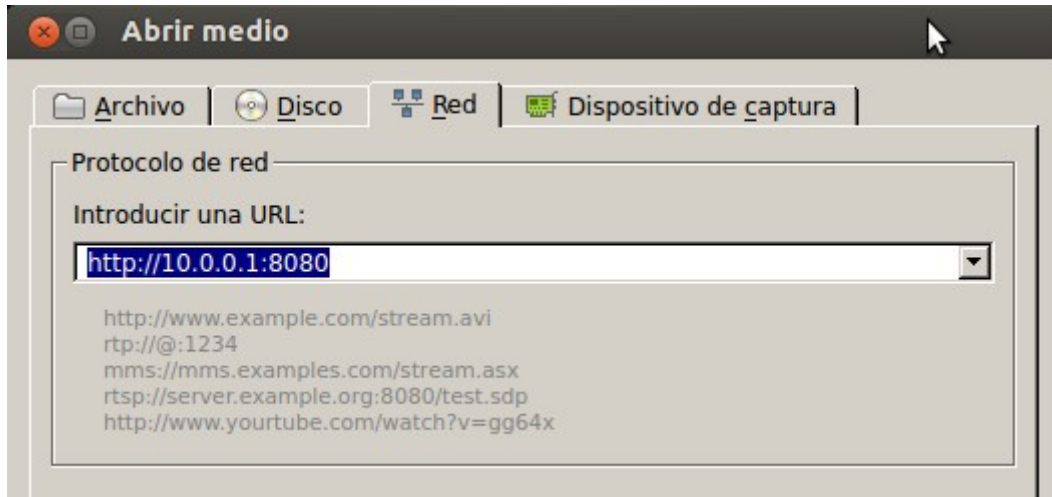


Figure 44 . Video transmission configuration in the client.

Una vez que comienza la emisión se llevará a cabo una captura del tráfico mediante wireshark, tanto en el controlador como en la interfaz s1-eth1. Los resultados de estas capturas en el controlador se muestran en la figura 45. En esta figura vemos como existen en el controlador mensajes de eco de petición y respuesta del protocolo OpenFlow.

No.	Time	Source	Destination	Protocol	Length	Info
2	1.114616	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
3	1.115130	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
26	6.112067	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
27	6.112714	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
49	11.112050	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
50	11.112691	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
70	16.112176	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
71	16.112813	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
107	21.112652	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
108	21.113093	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
138	26.112080	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
139	26.113090	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
162	31.112098	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
163	31.112891	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply
201	36.111553	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_request
202	36.112275	192.168.1.43	192.168.1.43	OF 1.0	74	of_echo_reply

Figure 45. Openflow messages in the controller

En la interfaz s1-eth1 nos encontramos con el tráfico HTTP propiamente dicho de la transmisión de video, tal y como se muestra en la figura 46.

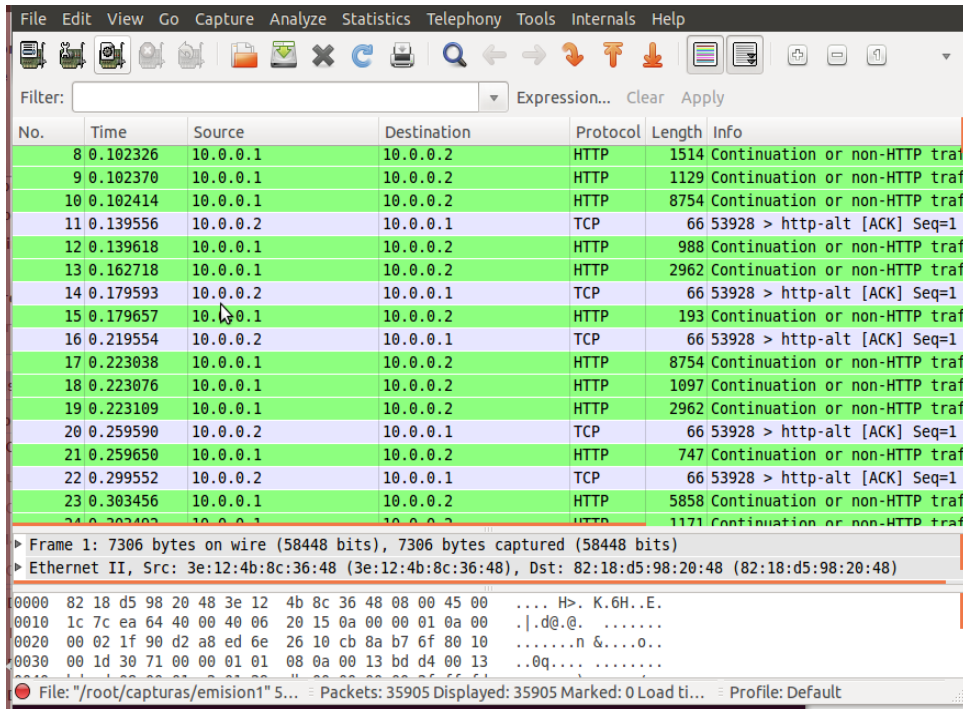
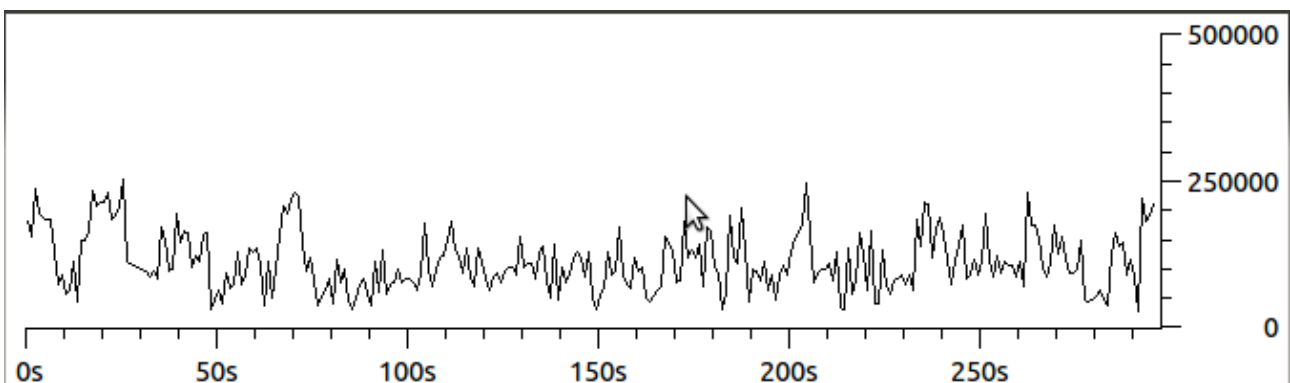


Figure 46. Wireshark capture of video transmission HTTP

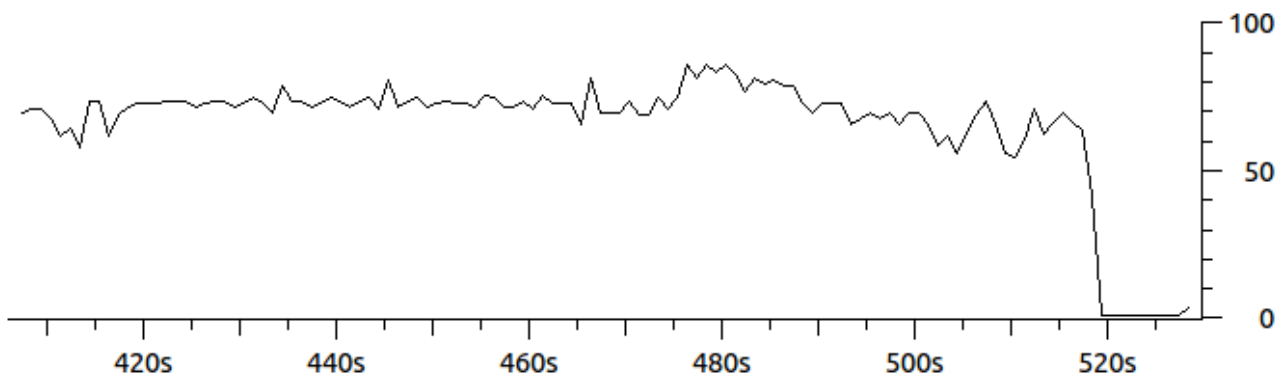
La funcionalidad del protocolo HTTP es permitir la transferencia de archivos con una comunicación del tipo cliente-servidor. Observando la gráfica, donde se nos muestra el throughput de la transmisión, podemos ver que este experimenta ciertas variaciones durante la transmisión. Estas variaciones aparecen en la gráfica 8 en forma de picos.



Graph 8. HTTP Transmisión

Estos picos son debidos a que el protocolo HTTP en el receptor almacena en un buffer las tramas recibidas. Estas tramas se transmiten desde el emisor y se almacenan en el buffer. Cuando este ya no puede almacenar ninguna trama más, las muestra en el emisor.

Aunque existen picos en la transmisión, podemos observar también que existe flujos de paquetes del tipo HTTP durante toda la transmisión. Este hecho se puede ver con más detalle en la gráfica 9.



*Graph 9. HTTP packets*

En la gráfica anterior, se puede observar que el flujo de paquetes HTTP es continuo en todo el intervalo de tiempo en el que se ha llevado a cabo la transmisión. Esto se debe a que, gracias a la caidad del enlace es buena, el buffer tiene que estar continuamente mostrando las tramas.

Este hecho está en consonancia con los resultados obtenidos en las pruebas de rendimiento de la red, puesto que en ellas, los resultados obtenidos demostraban que la transmisión de datos entre los nodos de la red se mantiene constante y fiable. Todo esto tiene como consecuencia que la calidad del vídeo transmitido será óptima, tal y como se puede observar en la figura 47. Aunque el ancho de banda durante la transmisión se mantiene prácticamente constante, hay que tener en cuenta que existe una variación en la latencia de la red, que aunque sea bastante pequeña hace que la visualización del vídeo en el emisor tenga un pequeño retardo.

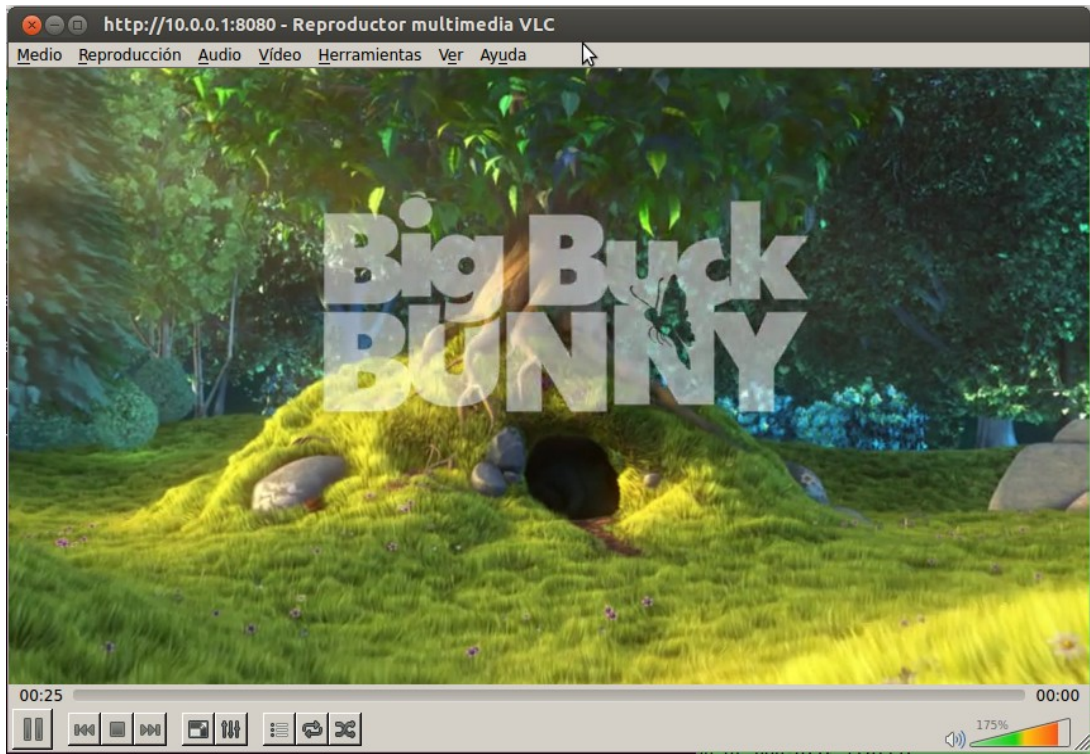
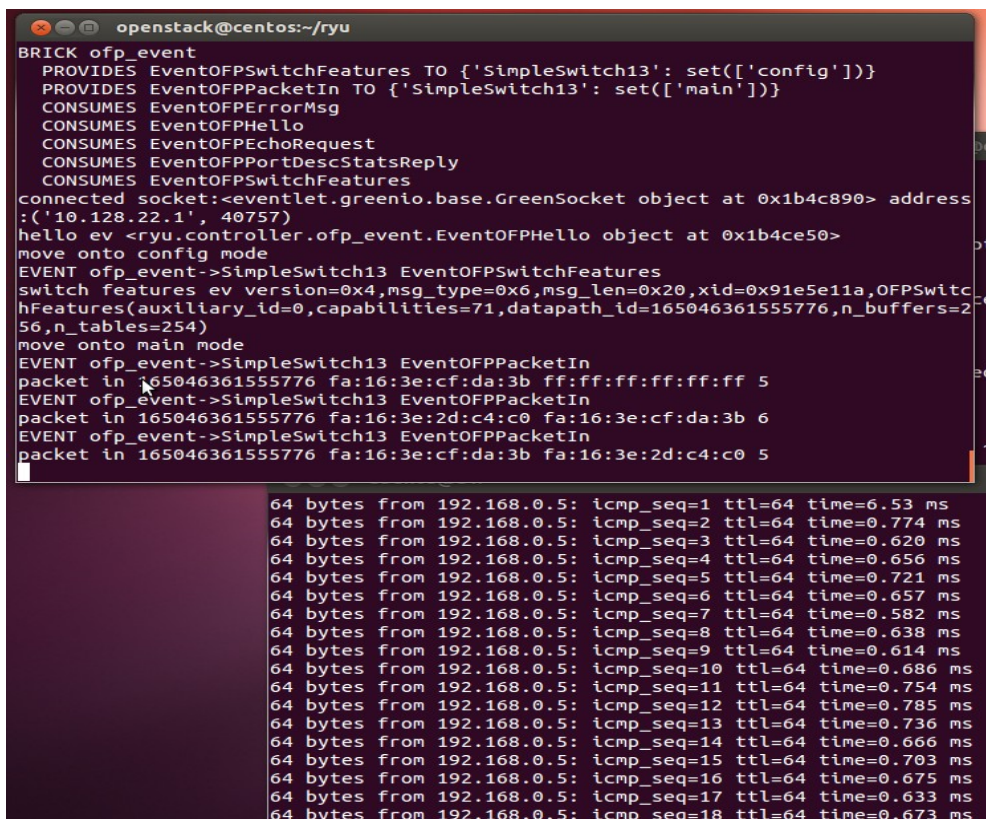


Figure 47. Transmitted video



## 5.4 Pruebas en CPD

Una vez realizadas las pruebas con tráfico sintético, se ha procedido a aplicar estos conocimientos al testbed diseñado en el capítulo anterior. Se ha comenzado haciendo las pruebas de conectividad mediante el comando ping. Disponemos de nueve máquinas virtuales que pueden ser utilizadas, pero solo se han utilizado dos, u1 y u2. Comprobaremos la conectividad entre las dos máquinas virtuales y capturaremos el tráfico utilizando la herramienta wireshark. Los resultados obtenidos se muestran en las figuras 48 y 49.



```
openstack@centos:~/ryu
BRICK ofp_event
PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
CONSUMES EventOFPErrormsg
CONSUMES EventOFPHello
CONSUMES EventOFPEchoRequest
CONSUMES EventOFPPortDescStatsReply
CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x1b4c890> address
:('10.128.22.1', 40757)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x1b4ce50>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x91e5e11a,OFPSwitch
Features(auxiliary_id=0,capabilities=71,datapath_id=165046361555776,n_buffers=2
56,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 165046361555776 fa:16:3e:cf:da:3b ff:ff:ff:ff:ff:ff 5
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 165046361555776 fa:16:3e:2d:c4:c0 fa:16:3e:cf:da:3b 6
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 165046361555776 fa:16:3e:cf:da:3b fa:16:3e:2d:c4:c0 5

64 bytes from 192.168.0.5: icmp_seq=1 ttl=64 time=6.53 ms
64 bytes from 192.168.0.5: icmp_seq=2 ttl=64 time=0.774 ms
64 bytes from 192.168.0.5: icmp_seq=3 ttl=64 time=0.620 ms
64 bytes from 192.168.0.5: icmp_seq=4 ttl=64 time=0.656 ms
64 bytes from 192.168.0.5: icmp_seq=5 ttl=64 time=0.721 ms
64 bytes from 192.168.0.5: icmp_seq=6 ttl=64 time=0.657 ms
64 bytes from 192.168.0.5: icmp_seq=7 ttl=64 time=0.582 ms
64 bytes from 192.168.0.5: icmp_seq=8 ttl=64 time=0.638 ms
64 bytes from 192.168.0.5: icmp_seq=9 ttl=64 time=0.614 ms
64 bytes from 192.168.0.5: icmp_seq=10 ttl=64 time=0.686 ms
64 bytes from 192.168.0.5: icmp_seq=11 ttl=64 time=0.754 ms
64 bytes from 192.168.0.5: icmp_seq=12 ttl=64 time=0.785 ms
64 bytes from 192.168.0.5: icmp_seq=13 ttl=64 time=0.736 ms
64 bytes from 192.168.0.5: icmp_seq=14 ttl=64 time=0.666 ms
64 bytes from 192.168.0.5: icmp_seq=15 ttl=64 time=0.703 ms
64 bytes from 192.168.0.5: icmp_seq=16 ttl=64 time=0.675 ms
64 bytes from 192.168.0.5: icmp_seq=17 ttl=64 time=0.633 ms
64 bytes from 192.168.0.5: icmp_seq=18 ttl=64 time=0.673 ms
```

Figure 48. Ping command in the controller

Como podemos ver en la figura anterior, se reciben en el controlador paquetes del tipo PACKET\_IN, correspondientes al comando ping. El primero de los paquetes será de broadcast y los siguientes serán los correspondientes al comando ping. Además, se puede ver el intercambio de paquetes icmp entre las dos máquinas virtuales. Este intercambio se puede observar en la figura 48. Como se observa esta figura, el ping se realiza satisfactoriamente, lo que significa que la conectividad de la red es satisfactoria.

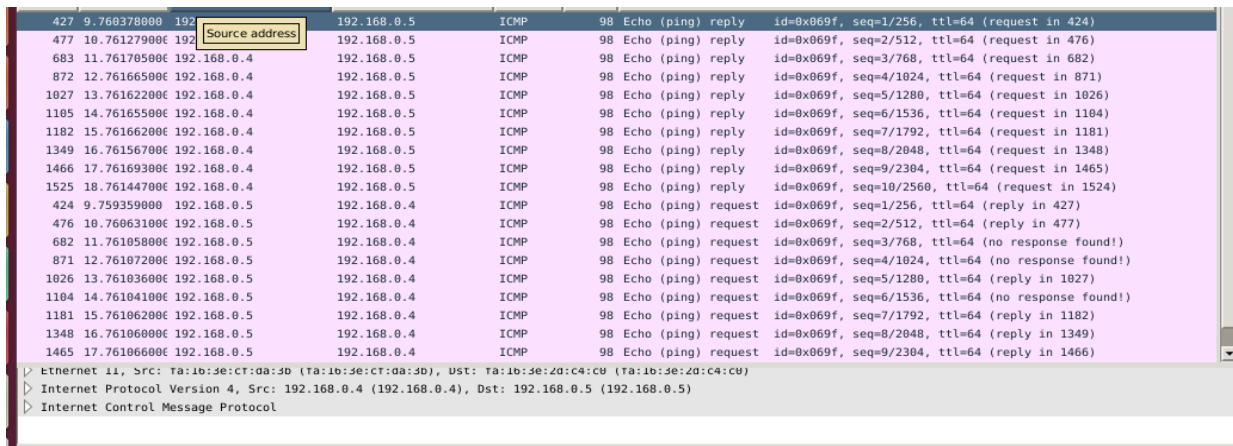


Figure 49. Ping between u1 and u2

El hecho de que no existan paquetes Openflow entre las máquinas u1 y u2 se debe a que el protocolo OpenFlow proporciona un canal seguro entre el controlador y el switch, haciendo que las comunicaciones entre las máquinas virtuales no tengan tráfico Openflow, reservandolo para las comunicaciones entre el switch y el controlador.

A continuación podemos ver el estado del switch mediante el comando `dpctl` tal y como hicimos con mininet. En este caso el comando sería.

```
ovs-dpctl show
```

Mediante este comando podemos ver las diferentes interfaces y los caminos de datos adjuntos a las mismas, como se muestra en la figura 50.

```

[root@centos ~]# ovs-dpctl show
system@ovs-system:
  lookups: hit:283287 missed:19433 lost:0
  flows: 11
  masks: hit:990626 total:4 hit/pkt:3.27
  port 0: ovs-system (internal)
  port 1: br-ex (internal)
  port 2: enp2s0f0
  port 3: br-int (internal)
  port 4: br-tun (internal)
  port 5: tapafae3cc0-ac (internal)
  port 6: qr-db6939d2-6b (internal)
  port 7: qvo6564e846-7a
  port 8: qg-1e0fce01-04 (internal)
  port 9: qvod86bc6ac-2b
  port 10: qvo9cad0bc8-4e
  port 11: qvodecf7f3c-1b
  port 12: qvo38993ab6-34
  port 13: qvo45072fdd-35
  port 14: qvo5e8f3857-5d
  port 15: qvod35df144-66
  port 16: qvoc89b948d-83

```

Figure 50 . ovs-dpctl show command

Para ver los flujos de datos que existen en el switch utilizaremos el comando:

```
ovs-dpctl dump-flows
```

Uno de estos flujos se muestra en la figura 51.

```

ovs-dpctl: opening datapath (No such device)
[root@centos ~]# ovs-dpctl dump-flows
recirc_id(0),skb_priority(0),in_port(2),eth(src=c2:9a:34:41:19:8d,dst=33:33:00:01:00:02),eth_type(0x86dd),ipv6(src=fe80::44b4:ef35:ecc:6c37/::,dst=ff02:
:1:2/::,label=0/0,proto=17/0,tclass=0/0,hlimit=1/0,frag=no/0xff), packets:0, bytes:0, used:never, actions:1,8
recirc_id(0),skb_priority(0),in_port(2),eth(src=18:a9:05:ea:e2:c5,dst=01:14:c2:44:1e:cc),eth_type(0/0xffff), packets:0, bytes:0, used:never, actions:1,8
recirc_id(0),skb_priority(0),in_port(2),eth(src=00:21:5a:f0:35:40,dst=ff:ff:ff:ff:ff:ff),eth_type(0x0806),arp(sip=10.128.3.2/255.255.255.255,tip=10.128.
5.3/255.255.255.255,op=1/0xff,sha=00:21:5a:f0:35:40/00:00:00:00:00:00,tha=00:00:00:00:00:00/00:00:00:00:00:00), packets:0, bytes:0, used:never, actions:
1,8

```

Figure 51. ovs-dpctl dump-flows command

A continuación se expone uno de estos flujos con más detalle:

```

[
  ID = 8
  Priority = 8
  In_port = 2
  eth ( src = c2:9a:34:41:19:8d, dst=33:33:08:01:00:02)
  eth_type (0x86dd)
  ipv6 (src=fe80::44b4:ef35:ecc:6c37/::, dst=ff02:1:2/::)

```

```
label = 8/0
proto= 17/0
tclass = 0/8
frag=no/0xff
```

]

Podemos ver como aparecen las características de este tipo de flujos coincidentes, entre las que destacan:

- in\_port hace referencia al número del puerto receptor
- eth src y eth dst son las direcciones MAC de origen y destino
- eth type es el tipo de trama Ethernet
- ipv6 src y dst son las direcciones IPv6 de origen y destino

Este comando es útil para conocer cuales son los flujos que se mantienen en el módulo de kernel de OpenvSwitch. Se reflejan paquetes que han pasado por la red en los últimos segundos. Esta herramienta nos sirve para verificar que los flujos han sido implementados de la forma esperada.

Por otra parte, para conocer las entradas en las tablas de flujo se usa el comando siguiente, cuyo resultado se muestra en la figura 52.

```
ovs-ofctl dump-flows br-int
```

```
cookie=0, duration_sec=354s, duration_nsec=755000000s, table_id=0, priority=32768, n_packets=11, n_bytes=1022, idle_timeout=0, hard_timeout=0, in_port=2, dl_dst=00:00:00:00:00:01, actions=output:1
cookie=0, duration_sec=354s, duration_nsec=753000000s, table_id=0, priority=32768, n_packets=10, n_bytes=924, idle_timeout=0, hard_timeout=0, in_port=1, dl_dst=00:00:00:00:00:02, actions=output:2
```

Figure 52. ovs-ofctl dump-flows br-int command

Como se puede ver en la figura anterior, se establecen dos entradas en la tabla de flujo. Esta utilidad es muy útil a la hora de consultar y manejar los switches y controladores OpenFlow. Adicionalmente, es posible consultar y verificar la configuración del bridge de integración (br-int) de OpenvSwitch con el comando

```
brctl show
```

Este comando nos permite conocer el ID del bridge y las interfaces asociadas a él, como se muestra en la figura 53.

```
[root@centos ~]# brctl show
bridge name      bridge id        STP enabled     interfaces
qbr38993ab6-34   8000.2e87df177b91  no              qvb38993ab6-34
tap38993ab6-34   qvb38993ab6-34
qbr45072fdd-35   8000.7eb957355341  no              qvb45072fdd-35
tap45072fdd-35   qvb45072fdd-35
qbr5e8f3857-5d   8000.7a44e2433ae4  no              qvb5e8f3857-5d
tap5e8f3857-5d   qvb5e8f3857-5d
qbr6564e846-7a   8000.9e5fdc7f4016  no              qvb6564e846-7a
tap6564e846-7a   qvb6564e846-7a
qbr9cad0bc8-4e   8000.822721b8f68b  no              qvb9cad0bc8-4e
tap9cad0bc8-4e   qvb9cad0bc8-4e
qbrc89b948d-83   8000.8ea5c4360d9b  no              qvbc89b948d-83
tapc89b948d-83   qvbc89b948d-83
qbrd35df144-66   8000.fe163ee1a89c  no              qvbd35df144-66
tapd35df144-66   qvbd35df144-66
qbrd86bc6ac-2b   8000.e2fade1b0c54  no              qvbd86bc6ac-2b
tapd86bc6ac-2b   qvbd86bc6ac-2b
qbrdecf7f3c-1b   8000.b6f6ba0be906  no              qvbdecf7f3c-1b
tapdecf7f3c-1b   qvbdecf7f3c-1b
```

Figure 53. brctl show command

Para obtener información sobre los procesos que se crean al iniciar OpenvSwitch se puede utilizar el siguiente comando.

```
ps -ea | grep ovs
```

```
[root@centos ~]# ps -ea | grep ovs
875 ?        00:00:55  ovsdb-server
904 ?        00:17:29  ovs-vswitchd
4305 ?       00:00:00  ovsdb-client
```

Figure 54. ps -ea | grep ovs command

Como se indica en la figura 54 mostrada anteriormente,, se crean tres procesos: ovsdb-server, ovs-vswitchd y ovsdb-client. El primero de estos procesos es un servidor de base de datos que OpenvSwitch consulta para obtener su configuración, es decir, en esta base de datos se encuentran todos sus parámetros de configuración necesarios. OVS-vswitchd es un demonio que implementa las funcionalidades del switch junto con su módulo kernel para conmutación basada en flujos. Por su parte, ovsdb-client es una herramienta de línea de comandos que permite interactuar con el proceso ovsdb-server.

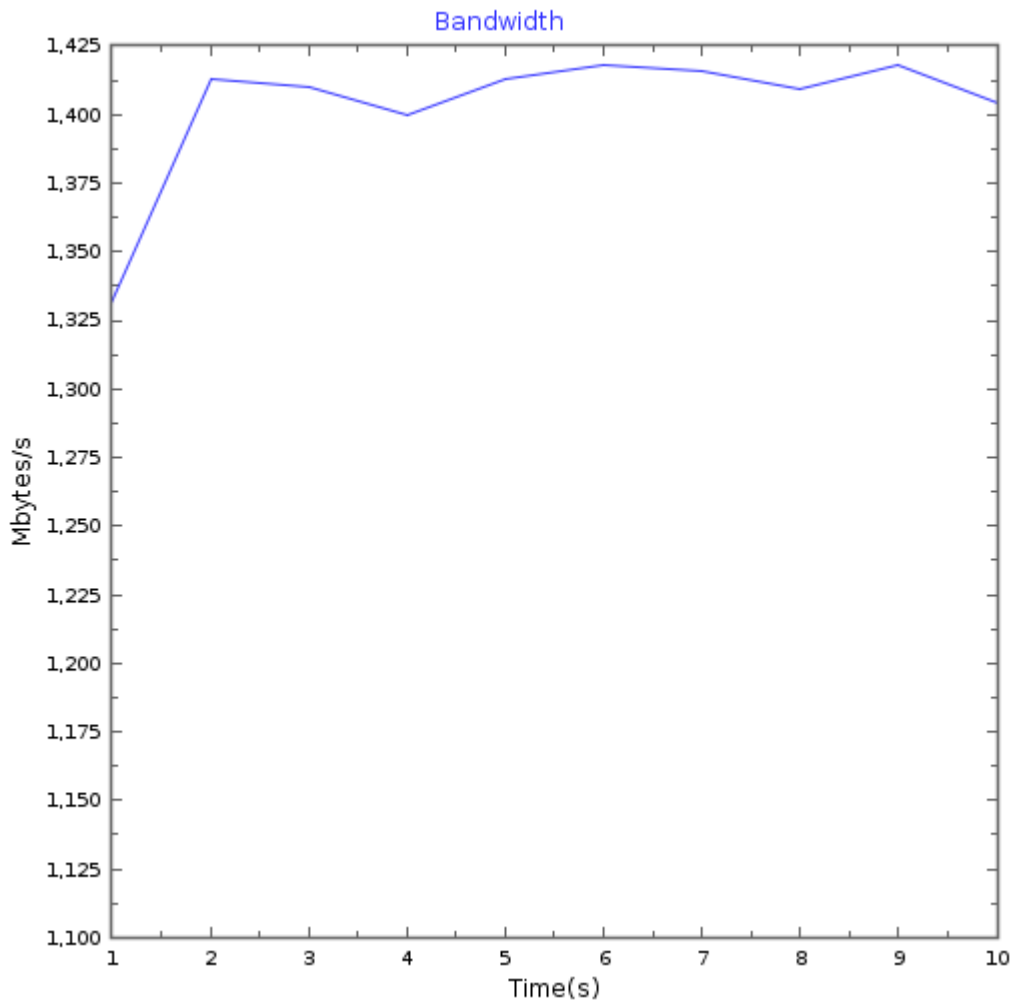
A continuación veremos cual es el rendimiento de la red. Comenzaremos midiendo la latencia de la red. Para ello, haremos una ejecución del comando ping entre las dos máquinas virtuales, tal y como se muestra en la figura 55.

```
ubuntu@u2:~$ ping 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(84) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=64 time=0.800 ms
64 bytes from 192.168.0.4: icmp_seq=2 ttl=64 time=0.650 ms
64 bytes from 192.168.0.4: icmp_seq=3 ttl=64 time=0.582 ms
64 bytes from 192.168.0.4: icmp_seq=4 ttl=64 time=0.644 ms
64 bytes from 192.168.0.4: icmp_seq=5 ttl=64 time=0.750 ms
64 bytes from 192.168.0.4: icmp_seq=6 ttl=64 time=0.626 ms
64 bytes from 192.168.0.4: icmp_seq=7 ttl=64 time=0.629 ms
64 bytes from 192.168.0.4: icmp_seq=8 ttl=64 time=0.745 ms
64 bytes from 192.168.0.4: icmp_seq=9 ttl=64 time=0.643 ms
64 bytes from 192.168.0.4: icmp_seq=10 ttl=64 time=0.753 ms
64 bytes from 192.168.0.4: icmp_seq=11 ttl=64 time=0.771 ms
64 bytes from 192.168.0.4: icmp_seq=12 ttl=64 time=0.622 ms
^C
--- 192.168.0.4 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 1099
rtt min/avg/max/mdev = 0.582/0.684/0.800/0.075 ms
```

Figure 55. Ping between u1 and u2

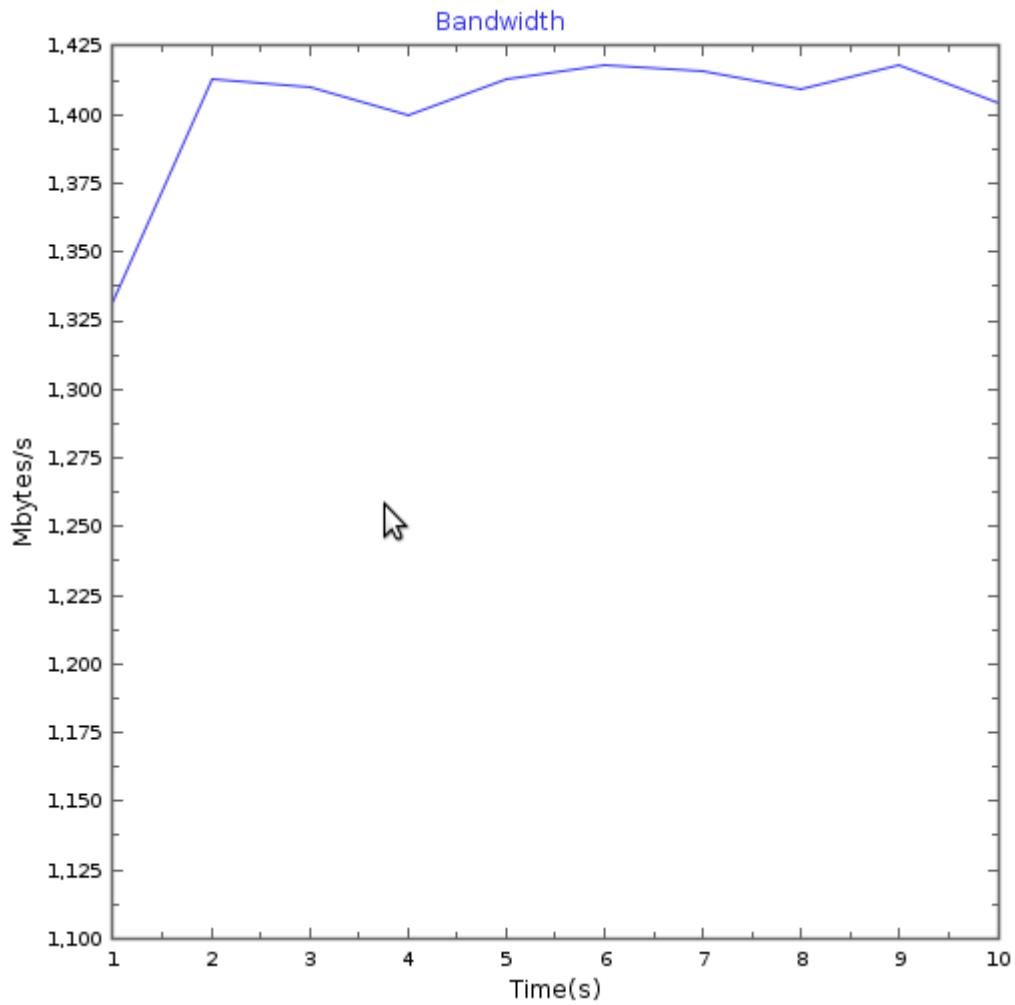
Para ver la latencia de la red, nos fijaremos, al igual que hicimos con mininet, en el parámetro rtt. Si nos fijamos en el valor medio de este parámetro, se observa que la red tiene una latencia de 0.684 ms.

A continuación, se llevan a cabo las pruebas relacionadas con el throughput y la pérdida de paquetes en la red. Para ello, se utilizó la herramienta Jperf. Comenzaremos con la medida del throughput de la red, mediante un flujo de tráfico TCP durante 10 segundos. El tamaño de la ventana es de 0,04 Mbytes, valor por defecto. Las gráficas 10 y 11 proporcionan una representación gráfica de esta medida, tanto en el cliente como en el servidor.



*Graph 10. Bandwidth in the server*

Observando las gráficas 10 y 11, se puede concluir que la transmisión de datos entre las dos máquinas de la red es fiable. No existe diferencia entre el ancho de banda obtenido en el servidor y en el cliente. En este caso, el funcionamiento de TCP no influye durante la transmisión y el canal se mantiene activo durante todo el intervalo de tiempo de la transmisión.



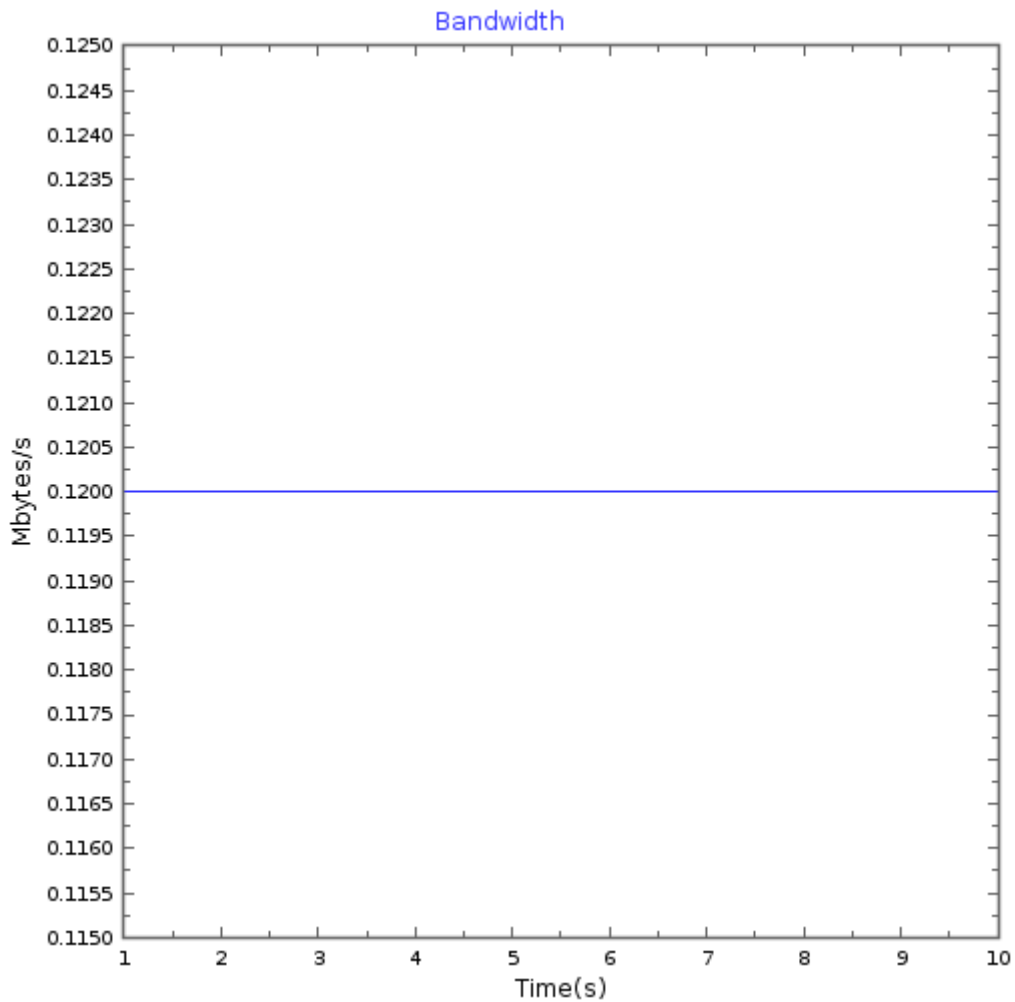
Graph 11. Bandwidth in the client

Realizamos a continuación una transmisión UDP de 10 segundos para comprobar si existe pérdida de paquetes. Se enviarán datagramas con un tamaño de 1470 bytes, con un tamaño de buffer de 0,20 Mbytes. Las estadísticas obtenidas se muestran a continuación:

[ ID]	Interval	Transfer	Bandwidth	Jitter	Lost/Total Datagrams
[ 3]	0.0-10.0 sec	1.19 MBytes	0.12 MBytes/sec	0.083 ms	0/ 852 (0%)

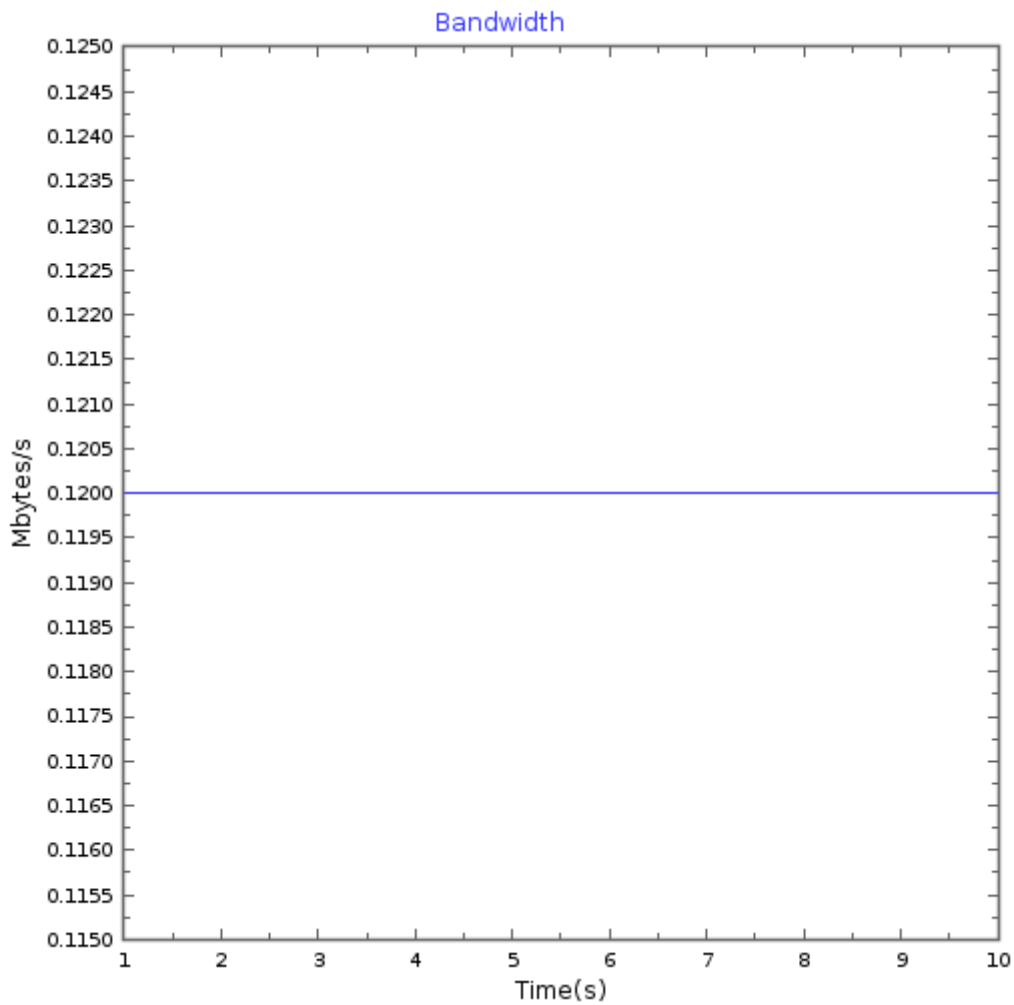


Basándonos en la información mostrada, se observa como no existe pérdida de paquetes en la transmisión. Este hecho quiere decir que tenemos una transmisión fiable. Los resultados de la transmisión se muestran en las gráficas 12 y 13.



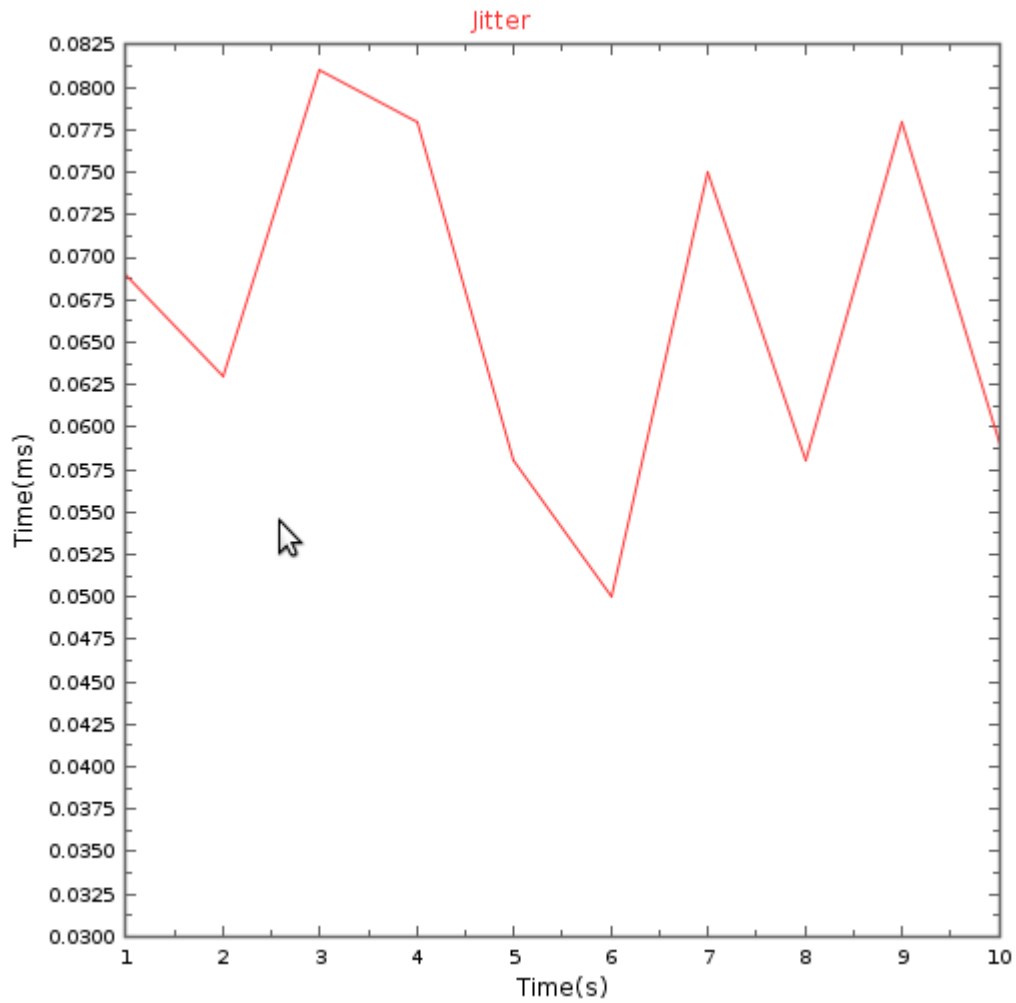
*Graph 12. UDP transmission in the server*

Como se puede observar en las gráficas 12 y 13, vemos como toda la información transmitida en llega al cliente siendo el ancho de banda constante en todo el proceso. Esto implica una transmisión fiable a la hora de el envío de datos.



Graph 13. *UDP transmission in the client*

A continuación estudiaremos la calidad del enlace mediante el jitter existente durante la transmisión. La medida del jitter se ha realizado mediante el tráfico UDP y se muestra en la gráfica 14.



Graph 14. Jitter

El valor medio del jitter es de 0,083 ms. Este valor puede considerarse bajo, por lo que la calidad del enlace es alta. Una vez visto los parámetros de rendimiento, se han realizado las pruebas de transmisión de video mediante el protocolo HTTP.

## Transmisión de video HTTP

Veremos ahora lo que ocurre al hacer una transmisión de video con el protocolo HTTP. Se ha tomado como servidor el ordenador personal y como cliente la máquina virtual u2. En primer lugar veremos lo que ocurre en el controlador. Para ello se realizó una captura de tráfico mediante la herramienta tcpdump. Para facilitar la comprensión de esta comunicación, se ha guardado la captura en un archivo del tipo .cap de forma que la visualización de la información se pueda llevar a cabo mediante wireshark. Los resultados obtenidos se muestran en la figura 56.

1	0.000000	10.128.22.1	10.128.22.1	OpenFlow	76	Type: OFPT_ECHO_REQUEST
2	0.000414	10.128.22.1	10.128.22.1	OpenFlow	76	Type: OFPT_ECHO_REPLY
3	0.000441	10.128.22.1	10.128.22.1	TCP	68	45330->6633 [ACK] Seq=9 Ack=9 Win=342
4	4.999520	10.128.22.1	10.128.22.1	OpenFlow	76	Type: OFPT_ECHO_REQUEST
5	5.039200	10.128.22.1	10.128.22.1	TCP	68	6633->45330 [ACK] Seq=9 Ack=17 Win=421
6	10.000655	10.128.22.1	10.128.22.1	TCP	68	45330->6633 [FIN, ACK] Seq=17 Ack=9 Wi
7	10.040249	10.128.22.1	10.128.22.1	TCP	68	6633->45330 [ACK] Seq=9 Ack=18 Win=421
8	10.999976	10.128.22.1	10.128.22.1	TCP	76	45419->6633 [SYN] Seq=0 Win=43690 Len=
9	11.000000	10.128.22.1	10.128.22.1	TCP	76	6633->45419 [SYN, ACK] Seq=0 Ack=1 Win
10	11.000017	10.128.22.1	10.128.22.1	TCP	68	45419->6633 [ACK] Seq=1 Ack=1 Win=4377
11	11.000047	10.128.22.1	10.128.22.1	OpenFlow	84	Type: OFPT_HELLO
12	11.000057	10.128.22.1	10.128.22.1	TCP	68	6633->45419 [ACK] Seq=1 Ack=17 Win=437
13	12.000131	10.128.22.1	10.128.22.1	TCP	68	45419->6633 [FIN, ACK] Seq=17 Ack=1 Wi
14	12.040186	10.128.22.1	10.128.22.1	TCP	68	6633->45419 [ACK] Seq=1 Ack=18 Win=437
15	14.000279	10.128.22.1	10.128.22.1	TCP	76	45420->6633 [SYN] Seq=0 Win=43690 Len=
16	14.000298	10.128.22.1	10.128.22.1	TCP	76	6633->45420 [SYN, ACK] Seq=0 Ack=1 Win
17	14.000314	10.128.22.1	10.128.22.1	TCP	68	45420->6633 [ACK] Seq=1 Ack=1 Win=4377
18	14.000344	10.128.22.1	10.128.22.1	OpenFlow	84	Type: OFPT_HELLO

Figure 56. Video transmission in the controller

Analizando la figura anterior, vemos como existe tráfico de varios protocolos, como TCP y OpenFlow. Centrándonos en el protocolo OpenFlow, se puede observar que existen mensajes de petición y respuesta del tipo OF\_ECHO\_REQUEST y OF\_ECHO\_REPLY además de mensajes OFPT\_HELLO.

Cabe destacar que los mensajes OFPT\_HELLO, que proporcionan el establecimiento de la conexión, no aparecen sólo al comienzo de la misma. Esto es debido a que la transmisión se ha detenido en algún momento y la conexión entre el controlador y las máquinas ha tenido que ser restablecida de nuevo.

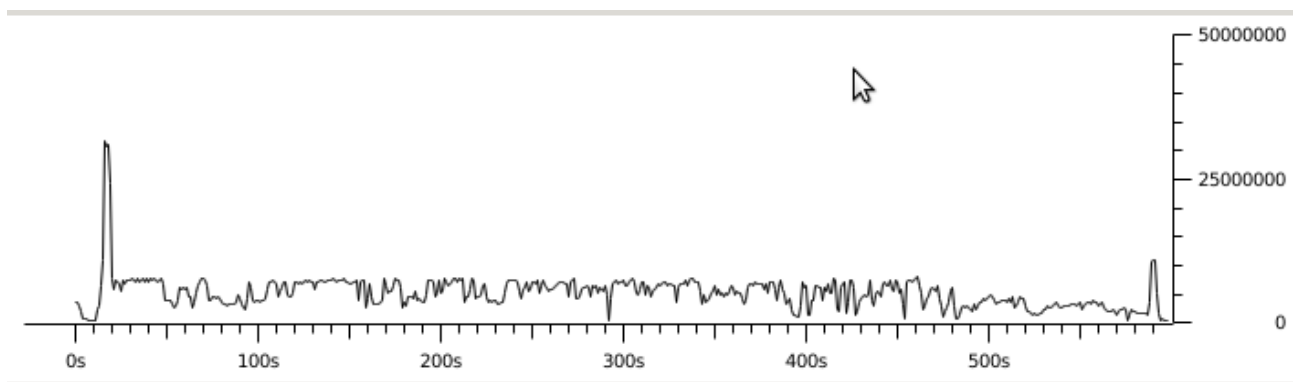
La aparición de tráfico TCP se debe a que este protocolo da soporte al protocolo HTTP. TCP utiliza el concepto número de puerto para identificar las aplicaciones. HTTP utiliza los puertos bien conocidos de este protocolo. En este caso el puerto usado es el 8080.

Analizamos ahora el tráfico que intercambiado durante la transmisión del video entre la máquina personal y la máquina virtual. Para ello se analizará el contenido mostrado en la figura 57 y en la gráfica 15. En esta captura de wireshark, se observa el intercambio de paquetes HTTP entre las dos máquinas.

Time	Source	Destination	Protocol	Length	Info
1015	0.736714000	192.168.60.24	HTTP	936	[TCP Previous segment not captured] C
1696	1.326570000	192.168.60.24	HTTP	2762	Continuation
8462	14.346918000	192.168.60.24	HTTP	2762	Continuation
10372	15.262666000	192.168.60.24	HTTP	2762	Continuation
67225	29.546525000	192.168.60.24	HTTP	1414	Continuation
72178	31.626915000	192.168.60.24	HTTP	2762	Continuation
78920	34.567604000	192.168.60.24	HTTP	1218	Continuation
83832	36.672813000	192.168.60.24	HTTP	2762	Continuation
84838	37.047090000	192.168.60.24	HTTP	2762	Continuation
87711	38.086558000	192.168.60.24	HTTP	2762	Continuation
95013	41.005555000	192.168.60.24	HTTP	2762	Continuation
128568	59.506008000	192.168.60.24	HTTP	2762	Continuation
130024	60.463033000	192.168.60.24	HTTP	1414	Continuation
131191	60.589260000	192.168.60.24	HTTP	1414	Continuation
141462	67.170814000	192.168.60.24	HTTP	2762	Continuation
142454	67.587028000	192.168.60.24	HTTP	2762	Continuation
156934	73.334524000	192.168.60.24	HTTP	2762	Continuation

Figure 57. HTTP video transmission

Prestando atención a la gráfica 14, se observa como al comienzo de la transmisión existe un pico que sobresale. Esto es debido a que el protocolo HTTP almacena en un buffer los paquetes al comienzo de la transmisión y después comienza a emitir. En el resto de la transmisión vemos que existen ciertos picos de bajada, debido a que existe un retardo en la transmisión entre los datos emitidos y los recibidos considerable, lo que hace que se ralentice.



Graph 15. Video transmission with HTTP

El resultado de la transmisión de vídeo es óptimo, aunque se puede apreciar una leve pérdida de calidad respecto al vídeo original. Al comienzo de la transmisión, la imagen emitida aparece en verde, como se muestra en la figura 58, pero a medida que la transmisión se hace más estable la imagen se estabiliza, como se observa en la figura 59.

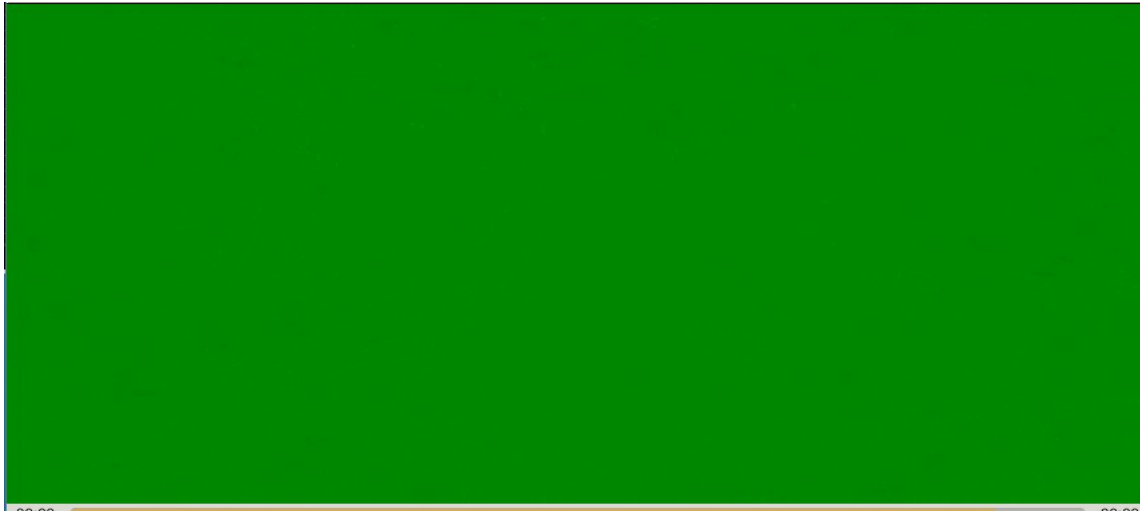


Figure 58. Image at the beginning of the transmission

La aparición de la pantalla en verde al comienzo de la transmisión se debe a que el almacenamiento en el buffer es lento, con lo cual el servidor empieza a mostrar la imagen con retardo.

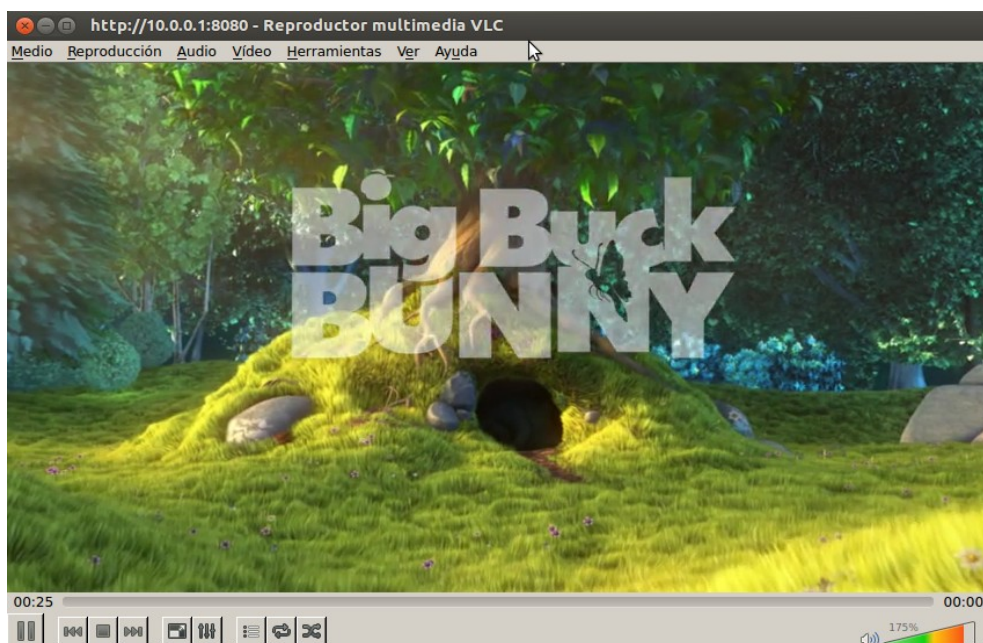


Figure 59. Image during the transmission

## 5.5 Comparativa de los resultados

Debido a que las pruebas realizadas en ambos entornos son iguales, parece interesante realizar una comparativa entre los resultados obtenidos en cada una de ellas.

Comenzando con la prueba de conectividad, se ha podido observar que los valores del comando ping son mayores en el entorno real que en el entorno emulado. En el entorno de Mininet el paquete ICMP se encuentra con menos elementos que puedan causar un retraso en la comunicación. Sin embargo, en un entorno real es más probable que exista un retardo en la comunicación, que dificulta el desarrollo normal de la misma, provocando una pérdida de calidad en la conectividad. En este caso, la comunicación con el entorno real se realiza a distancia mediante ssh, lo que hace que se introduzca ese retardo en la prueba de conectividad.

Los valores medios del comando ping son de 0,288 ms en el entorno emulado y 0,6852 ms en el entorno real. Los valores de esta prueba dependerán del tipo de servicio, pero se puede establecer un valor 200 ms como frontera para saber si la calidad de la conectividad es óptima o no. Si el valor está por debajo se puede considerar que existe una buena conectividad; sin embargo, un valor más alto que 200 ms quiere decir que la conectividad no es óptima. Observando los valores obtenidos en las pruebas, es posible concluir que la conectividad es muy buena en ambos casos.

En cuanto a las pruebas de rendimiento, se muestra la tabla 9, en la que se realiza una comparación de los resultados obtenidos. Se han utilizado valores medios. En ella, se puede ver como la latencia de la red es mayor en el testbed diseñado con máquinas físicas que en la red emulada con mininet. Esto quiere decir que existe mayor retardo en la red real que en la red emulada. La razón de este retardo viene a que en un entorno real las rutas que deben atravesar los paquetes son mayores que en el entorno virtual.

En cuanto al throughput podemos ver que se obtienen mayores más altos en el entorno emulado que en el entorno real. Esto es debido a que en el entorno emulado no existen factores que afecten al rendimiento del enlace sino que su comportamiento se asemeja mucho a un comportamiento ideal mientras que en el entorno real, la utilización del ancho de banda del enlace no es total.

Prueba realizada	Entorno real	Entorno emulado
Latencia	0.684 ms	1.419 ms
Throughput	1364 Mbytes/s	1403 Mbytes/s
Jitter	0.083 ms	0.022 ms
Pérdida de paquetes	0 %	0 %

Table 10. Performance comparative

El jitter es mayor en el entorno real, lo que hace que la calidad de los enlaces sea menor en la red diseñada real que en la red emulada con Mininet. La pérdida de paquetes es del 0% para ambas opciones, lo que hace que la transmisión de tráfico en la red es estable en ambos casos.

## 5.6 QoS en SDN

En este apartado se mostrará un ejemplo de implementación de QoS, añadiendo establecimiento de colas y reglas para reservar ancho de banda en la red. Se usará la misma topología usada anteriormente con Mininet.

En primer lugar se invocará Mininet con el siguiente comando:

```
sudo mn --mac --switch ovsk --controller remote -x
```

A continuación, se establecerá la versión de OpenFlow a utilizar (en este caso será la versión 1.3) y se establecerá como puerto de escucha el puerto 6632 para acceder a OVSDB. Para ello se utilizarán los siguientes comandos, mostrados en la figura, en el terminal del switch s1:

```
ovs-vsctl set Bridge s1 protocols=OpenFlow13
ovs-vsctl set-manager ptcp:6632
```



A continuación, se modifica la aplicación `simple_switch_13.py` para que registre la entrada de flujo en la tabla con `id:1`. Para ello se utilizan los comandos siguientes en el terminal del controlador `c0`.

```
sed '/OFPFlowMod(\/,\/)\/s\/)\/, table_id=1)\/' ryu/ryu/app/simple_switch_13.py  
> ryu/ryu/app/qos_simple_switch_13.py  
  
cd ryu; python ./setup.py install
```

Finalmente, ejecutamos Ryu con las aplicaciones `rest_qos`, `qos_simple_switch_13` y `rest_conf_switch` en el terminal del controlador `c0`. El comando utilizado será el siguiente (figura 60):

```
ryu-manager ryu.app.rest_qos ryu.app.qos_simple_switch_13 ryu.app.rest_conf_switch
```



```
root@pedro-TravelMate-5742:~/ryu# ryu-manager ryu.app.rest_qos ryu.app.qos_simp  
le_switch_13 ryu.app.rest_conf_switch  
loading app ryu.app.rest_qos  
loading app ryu.app.qos_simple_switch_13  
loading app ryu.app.rest_conf_switch  
loading app ryu.controller.ofp_handler  
instantiating app None of DPSet  
creating context dpset  
instantiating app None of ConfSwitchSet  
creating context conf_switch  
creating context wsgi  
instantiating app ryu.app.rest_conf_switch of ConfSwitchAPI  
instantiating app ryu.app.qos_simple_switch_13 of SimpleSwitch13  
instantiating app ryu.controller.ofp_handler of OFPHandler  
instantiating app ryu.app.rest_qos of RestQoSAPI  
(8528) wsgi starting up on http://192.168.60.24:8080/  
[QoS][INFO] dpid=0000000000000001: Join qos switch.
```

Figure 60 . Ryu startup with QoS

A continuación procederemos a establecer las colas en el switch. Para ello, en primer lugar se establecerá `ovsdb_addr` con el fin de acceder a `OVSDb` y ejecutaremos el establecimiento de colas. Se ejecutarán los siguientes comandos en el controlador.

```
curl -X PUT -d '"tcp:127.0.0.1:6632"' http://localhost:8080/v1.0/conf/switches/
curl -X POST -d '{"port_name": "s1-eth1", "type": "linux-htb",
"max_rate": "1000000", "queues": [{"max_rate": "500000"}, {"min_rate":
"800000"}]}' http://localhost:8080/qos/queue/00000000000000001
```

El resultado obtenido es el siguiente:

```
"switch:id": "00000000000000001" ,
  "command_result": {
    "result": "success",
    "details": {
      "0": {
        "config": {
          "max-rate": "500000"
        }
      },
      "1": {
        "config": {
          "min-rate": "800000"
        }
      }
    }
  }
```

Instalamos ahora una entrada de flujo en el switch con las características mostradas en la tabla 6.

Prioridad	Dirección destino	Puerto destino	Protocolo	ID cola	ID QoS
1	10.0.0.1	5002	UDP	1	1

Table 11. Flow entry in the switch

El comando utilizado es el siguiente:

```
curl -X POST -d '{"match": {"nw_dst": "10.0.0.1", "nw_proto": "UDP", "tp_dst": "5002"}, "actions": {"queue": "1"}}'  
http://localhost:8080/qos/rules/0000000000000001
```

Para verificar el contenido de la configuración del switch utilizamos el comando:

```
curl -X GET http://localhost:8080/qos/rules/0000000000000001
```

```
"switch_id": "0000000000000001",  
"command_result": [  
  {  
    "result": "success",  
    "details": "QoS added. : qos_id=1"  }  
]
```

Una vez que se han hecho todos los pasos anteriores, se medirá el ancho de banda mediante el uso de iperf. En el siguiente ejemplo, h1(servidor) escucha en el puerto 5001 y 5002 con el protocolo UDP. h2 (cliente) envía 1 Mbps de tráfico UDP al puerto 5001 en h1 y 1 Mbps de tráfico UDP al puerto 5002 en h1. Primero abrimos dos terminales, para h1 y h2. Los comandos utilizados serán los siguientes:

En h1:

```
iperf -s -u -i 1 -p 5001  
iperf -s -u -i 1 -p 5002
```

En h2:

```
iperf -c 10.0.0.1 -p 5001 -u -b 1M  
iperf -c 10.0.0.1 -p 5002 -u -b 1M
```

Los resultados obtenidos se muestran en las figuras 61 y 62:

```

root@pedro-TravelMate-5742:~# iperf -s -u -i 1 -p 5002
-----
Server listening on UDP port 5002
Receiving 1470 byte datagrams
UDP buffer size: 160 KByte (default)
-----
[ 12] local 10.0.0.1 port 5002 connected with 10.0.0.2 port 44159
[ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total  Datagrams
[ 12] 0.0- 1.0 sec   119 KBytes    976 Kbits/sec  0.415 ms    0/ 83 (0%)
[ 12] 1.0- 2.0 sec   118 KBytes    964 Kbits/sec  0.433 ms    0/ 82 (0%)
[ 12] 2.0- 3.0 sec   118 KBytes    964 Kbits/sec  0.463 ms    0/ 82 (0%)
[ 12] 3.0- 4.0 sec   119 KBytes    976 Kbits/sec  0.424 ms    0/ 83 (0%)
[ 12] 4.0- 5.0 sec   118 KBytes    964 Kbits/sec  0.461 ms    0/ 82 (0%)
[ 12] 5.0- 6.0 sec   118 KBytes    964 Kbits/sec  0.637 ms    0/ 82 (0%)
[ 12] 6.0- 7.0 sec   118 KBytes    964 Kbits/sec  0.630 ms    0/ 82 (0%)
[ 12] 7.0- 8.0 sec   118 KBytes    964 Kbits/sec  0.652 ms    0/ 82 (0%)
[ 12] 8.0- 9.0 sec   119 KBytes    976 Kbits/sec  0.626 ms    0/ 83 (0%)
[ 12] 9.0-10.0 sec   118 KBytes    964 Kbits/sec  0.663 ms    0/ 82 (0%)
[ 12] 0.0-10.3 sec   1.19 MBytes   968 Kbits/sec  0.659 ms    0/ 852 (0%)
read failed: Connection refused

```

Figure 61. QoS transmission in h1

```

[ 12] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 43475
[ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total  Datagrams
[ 12] 0.0- 1.0 sec   60.3 KBytes    494 Kbits/sec  11.653 ms    0/ 42 (0%)
[ 12] 1.0- 2.0 sec   58.9 KBytes    482 Kbits/sec  12.498 ms    0/ 41 (0%)
[ 12] 2.0- 3.0 sec   58.9 KBytes    482 Kbits/sec  12.556 ms    0/ 41 (0%)
[ 12] 3.0- 4.0 sec   58.9 KBytes    482 Kbits/sec  12.563 ms    0/ 41 (0%)
[ 12] 4.0- 5.0 sec   58.9 KBytes    482 Kbits/sec  12.560 ms    0/ 41 (0%)
[ 12] 5.0- 6.0 sec   58.9 KBytes    482 Kbits/sec  12.559 ms    0/ 41 (0%)
[ 12] 6.0- 7.0 sec   58.9 KBytes    482 Kbits/sec  12.559 ms    0/ 41 (0%)
[ 12] 7.0- 8.0 sec   58.9 KBytes    482 Kbits/sec  12.561 ms    0/ 41 (0%)
[ 12] 8.0- 9.0 sec   60.3 KBytes    494 Kbits/sec  12.564 ms    0/ 42 (0%)
[ 12] 9.0-10.0 sec   58.9 KBytes    482 Kbits/sec  12.561 ms    0/ 41 (0%)
[ 12] 10.0-11.0 sec  58.9 KBytes    482 Kbits/sec  12.557 ms    0/ 41 (0%)
[ 12] 11.0-12.0 sec  58.9 KBytes    482 Kbits/sec  12.560 ms    0/ 41 (0%)
[ 12] 12.0-13.0 sec  58.9 KBytes    482 Kbits/sec  12.561 ms    0/ 41 (0%)
[ 12] 13.0-14.0 sec  58.9 KBytes    482 Kbits/sec  12.563 ms    0/ 41 (0%)
[ 12] 14.0-15.0 sec  58.9 KBytes    482 Kbits/sec  12.561 ms    0/ 41 (0%)
[ 12] 15.0-16.0 sec  58.9 KBytes    482 Kbits/sec  12.561 ms    0/ 41 (0%)
[ 12] 16.0-17.0 sec  60.3 KBytes    494 Kbits/sec  12.557 ms    0/ 42 (0%)
[ 12] 17.0-18.0 sec  58.9 KBytes    482 Kbits/sec  12.557 ms    0/ 41 (0%)
[ 12] 18.0-19.0 sec  58.9 KBytes    482 Kbits/sec  12.560 ms    0/ 41 (0%)
[ 12] 19.0-20.0 sec  58.9 KBytes    482 Kbits/sec  12.558 ms    0/ 41 (0%)
[ 12] 0.0-20.7 sec   1.19 MBytes    484 Kbits/sec  12.561 ms    0/ 852 (0%)
read failed: Connection refused

```

Figure 62. QoS transmission in h2

Observando las figuras anteriores vemos como el tráfico enviado al puerto 5001 está conformado con un ancho de banda de hasta 494 Kbps y el tráfico que circula por el puerto 5002 está garantizado con un ancho de banda de 976 Kbps.

# Chapter 6. Conclusions and future work

In this project, an analysis about Software-Defined Networking and its performance has been realized. To do that, two tools have been used, Mininet and OpenStack, using Ryu as the external controller. Is because of this fact that, the integration of this controller with the tools aforementioned has gained becomes extremely important. After having studied several controllers, Ryu controller has been chosen due to its greater flexibility when using it in conjunction with the utilities mentioned before.

The fact that Mininet performs emulation, allows to use real instruments and, thus, extrapolate the obtained results to real networks. The great integration with SDN functionalities makes it a very attractive utility with regard to any other, as working with Open vSwitch makes it very dynamic.

OpenStack, however, lets implementing a real Software Defined Network and enabling to check if tests realized with Mininet are appropriate. The implementation of the network with this tool is more complex, since its installation could be rather hard. In addition, its integration with external controllers is also difficult because some controllers, such as, OpenDaylight and Floodlight, are complex to manage and to configure.

Through the deploying of the SDN network it is verified that this type of network provides a new way of using computer network, since its capacity for network administrators to establish resoucers according to its needs has been checked. During the prototyping of the network rules are defined to certain packets flows and it was able to test how those flows can be adapted to changes by simply modify the rules previously set.

SDN networks can have a similar structure to traditional ones, the difference takes root in that is in the controller where the decisions related with the network traffic are taken, while in existing networks those decisions are made by network devices that realized switching and routing. As it has been mentioned before, Ryu controller has been chose as the most appropriate controller to be used in the implementation of the switch. Ryu has a wide range of applications, which allows to make a great number of performance tests without extreme difficulty and its integration with Mininet and OpenStack is quite simple.

In addition to this, the exchange of OpenFlow messages has been studied. OpenFlow is the protocol in which this types of networks are based. It has been proved that those messages only appear in the network traffic flowing between the controller and the switch, not among virtual machines. It can be concluded then that links between hosts are not in charge of the management of the network but this belongs to the controller. This means having a greater capacity in the data channel, which is an advantage to networks which big amounts of data and may be at risk of congestion.

#### *Future work*

There are several aspects which deeper study can be of great interest. Firstly, performance of tests using other SDN controllers, which would allow a comparison between them and determine which controller is the most suitable for a Software Defined Network. Secondly, the use of other controllers, such as OpenDaylight or Floodlight, with Openstack is another point to continue working on. This would let checking out the performance of a real network and knowing which one is the best when working in a real environment. In addition to this, it would be the great interest to know the behaviour of the network implementing customized applications and rules.

In this project, the SDN paradigm has been tested in static machines, so it will be interesting to test this types of networks in mobile networks. It could be applied he knowledge developed in this work for developing protocols under the SDN technology to manage mobility in networks.

# Anexes

## Anex I

### OpenStack installation

Commands to install OpenStack

Step 1: Software repositories

Update your current packages:

```
sudo yum update -y
```

Setup the RDO repositories:

```
sudo yum install -y https://rdoproject.org/repos/rdo-release.rpm
```

Step 2: Install Packstack Installer

```
sudo yum install -y openstack-packstack
```

Step 3: Run Packstack to install OpenStack

```
packstack --allinone
```

Step 4: Allow any machine on the network to be able to access launched instances via their floating Ips.

```
packstack --allinone --provision-demo=n
```

Make /etc/sysconfig/network-scripts/ifcfg-br-ex resemble:

```
DEVICE=br-ex  
DEVICETYPE=ovs  
TYPE=OVSBridge  
BOOTPROTO=static  
IPADDR= #ip address  
NETMASK=# your netmask  
GATEWAY= # your gateway  
DNS1= # your nameserver  
ONBOOT=yes
```

Make /etc/sysconfig/network-scripts/ifcfg-eth0

```
DEVICE=eth0  
HWADDR=# your hwaddr (find via ip link show eth0)  
TYPE=OVSPort  
DEVICETYPE=ovs  
OVS_BRIDGE=br-ex  
ONBOOT=yes
```



# Anex II

## Mininet installation

To install natively from source, first you need to get the source code:

```
git clone git://github.com/mininet/mininet
```

To check the version:

```
cd mininet
git tag # list available versions
git checkout -b
cd ..
```

The command to install Mininet is:

```
mininet/util/install.sh [options]
```

Typicall install.sh options include:

- -a: install everything that is included in Mininet VM, including dependencies like OpenvSwitch as well the additions like the OpenFlow wireshark dissector and POX.
- -nfv: install Mininet, the OpenFlow reference switch, and OpenvSwitch
- -s mydir: use this option before other options to place source/build tress in a specified directory rather that in the home directory

After the installation has completed, test the basic Mininet functionality:

```
sudo mn --test pingall
```

## Iperf and jperf installations

Install iperf:

```
sudo apt-get install iperf
```

Install jperf

Download the latest file *jperf-2.0.2.zip*

The *jperf.sh* script within the */usr/local/src/jperf-2.0.2/* directory have to set executable permission by running:

```
pwd /usr/local/src/jperf-2.0.0.zip
pwd
unzip jperf-2.0.2.zip
cd jperf-2.0.2/
ls -al jperf.sh
sudo chmod u+x jperf.sh
ls -al jperf.sh
```

To run jperf:

```
./jperf.sh
```

## Install VLC

```
sudo add-apt-repository ppa:videolan/stable-daily
sudo apt-get update
sudo apt-get install vlc
```

## Install Wireshark

To install Wireshark the following commands have been used:

```
sudo add-apt-repository ppa:pi-rho/security
sudo apt-get update
sudo apt-get install wireshark
```

## Install VNC

### Step One - Install Desktop Environment and VNC Server

XFCE4 is installed whose packages can be get, along with the package for TightVNC, directly from Ubuntu's software repositories using apt:

```
sudo apt-get update
sudo apt-get install xfce4 xfce4-goodies tightvncserver
```

Vncserver command to set up a secure password:

```
vncserver
```

Once this command has been entered, the user has to set up an access password. Then, the user will be asked to established a view-only password. If users log with the view-only password, they will not be able to control the VNC instance with their mouse or keyboard.

### Step two: Configure VNC Server

Since the default configuration of VNC servers is going to be changed, first it is necessary to stop

the VNC server instance that is running on port 5901:

```
vncserver -kill :1
```

It is useful to back up the original xstartup file in case it will be used later:

```
mv ~/.vnc/xstartup ~/.vnc/xstartup.bak
```

Now a new xstartup file is opened:

```
nano ~/.vnc/xstartup
```

Insert these commands into the file so that they are performed automatically whenever you start or restart your VNC server:

```
#!/bin/bash
xrdb $HOME/.Xresources
startxfce4 &
```

To ensure that the VNC server will be able to use this new startup file properly, it is recommended to grant executable privileges to it:

```
sudo chmod +x ~/.vnc/xstartup
```

### **Step Three: Create a VNC Service File**

First, open a new service file in /etc/init.d with nano:

```
sudo nano /etc/init.d/vncserver
```

In this file the user declare some common settings that VN will be referring to a lot, like username and display resolution.

```
#!/bin/bash
PATH="$PATH:/usr/bin/"
export USER="user"
DISPLAY="1"
DEPTH="16"
GEOMETRY="1024x768"
OPTIONS="-depth ${DEPTH} -geometry ${GEOMETRY} :${DISPLAY}
-localhost"
. /lib/lsb/init-functions
```

Coming up next, the command instructions that will allow us to manage the new service are explained:

```
case "$1" in
start)
log_action_begin_msg "Starting vncserver for user '${USER}'
on localhost:${DISPLAY}"
su ${USER} -c "/usr/bin/vncserver ${OPTIONS}"
;;
```

The next block creates the command keyword stop. It will kill an existing VNC server instance.

```
stop)
log_action_begin_msg "Stopping vncserver for user '${USER}'
on localhost:${DISPLAY}"
su ${USER} -c "/usr/bin/vncserver -kill :${DISPLAY}"
;;
```

The final block is for the command keyword restart, which is simply the two previous commands

(stop and start) combined into one command.

```
restart)
$0 stop
$0 start
;;
esac
```

Once all of those blocks are in the service script, you can save and close that file. Make this service script executable, so that you can use the commands that you just set up:

```
sudo chmod +x /etc/init.d/vncserver
```

Now try using the service and command to start a new VNC server instance:

```
sudo service vncserver start
```

At this moment VNC server is able to use. To run it just type at /etc/init.d directory the command:

```
vncserver:1
```

#### **Step Four: Connect Your VNC Desktop**

To use VNC server a client that supports VNC connections might be installed. This client is located not in a remote machine but in user's own computer by using the following commands:

```
sudo aptitude install xvnc4viewer
```

Then simply run it by:

```
vncviewer <ip_address_virtual_machine>:5901
```

# References

- [1] <https://www.opennetworking.org>
- [2] <https://www.cs.princeton.edu>
- [3] <https://www.planet-lab.org/>
- [4] <https://tools.ietf.org/html/rfc3746>
- [5] [https://www.usenix.org/legacy/event/nsdi05/tech/full\\_papers/caesar/caesar.pdf](https://www.usenix.org/legacy/event/nsdi05/tech/full_papers/caesar/caesar.pdf)
- [6] <http://homepages.inf.ed.ac.uk/mmarina/papers/sdn-chapter.pdf>
- [7] <http://yuba.stanford.edu/ethane/>
- [8] <https://www.geni.net/>
- [9] <http://cleanslate.stanford.edu/>
- [10] <http://www.extremenetworks.com/>
- [11] <http://openvswitch.org/>
- [12] <http://www.noxrepo.org/>
- [13] <https://openflow.stanford.edu/display/Beacon/Home>
- [14] <http://www.projectfloodlight.org/floodlight/>
- [15] <https://www.opendaylight.org/>
- [16] <http://osrg.github.io/ryu/>
- [17] <http://read.cs.ucla.edu/click/click>
- [18] <https://www.sdxcentral.com/articles/contributed/the-sdn-gold-rush-to-the-northbound-api/2012/11/>
- [19] <http://www.frenetic-lang.org/>
- [20] [https://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/Guias/GUI A Cloud.pdf](https://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/Guias/GUI_A_Cloud.pdf)
- [21] <http://www.nist.gov/>
- [22] <https://www.openstack.org/>

- [23] <http://opennebula.org/>
- [24] <http://blogs.salleurl.edu/the-cpd-zone/2014/04/23/sdn-software-defined-networking/>
- [25] [www.cenits.es](http://www.cenits.es)
- [26] <http://mininet.org>
- [27] <http://daviwa.blogspot.com.es/>
- [28] <http://sourceforge.net/projects/cbench/>
- [29] <http://www.videolan.org/vlc/>
- [30] <https://www.wireshark.org/>
- [31] <http://www.tcpdump.org/>
- [32] <http://www.internet2.edu/presentations/jt2012winter/20120122-Wallace-jointtech-01%20-%20dpctl.pdf>
- [33] <https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-vnc-on-ubuntu-14-04>



# Agradecimientos

En primer lugar, agradecer a mis directores del proyecto, Jose Luis Gonzáles, director general de la Fundación CENITS-COMPUTAEX, y Javier Carmona Murillo por confiar en mi para el desarrollo de este proyecto.

En segundo lugar, agradecer a David Cortés Polo por la ayuda y apoyo mostrados durante el tiempo en el que se ha desarrollado este proyecto.

En tercer lugar, agradecer a mi familia y amigos por su apoyo en estos meses.