Universidad de Extremadura

Escuela Politécnica

Máster en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Performance comparison of layer 2 convergence protocols. A SDN approach.

Sergio Sánchez Herranz

Junio, 2018

Universidad de Extremadura

Escuela Politécnica

Máster en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Performance comparison of layer 2 convergence protocols. A SDN approach.

Autor: Sergio Sánchez Herranz

Tutor: Rafael Martín Espada

A mi familia
y la comunidad universitaria.

Contents

List of Figures

List of Tables

Abstract

This project aims to explore the most common network topologies of data-center environments its technologies and some of their defining properties as redundancy, number of devices required, and the costs to implement it. As well as reviewing and testing the main layer-2 convergence protocols that make this redundant networks usable.

Besides that, we will take a look at the concept of Software Defined Networking and the main concepts behind it such as; data and control layer separation, and centralized management. Additionally some SDN technologies and tools will be briefly reviewed to better understand the concepts.

The test of the convergence protocols have been realized in a simulated environment built on GNS3, a network simulation tool that uses VMWare to run the GNS3 VM and a variety of docker containers for the different devices that conform the data-center network.

Keywords:
convergence, time, spanning, tree, protocol, STP, link, layer, software, defined, networks, SDN, rapid, RSTP, docker, GNS3, OpenDaylight, controller, Open, vSwitch, time, data-center, data, center

Chapter 1

Introduction.

This project has been motivated by the increasingly important role that computer networks have taken in today's economy and the drive to know how high availability and redundancy is achieved and how the problems associated with its implementation are solved.

The typical data-center was historically just a fast and fairly scalable LAN to connect all data-center devices and equipment. But with the appearance of cloud computing arose a need for better integration with storage and computing resources within the network e.g, private clouds.

Those data center networks (DCN) are a key component of enterprises revenue and communications infrastructure of modern companies and its importance is enough for businesses keep spending on updating their infrastructure [1]. Those expectations are why data-center networks should be a example of resiliency and high-availability.

However, architecting for high availability means that the data center network will show redundancy of its links and nodes, and in most cases this redundancy will create loops in the topology that may cause network traffic issues if improperly managed.

Those problems with loops within the network is why convergence protocols and traffic engineering techniques are needed. Also, data center networks are usually built using high performance layer 2 devices and that rules out routing protocols that work over IP like OSPF or EIGRP.

With this in mind, the present document aims to analyze these convergence protocols, their performance, and the environments where they are primarily used.

Chapter 2

Objectives.

The objectives of this project are aimed towards understanding layer 2 convergence protocols and their use. Additionally, to understand these protocols is necessary to know the network environment they are present in.

The main objectives of this project will be:

- Identify the layer-2 protocols used to achieve convergence.
- Review these protocols and how they work.
- Analyze data center environments where these protocols could be used.
- Identify possible tools to test and simulate these environments.
- Identify and review technologies deployed in this environments.
- Simulate a test environment using the tools reviewed.
- Implement the protocols in the test environment.
- Test the performance of these protocols and analyze the results.
- Review use cases of SDN technologies related to convergence.

These objectives are proposed as a way to achieve a better understanding of data center networks, the technologies used in them and to take a glimpse at how future technology trends like Software Defined Networking (SDN) may be used as an alternative.

Chapter 3

Precedents & State of the art.

3.1 Convergence protocols.

3.1.1 Introduction.

In order to say that a network has achieved convergence, the devices of the network must share the same topological view of the network they are part of.

To achieve convergence is necessary to implement protocols that enable the network to build a logical loop-free topology. Also, these protocols allow networks to implement redundancy links which are used to provide backup links in case the active path gets shut down.

In this chapter the two main link layer convergence protocols will be reviewed, that is: the Spanning Tree Protocol and its evolution, the Rapid Spanning Tree Protocol. Vendor specific implementations will not be explained in this text but they may be briefly commented.

3.1.2 Spanning Tree Protocol.

"Spanning Tree Protocol (STP) is a Layer 2 protocol that runs on bridges and switches. The specification for STP is IEEE 802.1D. The main purpose of STP is to ensure that you do not create loops when you have redundant paths in your network. Loops are deadly to a network."[2]

As the Cisco website states, logical loops should never be present in networks. If for any reason a loop appears the network will eventually become congested due to the effects of broadcast radiation.[1]

STP works by creating a spanning tree within the layer 2 network. This means that between two network nodes there will only be one possible path, which prevents logical loops. To easily understand this concept we can examine Figure 3.1.

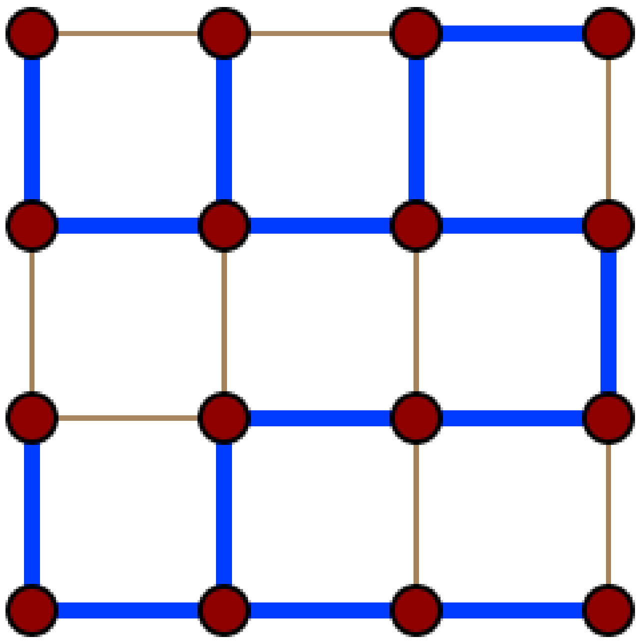


Figure 3.1: Grid network with spanning tree (in blue). *Wikipedia*

[1]Broadcast radiation is the accumulation of broadcast and multicast traffic on a computer network. Extreme amounts of broadcast traffic constitute a broadcast storm. A broadcast storm can consume sufficient network resources so as to render the network unable to transport normal traffic.

In Figure 3.1 a grid network topology is shown and the spanning-tree of the network is drawn in blue. We can observe that for two given nodes there is only one possible path and the other physical links are deactivated. All the nodes of the network share the same topological view, the network has converged. What STP tries to achieve is this functional and stable network state.

STP Operation

In order to create the spanning tree this protocols follows these steps:

Root Bridge Selection. The first step is selecting the Root Bridge (RB) of the network, this means identifying the bridge with the lowest ID. The ID of a bridge is the concatenation of the bridge priority (32768 by default) and the bridge MAC address. E.g., a bridge with MAC address 1111.1111.1111 will have a default ID of 32768.1111.1111.1111 . The priority can be changed in multiples of 4096. If two bridges have the same priority there is a tie and the MAC addresses are compared.

When configuring STP in the bridges we can select the Root Bridge simply by changing its priority to 4096 and is advisable to pick a secondary Root Bridge using a priority of 8192. This way in case the Root Bridge losses connectivity we know how the network will behave.

Computing least cost paths to the RB. In this second step the bridges start exchanging Bridge Protocol Data Units (BPDU) which contain the cost of their link to the Root Bridge in the **Root Path Cost** field. To better understand the how the protocol communicates take a glance at how a BPDU frame looks in Table 3.1

Table 3.1: BPDU frame structure

Protocol ID	**Version**	**Message type**	**Flags**
2 bytes	1 byte	1 byte	1 byte
Root ID	**Root Path Cost**	**Bridge ID**	**Port ID**
8 bytes	4 bytes	8 bytes	2 bytes
Message Age	**Maximum Age**	**Hello Time**	**Forward Delay**
2 bytes	2 bytes	2 bytes	2 bytes

- Protocol ID: zeros for STP specification 802.1D.

- Version: zero
- Message type: Configuration BPDU or Topology Change Notification (TCN).
- Flags: MSB in this field is the Topology Change flag and the LSB is the Topology Change ACK flag.
- Root ID: Root bridge identity. 2-byte priority plus its 6-byte MAC address.
- Root Path Cost: The cost of the path connecting the sender bridge (of this BPDU) to the Root Bridge.
- Bridge ID: Sender identity. 2-byte priority plus its 6-byte MAC address.
- Port ID: 2-byte fields with the identity of the sender port.
- Message Age: Specifies the age of the configuration data since it was sent from the Root Bridge.
- Maximum Age: If the Message Age exceeds the Maximum Age it's dropped.
- Hello Time: Time between Root Bridge Configuration messages.
- Forward Delay: How much time the bridge has to wait after receiving a TCN.

The Root Bridge starts by sending configuration messages with **Root Path Cost** of 0 with a periodicity specified in the **Hello Time** field. Each subsequent bridge adds the cost of the link in which they received the BPDU to the **Root Path Cost** and resends the newly computed BPDU through the remainder links until it reaches the edges of the network.

Different link technologies and configurations have different link costs. The link cost for STP can be found in Table 3.2. Also an administrator can manually specify the cost of a particular network segment.

A bridge compares the BPDU's it receives in its ports and selects the port that received the least cost path BPDU as its root port (RP).

Then the bridges start the configuration of Designated Ports (DP). Those are the ports in a LAN segment used to communicate directly with Root Ports.

Table 3.2: STP link costs. IEEE 802.1D

Data Rate	**STP path cost**
4 Mbps	250
10 Mbps	100
16 Mbps	62
100 Mbps	19
1 Gbps	4
2 Gbps	3
10 Gbps	2

In Figure 3.2 we can see the path cost and the computed spanning tree with root ports, designated ports and blocked ports in an example mesh network. This example network has achieved convergence with no logical loops but with many redundant links.[2]

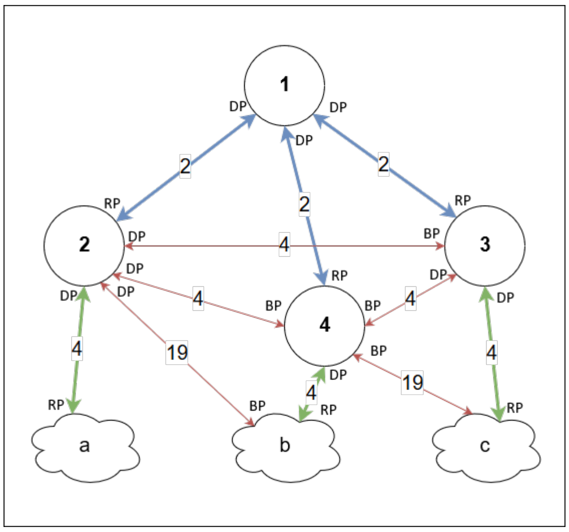


Figure 3.2: Spanning tree example. Shows the links cost and port roles assigned. In blue links to the root. In green links to hosts. In red blocked links.

[2]Additional rules exist to decide the role of ports when there are "ties", e.g., if the Root Bridge has more than one port connected to other bridge it will use the port with the lowest port ID (priority + interface number) as designated port and block the others. All additional rules revolve around the bridges and ports priorities and ID's.

3.1.3 Rapid Spanning Tree Protocol.

The Rapid Spanning Tree Protocol (RSTP) defined in the IEEE 802.1w is the evolution of the IEEE 802.1d Spanning Tree Protocol. It was published when the popularity of layer 3 routing environments was increasing and a new link-layer convergence protocol was needed. Before that Cisco had introduced additional STP features in their devices but these required additional configuration and were vendor specific [3]. Even with these modifications to STP, RSTP performs better than the Cisco-modified protocol. Later Cisco will also develop its own modifications for RSTP to further improve the performance.

Ports in RSTP can have 3 different states, that is: discarding, learning, and forwarding. The old disabled, blocking, and listening states of STP are unified in the RSTP discarding state (Table 3.3). However RSTP still has the same root, designated, and blocked port roles but it also adds the alternate and back-up roles.

Table 3.3: STP & RSTP Port States

STP	**RSTP**
Disabled	Discarding
Blocking	Discarding
Listening	Discarding
Learning	Learning
Forwarding	Forwarding

The BPDU that RSTP uses have few differences with STP's BPDU. It uses the same format but all bits in the flag field are used. These bits are used to manage the proposal/agreement method implemented in RSTP, and to encode the state and role of the source port. These similarities make RSTP compatible with STP bridges but if it inter-operates it loses most of its advantages and speed.

RSTP Operation

In contrast with STP, bridges generate their own BPDUs every hello-time which is 2 seconds by default and don't just relay the BPDUs that the Root Bridge sends. If a bridge does not receive BPDUs from an adjacent bridge, 3 times in a row, this bridge will consider that they have lost connectivity.

Edge ports, those connected to a host, can not create loops and therefore will change directly to a forwarding state.

Bridges start with their ports in the blocked port role (Figure 3.3) and will start to learn the network topology from their neighbors BPDUs. Two bridges will decide which one is the Root Bridge or which one has the least cost path to the Root Bridge.

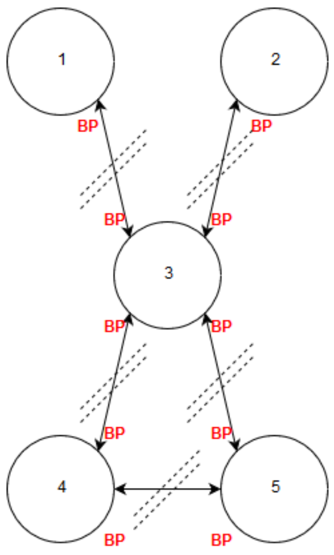


Figure 3.3: RSTP Initial state.

Once a bridge (ID:3) finds a Root Bridge (ID:1) or a least cost path to the Root Bridge, it blocks all non-edge ports and communicates to the Root Bridge (ID:1) that it should put its port in a forwarding state (and it will be a Designated Port). This negotiation process is unique to RSTP and it is not present in STP. When this happens the bridge (ID:3) puts its port in the forwarding state too and it becomes a Root Port (Figure 3.4).

This bridge will then communicate this newly discovered path to the Root Bridge in the BPDUs that it sends every hello-time, propagating this information through the network (Figure 3.5).

The algorithm will block ports in order to prevent logical loops, but blocked ports in RSTP can be alternate or backup ports. In Figure 3.6 the blocked port will become an alternate port because it is receiving a "bet-

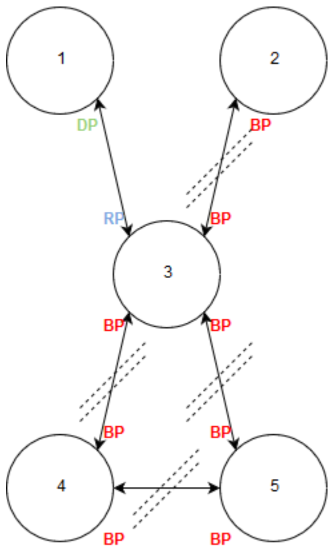


Figure 3.4: RSTP First negotiation (Shown only bridges 1 & 3).

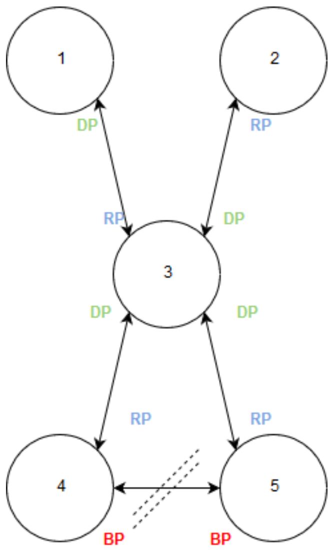


Figure 3.5: RSTP Second negotiation.

ter" BPDU in other port. A backup port will exist when a bridge has more than one port connected to same LAN segment.

When RSTP achieves convergence it should have the same topology as

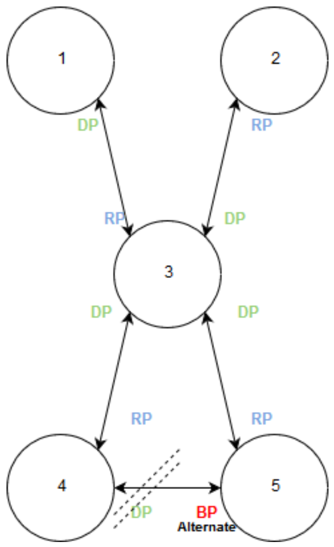


Figure 3.6: RSTP Convergence.

STP but it will theoretically do so in a shorter time. This is caused by the proposal/agreement method of deciding port roles, and because bridges don't just relay the information from the Root Bridge but communicate autonomously with their neighbors.

3.2 Data Center environments.

Data center infrastructure has received a lot of interest from researchers and academics. The growing importance of Internet-based applications like search engines, video distribution and social networks as well as data mining, or high performance computation has pushed the growth of data centers and companies in this market.

The importance of the data centers role in today's economy makes all aspects of these data center subject to study and enhancement including data center networking. Data center networking should be cost-effective in its implementation and its maintenance.

The topology of data centers is usually based on Servers connected to Top of Rack (ToR) switches, connected to End of Rack (EoR) switches which are connected through core switches. This forces core switches to support most of the bandwidth of the network and has motivated researches to come up with alternate approaches for scalable network architectures.

Data center network architectures are classified in fixed and flexible architectures and can be further classified as we can see in Figure 3.7.

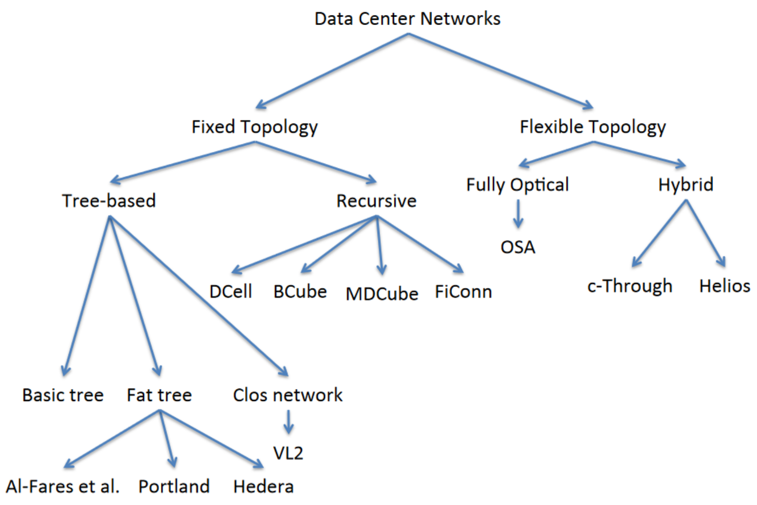


Figure 3.7: Data center networks [4].

3.2.1 Tree-based topologies.

These are the classic data center architectures with ToR, EoR and core switches but depending on the variant they show redundancy in the different levels. This classification includes Basic-Tree, Fat-tree and Clos network architectures.

Basic-tree architecture consist on two or more level of switches, usually core, aggregation and edge, with the servers being the leaves of the three. These architecture does not show any redundancy and the core switch (top level) has to be able to manage almost all the traffic in the network. An example Basic-tree is shown in Figure 3.8.

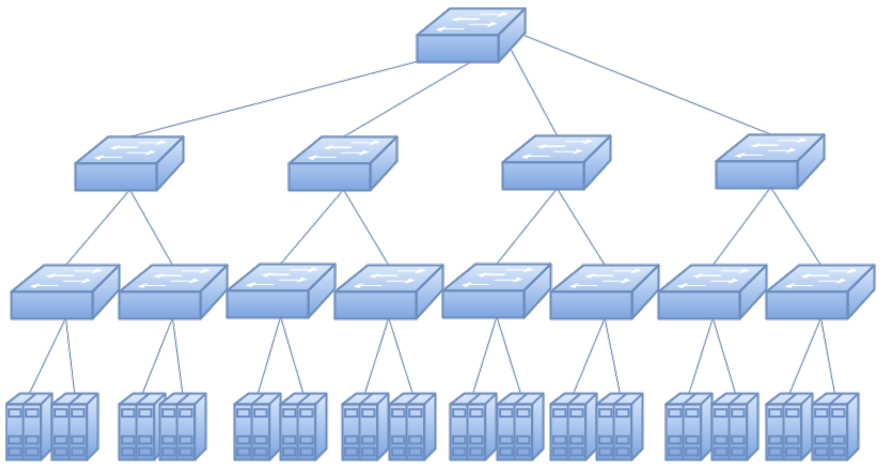


Figure 3.8: Basic Tree.

Fat-tree architecture is an evolution of the basic-tree. Take n-port edge switches connected to $n/2$ servers and to $n/2$ distribution switches, this construction is called a pod[5]. Then the remaining n ports in the distribution switches are connected to n core switches. Then pods may be added until all core switches have their ports used. This architecture and its variations have been for a long time the de-facto standard architecture for data-center networks. This architecture shows strong redundancy on almost all network levels but its core switches still have to support most of the network traffic. An example Fat-tree architecture is shown in Figure 3.9.

Clos network architecture is a variant of the three level tree network. The levels in this architecture are originally called intermediate, aggregation

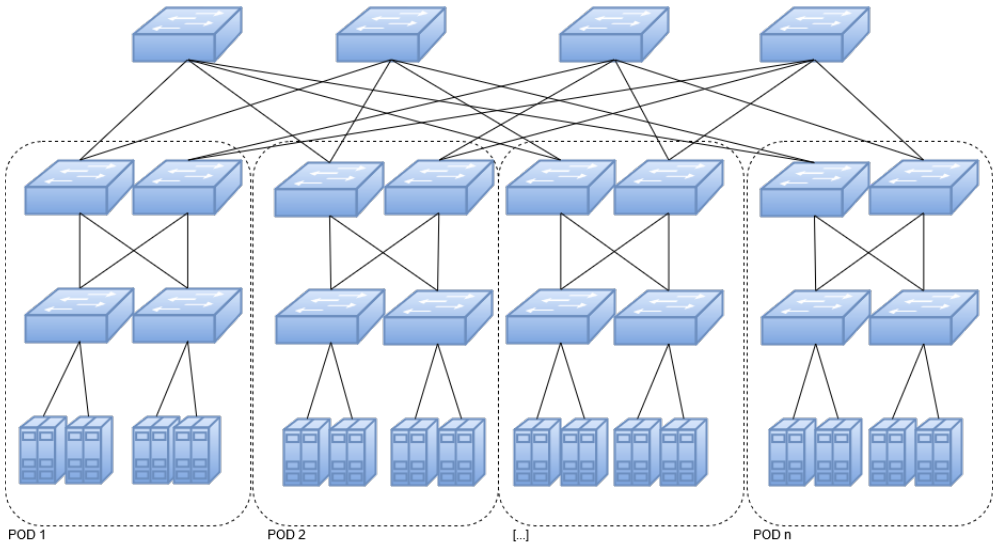


Figure 3.9: Fat-Tree.

and ToR. The number of switches in each level is given by the number of ports in the intermediate and aggregation levels. It will have n/2 n-port intermediate switches, n n-port aggregation switches and each one connected to n/2 ToR switches. The ToR switches are connected to 2 aggregation switches and to n-2 servers. This architecture exhibits a redundancy similar to that of the Fat-tree but uses less switches to do so. An example of n=6 Clos network is shown in Figure 3.10.

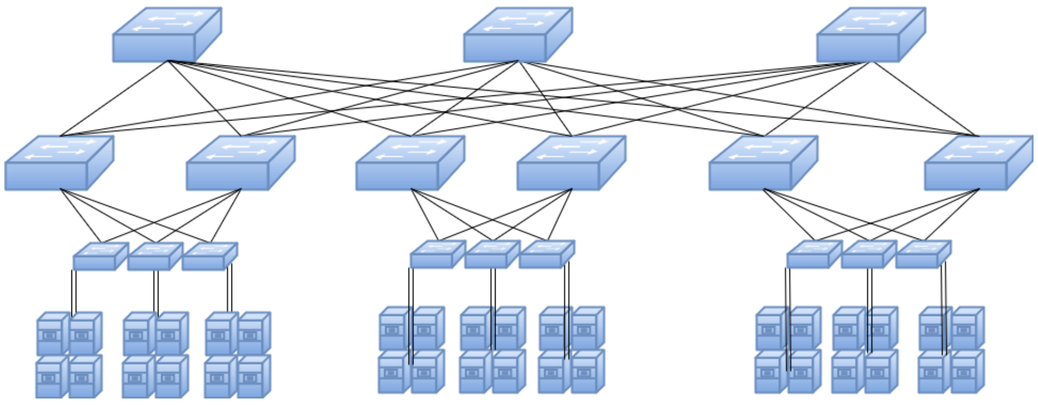


Figure 3.10: Clos network 6-ports.

3.2.2 Recursive topologies.

While it is beyond the scope of this document to explain this topologies, for illustrative purposes the DCell topology will be shown.

Recursive topologies are created from a basic Cell or Cube called Cell_0 or Cube_0 and higher level architectures Cell_ns are created interconnecting n Cell_0s [6]. The servers in this cells may be connected to different level switches or, sometimes to other servers.

A **DCell topology** is composed of smaller components called DCell_0s.

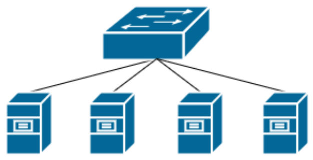


Figure 3.11: DCell−0 with 4-port switch.

Each DCell_0 has n servers connected to the same 1 n-port switch (Figure 3.11). A DCell_1 is composed of $n + 1$ DCell_0s with one link to every other DCell_0. The links are made between the servers of the DCell_0s (Figure 3.12).

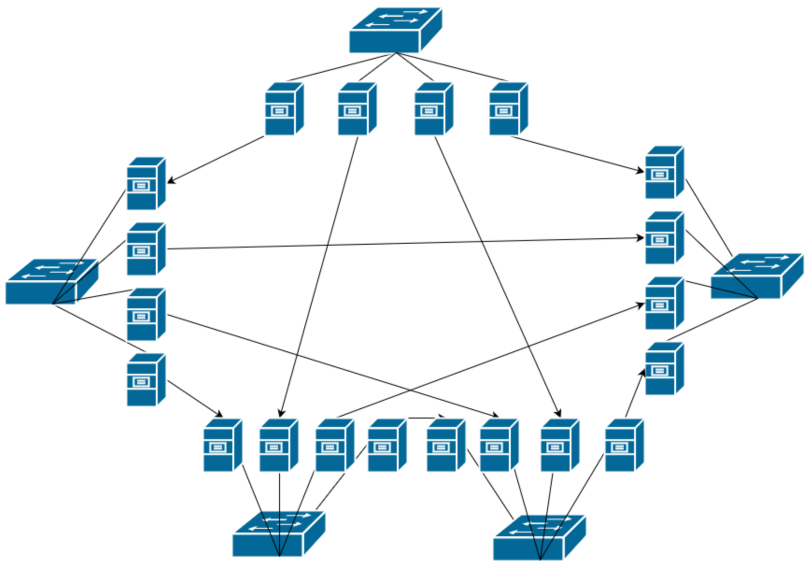


Figure 3.12: DCell−1 with 4-port switches.

3.2.3 Flexible architectures.

Flexible architectures make possible re-configuring the network architecture easily once deployed but use far more complex and costly network devices.

Once again this topologies are beyond the scope of this document but we can see it's main characteristics.

Hybrid architectures use both types of switching devices, electrical and optical.

In this category we find the **c-Through** network architecture which is composed of a typical tree network of electrical switching devices and a parallel optical re-configurable point-to-point network which automatically changes its configuration based on the current weights of the flows between servers [7].

Another example of hybrid architectures is the **Helios** topology. This is usually a two level tree topology in which the ToR switches are electrical but the core switches use optical technologies.[8] This topology have the advantage of addressing the bandwidth over-subscription that core switches have to support and are cheaper to maintain and deploy.

Lastly we can find **fully optical networks** which abandon the use of electrical switching. This fully optical architectures, like OSA[9], use optical switching devices which make this architectures fully re-configurable and capable of handling huge bandwidths.

3.3 Software Defined Networks

3.3.1 History.

The 90's saw the Internet grow with intensity and it experienced the appearance of more diverse applications and a greater use by the public. This new found demand drew researchers to test new protocols aimed to improve network services.

The way to do so was long and cumbersome and involved taking their ideas to the IETF (Internet Engineering Task Force) to approve and standardise them, which, over time, proved to be a slow and frustrating process.

Active Networking

Some researchers started working on an alternative way of network control. This approach was roughly based on the analogy of reprogramming a computer, which can be done relatively easy. Traditional networks are not programmable and this so called *Active Networking* involved a programming interface (API) that was capable of implementing custom functionality for a group of packets in a selected network node.

The issues that were to be solved by *Active Networking* included shortening the time needed to develop and deploy network services, more detailed control to meet the needs of applications or network conditions, and unified control of *middleboxes* as firewalls, proxies or transcoders, which started to be a concern for their vendor specific programming models.

It would seem that the problems that motivated the research on Active Networking were surprisingly similar to the ones that have motivated the push for SDN research today. Despite the similarities, it wasn't until the publication of an article about the OpenFlow API in 2009 that the term Software Defined Network (SDN) was used[13].

Software Defined Networks

Even though the term Software Defined Network appeared later, the concept of data-plane and control-plane separation appeared in the early 2000's and in a few years it would be one of the core concepts of what today is called SDN.

During this time the constant growth of traffic volumes and the need of a greater network reliability and performance revealed that the tools and methods used for traffic engineering were outdated and insufficient. Conventional routers and switches show a complete integration of data and control planes and this makes controlling the behavior of networks as a whole difficult.

Throughout the next years the OpenFlow API was developed within the Stanford University [15] and by the year 2010 it was the most used open interface with the help of controller platforms such as NOX.

OpenFlow compliant switches use a set of packet-handling rules called flow tables. These flow tables enable fine-grained network control and monitoring without the need of upgrading the present switches hardware. The SDN controllers are able to obtain a complete view of the network topology and modify the flow tables when needed.

OpenFlow initial target scenario was campus networks. So in the late 2000's the OpenFlow group at Stanford started an effort to deploy test networks on US campuses. As SDN materialized in universities, data center engineers started to use it to better manage network traffic at a large scale. It proved more cost-effective to write complex traffic controller programs using those SDN switches than to continue using vendor specific appliances with their own vendor specific programming and control models.

Additionally this new networking approach has made possible for smaller companies to compete in the established network equipment market by offering network devices which support SDN APIs such as OpenFlow.

3.3.2 SDN Architecture.

As stated before one of the main features of Software Defined Networks is the detachment of the control and data planes. This means that the main purpose of the network devices is forwarding and dropping packets while the routing protocols and other logic of the network is centralized in controllers.

The controller is the logical entity that receives the requirements of the network from an upper layer and a configures the network devices to fulfill these requirements. The controller has a general view of the network topology and it may be said that it acts as the "brain" of the network devices.

The network devices are responsible of forwarding and dropping data in the network. Being much less complex than their traditional counterparts are less costly and much easier to implement in virtualized environments.

We can compare the traditional switches and the SDN switches in Figures 3.13 and 3.14. Those are over simplifications for illustrative purposes.

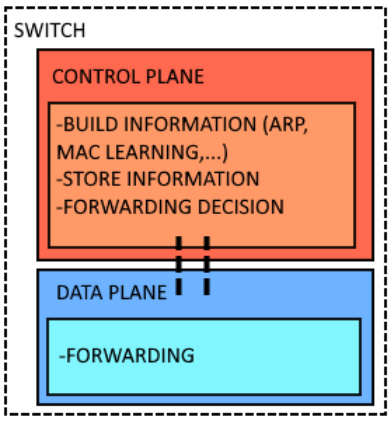


Figure 3.13: Data and Control Planes in a traditional switch.

It exists an additional layer in the SDN architecture, that is the application layer. This layer implements the business logic and manages the control plane as needed. Figure 3.15 shows a more complete view of the SDN architecture.

The separation of planes makes necessary to use APIs to communicate the different network layers. The southbound interface (SBI), which commu-

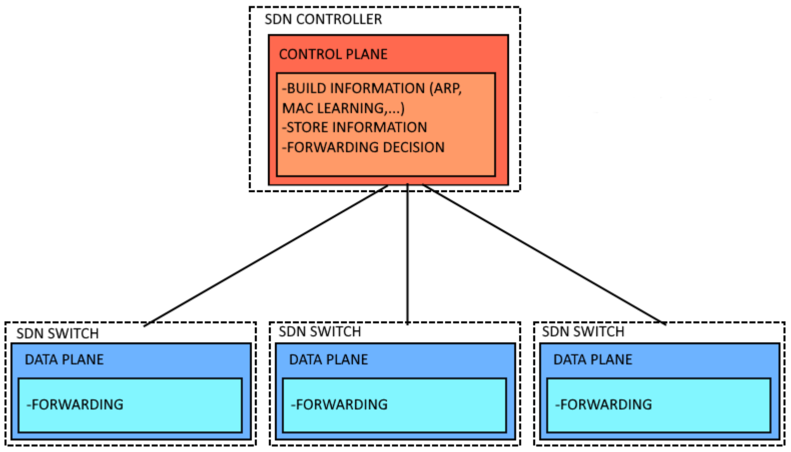


Figure 3.14: Data and Control Planes detached in switches and controller.

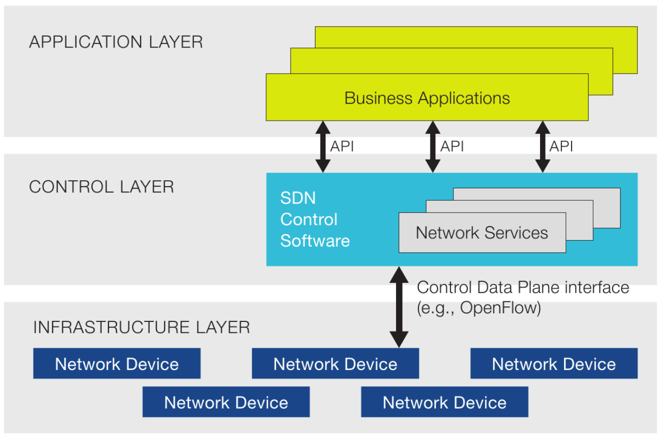


Figure 3.15: SDN architecture overview. [10]

nicates the controller with the network elements, is an API called OpenFlow. Others exist but OpenFlow is the standard for a lot of vendors and in open source projects. For the northbound interface (NBI), connecting the control and application layer, there is no standard API and usually each controller has its own implementation [11], for instance, OpenDaylight uses the OpenDaylight API.

3.3.3 Advantages & Disadvantages.

Advantages

The implementation of SDN can be difficult but it comes with important advantages:

- **Agility and speed.** Setting up an SDN can be as easy as configuring VMs. This makes them a much better candidate than traditional networks when working with VM-heavy environments.
- **Network flexibility.** The holistic network view of SDN means that experimenting with configurations does not have permanent consequences. You can always revert to a previous network configuration with ease.
- **Granular configuration and security.** The flexibility of SDN makes it able to implement security policies in a fine grained way. Making things like new VM or new devices easy to secure.
- **More efficiency & less Operating Expenditures.** The ease of management that comes with SDN means less down-times due to maintenance. And while changing all the devices on a network to be OpenFlow compliant is a big upfront investment, the long term benefits are important.

Disadvantages

Every new technology also comes with new problems and rough edges that need to be polished:

- **Centralized architecture.**This is both an advantage and a problem, the separation of control planes brings a new problem in security. SDN have a vector of attack that was not present in previous networks and could be potentially exploited in yet unknown ways.
- **Non-vendor support.** There still is a lack of code and, in some implementations, a lack of features. This is a temporary disadvantage that will be resolved with time and further technology adoption.
- **New network paradigm.** Network engineers are not programmers and programmers are not network engineers. SDN lives in the middle ground between this two jobs and professionals will need time to adapt to the challenge.

Chapter 4

Tools & Methodology

This chapter is composed of an overview of the tools used in this project, the test environment used and how the convergence times are measured.

4.1 Tools

4.1.1 GNS3

GNS3 is a software which allows the user to configure, emulate, test and experiment with computer networks. GNS3 can be used with physical devices or can be emulated devices. In its origin it was only compatible with CISCO but it has added compatibility with many other vendors.

GNS3 has two components; the GUI and the emulation virtual machine.

The GUI(Graphical User Interface) is where the user configures the network devices and topology. It uses a drag & drop system to add devices in the topology and other tools for connecting devices, adding labels,etc. In figure a network with emulated switches and VMs is shown.

In the last figure we can see a Firefox client running on a tinyLinux docker container, an ubuntu docker container and 4 OpenvSwitch instances. As we can see, with GNS3 you are able to deploy VMs and test network with simplicity. You can also use the GNS3 marketplace to download new appliances for your networks.

The second component of GNS3 is the virtual machine. All the compo-

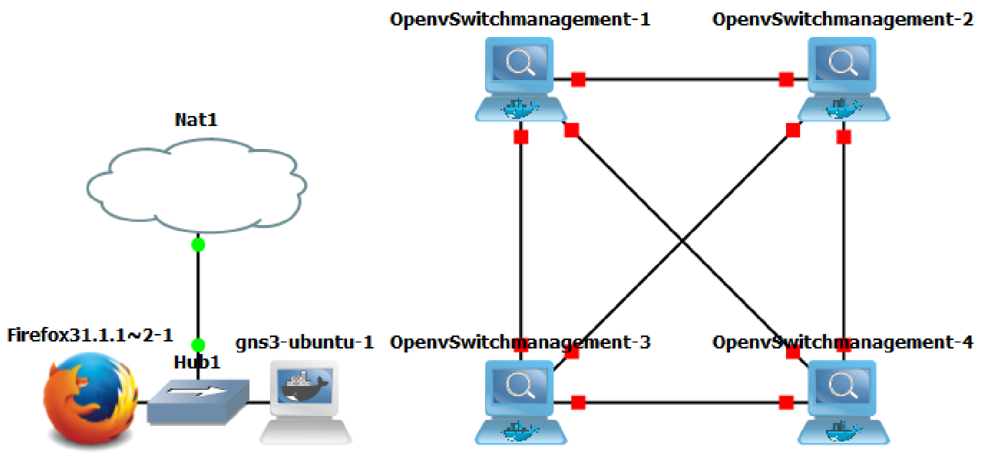
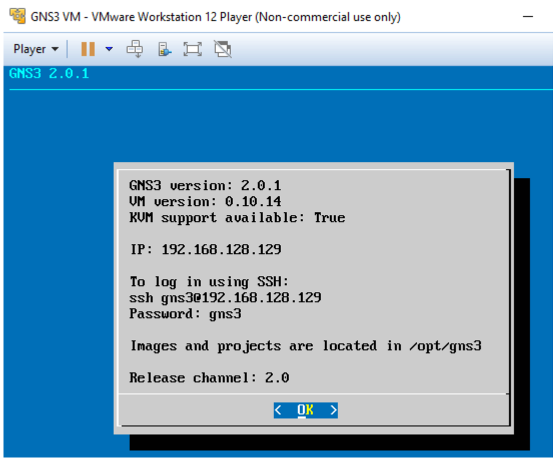


Figure 4.1: GNS3 Graphical User Interface

nents of a network, including the network devices and other appliances are run as a separate process within a virtual machine. This virtual machine can be a local server running in the same computer as GNS3 or a separate virtual machine running in a virtualization software as VMWare or VirtualBox or even in a remote machine. It is recommended to use a separate virtual machine to run GNS3 simulated devices because you can limit the resources allocated to them and are less prone to fail.

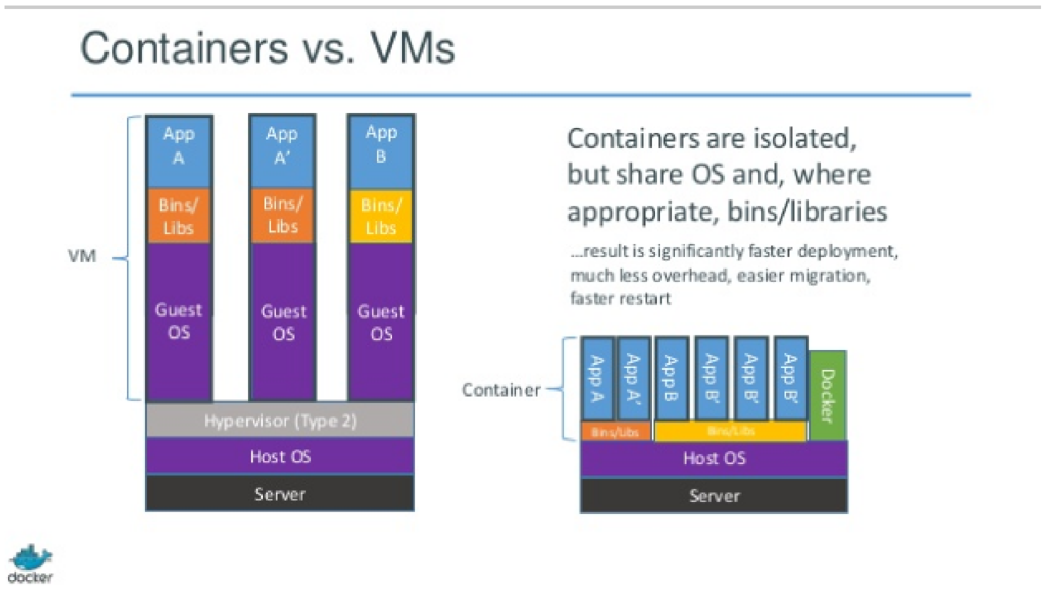

GNS3 VM - VMware Workstation 12 Player (Non-commercial use only)
Player
GNS3 2.0.1

```
GNS3 version: 2.0.1
VM version: 0.10.14
KVM support available: True

IP: 192.168.128.129

To log in using SSH:
ssh gns3@192.168.128.129
Password: gns3

Images and projects are located in /opt/gns3

Release channel: 2.0
```

Figure 4.2: GNS3 VM

4.1.2 Docker

Docker gives the user the ability to package and deploy an application in an isolated environment called a container. This isolation allows you to deploy many of this containers in a host with security. Containers are lightweight since they don't need an hypervisor and guest OS to run, they are executed directly within the host. This means that you could run more docker container than virtual machines in a given hardware (Figure 4.3).


Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart

Figure 4.3: Docker vs traditional VMs [12]

Docker advantages

Fast delivery of applications. Making changes to a deployed applications is as easy as updating the image the docker container is running. This simplifies the deployment process in your own projects or in the customer machines or cloud.

Lightweight and scalability. Docker containers are lightweight and can run on a laptop, containers don't waste resources running guest OS. Since they are easy and fast to set-up you can add or remove containers to tackle the needs of the project.

4.1.3 OpenDaylight

The OpenDaylight Project is a collaborative open source project hosted by The Linux Foundation. The goal of the project is to promote software-defined networking (SDN) and network functions virtualization (NFV). The software is written in the Java programming language. - Wikipedia.

OpenDaylight was created by several networking companies to develop an open virtual networking platform on top of existing standards as OpenFlow. The approach taken for this project resembles what happened with OpenStack, where companies come together to develop an open source project together instead of pushing their own vendor specific solutions.

The project encompass several components as an SDN Controller, protocol add-ons, and the API that make everything communicate and work together. This combination makes a complete SDN solution with a controller managing the data flows of the underlying infrastructure.

The open source nature of the project has encouraged vendors to build on top of it. For example, Cisco has an open source application OpenFlow Manager (OFM) [14] that enables users to visualize, program and monitor OpenFlow topologies and paths.

The first release of OpenDaylight saw the light on February 2014. It has been in continuous development since then enjoying a one or two of major updates every year.

4.1.4 OpenFlow

As previously explained in the SDN Architecture Section, SDN enabled devices separate the data plane and control plane of the classical network devices. However, detaching this planes means that we need a solution to communicate this two planes.

If the control plane, resides in the controller and the data plane resides on the switch, OpenFlow is the protocol that enables this two devices to communicate. This permits the abstraction of the physical and logical configurations and to develop solutions that work with independence of the equipment manufacturer. These characteristics open a faster path for network innovation and new the deployment of new services.

"*OpenFlow® is the first standard communications interface defined between the control and forwarding layers of an SDN architecture. OpenFlow® allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based).*" -Open Networking Foundation.

OpenFlow is maintained by the Open Networking Foundation (ONF) but was developed at the Stanford university[13] before the ONF was founded.

4.1.5 Open vSwitch

Open vSwitch is an open source virtual switch used typically in virtual environments to interconnect virtual machines or different hosts through a network. It is compatible with the OpenFlow protocol which makes it a good candidate when deploying SDN solutions.Open vSwitch is a production quality virtual switch and one of the most popular implementations of OpenFlow.

Open v Switch includes features as:

- VLAN tagging.
- 802.1q trunking protocol.
- Spanning Tree Protocol (STP).
- Rapid Spanning Tree Protocol (RSTP).
- 802.3ad Link Aggregation Control Protocol.
- Tunneling (IPSec, GRE, VXLAN).
- QoS administration.
- ...and more.

These included features make it a perfect candidate for our test environment, since it supports the two main layer 2 convergence protocols that are going to be tested.

OpenvSwitch is composed of three principal components, the *ovsdb-server*, the *ovs-vswitchd* and the kernel module [16] . *ovsdb-server* is lightweight database where the switch configuration and status is stored. *ovs-vswitchd* is the daemon that executes the functionality of the switch.

This two components are interconnected to enable the daemon to write and read information from the database. The information stored in the database is organized in tables, some tables are:

- Open vSwitch configuration.
- Bridge configuration.
- Port configuration.
- Interface configuration.

- OpenFlow configuration.
- QoS configuration.
- QoS output queue.

...

4.2 Methodology

4.2.1 Test Environment Overview.

For this project we are going to simulate a network within GNS3 making use of all the tools reviewed in the previous section of this document. We will use GNS3 as a graphical user interface to build our data center architecture.

The fat-tree architecture has been chosen as our test environment topology for the following reasons:

- Extensively used in real-world data center architectures[5].
- Big enough number of switches to test the simulation capabilities of GNS3.
- Tree topology with several recursive paths to do real performance test of the convergence algorithms.
- Good hardware redundancy in the Core and Distribution layers[4].
- Ease of scaling.
- Comprehensive visual representation.

GNS3 hosts only support one network interface.Due to this limitations on GNS3 hosts, recursive topologies with alternative host to host paths have been discarded.

For this environment we will be using a total of **20** 4-port *Open vSwitch* switches simulated inside *Docker* containers.

This switches have an additional management interface in a different adapter, this management interfaces will be connected to a couple of general purpose switches to communicate with *OpenDaylight*, the chosen SDN controller.

The environment will also include a *TinyCore Linux* distribution with *Firefox* installed, also simulated with Docker. This appliance will be used to access *Karaf* which is the OpenDayLight WebUI used. Karaf will enable us to access a topological view of the whole network and to inspect the flows of every node.

Lastly a NAT cloud appliance connected to TinyCore Linux, OpenDaylight and the management interfaces will be used to assign usable IP addresses for the management network.

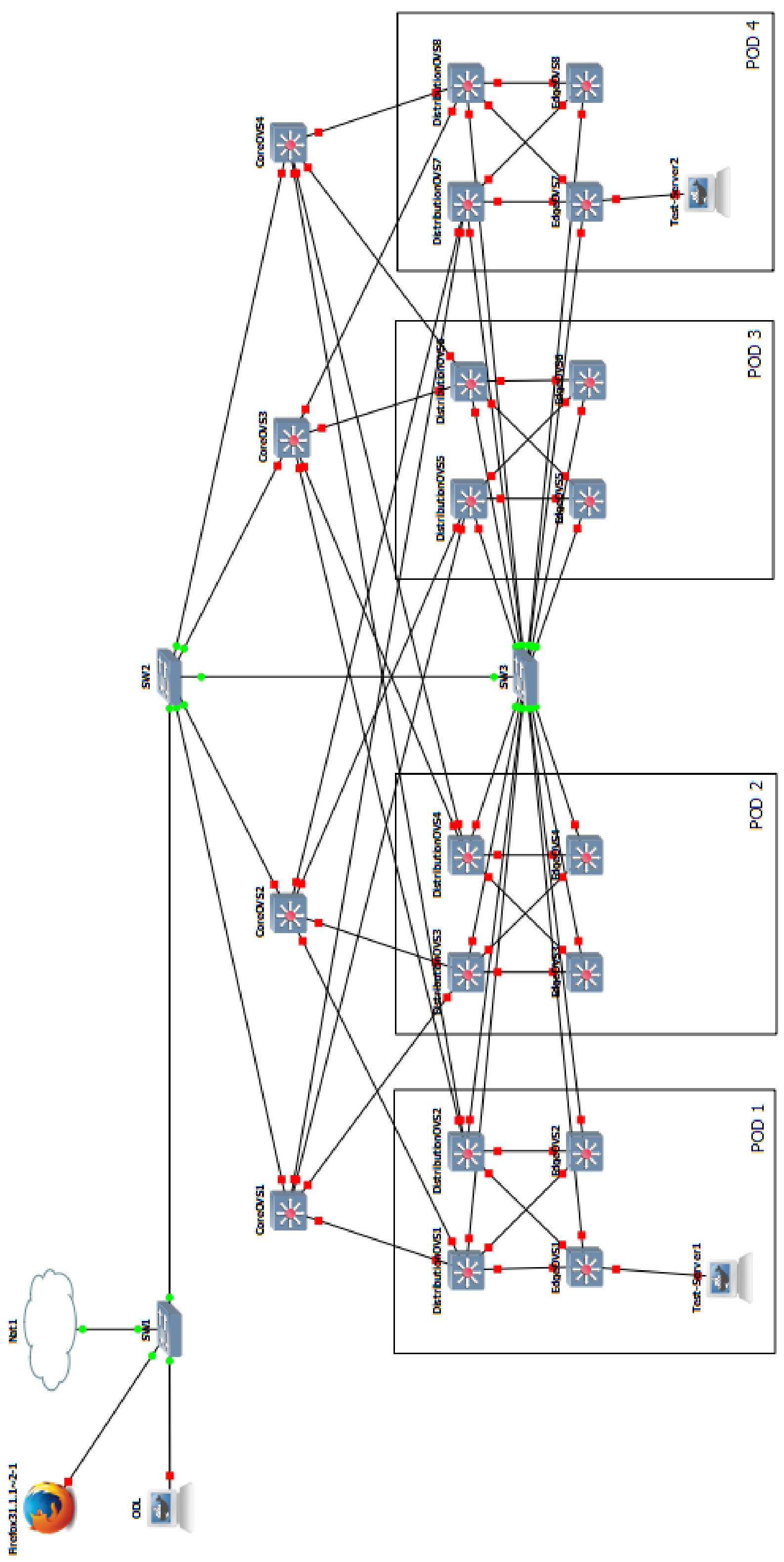


Figure 4.4: Test environment

As stated earlier one of the advantages of this architecture is its scalability. Just by adjusting the number of ports available on each switch we are able to increase the potential number of servers of the data center. For our 4-port switches we can have a maximum of 16 servers, but if we had 16-port switches the potential number of servers increases to 1024 (Table 4.1).

Table 4.1: NUMBER OF SERVERS for Tree-based topologies

n-ports	Basic Tree	Fat-tree	Clos network
4	64	16	8
6	216	54	36
8	512	128	96
16	4096	1024	896
48	110592	27648	24496

Of course increasing the number of servers means that more switches will be needed, e.g for a 8-port switches Fat-tree topology 80 switches are necessary. The number of switches on tree topologies can be calculated with the formulas in Table4.2) .

Table 4.2: NUMBER OF SWITCHES for Tree-based topologies.[4]

	Basic Tree	Fat-tree	Clos network
n of Switches	$\frac{n^2+n+1}{n^3}N$	$\frac{5N}{n}$	$\frac{3}{2}n + \frac{n^2}{4}$

In table 4.2 N is the number of servers running within the data center.

In total this test environment will be composed of:

- Docker container: Ubuntu + OpenDayLight + Apache Karaf +Dlux
- Docker container: TinyCore Linux with Firefox to access Karaf.
- 3 GNS3 general purpose switches.
- 20 Open vSwitches also running in Docker containers.
- 2 Docker container: Ubuntu + nettools to test the connectivity and convergence time of the network.
- 1 GNS3 NAT cloud.

A total of 24 Docker containers will be running over the GNS3 Linux VM within VMWare hyper-visor with allocated resources of 4 physical cores and 8GB RAM. The "bare-metal" consist on an AMD FX-8350 8-core CPU at 4.2GHz with a total of 16GB RAM.

Configuration

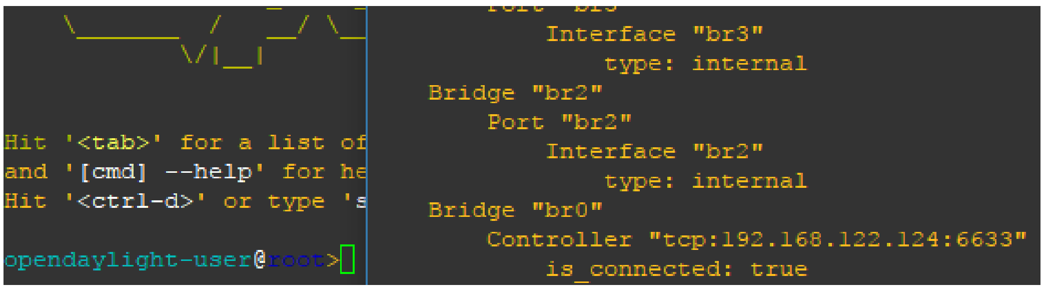
OpenDayLight is installed over an Ubuntu 14.04 container. To be able to run ODL we have to take the following steps:

- Import the Ubuntu appliance from the GNS3 market place.
- Open the file /etc/network/interfaces with *vi* or *nano* and uncomment the DHCP configuration for eth0 and restart.
- Install the Java run time environment, *unzip* and *wget*.
- Download the last available version of OpenDayLight from their repositories with *wget* and decompress it.
- Check the IP of this machine and run ./"ODL folder"/bin/karaf to start the controller.

TinyCore + Firefox is an appliance available on the GNS3 marketplace. It's ready to use just need to check if DHCP is enabled in the file /etc/network/interfaces. If it is commented uncomment and restart.

Open vSwitch is also a Docker container. In order to see them within DLux, the OpenDaylight WebUI, we need to ensure they have a proper IP address. The best way to do that is enabling DHCP in their configuration to request one from the NAT cloud device. The last step is to give them the IP address where the controller is running and executing the following OpenFlow commands:
ovs-vsctl set-controller br0 tcp:aaa.bbb.ccc.ddd:6633 - with the controller's IP
ovs-vsctl show - to check the connection.

Once it is assured that the controller can talk with the switches, we can start to build the data center architecture (Figure 4.5).



```
Hit '<tab>' for a list of
and '[cmd] --help' for he
Hit '<ctrl-d>' or type 's

opendaylight-user@root>
```

```
            Interface "br3"
                type: internal
    Bridge "br2"
        Port "br2"
            Interface "br2"
                type: internal
    Bridge "br0"
        Controller "tcp:192.168.122.124:6633"
            is_connected: true
```

Figure 4.5: Successful bridge and controller connection

One last thing to do is check if we can access DLux, the WebUI, from the Firefox client in the TinyCore container (Figure 4.6).

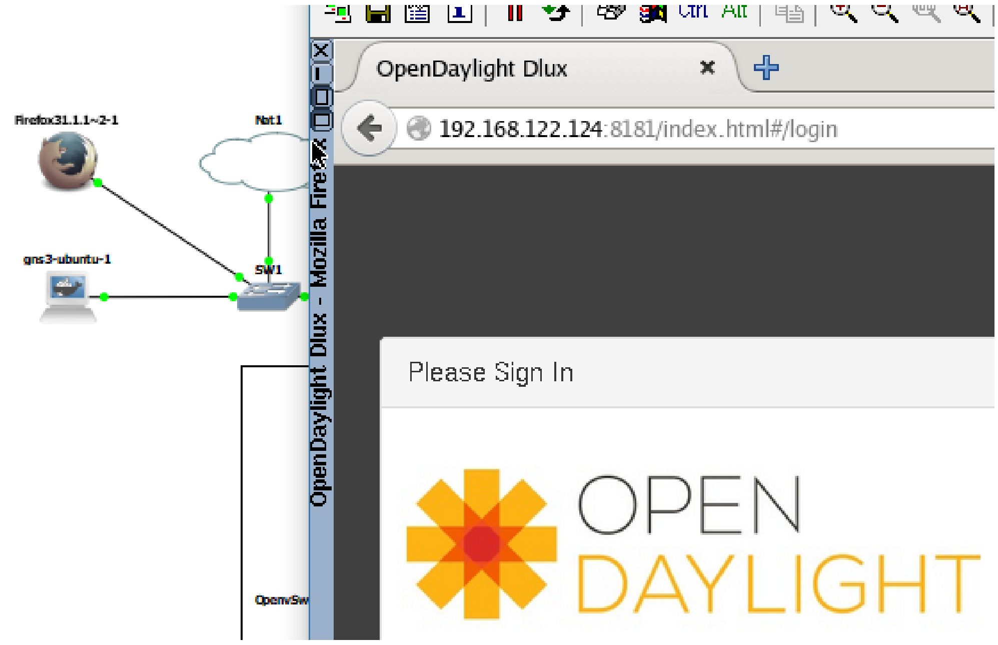


Figure 4.6: Accessing Dlux through the Firefox+TinyCore container.

Building the Fat-tree

This tree will be built using 4-port switches. A Fat-tree topology constructed with 4-port switches can have a maximum of 16 servers. Then ,per tables 4.1 and 4.2, this topology will need 20 4-port switches.

As explained in Section 3.2.1 of this document the Fat-tree will be composed of 4 core switches connected to 4 pods. A pod is a group of n/2 distribution switches connected to n/2 edge switches. This means 2 distribution switches and 2 edge switches.

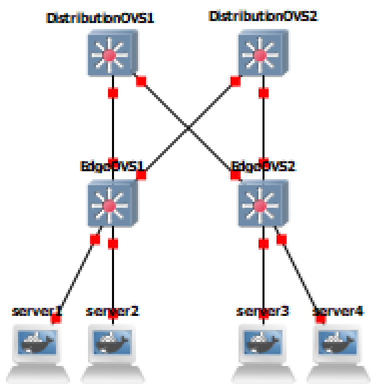


Figure 4.7: A 4-port switch pod.

Distribution 1 will connect to Core 1 and 2 and Distribution 2 to Core 3 and 4. Same connections in the remaining pods.

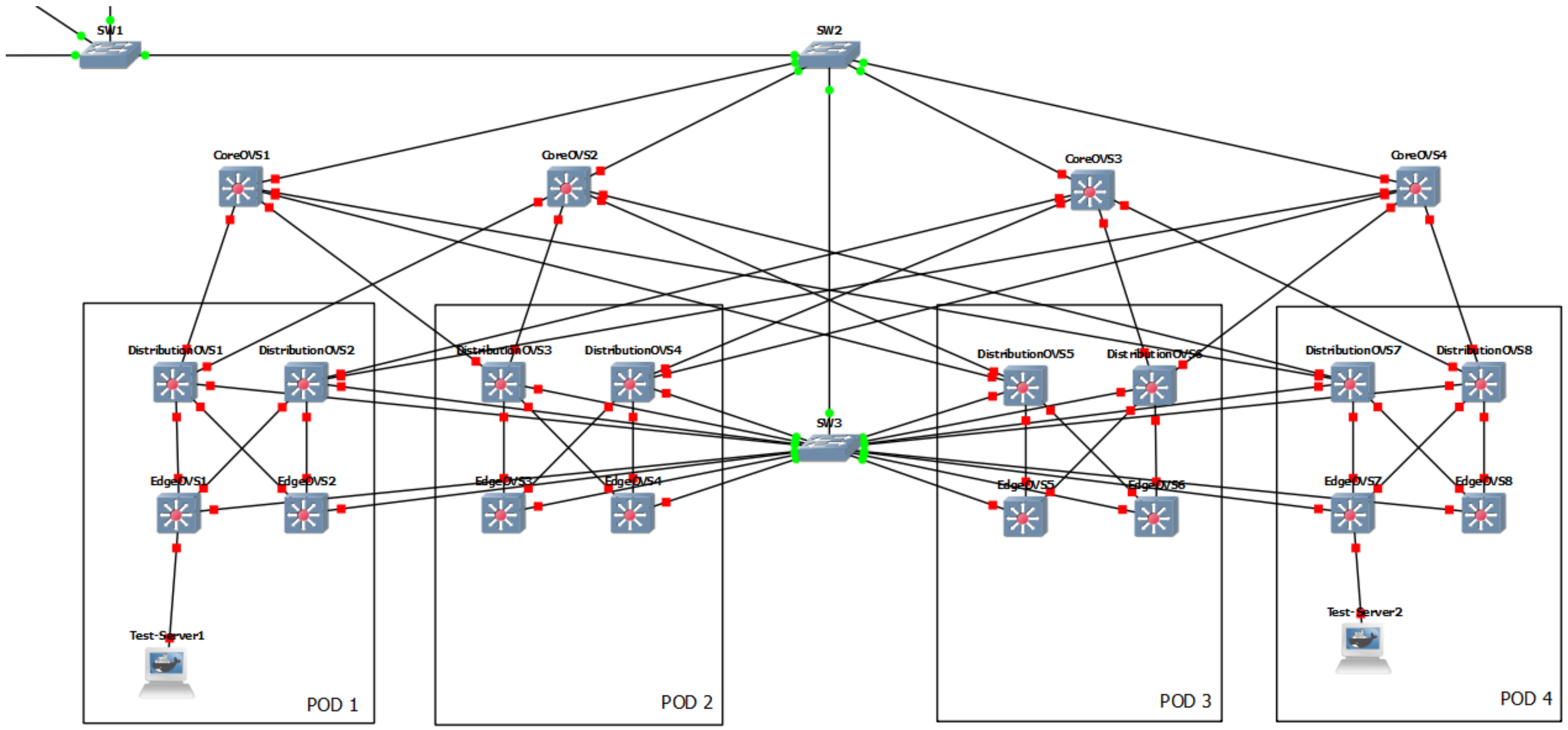


Figure 4.8: A 4-port switch Fat-tree. SW1, 2 & 3 are not part of the topology, they are for the management interfaces.

4.2.2 Convergence time measurement.

In order to measure the convergence time of the algorithms and how loss of connectivity or changes in the DCN topology affect the exchange of information we need to have a constant flow of information. This constant flow enable us to pin-point when the connectivity is lost and when it is restored. In order to perform this test two servers of the data center will exchange pings, while we will modify the architecture to simulate loss of connectivity of ports, a whole switch within the different layers of the architecture.

Since the convergence protocols react differently to topology changes (new links) and loss of connectivity both cases will be tested for.

Spanning Tree Protocol 802.1d configuration.

To test STP performance we will configure the bridges as follows:

- Core 1 Priority:0, Root bridge.
- Core 2 Priority: 4096 Alternate root bridge.
- Core 3 & 4: Priority: 8192.
- The first distribution node of each pod hasbetter priority than the rest.
- Edge bridges have the lowest priority in the network.

As explained in Chapter 3, priorities should be expressed as a product of 4096 decimal or products of 1000 if expressed in hexadecimal. In Figure 4.9 we can see the topology with its bridges priorities.

With this information STP will start to build the spanning tree sending BPDUs from the root every hello time interval, by default 2 seconds.

In Figure 4.10 the roles for each switch port is shown. We can see that every bridge has only one root port, which denotes the path with least cost to the Root bridge. To prevent loops and a potential congestion of the network some ports are blocked by STP

The logical link tree of the network can be seen in Figure 4.11.

As it can be seen the protocol maintains connectivity between all the nodes, and prevent the creation of logical loops. The nodes in grey (Figure

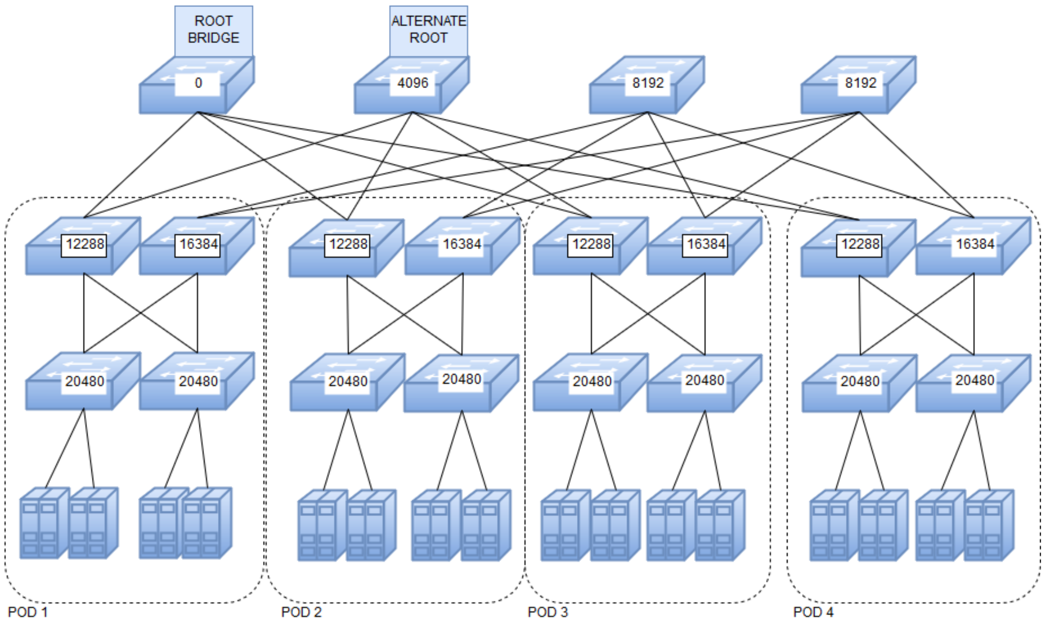


Figure 4.9: STP priorities in the test environment.

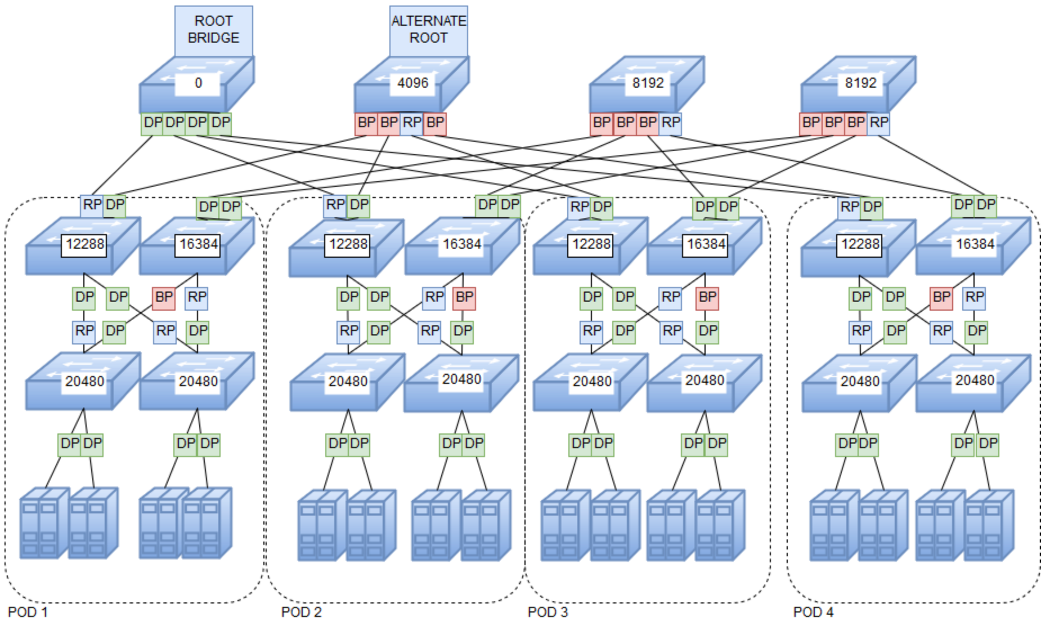


Figure 4.10: Port roles in the test environment.

4.11) are part of the STP but will act as alternate paths if the main path through C1 or one of the distribution switches is severed.

Also, we should remember that STP works in a per VLAN basis so in different VLANs different spanning trees could exist redistributing the network load in the other nodes, effectively sharing the total load of the network.

However, other thing that should be noted is that for the most part the inside traffic in a data center is made within the same pod, and pod to pod communication should be infrequent. This is mainly because the applications and deployments of a given client are usually within the same VM.

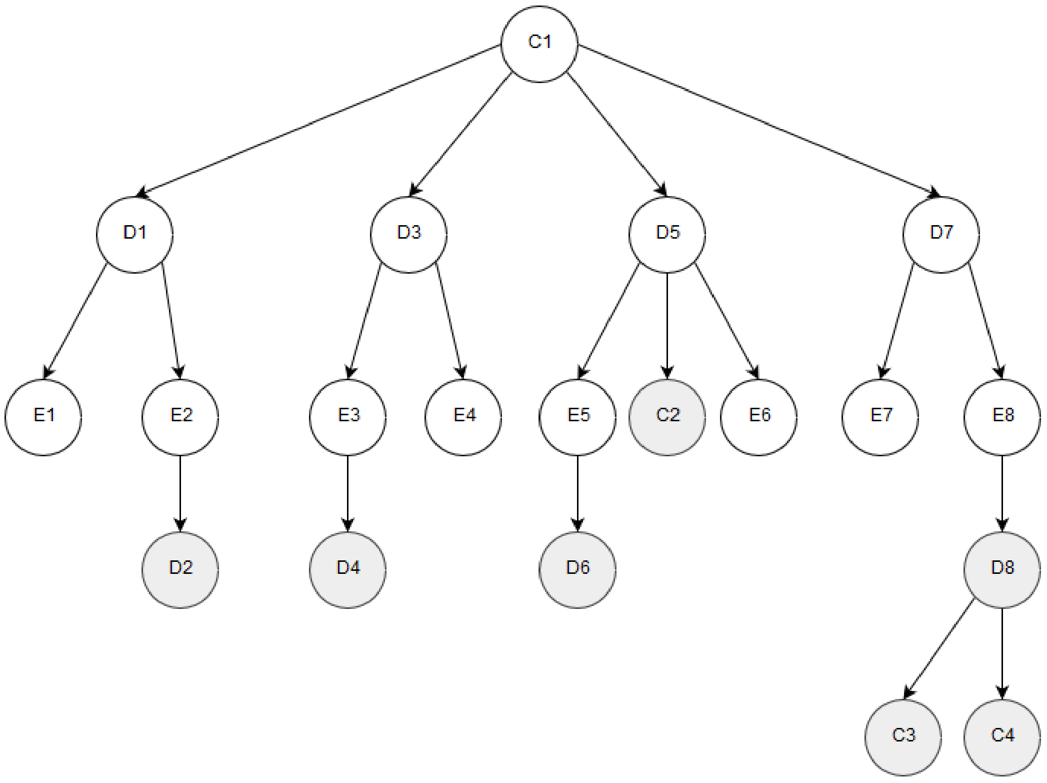


Figure 4.11: Spanning tree logical links computed.

Rapid Spanning Tree Protocol 802.1w configuration.

In order to be able to realize a valid time comparison we must maintain the priorities of every bridge. This way we will be sure that the resulting spanning trees will be similar to those used with STP.

To configure the bridges to use Rapid Spanning Tree Protocol instead of Spanning Tree Protocol we will issue ovs-ctl commands to every switch to activate RSTP. This needs to be done in a per bridge basis.

To make sure that the computed spanning tree is the same to that of STP we can take a look at the topology reported to the SDN controller in Figure 4.12.

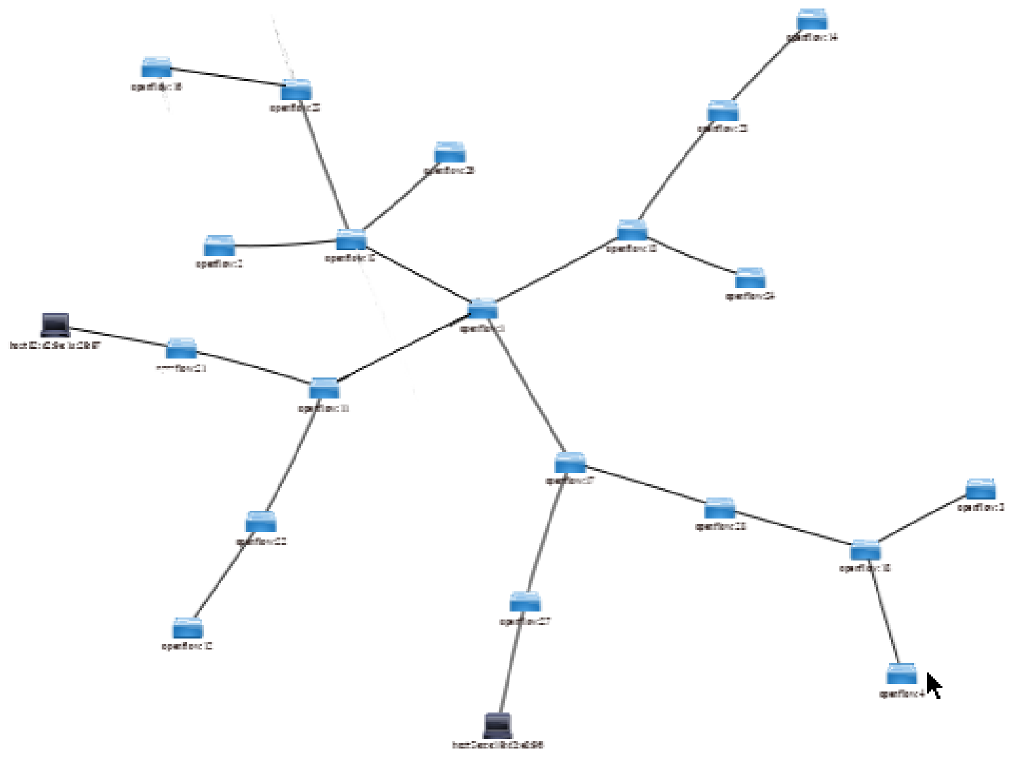


Figure 4.12: RSTP spanning tree computed as shown in OpenDaylight controller.

Times measurement

Now with the network fully functional we can test the performance of the protocols. To do that, we need to be able to identify when the connectivity is lost and when it is restored. Restoring the connectivity means that the tree has been rebuilt and the network is able to function as usual.

In order to be as exact as possible we will measure in parallel with two different methods:

First method. Using the ping command a constant flow of packets can be achieved using the -f (flood) modifier and using the -i (interval) option we can specify the time interval between packets. Also using the -c (count) option we can specify the number of packets that should be sent, e.g: *ping -c 6000 -f -i 0.01 192.168.123.3* with this command this host will send 6000 pings in 0.01s interval without waiting for a response to send the next ping. This means 60 seconds sending a constant flow of packets at known intervals. If we measure the packet loss we can deduce the time the network wasn't functioning properly.

Second method. The second method consist in using WireShark in the receiving end to compare the time stamps of the incoming ping request. And see when the connection is restored.

Test cases. We will do several test cases. Loss of connectivity (port), loss of connectivity (whole bridge), new link(port), new link(bridge). This cases will be done in both the Core Layer and the Distribution layer of the DCN. This amounts to a total of 8 test cases for each protocol

Number of tests. For each test case 10 measurements will be produced using each one of the described methods. The results of this tests are shown in the next chapter.

Chapter 5

Results & Discussion

In this chapter the results from the measurements for every test case previously mentioned are presented. After reviewing the test results a brief discussion and explanation of the numbers is presented.

5.1 Topology after changes

For the first round of measurements we will force the shutdown of the link that connects D1(11) and E1(21). Forcing the convergence protocols to reconverge to adapt to this change. The resulting topology is as shown in Figure 5.1.

For the second test D1(11) will be shutdown entirely. The topology in this case will remain as seen in Figure 5.2

For the next test the link that connects D1(11) and C1(1) will be cut. The topology in this case will converge in the spanning treeshown in Figure 5.3.

The last test will be done shutting down Core Bridge 1 completely. This will force a new spanning tree that will use the alternate root bridge Core 2 as its root bridge. The spanning tree can be seen in Figure 5.4.

These topology changes can be easily seen and detected in the OpenDaylight GUI. As it is constantly getting updates of the flow tables from the OpenFlow enabled switches.

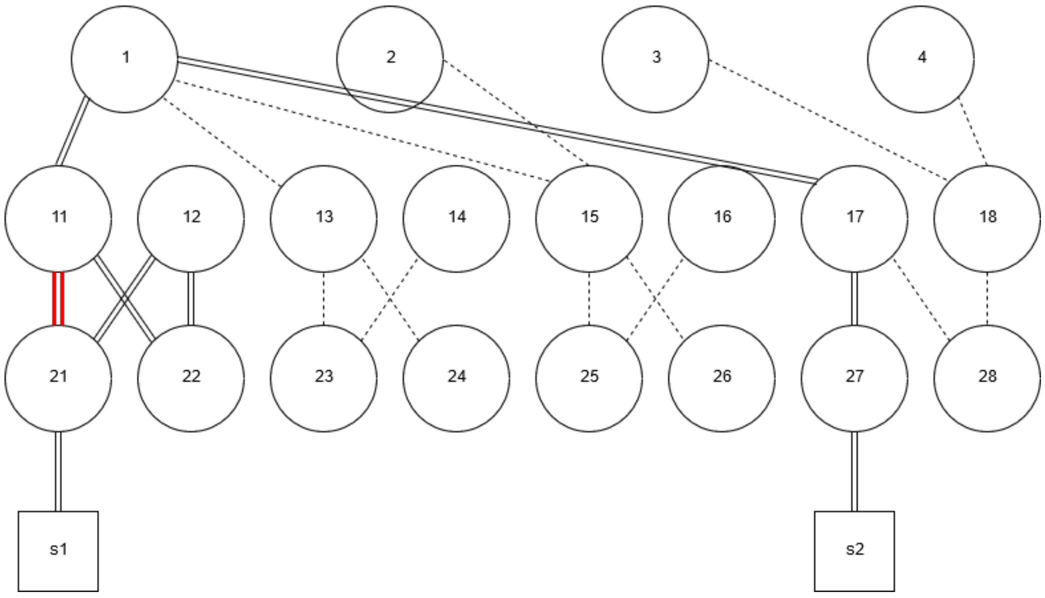


Figure 5.1: Spanning tree after shutdown of port connecting D1(11) and E1(21).

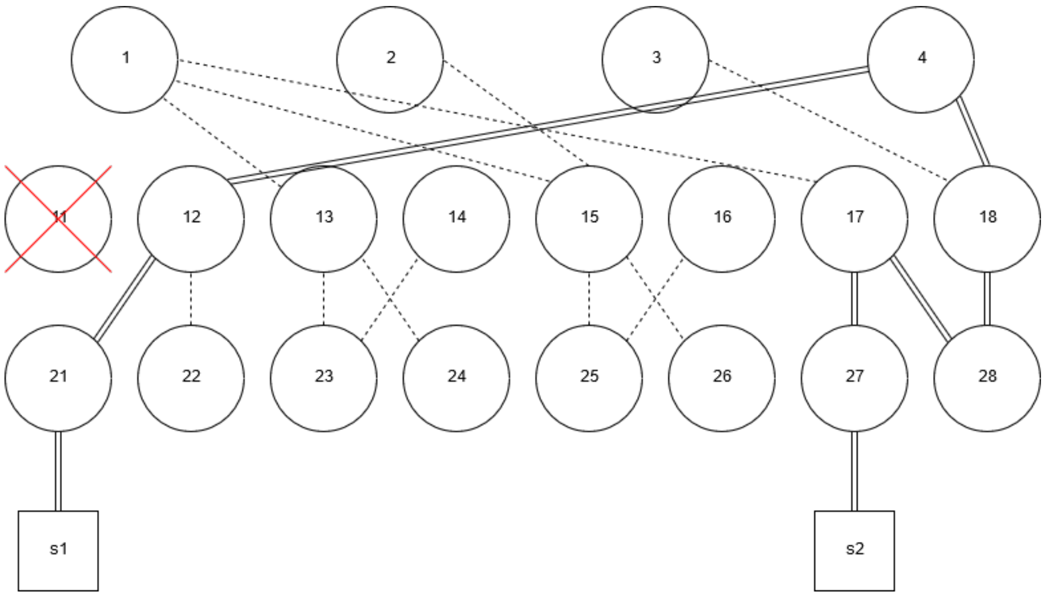


Figure 5.2: Spanning tree after shutdown of bridge D1(11).

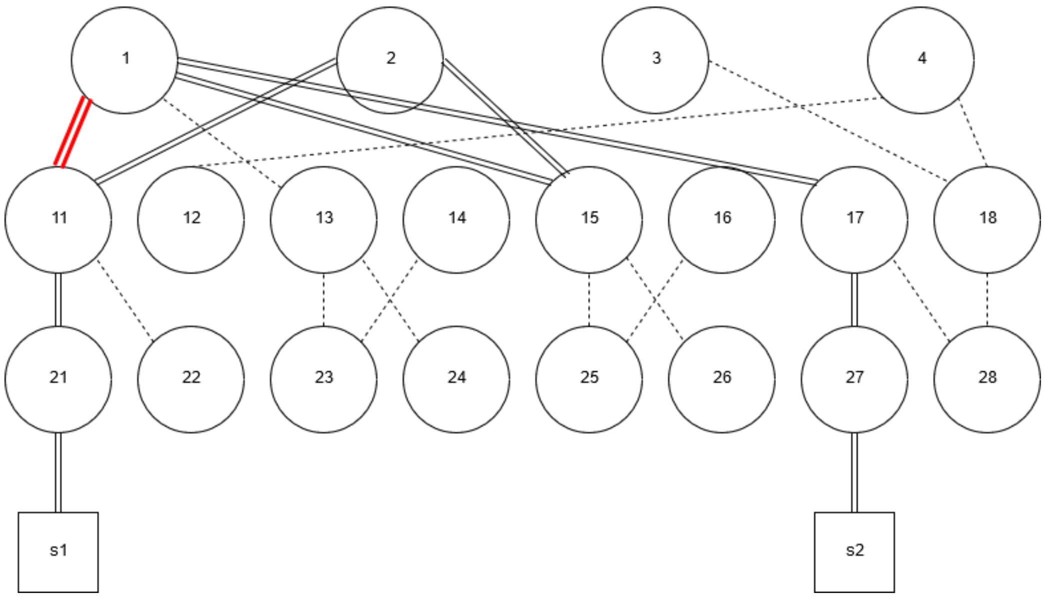


Figure 5.3: Spanning tree after shutdown of port connecting D1(11) and E1(1).

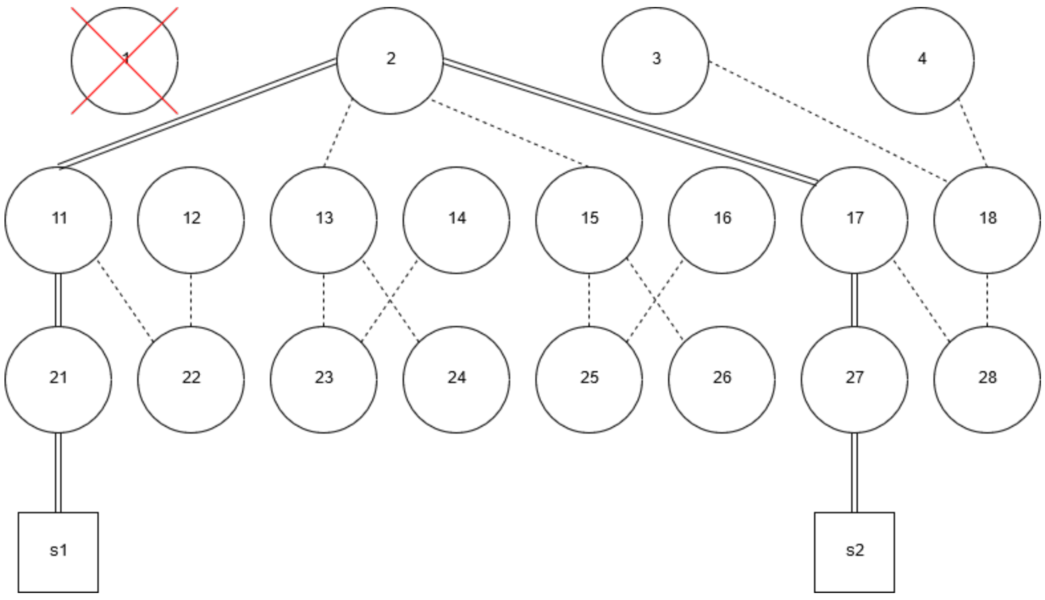


Figure 5.4: Spanning tree after shutdown of bridge C1(1).

5.2 STP Convergence Time.

This section will show all the convergence time test results obtained when using STP as the convergence protocol. Each test has its own subsection and the convergence times are presented in table format.

5.2.1 STP Connectivity loss on a port. Distribution layer.

Test was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.1).

The connectivity loss on a port was forced on interface *eth3* of switch D1 (Distribution 1) which is connected to E1 and therefore to server 2.

Table 5.1: Convergence times after connectivity loss on a port (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	36,056	40,1282	38,0921
2	42,664	46,1923	44,42815
3	40,801	47,6075	44.20425
4	34,437	39,5674	37,0022
5	37,532	42,4875	40,00975
6	35,512	41,8246	38,6683
7	39,186	46,4164	42,8012
8	38,154	44,9346	41,5443
9	37,764	41,7613	39,76265
10	35,941	40,0681	38,00455
Average	37,8047	43,09879	40,451745

We can observe the ”homogeneity” of the results obtained. These results coincide with the sum of timers that STP uses to communicate changes in its topology.

5.2.2 STP Connectivity restored on a port. Distribution layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7(Table 5.2).

Restored connectivity on interface *eth3* of switch **D1 (Distribution 1)** which is connected to E1 and therefore to server 2. This port is part of the original least cost path to the root, so the spanning-tree will configure itself again to use this port.

Table 5.2: Convergence times after restoring connectivity on a port (ms)

test n	**Method 1 (ms)**	**Method 2 (ms)**	**Mean (ms)**
1	885	1045	965
2	1837	2050	1943.5
3	1812	1988	1900
4	1613	1650	1631.5
5	921	1004	962.5
6	1536	1638	1587
7	1749	1833	1791
8	1234	1301	1267.5
9	1811	1894	1852.5
10	984	1056	1020
Average	1438,2	1545,9	1492.05

The moment when the original port goes back again, D1 switches the communication from the alternate path to the original path and communicates that topology change to E1 in the next Hello Message which are sent every 2 seconds.

5.2.3 STP Connectivity loss on a bridge. Distribution layer.

Test was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7(Table 5.3).

The connectivity loss was forced on switch D1 (Distribution 1) which is connected to E1 and therefore to server 2.

Table 5.3: Convergence times after full connectivity loss on a bridge (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	47,359	48,1172	47,738
2	42,712	50,5319	46,622
3	42,165	48,5577	45,361
4	42,246	48,6006	45,423
5	46,084	48,6590	47,371
6	45,615	48,1480	46,881
7	45,125	49,3498	47,237
8	43,457	47,9856	45,721
9	44,965	50,1248	47,545
10	46,471	48,6423	47,557
Average	48,8716	44,6198	46,74575

In this test case the topology change is much more severe and it takes a lot more of communication within the nodes to achieve convergence. Even then the 30 seconds from Learning + Forwarding time are the baseline for this times and then the accumulation of Hello Times to communicate the topology change through the bridges.

5.2.4 STP Connectivity restored on a bridge. Distribution layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.4).

Restored connectivity on interface *eth3* of switch D1 (Distribution 1) which is connected to E1 and therefore to server 2. This switch was part of the original spanning tree and will be restored as a bridge once it is booted.

Table 5.4: Convergence times after restoring a bridge (s)

test n	**Method 1 (s)**	**Method 2 (ss)**	**Mean (s)**
1	27,023	29,4561	28,239
2	28,977	29,7995	29,3883
3	28,443	30,3087	29,3757
4	25,930	29,5714	27,7508
5	28,861	29,8029	29,3320
6	27,847	29,7877	28,8173
7	28,012	29,8540	28,9328
8	27,819	30,1423	28,9804
9	27,694	29,8317	28,7627
10	28,946	30,0975	29,0719
Average	27,8651	29,86517	28,86517

It this case the time needed to restore connectivity through bridge D1 is roughly the sum of the learning time and forwarding time, during those times the bridge stop forwarding packets to give time to the bridges in the topology to receive the new spanning tree.

5.2.5 STP Connectivity loss on a port. Core layer.

Test was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.5).

The connectivity loss on a port was forced on interface *eth1* of switch C1 (Core 1) which is connected to D1 and therefore to E1 and server 2. This switch is also the root bridge of the topology.

Table 5.5: Convergence times after connectivity loss on a port in the Core layer(s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	44,337	49,5083	46,9206
2	42,924	50,0389	46,4813
3	47,027	50,3183	48,6728
4	47,574	49,8047	49,6891
5	47,699	49,9268	48,8130
6	46,911	49,7534	48,3322
7	45,462	50,2276	47,8448
8	43,853	49,9562	46,9046
9	44,967	50,1037	47,5354
10	46,742	49,7495	48,2458
Average	45,7496	49,9382	47,8440

Those results are similar to those obtained in the distribution layer. The connectivity loss on a port forces bridges to go through the Learning and Forwarding Timers and on top of that the time that the topology changes takes to spread to all bridges.

5.2.6 STP Connectivity restored on a port. Core layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7(Table 5.6).

Restored connectivity on interface *eth1* of switch **C1 (Core 1)** which is connected to D1 connected to E1 and therefore to server 2. This is the root bridge so the port will necessary be a designated port connected to D1 and the port on D1 will necessarily be a Root port, because is connected directly to the root bridge.

Table 5.6: Convergence times after restoring connectivity on a port in the Core layer(ms)

test n	**Method 1 (ms)**	**Method 2 (ms)**	**Mean (ms)**
1	2022	2171	2096,7
2	1011	1077	1044,1
3	808	1049	928,6
4	1213	1373	1293,1
5	1010	1175	1093,1
6	1213	1369	1291,1
7	1051	1209	1130
8	1109	1235	1147,2
9	1088	1272	1190,9
10	1059	1252	1170,5
Average	1158	1318,7	1238,5

As before restoring the port loss on a bridge does not provoke a significant loss of packets. It will take at most the time between Hello Times (2s) to revert connectivity through the original port.

5.2.7 STP Connectivity loss on a bridge. Core layer.

Test was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7(Table 5.7).

The connectivity loss was forced on switch C1 (Core 1) which is connected to D1 connected to E1 and therefore to server 2.

Table 5.7: Convergence times after full connectivity loss on a bridge in the core layer (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	45,963	49,4509	47,7071
2	45,844	48,3762	47,1101
3	44,438	51,2099	47,8237
4	47,116	49,4129	48,2646
5	47,047	49,5571	48,3022
6	46,082	49,6014	47,8415
7	46,105	49,6315	47,8684
8	46,158	49,8826	48,0201
9	46,502	49,6171	48,0594
10	46,379	49,6579	48,0183
Average	46,163	49,6398	47,9015

We can observe that the results are similar when a bridge is loss on the distribution layer or the core layer. A base time of 30 seconds and the time needed for the Hello messages to be sent to every bridge from the topology.

5.2.8 STP Connectivity restored on a bridge. Core layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.8).

Restored full connectivity on switch C1 (Core 1) which is connected to D1 connected to E1 and therefore connected to server 2. This switch was part of the original spanning tree and will be restored as a root bridge.

Table 5.8: Convergence times after restoring a bridge in the core layer (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	27,964	28,8285	28,3965
2	26,509	29,0819	27,7956
3	28,760	29,5747	29,1675
4	27,412	28,3321	27,8723
5	27,897	28,8211	28,3592
6	27,709	28,9277	28,3182
7	27,658	28,9475	28,3026
8	27,887	28,9206	28,4040
9	27,713	28,7898	28,2513
10	27,773	28,8813	28,3270
Average	27,728	28,9105	28,3194

Once again we can not observe differences with the results obtained with the test in the distribution layer. This times are approximately the learning times + forwarding times needed for the STP protocol to converge 30s.

5.3 RSTP Convergence Time.

5.3.1 RSTP Connectivity loss on a port. Distribution layer.

As with the STP tests this was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.9).

To force the connectivity lost of the port interface *eth3* of switch D1 (Distribution 1) was pulled down. This interface is directly connected to E1 and was part of the original least cost path.

Table 5.9: RSTP Convergence times after connectivity loss on a port (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	5,262	5,6290	5,4455
2	3,242	3,5040	3,3728
3	4,872	5,2159	5,0437
4	3,316	4,0033	3,6598
5	3,508	4,4363	3,9722
6	4,040	4,5577	4,2988
7	3,796	4,3434	4,0695
8	3,906	4,5113	4,2088
9	3,713	4,3704	4,0418
10	3,793	4,4438	4,1182
Average	3,945	4,5015	4,2231

In this case D2(12) has to unblock its link with E1(21) this is done in roughly two hello times, 4 seconds, since the bridges are able to craft their own BPDU(STP only relays the BPDU originating from the root) and communicate the topology change much faster.

5.3.2 RSTP Connectivity restored on a port. Distribution layer.

As before this test was done by checking connectivity of the servers in 1ms intervals. These servers are connected to E1 (Edge 1) and E7(Table 5.10).

The connectivity restoration was done by pulling up *eth3* of switch **D1 (Distribution 1)** which is connected to E1 and therefore to server 2. This port is part of the original least cost path to the root, so the spanning-tree will configure itself again to use this port.

Table 5.10: Convergence times after restoring connectivity on a port (ms)

test n	**Method 1 (ms)**	**Method 2 (ms)**	**Mean (ms)**
1	1,006	25,4372	11,7875
2	1,007	22,5680	11,7877
3	1,005	27,8730	14,4389
4	1,008	34,2600	17,6339
5	1,589	26,8470	14,2180
6	1,123	22,3096	11,7163
7	1,146	26,7715	13,9590
8	1,174	27,6122	14,3932
9	1,208	27,5601	14,3841
10	1,248	26,2201	13,7341
Average	1,151	24,2021	12,6768

During this test it was found that checking with method 1 (replies to the ping requests) gave us a constant of **1 lost packet per test**. Since the actual time of the test is not exactly 60 seconds the numbers differ a bit from each other. Meanwhile checking the time stamps of the ping requests in WireShark gave broadly different values but, as with method 1, only 1 ping request/reply was lost in the process. This may be caused by packets bouncing on loops before the network achieves real convergence.

In any case, we should note that convergence is achieved in less than 1 second and with minimal packet loss.

5.3.3 RSTP Connectivity loss on a bridge. Distribution layer.

Test done checking connectivity between server 2 and 17. Servers are connected to E1 (Edge 1) and E7(Table 5.11).

Switch D1 (Distribution 1) was brought down completely, and since it was part of the least cost path this forces the network to rebuild the spanning tree.

Table 5.11: Convergence times after full connectivity loss on a bridge (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	4,358	4,9930	4,6756
2	5,011	6,0753	5,5431
3	4,934	5,9457	5,4396
4	4,642	5,7457	5,1940
5	4,093	5,0656	4,5794
6	4,608	5,5651	5,0864
7	4,658	5,6795	5,1685
8	4,587	5,6003	5,0936
9	4,517	5,5312	5,0244
10	4,493	5,4883	4,9904
Average	4,590	5,5690	5,0795

Similar numbers obtained by both methods, the results appear to resemble twice the Hello Time (2 seconds). This is the time it takes C4(4) to communicate its root path to D2(12) plus the time D2 takes to update its BPDU and send it to E1(21).

5.3.4 RSTP Connectivity restored on a bridge. Distribution layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.12).

Restored connectivity on interface *eth3* of switch D1 (Distribution 1) which is connected to E1 and therefore to server 2. This switch was part of the original spanning tree and will be restored as a bridge once it is booted.

Table 5.12: Convergence times after restoring a bridge (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	12,273	14,6915	13,4824
2	11,517	14,0444	12,7807
3	12,142	14,6955	13,4187
4	12,735	14,5547	13,6449
5	12,152	13,9669	13,0596
6	12,164	14,3906	13,2773
7	12,142	14,3304	13,2363
8	12,267	14,3876	13,3274
9	12,292	14,3260	13,3091
10	12,204	14,2803	13,2419
Average	12,189	14,3668	13,2778

In this test we can observe that the result are fairly homogeneous. This time it takes more time to the topology to converge since the change is also greater compared to the loss of a link.

5.3.5 RSTP Connectivity loss on a port. Core layer.

Test was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.13).

The connectivity loss on a port was forced on interface *eth1* of switch C1 (Core 1) which is connected to D1 and therefore to E1 and server 2. This switch is also the root bridge of the topology.

Table 5.13: Convergence times after connectivity loss on a port in the Core layer(s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	3,410	4,1920	3,8008
2	2,968	3,6828	3,3253
3	3,200	3,9303	3,5652
4	3,195	3,7276	3,4614
5	4,501	5,7087	5,1049
6	3,455	4,2483	3,8515
7	3,464	4,2595	3,8617
8	3,563	4,3749	3,9690
9	3,636	4,4638	4,0497
10	3,724	4,6110	4,1674
Average	3,512	4,3199	3,9157

In this test we obtained very similar results to those of the distribution layer. It takes roughly two Hello Times for the topology change to spread within the network and recover connectivity.

5.3.6 RSTP Connectivity restored on a port. Core layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7(Table 5.14).

Restored connectivity on interface *eth1* of switch **C1 (Core 1)** which is connected to D1 connected to E1 and therefore to server 2. This is the root bridge so the port will necessary be a designated port connected to D1 and the port on D1 will necessarily be a Root port, because is connected directly to the root bridge.

Table 5.14: Convergence times after restoring connectivity on a port in the Core layer(ms)

test n	**Method 1 (ms)**	**Method 2 (ms)**	**Mean (ms)**
1	10,037	-	-
2	70,548	-	-
3	10,043	-	-
4	40,201	-	-
5	50,480	-	-
6	30,262	-	-
7	41,507	-	-
8	30,699	-	-
9	40,830	-	-
10	40,956	-	-
Average	36,556	-	-

In this case we only show the first method (ping) results because with the second method it was not clear when the messages got through or how many packets where lost. The frames where delivered in an erratic order making it impossible to obtain the time. This may be caused by frames bouncing in temporary loops formed in the transient state of convergence.

Nevertheless we could obtain the number of packets that were lost and saw that connectivity is not severely affected by the topology change.

5.3.7 RSTP Connectivity loss on a bridge. Core layer.

Test was done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7(Table 5.15).

The connectivity loss was forced on switch C1 (Core 1) which is connected to D1 connected to E1 and therefore to server 2.

Table 5.15: Convergence times after full connectivity loss on a bridge in the core layer (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	13,761	11,4156	12,5882
2	11,910	7,5081	9,7088
3	13,666	4,8476	9,2568
4	10,879	4,3863	7,6327
5	14,628	8,6899	11,6588
6	12,969	7,3695	10,1691
7	12,810	6,5603	9,6852
8	12,990	6,3707	9,6805
9	12,855	6,6753	9,7653
10	13,250	7,1331	10,1918
Average	12,972	7,0956	10,0337

The results of this test are similar to those on the distribution layer. The disconnection of a complete bridge takes more time to be absorbed by the network. It takes around twelve seconds in total for the connectivity to be fully restored.

5.3.8 RSTP Connectivity restored on a bridge. Core layer.

Test done checking connectivity between server 2 and 17. These servers are connected to E1 (Edge 1) and E7 (Table 5.16).

Restored full connectivity on switch C1 (Core 1) which is connected to D1 connected to E1 and therefore connected to server 2. This switch was part of the original spanning tree and will be restored as a root bridge.

Table 5.16: Convergence times after restoring a bridge in the core layer (s)

test n	**Method 1 (s)**	**Method 2 (s)**	**Mean (s)**
1	12,043	14,3465	13,1948
2	12,151	14,7856	13,4682
3	11,876	14,2587	13,0672
4	12,323	14,6463	13,4845
5	11,906	14,0736	12,9898
6	12,060	14,4221	13,2409
7	12,063	14,4373	13,2501
8	12,045	14,3676	13,2065
9	12,079	14,3894	13,2344
10	12,031	14,3380	13,1843
Average	12,058	14,4065	13,2321

As with the previous test, the results are similar when the bridge is restored in the distribution layer. It takes twelve seconds to register the addition of the bridge and modify the topology to use C1 as the root bridge of the spanning tree.

Chapter 6

Conclusions

After analyzing the results obtained in the tests we can conclude that RSTP was an important advance in terms of time needed for the topology to remake the spanning tree.

We should note how fast both STP and RSTP converge when a port gets shutdown but we could argue that this scenario may be be less frequent than the loss of a full bridge.

The biggest disadvantage of those protocols maybe the loss of bandwidth that comes with blocking ports. This effect may be dampened using multiples VLAN to create multiple spanning trees. Even then, it is not possible to use multiple links for the same purpose and take advantage of the parallel capacity of the network.

This disadvantages may be the reason why layer-2 convergence protocols are steadily losing presence in today's networks. The alternatives maybe using layer-$2\frac{1}{2}$protocols like SPB (Shortest Path Bridge) or Cisco's TRILL (Transparent Interconnection of Lots of Links) to achieve convergence and even use multiple paths to take advantage of redundant links but with them comes the cost of using layer-3 enabled switches and a much more complex configuration.

Another alternative would be using layer-3 IP protocols as IGP over routers but that maybe ruled out if the data-center network architecture is built with layer-2 devices and the throughput of routers is less than those of switches.

In any case, the suitability of the usual convergence protocols for the next generation of data-center networks is heavily questioned.

The most future-proof and flexible choice would be to use software defined traffic engineering. While it is true that SDN changes the network paradigm and poses a new set of challenges it would give us the means to implement a resilient and highly available data-center network.

In addition, it would enable our network to be easily re-configured based on different business decisions like QoS, security, power management, multi-tenancy, auto-scalable applications... This may probably be the only way to achieve a network ready for the next challenges that enterprise networks will face as Hyper-Convergent Infrastructure and Private Clouds.

Bibliography

[1] Gartner, *Magic Quadrant for Data Center Networking 2017.*

[2] Cisco Support Website., *Understanding and Configuring Spanning Tree Protocol (STP) on Catalyst Switches*, Last accessed May 2017. Cisco [online].

[3] Cisco Support Website., *Understanding Rapid Spanning Tree Protocol (802.1w)*, Last accessed May 2017. Cisco [online].

[4] Liu, Y. et. al, *A Survey of Data Center Network Architectures.*

[5] M. Al-Fares, A. Loukissas, and A. Vahdat,*A scalable, commodity data center network architecture*, in Proceedings of the ACM SIGCOMM 2008 conference on Data communication.

[6] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, *DCell: A scalable and fault-tolerant network structure for data centers*, ACM SIGCOMM Computer Communication Review, vol. 38.

[7] G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, *"c-through: Part-time optics in data centers"*

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, *"Helios: a hybrid electrical/optical switch architecture for modular data centers"*

[9] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, *"Osa: An optical switching architecture for data center networks with unprecedented flexibility"*

[10] SDx Central Website, *Understanding the SDN Architecture*, Last accessed May 2017. SDx Central. [online].

[11] SDx Central Website, *SDN & NFV APIs and SDKs*, Last accessed May 2017. SDx Central. [online].

[12] dotCloud *Introduction to Docker*, November 2013. Slideshare.[online].

[13] Greene, K., *TR10: software-defined networking.* MIT Technology Review (March/April), 2009.

[14] Open Flow Manager Github, *https://github.com/CiscoDevNet/OpenDaylight-Openflow-App*

[15] Nick McKeown. et. al., *OpenFlow: Enabling Innovation in Campus Networks.* March 14, 2008

[16] B. Pfaff. et. al., *The Design and Implementation of Open vSwitch.* Ben Pfaff website. [online].