UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

NEW PARADIGMS OF LEGACY NETWORK FEATURES OVER SDN ARCHITECTURE

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de

Barcelona

Universidad Politécnica de Cataluña

by

PABLO ANDRÉS BARBECHO BAUTISTA

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisor: Jaume Comellas

Barcelona, June 2017

Title of the thesis: NEW PARADIGMS OF LEGACY NETWORK FEATURES OVER SDN ARCHITECTURE

Author: PABLO BARBECHO

Advisor: JAUME COMELLAS

Abstract:

The Software Define Networking (SDN) paradigm proposes faster implementations, flexibility, and a simplified network management, resulting very attractive to new carrier deployments. Nevertheless, migrating legacy networks to SDN scenarios has been slowed down. Traditional network features, such as high availability, load balancing, and scalability are constrained by the centralized nature of SDN architecture. This study evaluates legacy network features, applied to an SDN network, analyzing the impact of this evolution on the network performance. For the evaluation, a set of virtual scenarios has been implemented, assessing different network parameters, in order to measure the impact on the network performance.

Index Terms: Software Defined Networking, Performance Evaluation, High Availability, Load Balance, Scalability

Contents

List of Figures

List of Tables

Nomenclature

API	Application Programming Interface
BGP	Border Gateway Protocol
CE	Customer Edge Device
CI	Confidence Interval
E2E	End to End
FIB	Forwarding Information Base
FPM	Forwarding Plane Management
GRE	Generic Routing Encapsulation
GUI	Graphic User Interface
IGP	Interior Gateway Protocol
IS-IS	Intermediate System-to-Intermediate System Protocol
LACP	Link Aggregation Control Protocol
LDP	Label Distribution Protocol
L2VPN	Layer 2 Virtual Private Networks
LLDP	Link Layer Discovery Protocol
MLB	Mastership Load Balancer
MSTP	Multiple Spanning Tree Protocol
NBI	Northbound Interface
NFV	Network Function Virtualization
NOS	Network Operating System
NV	Network Virtualization
ODL	OpenDaylight
ONOS	Open Network Operating System
OXM	OpenFlow Extensible Match
PE	Provider Edge Router
PF	Proactive Forwarding
RF	Reactive Forwarding
RSTP	Rapid Spanning Tree Protocol

RTT	Round Trip Time
SDN	Software Defined Networking
SFC	Service Function Chaining
STP	Spanning Tree Protocol
SBI	Southbound Interface
NBI	Northbound Interface
VPLS	Virtual Private LAN Service

Chapter 1

1.1 Introduction

Traditional internet networks, are the result of different protocols that in most of the cases are defined in an isolated way, addressing particular issues, and at the end incrementing technological fixes, a problem sometimes called *Internet Ossification* [1]. As a result, we have a stack of protocols and overlays which make the network management very complex. Indeed, the own physical network structure shows a high complexity as well. In addition, proprietary implementations increase compatibility issues in multi-vendor environments. Different vendors with proprietary protocols and specific configurations result in complex scenarios with clear network limitations.

Furthermore, monitoring and administration of all this network equipment that nowadays, are managed mainly in a distributed manner, makes it even more complex.

Software defined networking SDN [2], has emerged as a new network architecture that increases programmability and centralizes the intelligence of the network, breaking the control plane out of the switch itself and delegating this control to a central equipment increasing the flexibility of the network.

Instead of having the same distributed scheme of legacy networks, the SDN architecture centralizes the management of the network infrastructure in one device, the SDN Controller. The main concepts of SDN such as abstraction, orchestration, and virtualization will be reviewed throughout this work, as well as the architecture components including a brief description of northbound and southbound interfaces and the SDN concept applicability in operational environments.

This study aims to evaluate the impact of different network properties like the high availability, load balancing, and scalability on the network performance. This analysis is carried out by proposing different virtual scenarios and topologies, assessing different network parameters, in order to measure the overall network performance and the interaction between these network properties.

As a first part of the work, basic concepts and the state of the art of Software Define Networking, as well as OpenFlow Protocol [3] will be reviewed to visualize capabilities and constraints that could appear in a migration scenario from legacy networks to this new architecture. A brief description of the available SDN applications from different controller's developments has been also included in order to understand actual features available for the architecture.

Moreover, a brief description of the most implemented controllers [4] OpenDaylight ODL [5] and Open Network Operating System ONOS [6] has been included. A benchmarking between ODL and ONOS controllers has been performed in order to measure controller's performance and choose the optimal one for the proposed scenarios.

A second part of the work describes the simulation environment for testing the applicability of some SDN tools and apps in a real scenario and with the premises of network features migration from legacy to SDN networks. The simulation covers two kinds of implementations, from a carrier perspective, SDN-WAN, with support for IP routing and from an enterprise perspective at Data Center SDN – LAN.

Finally, a migration scenario is analyzed in order to check the feasibility of a migration scenario from a carrier perspective with L3 forwarding.

1.2 SDN architecture

In the SDN architecture, the idea is to decouple the control and data planes, including a uniform vendor-agnostic interface (OpenFlow) between control and data layers, centralizing the network intelligence in a single high performance device. SDN architecture is illustrated in Fig 1.1. Centralized intelligence is supported by a unique equipment known as the controller. It has a logical map of the entire network or infrastructure layer and also the network functions or application layer, providing an efficient monitoring and control of the network.

This new paradigm allows implementing a programmable, flexible and scalable network at a faster pace representing a solution for the limitations of traditional networking. From the bottom to the top, the SDN architecture defines different work planes [7] described below.

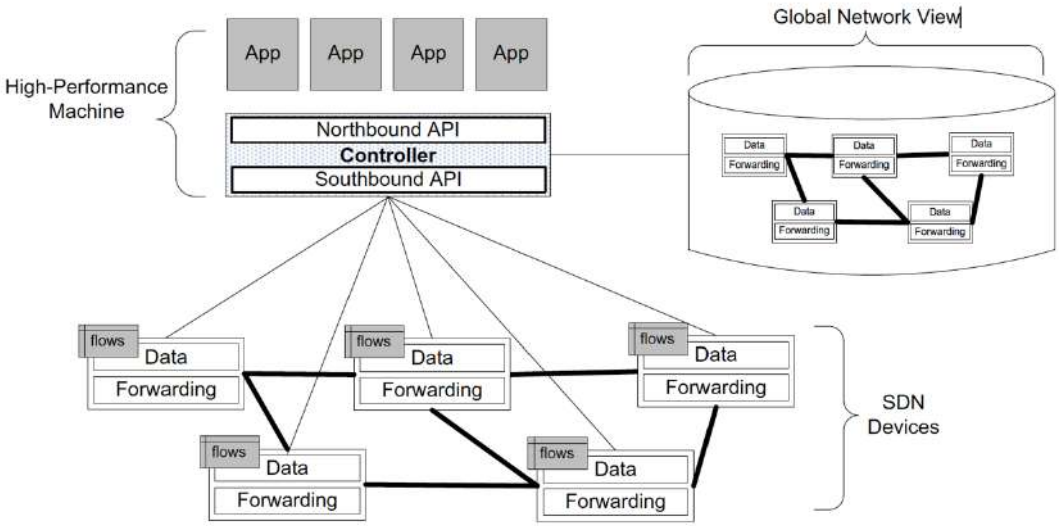


Figure 1.1 SDN Architecture [8]

The ***Forwarding Plane,*** located at the infrastructure layer is responsible for the forwarding, dropping or changing packets. Each forwarding action is based on instructions received from the *Control Plane*. Switches and routers are in general represented as an infrastructure resource (abstraction), reducing the network complexity. The forwarding plane is also known as Data Plane.

The ***Control Plane*** in conjunction with the Management Plane provides intelligence to the network. The control plane is implemented in the SDN Controller. It works as the intermediary between the infrastructure and application layer. This plane is responsible for making decisions on how packets should be forwarded by network devices and translate those decisions to the Data Plane which is in charge of the execution. The Control plane is also in charge of fine-tuning the Forwarding Tables [9] that resides in the Data Plane.

The ***Management Plane***, as well as the Control Plane, is located at the Control Layer and is responsible for monitoring, configuring, and maintaining network devices through the southbound interface SBI.

Finally, the ***Application Plane*** allocates applications and services that define the network behavior. Applications are modular and communicate with the Control Plane through the northbound interface, NBI. The network behavior can be modified using the applications located at this plane.

1.3 SDN components

SDN architecture is comprised of the SDN Controller and SDN devices. In addition, there are different northbound/southbound interfaces for the communications between application and data planes. The most used southbound interface is OpenFlow, and it will be explained in more detail in this Chapter.

1.3.1 SDN Controller

The SDN controller is considered as the core or brain in the network. It is referred as the Network OS, centralizing the control of the network. Located between the Application and the Infrastructure Layer, it has a global view of the network. It is in charge of flows management and communicating the action set to the switches and routers located at Infrastructure Layer. The communication with devices is done via the southbound interfaces and with the applications and business logic is done via the northbound interfaces.

The first protocol developed for communicating the SDN controller with switches as a southbound interface is OpenFlow [3], illustrated Fig 1.2.

Moreover, there is another SBI, the Open vSwitch Database OVSDB [10]. It is a management protocol also used in commercial equipment or as secondary implementation for controllers clustering. Another option to program SDN devices at the infrastructure layer is through the use of APIs. In that sense, the network configuration protocol NETCONF could be used as SBI. Furthermore, REST, Python, and Java are defined as the current APIs [11] for communicating the application layer with the controller, working as NBIs, depicted in Fig 1.2.

Independently of the SDN Controller, typically a set of pluggable modules have been deployed performing different network tasks. Some tasks include inventorying of devices, network statistics, flow tables monitor of switches and topology viewers in real time.

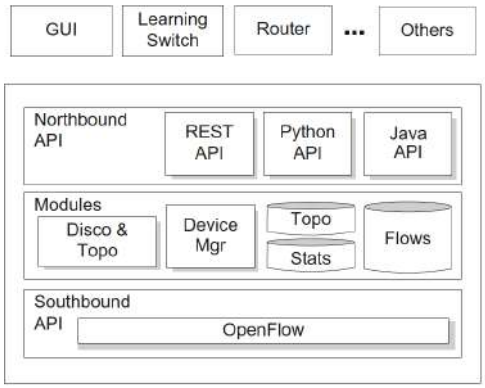


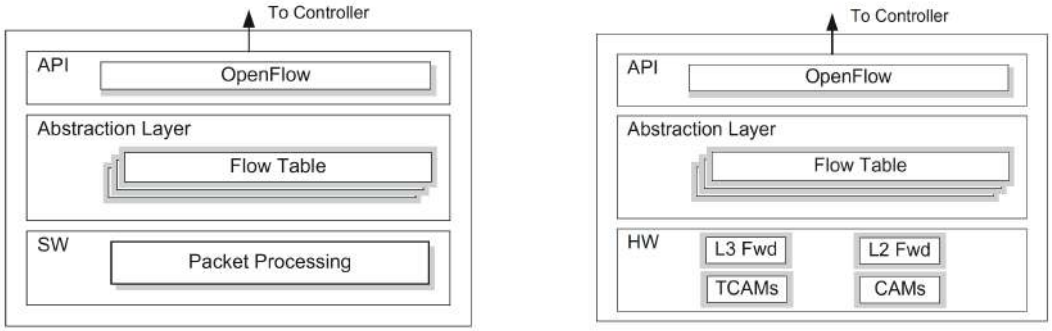
Figure 1.2 SDN Controller Structure [8]

Controller's functionalities can be expanded installing or developing additional modules. Usually, these modules are open source as the controller, but there also are proprietary software with a required license agreement.

ODL and ONOS are examples of open source controllers with several network functionalities at the application plane. Controllers performance will be reviewed more in detail in the next Chapter.

1.3.2 SDN Devices

SDN devices receive packets on a port and perform a certain network function. Any network equipment that could implement an API to connect to the controller, can be considered as an SDN device. Figure 1.3 illustrates an SDN switch composed by an API to communicate with the controller, flow tables, and a packet processing function either implemented in software Fig 1.3 (a) or embodied in hardware Fig 1.3 (b).



(a) (b)

Figure 1.3 (a) SDN hardware switch, (b) SDN software switch [8]

1.4 OpenFlow

OpenFlow was initially proposed by Stanford University [12], and it is now standardized by The Open Networking Foundation ONF [13], a non-profit organization established in 2011 by Deutsche TeleKom, Google, Facebook, Microsoft, Verizon, and Yahoo. ONF is dedicated to accelerating the adoption of SDN.

OpenFlow appears as the first standard for the communication interface between the Control and Data layers and the most used SBI at the industry.

Basically, OpenFlow allows the SDN controller, to manage the network devices located at the infrastructure layer. OpenFlow protocol must be implemented on both sides of the interface, between the devices at the infrastructure layer and at the SDN controller in the control layer. The most important idea of OpenFlow is that it uses the concept of flows to manage the traffic, instead of IP directions. The flows are managed based on the rules allocated in the flow tables.

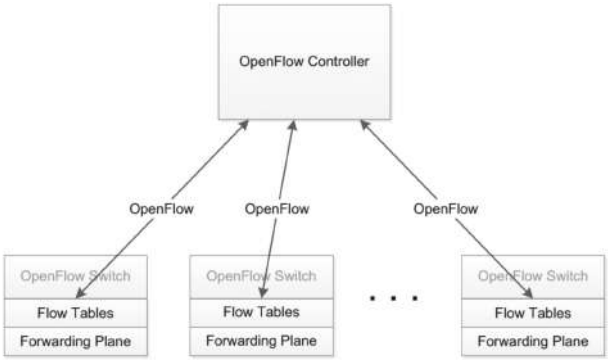


Figure 1.4 OpenFlow - Based Controller [8]

In addition, OpenFlow-based SDN controller allows the integration with enterprise carrier's existing infrastructure and provides a simple migration path for those segments of the network that need SDN functionality [14]. Based on OpenFlow protocol, the controller manages directly the Flow Tables of each switch and then gives the set of actions to the forwarding plane as is shown in Fig. 1.4.

1.4.1 OpenFlow Switch and Components

An OpenFlow Switch is composed of one or more flow tables and an OpenFlow channel. The first element performs packet lookups and forwarding, and the OpenFlow channel provides the communication with the controller.

An overview of the packet forwarding process in OpenFlow is described in Fig 1.5. When a packet arrives at the OpenFlow switch "*Packet in from Network*", the header of the packet is analyzed *"Parsing Header Fields",* and matched against the inner flow table *"Match against tables",* if a match is found in the table, the *action* is performed *"Perform actions on packets"*. Prioritization of each flow entry is also available if more than one match is found in the local table. Multiple types of actions could be performed, forwarding the packet, blocking traffic, port forwarding, and a general QoS treatment for accomplishing a basic traffic engineering.

On the other hand, if no match of the packet header is found *"No match found"* at the flow table, the switch sends a PACKET_IN message to the controller *"Notify controller"*. At this instance, the controller has to determine the corresponding action to handle the packet. It has to update the flow tables along the path to the destination of the packet through a PACKET_OUT message. A further explanation of the OpenFlow process is described in section 2.3.1.

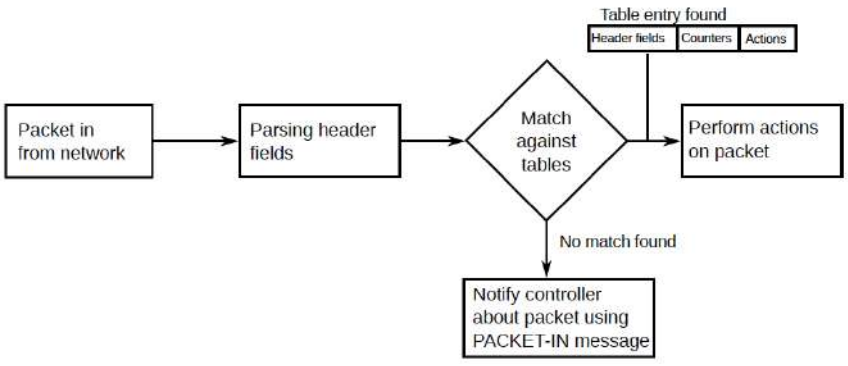


Figure 1.5 Packet Forwarding process - OpenFlow Switch [15]

For an integration with OpenFlow protocol, traditional switches require a firmware or software upgrade. In this case, the switch is considered Hybrid Type and handle the traffic as an Ethernet switch as well as OpenFlow tables to forward packets, at the same time.

Furthermore, if the switch just supports OpenFlow protocol but not legacy traffic, it is considered Only-OpenFlow switch type and the action in the local OpenFlow table is performed. The Table below shows some hybrid-type switches. In most of the cases, just an upgrade of the firmware make them OpenFlow compliant. Some important vendor references as Cisco, HP, and Huawei are included in Table 1.1.

Vendor	Product
Arista	7124FX
Broadcom	Strata XGS
Cisco	cat6k, catalyst 3750, 6500 series
Dell	Z9000 and s4810
HP	5900
Huawei	SN-640
IBM	RackSwitch G8264
Juniper	MX-240, T-640
NEC	IP8800, PF5240, PF5820
Pica8	P-3290,P-3295,P-3780

Table 1.1 OpenFlow Compliant Switches [16]

OpenFlow protocol works through the controller. It is in charge of add, update, or delete flow entries in one or multiple flow tables at switches as depicted in Fig. 1.6. Each flow table in the switch contains a set of flow entries and each flow entry consists of match fields and a set of instructions to apply to matching packets. Flow entries consist of header fields, counters, and actions associated with each entry. The header fields are used as match criteria to check if the incoming packets match this entry.

Multi-flow tables allow a subsequent packet processing, looking for a match in succeeding tables. Each flow entry can be chained to another flow table. The processing pipeline provides a greater flexibility for packet processing, commonly used in QoS environments [15].

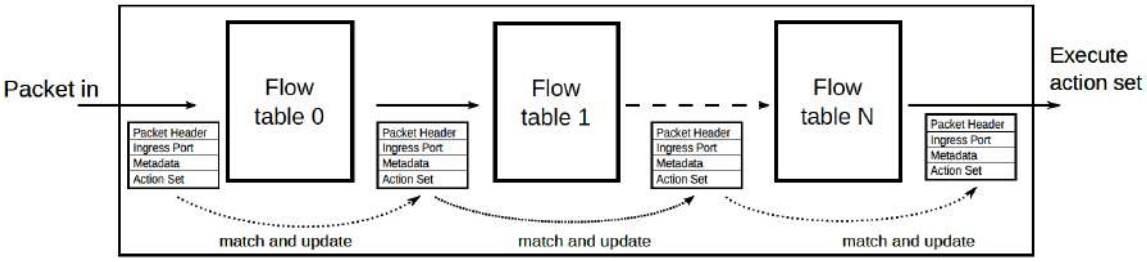


Figure 1.6 Multiple tables in a pipeline [15]

Some modifications and enhancements have been done since the first OpenFlow protocol release 1.0 [17]. Main changes, capabilities, and support of later releases, are described in Table 1.2.

The OpenFlow version V1.0 initially presented support for a single table with a limited number of entries and also a limited packet processing capabilities [18]. Later, V1.2 implemented a multi-table and group abstraction support, allowing group entries and action buckets for multicast applications. Release 1.2 also enhance the matching process through the OpenFlow Extension Match OXM. Version 1.3 [19] added the metering capability to the OpenFlow protocol, which allowed a basic QoS service, based on rate limit policy. Version 1.3 also implemented multiple tunnel encapsulations for tunneling support. Version V1.4 [20] added the bundles, which provide an enhanced transactional capability to the OpenFlow controller. Finally, the main improvement in V1.5 [3] is the port recirculation that allows service Chaining functions for firewalls and load balancers.

Despite different OpenFlow versions, there are general types of messages used by an OpenFlow Controller. The switch to controller connection is discovered using a symmetric protocol and it is maintained using periodic echo request/reply messages. There are also specific unidirectional messages, sent from the controller to the switch, or from the switch to the controller to perform specific tasks. As an example, the FLOW_MOD message is used to modify a certain flow entry in a switch. Asynchronous messages may also pass from the switch to the controller announcing changes in the switch or network state [21].

OpenFlow	
Version	**Features**
1.0	Single Table
1.1	Multi- table, Groups, VLAN and MPLS support
1.2	OXM, Multiple controllers support - Switch maintain simultaneous connections to multiple controllers
1.3	Meters for QoS Capabilities - Rate limiting, Tunneling Support for multiple tunnel encapsulations
1.4	Support for Bundles, Eviction, Vacancy Events
1.5	Port Recirculation allows Service Chaining for Firewalls or Load Balancers

Table 1.2 OpenFlow Version Support

1.5 State of the Art

SDN appears as an interesting concept of centralized management with southbound and northbound interfaces in order to orchestrate the overall solution. OpenFlow protocol becomes the most used southbound interface to connect the control and data planes.

Several SDN networks have been deployed in large data centers, portals and cloud providers, which demonstrate that the technology already crossed the academic - commercial gap. According to Gartner survey, "*By the end of 2016, more than 10,000 enterprises worldwide will have deployed SDN in their networks, a tenfold increase from end-of-year 2014"* [22]. The Figure below illustrates Gartner survey conducted in 2014, presenting an idea of SDN maturity.

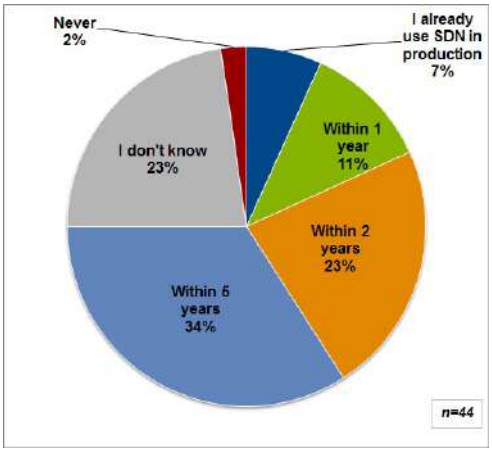


Figure 1.7 Gartner SDN Adoption – 2014 [22]

With the SDN momentum in both industry and research fields, a lot of medium-big size operators and vendors decided to be involved in SDN networking. Particularly, ONF concentrates Deutsche TeleKom, Google, Facebook, Microsoft, among others.

As a reference example, ***Google*** has already implemented lightweight OpenFlow switches, a cluster of OpenFlow controllers and SDN applications for managing its inter-DC WAN backbone. Main goals leading Google's project are savings and efficiency. Google's SDN solution includes [23]:

- Custom data center Edge switches.
- A cluster of OpenFlow controllers at each DC.
- BGP and IS-IS between OpenFlow controllers.
- Centralized traffic engineering application.

Furthermore, ***Microsoft Azure*** also has an SDN deployment based on overlays. Implementing overlay technologies, based on GRE tunnels for providing compatibility with legacy networks. Microsoft implements NVGRE similar to VXLAN [24]. The idea comprises a vSwitch and an agent installed in each hypervisor in order to communicate with the Microsoft's SDN controllers. The virtual network VN agent sends flows onward to the vSwitch, creating millions of virtual networks. Users define their own network policies by the frontend Azure portal (management plane). The portal sends the configurations through the northbound API to the controllers in order to set up the virtual network [25].

Ebay, also proposes an SDN implementation using Overlays, creating public cloud VN using VMware Nicira solution. A similar solution has been proposed by ***Rackspace*** in the reference [26].

Enterprise	SDN Type
Google	Open SDN
Microsoft Azure	Overlays
Ebay	Overlays
Rackspace	Overlays

Table 1.3 SDN Implementations [8]

As has been mentioned in section 1.4, OpenFlow appears as the most used SBI between the control and data plane. This protocol has been broadly adopted by both research and industry implementations. Actually, there are a lot of devices for testing the SDN concept.

Hybrid Type switches, supporting OpenFlow protocol and traditional forwarding, present a route map for enterprise and carrier networks to progressively introduce SDN technologies, even in multi-vendor network environments.

Moreover, the compatibility with legacy switches is necessary for a migration scenario, but a new concept comes with SDN networks, the *"White-box switches"*. Those eliminate vendor lock-in by delivering open hardware, which means an agnostic networking, making it easy for the final user to achieve operational and financial freedom through a disaggregated white box model. Pica8 white boxes is a representative example of this new concept [27].

1.6 Commercial devices

Software Defined Networking has gone beyond the proof of concept stage. Nowadays it is deployed in real environments, with representative use cases, as those described in the previous section. Research contributions, as well as private developments, have been growing exponentially, deriving in commercial equipment.

1.6.1 SDN Controllers

Developments from different vendors are currently available, Cisco with Cisco Open SDN Controller [28] mainly based on ODL is one example. HP also has an early implementation controller, the HP VAN Controller [29], which presents an interesting HP SDN app store [30] that provides a simple way to download and install applications, both paid and free apps from the community. Moreover, commercial implementations are commonly based on previous open source projects, details are provided in section 2.5.

On the other hand, open source deployments, ODL and ONOS network operating systems, currently are the most used according to SDxCentral survey [31]. Next chapter presents an analysis of ODL and ONOS with performance evaluation in order to select one for further simulations.

Last year ONF release an interesting project called Atrium 16A [32]. This integrates a set of open source components which combined form a complete SDN stack with extended functionalities for carrier environments, mainly routing at L3 capabilities.

Atrium presents two options, based on ODL and ONOS. The development based on OpenDaylight, presents as main feature the implementation of a virtual router app with Quagga router suit [33], added in this release. This functionality is implemented for carrier's environments in order to provide a BGP compatibility with legacy scenarios and intra-DC connectivity.

Atrium looks to close the large integration gap of the elements in an SDN network. Three elements have been released and at the moment are in its testing version:

Atrium Router with OpenDaylight: The second release of Atrium router was built on OpenDaylight (ODL), controlling hardware OpenFlow switches and using Quagga as the BGP control plane protocol.

Atrium Router with ONOS: scalability and stability improved of the Atrium Router on ONOS, and added experimental support for an IGP (OSPF or IS-IS), in addition to the existing support of BGP.

Atrium Leaf-Spine Fabric: Presents a vertically integrated solution for a data-center leaf-spine fabric. This is the first time an L2/L3 Clos Fabric has been built in open-source, on white-box hardware [34]. The Atrium Fabric is designed to scale up to 16 racks.

Examples of current commercial and open source SDN-enabled devices are listed in Table 1.8.

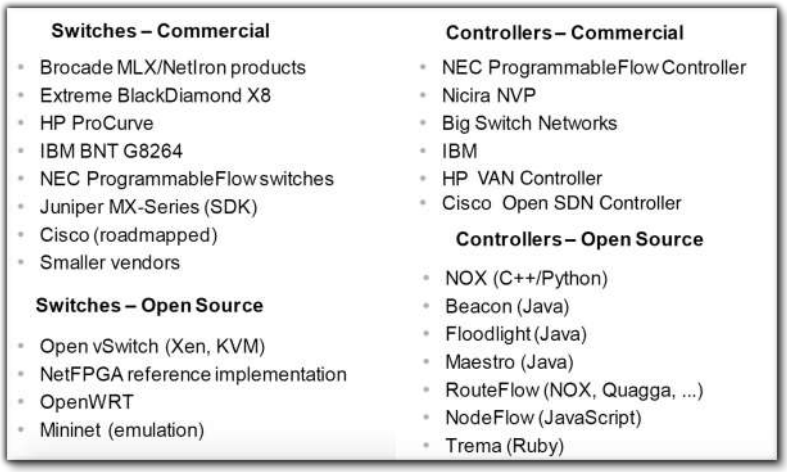


Figure 1.8 Available OpenFlow Enabled Devices [23]

1.6.2 SDN Switches

In addition to controller developments that corresponds to the control layer, in an SDN architecture, there is also the virtual and hardware switches component, covering the data layer of the architecture.

Currently, there are popular open and industry switches developments, in addition to those shown in Fig 1.8, Open vSwitch (OVS) from Nicira [35], VMware virtual switch - NSX [36], Cisco Nexus 1000V [37], and Indigo from Big Switch [38] are also examples of SDN-enabled switch implementations.

Open vSwitch OVS was created by the team at Nicira, that was later acquired by VMware. Some functionalities of OVS are the support of NetFlow, sFlow, port mirroring, VLANs, among others. It differs from the commercial offerings from VMware and Cisco due to the fact that OVS do not have a native SDN Controller, like the Virtual Supervisor Manager-VSM in the Cisco 1000V or vCenter in the case of VMware's distributed switch. The controller could be ONOS, ODL or any third party OpenFlow or OpenVSwitch Data Base Controller.

As is described in section 1.4, OpenFlow appears as the most used southbound interface and therefore network equipment manufacturers, such as Cisco, HP, NEC, IBM, Juniper, and Extreme, have added OpenFlow support to some of their legacy

switches [8]. These switches compose the Hybrid Type of OpenFlow-Based devices and may operate in both legacy mode or OpenFlow mode.

PICA8 is another commercial example of NOS and white box OpenFlow-enabled vendor. The main feature of PICA8 is the software, although they are also white boxes vendor with preload Pica8 OS. Some details of Pica8 are listed in Fig 1.10, among other, the software is designed with L2/L3 capabilities, traditional VLANs, dynamic routing OSPF/BGP, ACL, and OpenFlow 1.4.

The CrossFlow mode of PicaOS lets users run traditional Layer 2 and Layer 3 protocols alongside OpenFlow, all on the same switch at the same time *a true seamless migration to SDN* [39] which corresponds to a hybrid switch type, supporting OpenFlow and legacy protocols.

PICA8 WHITE BOX SDN

		Pica8™ Pre-loaded Switch Guide		
		 P-3297 1 RU low-latency, high performance 1G/10G Ethernet switch for 1G server access and management network deployments	 P-5101 1 RU low-latency, highest performance 10G/40G Ethernet switch for data center top of rack and a fixed form factor aggregation	 P-5401 High-performance, high-throughput 40G Ethernet switch for fixed form factor, data center aggregation
OPENFLOW	Large TCAM	Yes	Yes	Yes
	Open vSwitch	v2.0	v2.0	v2.0
	MPLS over OVS	Yes	Yes	Yes
	GRE tunneling	Yes	Yes	Yes
PORTS	Base Unit	48-Port, 10/100/1000BASE-T	40 x 10 GbE SFP+	32 x 40 GbE QSFP+
	Uplink Options	4 x 1 GbE (SFP) or 4 x 10 GbE (SFP+)	32 x 10 GbE (QSFP+ to SFP+) or 8 x 40 GbE (QSFP+)	NA
	SFP+ / QSFP+ Options	SR, LR, LRM, CR4	SR4, LR4, CR4 / SR, LR, SR4	SR4, LR4, CR4
	Console Port	1 x RJ45 Serial	1 x RJ45 Serial	1 x RJ45 Serial
	Management port	1 x 10/100/1000BASE-T	1 x 10/100/1000BASE-T	1 x 10/100/1000BASE-T

Figure 1.9 PicaOS Preloaded White Box Switches

NoviFlow is also a good example of switches vendor with OpenFlow V1.3, 1.4, 1.5 support in their devices. Noviflow design switches for carrier and Data Center environments. Some features that present ***Noviswitches*** [40] are capacity up to 240 gigabits per second, up to 1 million wildcard match flow entries, up to 15 million exact match flow entries.

IBM with the switch G8264 is also going for OpenFlow–enable switches. It has 48 x 10 GbE SFP+ ports and 4 x 40 GbE QSFP+ ports. The switch runs in either of two modes, traditional L2/L3 mode or OpenFlow mode but not both at the same time [41].

In OpenFlow mode, the switch supports up to 97,750 flows configured by a remote controller but also Static flow entries can be configured up to 750, but the remote controller cannot modify these entries. The G8264 supports two flow tables (OpenFlow 1.0 or 1.3) per switch instance, basic flow table, and emergency flow table.

Figure 1.10 IBM OpenFlow switch G8264 [42]

Chapter 2

2.1 Legacy Network Properties

In order to create a frame of reference, common features used in a medium-size enterprise is analyzed based on a previous work implemented in a real scenario [43]. This study was done years ago to a private enterprise. The network infrastructure is composed of several data centers distributed along the country in a mesh topology. On the other hand, intra-DC connectivity illustrated in Fig. 2.1 presents the data center topology corresponding to a hierarchical structure, with well-defined access, distribution and core switches. The network corresponds to a multi-vendor implementation and the topology describes a distributed scenario. In order to manage the network, configurations at each device have to be performed.

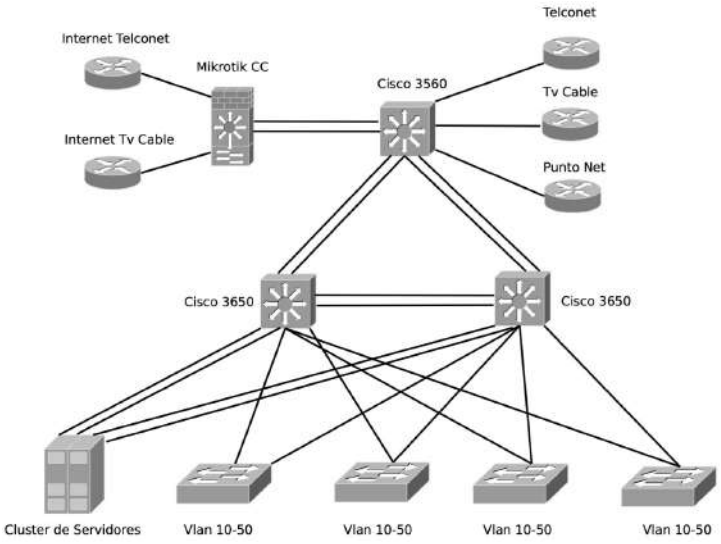


Figure 2.1 Legacy Intra - Data Center Topology

A common Ethernet fat tree data center is depicted in Fig. 2.1. Local provider equipment (TVCable and Telconet) are connected to the Core device (Cisco 3560) in order to establish a private connectivity between data centers through private links. From the use case shown in Fig. 2.1 several common services and features required in legacy networks are also basic requirements for new DC scenarios.

SDN networking as a solution for new DCs must be able to support features as, high availability, scalability and load balancing, offering high performance, reduced complexity and allowing faster deployments.

In addition, new paradigms appear concerning to the SDN architecture, like the single point of failure in a centralized environment. Some features listed in Table 2.1 have been analyzed through simulations, in order to test SDN network performance.

Legacy Networks	Features / Services
Port Channel - LACP	Link Aggregation
Link Redundancy - RSTP	High Availability
Virtual Local Area Networks - VLANs	Segmentation
Hierarchical Model - Aggregation Layer	Scalability
DiffServ Architecture	QoS
Generic Routing Encapsulation - GRE	Tunneling
OSPF	Dynamic Routing
Route Map	Source Routing
STP - LAN/ OSPF (BW metric) - WAN	Load Balancing

Table 2.1 Legacy Network Features and Services

On the other hand, inter-DC connectivity also requires similar network features, high availability, load balancing, scalability and advanced routing support. On Fig. 2.2 a basic WAN topology took from the use case [29] is implemented in order to explore the impact of above mentioned properties for inter-DC communication. The figure shows three DC locations, each one connected to the backbone of the network. Services as dynamic routing, traffic engineering, tunneling, are also used in the scenario.

Solutions explored in the use case [29] make use of Overlay networks composed by GRE tunnels. As IGP, OSPF has been implemented with the overall network visibility at each device through the OSPF update messages going through the tunnels. The HA is provided by the redundancy at the backbone, based on bandwidth metrics assigned to each link.

The presented scheme at the use case, does not represent the optimal use of resources due to the fact that the redundant link is not used all the time. Although, using a static routing in addition to the dynamic protocol, the redundant link could be used, configuring the routing table manually based on predefined policies. In small environments as the presented in the use case [29], using dynamic and static routing protocols results in a better solution for optimizing the use of resources. Nevertheless, the overall solution is not scalable due to the complexity of the network.

Despite the networks configuration proposed in the mentioned study [29], fulfill the requirements inside the data center comparable to a TIER III specification, the overall solution result very expensive (CAPEX/OPEX). In addition, there is a provider dependency resulting in a fixed network and difficult to scale.

The rapid expansion of new services also requires an open and more flexible network, characteristics that are not available in fixed legacy networks. New models like Infrastructure as a Services (IaaS) are in the focus of enterprises. The old model of enterprises purchasing dedicated network equipment from NEMs is just attractive for vendors, but not for clients that require a flexible network and fast deployments.

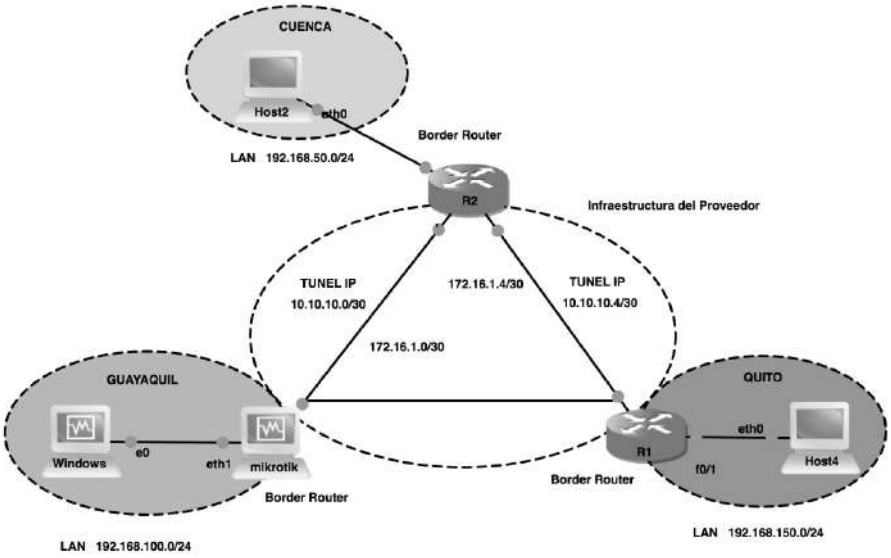


Figure 2.2 Inter-DC Topology

This chapter analyzes the feasibility to implement legacy network features over an SDN network. Some legacy features and services described in Table 2.1 have been implemented in a simulation environment to test the overall performance of SDN networks.

2.2 Data Center Properties

Nowadays, new Data Centers features as virtualization present important challenges to traditional networks. A new network architecture is required to cover this new challenge as fast deployments and reduced complexity for a better scalability.

As we know, virtual machines are created, moved and deleted in a fast way, a process that may take just a few minutes, instead of days that would take deploy a traditional network.

With new networking paradigms as Network Virtualization **NV**, which means virtualization of network hardware, Network Functions Virtualization **NFV** corresponding to the virtualization of network functions as address translation, load balancers or even firewalls, and at the end the Software Defined Network **SDN** which controls the entire environment, a fully virtualized infrastructure will be achieved to provide NaaS environments.

From a general perspective, network features as resiliency, high availability, load balancing, scalability are described below with constraints and issues from legacy networks and how SDN solutions cope with these challenges.

2.2.1 High Availability

Some HA solutions are available in legacy networks for data center, the use of STP protocol is one of them. It has been implemented in legacy networks to provide a certain grade of redundancy to the environment. Nowadays with the fast DC implementations (virtualization) and administration, the requirements of the supporting network below include a fast deployment, high throughput, and programmability. Some drawbacks of the STP solution for HA as the recovery latency, the number of packet losses and the

overall resiliency of the network represents a non-viable solution for new data centers scenarios.

It is important to take into account that designing an HA network is more than just adding redundancy. In the SDN architecture, the centralized logic control provides a view of the whole network allowing it to make decisions predictably and optimally based on the known topology. Furthermore, the controller as the brain of the system controls the individual forwarding tables of each device, which means a direct control over routing and forwarding decisions [8].

An important aspect of HA networking property is the path selection. In this sense, SDN provides added features compared to the traditional mechanism as the shortest path used by IGPs as OSPF or IS-IS. From a general point of view, traditional path selection uses the shortest path based on the number of hops. In contrast, the SDN optimal path selection is done for each flow and takes into account more parameters than the traditional shortest path, for example, the link occupancy for making the decision of an optimal path Fig 2.3.

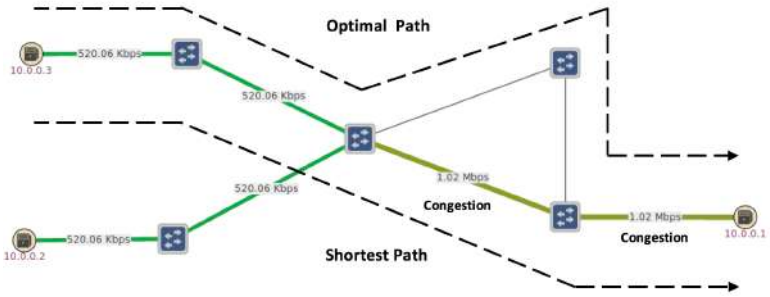


Figure 2.3 SDN Optimum path selection

An optimal path selection is illustrated in Fig. 2.3, at the bottom, takes into account the congestion of the links. The data traffic at each link is an important parameter for an optimal path selection, and also contributes to the load balancing across the network. In contrast, legacy shortest path selection just takes into account the dimension of the path (number of hops), at the top of Fig. 2.3.

In order to provide an HA to the network, the topology shown in Fig. 2.4 is used as a simple example.

The topology provides a framework to test aspects as speed, efficiency, and reliability in SDN networks, as a solution for new DC requirements. The topology also allows the implementation of other network properties inside the data center as load balancing feature, reviewed in the next section.

As is shown in the Fig. 2.3 a full mesh structure is included in the topology, with multiple channels to provide alternate paths for communication in case of failures.

Besides the ability of the system to adapt to a failure in an efficient way when the failure has been solved, provides the required resiliency for new Data Centers. The test is focused in analyze if the SDN architecture is an applicable system that optimizes the resources and makes the network resistant to any kind of failure, based on a configuration that takes the shortest path algorithm for providing a HA environment.

For the simulation in the next chapter, a random availability for each link is supposed, setting the cost of the links to 1. An optimal implementation should take into account

that each link has a certain probability of being available and must be configured based on this parameter.

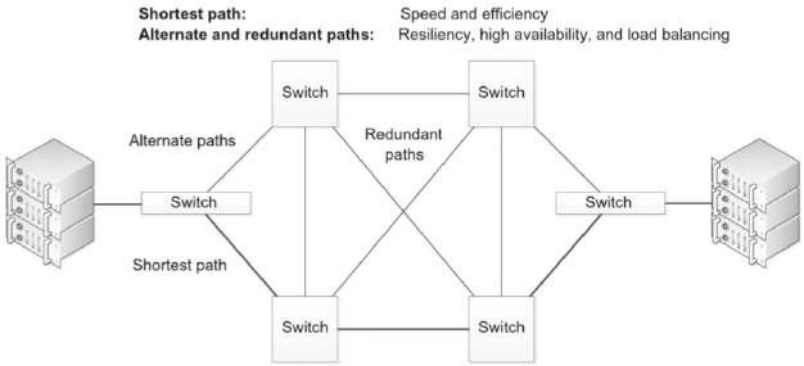


Figure 2.4 Proposed Topology – Simple Environment [8]

Comparing the proposed topology Fig. 2.3 with a typical one shown in Fig 2.1, it can be found in the typical one a hierarchical topology with a distributed control that makes the network recovery slow and lack of load balancing. The proposed topology on the other side, handled by an SDN controller reacts in an efficient way and provides a faster link recovery time as has been shown in the next chapter.

Besides the data plane HA, there is the control plane where a centralized SDN architecture represents a big problem. As the brain of the network is the controller, it is in charge of the overall orchestration of the elements. In that sense, the controller plays a critical role and must guarantee the HA of the overall system. Independent of the controller (OD, ONOS, RYU, etc) every SDN solution requires some resiliency level and HA at the control plane.

In section 2.2.3, some solutions to cover the HA at data plane has been described, such as clustering, teaming or hierarchical models that provide a constant controller availability. Due to this feature is interrelated with the scalability, the analysis of control plane HA has been done later in this work.

2.2.2 Load Balancing

In legacy networks, to implement an L2 LB the STP protocol could also be used. The relation cost-benefit of STP result proper for implement LB in traditional data centers. With new data center requirements, this solution results useless.

As well as in HA, the LB of an SDN network is based on the optimal path selection. As the traffic data in each link is known by the SDN controller as well as the overall network topology, the path selection for each individual flow is done taking into account not only the number of hops but also the traffic occupation, which means an optimal LB implementation between different flows intents. Some implementations of OpenFlow-based load balancers [44] demonstrate effective load-balancing systems taking into account the network congestion and in advance server congestion providing a customized flow routing for data center optimizations. Furthermore, at control layer load balancing also represent a challenge for SDN architecture, but current implementations [45] shows multi instance environments as a possible solution.

By this moment, controllers do not present a mature internal load balancer app, instead, some works are testing applications for manage external load balancers [46]. Another mechanism to implement a load balancer is based on SFC [47], but also refers to experimental phase. Nevertheless, those solutions are proposed by the development community and are not included in the official platform.
As an example, from the HA point of view, with the reactive forwarding active on ONOS or ODL controllers, by default the link costs are set to 1. The metric that the shortest path algorithm uses to install the end to end path in the switch flow tables, uses the number of hops, so the link cost in each link is set to 1 and the Load Balancing could not be implemented having just the number of hops as a metric. Furthermore, the link bandwidth and the occupation of the link should be included as metrics for a LB implementation.

An implementation of LB based on extended Dijkstra's algorithm which not only uses the number of hops, but also the link occupation is demonstrated in the reference but is not yet included as a production feature [48].

Actually, to implement Load Balancing in real DC SDN network is not a direct app but an additional subsystem, with ONOS controller, a stable solution is the use of policy–based directives called "Intents".

The Intent Framework is a subsystem that allows the user or programed application to specify a deterministic network behavior in form of policy (policy-based). This policy-based directive allows a high level of granularity for control the packet flows. This configuration means a static setting up, that at the end, results in a problem for a good scalability. Nevertheless, the idea is to be used for a further development applicable to any environment.

On the other hand, there are specific requirements for LB at the control layer. This constraint is directly related to the scalability of the system, which constitutes a great issue in SDN architecture. Some approaches have been done as distributed designs where the workload is shared between the different instances that compose the cluster. More details of distributed approach are reviewed in the next section and in Chapter 3.

2.2.3 Scalability

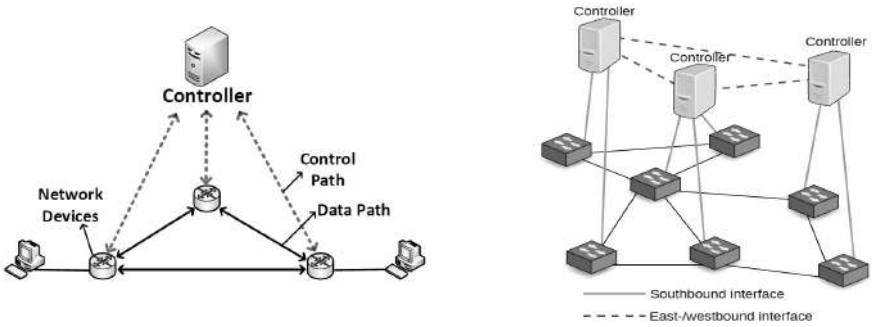
SDN architecture appears as a solution for the traditional network limitations. In SDN the control plane is focused on a centralized solution, where a single control entity has a global view of the network. However, the paradigm of a centralized control logic results in scalability issues. Abstracting the lower levels functionalities (data plane) allows the management of a network through the control plane (controller). This means, that the SDN controllers must have enough processing power to deal with the high amount of flows (RF more than PF) and also be aware of the number of devices that they can handle, fixed capacity.

Some mechanisms are developed in order to solve the SDN architecture scalability problems. The controller role change mechanism is an example. This one enables an OpenFlow (from V1.2) switch to maintain a connection, at the same time, with multiple controllers in parallel. Each controller then, has to be designed with a specific role master, equal, or slave [2].

The controller role change mechanism corresponds to a Distributed Flat Control Design Fig 2.5 (b) with consistency and synchronization constraints. In order to cope with this issue, the controller communicates with the rest of controllers (synchronization

messages) through a controller to controller channel to exchange needed state information. In distributed designs, each controller has a global view of the whole network.

In addition to the Flat Distributed control design, a controller hierarchical design (not well defined in OpenFlow specifications) and hybrid design could be implemented from a topology point of view.



(a) (b)

Figure 2.5 (a) Centralized - Single Control Design [2], *(b) Distributed - Flat Control Design* [15]

As it was mentioned above the Distributed design comes with some issues that the different controller implementations like ODL or ONOS, the two-open source more used controllers, have to deal with. For example, the reception of the updates between the controllers is not guaranteed. In addition, if a controller loss the connectivity to the cluster and just before receiving a topology event, this update will not be broadcasted to the cluster. In this case, the topology will not be correctly updated.

An anti-entropy mechanism implemented in some controllers helps with the non-updated controllers. This mechanism works at a fixed interval, usually 3-5 seconds, a controller randomly picks another controller and they both synchronize their respective topology views [49].

On the other hand, limitation in legacy networks like MAC tables size and VLANs number, in SDN architecture are more suited for a scalable provisioning. Initially, OpenFlow v1.0 has been designed to implement the flows entries in just one table. Latest OpenFlow versions, introduce a group table mechanism which enables multiple flow tables. In addition, some solutions using tunnel encapsulations are also available in new OF versions.

Both solutions, distributed controller architecture, and multiple flow tables are proposed to mitigate the scalability issues of SDN networks. Moreover, the distributed controller architecture provides a fault tolerant environment due to dismissing the only point of failure at the data plane with just one controller.

In addition, a Load Balancing between the controllers is achieved with this type of design.

2.2.4 Virtual Private LAN Service

In addition to the network features reviewed above, a relevant service inside current and new DCs infrastructures correspond to VPN services. In that sense, Virtual Private LAN Service (VPLS) a type of layer 2 virtual private network L2VPN, emulates LAN

networks across WAN environments. Some current uses of VPLS are for inter-DC connections in the same layer 3 networks. Furthermore, it is used for point to point and multipoint connectivity.
As VPLS correspond to a LAN services, the same scalability challenges arise. Nevertheless, VPLS could be implemented at the core of the network with scalability constraints, when the number of branches is high, or it could be implemented with an MPLS core helping with scalability issues.
Diagram in Fig.2.6 illustrates a VPLS reference model where the provider edge router (PE) gives a logical interconnection, such that the customer edge devices (CE) belongs to a specific VPLS and appear to be on a single Ethernet bridge. Each VPLS located at the CEs can contain a single VLAN or multiple tagged VLANs [50].

VPLS is a relevant service to test in an SDN environment, analyzing the overall support and compatibility with OpenFlow devices.

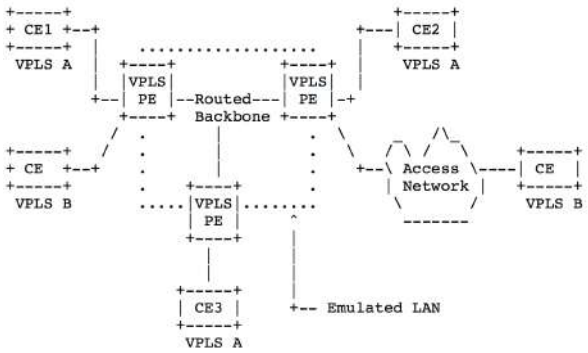


Figure 2.6 VPLS Reference Model [50]

2.3 Forwarding Mechanisms

In SDN networks, depending on the controller (OpenFlow-based), some operational modes can be configured. Basically, OpenFlow defines two forwarding mechanisms, Reactive and Proactive. Furthermore, a hybrid operational mode can also be implemented.

2.3.1 Reactive Forwarding

RF mechanism is used to install flow entries into the network switches, defining flow rules in L2 switches in anticipation of traffic and constructing end-to-end paths.

In this operational mode, when a new packet arrives at a switch, it makes a lookup in its flow tables. If the match is not found, it sends to the controller to delegate the decision to handle the packet. After the controller processing the packet, it defines a flow entry that is sent to the switches.

The entries are installed on-demand after a sender starts transmitting the first packets. These flow entries have a defined expiration time for unused paths, passed this time the flow entries are removed from the switches. As an example, ONOS uses the Reactive Forwarding App (org.onosproject.fwd) for instantiating the reactive forwarding operational mode.

Most of the SDN Controllers comes by default with Reactive forwarding enabled. For example, ONOS controller performs an LLDP to learn the links in the topology and computes the path between the hosts using the RF mechanism. With

reactive forwarding app enabled, the shortest path computation is done based on the discovered topology, in which Dijkstra algorithms are used for shortest path computation.

The reactive behavior allows a more efficient use of memory (MAC table) at the cost of a reestablishing path later, which means latency constraints. In contrast, the proactive instantiation requires less time for forwarding due to it has an already installed flow entry.

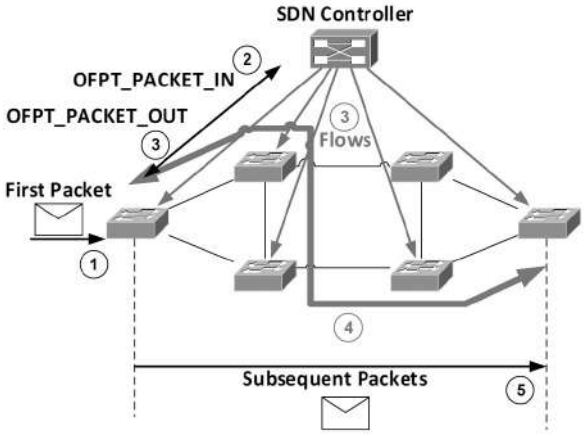


Figure 2.7 Reactive Forwarding Mechanism [51]

The process of the packet in RF begins at the OpenFlow enabled switch, which receives the first packet and attaches it to an OpenFlow message OFPT_PACKET_IN and sends it to the SDN controller in order to check the route. The controller computes the shortest path to the destination (MAC address header) based on the number of hops, link bandwidth, and delay. Once the controller computes the end to end path, it sends the forwarding actions to the involved switches into an OFPT_PACKET_OUT.

Subsequent packets will use the end to end already established path, without consulting the SDN controller. After a timeout, in reactive forwarding, unused entries are excluded from the flow table [51], while in proactive forwarding the flow entry remains until a physical change in the path is produced.

2.3.2 Proactive Forwarding

Refers to installing the flow entries proactively, which means before a sender starts the transmission. Unlikely Reactive forwarding, Proactive instantiation maintains the flow entries in the switches (flow tables) and maintains indefinitely until there is a physical change in the path. For example, ONOS implements the *Intent Forwarding app*, a kind of proactive forwarding. ODL also provides a proactive forwarding framework called *"proactive-flood-mode"*.

2.4 Available SDN controllers review

Nowadays, there is a great collection not only of OpenFlow-based devices but also controllers and applications. From a practical point of view, a lot of software controller solutions are available for research and for a commercial usage. Nevertheless,

representative use cases are quite scarce. It is clear that SDN is real and no longer just academic, as far as the great number of available commercial devices, although the implementations are not fully-SDN networks but partially or using overlays.

In this section, mainly controllers will be analyzed in order to establish a possible open source solution for a migration scenario. The preeminent open source controller solution ODL and the emerging open source ONOS controller will be compared in order to simulate migration of legacy network properties to an SDN environment.

2.4.1 Open source SDN Controllers

The idea is to analyze the supported protocols and capabilities of different open source SDN controllers, implementations that are currently available for development and operation. Although there are a great number of commercial controller developments, some of them are just adaptations of open source developments mainly from the ODL Controller. Commercial solutions as XNC Cisco and HPE SDN controller are in fact based on ODL. In addition, Floodlight has been also used as starting point for commercial developments like Big Network Controller from Big Switch Networks [24].

Some of the more important open source controller projects are shown in Table 2.2. The target user of all of them includes operators, development, and research. An important comment from Table 2.2, is related to Beacon as an influent controller. Implementations as Floodlight and ODL have been influenced by the Beacon project [8].

Name	Source	Lenguage	Target User
Opendaylight	Opendaylight	Java	Operators, Development, Research
ONOS	ON.Lab	Java	Operators, Development, Research
NOX	ICSI	C++	Operators, Research
POX	ICSI	Python	Research
Beacon	Stanford University	Java	Research
Floodlight	Big Switch Networks	Java	Operators, Development, Research
Ryu	NTT Communications	Python	Operators, Development, Research

Table 2.2 Current Open Source Controllers [8]

SDxCentral [52], an organization sponsored by companies leading the SDx and NV market as Arista, Brocade, Juniper, Linux Foundations, Netcracker, VMware, Nuage Networks, among others, deliver news, research, and analysis on software defined infrastructure markets. SDxCentral also provides free reports focuses on hardware and software resources related with SDN adoption and use cases. From the report "The Future of Network Virtualization and SDN Controllers", released in 2016, the most deployed SDN controller solutions according to the SDN Central Survey, allocate ODL as the more deployed controller in commercial scenarios.

Companies like Orange, China Mobile, AT&T, T-Mobile, Comcast, KT Corporation, Telefonica, TeliaSonera, China Telecom, Deutsche Telekom, and Globe Telecom have deployments based on ODL [31]. Furthermore, from the Fig 2.8, SDN

Central Survey shows that ONOS is clearly not widely adopted as ODL, but it has been gaining momentum around service providers in a short period of time.

ONOS is allocated in the third position of most deployed open source SDN controllers and in second with the version of OPNFV [53] based on ONOS. Characteristics such as fast roll out services, reduction of complexity, network visibility, scalability (tested in the next Chapter), all oriented to carrier's market, makes ONOS an interesting solution for evaluating. Current commercial controller developments also include Huawei in the ONOS project [54] with real implementation scenarios.

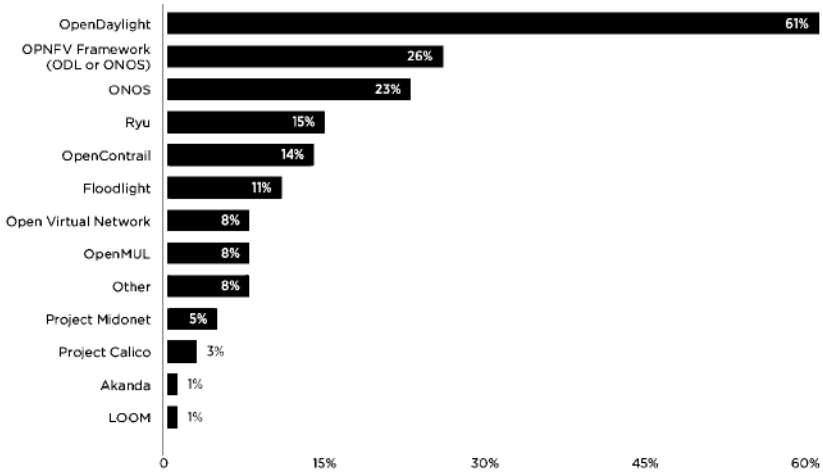


Figure 2.8 NV/SDN deployed solutions [31]

2.4.2 ONOS and ODL

ONOS is an open source SDN NOS intended to cover needs of service providers and help in the transition to a real SDN network. This NOS present functions for scalability, high availability, and abstraction that makes easy to develop new applications and network services due to its modular structure. Another interesting approach of this controller is a well-established community with several proposals, support for the active modules and vendor collaboration.

As is shown in Fig 2.8, ONOS project is the main competitor of ODL. Although ONOS and ODL are supported by some common vendors and service providers such as Cisco, Ericsson, Huawei, Intel, NEC, AT&T, Alcatel Lucent and also the Linux Foundation support both.

ONOS			
Intended For	Southbound Protocols	Programming Language	Customers
Intra - Datacenter, WAN, Transport Network, Telecom Infrastructures	OpenFlow, OVSDB, BGP, NETCONF	Java	Huawei, Internet2, GEANT[55], FIU

Table 2.3 ONOS Controller description [31]

ODL is the NOS leading the open source platform for programmable, software-defined networks as has been shown in Fig 2.8. ODL is a modular and multiprotocol controller for SDN deployments on multi-vendor networks, mainly directed by its sponsors. Currently is sponsored by Cisco, Brocade, Ericsson, Citrix, Intel, HP, Dell, Huawei,

Red Hat, among others.

To date, ODL developers have formed 92 projects to address ways to extend network functionality according to the ODL spectrometer [56] most of them oriented to support legacy network features.

The last version of ODL at the time of this writing is Boron, released in April 2017, which also includes DLUX, a web interface for manage the network. The solution is demanded for cloud service providers, government, education, business and general enterprise.

ODL			
Intended For	Southbound Protocols	Programming Language	Customers
Intra - Datacenter, Inter-Data, WAN	OpenFlow, OVSDB, BGP, NETconf	Java	University of Luxembourg, Cornell University, Packet Design, Brocade Vyatta

Table 2.4 ODL Controller Description [31]

2.5 Controllers Performance - ONOS vs ODL

A simple environment with 1 Virtual OpenFlow switch and 1000 MACs has been tested in order to check the overall throughput of ONOS and ODL controllers. The environment components have been described in Appendix A.

The results are presented in Fig 2.9 with a 95% confidence that the mean is maintained for ONOS within 201736 flows/s - 226793 flows/s and for ODL within 79838 flows/s - 109887 flows/s.

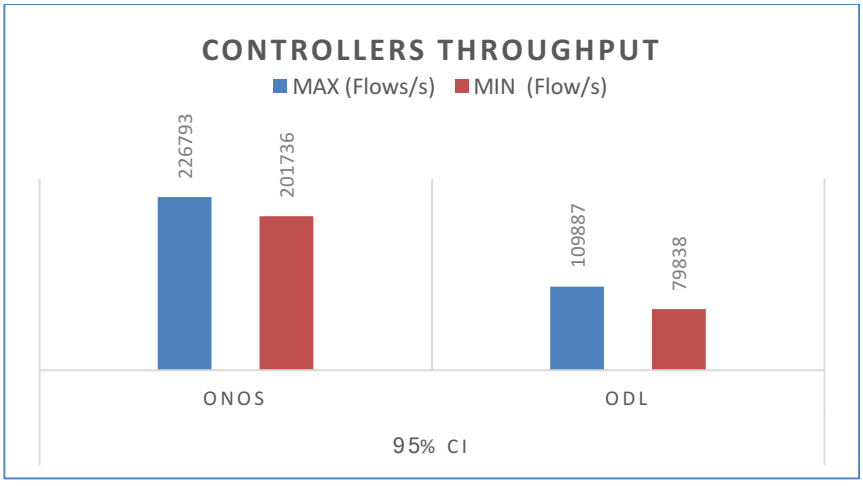


Figure 2.9 ONOS / ODL Throughput Evaluation – 95% CI

It is clear that ONOS outperforms ODL with a considerable difference 119402 flows/s based on the average values for the test with only 1 switch. The results show that ONOS double the capacity of ODL controller under the same virtual scenario. In

addition, based on the standard deviation, ONOS presents a better stability handling the flows than ODL, Appendix A.

A further comparison between ONOS and ODL could be done analyzing the amount of code and number of contributors in order to have an idea of the current activity for each project. In that sense, BlackDuck - Open Hub [57] present statistics of open source projects such as ONOS and ODL.

The statistics presented in Tables 2.5 and 2.6 corresponds to the preceding months of this work (March 2016 – February 2017). Despite ODL has much more contributors and lines of code, ONOS is in the SDN industry just for a few years.

	All Time	12 Month	30 Day
Commits:	9651	2896	179
Contributors:	293	197	47
Files Modified:	24794	11080	903
Lines Added:	4391239	731627	30420
Lines Removed	3582026	472015	21460

Table 2.5 ONOS Contribution Statistics [58]

	All Time	12 Month	30 Day
Commits:	71377	23084	2068
Contributors:	1017	511	134
Files Modified:	79788	30213	3741
Lines Added:	14647772	2653515	106778
Lines Removed	7775240	1151512	48898

Table 2.6 ODL Contribution Statistics [59]

Based on the *Throughput Evaluation (Fig 2.9)*, the continuous growing of the *Project Activity* (*Table 2.5)* and applications available, **ONOS** has been selected as the controller to test different environments in order to migrate legacy features.

In addition, SDN architecture describes a new perspective for traditional network features, High Availability is present not only at Data Plane but also at the Control Plane, Load Balancing with a distributed controllers cluster and also Scalability of the controllers.

The next chapter analyses these constraints, based on simulation scenarios, with a perspective of migration feasibility and coexistence between legacy and SDN networks.

Chapter 3

3.1 Simulation of traditional network features over an SDN architecture

In this chapter, several simulations based on simple topologies are proposed. Different testing scenarios check the feasibility of common network features migration from legacy to an SDN environment. Features and services described in section 2.2 are analyzed from an enterprise point of view going through high availability, load balance, and scalability features. A brief study of network segmentation service is also proposed, analyzing the overall performance and available solutions for SDN networks.

ONOS controller shows a better performance, section 2.5, than ODL in the proposed virtual scenario, by this reason ONOS has been selected for all the simulations. In addition, different software tools have been deployed with the ONOS controller in different VMs. The well know *Mininet* [60], which allows to create a realistic virtual network. Furthermore, the software tool *cbench* [61] a benchmarking tool for test controller's performance has been included.

In advance, the GSN3[62] network tool has also been used to simulate legacy network environments and compare results with SDN simulations.

3.1.1 High Availability

In order to study HA in an SDN environment, first of all, a simple simulation of legacy HA environment is analyzed in order to have a clear view of current problems and limitations and compare those with the proposed SDN HA solution.

The overall analysis has been done for a simple DC scenario [29] were a hierarchical structure corresponding to a typical implementation. Redundant links provided at traditional topologies gives a kind of redundancy to the network, but HA is not just the addition of redundant links, the HA solution must provide a certain resiliency while optimizing resources.

Parameters as recovery time, packet losses and network event response time has been compared with the SDN simulation described below in this section.

In order to compare the recovery time needed to re-establish the connectivity in case of failures, the proposed legacy network topology has been shown in Fig 3.1. implements spanning tree protocol STP for high availability, and the proposed network topology with SDN architecture shown in Fig. 2.4, uses the reactive forwarding method to automatically set up high availability with the shortest path configuration.

For the first case, legacy environment, the network simulator GNS3 [62] has been used. The environment includes:

- 3 Legacy Switches (S1, S2, S3)
- 2 Generic Switches (SW1, SW2)
- 2 VirtualBox VM with Ubuntu (H1, H2)

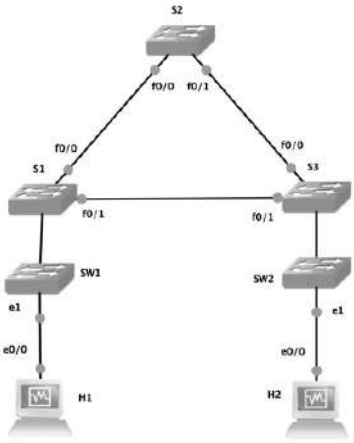


Figure 3.1 GNS3 STP Simulation Topology

The simulation scenario Fig. 3.1 includes the default VLAN 1, trunk ports between the switches S1, S2, S3 and access ports to the hosts H1 and H2.

The spanning tree is configured to have the switch S1 as *ROOT switch* and S2 and S3 as *BRIDGE Switches*. In that case, the port f0/0 is blocked (BLK) by STP protocol and the path from H1 to H2 corresponds to the shortest path *H1-SW1-S1-S3-SW2-H2*.

The STP process Fig 3.2, after a link failure (Shutdown *ROOT port 0/1* – Switch S3) shows the blocking port process, the new *ROOT port* establishment port 0/0 as *ROOT*, the topology changes notice in charge of STP and finally, the learning and forwarding process at new *ROOT port 0/0* in order to re-establish the connectivity.

```
S3(config-if)#
*Mar  1 02:14:35.263: STP: VLAN1 Fa0/1 -> blocking
*Mar  1 02:14:35.263: STP: VLAN1 new root port Fa0/0, cost 38
*Mar  1 02:14:35.339: STP: VLAN1 Fa0/0 -> listening
S3(config-if)#
*Mar  1 02:14:35.843: %DTP-5-NONTRUNKPORTON: Port Fa0/1 has become non-trunk
*Mar  1 02:14:37.135: %LINK-5-CHANGED: Interface FastEthernet0/1, changed state to administratively down
S3(config-if)#
*Mar  1 02:14:37.263: STP: VLAN1 sent Topology Change Notice on Fa0/0
S3(config-if)#
*Mar  1 02:14:38.135: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1, changed state to down
S3(config-if)#
*Mar  1 02:14:50.359: STP: VLAN1 Fa0/0 -> learning
S3(config-if)#
*Mar  1 02:15:05.379: STP: VLAN1 sent Topology Change Notice on Fa0/0
*Mar  1 02:15:05.383: STP: VLAN1 Fa0/0 -> forwarding
S3(config-if)#
```

Figure 3.2 STP Process

The link reestablishment process takes 30 seconds' delay that occurs during the transition from blocking to forwarding mode to prevents a temporal loop condition in the network when switches are connected with redundancy. Furthermore, the overall time required to re-establish the connectivity is measured at the end points H1 and H2. It has been shown a total recovery time of 34 seconds with 33 packets losses Table 3.1.

```
pablo@pablo-VirtualBox: ~
pablo@pablo-VirtualBox:~$ ping 10.0.0.3 | while read pong; do echo "$(date): $pong"; done
sáb mar 25 02:07:39 CET 2017: PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
sáb mar 25 02:07:39 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.750 ms
sáb mar 25 02:07:40 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.729 ms
sáb mar 25 02:07:41 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.987 ms
sáb mar 25 02:08:15 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=36 ttl=64 time=1.16 ms
sáb mar 25 02:08:16 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=37 ttl=64 time=0.767 ms
sáb mar 25 02:08:17 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=38 ttl=64 time=0.849 ms
sáb mar 25 02:08:18 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=39 ttl=64 time=0.769 ms
sáb mar 25 02:08:19 CET 2017: 64 bytes from 10.0.0.3: icmp_seq=40 ttl=64 time=1.04 ms
```

Figure 3.3 ICMP Recovery Time H1 to H2

At the end, for providing an operation of an autonomous device participating in a distributed decision-making process in order to provide high availability in legacy networks, results in a correct operation of transparent bridging at the expense of high converge latency and a possible arbitrary configuration that do not guarantee an efficient use of resources.

Network Event Response Time	Convergence Time	Packet Losses
34ms	34ms	33 packets

Table 3.1 HA Legacy Network Test Results

On the other hand, the previous chapter describes ONOS as the controller to implement in the simulation, as well as Mininet in order to have a virtual network environment. The last one acts as the data plane with abstracted networking and ONOS SDN controller plays as the control plane of the architecture. Above this two layers, the application layer describes services as topology discovery, Reactive Forwarding (org.onosproject.fwd), ACL and in the particular case of ONOS the "Intent based app" that allows to installing "intents" that helps with traffic engineering.

For SDN simulation a simple topology at Fig. 3.4 is used. It offers high availability (alternate paths). As well as in traditional networks with STP, the shortest path route type is configured activating the Reactive Forwarding app in ONOS.

Using MiniEdit [63] the topology is implemented and configured in the simulator. Furthermore, Fig 3.5 shows the test topology implemented in a virtual scenario and managed by the ONOS controller.

The topology is composed of six OpenFlow switches and a unique remote controller:

- 6 OpenFlow Switches (S1, S2, S3, S4, S5, S6)
- 2 Hosts (H1 - 16:B8:E1:C6:A89C, H2 – 2A:A4:28:7F:A3:01)
- 1 Remote Controller (ONOS)

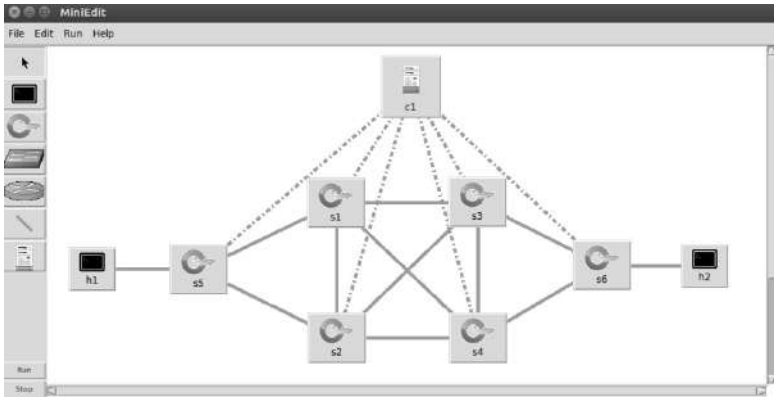


Figure 3.4 HA Simulated Topology

All links are configured at 1Gbps and the controller has defined a custom cell [64] with basic functions enabled as drivers, onos-app-proxyarp and onos-app-fwd applications. The last one corresponds to the activation of the reactive path establishment mechanism.

As well as in traditional network simulation, two hosts, H1 and H2, are included to test the recovery time of the SDN network using ICMP packets from H1 to H2.

Despite the Reactive Forwarding app is configured by default in the custom cell at ONOS fist configuration, flow tables at switches do not have any flow entry for hosts h1 and h2, this is due to the reactive forwarding methods. The RF mechanism needs a fist packet flow through the network, in order to the controller manage the incoming packet, computes the shortest path ant then install the flow entries in each switch of the corresponding path.

Using the ONOS tool **flow statistics** there can be shown the end to end path that the traffic uses at real time through the topology viewer at GUI.

In a first test, the traffic goes from H1 to H2 with the end to end shortest path H1-S5-S1-S4-S6-H2, Fig 3.5 green line.

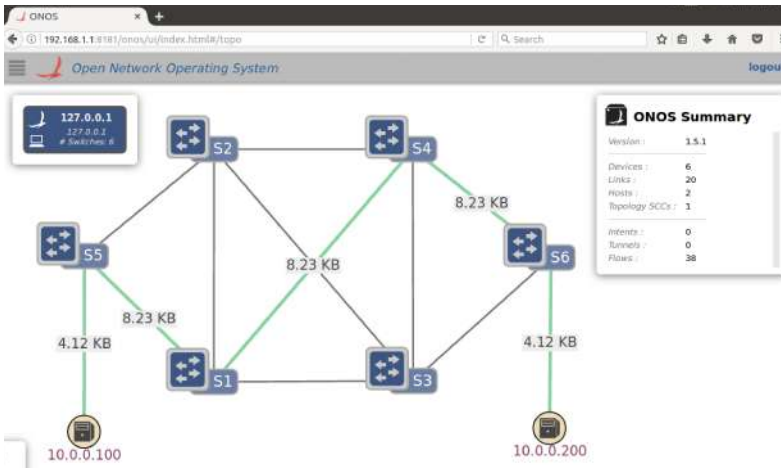


Figure 3.5 Network Topology GUI ONOS

The installed flows are available while the traffic still flowing by the network, once the traffic stops, the timeout erases unused flows at switches, Reactive Forwarding-property. This behavior means a correct use of flow tables, that guaranties high performance (number of flows) handles by OpenFlow switches, but at the expenses of more delays and processor consumption at each flow installation.

For illustrate the flow tables, corresponding flows entries at switch S5 are described. Two flows entries are shown in Fig. 3.6. The first one with ID 0X440000d83860c4, belonging to Table ID = 0, describes the traffic rule for packets from **H2** (MAC 2A:A4:28:7F:A3:01) to **H1** (MAC 16:B8:E1:C6:A8:9C)

- **IN_PORT:1** → Corresponding to the link S1-S5, shortest path selection
- **ETH_DST:** 16:B8:E1:C6:A89C
- **ETH_SRC:** 2A:A4:28:7F:A3:01
- **Treatment → OUTPUT:3 →** Corresponding to H1

The second flow entry with ID 0X440000d8386102, belongs to Table ID = 0, describes the traffic rule for packets from **H1** (MAC 16:B8:E1:C6:A8:9C) to **H2** (MAC 2A:A4:28:7F:A3:01). The match traffic criteria to fall in this flow entry:

- **IN_PORT:3** → Corresponding to the link H1-S5
- **ETH_DST:** 2A:A4:28:7F:A3:01
- **ETH_SRC:** 16:B8:E1:C6:A8:9C

- **Treatment → OUTPUT:1 →** Corresponding to lnik S5-S1, the shortest path selection

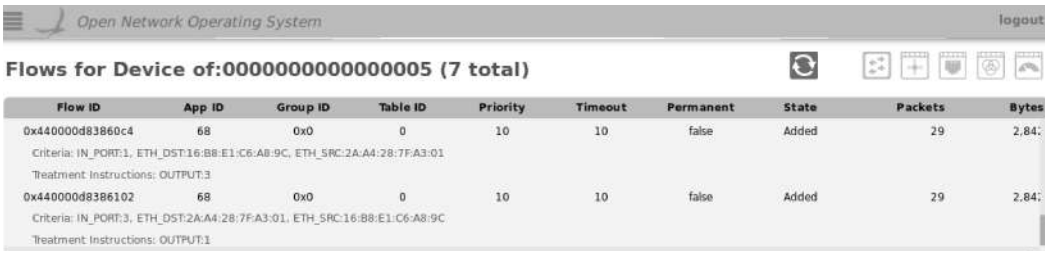


Figure 3.6 Flow Table - Switch S5

The high availability evaluation is done simulating links failures, shutting down the links in Mininet and measuring the time that takes the network to recover the end to end connectivity from H1 to H2 with a simple ping traffic generation.

To test the network event response (link failures), different links (S5-S1, S2-S4, S3-S6) are shut down. The resulting topology is shown in Fig. 3.7. As the flows entries are installed when traffic is active, a continuous ping is generated and the network performance and response to events are evaluated.

The flows at each switch are installed according to the new topology with the first packet through the network. The previous entries are removed from the switches at the time out. As all the links have configured the same bandwidth, the resulting shortest path is H1-S5-S2-S3-S4-S6-H2, Fig. 3.7.

It is important to comment that all the configurations required in traditional networks for implement STP or RSTP, in SDN networks and particularly with ONOS controller, with the reactive forwarding app active, all configurations are automatically performed.

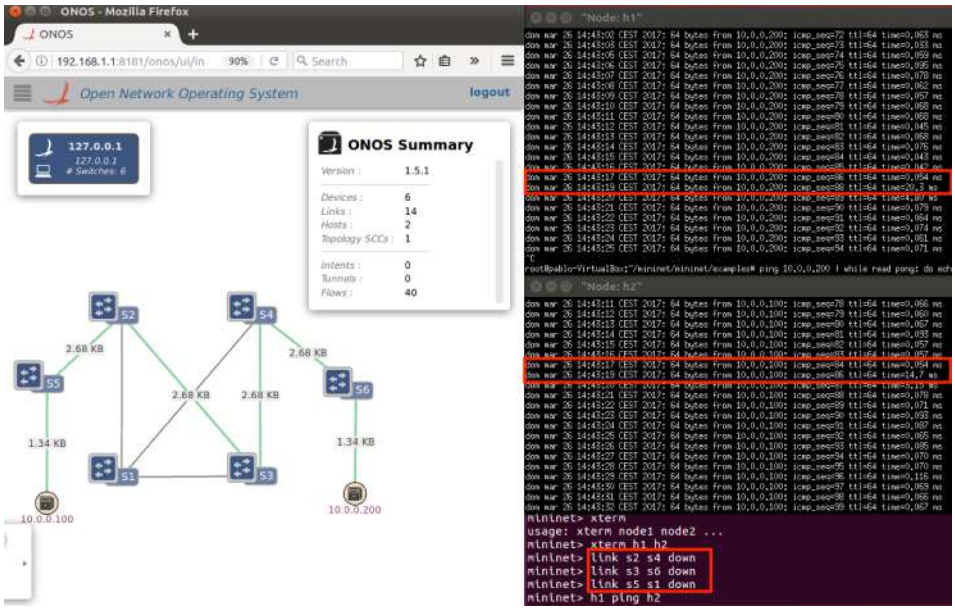


Figure 3.7 SDN ONOS Controller Event Response

In this simple test environment, some performance degradation is shown affecting the latency of some packets with one packet lost. This result is acceptable for new Data Centers scenarios, instead of non-acceptable latency times shown in legacy networks with STP (IEEE 802.1D specifies 15 seconds for listening and 15 seconds for learning). Figure 3.8 shows that SDN architecture responds faster in the case of failure or events in the network. It shows that High Availability is possible to implement with SDN and it presents a much better performance with less recovery and convergence time than legacy networks.

The efficient use of resources is also guaranteed with SDN and the shortest path selection. IEEE 802.1D-2004 amendments corresponding to rapid spanning tree protocol (RSTP) improves the convergence latency time in traditional networks, but some devices do not support it.

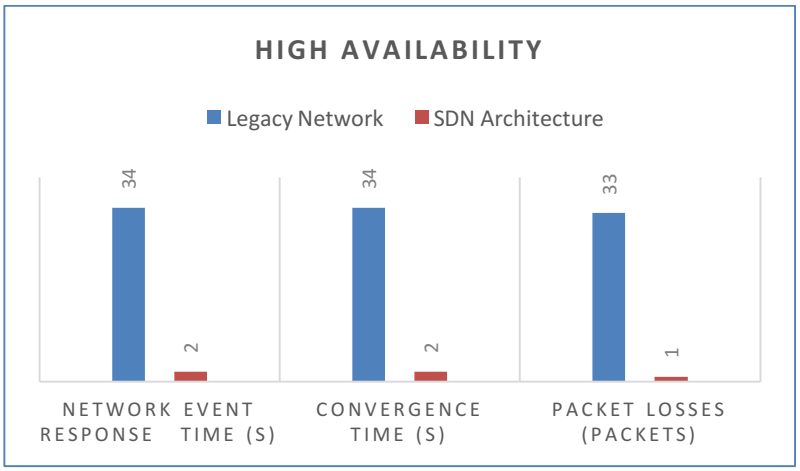


Figure 3.8 High Availability SDN - Legacy Networks

It has been shown that the recovery time, packet losses and the overall network event response in SDN is much better than the achieved by legacy networks. Furthermore, if RSTP is used instead of STP in traditional networks, an optimal shortest path selection is not guaranteed due to metrics, which means a non-efficient use of resources.

3.1.2 Load Balancing

Using the same hierarchical topology depicted on Fig 3.1, the legacy network makes use of VLANs, Trunk ports, and Port Priority STP for implement L2 LB.

The configuration is based on implement two or more root switches. Each root switch is in charge of a certain number of VLANs. The blocked links derivative of the root switch distribution, which is previously established according to the desired behavior. This scenario constitutes a policy – based static configuration.

In a similar way, an SDN network handled by an ONOS controller allows the LB implementation using a policy – based configuration. The controller drives an Intents framework subsystem that allows a granular configuration. In this sense, the LB is statically configured. As has been shown in Fig 3.6 the Flow table of switch S5 is auto updated thanks to the reactive forwarding mechanism. The flow entries for H1 to

H2 and H2 to H1, bidirectional traffic, have been automatically installed along all the end to end path. Those entries result in an efficient implementation against failures, but if the traffic in a

link increases, a load balancing mechanism must be implemented for maintaining the overall network efficiency. An additional host H3 is added to the topology Fig 3.9 for test the LB environment.

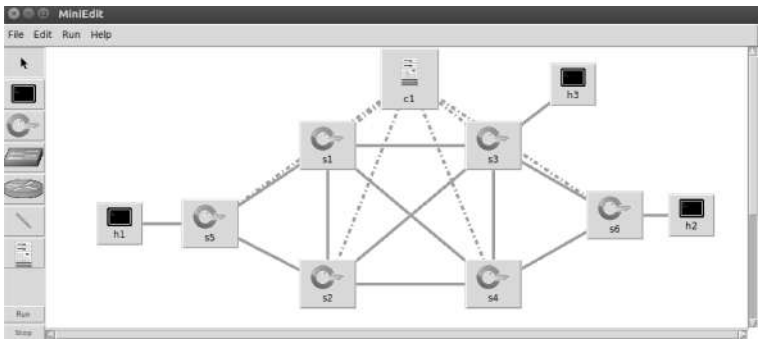


Figure 3.9 LB Simulated Topology

The Intent forwarding method in ONOS controller is in fact response to a Proactive Forwarding mechanism, where the controller pre-establishes the end-to-end path to a certain match of incoming traffic.

Intent comes from the word intention as "state your intentions", which is the analogy applied, in which an application state their intentions of conditioning the network through the use of policy-based directives.

For configure the Intents in ONOS, the type of intent has to be defined. Two types of intents are available, 'Host Intent' for connectivity between hosts, and 'Point Intent' for fine grained control over network resources for routing control. The LB implementation uses the second one.

For the test, Fig 3.10 shows a 5 minutes evaluation between H2- H1 and H3 – H1. Each ping with a defined charge of 65000 bytes has a pre-established end to end path, previously configured with the Proactive Forwarding method based on ONOS Intents subsystem, Appendix B.

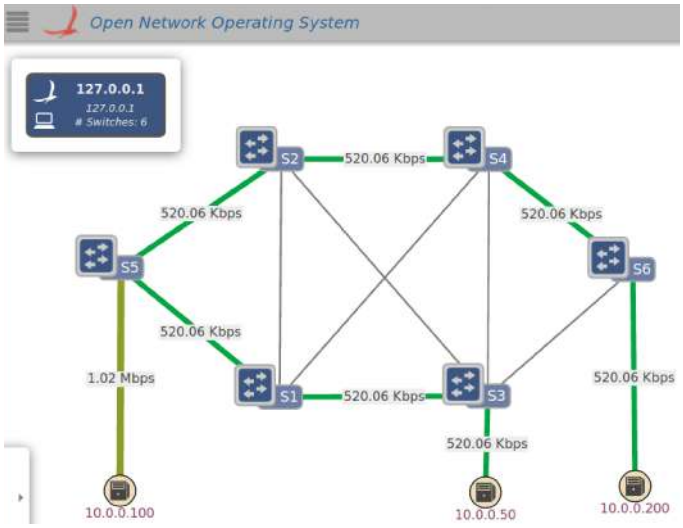


Figure 3.10 LB Proactive Forwarding Paths

Table 3.2 shows the minimum, average and maximum time of response of the Proactive forwarding implementation. As is shown in Fig 3.11, the LB feature is enabled. The E2E connections H1-H2 goes through the path *H1-S5-S2-S4-S6-H2* according to the policies installed in each switch. Besides, the resulting path for E2E connection for H1-H3 goes through the path *H1-S5-S1-S3-H3*. In dark green is shown the shared link at H1-S5 and in light green are shown the different paths used for balance the charge.

Proactive Forwarding		
RTT	**H2 - H1**	**H3 - H1**
Minimum	1,452ms	1,377ms
Average	4,691ms	8,317ms
Maximum	36,098ms	46,04ms

Table 3.2 Proactive Forwarding

On the other hand, if the LB is not implemented, the latency and the overall efficiency of the SDN network could be comprised when the links are not correctly sized according to a traffic analysis. A simulation of the same topology used for PF is used with RF and its auto flow entries installation process. The minimum, average and maximum time of response latency times is analyzed for each connection.

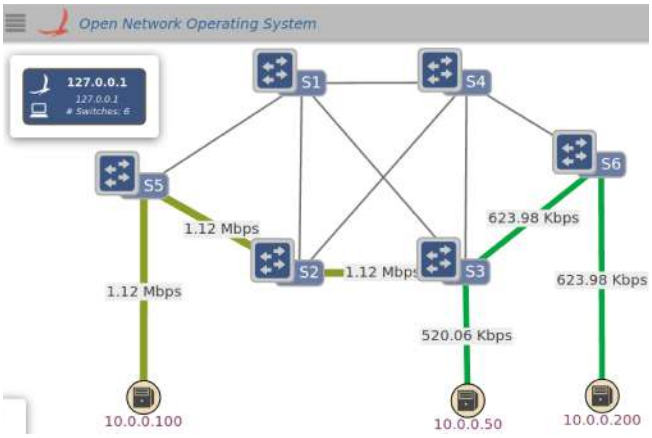


Figure 3.11 Reactive Forwarding Implementation without LB

Figure 3.11 shows the result of an SDN network without L2 LB implemented with RF. The path for both connections H1-H2 and H1-H3 are shared in the section S5-S2-S3 in dark green color. It is shown that the data rate in this shared section corresponds to both connections, 1.12Mbp. In light green, the rest of the path for each connection shows its particular data rate 520 Kbps for H3 (10.0.0.50) and 623Kbps for H2 (10.0.200). The resulting latency times are described below in Table 3.3.

With the ONOS controller and the Reactive Forwarding application enabled, the average RTT for H2-H1 is 9,128ms and H3-H1 7,439ms with a ping packet of size 65000 bytes.

Reactive Forwarding		
RTT	**H2 - H1**	**H3 - H1**
Minimum	1,543ms	1,379ms
Average	9,128ms	7,439ms
Maximum	90,098ms	113,131ms

Table 3.3 Reactive Forwarding

Finally, Table 3.4 shows a comparison between both forwarding methods, PF with LB intents based and RF without LB auto flow entries installation. Both mechanisms are tested with the same topology, simulation environment and observation time.

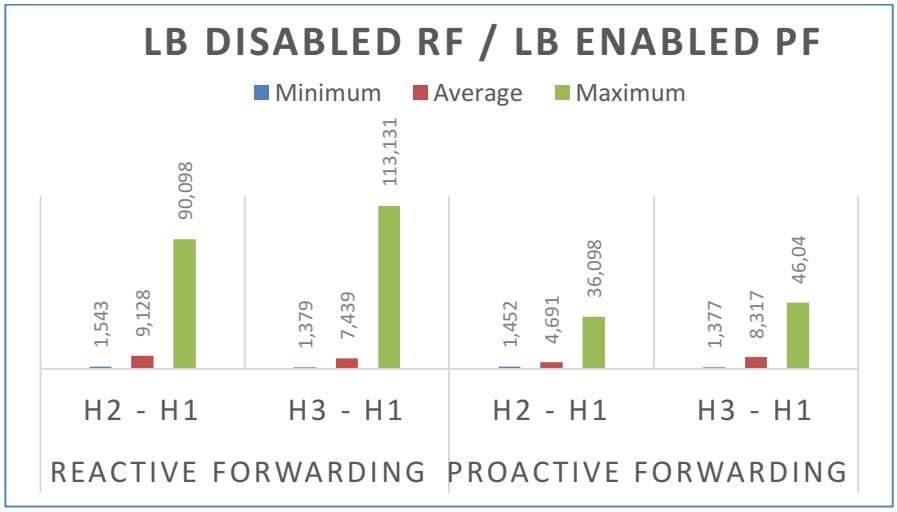


Figure 3.12 PF LB Enabled and RF LB Disabled

The Figure 3.12 shows latency times during a ping test with 65000 bytes of size. For a trace from H3 to H1, RF without LB presents an average RTT of 9,128ms while the PF with LB has an average RTT of 4,691ms. The path that both methods follow is the same with an equal number of hops (H3-S3-S2-S5-H1 for RF and H3-S3-S1-S5-H1 for PF) but in the RF, case the path S5-S2-S3 correspond to shared segments. This behavior of RF does not guarantee an optimal shortest path computation, it means do not take into account the link occupation as a metric to compute the path, as was indented theoretically in SDN networks.

It is clear that due to the LB the performance is better for PF, almost two times faster than RF, which means a great difference. Although, the average RTT presented in PF 8,317ms for the connection H2-H1 is highest than RF 7,439ms. The difference, in this case, is not that high that in the previous one, which means that the performance could be assumed as equal for this second path.

In addition, the maximum RTT of both connections H2-H1 and H3-H1 are highest with the reactive forwarding method since the process of the Reactive Forwarding includes more actions than the Proactive Forwarding. It has been described in section 2.3.1 that the RF process includes, with the first packet, extra actions like the OFPT_PACKET_IN and OUT in order to install the flow entries in the switches.

An open issue in the PF method is the scalability due to the number of intents that must be configured (bidirectional) in order to implement HA. It is true that using the GUI web console the intents are easy implemented, but there is no specification the maximum number of intents that the controller support.

At the end, both network properties HA and LB could be implemented with RF or PF but with the performance constraints shown in this section.

3.1.3 Scalability

In order to analyze the scalability of SDN networks, the multi instance functionality of ONOS is implemented in the simulation environment. This functionality Fig 3.13 refers to a cluster-based implementation. The figure shows some server instances with a distributed core and an intrinsic controller to controller channel.

The cluster is represented by a group of servers that coordinates with each other to provide scalability to the system, sharing the workload between the elements of the cluster, load balancing at control plane.

Furthermore, the multi-instance functionality corresponds to a Distributed Flat Control Design described in section 2.2.3. With a distributed design the overall system can process higher amounts of load (devices) than a Centralized - Single Control Design, and also scale the processing capacity for a high number of requests. In addition to the scalability, the distributed design also provides fault tolerance and resilience to the system at the control layer (controller), in addition to the HA provided by the Reactive Forwarding and optimal shortest path computation at data layer.

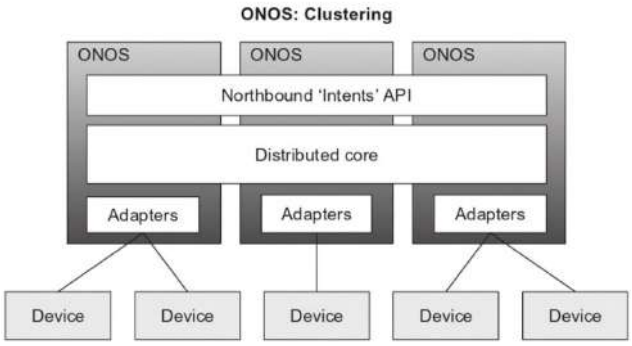


Figure 3.13 Multi Instance ONOS Architecture [8]

At the data plane, OpenFlow V1.2 specifications [49] define three roles that a device can take with respect to the controller or node in the cluster. ONOS adapt its multi-instance functionality through a *mastership subsystem* to the OpenFlow specifications [65]:

- *Equal*: The node may or may not have knowledge of the device, and cannot interact with it.
- *Slave*: The node has knowledge of the device, and can read the state of, but not manage the device.
- *Master*: the node has knowledge of the device, and has full control of the device.

For the scalability implementation, a simple topology is proposed in Fig. 3.14. The environment is composed of 11 switches, 2 ONOS controllers (VMs) and some apps to manage the environment RF app, Master Election app (ME) and Mastership Load Balancer app (MLB). The RF has been activated at ONOS for the automatic path establishment, with delay constraints reviewed in the previous section. The MLB app, which runs in background, is intended to monitoring the switches distribution along the servers that compose the cluster. In addition, the ME app is in charge of assigns the master role to each switch, in the case of an event.

Figure 3.14 illustrates the simulation environment. An automatic distribution of the workload among the controllers has been performed. This configuration, as was mentioned at section 2.2.2 corresponds to a proposal to provide LB and Scalability to the system.

The *Controller 1* (192.168.1.1) is in charge of *5 OpenFlow switches* and the *Controller 2* (192.168.1.2) manage *6 switches*. The balance of charge (control plane) is done by the MLB application dynamically. It represents a suitable solution for scalability which is one of the most important problems related to SDN architecture and also provides a solution for the single point of failure proper to the centralized SDN architecture.

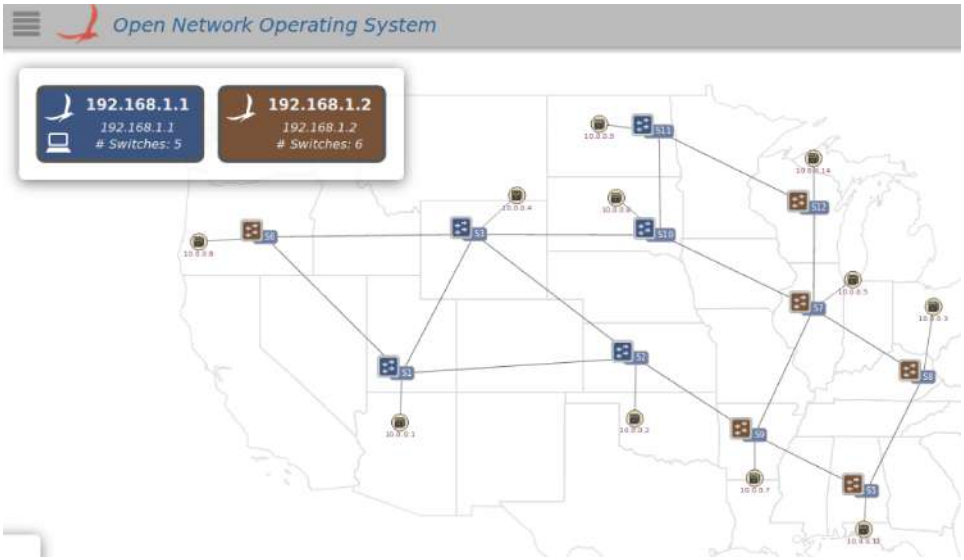


Figure 3.14 Scalability Topology - Multi Instance Controller

With FWD, ME, MLB apps enabled, the milti-instance implementation allows to handle a great number of switches and host, which means a possible solution for SDN Scalability.

The devices are balanced and distributed along the Cluster. The simulation has been done with different VM using VirtualBox, Appendix C.

As new elements have been added in the architecture for supporting a Distributed Control (mastership subsystem, synchronization channel) the overall performance of the system could be affected.

Besides the capability of the design to support a scalable environment, the result is a tradeoff between scalability and performance. Using the *cbench* [61] software, a benchmarking tool for controllers, the throughput of the controllers is measured based on the total number of flow requests sent by switches to the controller. As a point of

reference, the throughput of a Single instance (1 controller) is evaluated for later compared the performance with the multi-instances solution.

There have been tested 1, 4, 8, 10, 16, 32 and 64 switches. The different throughput results shown in Fig 3.15, are based on an average of ten probes for each switch number and 100 MACs per switch.

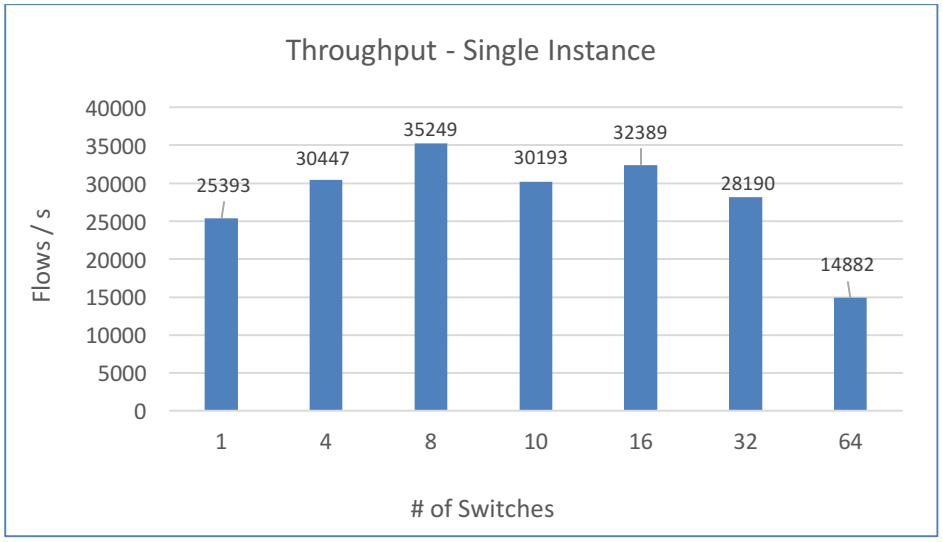


Figure 3.15 ONOS - Throughput - Single Instance

Each simulation has been included within 95% of confidence that the mean value is within the CI. There have been tested 2 ONOS controllers implemented as a cluster (multi-instance) with 1, 4, 8, 10, 16, 32 and 64 switches with 100 of MACs per switch.

The multi-instance results are shown in Fig 3.16 and are compared with the previous results obtained with a single controller scenario.

It is clear that the throughput presented by a single instance is better than the multi-instance environment on every test. There can be estimated a performance degradation range of 1% to 21% depending on the number of switches tested.

Taking into account that the test has been done over VMs with poor performance, the overall difference in a real scenario could result higher, depending on the number of instances installed in the cluster. In this test, just 64 switches have been tested with only 100 MACs per switch. Nevertheless, in a real scenario of a carrier grade network, the number of switches and devices are much greater.

It is also important to take into account that the synchronization process increases the CPU consumption and result in a considerable degradation of the system performance. With the virtualization of the controller, this could result in a minor problem due to the virtualization resources disaggregation concept in new data centers [66].

The performance degradation is proportional to the number of devices composing the cluster. In that sense, result interesting test this solution in a real environment with real traffic to verify if the scalability solution is feasible for carrier environments with a great number of open switch devices.

At the end, the multi-instance scalability solution entails a degradation in the performance of the system. Nevertheless, aspects as LB and HA at control plane are also covered in multi-instance solution.

The overall performance is a trade-off between scalability and throughput, that at the end stills represent a challenge to carrier's grade topologies. Nevertheless, the overall complexity of the system is reduced with an auto configuration modules for distributing the charge along the cluster instances.

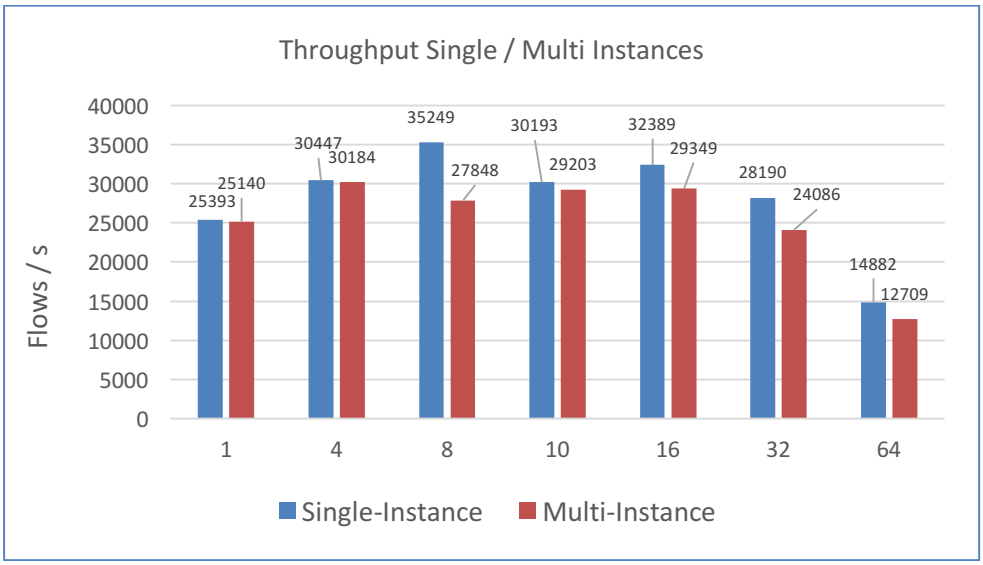


Figure 3.16 Single / Multi Instances - Throughput

3.1.4 Virtual Private LAN Service VPLS – Use Case

By default, the ONOS project implements a VPLS application that allows the provisioning of multi-point broadcast L2 circuits between multiple end points in an OpenFlow networks. In order to join logically several individual LANs to function as a single LAN, VPLS incorporates functions as MAC address learning, flooding, and forwarding functions in the context of pseudowires that connect these individual LANs across the network [50].

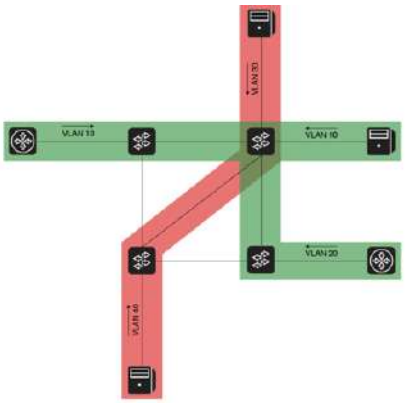


Figure 3.17 VPLS Pseudowires - Point to Multipoint (Green Path) / Point to Point (Red Path) [67]

Previous Figure analyzes a use case, nevertheless VPLS app is still in a test phase, a stable development will be available for production in following ONOS releases. Based on ONOS documentation, there has been defined several steps to establish the VPLS

pseudowires. First, a VPLS needs to be defined, this could be done through the ONOS CLI. Later the corresponding interfaces have to be configured and associated to a certain VPLS and finally, the hosts participating to the same VPLS can send in packets tagged with the same VLAN IDs, different VLAN ID or non-VLAN IDs at all.

In order to configure the SDN environment, the VPLS application must be included in the cell configuration (org.onosproject.vpls) for an automatic start, an example of this kind of configuration is shown in Appendix C.

Due to the test phase of the VPLS app, just an analysis of the procedure has been presented in this section instead of a simulation. Some enablers apps are needed in order to support the VPLS function. The main is the Intents Framework of ONOS an approach of PF.

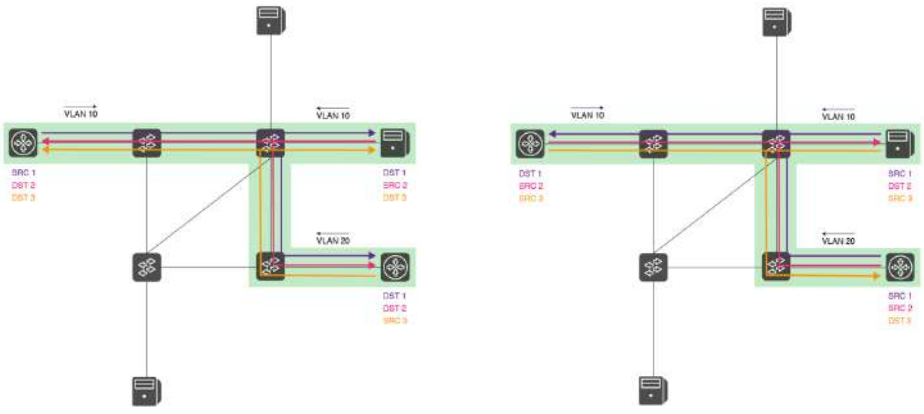
VLPS uses the Intents subsystem to install broadcast and unicast flow entries in the switches [67]:

Single-Point to Multi-Point intents: according to the simple example topology at Fig. 3.17 the required flow entries to install in order to configure a broadcast connectivity between the green path elements are shown in Fig. 3.18 (a). The installed flows follow the optimal path based on the optimal shortest path algorithm. If the host or edge devices are in a different VLAN as in the showed case, the intent framework automatically performs the VLAN ID translation. N edge ports associated to the same VPLS corresponds to N Single – Point to Multi-Point intents installations. The corresponding installed flows are:

- SR1 → 2 DST1(purple line), broadcast traffic
- SR2 → 2 DST2 (pink line), broadcast traffic
- SR3 → 2 DST3 (orange line), broadcast traffic

Multi-Point to Single-Point intents: in the same way, at the Fig 3.18 (b) the required flow entries to install at switches and provide a unicast traffic between the elements of the green path are shown below. N edge ports associated to the same VPLS corresponds to N Multi-Point to Single – Point intents installations.

- 2 SR1 → 1 DST1(purple line), unicast traffic
- 2 SR2 → 1 DST2 (pink line), unicast traffic
- 2 SR3 → 1 DST3 (orange line), unicast traffic



(a) *(b)*

Figure 3.18 (a) Single-Point to Multi-Point, Broadcast Traffic, (b) Multi-Point Single-Point, Unicast Traffic [30]

The VPLS function is in charge of automatically install the related intents for each configured VPLS. In the use case at Fig 3.17, in order to provide two different pseudowires green and red paths, two VPLS labels has to be configured.

Depending on the DC size or even the WAN size (number of branches) for an L2VPN environment, the overall required installations could represent a challenge to a proper scalability due to the high number of required intents. In that sense, the HA and scalability solution presented in the distributed approach off multi – instance controllers at control layer, should also take into account the synchronization of the intents. This last one is the reason to do not simulate the overall VPLS environment in this section, due to a non-stable replication of flows installations by intents, between controllers.

The VPLS solution could solve the actual scalability problems of VLANs which has a limited number of 4096, which is not a new solution, but the future ONOS release will offer a stable support for this service.

Chapter 4

4.1 Coexistence of Legacy and SDN Networks

In contrast with the scenarios in the last chapter, the analysis of the coexistence legacy networks and SDN networks has been done from a carrier perspective. At the DC, configurations are performed at L2, providing a LAN environment. Connectivity in a WAN environment could also be implemented at L2, interconnecting different branch offices with VPLS.

On the other hand, to provide the same WAN scenario at L3 some proposals for SDN networks are currently in production. Mainly the proposals are based on Quagga routing suite [68].

An example, Router-Flow proposal is composed of an OpenFlow controller application and an independent RouteFlow server that manages a virtual network environment to interconnect virtualized IP routing engines [69], Fig 4.1.

In the same direction, Quagga software demonstrates interoperability between Quagga suit and an SDN controller. A real use case [70], demonstrate the features of Quagga managing an OpenFlow 1.3 Switch and 2 traditional Cisco routers, sharing with the Quagga installed at the controller, OSPF, and BGP routing updates.

A proper implementation of SDN and legacy networks coexistence at L3, must not require modifications of existing routing protocols. Moreover, legacy infrastructure should be transparently integrated allowing communication between virtual routers implemented at the controller (Quagga) and physical in legacy networks.

The final solution has to ensure that routing control messages are correctly delivered from OpenFlow switches to virtual routers and vice versa, avoiding inconsistencies between the IP and the OpenFlow network.

4.1.1 Quagga Routing Suite

A system with Quagga installed acts as a dedicated router. It is defined as an open source virtual router implementation that provides TCP/IP based routing features with routing protocols support. Quagga supports static routing for small environments and dynamic routing for more demanding environments [33] such as a carrier grade topologies.

Currently, Quagga supports BGP, OSPF, RIP and IS-IS and a future support for MPLS. In addition to traditional IPv4 routing protocols, Quagga also supports IPv6 routing protocols that make ideal for further investigations and possible implementations in IoT scenarios.

The architecture is based on a collection of daemons running on LINUX instances. The static routing entries are configured directly at *Zebra,* which works as the kernel interface, and the dynamic updates of the dynamic protocols are installed at each corresponding service.

It is also important at the configuration to enable the forwarding capabilities at the localhost where the Quagga runs, due to this virtual router uses the Linux capabilities for forwarding the packets.

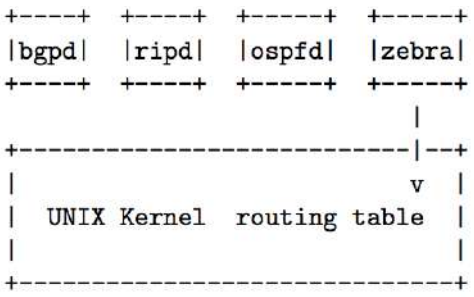


Figure 4.1 Quagga Architecture [33]

Inside Quagga suit several routing protocols have been described by daemons and zebra kernel for changing the kernel routing table and for redistribution of routes between different routing protocols. Some example of running daemons at Quagga and current support are shown in Table 4.1.

Daemon	Protocol / Function
ripd	RIP
ripngd	RIPng
ospfd	OSPF V2
bgpd	BGP-4
isisd	IS-IS
zebra	kernel Routing Table Manager

Table 4.1 Quagga Daemons

In order to simulate the Quagga vRouter and test the coexistence of SDN and legacy networks in L3, the GNS3 software is used. From a provider point of view, the proposed topology is shown in Fig 4.2, where a simple example is described by three blocks:

1. *Legacy Network Block* (components Legacy Router and Switch)
2. *Routing Protocol Block* (BGP)
3. *SDN Network Block* (components Quagga-vRouter and Mininet).

The general idea of the test is to check the connectivity from segment *192.168.200.0/24* at traditional network and the segment *192.168.30.0/24* located at the SDN network going through a BGP session between two BGP peers which is commonly related to an internal BGP (iBGP) session where the BGP peers are in the same autonomous system, in this case AS 65000.

Despite the configuration shows an internal BGP session in the same AS, exterior BGP sessions (eBGP) to perform inter-domain routing in TCP/IP networks, are currently supported by Quagga based SDN networks.

Although the BGP study is out of the scope of this work [71], it has been used for the simulation environment due to currently it plays a critical role in the internet, providing communication between AS and also used in MPLS environments.

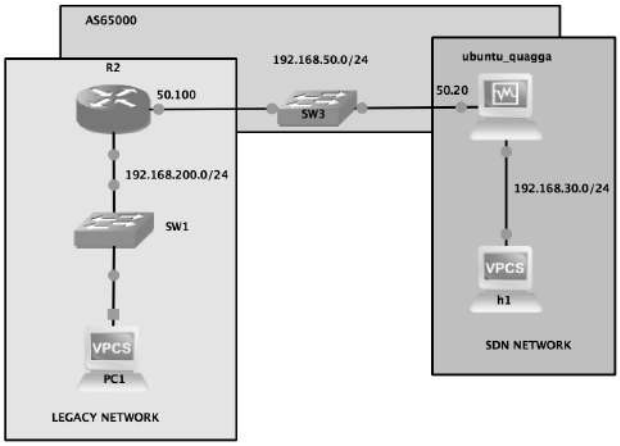
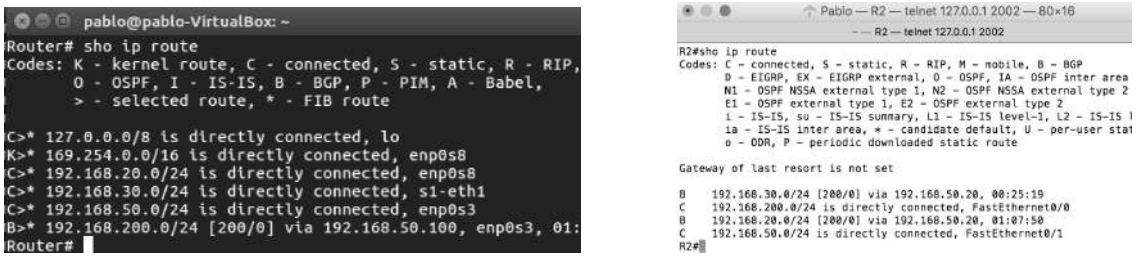


Figure 4.2 Quagga vRouter Simulation on GNS3

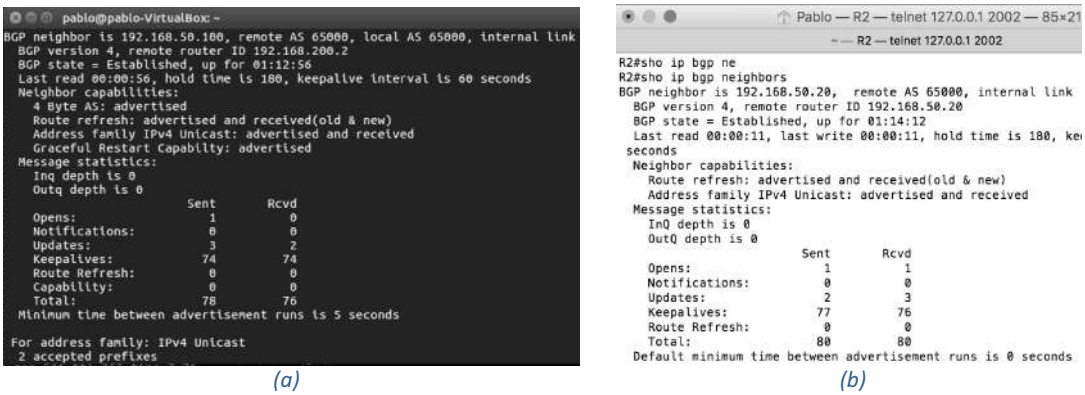
A general approach to BGP router requires to establish a connection on TCP port 179 to each of its BGP peers, in the simulation case two BGP peers has been configured for each network, legacy, and SDN. Once the session between peers has already established BGP updates can be exchanged, Fig.4.3.



(a) *(b)*

Figure 4.3 (a) BGP Learned Routes at vRouter, (b) BGP Learned Routes at Legacy Router

The BGP session between two BGP peers is said to be an external BGP (eBGP) session if the BGP peers are in different AS. In the other hand, if a BGP session is done between two BGP peers located in the same AS an internal BGP (iBGP) session is established. Figure 4.4 shows the iBGP established a session from the simulation environment, with an AS 65000, BGP-4 and the neighbors, legacy Cisco router and SDN virtual router Quagga.



(a) *(b)*

Figure 4.4 (a) BGP Status vRouter, (b) BGP Status Legacy Router

In the previous scenario, just a few *direct connected* routes have been exchanged in the iBGP sessions. Nevertheless, complex scenarios are also supported based on BGP-4 [71].

Routing capabilities as IntraAS and InterAS or a transit routing are also supported. In that sense, SDN networks could be implemented in carrier grade topologies running any of this scheme, ***overlaying BGP-4*** as the main protocol used between providers.

A proof of concept of an overlay BGP scenario is available in the reference [72]. This specific example shows a practical BGP based transition solution named *BTSDN* for a coexistence environment between legacy and SDN networks. The reference proposes to continue using the current BGP protocol and retains the legacy BGP border routers to connect the OpenFlow network with the rest of Internet in the same way that this section describes using a vRouter and overlaying BGP.

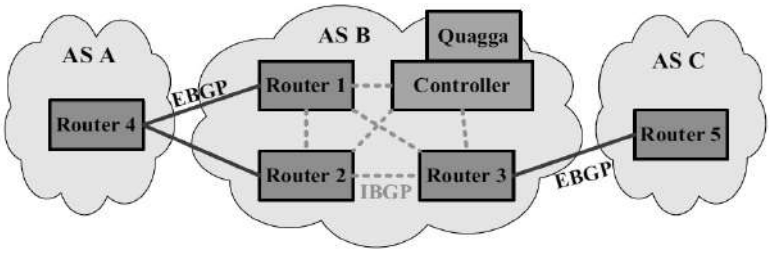


Figure 4.5 Proof of concept – Inter Domain Topology [72]

Unlike to IGP protocols, such as RIP or OSPF, BGP takes into account other metrics for establishing the path, such as predefined network policies which go according to SDN architecture, making it feasible for implementing in a coexistence scenario between the virtual router and the SDN controller.

A further investigation for a proper metrics and policy distribution performed at the control plane (controller) is still pendent. In that sense, a controller able take decisions based on BGP metrics and its own policies could result in a similar implementation such as BGP-RR nowadays, but with an optimal path computation.

CONCLUSIONS AND FUTURE WORK

In an SDN architecture, the centralized concept provides to the controller a complete view of the entire network. In that sense, some OpenFlow-based forwarding methods have been defined, Reactive, Proactive and hybrid. Independently of these mechanisms, implemented at the control layer, it has been shown that the recovery time, packet losses and the overall network performance, in an SDN network, presents better results than those achieved with legacy networks with a clear applicability to DC and carrier networks, with oriented applications based on above-mentioned forwarding modes. This allows migration scenarios from legacy to SDN networks and also migration paths for coexistence environments.

Network features such as high availability and load balancing are currently implemented in SDN scenarios with solid but open source NOS at the control plane.

Furthermore, solutions to the major SDN constraint, the scalability, have been analyzed along this document. Using a distributed flat control design, scalability in an SDN network is achievable with a certain performance degradation, due to the added controller to controller channel, which allows the synchronization and the overall consistency required by a multi-instance scenario.

Proposed simulations compare legacy and SDN networks to understand typical problems and SDN proposed solutions. For example, STP is commonly used in legacy networks to provide a certain grade of redundancy at L2, nevertheless, the performance of STP in front of an event is not affordable by new data center scenarios, where latency times and resiliency of the network present completely different requirements. In this context, SDN results in a suitable solution with minimum recovery delays.

Furthermore, if RSTP is used instead of STP in traditional networks, SDN and the forwarding mechanisms work as the shortest path bridging protocol SBP, which in fact is oriented to replace any STP versions. In the case of a link failure, STP does not guaranty an optimal path computation. If the shortest path computed by STP corresponds to the most congested route, an efficient use of resources could not be guaranteed.

In contrast, SDN not only computes the shortest path but also all the available paths from source to destination with the possibility to include other metrics in the computation such as the bandwidth usage or the data rate at each link.

Besides the data plane HA, there is the control plane where a centralized SDN architecture represents a big problem. In that sense, the controller plays a critical role and must guarantee the HA of the overall system in SDN environments.

In the simulations with a multi-instance environment, which constitutes a distributed flat control design, the results show that the high availability at control plane follows a trade-off between scalability (clustering) and system performance. Furthermore, the controllers clustering also represents a load balancing solution for the control plane, distributing the overall switch charge between the components of the cluster. The processor capabilities at each controller, for the first implementations, must be considered, meaning a high CAPEX to provide a solution with future flexibility and scalability.

On the other hand, in order to provide a load balancing scenario in legacy networks at L2, some solutions based on STP are currently used as the MSTP used to achieve trunk load-balancing. Issues as the VLAN scalability appear with this kind of

solutions. Nevertheless, the solution corresponds to a policy-based scenario, with no dynamic reconfiguration.

By the moment, SDN provides policy-based and dynamic load balancing solutions. The first one, PF is considered a policy-based forwarding mechanism with pre-established flow entries at each switch. This PF method reduces the computation time and results in a faster packet forwarding solution.

The second solution RF is based on flows entries, which are automatically installed at each switch along the path, based on shortest path computation, without taking into account the bandwidth occupation at each link.

Compared with PF, RF results in higher latency due to the OpenFlow messages OFPT_PACKET_IN/OFPT_PACKET_OUT with the controller to establish the E2E path, although the RF guarantees an efficient use of flow tables due to the time-out configuration.

A migration scenario from legacy to SDN networks present clear challenges, but a coexistence solutions are currently in production. BGP as the major routing protocol is presented as a clear migration path to L3 features, interconnecting DC through open source virtual routing as Quagga suite, with support for BGP, OSPF, and LDP in order to support MPLS.

Furthermore, overlays are also a use case for a migration scenario, not only for L3 but also for L2, such as VPLS.

The present document could be used as a starting point for future works with respect to migrations or coexistence scenarios but also to improve actual mechanisms and applications for SDN networks:

- An extended Dijkstra's algorithm, corresponding to a dynamic load balancing, could be used in SDN networks. It considers both the edge weights and node weights. Reference [48] probes that latency reduces in SDN networks using this extended algorithm. Further research is needed in order to develop an application that includes more metrics that the number of hops in order to compute an optimal E2E path.
- As MPLS results complex and requires high maintenance, new forwarding mechanisms as Segment Routing appears as a solution which makes SDN network more scalable, less complex and flexible. The work in [73] provides an overview of future networks with segment routing and SDN concept. Development of a segment routing application requires a further research.

BIBLIOGRAPHY

[1] S. Mcquistin and C. Perkins, "Reinterpreting the Transport Protocol Stack to Embrace Ossification – Position Paper," 2015.
[2] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Comput. Networks*, vol. 112, pp. 279–293, 2017.
[3] ONF TS-025, "OpenFlow Switch Specification 1.5.1," *Current*, vol. 0, pp. 1–36, 2015.
[4] M. Report, "The Future of Network Virtualization and SDN Controllers."
[5] "The OpenDaylight Platform | OpenDaylight." [Online]. Available: https://www.opendaylight.org/. [Accessed: 29-May-2017].
[6] "ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out." [Online]. Available: http://onosproject.org/. [Accessed: 29-May-2017].
[7] E. S. Denazis, J. Hadi, S. Mojatatu, N. D. M. Brocade, and O. Koufopavlou, "RFC 7426 - Software-Defined Networking …SDN—: Layers and Architecture Terminology," pp. 1–35, 2015.
[8] P. Goransson, C. Black, and C. Timothy, *Software Defined Networks A Comprehensive Approach*, Edition, 2. 2017.
[9] Z. L. K. Ben Pfaff, Bob Lantz, Brandon Heller, Casey Barker, Dan Cohn, Dan Talayco, David Erickson, Edward Crabbe, Glen Gibb, Guido Appenzeller, Jean Tourrilhes, Justin Pettit, KK Yap, Leon Poutievski, Martin Casado, Masahiko Takahashi, Masayoshi Kobayashi, Nick M, "OpenFlow Switch Specification," *Open Netw. Found.*, pp. 1–56, 2011.
[10] "What is Open vSwitch Database or OVSDB? - Definition." [Online]. Available: https://www.sdxcentral.com/cloud/open-source/definitions/what-is-ovsdb/. [Accessed: 30-May-2017].
[11] SDNxCentral, "What are SDN Northbound APIs?," 2015.
[12] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," *Proc. 4th ACM/IEEE Symp. Archit. Netw. Commun. Syst. - ANCS '08*, p. 1, 2008.
[13] ONF, "ONF Overview," 2014.
[14] O. N. Fundation, "Software-Defined Networking: The New Norm for Networks [white paper]," *ONF White Pap.*, pp. 1–12, 2012.
[15] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Futur. Internet*, vol. 6, no. 2, pp. 302–336, 2014.
[16] V. Dobrota, "A Telecom Perspective on SDN and OpenFlow," no. November, 2015.
[17] B. Heller, "OpenFlow Switch Specification 1.0.0," *Current*, vol. 0, pp. 1–36, 2009.
[18] C. Hao, Chang, and Y.-D. Lin, "OpenFlow Version Roadmap," pp. 1–15, 2015.
[19] ONF, "OpenFlow Switch Specification 1.3.0," *Current*, vol. 0, pp. 1–36, 2012.
[20] ONF, "OpenFlow Switch Specification 1.4.0," *Current*, vol. 0, pp. 1–36, 2013.
[21] C. Decusatis and D. Ph, "Towards an Open Data Center with an Interoperable Network (ODIN) Volume 3 : Software Defined Networking and OpenFlow," vol. 3, no. May, 2012.

[22] Gartner, “Predicting SDN Adoption - Andrew Lerner.” [Online]. Available: http://blogs.gartner.com/andrew-lerner/2014/12/08/predicting-sdn-adoption/. [Accessed: 26-Apr-2017].
[23] I. Pepelnjak, “OpenFlow and SDN : hype , useful tools or panacea ?,” *PPT Present.*
[24] W. Paper, “Virtual Extensible LAN (VXLAN) Best Practices,” no. January, pp. 1–32, 2016.
[25] “Microsoft’s Windows Azure network is a massive, virtual SDN.” [Online]. Available: http://searchsdn.techtarget.com/news/2240215890/Microsofts-Windows-Azure-network-is-a-massive-virtual-SDN. [Accessed: 24-Apr-2017].
[26] SDxCentral, “Rackspace Cloud Networks by Rackspace | SDN & NFV Product.” [Online]. Available: https://www.sdxcentral.com/products/rackspace-cloud-networks/. [Accessed: 26-Apr-2017].
[27] “SDN Solutions Overview | Pica8.” [Online]. Available: http://www.pica8.com/sdn-solutions. [Accessed: 24-Apr-2017].
[28] Cisco, “Cisco Open SDN Controller 1 . 2,” pp. 1–6, 2015.
[29] HP, “Software-defined networking and network virtualization,” p. 5, 2014.
[30] “HpOnline | AppStore.” [Online]. Available: https://marketplace.saas.hpe.com/sdn. [Accessed: 24-Apr-2017].
[31] SDxCentral, “The Future of Network Virtualization and SDN Controllers - Market Report,” 2016.
[32] “Atrium 2016/A: ONF + ODL | ONF BLOG.” [Online]. Available: https://www.opennetworking.org/?p=2015&option=com_wordpress&Itemid=316 . [Accessed: 24-Apr-2017].
[33] K. Ishiguro, “Quagga A routing software package for TCP/IP networks,” no. September, 2011.
[34] “Open Source SDN || Home.” [Online]. Available: http://opensourcesdn.org/our-projects/?id=2. [Accessed: 24-Apr-2017].
[35] “Open vSwitch.” [Online]. Available: http://openvswitch.org/. [Accessed: 24-Apr-2017].
[36] Vmware Inc., “The Network Virtualization and Security Platform,” pp. 1–4.
[37] Cisco, “Cisco Virtual Networking Solution for OpenStack,” 2013. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-kvm/datasheet-c78-730833.pdf.
[38] Indigo, “Indigo - Open Source OpenFlow Switches - Indigo - Project Floodlight - OpenFlow news and projectsProject Floodlight.” [Online]. Available: http://www.projectfloodlight.org/indigo/. [Accessed: 24-Apr-2017].
[39] SDxCentral, “Pica8 CrossFlow Integrates OpenFlow with L2 and L3.” [Online]. Available: https://www.sdxcentral.com/articles/news/pica8-crossflow-mixes-openflow-old-school-layer-2-layer-3/2014/11/. [Accessed: 24-Apr-2017].
[40] NoviFlow, “NoviSwitch - Noviflow.” [Online]. Available: http://noviflow.com/products/noviswitch/. [Accessed: 24-Apr-2017].
[41] IBM, “Application Guide G8264.”
[42] OpenFlow, “OpenFlow » Blog Archive » 10 Gigabit Ethernet OpenFlow switch from IBM.” [Online]. Available: http://archive.openflow.org/wp/2011/11/10gigabit-switch-from-ibm/. [Accessed: 24-Apr-2017].
[43] P. Barbecho, “Diseno y simulación de una topología y gestión de red basadas en túneles gre y enrutamiento dinamico OSPF y EIGRP,” PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR, 2017.

[44] N. Handigol and S. Seetharaman, "Plug-n-Serve: Load-balancing web traffic using OpenFlow," *Acm Sigcomm ...*, 2009.
[45] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild Into the Wild : Core Ideas," *Hot-ICE'11 Proc. 11th USENIX Conf. Hot Top. Manag. internet, cloud, Enterp. networks Serv. Serv.*, p. 12, 2011.
[46] Fujitsu Laboratories, "ONOS Application Load-balancing," 2016.
[47] ONOS, "Service Function Chaining (SFC) - onos test - Wiki." [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=12419330. [Accessed: 24-Apr-2017].
[48] J. Jiang, H. Huang, J. Liao, and S. Chen, "Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking," *IEICE - Asia-Pacific Netw. Oper. Manag. Symp.*, pp. 3–7, 2014.
[49] ONOS, "Cluster Coordination - ONOS - Wiki." [Online]. Available: https://wiki.onosproject.org/display/ONOS/Cluster+Coordination. [Accessed: 24-Apr-2017].
[50] K. Kompella and Y. Rekhter, "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling. IETF. RFC 4761," *RFC4761*, no. 7257. pp. 1–28, 2007.
[51] D. Quiroz, "Influence of the path establishment method on an SDN controller time response," *JITEL*, 2015.
[52] SDNCentral, "SDNCentral About Us - SDxCentral." [Online]. Available: https://www.sdxcentral.com/sdncentral-about-us/. [Accessed: 25-Apr-2017].
[53] OPNFV, "OPNFV Overview — Danube documentation." [Online]. Available: http://docs.opnfv.org/en/stable-danube/release/overview.html. [Accessed: 25-Apr-2017].
[54] ONOS Project, "Huawei Archives - ONOS." [Online]. Available: http://onosproject.org/tag/huawei/. [Accessed: 25-Apr-2017].
[55] "GEANT - SDN." [Online]. Available: https://www.geant.net/opencall/SDN/Pages/Home.aspx. [Accessed: 27-Apr-2017].
[56] OpenDaylight, "OpenDaylight Spectrometer." [Online]. Available: https://spectrometer.opendaylight.org/. [Accessed: 25-Apr-2017].
[57] Open HUB, "Open Hub, the open source network," 2015. [Online]. Available: https://www.openhub.net/. [Accessed: 25-Apr-2017].
[58] Open Hub, "The ONOS Open Source Project on Open Hub : Commits Summary Page." [Online]. Available: https://www.openhub.net/p/onos/commits/summary. [Accessed: 25-Apr-2017].
[59] Open HUB, "The OpenDaylight Open Source Project on Open Hub : Commits Summary Page." [Online]. Available: https://www.openhub.net/p/opendaylight/commits/summary. [Accessed: 25-Apr-2017].
[60] W. A. V. Vargas, "Emulación de una red definida por software utilizando MiniNet," *Esc. Técnica Super. Ing. en Telecomunicaciones (ETSIT - UPM)*, p. 8, 2013.
[61] "Cbench - Floodlight Controller - Project Floodlight." [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343657/Cbench+New. [Accessed: 25-Apr-2017].
[62] G. Technologies, "Software | GNS3." [Online]. Available: https://gns3.com/software. [Accessed: 25-Apr-2017].

[63] "How to setup the emulated Lab environment using Mininet VM and PFCv6 Trial VM . NEC Corporation of America," 2014.
[64] ONOS Project, "Environment setup with cells - onos test - Wiki." [Online]. Available: https://wiki.onosproject.org/display/test/Environment+setup+with+cells. [Accessed: 25-Apr-2017].
[65] V. Pashkov, A. Shalimov, and R. Smeliansky, "Controller failover for SDN enterprise networks," *SDN NFV Next Gener. Comput. Infrastruct. - 2014 Int. Sci. Technol. Conf. - Mod. Netw. Technol. Monet. 2014, Proc.*, vol. 1, 2014.
[66] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," *HotNets-XII Proc. Twelfth ACM Work. Hot Top. Networks*, pp. 1–7, 2013.
[67] ONOS Project, "VPLS Architecture Gude - ONOS - Wiki." [Online]. Available: https://wiki.onosproject.org/display/ONOS/VPLS+Architecture+Gude. [Accessed: 25-Apr-2017].
[68] "Quagga Software Routing Suite." [Online]. Available: http://www.nongnu.org/quagga/. [Accessed: 25-Apr-2017].
[69] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães, "Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks," *Proc. 6th Int. Conf. Futur. Internet Technol. - CFI '11*, vol. 1, p. 34, 2011.
[70] OpenSourceRouting, "ONS 2014 Demo Setup," 2014.
[71] D. Meyer, K. Patel, and C. Systems, "BGP-4 Protocol Analysis - RFC4274," pp. 1–16, 2006.
[72] P. Lin, J. Bi, and H. Hu, "BTSDN: BGP-Based Transition for the Existing Networks to SDN," *Wirel. Pers. Commun.*, vol. 86, no. 4, pp. 1829–1843, 2016.
[73] D. Cai, A. Wielosz, and S. Wei, "Evolve carrier ethernet architecture with SDN and segment routing," *Proceeding IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks 2014, WoWMoM 2014*, 2014.

APPENDIX A

In order to measure the general throughput of ODL and ONOS controllers, a simple virtual environment has been implemented. The components of the simulation are:

- VirtualBox - VMs
- Ubuntu Server 16.04 LTS - ODL
- Ubuntu Server 16.04 LTS - ONOS
- 2 GB RAM per VM
- 2 CPUs per VM

In addition, an external VM with the software *cbench* has been installed to benchmark controllers. This tool simultaneously emulates an OpenFlow controller – switch traffic, comprising the available capacity at the controller, which is mainly determine by the CPU at the VM. In that sense, cbench flood the controller and analyze the performance based on the number of flows per second processed by the controller.

Ten measurements have been done for each scenario ONOS and ODL with 1 switch and 1000 MACs. Annex 1 describes the CI for ONOS Controller. The results are presented with a 95% confidence that the mean is maintained within 219383 flows/s and 238859 flows/s with the described VMs characteristics.

	Flows/sec	Sample Mean	$(X_i - \overline{X})^2$
ONOS	190583	214265	560837124,00
	181304	214265	1086427521,00
	213387	214265	770884,00
	191104	214265	536431921,00
	223818	214265	91259809,00
	233918	214265	386240409,00
	241511	214265	742344516,00
	235616	214265	455865201,00
	226232	214265	143209089,00
	205184	214265	82464561,00

σ	95% CI
20213,48816	201736 flows/s - 226793 flows/s

Annex 1 ONOS Controller Throughput

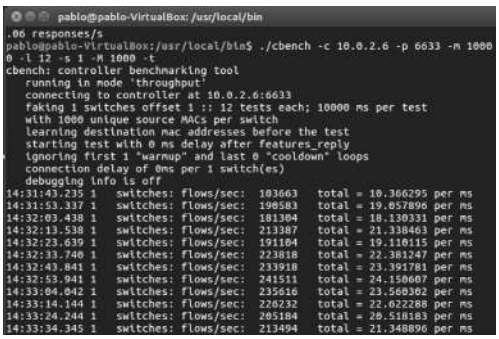
Under the same test parameters, Annex 2 describes the CI for ODL Controller. The results are presented with a 95% confidence that the mean is maintained within 79838 flows/s and 109887 flows/s.

	Flows/sec	Sample Mean	$(X_i - \overline{X})^2$
ODL	36814	94863	3369686401,00
	64621	94863	914578564,00
	111634	94863	281266441,00
	102873	94863	64160100,00
	91849	94863	9084196,00
	110554	94863	246207481,00
	96346	94863	2199289,00
	104487	94863	92621376,00
	122044	94863	738806761,00
	107410	94863	157427209,00

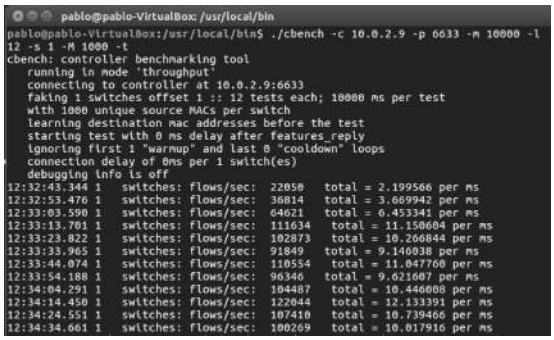
σ	95% CI
24240,54005	79838 flows/s - 109887 flows/s

Annex 2 ODL Controller Throughput

Annex 3, shows the test results for cbench running in mode throughput, in order to benchmark the controllers. Twelve test has been performed, but the first and the last has been discarded by the *warmup* and *cooldown* effect of the tool.



(a)



(b)

Annex 3 (a) ONOS Controller Throughput Test, (b) ODL Controller Throughput Test

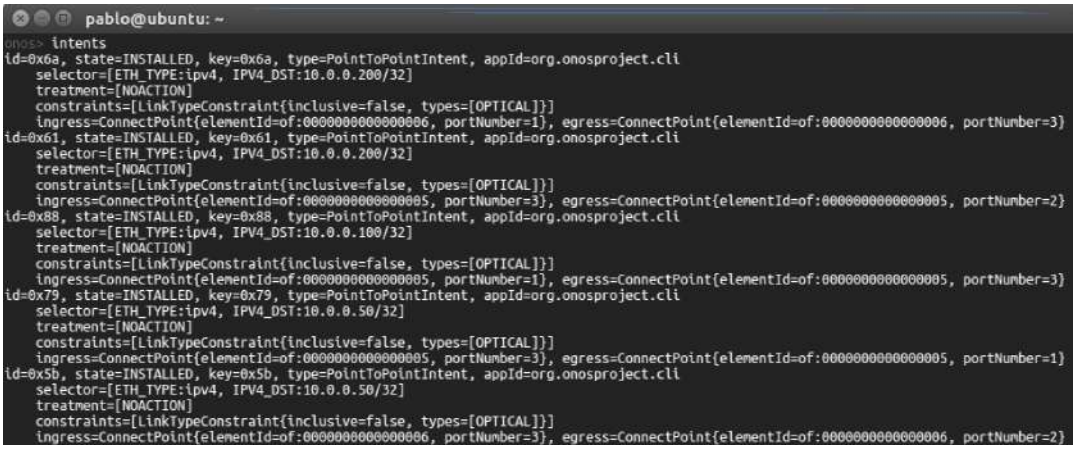
APPENDIX B

In order to install static flow entries along the E2E path, the ONOS CLI has been used, configuring two *'Point Intents'* at each switch per path (bidirectional). An example of the required command is shown below:

- Onos> add-point-intent –t IPV4 –p 200 –ipDst 10.0.0.50/32 of:0000000000000005/3 0f: 0000000000000005/1

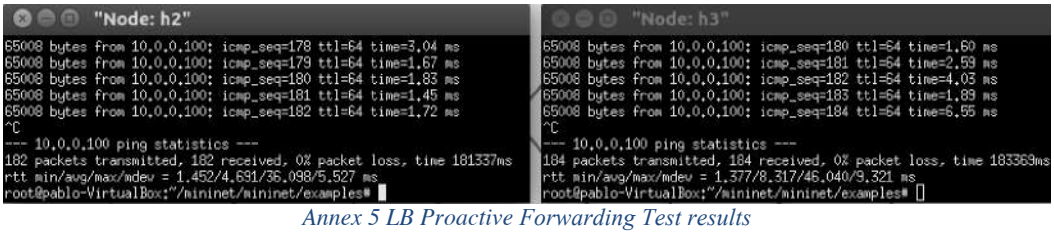
Where the –t option describes the version of internet protocol (IPV4/IPV6). The –p correspond to the priority and the –ipDst, as an option to match the destination of the packets. The Flow ID is described by of:0000000000000005/3, with the last two numbers describing the switch and the port in the switch (5/3 – switch S5 port 3). The first flows ID correspond to the incoming port and the second to the outgoing port.

Another way to instantiate an intent is from the GUI web console, where, in an easy form the end points have to be selected and then the option to create a path between them appears. In this case, the installation of flow is done in a PF way and taking into account the LB between different paths based on policy previously defined. Annex 4 shows some of the installed intents at each switch for the LB scenario.



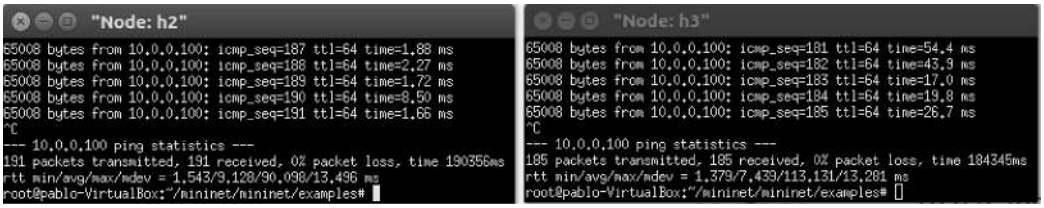
Annex 4 Intents installed for LB scenario

In addition, Annex 5 shows the minimum, maximum, and average time of response obtained in the simulation for the *proactive forwarding method*. The latency results have been included for each host H2 and H3.



Annex 5 LB Proactive Forwarding Test results

For the RF simulation, the intents are auto installed by the reactive forwarding subsystem. The Annex 6 shows the minimum, maximum, and average response time obtained for the reactive forwarding method.



Annex 6 Reactive Forwarding Test Results

APPENDIX C

The dynamic clustering test has been simulated using the script that comes with the ONOS Cardinal distribution. The required configuration is shown in Annex 7.

```
pablo@ubuntu: ~
pablo@ubuntu:~$ onos-form-cluster 192.168.1.1 192.168.1.2
+ ssh pablo@192.168.1.1 /opt/onos/bin/onos-form-cluster -u onos -p rocks 192.168.1.1 192.168.1.2
pablo@192.168.1.1's password:
Forming cluster on 192.168.1.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    91    0     0  100    91      0    419 --:--:-- --:--:-- --:--:--   421
Forming cluster on 192.168.1.2...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    91    0     0  100    91      0    112 --:--:-- --:--:-- --:--:--   112
pablo@ubuntu:~$
```

Annex 7 Clustering ONOS controllers

In the same way, cbench tool has been used to test the controller's performance when the cluster is built. Cbench perform continuous request of flows from every switch according to the number of MACs per switch, flooding the controller and analyzing the performance based on the number of flows per second processed by the controller.

Parameters as the number of switches, the number of tests, the time of each test and the number of MACs per switch are configured as the Figure 6 shows.

As an example, the figure shows the results of 10 tests, 8 switches and 1 controller before the cluster establishment.

```
pablo@OC2:/opt/oflops/cbench$ sudo ./cbench -c OC2 -p 6633 -m 10000 -l 10 -s 8 -M 100 -t
cbench: controller benchmarking tool
   running in mode 'throughput'
   connecting to controller at OC2:6633
   faking 8 switches offset 1 :: 10 tests each; 10000 ms per test
   with 100 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
02:25:20.207 8   switches: flows/sec:  33397  35528  15574  26235  23888  19725  19943  0   total = 17.428981 per ms
02:25:30.307 8   switches: flows/sec:  42124  43075  35486  42231  34837  64952  69699  21522   total = 35.392579 per ms
02:25:40.407 8   switches: flows/sec:  34919  29382  34649  55910  69123  51609  45804  51465   total = 37.286025 per ms
02:25:50.509 8   switches: flows/sec:  33357  42318  37148  48740  54576  41289  41404  37995   total = 33.679591 per ms
02:26:00.610 8   switches: flows/sec:  48391  69475  41087  61863  32628  36479  40067  51286   total = 38.127554 per ms
02:26:10.710 8   switches: flows/sec:  27694  45618  38400  24062  53706  51514  37421  36658   total = 31.507237 per ms
02:26:20.811 8   switches: flows/sec:  29967  36198  55759  41619  27055  61319  53243  22328   total = 32.748613 per ms
02:26:30.931 8   switches: flows/sec:  40853  49018  30165  33395  78131  37214  26379  66591   total = 36.106247 per ms
02:26:41.036 8   switches: flows/sec:  21714  34959  45976  44557  46384  42157  59318  41632   total = 33.654885 per ms
02:26:51.137 8   switches: flows/sec:  62251  55965  38054  48892  30679  45288  70434  35838   total = 38.740096 per ms
RESULT: 8 switches 9 tests min/max/avg/stdev = 31507.24/38740.10/35249.20/2371.96 responses/s
```

Annex 8 ONOS Controller Performance Results - 1 Switch

In addition, the processor consumption at the controller, when cbench is in execution is show in Annex 9. As has been described, the VM is completely overload due to cbench test in order to check the overall throughput of a single controller instance.

```
~ — pablo@OC2: ~ — ssh pablo@192.168.1.178
  1  [||||||||||||||||||||||||||||||||||||||||||||||||||99.3%]     Tasks: 38, 242 thr; 9 running
  2  [||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]   Load average: 3.84 2.82 2.53
  Mem[||||||||||||||||||||                              796/2656MB]   Uptime: 00:28:40
  Swp[                                                    0/2043MB]

   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  5627 pablo      20   0 1052M  700M 19556 S 151. 26.4 16:04.00 /usr/lib/jvm/java-8-oracle/bin/java -server -Xms128
  5973 root       20   0  6400  4636  1788 S 49.5  0.2  0:13.01 ./cbench -c OC2 -p 6633 -m 10000 -l 10 -s 8 -M 100
```

Annex 9 Processor status at the controller with cbench running

Furthermore, analyzing the multi-instance environment, 2 controllers had been installed in the cluster. The simulation environment is composed by:

- VM Controller 1: Ubuntu Server 16.04 LTS, 2 Cores, 16G SDD
- VM Controller 2: Ubuntu Server 16.04 LTS, 2 Cores, 16G SDD

The same simulation test has been performed at each controller ones the cluster is established.

There also a cell reconfiguration is needed, including the IPs of each controller in the script. The configuration has to be done at each cell script before launching the cluster. Cell configuration is shown in Annex 10.

```
export ONOS_NIC="192.168.1.*"
export OC1="192.168.1.186"
export OC2="192.168.1.178"
export OCI="192.168.1.186"
export OCN="192.168.1.187"
export ONOS_APPS="drivers,proxyarp,openflow,vpls,fwd"
```

Annex 10 Cell configuration