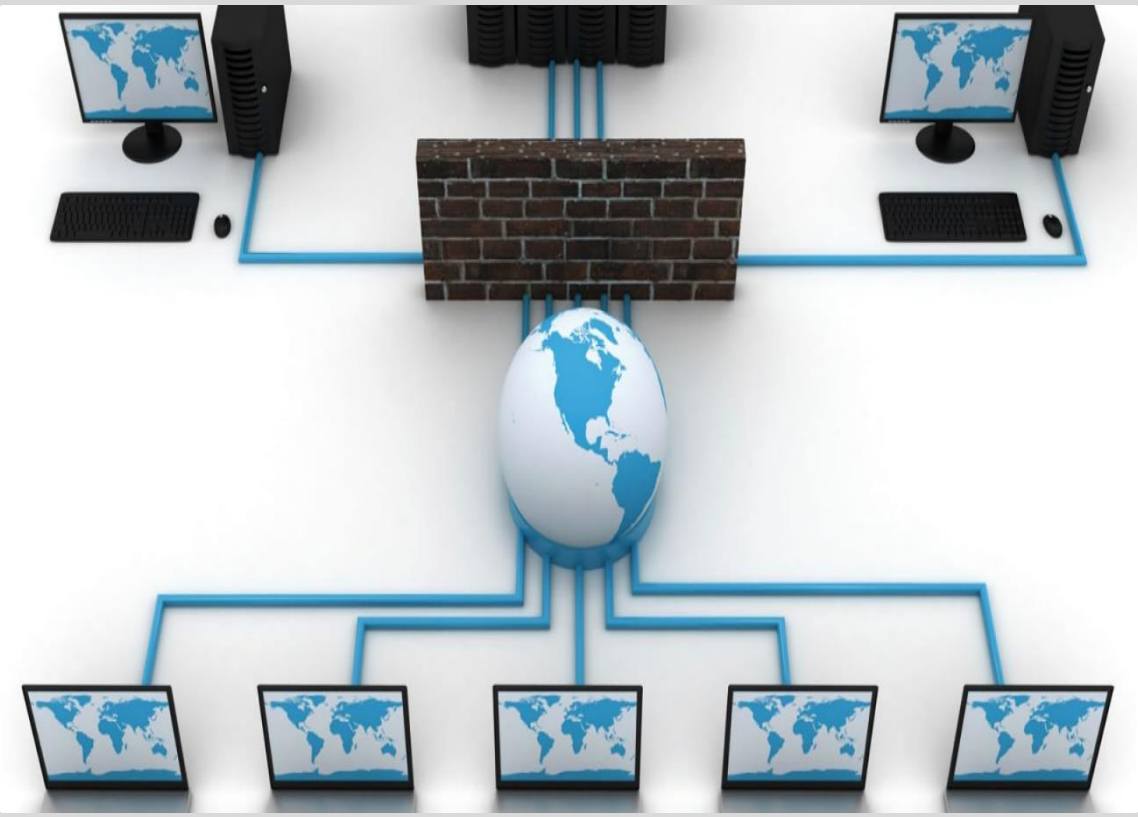
Universidad de La Laguna

Simulación de un switch Comercial sobre Open vSwitch



Kapil Ashok Melwani Chugani

Escuela Superior de Ingeniería y Tecnología

Junio, 2018

D. **Jonas Philipp Luke**, con N.I.E. X0581666-L profesor Ayudante Doctor adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como tutor.

C E R T I F I C A

Que la presente memoria titulada:

"Simulación de un switch Comercial sobre Open vSwitch"

ha sido realizada bajo su dirección por D. **Kapil Ashok Melwani Chugani**,

con N.I.F. 78725281-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de junio de 2018

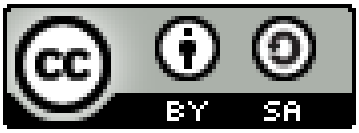
Agradecimientos

Gracias a mi familia y mis grandes amigos/as por apoyarme todos estos años, guiarme para llegar hasta aquí y por acompañarme en este duro camino en los mejores y peores momentos

Os dedico este Trabajo de Fin de Grado.

Sobre todo, a ti, gracias Mami.

Licencia



Índice de Contenido

Resumen 1
Abstract 2
Capítulo 1. Introducción 3
1.1 Objetivos 3
1.2 Motivación y antecedentes 3
1.3 Herramientas Propuestas 7
1.3.1 Python 7
1.3.2 Open vSwitch 7
1.3.3 Librerías 9
1.3.4 Simuladores 9
1.3.5 Herramientas de Empaquetado 12
1.4 Herramientas Utilizadas 15
1.5 Metodología 16
Capítulo 2. Implementación 18
2.1 Arquitectura y Funcionamiento del Sistema 18
2.2 Configuración Inicial del Programa 19
2.3 Funciones Implementadas 20
2.3.1 Comandos Cisco 20
2.3.2 Comandos Open vSwitch 25
2.3.3 Relación de Comandos Cisco a Open vSwitch 27
2.3.4 Empaquetado Docker 31
Capítulo 3. Pruebas y Resultados 33
3.1 Resultados Obtenidos 33
3.2 Dificultades Encontradas 38
Capítulo 4. Conclusiones y Trabajos Futuros 39
Chapter 4: Conclusions and Future Work 40
Referencias 41

Índice de Tablas

Tabla 1: Comparación de VirtualBox, VMware y QEMU 13
Tabla 2: Comandos Cisco Implementados 24
Tabla 3: Función ovs_vsctl_add_bridge(bridge) 25
Tabla 4: Función ovs_vsctl_del_br(bridge) 25
Tabla 5: Función iplink(port,status) 25
Tabla 6: Función ovs_vsctl_admin_port(bridge,port,tag) 26
Tabla 7: Función ovs_vsctl_add_port_trunk(port,trunk) 26
Tabla 8: Función ifconfig(ip,port) 26
Tabla 9: Función ovs_vsctl_set(table,record,column,key,value) 26
Tabla 10: Función ovs_vsctl_set_admin(port,type) 27
Tabla 11: Configuración Prueba 1 KV-SWITCH 33
Tabla 12: Configuración Prueba 2 KV-SWITCH 36

Índice de Figuras

Figura 1: Top 5 Empresas de Conmutación Ethernet [1] 4
Figura 2: Kernel Space LiSA [2] 6
Figura 3: User Space LiSA [2] 6
Figura 4: Proceso de intercambio de paquetes con OVS [3] 7
Figura 5: Componentes de OVS [4] 8
Figura 6: Esquema de Arquitectura GNS3 11
Figura 7: Esquema de Funcionamiento de Docker [3] 14
Figura 8: Comparativa de funcionamiento Container y VM [4] 14
Figura 9: Esquema de Metodología utilizada 16
Figura 10: Arquitectura KV-SWITCH 18
Figura 11: Funcionamiento Inicial de KV-SWITCH 18
Figura 12: Configuración Inicial de KV-SWITCH 19
Figura 13: Esquema de Comandos KV-SWITCH 21
Figura 14: Relación de Comandos Cisco a Open vSwitch en Creación de VLAN y asignación de puertos a las VLAN's 29
Figura 15: Relación de Comandos Cisco a Open vSwitch en el Reinicio de la configuración del Switch 29
Figura 16: Relación de Comandos Cisco a Open vSwitch en la Configuración de VLAN's administrativas 30
Figura 17: Dockerfile de la imagen KV-SWITCH 31
Figura 18:Prueba 1 de KV-SWITCH 33
Figura 19: Resultado Prueba 1 de KV-SWITCH 34
Figura 20: Resultado Prueba 1 de KV-SWITCH (2) 35
Figura 21: Resultado Prueba 1 de KV-SWITCH (3) 35
Figura 22: Prueba 2 de KV-SWITCH 36
Figura 23: Prueba 2 de KV-SWITCH (2) 37
Figura 24: Resultado Prueba 2 KV-SWITCH 37
Figura 25: Resultado Prueba 2 KV-SWITCH (2) 37

Resumen

"Una red de computadores es un conjunto de equipos informáticos y software conectados entre sí por medio de dispositivos físicos o inalámbricos con la finalidad de compartir información, recursos y ofrecer servicios."

Para ello, se utilizan distintos dispositivos tales como switches, routers, firewalls, etc. El aprendizaje de la administración de este tipo de dispositivos es crucial para todo aquel que se vaya a dedicar al mundo de las redes de computadores.

Para aprender el uso y manejo de los distintos dispositivos de red se suele hacer uso de simuladores que permiten simular partes de una topología de red real aplicando las configuraciones necesarias a través de una interfaz de comandos.

Este trabajo de fin de grado se centra en esta línea de trabajo, aportando la posibilidad de simular switches con interfaz de usuario Cisco dentro del simulador GNS3, sin necesidad de utilizar elementos de software propietario. Para ello, se ha implementado una capa de software – una shell – sobre un switch software ya existente, el Open vSwitch, con el fin de que este se pueda configurar utilizando comandos Cisco. A este switch se le ha denominado KV-Switch.

Previamente se han revisado los antecedentes, existiendo diversas alternativas. La más parecida al desarrollo que aquí se presenta es el LiSA, por lo que se ha optado por implementar al menos las características que este proporciona.

KV-Switch se ha implementado y empaquetado de forma que pueda ser utilizado en sustitución de LiSA, pero consumiendo menos recursos y facilitando la enseñanza en el uso de este tipo de dispositivos.

Abstract

"A computer network is a set of Computer equipment and software connected to each other by means of physical or wireless devices with the purpose of sharing information, resources and offering services."

For this, different devices such as switches, routers, firewalls, etc. are used. The learning of the handling of this type of devices is crucial for all those who are going to dedicate themselves to the world of computer networks.

In order to learn the use and management of the different network devices, simulators are usually used to simulate parts of a real network topology by applying the necessary configurations through a command interface.

This end-of-degree project focuses on this line of work, providing the possibility of simulating switches with Cisco user interface within the GNS3 simulator. For this, a software layer – an interactive shell - has been implemented on an existing software switch, Open vSwitch, so that it could be configured using Cisco commands. This switch has been called, KV-SWITCH.

Previously, the background has been reviewed, and there are several alternatives. The most similar to the development presented here is the LiSA, so it has been chosen to implement at least the features it provides.

The KV-SWITCH has been implemented and packaged in such a way that it can be used in substitution of LiSA, but consuming less resources and facilitating teaching in the use of this type of devices.

Capítulo 1. Introducción

En este capítulo se detallan objetivos, tanto generales como específicos, del proyecto, además de la motivación y los antecedentes del mismo. Además, se indicarán la metodología y las herramientas utilizadas en el desarrollo propuesto.

1.1 Objetivos

El objetivo general de este Trabajo de Fin de Grado es la simulación de un switch Cisco[1] sobre Open vSwitch [2] al cual se le ha denominado como "KV-SWITCH" con la finalidad de que sea utilizado en la docencia práctica relacionada con la configuración de este tipo de dispositivos de red.

Los objetivos específicos considerados en este proyecto son los siguientes:

- Investigación y estudio de Open vSwitch para conocer su funcionamiento.
- Elección de un conjunto de funcionalidades Cisco a implementar.
- Desarrollo de una shell interactiva que reciba comandos Cisco y ejecute traduciéndolos a comandos Open vSwitch.
- Empaquetado del programa desarrollado para que pueda integrarse en el simulador de redes GNS3 [3]
- Realización de las pruebas y tests necesarios para demostrar el correcto funcionamiento del desarrollo

1.2 Motivación y antecedentes

La realización de este proyecto va dirigida al aprendizaje de los conceptos relacionados con el switching. Más concretamente, se pretende obtener una simulación de un switch con una interfaz de usuario similar a la de los dispositivos del fabricante Cisco que pueda ser integrada en un simulador de red.

La razón por la que se quiere que los usuarios tengan conocimientos sobre los comandos que proporciona el fabricante Cisco es porque su tecnología además de ser una de las de más alta calidad frente a otras empresas, es la que lidera el mercado con una cuota del 54,9% en 2016-2017 [4].

En la figura 1, podemos ver un gráfico comparativo proporcionado por la empresa IDC [5] en la que se muestran los beneficios obtenidos por las cinco

principales empresas de conmutación Ethernet en todo el mundo, pudiéndose observar que Cisco es el líder.

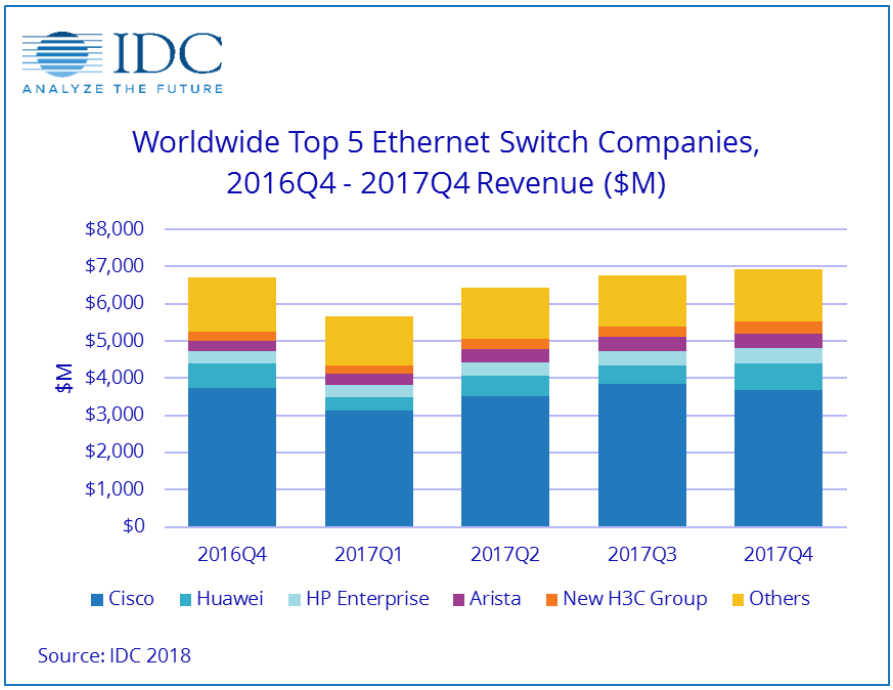


Figura 1: Top 5 Empresas de Conmutación Ethernet [4]

Actualmente, existen varias alternativas para lograr el objetivo que se ha propuesto. La primera de ellas es utilizar el simulador Packet Tracer [6] proporcionado por la propia Cisco. Este simulador permite simular distintos aspectos de una red, entre ellos switches, utilizando dispositivos propios de Cisco. El principal inconveniente de este simulador es que su uso está restringido a los alumnos de la Cisco Networking Academy [7].

Como alternativa se puede considerar el simulador NetSimk [8], que es gratuito pero su desarrollo avanza muy lentamente. Además, existen otras opciones como el uso de GNS3, que es un simulador libre que permite utilizar las imágenes del software de un dispositivo Cisco y ejecutarlas en un entorno virtual. Sin embargo, sí que es necesario adquirir una licencia para las imágenes del sistema operativo de Cisco. Lo más atractivo de GNS3 es que además permite simular topologías con dispositivos de múltiples fabricantes.

Para el aprendizaje y manejo de switches Cisco utilizando GNS3, podemos hacer uso del software LiSA (Linux Switch Appliance) [9], pero el desarrollo se encuentra paralizado desde el año 2009 y las características implementadas son muy básicas.

Por ello, se ha decidido desarrollar una alternativa libre que permita simular un switch que pueda ser operado mediante comandos Cisco, aunque en este proyecto no se contemplan todas las funcionalidades, sino únicamente las proporcionadas en LiSA, que se han tomado como punto de partida para futuros desarrollos.

Como LiSA es la alternativa más cercana al desarrollo que se ha realizado durante este TFG, a continuación, se profundizará en su arquitectura, lo que permitirá comprender mejor las ventajas e inconvenientes del desarrollo realizado.

Las características del switch LiSA son las siguientes:

- Conmutación de paquetes de capa 2 y 3 utilizado una arquitectura de PC estándar con Linux
- Resolver problemas de escalabilidad de VLAN de Linux
- Resolver el rendimiento con paquetes de difusión en puertos de acceso y troncales.
- Funciones básicas de conmutación: Conmutación de VLAN, etiquetado de VLAN, enrutamiento entre VLAN
- Comandos Cisco como interfaz de usuario

Las figuras 2 y 3 muestran un esquema de la arquitectura del LiSA Switch. La arquitectura consta de dos partes claramente diferenciadas, una se ejecuta en el espacio de usuario y otra en el espacio de kernel. La conmutación de paquetes se implementa en el espacio de kernel, mientras que en el espacio de usuario se realizan las configuraciones, el monitoreo y se controla el proceso de comunicación.

En la figura 3, podremos observar el diseño de la arquitectura LiSA donde se especifican los elementos que lo componen y los enlaces que existen entre ellos:

La parte que se ejecuta en el espacio de kernel, contiene el módulo del kernel de Linux, que es responsable del mecanismo de conmutación de paquetes. Este módulo está compuesto por varias partes:

- Conmutador: Es la base del módulo debido a que recibe los paquetes y toma las decisiones de conmutación y aplica algoritmos de manera eficiente para comunicarse con los puertos del switch.

- Forwarding DataBase: Cada switch debe contener dicha base de datos para crear la tabla CAM (tabla retenida por un conmutador de red para asignar direcciones MAC a los puertos)
- VLAN Database: Contiene toda la información necesaria para el enrutamiento de VLAN
- Interfaces VLAN: Necesarias para implementar el enrutamiento de VLAN utilizando las funcionalidades que Linux ya ofrece

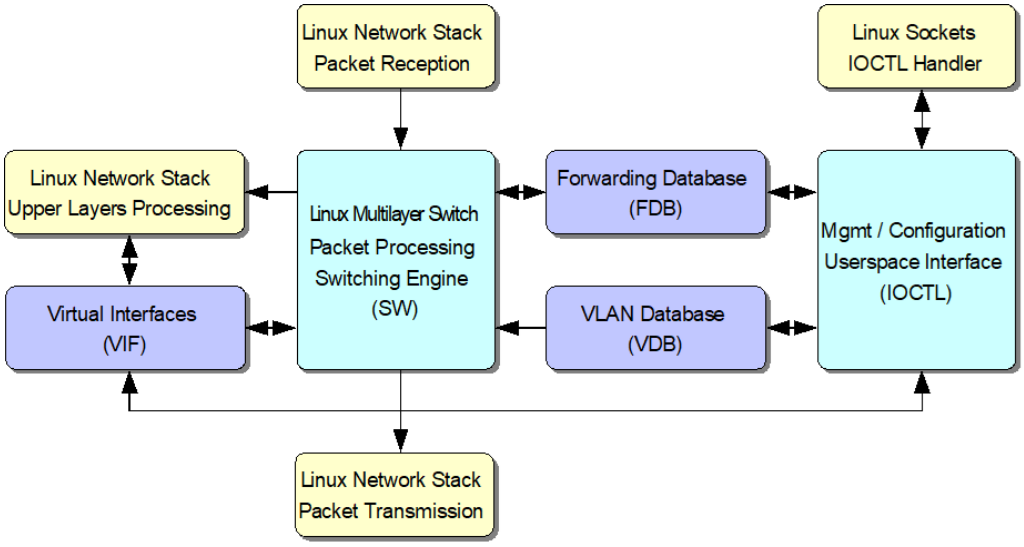


Figura 2: Kernel Space LiSA [10]

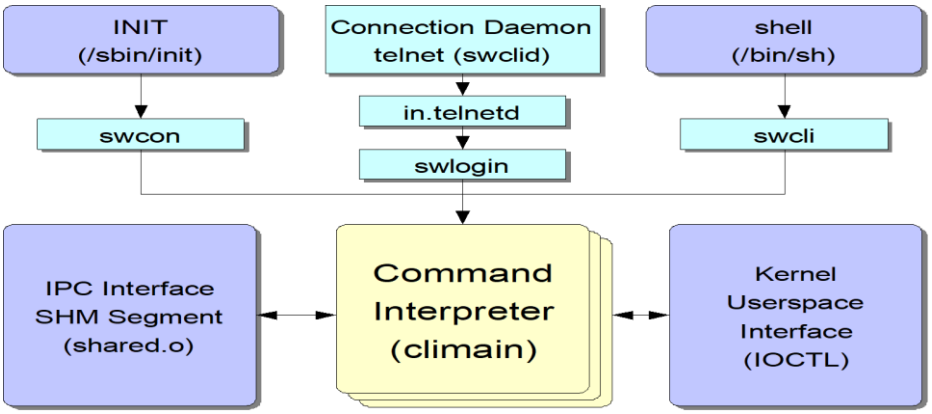


Figura 3: User Space LiSA [10]

Para configurar el conmutador, el usuario debe interactuar con el intérprete de comandos Cisco que se ejecuta en el espacio de usuario. Las funcionalidades se empaquetan en una biblioteca compartida que expone las funciones necesarias para administrar el switch [10].

1.3 Herramientas Propuestas

En esta sección, se expondrán las herramientas planteadas y finalmente utilizadas en el desarrollo del "KV-Switch"

1.3.1 Python

Se ha elegido Python [11] como lenguaje para la implementación de los módulos del proyecto. Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Además, se trata de un lenguaje de programación multi-paradigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

1.3.2 Open vSwitch

Open vSwitch, abreviado OVS, es un software de código abierto, diseñado para ser utilizado como un switch virtual en entornos de servidores virtualizados. Es el encargado de reenviar el tráfico entre diferentes máquinas virtuales (VMs) en el mismo host físico y también reenviar el tráfico entre las maquinas virtuales y la red física.

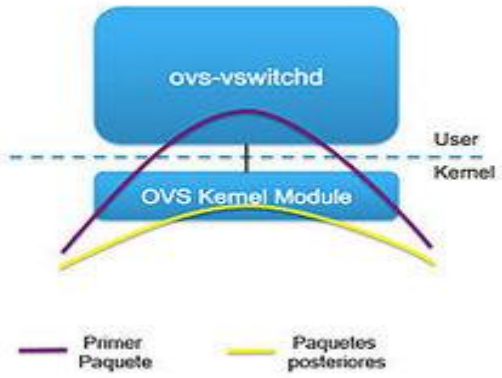


Figura 4: Proceso de intercambio de paquetes con OVS [12]

Nuevamente, la arquitectura de Open vSwitch distingue dos partes claramente diferenciadas. La primera se ejecuta en espacio de usuario y la segunda en espacio de kernel

En el espacio del kernel se ejecutarían los módulos y los drivers del sistema, mientras que en el espacio del usuario se encontraría la mayoría del software. En la figura 4, se muestra cómo interactúan ambas partes cuando durante el reenvío de paquetes. Cuando se envía el primer paquete de un flujo, sobre el cual no se ha realizado nunca una decisión de reenvío (forwarding), la parte

que se ejecuta en espacio de usuario toma la decisión relativa a ese paquete. Sin embargo, para los paquetes posteriores de ese mismo flujo, al no requerirse ninguna toma de decisión, se ejecuta completamente en espacio de kernel, utilizando para ello la información almacenada en una caché.

Esto hace que el proceso de reenvío sea mucho más rápido para los paquetes siguientes.

Los componentes de Open vSwitch que usamos este proyecto son:

- ovs-vswitchd: Un demonio que implementa las funcionalidades del switch, con un módulo del kernel de Linux para conmutación basada en flujos. Implementa el comportamiento ante VLANs, el bondig o la monitorización.
- ovsdb-server: Un servidor de base de datos ligero que ovs-vswitchd consulta para obtener su configuración. En la base de datos que gestiona se encuentran todos los parámetros de configuración del Open vSwitch, los cuales se almacenan de forma que dicha información se mantenga tras un reinicio del host.
- ovs-vsctl: Es utilizado para consultar y actualizar la configuración del ovs-vswitchd.

La relación entre los distintos componentes se muestran en la figura 5.

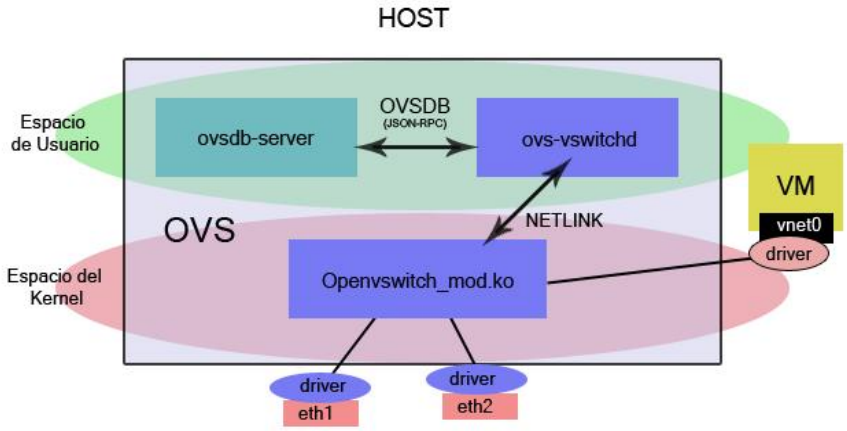


Figura 5: Componentes de OVS [13]

1.3.3 Librerías

1.3.3.1 IShell

Ishell [14] es una librería Python la cual nos permite la creación de una shell interactiva donde se implementan los comandos que queremos. Ishell es compatible con la finalización de comandos, argumentos dinámicos, un historial de comandos y un encadenamiento de comandos. Esta librería es básica en el desarrollo del proyecto.

1.3.3.2 Netifaces

Módulo en Python que permite obtener información de las interfaces de red. Con él se puede obtener la lista de interfaces, obtener información de enlace de red de cada interfaz, su dirección IPv4 y su dirección IPv6. [15]

1.3.4 Simuladores

Cuando estamos pensando en montar una pequeña red doméstica o de una pequeña empresa, antes de proceder con el montaje real, los administradores de redes suelen probar estos en entornos de pruebas para descartar posibles errores, optimizar configuraciones y, sobre todo, poder sabe con certeza que la red funcionará sin problemas y garantizará a los usuarios un entorno libre de errores.

Existen varios simuladores de redes para montar nuestras propias topologías virtuales y hacer pruebas con ellas, que se describirán en las siguientes secciones.

1.3.4.1 Cisco Packet Tracer

Este programa es uno de los simuladores de redes más completos. Desarrollado directamente por Cisco, es el recomendado por ejemplo para realizar pruebas con sus propios routers, switchs, hubs y servidores. Este programa es uno de los más sencillos de usar y permite, realizar todo tipo de virtualizaciones de redes.

La versión actual soporta un conjunto de Protocolos de Capa de Aplicación simulados, al igual que el enrutamiento básico con RIP, OSPF y EIGRP. Aunque Packet Tracer provee una simulación de redes funcionales, utiliza solo un pequeño numero de características encontradas en el hardware real corriendo una versión actual del Cisco IOS. Packer tracer no es adecuado para redes en producción [9].

Actualmente, su uso requiere la adquisición de una licencia disponible para los alumnos de la Cisco Networking Academy.

1.3.4.2 GNS3

GNS3 es un simulador de red gráfico que permite la emulación de redes complejas, utiliza los IOS de los equipos de Cisco y los ejecuta en un entorno virtual en el ordenador. También permite la simulación de dispositivos de otros fabricantes. Se puede utilizar en computadoras basadas en Windows, Linux, Mac OS X.

Las principales características que tiene el emulador GNS3 son las siguientes:

- Diseño de topologías de redes de alta calidad y complejidad
- Emulación de muchas plataformas de IOS de routers Cisco IOS, switch, firewalls y otros.
- Ejecución de software en forma de maquinas virtuales de distintos fabricantes.
- Emulación de Ethernet simple, ATM, switches Frame Relay, etc.
- Captura de paquetes utilizando Wireshark
- Conexión de redes simuladas al mundo real
- Es de fácil instalación debido a que todos los programas que se necesita para que funcione se encuentra en un solo paquete de instalación
- Está en constante actualización y periódicamente se puede encontrar versiones de la aplicación más robustas y con nuevas funcionalidades.

Para permitir simulaciones completas, el emulador GNS3 utiliza componentes importantes:

- Qemu, VirtualBox, VMWare, máquina emuladora y virtualizadora de código abierto.
- Docker Container, automatizador de aplicaciones dentro de contenedores de software.

La arquitectura de GNS3 está compuesta por:

- Dynamips, es el motor de emulación que permite simular diferentes plataformas hardware usando imágenes de sistemas operativos Cisco en un mismo host. Entre dichas plataformas se encuentran switcher Ethernet, Frame – Relay y ATOM con funcionalidades básicas.

- IDLE – PC, se trata de una herramienta que realiza un análisis en el código de una imagen IOS para determinar los puntos mas probables que representen un bucle de inactividad, de modo que, cuando se detecten, haga que los routers virtuales “duerman” durante ese instante. Es decir, IDLE-PC ayuda a Dynamips a emular el estado inactivo de la CPU virtual de un router.
- Dynagen, es una interfaz escrita en Python que provee la gestión, mediante línea de comando, de las plataformas emuladas por Dynamips haciendo más fácil su uso. Usa el modo “Hypervisor” para comunicarse con Dynamips y ambas pueden correr en el mismo o en diferente PC. También, simplifica la gestion de las redes virtuales ya que implementa comandos para listar, iniciar, parar, reiniciar, suspender, reanudar los diferentes dispositivos emulados, además determina los valores de IDLE – PC y realiza captura de paquetes.
    - Network File, se trata de un archivo, escrito usando sintaxis INI (INI file Syntax), que almacena la configuración de todos los dispositivos de red de la tipología virtual a simular, como son los routers, switches y las interconexiones entre ellos.

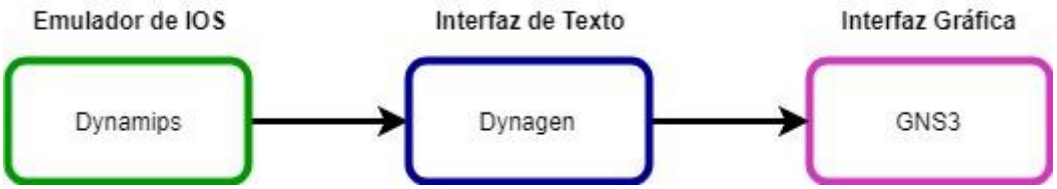


Figura 6: Esquema de Arquitectura GNS3

1.3.4.3 NetSim

NetSim [16] es un simulador de redes basado en eventos discretos. Se usa principalmente en ambientes educativos y de investigación. Permite simular tanto protocolos unicast como multicast y se utiliza intensamente en la investigación de redes moviles ad-hoc. Implementa una amplia gama de protocolos tanto de redes cableadas como de redes inalámbricas.

La versión actual, ns-3, esta diseñada para soportar todo el flujo de trabajo de la simulación desde la configuración hasta la recolección y análisis de tramas.

NS es un software libre el cual se ofrece bajo la versión 2 de la GNU General Public License y cuenta con dos versiones ns-2 y ns-3 que en general, son incompatibles.

1.3.4.4 NetSimk

Netsimk un simulador más para crear redes y poder realizar pruebas con ellas. Las funciones que nos ofrece son muy similares a las de los anteriores simuladores, aunque podemos destacar una implementación de herramientas y funciones adaptadas para el aprendizaje de comandos Cisco.

También podemos destacar que los escenarios que ofrecen son realistas, no virtuales, por lo que los resultados obtenidos se asemejan más a la realidad en cuanto a posibles fallos que podamos encontrar [8].

Concluimos este apartado de Simuladores, determinando que el que mejor características ofrece, además, de por su libre uso, es GNS3.

1.3.5 Herramientas de Empaquetado

Para que el usuario pueda hacer uso del programa dentro del simulador, es necesario realizar un empaquetado mediante software de virtualización (QEMU[17], VMware[18], VirtualBox[19]) o Docker[20] ya que GNS3 solamente permite la adición de software-propietario mediante estas herramientas.

En este apartado conoceremos cuales son las herramientas de empaquetado que disponemos para poder preparar nuestro proyecto y cual se ha decidido utilizar definitivamente por sus características notorias frente al resto.

1.3.5.1 Virtualizadores

En la siguiente tabla podremos ver las diferentes características de los virtualizadores que GNS3 soporta (VirtualBox, VMware, QEMU):

Característica/ Software	VMware	VirtualBox	QEMU
Conocimiento requerido para administración	Medio	Bajo	Bajo
Integración video, I/O	Medio	Alto	Bajo
Capacidad de Para-Virtualización	No	No	Sí

Driver para los Guest	Sí (vmware-tools)	Sí (vbox-additions)	Sí (Qemu-guest-agent)
Requerimientos del Guest	Ninguno	Ninguno	Ninguno
Discos RAW	Configuración adicional	Configuración Adicional	Configuración Adicional
Soporte Network Bridge	Sí	Sí	Sí
Sistemas Operativos Invitados Soportados	DOS, Windows, Linux, FreeBSD, Netware, Solaris, Virtual Appliances	Dos, Windows, Linux, OS/2, OpenBSD, FreeBSD, Netware, Solaris	Windows, Solaris, Linux, FreeBSD, OpenBSD, Mac OS, Zeta, BeOS
Sistemas Operativos Anfitrión Soportados	Windows, Linux	Windows, Linux, Mac OS	Windows, Linux, Mac OS
Arquitecturas	PROPIETARIO	686,x86-64	686,x86-64, Power PC, ia64, SPARC, ARM, MIPS
Rendimiento	Rápido/Muy Rápido	Rápido/Muy Rápido	Lento/Medio (con Kqemu)

Tabla 1: Comparación de VirtualBox, VMware y QEMU

1.3.5.2 Docker

A partir de la versión GNS3 1.5 aparece una nueva herramienta de "virtualización", Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, podemos implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que el código se ejecutará.

Docker proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a como una maquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores [21].

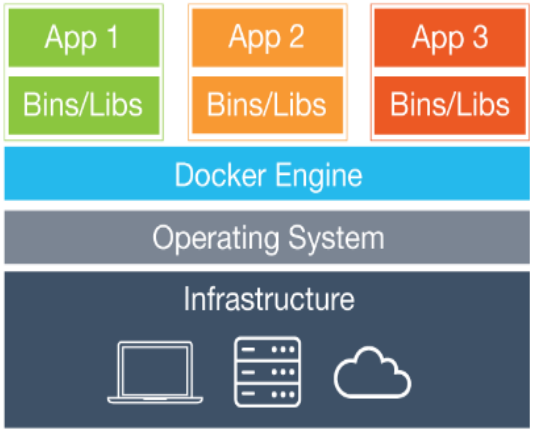


Figura 7: Esquema de Funcionamiento de Docker [22]

1.3.5.3 Docker vs Maquina Virtual (VM)

Cuando virtualizamos un sistema operativo utilizamos uno de los software de virtualización, el cual se encarga de crear el hardware virtual, configurarlos y permitirnos trabajar con él. Este tipo de maquinas virtuales trabajan directamente con un Hypervisor (Figura 9: Comparativa de funcionamiento Container y VM), y dentro de él se instala el sistema operativo completo, no utilizando ninguna dependencia del sistema principal ni compartiendo dependencias entre otros sistemas virtuales

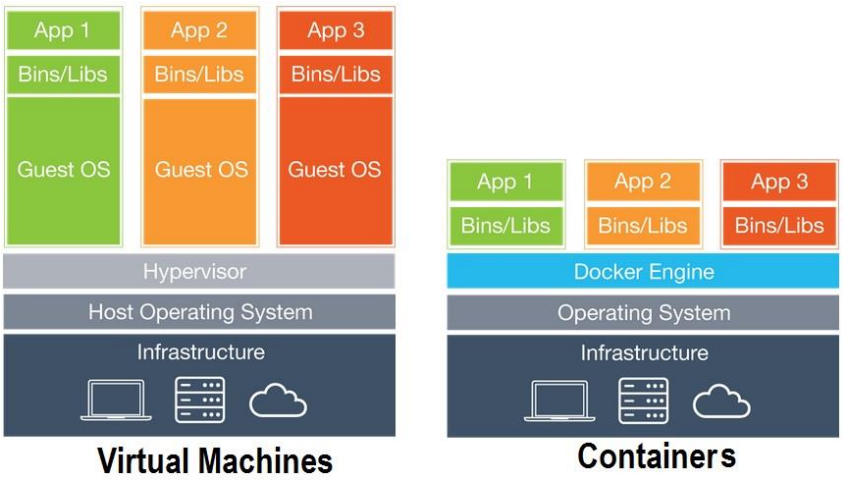


Figura 8: Comparativa de funcionamiento Container y VM [23]

Cuando hablamos de contenedores es diferente. Este tipo de tecnología no solo comparte la infraestructura y delega el resto al Hypervisor, sino que además depende, por un lado, del sistema operativo principal, y por otro de una herramienta encargada de hacer funcionar estos contenedores (Docker).

Docker cuenta con las librerías y dependencias necesarias para crear contenedores, por lo que dentro de estos, en lugar de existir un sistema operativo completo, solo se incluyen los binarios y librerías necesarias (las dependencias) y la aplicación en cuestión que queremos virtualizar.

Podemos concluir pues, que la principal diferencia entre ambas tecnologías es que, por un lado virtualizamos un sistema operativo con VirtualBox instalamos y ejecutamos el 100% del sistema operativo. Cuando utilizamos los contenedores de Docker, en lugar de virtualizar un sistema operativo completo, creamos un container que, a partir de la imagen de un sistema operativo determinado, instalamos sobre ella las librerías y dependencias necesarias para realizar nuestra tarea, obviando todo lo demás. De esta manera, los contenedores no son un sistema operativo virtual como tal, sino más bien se entienden como "paquetes" que se ejecutan aislados sobre el sistema operativo principal, pero sin depender de un sistema virtual [24].

Se ha decidido hacer uso de Docker, por su clara diferencia en eficiencia y capacidad frente a una herramienta de virtualización.

1.4 Herramientas Utilizadas

En la sección 1.3, hemos analizado y comparado cada una de las herramientas que nos podían ser útiles en el desarrollo del proyecto, de todas ellas, se han elegido:

- Lenguaje de Programación: Python
- Simulador: GNS3
- Librerías: iShell y Netifaces
- Software: Open vSwitch
- Herramienta de Empaquetado: Docker

1.5 Metodología

A través del siguiente esquema podremos ver las fases en las que se ha dividido el desarrollo del TFG.

Se han considerado varias etapas:

- Fase de Investigación
- Fase de Desarrollo
- Fase de Pruebas
- Fase de Empaquetado

El esquema de la figura 9, muestra cada una de las etapas con las tareas asociadas.

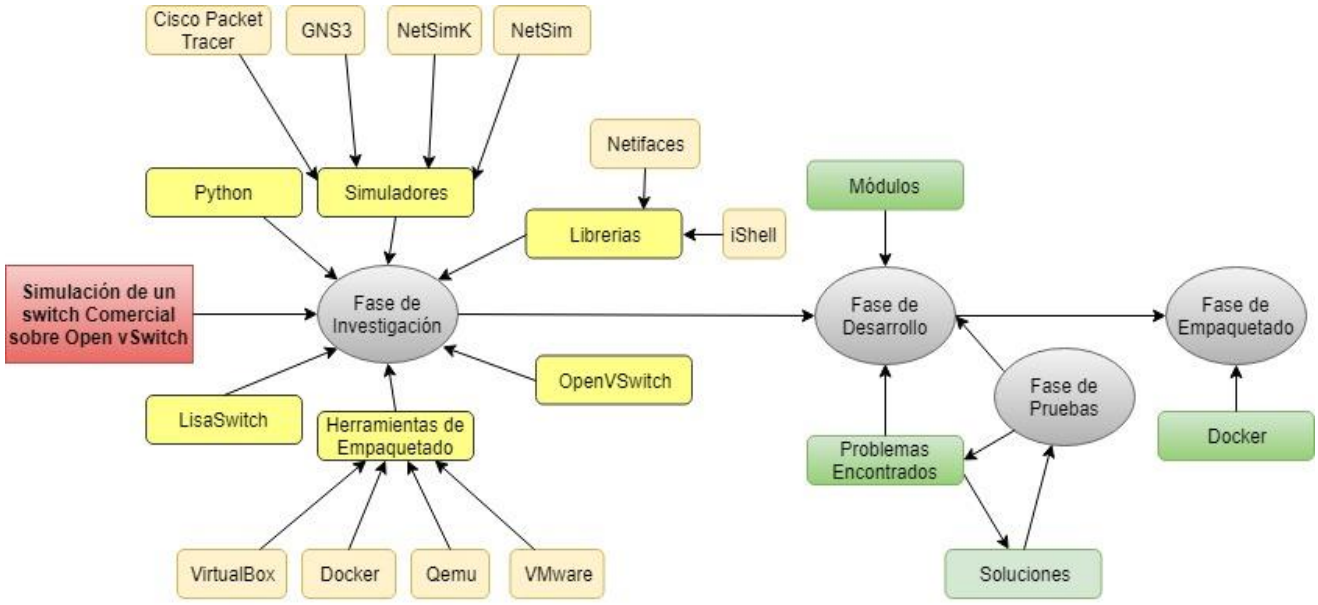


Figura 9: Esquema de Metodología utilizada

Para la simulación de un switch comercial sobre Open vSwitch comenzamos con una primera fase de investigación y estudio sobre cada una de las herramientas necesarias para el desarrollo del mismo. Entre ellas, encontramos primeramente, la elección del lenguaje de programación a utilizar para su desarrollo, entre los cuales, por su simplicidad y robustez, se decidió hacer uso del lenguaje interpretado, Python.

Ya que intentamos simular las características de LiSA Switch, se analizaron los antecedentes sobre el desarrollo del mismo.

Para la implementación del código, es necesario hacer uso del módulo Python que nos ofrece Open vSwitch, sobre el cual se ha realizado un traductor de comandos Cisco mediante el uso de una shell interactiva implementada mediante la librería iShell de Python. Además, ha sido necesario añadir la librería netifaces de Python para acceder de una forma más sencilla a los adaptadores de red que se disponen en el switch creado.

Para la utilización de la herramienta, es necesario una plataforma de simulación de redes, entre las cuales destacamos, Cisco Packet Tracer, GNS3, NetSimK y NetSim, los cuales fueron investigados y analizados para saber cual es el óptimo para el desarrollo de nuestro proyecto. Tras este estudio, se decide usar GNS3.

GNS3, nos obliga a realizar un empaquetado del programa. Se estudiaron las distintas posibilidades que nos ofrece el simulador (herramientas de virtualización y Dockerización) para empaquetar y añadir el dispositivo a la plataforma donde se decide utilizar Docker por la eficiencia que ofrece al usuario frente al resto de virtualizadores.

Terminada la fase de investigación y estudio, pasamos a la fase de desarrollo puesto que ya se tienen a disposición las herramientas necesarias para la implementación del programa. Se programan los módulos y clases Python de la shell interactiva y el traductor de comandos, desembocando en una fase de pruebas las cuales, a través de un método de “prueba-error”, añadimos excepciones a nuestro programa, solucionando problemas de sintaxis y usabilidad.

Una vez realizadas todas las pruebas necesarias para que el código funcione correctamente. Pasamos a la fase de empaquetado, donde, como se comentó anteriormente, hacemos uso de la herramienta Docker para crear la correspondiente imagen con las dependencias necesarias para su uso.

Capítulo 2. Implementación

2.1 Arquitectura y Funcionamiento del Sistema

La implementación del proyecto se ha realizado a partir de la siguiente arquitectura (figura 10).

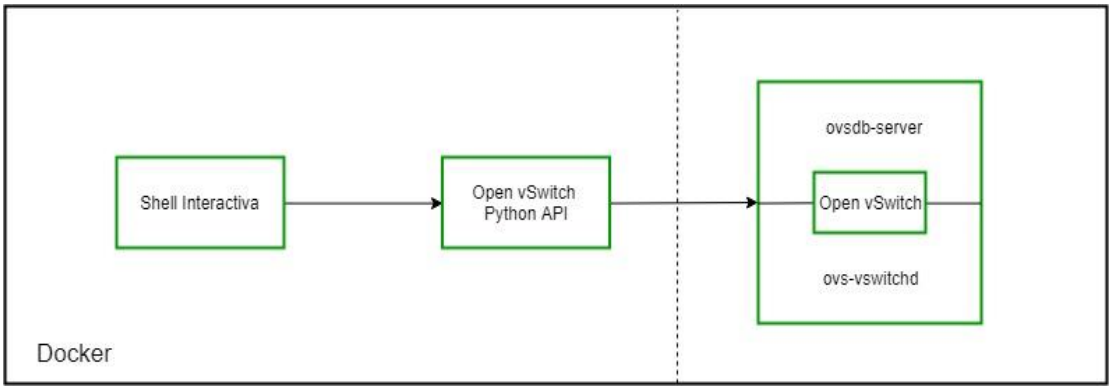


Figura 10: Arquitectura KV-SWITCH

Se ha creado una shell interactiva la cual traduce comandos Cisco a Open vSwitch, esta, interacciona con la API Python de Open vSwitch [25]. Como la API original ofrecía únicamente un número reducido de funciones y su desarrollo parecía no estar completo se optó por modificar el código original para incorporar las funcionalidades necesarias para el desarrollo del proyecto. La API se comunica con el software de Open vSwitch, para ejecutar los correspondientes comandos.

Finalmente, todo este proceso ha sido empaquetado mediante la herramienta Docker.

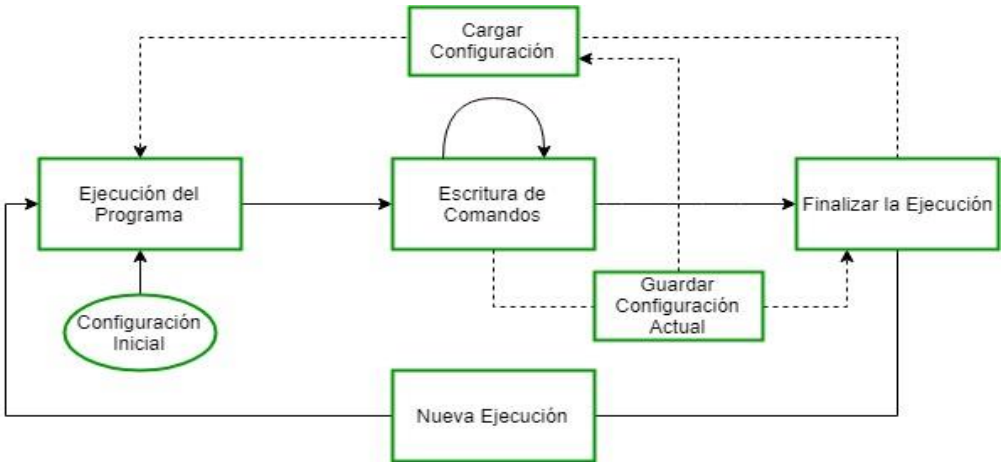


Figura 11: Funcionamiento Inicial de KV-SWITCH

Si nos adentramos en el funcionamiento del sistema, tal y como se muestra en la figura 11. Se establece una configuración inicial al iniciar el programa. Posteriormente, el usuario podrá realizar las configuraciones que se desee a través de la introducción de comandos, teniendo la posibilidad de finalizar la ejecución del mismo.

A la hora de iniciar el programa, el usuario puede cargar la configuración anterior o ejecutar el programa con una nueva configuración.

2.2 Configuración Inicial del Programa

Antes de que el usuario/cliente interactúe con el programa se deben realizar una serie de configuraciones iniciales como se muestra en la figura 12.

Una vez ejecutamos el programa, internamente iniciamos el demonio openvswitch-switch. Seguidamente, procedemos con la creación del bridge "br0". Un bridge es un dispositivo de interconexión de redes que opera en la capa 2 (nivel de enlace de datos) del modelo OSI. El bridge responde al protocolo IEEE 802.1D. Un bridge conecta segmentos de red formando una sola subred [26]. Lo crearemos mediante el comando Open vSwitch: `ovs-vsctl add-br br0`. Como en GNS3 podemos especificar cuantos adaptadores de red tendrá nuestro switch, esos adaptadores tendrán que ser añadidos al bridge. La librería netifaces a través de la función `interfaces()` nos proporciona el listado de interfaces de red del dispositivo, las cuales serán insertadas a través del comando Open vSwitch: `ovs-vsctl add-port br0 [port]`. Por último, mediante el listado que nos proporciona `netifaces.interfaces( )`, creamos un array de puertos, para proceder con su apagado m ediante el comando: `ip link set [port] status=Down`. Esto último se hace porque en un switch real las interfaces inicialmente se encuentran apagadas.

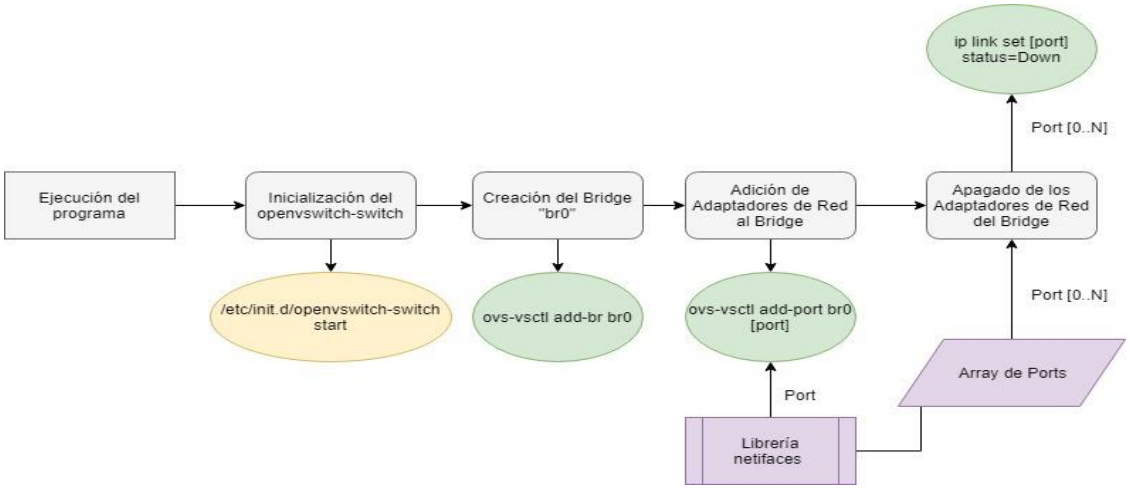


Figura 12: Configuración Inicial de KV-SWITCH

2.3 Funciones Implementadas

En el siguiente apartado procederemos a explicar cada uno de los comandos y métodos implementados dentro del código Python.

2.3.1 Comandos Cisco

Dentro de este apartado, veremos todos los comandos Cisco que han sido implementados dentro del programa y las relaciones entre cada uno de ellos Hay que recalcar que se trata de un conjunto de comandos y funciones reducido que no corresponde con el que tendríamos en un switch real.

El funcionamiento de la interfaz de comandos de una consola de un switch Cisco está basado en distintos modos de funcionamiento en cada uno de los cuales se puede ejecutar distintos comandos.

- Modo EXEC privilegiado:
  - `reload`
  - `hostname [name]`
  - `history`
  - `copy`
    - `startup-config`
    - `running-config`
  - `show`
    - `vlan`
    - `interfaces`
  - `configure`
  - `exit`
- Modo Config:
  - `vlan [id]`
  - `no vlan [id]`
  - `exit`
  - `interface [ethX]`
  - `interface range [ethX – ethY]`
  - `interface vlan [id]`
- Modo Config – vlan:
  - `name [vlan_name]`
  - `exit`
- Modo Config – if:
  - `switchport mode access`
    - `switchport access vlan [id]`
    - `shutdown`

        - no shutdown
    - switchport mode trunk
        - switchport mode trunk allowed vlan [ids]
        - shutdown
        - no shutdown
    - exit

En la figura 13, veremos un esquema de relación de comandos con sus respectivos modos:

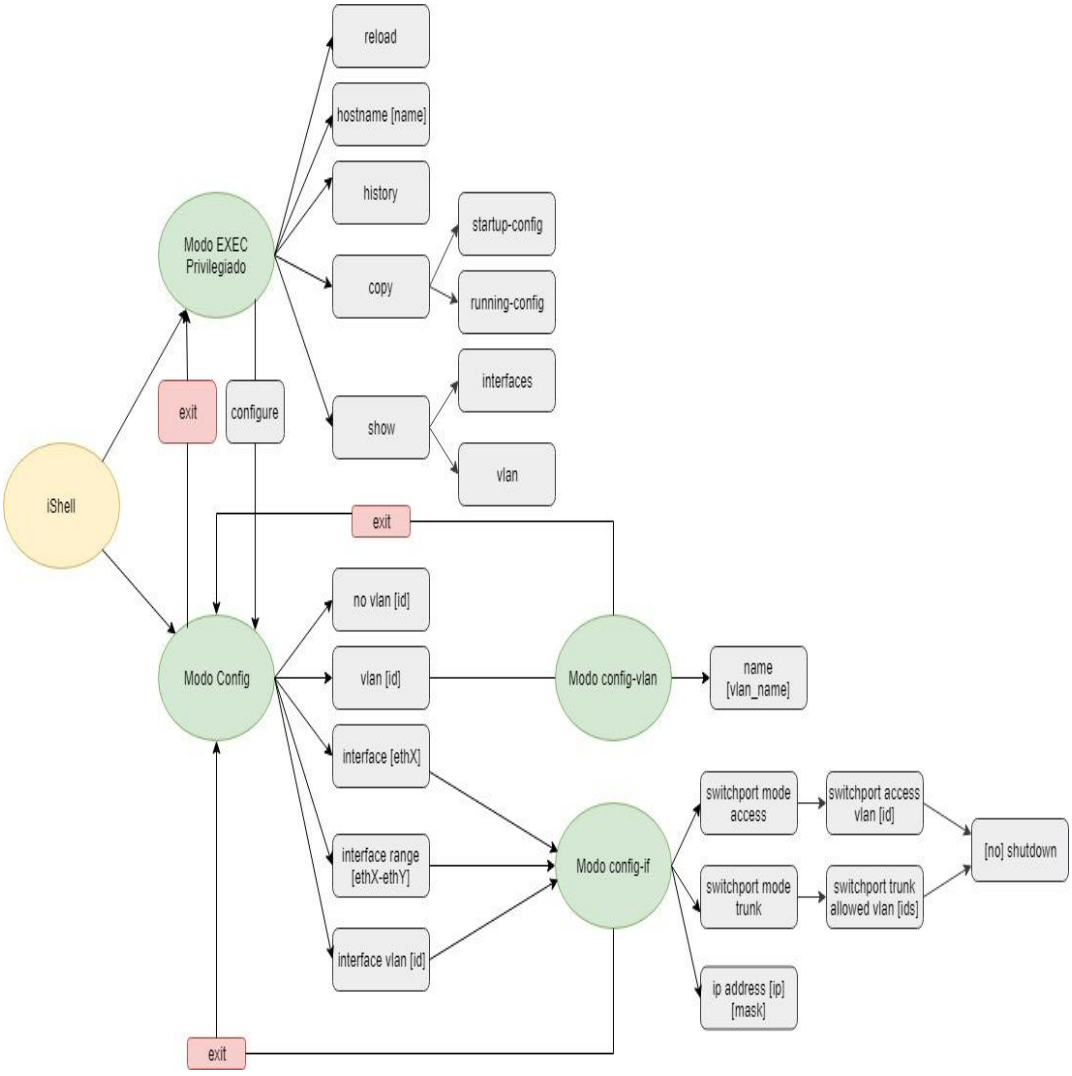


Figura 13: Esquema de Comandos KV-SWITCH

A continuación, se explica cada uno de los comandos implementados:

COMANDO	EXPLICACIÓN	MODO
reload	Resetea toda la configuración actual del switch.	EXEC privilegiado
hostname [name]	Cambia el nombre del prompt inicial (S1#) al que ponga el usuario en [name] → (name#).	EXEC privilegiado
history	Muestra un historial de comandos escritos.	EXEC privilegiado
copy startup-config running-config	Ejecuta la configuración que se encuentre en el fichero config.txt el cual guarda una configuración previa.	EXEC privilegiado
copy running-config startup-config	Copia la configuración actual en el archivo config.txt	EXEC privilegiado
show interfaces	Visualiza las interfaces disponibles del switch. Además, muestra el estado actual de la interfaz (Up, Down).	EXEC privilegiado
show vlan	Visualiza las VLAN creadas en el switch junto a su interfaz asignada y dirección IP (si es interfaz administrativa).	EXEC privilegiado
configure	Comando que utilizamos para pasar al prompt de configuración del switch.	EXEC privilegiado
vlan [id]	Utilizamos este comando para la creación de VLAN's.	config

exit	Comando que utilizamos para pasar de un Prompt actual, al previo: EXEC privilegiado → Termina el programa. config → EXEC privilegiado. config-if → config. config-vlan → config.	EXEC privilegiado config, config-if, config-vlan
no vlan [id]	Utilizamos este comando para borrar una VLAN previamente creada y toda la configuración que se haya realizado con esa VLAN.	config
interface [ethX]	Hacemos uso de este comando para entrar dentro de la configuración de una interfaz determinada y pasar al prompt siguiente (config-if).	config
interface range [ethX-ethY]	Utilizamos este comando para entrar dentro de la configuración de un rango de interfaces determinado.	config
interface vlan [id]	Este comando nos permite entrar dentro de la configuración de una VLAN previamente creada.	config
switchport mode access	Este comando es necesario para la configuración de una interfaz como puerto de acceso a una VLAN.	config-if

switchport mode trunk	Este comando es necesario para la configuración de una interfaz de tipo trunk para la conexión entre VLAN's a través de dos switches.	config-if
switchport access vlan [id]	Este comando es utilizado para especificar de qué VLAN será puerto de acceso la interfaz en la que nos encontremos.	config-if
shutdown	Comando que utilizamos para mantener apagada la interfaz correspondiente que estemos configurando.	config-if
no shutdown	Comando que utilizamos para mantener encendida la interfaz que estemos configurando.	config-if
ip address [ip] [mask]	Una vez entramos dentro de la configuración de una VLAN determinada. Podremos asignarle una dirección IP para configurar la interfaz administrativa.	config-if
name [vlan_name]	Comando que utilizamos para asignarle un nombre a la VLAN que acabamos de crear. Si no asignamos un nombre, se designará automáticamente como "default-X".	config-vlan

Tabla 2: Comandos Cisco Implementados

2.3.2 Comandos Open vSwitch

Como se ha explicado en el Capítulo 1, en este proyecto se hará uso de Open vSwitch. Esto hace que sea necesario traducir los comandos de Cisco a Open vSwitch estableciendo una correspondencia entre ellos. Por ello previamente, se hará una revisión de los comandos y las funciones de la API.

A continuación, se explican cada una de las funciones utilizadas en el desarrollo del proyecto, de las cuales, las funciones **ovs_vsctl_add_bridge(bridge), ovs_vsctl_del_bridge(bridge) y ovs_vsctl_set(table, record, column, key,value)**, fueron previamente creadas en la API Python de Open vSwitch.

Comando:	**ovs_vsctl_add_bridge(bridge)**
Explicación	Ejecuta el comando: `ovs-vsctl add-br [bridge]` Crea un bridge llamado [bridge], inicialmente el bridge no tendrá puertos creados.

Tabla 3: Función ovs_vsctl_add_bridge(bridge)

Comando:	**ovs_vsctl_del_br(bridge)**
Explicación	Ejecuta el comando: `ovs-vsctl del-br [bridge]` Elimina el bridge [bridge] y todos sus puertos.

Tabla 4: Función ovs_vsctl_del_br(bridge)

Comando:	**iplink(port,status)**
Explicación	Ejecuta el comando: `ip link set [port] [status=”Up” or “Down”]` A través de esta función encendemos o apagamos un puerto [port].

Tabla 5: Función iplink(port,status)

Comando:	**ovs_vsctl_admin_port(bridge,port,tag)**
Explicación	Ejecuta el comando: `ovs-vsctl add-port [bridge] [port] tag=[tag]` A través de esta función, creamos un nuevo puerto en el [bridge] cuyo [port] no es una interfaz de red existente, sino "fake" a la cual se le asigna un tag, lo cual es equivalente a asignarle la VLAN a la que pertenece el puerto [port].

Tabla 6: Función ovs_vsctl_admin_port(bridge,port,tag)

Comando:	**ovs_vsctl_add_trunk_port(Port,trunk)**
Explicación	Ejecuta el comando: `ovs-vsctl set port [port] trunks=[trunk]` Mediante esta función, configuramos un puerto del switch tipo trunk.

Tabla 7: Función ovs_vsctl_add_port_trunk(port,trunk)

Comando:	**ifconfig(ip,port)**
Explicación	Ejecuta el comando: `ip addr add [ip] dev [port]` Esta función se encarga de asignarle una dirección IP a un puerto port.

Tabla 8: Función ifconfig(ip,port)

Comando:	**ovs_vsctl_set(table, record, column, key, value)**
Explicación	Utilizamos esta función para la asignación de puertos a las correspondientes VLAN's. Ejecuta el comando: `ovs-vsctl set Port [record=ethX] tag=[value].`

Tabla 9: Función ovs_vsctl_set(table,record,column,key,value)

Comando:	**ovs_vsctl_set_admin(port,type)**
Explicación	Ejecuta el comando: `ovs-vsctl set-Interface [port] type=[type]` Hacemos uso de esta función durante la creación de interfaces administrativas, como vimos en el comando anterior, creamos el puerto en el bridge, pero es necesario que el puerto sea de tipo "Internal", de forma que hacemos uso de este comando.

Tabla 10: Función ovs_vsctl_set_admin(port,type)

2.3.3 Relación de Comandos Cisco a Open vSwitch

A continuación, se mostrará un esquema de la relación que tienen los comandos que hemos visto en los anteriores apartados.

- Creación de VLAN's y Asignación de Puertos a las VLAN's.

Una vez ejecutamos el programa, procedemos a crear VLAN's a partir de los comandos Cisco: `vlan [id]` y `name [vlan_name]`.

Una vez creada la VLAN correspondiente, elegimos el puerto que queremos configurar a través del comando Cisco: `Interface [port]`.

Posteriormente, pasamos al tipo de puerto que queremos configurar, es decir, de acceso o trunk.

- Un puerto de acceso pertenece únicamente a una VLAN asignada de forma estática.
- Un puerto trunk, puede ser miembro de múltiples VLAN.

Para poder establecer enlaces troncales entre dos dispositivos, es necesario que las tramas pertenecientes a una VLAN en el dispositivo de salida sean reconocidas como tramas de dicha VLAN en el dispositivo de destino, ya que la premisa de que las tramas pertenecientes a una VLAN solo deben reenviarse a dispositivos de la misma VLAN debe seguir cumpliéndose. Para conseguir esto, se utilizan las tramas etiquetadas que son la trama original añadiendo datos relativos a la VLAN. Esto permite al switch o router de destino determinar los puertos a los que puede reenviar la trama en función de la VLAN.

Existen distintos métodos para etiquetar tramas, los más utilizados son el IEEE 802.1Q y el ISL. Lo importante es recordar por los enlaces troncales deben viajar tramas etiquetadas, ya sea con el estándar IEEE 802.1Q o ISL [27].
Destacamos, que este proyecto se hace uso del protocolo IEEE 802.1Q (protocolo estándar) y no ISL puesto que es un protocolo propietario del fabricante Cisco [28].

Esta configuración, la realizamos mediante el comando Cisco: `switchport mode [access|trunk]`.
Una vez elegido el tipo de puerto a configurar, pasamos a asignar el puerto:

- Si es de acceso: `switchport access vlan [id]`.

(siendo [id] la VLAN que queremos asignar a dicho puerto del switch).

- Si es tipo trunk: `switchport trunk allowed vlan [ids]`.

(siendo [ids], las VLAN's de las que será miembro el puerto del switch).

Ya hemos realizado todos los comandos Cisco necesarios para la creación de las VLAN's y asignación de puertos del switch, ahora, haremos uso de ellos para su traducción a Open vSwitch.

Necesitamos los siguientes valores:

- VLAN a la que pertenece el puerto de acceso (`switchport access vlan [id]`).
- Nombre del puerto que se está configurando (`interface [port]`).
- Tipo de puerto (`switchport mode [access|trunk]`).
- VLAN's a las que pertenece el puerto tipo trunk (`switchport trunk allowed vlan [ids]`).

Finalmente, con estos datos, procederemos a escribir el comando Open vSwitch correspondiente:

- Si es puerto de acceso: `ovs-vsctl set Port [port] tag=[id]`.
- Si el puerto es tipo trunk: `ovs-vsctl set Port [port] trunks=[ids]`.

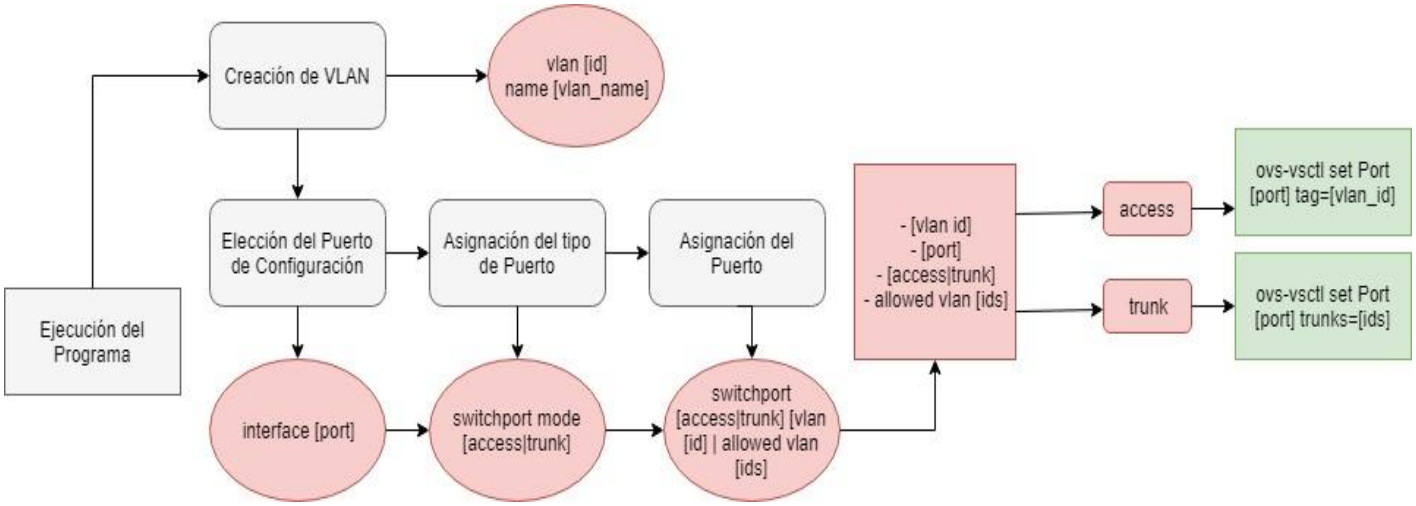


Figura 14: Relación de Comandos Cisco a Open vSwitch en Creación de VLAN y asignación de puertos a las VLAN's

- Reinicio del Switch.

Una vez ejecutamos el programa y creamos una determinada configuración, si queremos reiniciar la configuración del switch totalmente, procederemos a escribir el comando Cisco: `reload`.

A continuación, se ejecutarán los correspondientes comandos Open vSwitch para el reinicio del switch:

- Borrado del bridge: `ovs-vsctl del-br [bridge]`.
- Creación del bridge: `ovs-vsctl add-br [bridge]`.

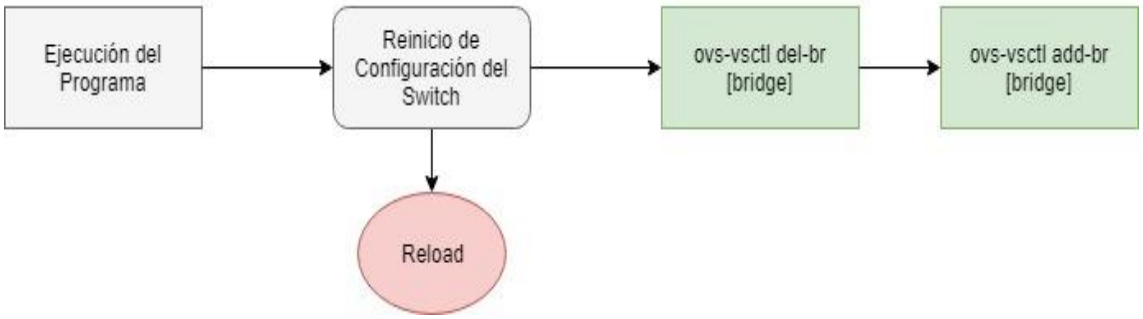


Figura 15: Relación de Comandos Cisco a Open vSwitch en el Reinicio de la configuración del Switch

- Configuración de interfaz administrativa.

Una vez ejecutamos el programa y previamente se ha creado una o más VLAN's. Pasamos a la elección de la interfaz que queremos configurar mediante el comando Cisco: `interface vlan [id]`.

Cuando entramos en el modo de configuración de la VLAN correspondiente, procedemos a asignarle la dirección administrativa mediante el comando Cisco: `ip address [ip] [mask] o ip address [ip/mask]`.

Ya se han escrito todos los comandos Cisco necesarios para la configuración de una interfaz administrativa.

Necesitamos los siguientes argumentos de los comandos Cisco escritos:

- ID de la VLAN que se encuentra configurando (`interface vlan [id]`).
- Nombre del bridge (valor que se encuentra de forma interna dentro del programa).
- Dirección IP y máscara de red (`ip address [ip] [mask]).`

Ahora, veremos los comandos Open vSwitch que le corresponden:

- Creamos un puerto "fake" que se llamará "vlan[id]" (por ejemplo, si configuramos la VLAN 10, el puerto se llamara "vlan10"): `ovs-vsctl add-port br0 "vlan[id]" tag=[id]`.
- El puerto "fake" creado, lo configuramos como tipo Internal (para poder asignarle la dirección IP): `ovs-vsctl set Interface "vlan[id]" type=Internal`.
- Asignamos la dirección al puerto creado: `ip addr add [ip] [mask] dev [port]`.
- Activamos el puerto: `ip link set "vlan[id]" status=Up`.

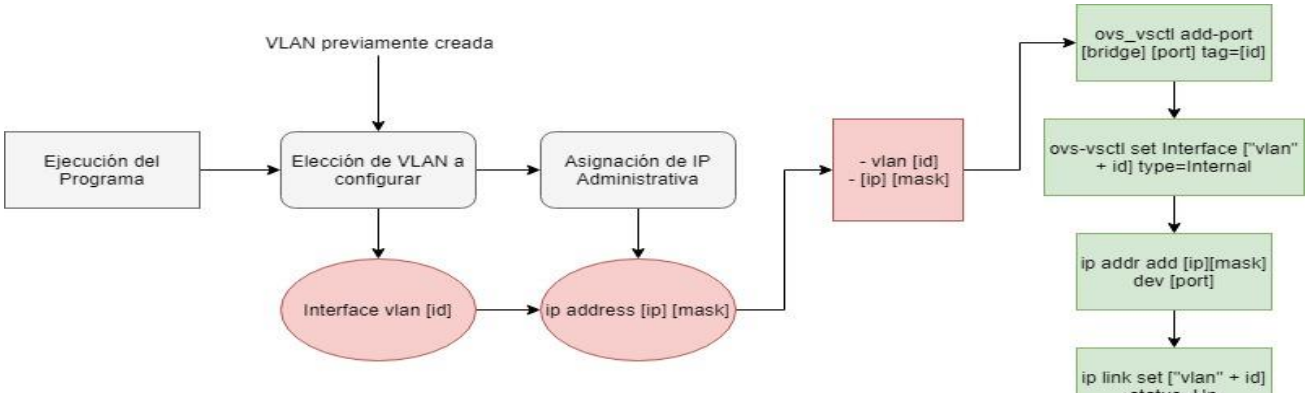


Figura 16: Relación de Comandos Cisco a Open vSwitch en la Configuración de VLAN's administrativas

2.3.4 Empaquetado Docker

Para poder crear imágenes bases con los requerimientos que se necesiten creamos un documento denominado “Dockerfile”, este, contiene todos los comandos que un usuario puede llamar a la línea de comandos para montar una imagen.

Para crear e instalar la imagen “kv-switch”, hacemos uso del comando:

```
docker build --network=host –t kv-switch
```

(El argumento --network=host lo utilizamos para establecer el modo de red para las instrucciones de ejecución durante la compilación).

Para ejecutar la imagen “kv-switch”:

```
docker run --net=host –v /home/shared:/home –ti kv-switch
```

Donde net=host permite que el contenedor tenga acceso total a Internet compartiendo todas las interfaces del host.

Donde –v es un argumento donde definimos un volumen de la imagen asociado a un volumen local, es decir, aplicado a la situación del proyecto, el directorio `/home` del contenedor, tendrá dentro de él, los archivos que están contenidos dentro del directorio `/home/shared` en local.

```
FROM debian:8.10
RUN apt-get update
RUN apt-get install -y python-pip python-dev lib32ncurses5-dev net-tools
RUN apt-get install -y openvswitch-common openvswitch-switch python-openvswitch
RUN pip install ishell
RUN pip install netifaces
ADD command.py /
ADD console.py /
ADD log.py /
ADD util.py /
ADD utils.py /
ADD vswitch.py /
ADD common.py /
ADD shell.py /
CMD ["python","./shell.py"]
```

Figura 17: Dockerfile de la imagen KV-SWITCH

Los argumentos del Dockerfile que vemos en la imagen anterior tienen la siguiente finalidad:

- `FROM:` Definir una imagen base para crear nuestra nueva imagen con `Dockerfile.`
- `RUN:` Nos permite ejecutar comandos en la imagen base antes de ser creada.
- `ADD:` Nos permite agregar o copiar archivos desde el equipo local a la imagen.
- `CMD:` Ejecutar una acción por defecto al crear el contenedor.

Capítulo 3. Pruebas y Resultados

3.1 Resultados Obtenidos

Se han realizado las siguientes pruebas en un ordenador con las siguientes características:

- CPU: *Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz.*
- Memoria RAM: *12,0GB Dual-Channel DDR3 @ 798MHz.*
- Sistema Operativo: *Kali GNU/Linux.*
- GNS3 Versión: *GNS3 v2.1.6.*

Prueba 1: Se procede a la realización de una práctica de la asignatura Laboratorio de Redes del Grado en Ingeniería Informática. El objetivo de esta práctica es el uso de los comandos Cisco a través del LiSA Switch para la creación de VLANs, creación de puertos de acceso y trunk y la configuración de interfaces administrativas. La topología consiste en dos switches interconectados entre sí mediante un enlace troncal a cada uno de los cuales se les conectan dos PC's como se muestra en la Figura 18.

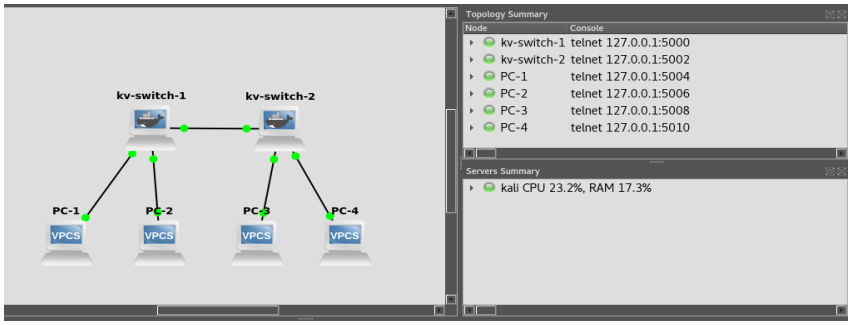


Figura 18:Prueba 1 de KV-SWITCH

Las configuración de los PC's será la siguiente:

PC	Dirección IP	VLAN
PC-1	192.168.0.1	10
PC-2	192.168.0.2	20
PC-3	192.168.0.3	10
PC-4	192.168.0.4	20

Tabla 11: Configuración Prueba 1 KV-SWITCH

Inicialmente la CPU se encuentra en un 2.6% y la memoria RAM en un 17.2% y como podemos comprobar en la imagen anterior, tras encender los

dispositivos, tendremos nuestra CPU a un 23.2% y la memoria RAM a 17.3%.

Una vez realizada la configuración de los switches, comprobamos la conexión entre los PC's.

Tal y como se ve en la Figura 19, se realizan comprobaciones con los 4 PC's que forman la topología, probando conexiones de cada uno de ellos con los otros tres. Como PC-2 y PC-4 pertenecen a la VLAN 20, conectan sin problemas al igual que sucede con PC-1 y PC-3, los cuales pertenecen a la VLAN 10. Además, se observa como los PC's de la VLAN 10 no conectan con los de la VLAN 20 y viceversa.

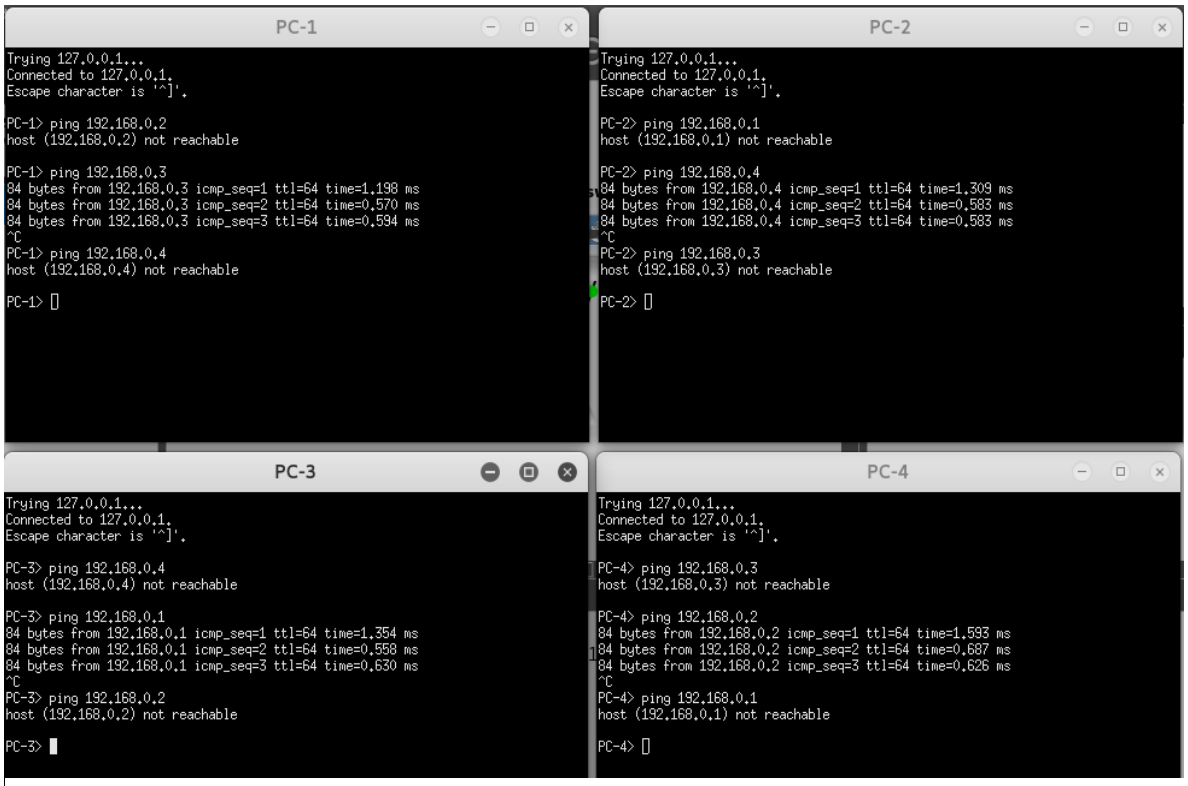


Figura 19: Resultado Prueba 1 de KV-SWITCH

A continuación, configuramos la interfaz administrativa de la VLAN 10 en el kv-switch-2 con la dirección 192.168.0.10. En la Figura 20, podremos ver como solamente podrá conectarse a la dirección 192.168.0.10 el PC-3 puesto que pertenece a la VLAN 10.

Configuramos la interfaz administrativa de la VLAN 10 en el kv-switch-2 con la dirección 192.168.0.20, realizamos las correspondientes pruebas y veremos como, al igual que el caso anterior, tal y como vemos en la Figura

21, solamente podrá conectarse a la dirección 192.168.0.20 el PC-2 puesto que pertenece a la VLAN 20.

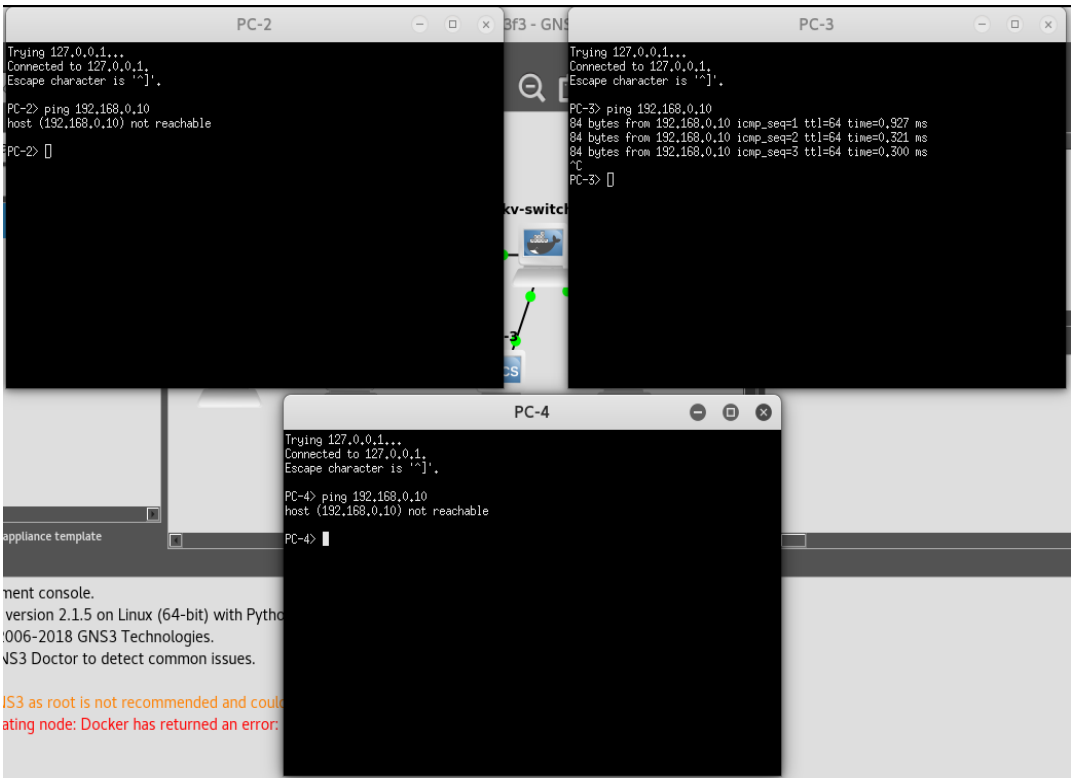


Figura 20: Resultado Prueba 1 de KV-SWITCH (2)

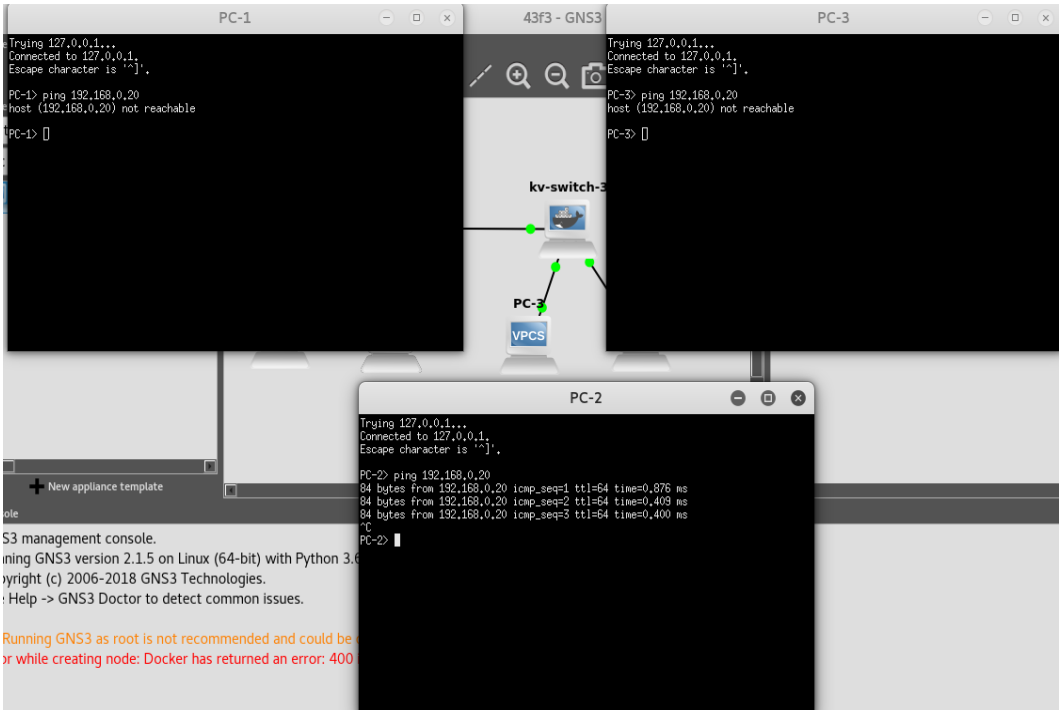


Figura 21: Resultado Prueba 1 de KV-SWITCH (3)

Prueba 2: Se han insertado 13 switches KV-SWITCH interconectados en topología de árbol con 8 PC's. Como podemos comprobar, inicialmente la CPU se encuentra en un 2.6% y la memoria RAM en un 17.2%.

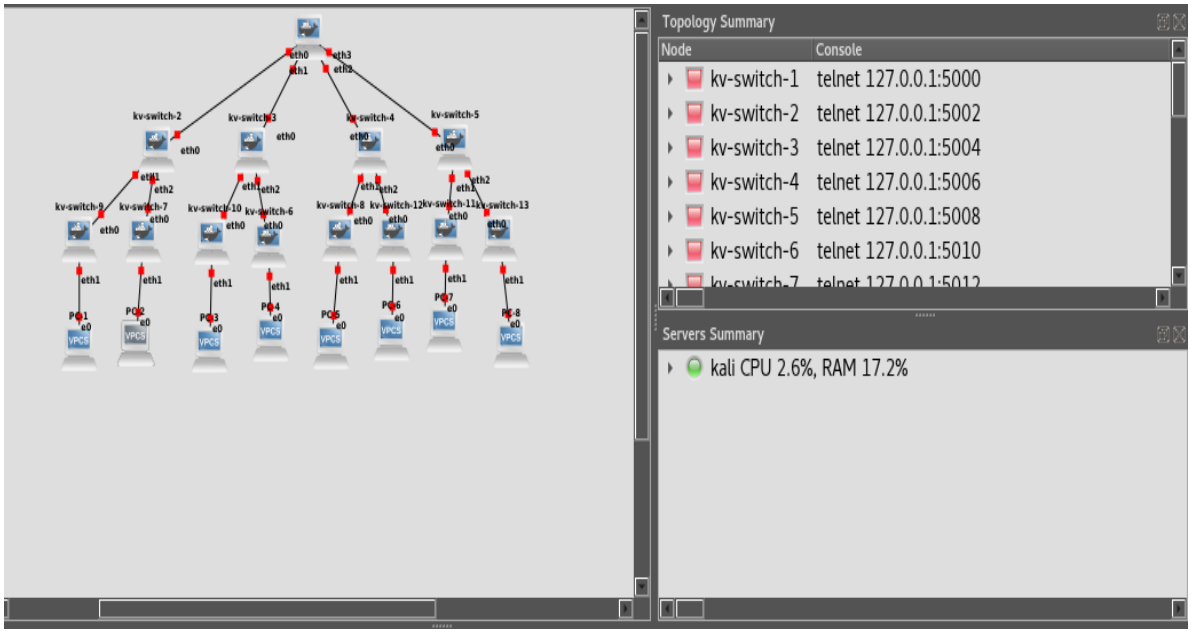


Figura 22: Prueba 2 de KV-SWITCH

Las configuración de los PC's será la siguiente:

PC	Dirección IP	VLAN
PC-1	192.168.0.1	10
PC-2	192.168.0.2	20
PC-3	192.168.0.3	10
PC-4	192.168.0.4	20
PC-5	192.168.0.5	10
PC-6	192.168.0.6	20
PC-7	192.168.0.7	10
PC-8	192.168.0.8	20

Tabla 12: Configuración Prueba 2 KV-SWITCH

Una vez configurados los switches, obtenemos los siguientes datos:

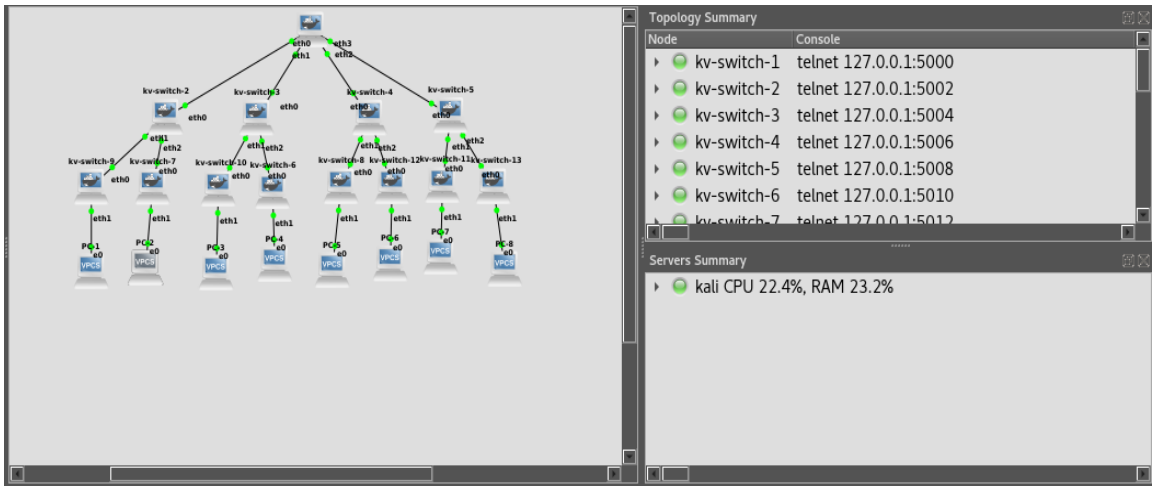


Figura 23: Prueba 2 de KV-SWITCH (2)

Como podemos ver en la imagen anterior, la CPU se encuentra en un 22.4% y la memoria RAM en un 23.2%.

A continuación, procederemos a la comprobación de la conexión entre PC's:

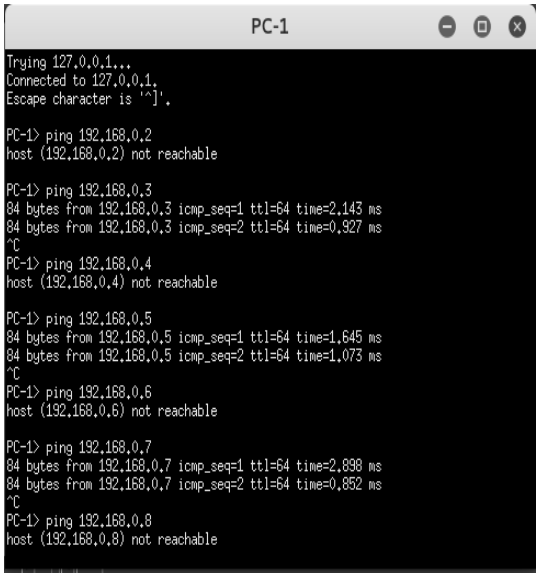


Figura 24: Resultado Prueba 2 KV-SWITCH

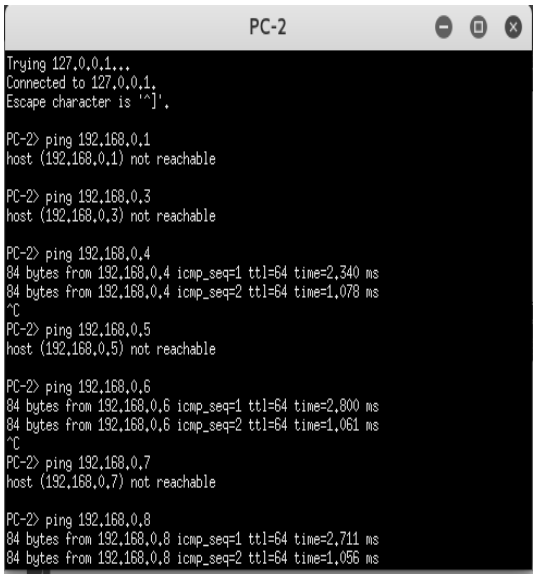


Figura 25: Resultado Prueba 2 KV-SWITCH (2)

Como podemos comprobar en las anteriores imágenes, se han hecho pruebas con los PC1 y PC2, realizándose conexiones concluyentes con los PC's pertenecientes a su misma VLAN y fallidas con los que no pertenecen a su misma VLAN.

3.2 Dificultades Encontradas

Durante el desarrollo del Trabajo de Fin de Grado, se han encontrado dificultades en la implantación del software KV-SWITCH.

- Uso de VirtualBox como herramienta de empaquetado: Inicialmente, se iba a implementar el software en la plataforma de virtualización VirtualBox. El principal problema es el uso masivo de recursos, que ya se mencionó cuando se compararon las VM frente a Docker Por ello, se decide utilizar la herramienta Docker, la cual permite añadir una cantidad elevada de dispositivos en GNS3 sin uso excesivo de recursos.
- Conexión con Internet mediante Docker: Durante la etapa de investigación se realizaron pruebas con Docker para conocer su funcionamiento, donde se encontró un problema con la conexión a Internet desde el container ya que era necesario el uso del FLAG network=host.
- Usabilidad de KV-SWITCH: Puesto que el proyecto tiene como objetivo principal la enseñanza de los comandos que nos ofrece Cisco, hubo que ser excesivamente cuidadoso con cada uno de los comandos implementados para que tratasen cualquier tipo de excepciones en el caso de un uso inapropiado por parte del usuario.

Capítulo 4. Conclusiones y Trabajos Futuros

Las conclusiones del trabajo realizado son las siguientes:

- Se han estudiado distintas opciones existentes para la simulación de switches en el simulador GNS3, concluyéndose que las opciones existentes tenían una serie de limitaciones, ya sea en el desarrollo o de licencia de uso.
- Se ha desarrollado una shell tipo Cisco para traducir los comandos de dicho fabricante a comandos Open vSwitch de forma que el usuario puede introducir comandos Cisco que luego serán ejecutados de forma transparente sobre Open vSwitch.
- Se han implementado las funcionalidades que actualmente proporciona el LiSA, que es lo mínimo requerido para el aprendizaje, con la posibilidad de extenderlo en el futuro a otros escenarios.
- Esto último, facilita la enseñanza de la configuración de switches en el entorno de un fabricante con una gran cuota de mercado, sin incurrir en violaciones de licencia, permitiendo, además, la simulación de configuraciones multi-fabricante.
- El empaquetado basado en Docker permite la simulación de un elevado número de dispositivos consumiendo una cantidad reducida de recursos, sobre todo memoria RAM.
- El hecho de que el proyecto esté basado en un software consolidado como Open vSwitch permite que sea fácilmente mantenible y extensible por otros sin tener que entrar en los detalles técnicos de la implementación del switch, ya que lo único que se necesita es encontrar una equivalencia entre los comandos de Cisco y Open vSwitch.

Los trabajos futuros son los siguientes:

- Mejorar los mensajes de salida de los comandos para que sean más parecidas a un switch Cisco real.
- Añadir otras funcionalidades tales como la posibilidad de usar protocolos Spanning-Tree o bounding entre otras.
- Prueba y testeo con usuarios reales en las prácticas del Grado de Ingeniería Informática

Chapter 4: Conclusions and Future Work

The conclusions of the work carried out are the following:

- Different existing options for the simulation of switches in the GNS3 simulator has been studied, concluding that the existing options had a lot of limitations, either in the development or License.
- A Cisco shell has been elaborated to translate that manufacturer's commands to Open vSwitch commands so that the user can enter Cisco commands that will following run transparently in Open vSwitch.
- The functionalities currently provided by the LiSA have been implemented, which is the minimum required for learning, with the possibility of extending it in the future to other scenarios.
- This last mentioned, facilitates the teaching of the configuration of switches in the environment of a manufacturer with a large market share, without incurring license violations, and allowing the simulation of multi-manufacturer configurations too.
- The packaging based on Docker allows the simulation of a large number of devices consuming a reduced amount of resources, especially RAM.
- The fact that the project is based on consolidated software such as Open vSwitch allows it to be easily maintainable and extensible by others without the need of going through the technical details of the switch implementation, as all needed is to look for an equivalence between Cisco commands and Open vSwitch.

Future works:

- To improve the output messages of the commands to be more similar to a real Cisco switch.
- To add other functionalities such as the possibility of using Spanning-Tree or bounding protocols among others.
- Testing with real users.
- Testing with real users in the practices of Computer Science Engineering Degree.

Referencias

[1] https://www.cisco.com/c/es_es/index.html
[2] http://www.openvswitch.org/
[3] https://www.gns3.com/
[4] https://www.muycanal.com/2018/03/07/switches-routers-ethernet
[5] https://idcspain.com/
[6] https://www.netacad.com/es/courses/packet-tracer
[7] https://www.netacad.com/es
[8] http://netsimk.com/
[9] http://lisa.mindbit.ro/
[10] http://lisa.mindbit.ro/ LiSA slides at RoEduNet International Conference 2006 lisa-roedu-slices.pptx
[11] https://www.python.org/
[12] https://es.wikipedia.org/wiki/Open_vSwitch#/media/File:Openvswitch.jpg
[13] https://es.wikipedia.org/wiki/Open_vSwitch#/media/File:OVS_sin_Controller.jpg
[14] https://pypi.org/project/ishell/
[15] https://pypi.org/project/netifaces/
[16] https://www.tetcos.com/
[17] https://www.qemu.org/
[18] https://www.vmware.com/es.html
[19] https://www.virtualbox.org/
[20] https://www.docker.com/
[21] https://aws.amazon.com/es/docker/
[22] https://platzi.com/blog/docker-security-scanning/
[23] http://www.martindoestheblog.com/2017/01/11/primeros-pasos-con-las-herramientas-de
[24] https://www.redeszone.net/2017/12/31/diferencias-virtualbox-contenedor-docker/
[25] https://github.com/openvswitch/ovs/tree/master/python/ovstest
[26]https://sites.google.com/site/redeslocalesyglobales/2-aspectos-fisicos/5-dispositivos-de-interconexion-de-redes/3-puentes
[27] Práctica 7. VLANs y enrutamiento entre VLAN's (Grado en Ingeniería Informática – Laboratorio de Redes). Universidad de La Laguna
[28] https://es.wikipedia.org/wiki/Inter_Switch_Link