UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

SIMULACIÓN DE UNA RED SDN DE VIDEOVIGILANCIA IP BASADA EN GNS3

David Casado Jiménez

Tutor: José Óscar Romero Martínez

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 9 de septiembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG NO. ES-CV-000030

Agradecimientos

En primer lugar, dar las gracias a mi madre. Por todos los sacrificios que ha tenido que hacer para que yo esté aquí. También por enseñarme que los límites se los marca uno mismo. Ni el tiempo ni las circunstancias.

A mis compañeros de la universidad, con los que tantos momentos buenos y malos pasé. Por la compañía y el apoyo prestado.

A todos esos profesores que me he cruzado a lo largo de esta carrera, que hacen de su profesión, su pasión por lo que hacen. Gracias por las lecciones en todos los aspectos.

A mi tutor por la idea de proponer esa fusión entre entorno real y redes SDN. Por la confianza de poder contar con él para este proyecto.

Resumen

En este trabajo, se propone la implementación de un sistema de videovigilancia IP basado en SDN cuyo encaminamiento de los paquetes se verá condicionado por el volumen del tráfico y el estado de la red.

Al no disponer de los equipos necesarios para una prueba real, se realizará una simulación de la infraestructura de red mediante el simulador GNS3. En esta simulación se plantearán distintos escenarios en los que el encaminamiento de los distintos flujos de tráfico variará en función del estado de la red. Para así demostrar el dinamismo a la hora de implementar políticas de priorización y calidad del servicio mediante redes SDN.

El objetivo de este proyecto consiste en priorizar el tráfico de videovigilancia IP sobre el resto, para conseguir las máximas prestaciones, de este servicio en particular, y del resto de la red en general.

Palabras clave: SDN, GNS3, Openflow, switching, ancho de banda, QoS.

Resum

En aquest treball es proposa la implementació d'un sistema de videovigilància IP basat en tecnologia de xarxes SDN, l'encaminament de la qual els paquets es condicionarà pel volum de tràfic i l'estat de la xarxa.

Pel fet que no es disposa de l'equip necessari per a una prova real, es realitzarà una simulació de la infraestructura de la xarxa utilitzant el simulador GNS3. En aquesta simulació es consideraran uns quants escenaris en què els diferents fluxos d'encaminament de tràfic variaran en funció de l'estat de la xarxa. A fi de demostrar el dinamisme a l'hora d'implementar polítiques de priorització i qualitat del tràfic mitjançant xarxes SDN.

L'objectiu d'aquest projecte consisteix a prioritzar el tràfic de videovigilància IP sobre la resta, per a aconseguir les màximes prestacions d'aquest servei i de la resta de la xarxa.

Paraules clau: SDN, GNS3, Openflow, switching, ample de banda, QoS.

Abstract

This work proposes an IP-surveillance system implementation based on Software-Defined Networking (SDN) technology, where packet switching depends on network traffic.

Due to the fact that the equipment required for a real test is not available, a simulation of the network infrastructure will be carried out using the GNS3 simulator. In this simulation, several scenarios will be considered, where different traffic flows will vary depending on the network status, with the aim to test the flexibility given by SDN technology to implement prioritization and quality of service policies.

The aim of this project is to prioritise IP-surveillance traffic over the rest, in order to achieve maximum performance for this particular service and for the rest of the network.

Keywords: SDN, GNS3, Openflow, switching, bandwidth, QoS.

Índice

Índice de figuras

Capítulo 1. Introducción y objetivos

El desarrollo de este trabajo surge de la idea de utilizar las redes SDN para solucionar algunos problemas de una red en producción. Por lo que se propone recrear el tipo de red real, por medio de un software que pueda emular el comportamiento de esa red y un controlador que gestione la red de una manera eficiente. Además de plantear soluciones para resolver esos problemas, idear mecanismos para la optimización de dicha red. Una vez pase la evaluación y se consiga obtener los resultados esperados, poderla trasladar a la red en producción. Algunos de los problemas que ocurrían en la red real, eran debido a caídas de enlace, o dispositivos que se sobrecalientan y funcionan de una manera anómala. Esto provoca que encontrar los equipos con mal funcionamiento en una red tan grande, sea una tarea tediosa. Por lo que, en ocasiones, se decanta por forzar el apagado completo de la red, con los inconvenientes que ello supone. Aspectos como la seguridad, funcionalidad, o aplicaciones que tenga la red y que pueden verse afectados, pueden provocar problemas si no se tienen en cuenta. Otra clase de problemas son debido a la congestión de tráfico y la pérdida de datos, que en ocasiones conlleva datos sensibles que no deben perderse.

La finalidad era crear un proyecto que abarque todos los aspectos imprescindibles de la red real y reproducirlos en una plataforma de la que se pueda obtener unos resultados lo más próximos a un entorno real. Una buena elección fue el software de GNS3. Un software que brinda muchas posibilidades para poder crear una red con la máxima fidelidad posible a la red real. Y así mantener aspectos importantes, para poder evaluar posibles soluciones. Se va a realizar la simulación mediante redes definidas por software (SDN), ya que es una manera de reducir la complejidad de la arquitectura y los costes de equipamiento y así poder programar la red para resolver los problemas de la red física. Una vez desplegada la red, comenzar a barajar diferentes planteamientos. La elección del controlador SDN ONOS y el diseño de un servicio (1) para el redireccionamiento de los flujos, como solución a la pérdida de datos de los diferentes tráficos. Y el diseño de otro servicio (2) que junto al soporte de ONOS, de una solución a la caída de dispositivos. Retratando las características de la red, donde un cliente solicita un flujo de datos de videovigilancia IP, y un servidor envía dicho tráfico a través de una red. Y dónde existen otros tipos de tráfico dentro de esa misma red, ajenos al cliente-servidor.

Por lo que se propone un laboratorio con los diferentes escenarios, donde reflejar los problemas, y las diferentes soluciones propuestas.

1.1 Objetivos

1 - Congestión del tráfico y la pérdida de datos.

2 - Uso del servicio (1) para redireccionar los flujos y evitar pérdidas.

3 - Caída de dispositivos y la aplicación del servicio (2).

Capítulo 2. SDN

2.1 Introducción

Las empresas están avanzando hacia arquitecturas de red más nuevas para la construcción y el despliegue de aplicaciones. Con tecnologías como la virtualización y la contenedorización es posible desplegar rápidamente aplicaciones complejas y altamente escalables.
Las aplicaciones informáticas requieren recursos de computación, almacenamiento y redes. Tradicionalmente, la infraestructura de servidores y almacenamiento era instalada y configurada por los administradores de sistemas y, por separado, los administradores de redes conectaban los servidores y configuraban la red. Actualmente este proceso no se adapta a las necesidades de despliegue de aplicaciones bajo demanda y altamente automatizadas. Además, la arquitectura de red tradicional fue diseñada para un entorno de aplicación más estático, mientras que ahora es necesaria la flexibilidad.
Como solución entra en escena **SDN**, que hace que la infraestructura de la red sea más fácil de manejar e integrar con la infraestructura del servidor y el almacenamiento.

El concepto central de la *Red Definida por Software* es separar la inteligencia y el control (por ejemplo, el enrutamiento) de los elementos de reenvío (es decir, los conmutadores) y concentrar el control de la gestión y el funcionamiento de la red en un componente lógicamente centralizado: un Controlador SDN.

2.2 Arquitectura

Una arquitectura SDN consiste en tres planos o capas diferentes y un conjunto de interfaces que permite la comunicación entre ellos:

El **Plano de Control,** situado en la capa intermedia, se encarga de tomar decisiones sobre la manera en que los paquetes deben ser reenviados por uno o más dispositivos de red y de imponer tales decisiones a los dispositivos para su ejecución. Este plano consiste en un conjunto centralizado de controladores basados en software que tiene una visión abstracta de toda la infraestructura de la red, lo que permite al administrador de la red aplicar políticas personalizadas en todos los dispositivos de la red, basadas en la topología o en las solicitudes de servicios externos. El controlador SDN está en el corazón de la arquitectura y es la entidad inteligente que controla los recursos para entregar los servicios.
Algunas funciones del plano de control son descubrir y mantener la topología, seleccionar e instalar rutas de paquetes, detectar fallos o caídas de dispositivos, instalar políticas, recopilar información del estado, etc.

Se comunica con las aplicaciones mediante un interfaz Northbound (Northbound APIs).

La capa inferior o **Plano de Datos** consiste en dispositivos de forwarding físicos y virtuales. Estos dispositivos pueden soportar funciones básicas como el reenvío, pero también otros tipos de funciones, como: almacenamiento, transcodificación y vigilancia, entre otras.

Este plano se comunica con el plano de control mediante un interfaz Southbound (Southbound APIs).

En el **Plano de Aplicaciones** o capa superior se encuentran las aplicaciones y los servicios que definen el comportamiento de la red. Estas aplicaciones interactúan con el plano de control para lograr una función de red específica como: calidad de servicio, seguridad, virtualización y funciones de ingeniería de tráfico.

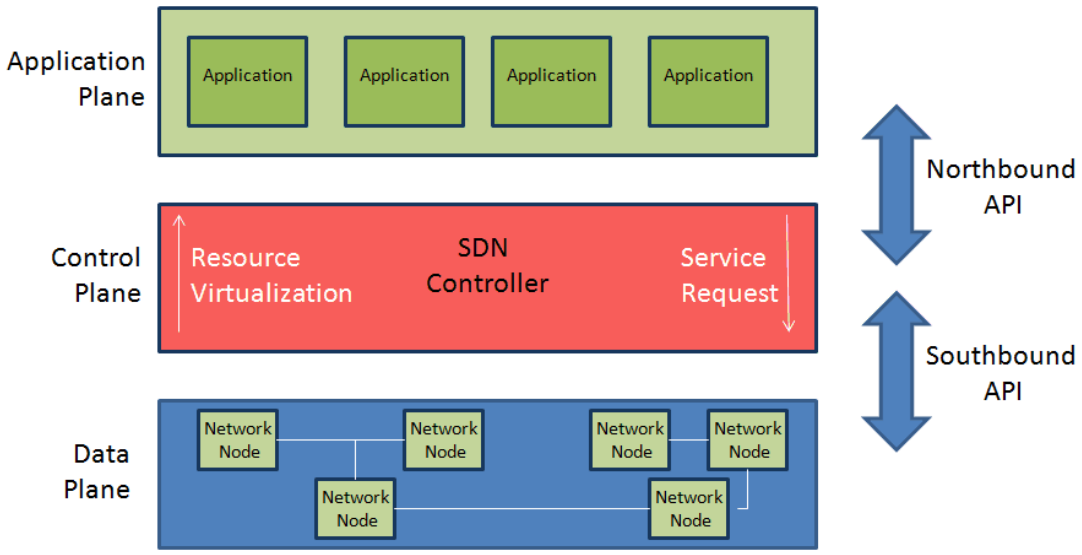

Figura 1. Arquitectura SDN

2.3 OpenFlow

En SDN se han definido diferentes protocolos de comunicación, uno de los más importantes es Openflow. Este protocolo permite la comunicación entre el plano de datos y el de control de las redes definidas por software, siendo actualmente el protocolo más utilizado por los sistemas SDN.
Un sistema Openflow consiste en un controlador que se comunica con uno o varios switches con soporte Openflow. El protocolo define mensajes que se intercambian entre el controlador (plano de control) y el dispositivo (plano de datos) por un canal seguro. Existen tres tipos distintos de mensajes: de controlador a switch, simétricos y asíncronos, y especifican cómo debe reaccionar el dispositivo en diversas situaciones, y cómo debe responder a los comandos del controlador. Mensajes de controlador-switch son iniciados por el propio controlador y usados para examinar el estado del switch. Los mensajes asíncronos se inician en el switch y sirven para actualizar el controlador de los sucesos en la red y cambios de estado del propio conmutador. Los mensajes simétricos comienzan en el switch o el controlador y se envían sin solicitud.

No.	Time	Source	Destination	Protocol	Length	Info
116	128.095942	192.168.122.39	192.168.122.100	OpenFlow	74	Type: OFPT_HELLO
118	128.241917	192.168.122.100	192.168.122.39	OpenFlow	90	Type: OFPT_FEATURES_REQUEST
120	128.243732	192.168.122.39	192.168.122.100	OpenFlow	98	Type: OFPT_FEATURES_REPLY
122	128.244639	192.168.122.100	192.168.122.39	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
124	128.245541	192.168.122.39	192.168.122.100	OpenFlow	1106	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
126	128.247388	192.168.122.100	192.168.122.39	OpenFlow	94	Type: OFPT_GET_CONFIG_REQUEST
128	128.250079	192.168.122.39	192.168.122.100	OpenFlow	74	Type: OFPT_BARRIER_REPLY
129	128.250079	192.168.122.39	192.168.122.100	OpenFlow	78	Type: OFPT_GET_CONFIG_REPLY

Figura 2. Mensajes Openflow

Estas decisiones son reglas de flujo almacenadas en tablas dentro de esos dispositivos. Las tablas de flujo y de grupos se encargan de realizar búsquedas de flujos y coincidencias (matches) y reenviar mensajes. La tabla de reenvío consiste en una lista de entradas de flujo con formato: match, acción y contador. Cuando llega un paquete, se compara su cabecera con las entradas existentes. En caso de que haya una coincidencia, los contadores asociados a esa regla incrementan y se realiza una acción en concreto. Las reglas de flujo van ordenadas de mayor a menor prioridad, por lo que si no hay ninguna regla de mayor prioridad que coincida con los detalles del paquete recibido, el dispositivo intenta aplicar una regla predeterminada, la cual se encarga de encapsular el paquete y enviarlo al controlador para su posterior análisis.

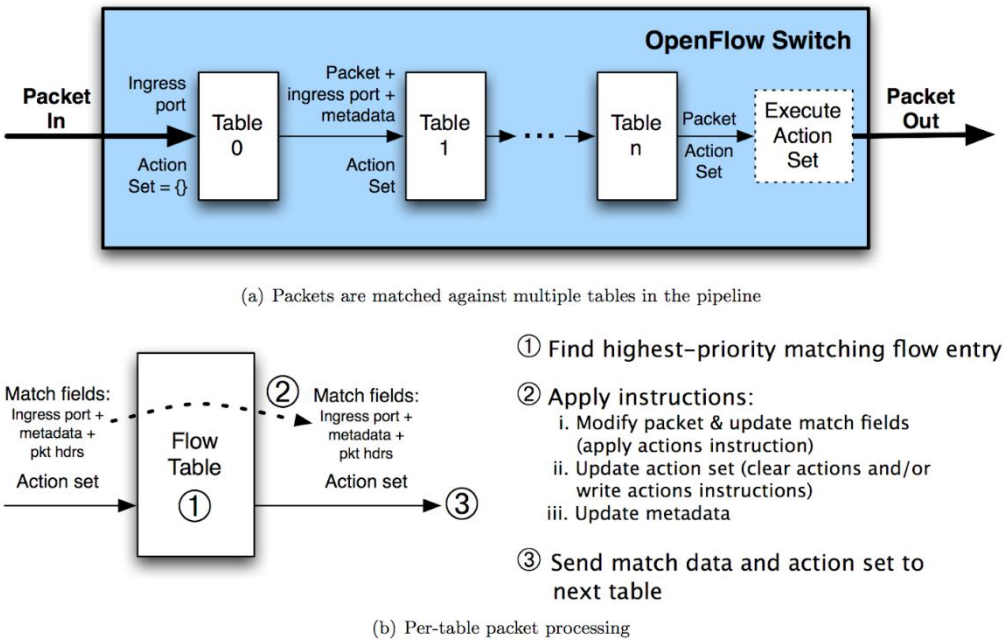

Figura 3. Tablas de flujo

2.4 Componentes

2.4.1 Controlador SDN

La Componente más importante de SDN. Ubicado en la capa de control, administra los protocolos y los recursos de la red. También gestiona el tráfico de los elementos subyacentes de la red a través de reglas de flujo. Proporcionándoles la información necesaria (acción) para saber cómo tratar los paquetes entrantes que coinciden con esa regla de flujo: enviar el paquete por uno o más puertos de salida, aplicarle QoS, eliminarlo o devolverlo al controlador si no hay ninguna coincidencia.
La comunicación con los dispositivos de red de la capa de datos se realiza a través de la Southbound API y con las aplicaciones de la capa superior a través de la Northbound API.

Con la finalidad de comparar algunos controladores SDN, hemos realizado una búsqueda y analizado su funcionamiento. A pesar de que no hay una clasificación para diferenciar los diferentes controladores, ya que el funcionamiento y las responsabilidades son similares. Es habitual utilizar como criterio la arquitectura de despliegue. Al comienzo SDN quería centralizar el plano de control, por lo que los controladores que se utilizaban eran solo uno. La arquitectura distribuida permite el uso de múltiples controladores dentro del dominio. En nuestro caso, escogimos los siguientes criterios para la elección del controlador: Documentación, interfaz, modularidad, lenguaje de programación y plataformas soportadas. Con estos criterios junto a la elección de software que escogiéramos sería lo que limitaría la elección. Finalmente nos quedaron los controladores Opendaylight, ONOS, Floodlight, Faucet.

2.4.2 Open vSwitch (OVS)

Un OVS es un conmutador virtual multicapa desplegado en entornos virtualizados, de código abierto Apache 2.0. Está diseñado para permitir la automatización del tráfico de la red mediante programación. Soporta interfaces y protocolos como Openflow, sFlow, NetFlow, CLI, 802.1ag, y múltiples tecnologías de virtualización, entre ellas VirtualBox y KVM.
Algunas de las funciones más importantes que tiene son: calidad de servicio (QoS), 802.1Q VLAN, tunneling (GRE y VXLAN, con soporte IPsec), IPv6, traffic policing, balanceo de carga.

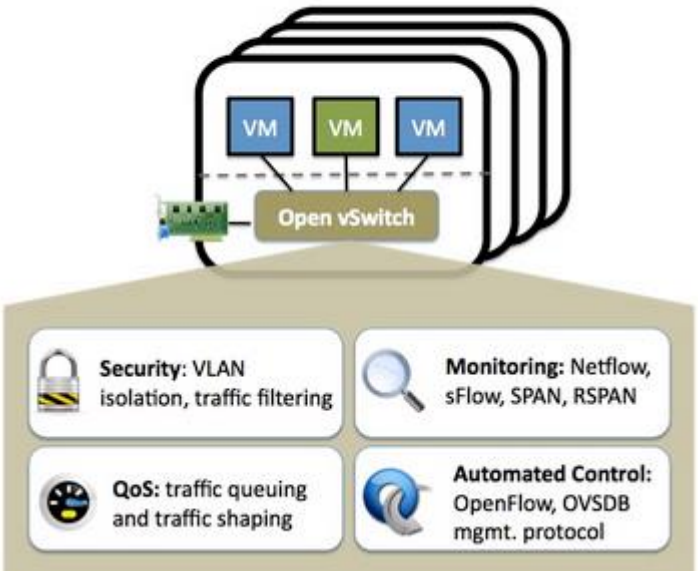

Figura 4. Open vSwitch

Los principales componentes y herramientas que se ejecutan en la aplicación de Open Vswitch:

- **ovs-vswitchd**: *daemon* que implementa el switch, junto con un módulo del kernel de Linux para la conmutación basada en flujos.
- **ovsdb-server**: servidor de base de datos que *ovs-vswitchd* consulta para obtener parámetros de su configuración. Esta base de datos puede ser modificada utilizando el protocolo OVSDB.
- **ovs-dpctl**: configura el módulo del kernel del switch.
- **ovs-vsctl**: consulta y actualiza la configuración de *ovs-vswitchd*.
- **ovs-appctl**: envía comandos para ejecutar los *daemon* de Open vSwitch.
- **ovs-ofctl**: consulta y controla los switches y controladores de OpenFlow.
- **ovs-pki**: crea y administra la llave pública de los switches OpenFlow.

Para realizar la conexión con el controlador, se debe configurar:

- La dirección IP del controlador.
- El número de puerto que escucha el controlador (6653).
- El protocolo de transporte que se utilizará para la conexión (TLS o TCP).

Capítulo 3. GNS3

3.1 Introducción

GNS3 es un software libre y de código abierto que proporciona una interfaz, que virtualiza dispositivos de hardware real para hacer funcionar diferentes programas de emulación como Dynamips, VMware o Qemu.
Usa imágenes del sistema operativo de Cisco (IOS) que funciona a través de Dynamips, emulaciones de Qemu de múltiples proveedores, e imágenes Docker para crear contenedores de software que automatizan diferentes aplicaciones. Posee una interfaz gráfica de usuario intuitiva y sencilla, por lo que es muy útil para emular, configurar y realizar pruebas y así solucionar problemas de redes virtuales y reales.

Para un rendimiento óptimo, GNS3 consiste en 2 componentes: Por un lado, el software cliente (Windows, MAC o Linux) y por otro lado un hipervisor (máquina virtual). La parte del cliente es un ejecutable que funciona de manera local en el sistema operativo anfitrión, y aunque puede funcionar sin el uso de un hipervisor, no se recomienda como componente del servidor, el servidor local. Ya que su uso es limitado y no permite el uso de topologías complejas.
La parte del hipervisor, como puede ser VMware, VirtualBox, Hyper-V, se recomienda cuando el sistema anfitrión usa Mac o Windows. Este contiene una máquina virtual que corre en el hipervisor, proporcionando mayor fluidez y más recursos. Además, permite añadir dispositivos de alto rendimiento, diferentes sistemas operativos en diferentes elementos.

Permite la emulación y configuración de diferentes dispositivos de red (Cisco, Fortinet, Juniper, Microsoft, Aruba) y hay varias categorías de elementos que se pueden instalar en el servidor de GNS3: firewalls, switches, routers, hosts, etc. También ofrece diferentes utilidades, herramientas, servicios, dispositivos, llamados en GNS3 *appliances*, para poder dar una amplia variedad de opciones al usuario: Generadores de tráfico, servidores, navegadores, herramientas de seguridad, etc, y acepta diferentes lenguajes de programación además de añadir herramientas útiles como son NAT o Cloud. Son las que permiten dar conectividad de red a la topología permitiendo decidir al que la diseña si pretende o no que tenga salida al exterior.
Dispone de una comunidad de usuarios de unos 800000 usuarios [6], lo que permite poder dar soporte de manera efectiva a los inconvenientes y bugs que aparecen. El uso de este software cada vez va creciendo más y haciéndose más extendido, ya que antiguamente las opciones eran mucho más limitadas. A este software se le puede dar un uso para diferentes finalidades, las opciones son muchas y muy variadas.

Para comenzar con GNS3 primero se debe crear un proyecto nuevo, que se guardará al cerrar el programa. *File>New blank project*: Una buena práctica, es el uso de diferentes proyectos para distintas pruebas de evaluación. Estos proyectos pueden ser abiertos o importados, en el caso de querer ejecutar el software en otro equipo.

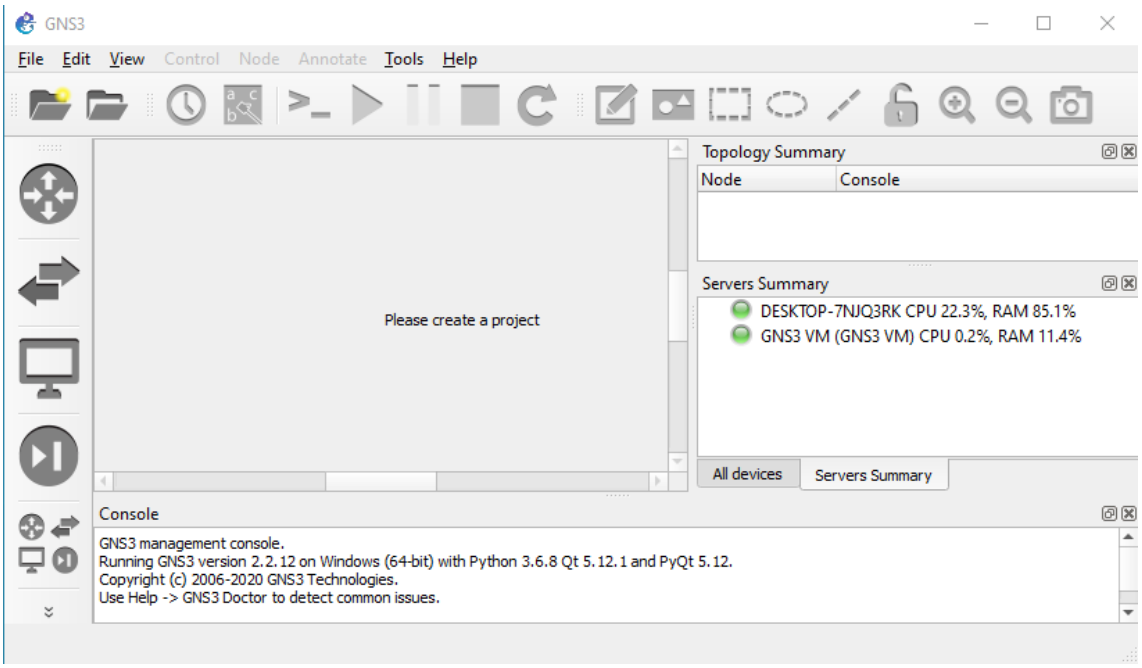

Figura 5. Pantalla inicial GNS3

Nota: *Se ha de tener en cuenta que cuando se cierre el software de GNS3, se perderá todos los paquetes, herramientas, o algunas configuraciones de los elementos que no sean persistentes.*

3.2 Servidor

Al realizar el despliegue de la red, los dispositivos necesitan ser alojados y ejecutados por un proceso del servidor. Se tiene una serie de opciones sobre la configuración del servidor: Servidor local de GNS3, **GNS3 VM** local, y remoto. Como servidor local o VM local, el cliente y el servidor se utilizan como procesos en el host anfitrión. La diferencia es que en este último se ejecuta utilizando un software de virtualización. El GNS3 VM remoto, la ejecución del software se realiza de forma remota.

Para nuestro laboratorio, vamos a utilizar GNS3 VM de manera local. Virtualizaremos nuestro servidor con el software VMware Workstation Pro. Antes de comenzar con GNS3, descargamos el archivo GNS3 VM con la extensión .OVA y lo importamos dentro de VMWare. A continuación, procedemos a configurar la máquina virtual. Se ha de tener en cuenta que tanto la versión del archivo GNS3 VM como la versión del software ha de ser la misma. Si no habrá problemas de conexión entre ellos.

Una vez ejecutado el software, habilitamos en *>preferences* GNS3 VM y configuramos el Setup de Wizard escogiendo la opción de correr las *appliances* en una máquina virtual.

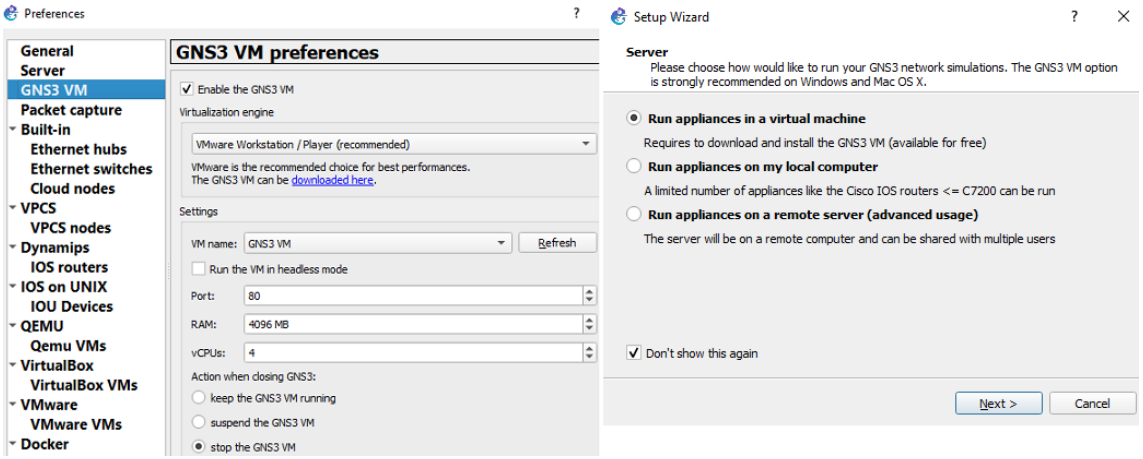

Figura 6. Preferences y Setup Wizard

Una vez realizada todos los ajustes, se podrá observar que tenemos activos los 2 servidores, tal como se ve en la figura 7.

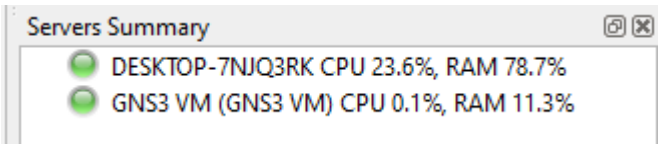

Figura 7. Servidores activos

El servidor lleva preinstalado y configurado Ubuntu (figura 8). Al ir añadiendo los elementos de la red al área de trabajo, el consumo del servidor va creciendo. Conforme más grande sea la red y los elementos requieran mayor uso de procesamiento, la carga de trabajo en el servidor será mayor. Por ello, se ha de escoger los elementos de la manera óptima. Teniendo en cuenta que hay algunos elementos que no pueden instalarse en el servidor local del cliente.

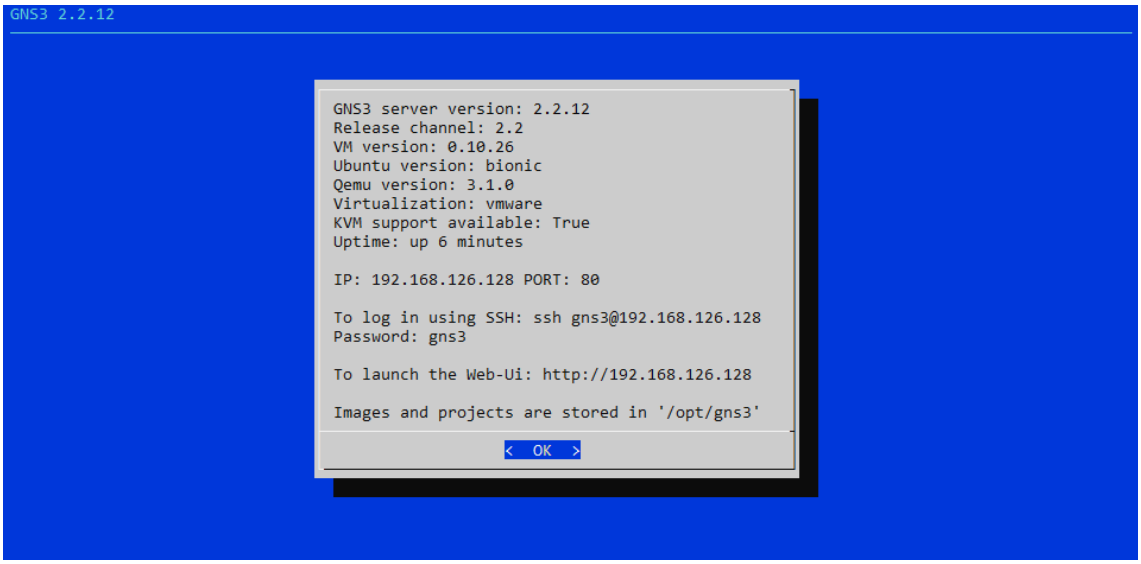

Figura 8. Servidor GNS3 VM

Al servidor se puede acceder mediante ssh desde el terminal del ordenador host:

ssh gns3@<ip_servidor>

y después de introducir la contraseña iremos al apartado Shell del servidor para comunicarnos con un elemento de la red virtual, ejecutar scripts, instalar imágenes de Docker e incluso conectar con el controlador de la red.

3.3 Dockers

Una de las ventajas del uso de la máquina virtual de GNS3 como servidor es que soporta la integración de imágenes de Docker. Una imagen Docker es un archivo, con múltiples capas, que se usa ejecutando un código de un contenedor Docker. Está formada a partir de instrucciones de una versión completa y ejecutable de una aplicación, basada en Kernel del sistema operativo anfitrión. Al ejecutar la imagen se convierte en una instancia/as del contenedor. La instalación de una imagen de Docker se realiza en el servidor para poder después cargarla. Una vez dentro, en el servidor introducimos los comandos:

gns3@gns3vm:~$docker images
gns3@gns3vm:~$docker pull <tag>

Nos permite ver las imágenes de Docker que tenemos y con un pull poder añadir la imagen al servidor. En *Edit>preferences>Docker containers* añadimos la imagen creada recientemente y se instalará dentro de GNS3. Y una vez ya añadido en el área de trabajo, ya podemos editar algunos settings y trabajar con él. Esto simplifica mucho el trabajo ya que se tiene un contenedor con todo lo necesario para hacer funcionar la aplicación.
Para nuestro laboratorio se añadió en primera instancia un Docker Ubuntu para correr sobre él el controlador. Actualizando repositorios, instalando herramientas, y añadiendo todo el software necesario para que se ejecutara correctamente. Posteriormente, se simplificó descargando de Docker Hub [17] una imagen del controlador que contenía todo el software necesario y ejecutándose en GNS3. De este modo, al iniciar el Docker ONOS, ya contenía lo necesario para que funcionara el controlador sin necesidad de nada más.

3.4 Elementos

Un aspecto positivo de GNS3, es que se puede importar máquinas virtuales externas desde VirtualBox o VMware. Otra herramienta útil es, descargar e instalar dispositivos como router, switches, e incluso dispositivos de alto rendimiento, mediante Cisco Virl. Equipos como IOSv, IOS-XRV, que puedes descargar a través de Cisco [12].
También se incluyen otras características como conexión de la red virtual a reales mediante *Cloud* o capturas de paquetes usando *Wireshark*. Esta herramienta es muy útil y tiene muchos usos de aplicación. Además de estar integrada perfectamente con GNS3. Ya que permite capturar en los diferentes links de la arquitectura.
Una vez dentro de GNS3, se tiene una serie de elementos por defecto para añadir al espacio de trabajo cuando se inicia, como: Switches (sin CLI), VPCS (host virtual), NAT o CLOUD.
Como vamos a utilizar GNS3 VM podemos añadir nuevos elementos. Estos elementos los podemos añadir desde *File>import appliance*, y se pueden cargar una vez descargado el archivo de la página de GNS3 [16].

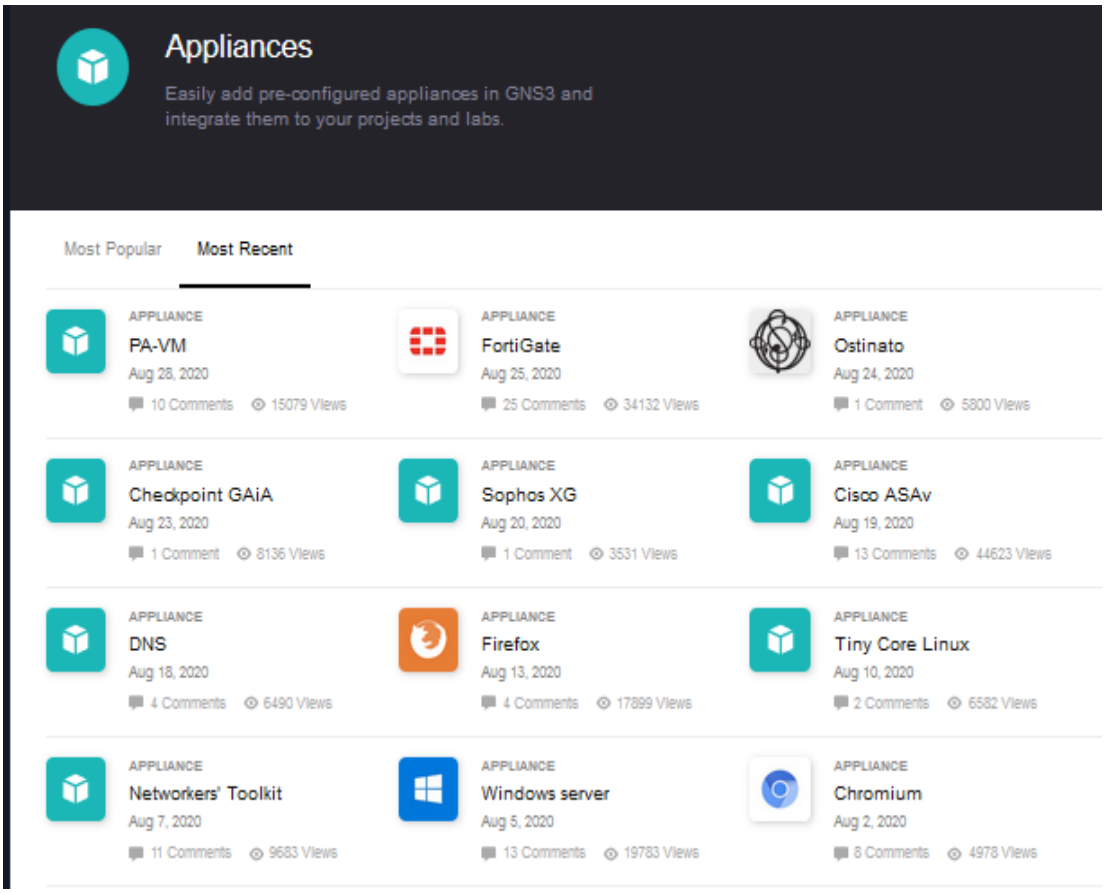

Figura 9. *Appliances*

Otra manera de descargar *appliances* es *File>+New template*. Separados por categorías: Firewall, Guest, Routers, Switches. Se descarga la *appliance* desde el servidor GNS3. Hay una amplia variedad de *appliances* o software, entre los que se encuentran, Cisco ASAv, FortiGate, Kali linux, Firefox, etc. Se van actualizando constantemente, para ir solucionando bugs y otros problemas. Al igual que cada vez van saliendo más *appliances* y de manera más rápida, hay elementos que contienen varias herramientas dentro del mismo. Como Ipterm, de SO Linux, que contiene diferentes herramientas dentro de él, como: net-tools, iproute2, ping, traceroute, iperf3, ssh client, tcpdump. Una vez iniciamos el elemento, podemos hacer uso de las diferentes herramientas a través de un comando. Estas herramientas se pueden utilizar poniendo su nombre en primer lugar y después usando los argumentos necesarios de la misma herramienta, dependiendo de lo que se pretenda hacer.

Dentro del terminal de Ipterm podemos ejecutar comandos como:

root@ipterm-1:~#iperf3 -s

root@ipterm-1:~#iperf -c 10.0.0.3 -u -b 10M

root@ipterm-1:~#ping 10.0.0.5

Otras *appliances* como Webterm tienen un tipo de consola vnc, por lo que puedes acceder por interfaz gráfica si el elemento lo permite. Es una manera muy práctica para algunos procesos que son más complejos por CLI.

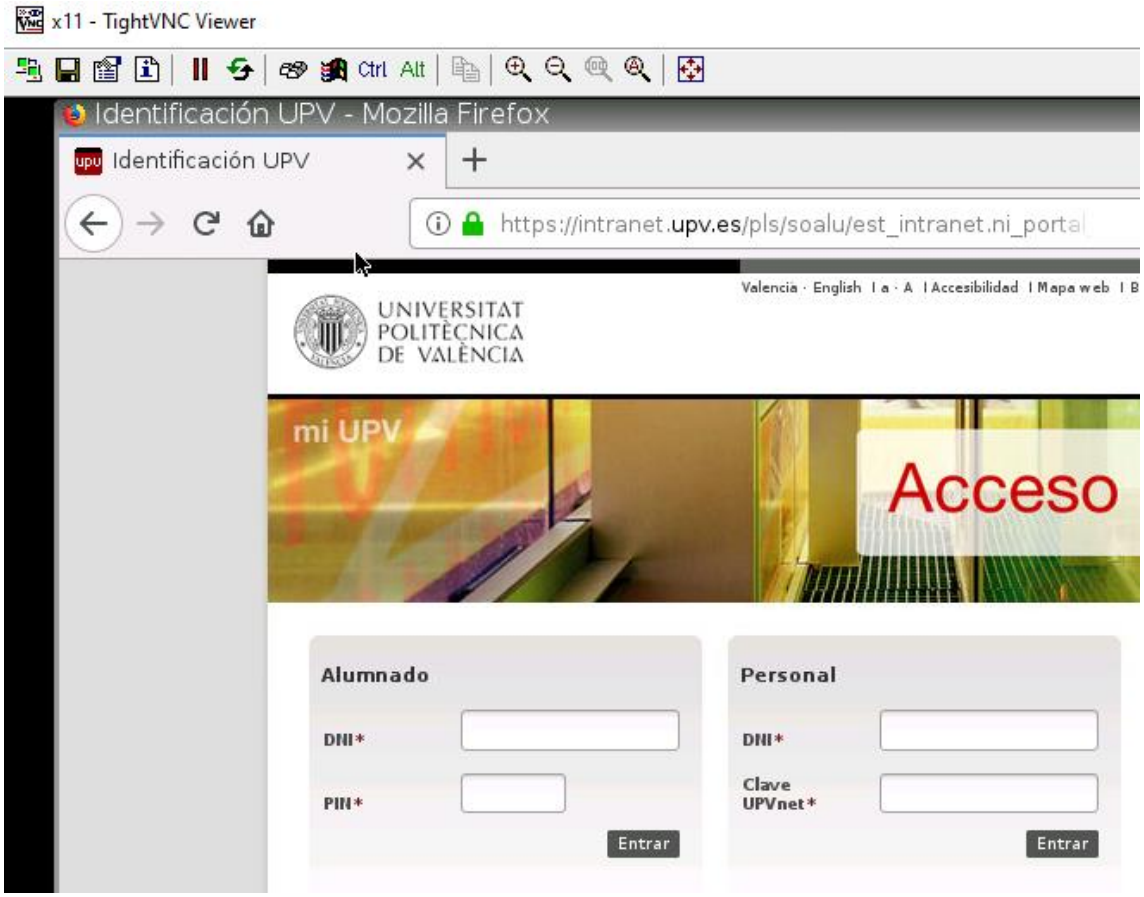

Figura 10. Webterm

Se pueden usar elementos que incluyan diferentes herramientas, o paquetes que incluyen una o más herramientas. *Appliances* que al actualizar el elemento descargan nuevas herramientas para utilizar. Existen también elementos como Dockers, donde puedes actualizar el propio elemento, descargar nuevas herramientas dentro de él, instalar software por comandos en ese elemento:

root@UbuntuDockerGuest-1:~#apt-get update

root@UbuntuDockerGuest-1:~#apt install software-properties-common

root@UbuntuDockerGuest-1:~#apt-get install wget

Otro elemento esencial es el link, que permite conectar los elementos de la red y crear las conexiones necesarias de la red.
Como ya hemos mencionado, la creación de máquinas virtuales con un SO específico, y el uso de aplicaciones en ese SO es posible con GNS3. Dentro de *Edit>preferences* tenemos la opción de VirtualBox o VMware, donde podemos crear máquinas virtuales que podemos añadir a GNS3, soportadas en el hipervisor. Se crearán y funcionarán como si de un sistema independiente se tratara. El inconveniente de ello es el uso de recursos que necesita para el funcionamiento del mismo. Por ello cuando hablamos de hacer un uso óptimo al escoger ciertos elementos, nos referimos a casos como este. Donde a menos que sea imprescindible, usaremos Dockers o elementos de acceso por CLI dónde los recursos necesarios no son tan elevados. No siendo lo mismo en elementos del tipo: Qemu, Dynamips o máquinas virtuales.

Un aspecto a tener en cuenta es el tipo de consola que usa el elemento, como por ejemplo el Solar-Putty de SolarWinds y que viene por defecto en los elementos que se acceden por terminal. Hay otros tipos de consola, como son telnet, vnc, http, https, etc.
Otro aspecto importante es la configuración del elemento en la red. Una vez el elemento está en el área de trabajo podemos editar algunos aspectos de él. Editar los adaptadores de red, la configuración de red o hacer algunos elementos persistentes.
Dentro del desplegable del elemento, *>Configure* permite acceder a diferentes settings como interfaces de red, tipo de consola, configuración de la red. También elegir qué directorios deseas que se guarden una vez cierres GNS3 para mantenerlos y así que sean persistentes y no se eliminen.
Otra de las opciones desplegables es *>Edit Config*, donde editas la dirección de red, asignándole de manera estática o dinámica.
Y la opción *>Auxiliary Console* en caso de que tengas ejecutando un proceso en un elemento y necesites acceder a un recurso o realizar varios procesos.

Elementos esenciales de nuestra red GNS3:

- **NAT**: Sirve para dar conectividad y es a la misma vez un servidor DHCP
- **Switch Ethernet**: Permite conectar dispositivos entre sí
- **Webterm**: Navegador web
- **Openvswitch Management**: Switch con protocolo openflow
- **VMWare** y **VirtualBox**: Hipervisores donde importar máquinas virtuales
- **Ipterm**: Host con varias herramientas, una de las cuales es Iperf
- Contenedor Docker del controlador ONOS

Capítulo 4. ONOS

El sistema operativo de red abierta (ONOS) es uno de los principales controladores de SDN de código abierto, por la fundación Linux. Comenzó en 2011, una colaboración entre el Centro de Investigación de Redes Abiertas (ONRC) de la Universidad de Standford, y ON.Lab. Proporciona el plano de control para redes SDN, administra los elementos de la red, ejecuta programas o módulos de software para suministrar servicios de comunicación a dispositivos finales. Fue creado para cubrir algunos requisitos de los operadores que buscaban soluciones de operador y que aprovechasen la economía del hardware y dieran la posibilidad de crear y desarrollar nuevos servicios de red dinámica con interfaces programables simplificadas. Así convertirse en la plataforma SDN para controlar las redes de proveedores de servicios. Permite la configuración y control de la red y todo ello en tiempo real. Elimina la necesidad de utilizar protocolos de enrutamiento o control dentro de una red. Permitiendo centralizar, todas las decisiones de la red sin variar la red ni alterar el plano de datos. Proporcionando al usuario la facilidad de crear nuevas aplicaciones sin que ello suponga ningún inconveniente. Integrando aplicaciones que actúan como controlador SDN modular. Donde gestiona y configura programas, hardware y servicios. Al ser modular permite extender o ejecutar según la intención del usuario. Proporcionando una robustez al sistema y escalabilidad. Para así conseguir un rendimiento óptimo del sistema y permitir que crezca tanto como se pretenda.

Al comienzo de este laboratorio, se usó Opendaylight. Un controlador con el que hubo ciertos problemas de compatibilidad de versiones a la hora de que se integrara y comunicara con otros elementos de nuestra red de GNS3. La interfaz gráfica estaba limitada y se había quedado desfasada. Por lo que el uso de versiones también era limitado. Lo que conllevaba a errores que ocurrían en versiones antiguas. No se encontró documentación suficiente para otras alternativas, por lo que se barajó el uso de una versión más actual para arreglar algunos inconvenientes, pero no fue suficiente para nuestro caso, por lo que se optó por examinar otros controladores. Al final se acabó escogiendo ONOS, motivada por varios aspectos. La interfaz gráfica funcionaba correctamente, y estaba actualizada, como las últimas versiones. Cuanto más actual sea la versión, supone la corrección de más posibles fallos a la hora de integrarlo con GNS3, lo que se traduce a tener menos problemas de compatibilidad con otros elementos de la red y menos errores. Así como en algunos sistemas, las actualizaciones no tienen grandes diferencias. En SDN, sistemas como ONOS o software como GNS3 tiene una elevada repercusión en las actualizaciones. Por ello, también utilizar una versión de GNS3 actualizada de software cliente y de GNS3 VM para poder sacar el máximo provecho. ONOS dispone de documentación amplia que nos podría ser útil. El aspecto modular del CLI, el uso de intent, y la programabilidad para comunicarse con el controlador fue lo que acabó por motivar su elección.

4.1 Funcionamiento y arquitectura

ONOS está diseñado como una arquitectura de tres niveles: El nivel 1 comprende los módulos relacionados con los protocolos que se comunican con los dispositivos de la red (Southbound). El nivel 2 compone el núcleo de ONOS y proporciona el estado de la red sin depender de ningún protocolo. El Nivel 3 comprende aplicaciones, aplicaciones de ONOS, que utilizan la información del estado de la red presentada por el Nivel 2. Este sistema está formado por varias capas que progresivamente abstraen de errores de los dispositivos y protocolos, para dejar a las aplicaciones como conjunto unificado y uniforme. Proporciona distintas funcionalidades, APIS, abstracciones, permiso de accesos, asignación de recursos, software, todo ello para que el usuario tenga acceso de una manera simplificada por medios como CLI, GUI, REST API o aplicaciones del sistema. ONOS puede ejecutarse en varios servidores y así reorganizar el uso de recursos, lo que le permite una cierta tolerancia frente a fallos. El software está en JAVA y proporciona una infraestructura distribuida sobre Apache Karaf OSGi. Permitiendo que los módulos se ejecuten e instalen dinámicamente.

La capa central (**Distributed Core**) es el núcleo de ONOS. Permite la separación física de los datos y las funciones de control, siendo el encargado de centralizar la red y dar acceso a las funciones de control. El núcleo está delimitado de los otros niveles. La interfaz dirección Sur (**SouthBound)** es una API de alto nivel a través de la cual el núcleo interactúa con la red. El núcleo utiliza una serie de adaptadores específicos de protocolo para llevar a cabo el control sobre la red. Estos adaptadores específicos de protocolo pueden ser Openflow, OVSDB, Netconf o incluso otros medios como CLI.

La API del SouthBound es el intermediario entre el núcleo que ejecuta órdenes de control y recibe datos de los diferentes proveedores de protocolos. Siendo el encargado de recabar los datos de interés para el controlador y que no se filtren los no relevantes tanto al propio controlador como a las aplicaciones. Por encima del núcleo se encuentra la interfaz dirección Norte (**Northbound**) y las aplicaciones. El núcleo a través de su API (Northbound) expone un conjunto de abstracciones a aplicaciones y servicios de red. Esto es una forma de acceso a la información de la red, desde abstracciones de bajo nivel como elementos de la red, hasta abstracciones de nivel superior como el gráfico de topología de red. Programar cambios en el estado de la red con objetivos de flujos e intenciones. Dentro de la capa central, tenemos un conjunto de subsistemas, cada uno encargado de realizar sus propios servicios. Algunos servicios dependientes de otros subsistemas, y que en su conjunto forman la API del núcleo de ONOS en la dirección norte. Hay subsistemas dentro del núcleo, que no están relacionados con el control de la red y son infraestructura distribuida. Y otros subsistemas, del dominio de control de la red. Por ello, el sistema modular, es lo que define poder ampliar, reducir o adaptar según la intención del usuario.

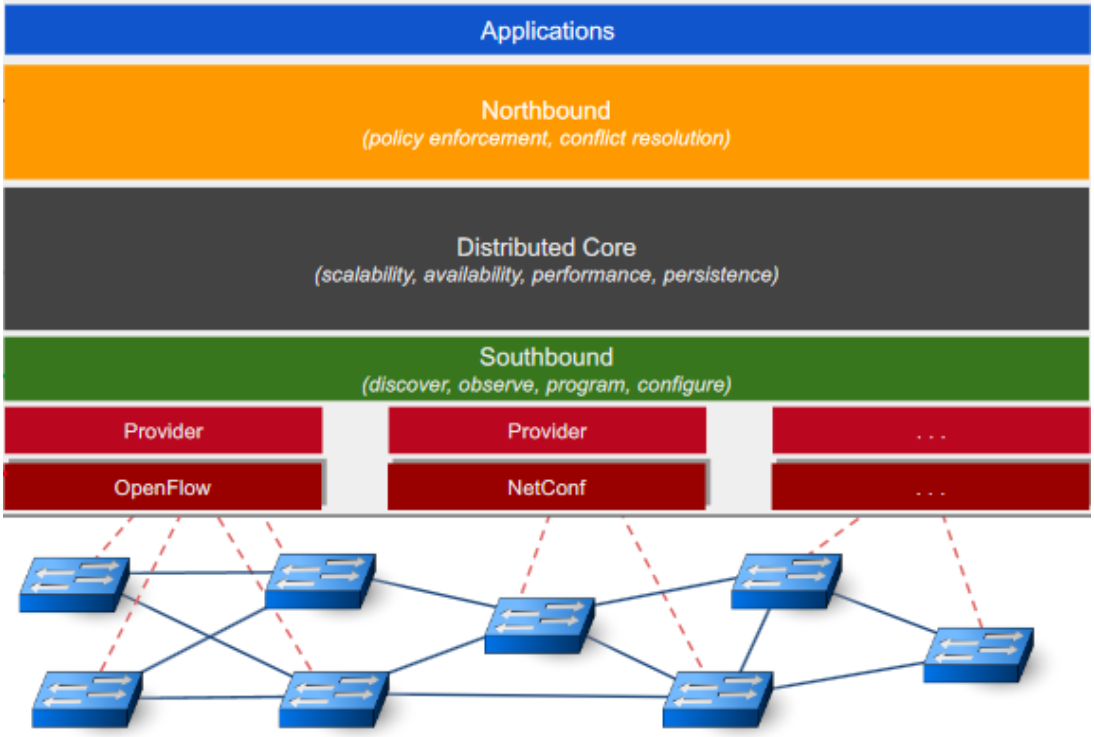

Figura 11. Arquitectura de ONOS

4.2 CLI e interfaz gráfica

ONOS posee diferentes formas de interactuar, como son CLI, GUI y REST API.
La interfaz de línea de comandos es una extensión del **CLI** de Karaf, con comandos propios, que permite agregar o quitar paquetes, programar aspectos del controlador, declarar intenciones o políticas sobre la red. Además de poder usar un conjunto de comandos disponibles con documentación al respecto, también permite su acceso por SSH.

Accederemos a ella mediante ssh en el shell del servidor:

ssh -p 8101 onos@<ip_controlador>

Figura 12. ONOS CLI

También proporciona una interfaz gráfica (**ONOS-GUI**) vía web que funciona en cualquiera de los principales navegadores web como Google Chrome o Firefox.

http://<ip_controlador>:8181/onos/ui

Al acceder, debemos autentificarnos con el usuario y contraseña:

User: onos Password: rocks

Una vez dentro, deberemos activar las aplicaciones que vayamos a utilizar, dentro del menú desplegable de la interfaz gráfica. Una de las herramientas más interesantes de la GUI, es poder ver el análisis de tráfico de la topología en tiempo real dentro del apartado Topology.

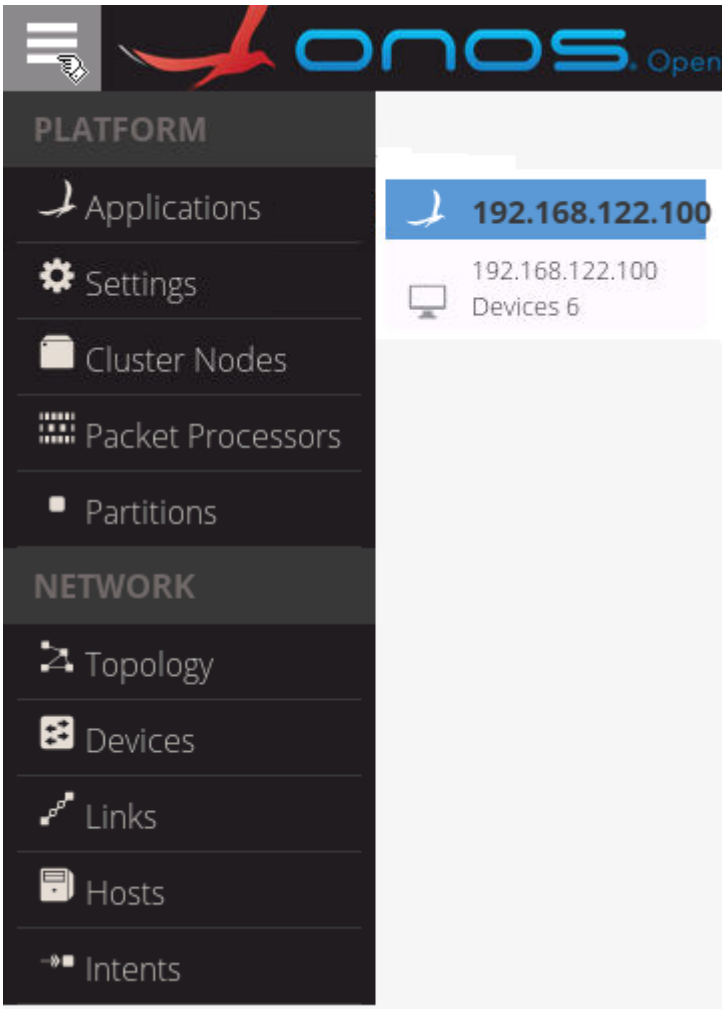

Figura 13. ONOS GUI

Otras de las características y herramientas que ofrece son:

- Resumen de las propiedades de la topología de la red.
- Panel con las características del controlador como ip, número de dispositivos o clústers.
- Barra de herramientas que proporciona botones para interactuar con la vista de la topología.
- Menú de navegación por las aplicaciones, dispositivos, enlaces, hosts, intents y flujos.
- Visualización de flujos por los caminos y sus velocidades.

4.3 Intent

Dentro de los diferentes subsistemas en el núcleo, hay uno que permite hacer intenciones dejando a las aplicaciones especificar ciertos deseos de control en la red. Es decir, declarar la intención en forma de política y no de mecanismo, formulando una solicitud al núcleo para alterar el comportamiento de la red. Este tipo de política llamado intent, es compilado por el núcleo de ONOS como una directiva. Esta intención es instalada y llevada a cabo mediante aprovisionamiento de enlaces de túneles, reservas de longitudes de onda o la que nosotros utilizaremos, la instalación de reglas de flujo en un conmutador. Este es el diagrama de los cambios de estado del intent tras instanciarlo por medio de una aplicación:

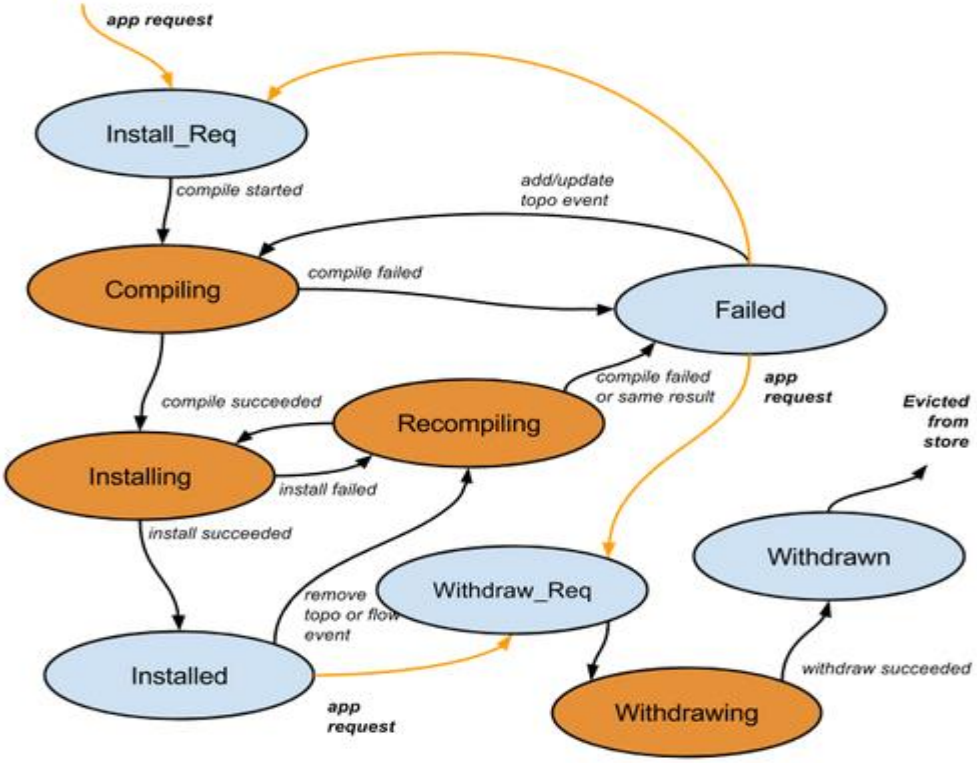

Figura 14. Diagrama de estados

Hay diferentes tipos de intent, que pueden ser expuestos de diferentes formas. Entre los que destacan en base a un criterio, una restricción, un elemento de red o una instrucción. Se identifican por el *appId* que proviene de la aplicación que ha realizado la solicitud y un único identificador del intent *Id* que se genera al crearlo.

- **add-host-intent**: Instala un intent con conectividad de host a host.
- **add-point-intent**: Instala un intent con conectividad de un puerto de entrada a uno de salida.
- **add-multi-to-single-intent**: Instala un intent con conectividad de varios puertos de entrada a uno de salida.
- **add-single-to-multi-intent**: Instala un intent con conectividad entre un puerto de entrada y varios de salida.

Se ha de tener en cuenta que las intenciones se deben declarar en ambos sentidos, ya que no son caminos unilaterales.

Para nuestro laboratorio, se comenzó utilizando un intent *SingleToMultiPoint* y viceversa. Ya que nuestra red disponía de dos entradas de tráfico por medio de distintos generadores, cuyos destinos eran diferentes. El tráfico de vídeo debería anteponerse al otro tráfico UDP. Tras observar que trataba los dos flujos de igual manera, a pesar de añadir ciertos parámetros para diferenciar uno y otro. Observamos otros problemas, como a la hora de inyectar tráfico por ambas partes de manera aleatoria, o un mirroring que se producía en el conmutador último antes de llegar al destino. Optamos por implementar un tipo de intent diferente para cada tráfico y así resolver el problema. Un intent *HostToHost* para el tráfico de vídeo, y un intent *PointToPoint* para UDP.

4.4 Aplicaciones

ONOS por defecto proporciona una serie de aplicaciones propias, algunas de las principales son:

Default Drivers, Host Location, ONOS GUI2, Control Plane Manager.

Algunas de ellas vienen activadas por defecto como es el caso: Default Drivers, ONOS GUI2.
La app de Forwarding permite la conexión entre todos los dispositivos de la red. La deshabilitamos, ya que crearemos caminos de conexión mediante intent.
Openflow indica al controlador que se comunicará mediante el protocolo Openflow con los switches.

Las aplicaciones se pueden importar y activar desde la interfaz gráfica o por comandos:

app activate org.onosproject.<app>

Instalaremos dos aplicaciones necesarias para nuestro laboratorio

app activate org.onosproject.openflow
app activate org.onosproject.proxyarp

La primera es para activar el protocolo Openflow en los conmutadores y la segunda es para la resolución de conmutadores vecinos.
También permite que se pueda interactuar con aplicaciones externas (OPA) programadas por el usuario. Estas aplicaciones, pueden dirigir intenciones, recabar estadísticas de flujos, y estipular objetivos lógicos. A través de un lenguaje de programación.

4.5 REST API

Proporciona una manera sencilla de interactuar con el sistema de ONOS mediante aplicaciones web y scripts, y así poder obtener información y configurar los elementos de la red. No suele recomendarse para aplicaciones de red de alto rendimiento, ya que los procesamientos pueden ser algo más lentos. El mayor uso que se le da es para gestiones del sistema o pruebas.
Orientada a dispositivos, flujos, intent, links, hosts, topología, aplicaciones y configuraciones.

http://<ip_controlador>:8181/onos/v1/

Para ello se usan los métodos de http GET, POST, y DELETE:

- Las solicitudes GET extraen información.
- Durante una solicitud POST, los datos (flujos, intent, etc) se pasan en formato JSON, los cuáles contienen diferentes criterios.
- DELETE permite eliminar la información requerida.

Capítulo 5. Laboratorio

Se va a crear una estructura de red de conmutadores (Open vswitch) con un único controlador centralizado (no clustering) en el mismo dominio, sobre la que irán 2 tipos de tráfico. Un tráfico de videovigilancia ip y otro tráfico, en el que será fundamental que el cliente reciba la transmisión de vídeo que ha solicitado al servidor. Ambos tráficos serán identificados por la dirección MAC origen y destino, ya que los dispositivos pertenecerán al mismo dominio de broadcast. En caso de que los dispositivos estuvieran en un dominio de broadcast diferente, como caso general, se podría utilizar la IP. Y como en el ámbito real, que existen otros tráficos que interfieren en la red, aquí lo representaremos como un tráfico de UDP.
Se va a tener en cuenta que el tráfico de vídeo es esencial y no debe perderse. El tráfico UDP puede o no perderse, según el escenario de la red. El flujo de videovigilancia se implementará a través de un streaming donde la fuente es un video, el cual mediante el protocolo RTSP se transportará a su destino. Se usará la aplicación VLC para el envío y recepción de dicho streaming. El envío del otro tráfico se hará a través de Iperf.

Una vez arrancado el GNS3, y teniendo activo el servidor GNS3 VM, se comienza creando la arquitectura de la red. Primero se añade una NAT para dar conexión a internet a los Open Vswitches y asignarles direcciones ip con DHCP, y un switch ethernet al que le añadiremos un número suficiente de interfaces de red.
Nota: Para nuestro Switch Ethernet necesitaremos 9 interfaces.

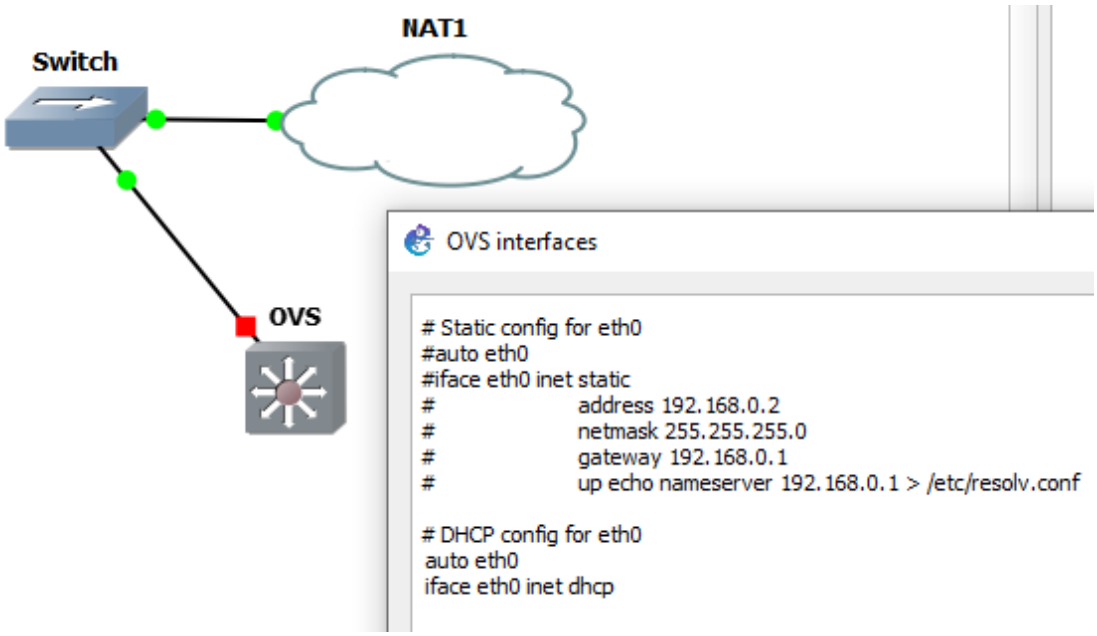

Figura 15. Edit Config del switch

Para asignarle una dirección ip o añadirle un link, primero debemos asegurarnos de que tenemos apagado el elemento. Una vez configurado las propiedades de red y enlazado, procedemos a encenderlo.
A continuación, conectamos el controlador ONOS al switch y un navegador web (Webterm) para conectarse a la interfaz gráfica del controlador y los 6 Open Vswitches Management.
Editamos el controlador con la dirección ip 192.168.122.100 y habilitamos el DHCP en el navegador.

Nota: Si el controlador es un Docker container, tarda un tiempo en instalar todos los paquetes necesarios del controlador. Se puede ver el proceso abriendo la consola.

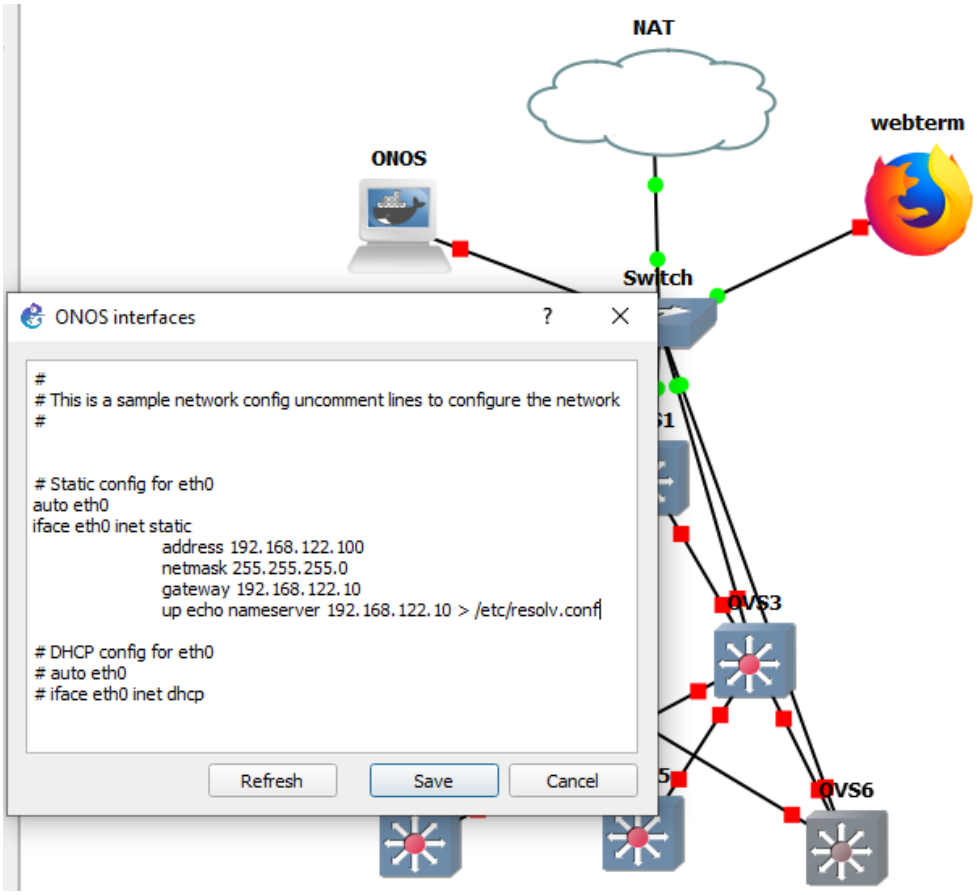

Figura 16. Edit Config del controlador

Los conmutadores seguirán una jerarquía de red basada en un modelo de tres capas, con un primer switch al que se conectan una máquina virtual Ubuntu como servidor de video y un Ipterminal con Iperf para inyectar tráfico. Dos switches intermedios y tres de acceso conectados a los hosts clientes.

Estos switches obtendrán una ip de la red 192.168.122.0/24 a través de su enlace ethernet 0 conectado a la NAT. Eth0 es el interfaz de gestión o mantenimiento, no incluido en el puente de los Open vswitches, y permitirá que el controlador pueda verlos y comunicarse con ellos mediante el protocolo Openflow. Para ello a cada conmutador hay que indicarle la dirección del controlador, junto con un cambio de id y MAC para poder identificarlos mejor.

ovs-vsctl set-controller br0 tcp:192.168.122.100:6653

ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000001

ovs-vsctl set bridge br0 other-config:hwaddr=00:00:00:00:00:01

Los cambios de id y MAC son para visualizar la arquitectura a través de la interfaz y así poder identificarlos.

Por último, se añaden 6 hosts, habiendo un cliente de vídeo (máquina virtual Ubuntu) y otro de tráfico (Ipterm) por cada switch de acceso. Estos hosts tendrán instalados VLC e Iperf.
Los clientes y servidores pertenecerán a la red 10.0.0.0/24

Se ha de tener en consideración que los enlaces tienen una velocidad de 10 Mb/s.
En alguna ocasión, puede que el valor proporcionado por la arquitectura de red en la interfaz gráfica no de la velocidad del enlace exacto.

Categoría	Tecnología
Emulador de red	GNS3 2.2.12
Switch	OpenvSwitches Management 2.4.0
Controlador SDN	ONOS 2.4.0
Protocolo de comunicación	Openflow 1.3
Hypervisor	VirtualBox, Vmware
SO Máquina Virtual	Ubuntu 18.04
Appliance end device	Ipterm
Generador de tráfico	Iperf
Aplicación de emisión de vídeo	VLC
Navegador web	Webterm

Tabla 1. Características red

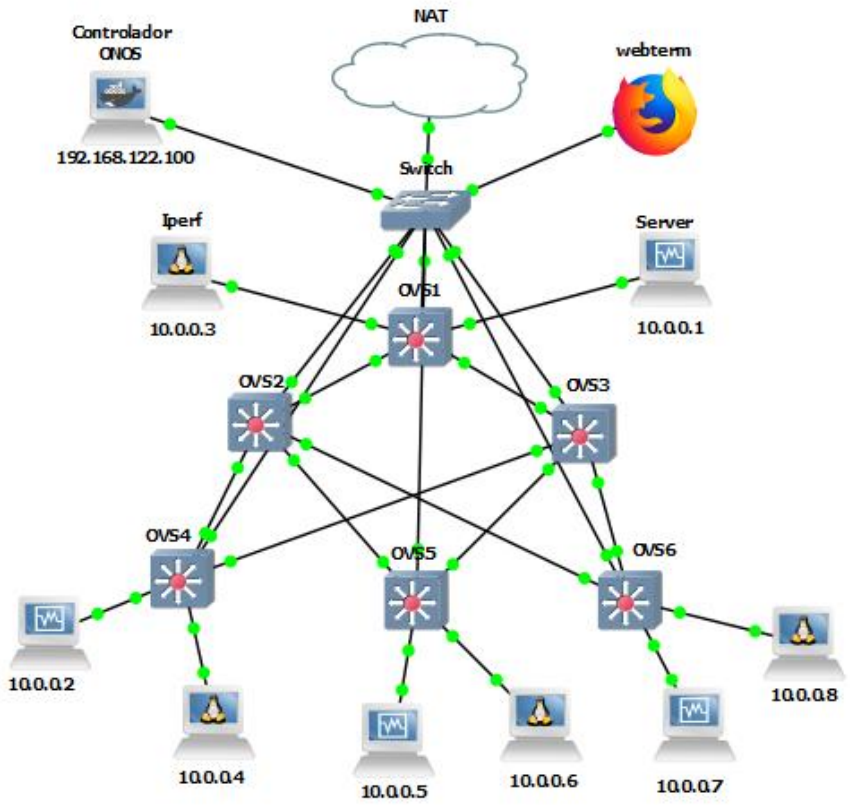

Figura 17. Topología de red en GNS3

Con todos los dispositivos encendidos y con las direcciones ip asignadas, se comprueba si el controlador ha descubierto y visualizado correctamente la red, tras haber activado las aplicaciones necesarias para nuestro laboratorio (arp, openflow). Además, debemos asegurarnos de que la aplicación Forwarding está desactivada.

Para ello se puede utilizar la línea de comandos del controlador o su interfaz gráfica.

Para poder examinar los dispositivos por línea de comandos:

```
onos@root > devices                                                                                    09:47:18
id=of:0000000000000001, available=true, local-status=connected 18s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=O
pen vSwitch, sw=2.4.0, serial=None, chassis=1, driver=ovs, channelId=192.168.122.172:46264, managementAddress=192.168.12
2.172, protocol=OF_13
id=of:0000000000000002, available=true, local-status=connected 12s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=O
pen vSwitch, sw=2.4.0, serial=None, chassis=2, driver=ovs, channelId=192.168.122.112:53108, managementAddress=192.168.12
2.112, protocol=OF_13
id=of:0000000000000003, available=true, local-status=connected 12s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=O
pen vSwitch, sw=2.4.0, serial=None, chassis=3, driver=ovs, channelId=192.168.122.99:57834, managementAddress=192.168.122
.99, protocol=OF_13
id=of:0000000000000004, available=true, local-status=connected 18s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=O
pen vSwitch, sw=2.4.0, serial=None, chassis=4, driver=ovs, channelId=192.168.122.224:34852, managementAddress=192.168.12
2.224, protocol=OF_13
id=of:0000000000000005, available=true, local-status=connected 13s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=O
pen vSwitch, sw=2.4.0, serial=None, chassis=5, driver=ovs, channelId=192.168.122.58:57850, managementAddress=192.168.122
.58, protocol=OF_13
id=of:0000000000000006, available=true, local-status=connected 13s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=O
pen vSwitch, sw=2.4.0, serial=None, chassis=6, driver=ovs, channelId=192.168.122.67:56808, managementAddress=192.168.122
.67, protocol=OF_13
onos@root >                                                                                            09:47:46
```

Figura 18. Devices/ switches identificados por ONOS

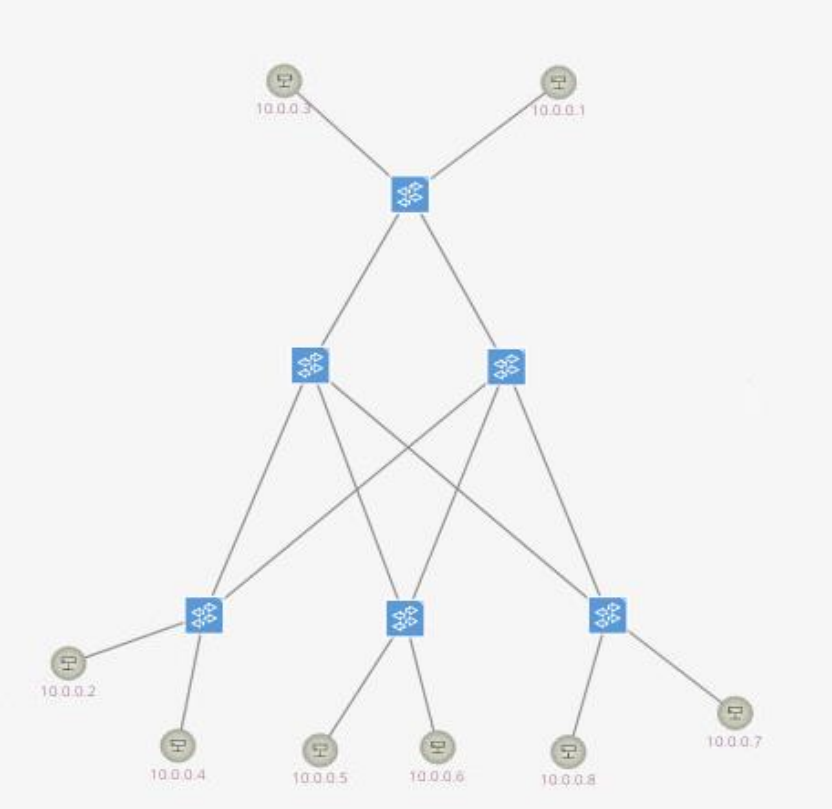

Figura 19. Topología de red por interfaz gráfica ONOS

ONOS solo identifica la red de openswitches y hosts, por lo que en ningún caso los enlaces de mantenimiento ni los dispositivos conectados aparecerán en la GUI ni en la CLI. Previamente, hemos indicado a los conmutadores que se conectaran al controlador.

Podemos examinar los hosts y sus características por línea de comandos. Se debe tener en cuenta que cuando ONOS visualiza un host, establece un identificador formado por la dirección MAC seguido de *"/None*".

```
onos@root > hosts                                                                                                    11:28:58
id=02:93:62:1D:7A:89/None, mac=02:93:62:1D:7A:89, locations=[of:0000000000000006/3], auxLocations=null, vlan=None, ip(s)
=[10.0.0.7], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=08:00:27:54:0B:BC/None, mac=08:00:27:54:0B:BC, locations=[of:0000000000000001/3], auxLocations=null, vlan=None, ip(s)
=[10.0.0.1], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=08:00:27:D8:53:D5/None, mac=08:00:27:D8:53:D5, locations=[of:0000000000000004/3], auxLocations=null, vlan=None, ip(s)
=[10.0.0.2], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=56:DA:E7:9B:90:81/None, mac=56:DA:E7:9B:90:81, locations=[of:0000000000000005/3], auxLocations=null, vlan=None, ip(s)
=[10.0.0.6], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=8A:4D:CC:AA:B2:FD/None, mac=8A:4D:CC:AA:B2:FD, locations=[of:0000000000000004/4], auxLocations=null, vlan=None, ip(s)
=[10.0.0.4], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=C6:0C:D1:6D:F8:99/None, mac=C6:0C:D1:6D:F8:99, locations=[of:0000000000000005/4], auxLocations=null, vlan=None, ip(s)
=[10.0.0.5], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=D2:F6:BD:6D:38:5E/None, mac=D2:F6:BD:6D:38:5E, locations=[of:0000000000000006/4], auxLocations=null, vlan=None, ip(s)
=[10.0.0.8], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
id=E2:A5:60:44:FE:38/None, mac=E2:A5:60:44:FE:38, locations=[of:0000000000000001/4], auxLocations=null, vlan=None, ip(s)
=[10.0.0.3], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.provider.host, configured=false
```

Figura 20. Hosts identificados por ONOS

5.1 Escenario 1

En este primer caso, vamos a enviar vídeo y tráfico UDP al mismo switch con hosts, coincidiendo por el mismo camino y causando congestión en la red, debido a que se superará el máximo del ancho de banda de los enlaces (10 Mbps).

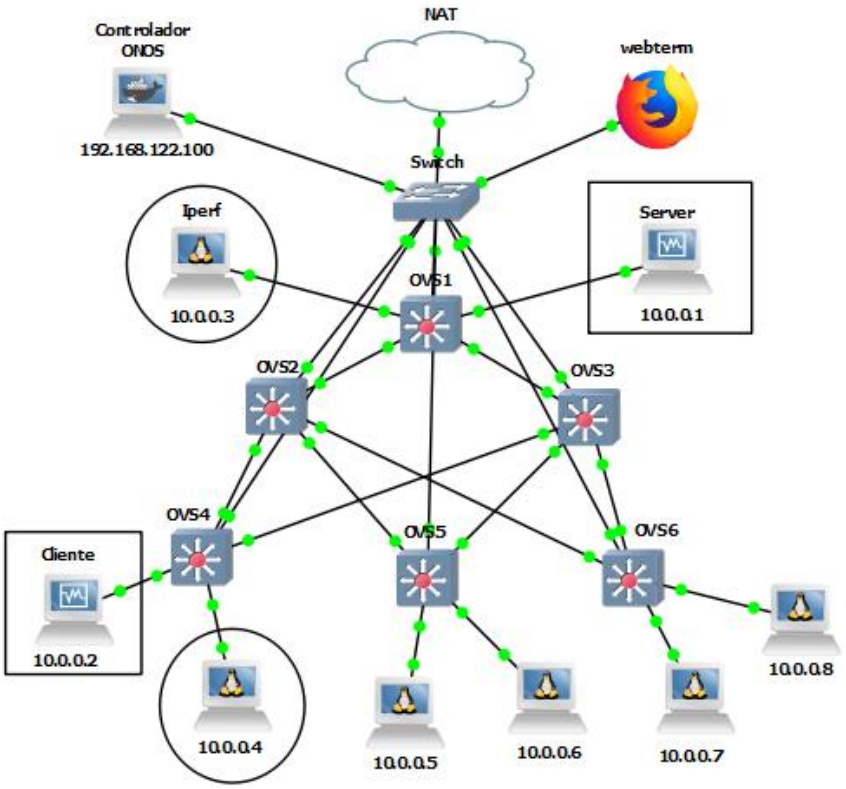

Figura 21. Topología de red para escenario 1

Como en el controlador no está activada la aplicación de forwarding que permite la comunicación entre los dispositivos, se van a instalar intent o reglas de flujos para indicar el camino a seguir de los paquetes.

En primer lugar, conectaremos el servidor de vídeo 10.0.0.1 con el cliente 10.0.0.2 mediante un intent *HostToHost* creado en la interfaz gráfica de ONOS. Se observa el camino a seguir en amarillo:

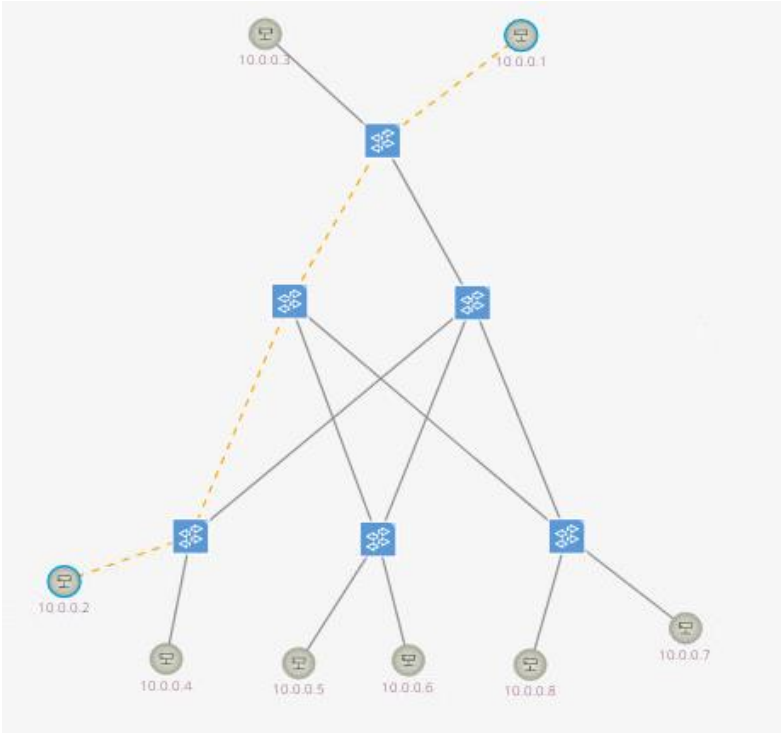

Figura 22. Intent *HostToHost* en GUI

El intent también se podría haber creado mediante comandos indicando los id de los hosts de origen y destino. Si no se indica la prioridad, por defecto es de 100.

add-host-intent 08:00:27:54:0B:BC/None 08:00:27:D8:53:D5/None

```
Id: 0x0
State: INSTALLED
Key: 0x0
Intent type: HostToHostIntent
Application Id: org.onosproject.gui
Leader Id: 192.168.122.100
Resources: [08:00:27:54:0B:BC/None, 08:00:27:D8:53:D5/None]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
Source host: 08:00:27:54:0B:BC/None
Destination host: 08:00:27:D8:53:D5/None
```

Figura 23. Intent *HostToHost* instalado

Tras haber instalado el intent *HostToHost* y saber por cuál switch intermedio pasarán los paquetes, se instalarán los intent *PointToPoint* que conectarán al inyector de tráfico con el otro cliente por el mismo camino que seguirá el vídeo. Estos intent solo se pueden instalar mediante comandos, o usando la REST API, y en total serán 6 (3 de ida y otros 3 de vuelta).

add-point-intent of:0000000000000001/1 of:0000000000000001/4

```
Id: 0x5
State: INSTALLED
Key: 0x5
Intent type: PointToPointIntent
Application Id: org.onosproject.cli
Leader Id: 192.168.122.100
Treatment: [NOACTION]
Ingress connect points and individual selectors
 -> Connect Point: of:0000000000000001/4   Selector: Inherited
Egress connect points and individual selectors
 -> Connect Point: of:0000000000000001/1   Selector: Inherited
```

Figura 24. Intent *PointToPoint* instalado

Con las 7 reglas de flujo instaladas, se comprueba que existe conexión entre los servidores y clientes y se empieza a enviar los tráficos para generar congestión.
Desde el servidor de vídeo se usa VLC para el envío de tráfico, indicando el protocolo a usar (RTSP) y el puerto:

```
gns3@gns3-VirtualBox:~$ vlc -vvv video720p.mp4 --sout '#rtp{sdp=rtsp://:8554/}'
```

Figura 25. VLC Server

Y dejando al cliente en escucha indicando la ip del servidor y el puerto:

```
gns3@gns3-VirtualBox:~$ vlc rtsp://10.0.0.1:8554/
```

Figura 26. VLC Client

Con Iperf se envía tráfico UDP a 10 Mbps destino el cliente 10.0.0.4 durante 50 segundos:

```
root@Iperf:~# iperf -u -c 10.0.0.4 -b 10M -t 50
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 49636 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-50.0 sec  59.5 MBytes  9.99 Mbits/sec
[  3] Sent 42467 datagrams
[  3] Server Report:
[  3]  0.0-50.1 sec  53.2 MBytes  8.91 Mbits/sec   0.739 ms 4482/42465 (11%)
```

Figura 27. Iperf

Según se van enviando ambos tráficos y la red se congestiona, podemos observar (figura 28) que el vídeo empieza a perder calidad, se pixela y en momentos no consigue avanzar. También en Iperf se observa que se han llegado a perder un 11 % de los paquetes.

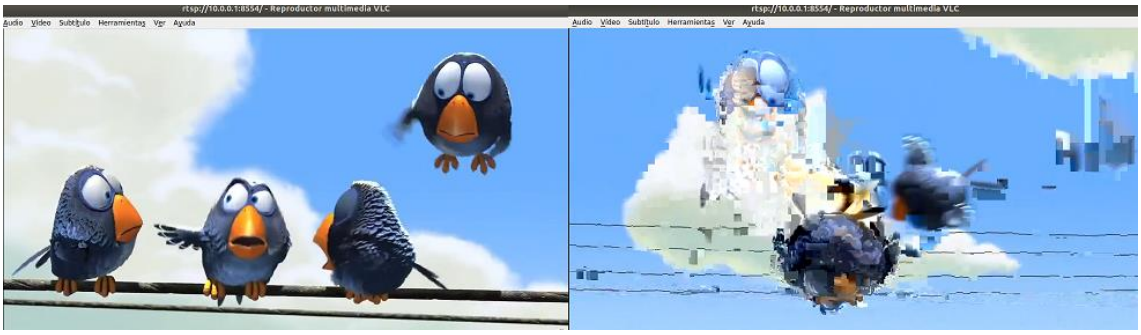

Figura 28. Vídeo normal vs Vídeo con la red congestionada

Mediante la interfaz gráfica de ONOS se pueden ver los flujos de tráfico que circulan por la red junto con sus velocidades en Mbps a tiempo real.

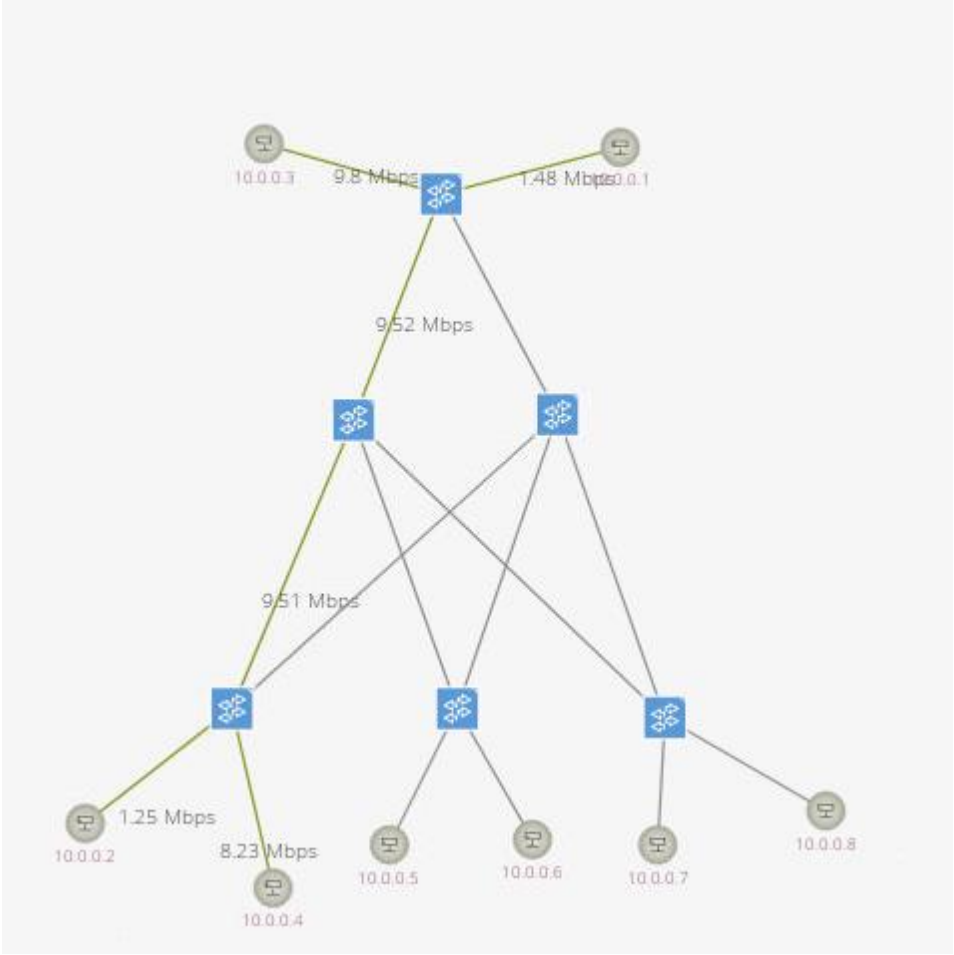

Figura 29. Ambos flujos circulando por la red

La suma de ambos flujos desde el switch 1 al switch 4 es superior a 10 Mbps, pero en el controlador no pasa de esa cifra al ser el ancho de banda de los enlaces, por lo que los paquetes se pierden.

5.2 Escenario 2

En el segundo caso, se pretende solucionar el problema de congestión del escenario anterior, mediante la creación de una aplicación externa conectada al controlador que cree caminos para redirigir el tráfico UDP en función de si puede haber congestión con el tráfico de vídeo y así dar prioridad al tráfico de videovigilancia.

La aplicación constará de varios scripts en lenguaje Python ejecutados en el shell del servidor de GNS3 y se comunicará usando la REST API del controlador, para poder mandar o leer datos de la red.

Se ha comenzado creando un script de configuración con todas las variables que se necesitarán y pueden cambiar una vez se haya apagado la red, como las MACs de los dispositivos o URLs.

Las MACs de origen y destino serán las que determinen qué camino se establecerá en cada tráfico, con la creación de cada intent y la diferenciación de los tráficos que circulen por la red.

```
#MACs

macIperf = "E2:A5:60:44:FE:38"

# video
mac1 = "08:00:27:D8:53:D5"
mac2 = "C6:0C:D1:6D:F8:99"
mac3 = "02:93:62:1D:7A:89"

# destino tráfico
macA = "8A:4D:CC:AA:B2:FD"
macB = "56:DA:E7:9B:90:81"
macC = "D2:F6:BD:6D:38:5E"

# URLs

url = 'http://192.168.122.100:8181/onos/v1/intents' #REST API a intents
```

Figura 30. Config.py

Para poder hacer solicitudes al controlador se usarán varias funciones python con métodos HTTP. Entre ellas el método GET para poder leer datos, POST para crear y DELETE para eliminar. A cada una de ellas se les pasará un link, la autenticación de ONOS y un json en el caso del POST.

```
import json
import urllib3

def json_get_req(url):
        http = urllib3.PoolManager()
        header = urllib3.util.make_headers(basic_auth='onos:rocks')
        response = http.request('GET', url, headers=header)
        return json.loads(response.data)

def json_post_req(url, json_data):
        data = json.dumps(json_data)
        http = urllib3.PoolManager()
        headers={'Content-Type':'application/json'}
        headers.update(urllib3.util.make_headers(basic_auth='onos:rocks'))
        r = http.request('POST', url, body=data, headers=headers)
        return r.data

def json_del_req(url):
        http = urllib3.PoolManager()
        header = urllib3.util.make_headers(basic_auth='onos:rocks')
        response = http.request('DELETE', url, headers=header)
        return json.loads(response.data)
```

Figura 31. Functions.py

El siguiente script contendrá los distintos Jsons de datos requeridos al hacer el POST. Estos Jsons son los intent *PointToPoint* necesarios para la creación de un camino según el tráfico de la red y al cliente al que se dirija.

Está formado por arrays de 6 jsons, uno por intent. Los caminos pasarán por el switch 2 o 3 y dirigiéndose a los clientes 10.0.0.4, 10.0.0.6 o 10.0.0.8. Cada intent contiene la MAC del host origen (inyector de tráfico) y destino, y también los puertos de entrada y salida de cada switch por donde irá el tráfico.

```
from config import *

json_2_A = [{
  "type": "PointToPointIntent",
  "appId": "org.onosproject.cli",
  "selector": {
   "criteria": [
      {
        "type": "ETH_DST",
        "mac": macA
      },
      {
        "type": "ETH_SRC",
        "mac": macIperf
      }
    ]
  },
  "treatment": {
    "instructions": [],
    "deferred": []
  },
  "constraints": [],
  "ingressPoint": {
    "port": "4",
    "device": "of:0000000000000001"
  },
  "egressPoint": {
    "port": "1",
    "device": "of:0000000000000001"
  }
},
```

Figura 32. Json Intent *PointToPoint*

Por último, la creación del código principal. Al iniciar la aplicación realizará una consulta al usuario para saber qué cliente solicita el vídeo (10.0.0.2, 10.0.0.5 o 10.0.0.7).
Tras conocer el destino del vídeo, le añadirá a la MAC el string "/None" formando así el id que usa el controlador para identificar a los hosts, el cual usará para la creación del intent *HostToHost* en formato json.
Este json se incluye en el método POST de la petición HTTP junto con la url de los intent de ONOS, dando lugar a la creación de un camino para enviar el vídeo.

```
import numpy as np
import json
from functions import *
from jsons import *
from config import *
import time

# Petición destino video

print("Destino vídeo: (a,b,c)")
dst = input()

if (dst == "a"):
        mac_cliente = mac1 + "/None"
if (dst == "b"):
        mac_cliente = mac2 + "/None"
if (dst == "c"):
        mac_cliente = mac3 + "/None"

# JSON host to host

json_host = {
        "type": "HostToHostIntent",
        "id": "0x2",
        "key": "0x2",
        "appId": "org.onosproject.cli",
        "priority": 100,
        "one":"08:00:27:54:0B:BC/None",
        "two":mac_cliente
}


json_post_req(url, json_host) #post host to host
print("Intent Host to Host añadido")

time.sleep(1)
```

Figura 33. Main.py parte 1

Como el paso intermedio del *HostToHost* creado es aleatorio, no se sabe por dónde pasará el vídeo, por lo que es necesario hacer una consulta a ONOS y ver en qué switch se ha instalado la regla de flujo.
Con la función creada para hacer GET, se obtendrán los flujos de los switches intermedios 2 y 3. Y se buscará la MAC del host destino del vídeo anteriormente usada.
El destino del tráfico UDP va a ser aleatorio, ya que lo importante es el tráfico del vídeo.
Si la MAC del cliente de vídeo aparece en el switch 2, se creará un camino por el switch 3 con los Jsons de los intent *PointToPoint* de ese switch hacia el destino que haya salido aleatoriamente. De la misma forma si la MAC está en el switch 3.

```
# Comprobación de camino mediante flujos

true = True # para que no de fallos
flows3 = json_get_req('http://192.168.122.100:8181/onos/v1/flows/of:0000000000000003')
flows2 = json_get_req('http://192.168.122.100:8181/onos/v1/flows/of:0000000000000002')


dst1 = np.random.randint(2)

# Intents point to point
print("Redirigiendo tráfico")

if mac_cliente[0:17] in str(flows3):
        print("Camino por switch 2")
        if (dst1 == 0):
                print("Destino 10.0.0.4")
                for i in range(6):
                        json_post_req(url, json_2_A[i])
        if (dst1 == 1):
                print("Destino 10.0.0.6")
                for i in range(6):
                        json_post_req(url, json_2_B[i])
        if (dst1 == 2):
                print("Destino 10.0.0.8")
                for i in range(6):
                        json_post_req(url, json_2_C[i])
if mac_cliente[0:17] in str(flows2):
        print("Camino por switch 3")
        if (dst1 == 0):
                print("Destino 10.0.0.4")
                for i in range(6):
                        json_post_req(url, json_3_A[i])
        if (dst1 == 1):
                print("Destino 10.0.0.6")
                for i in range(6):
                        json_post_req(url, json_3_B[i])
        if (dst1 == 2):
                print("Destino 10.0.0.8")
                for i in range(6):
                        json_post_req(url, json_3_C[i])

print("Proceso completo")
```

Figura 34. Main.py parte 2

Con todo el código creado, es necesario que no haya ninguna regla de flujo instalada previamente en la red para que no interfiera. Se eliminarán con la función de DELETE creada, aunque también se puede realizar directamente desde la CLI del controlador.

```
onos@root > remove-intent -p
Using "sync" to remove/purge intents - this may take a while...
Check "summary" to see remove/purge progress.
```

Figura 35. Eliminación de intent

Una vez comprobado que no hay ningún intent, ejecutamos la aplicación escribiendo *python3 main.py*. Se van a realizar tres pruebas en la que, en cada una de ellas, el cliente realiza una solicitud de vídeo y el servidor responde haciendo el streaming del video. Por otro lado, se recibirá de manera aleatoria el tráfico UDP, como tráfico no prioritario existente en la red y se observará en la interfaz gráfica de ONOS el paso de ambos flujos (figura 37, 38 y 39).

```
gns3@gns3vm:~/python$ python3 main.py
Destino vídeo: (a,b,c)
c
Intent Host to Host añadido
Redirigiendo tráfico
Camino por switch 3
Destino 10.0.0.4
Proceso completo
```

Figura 36. Proceso aplicación

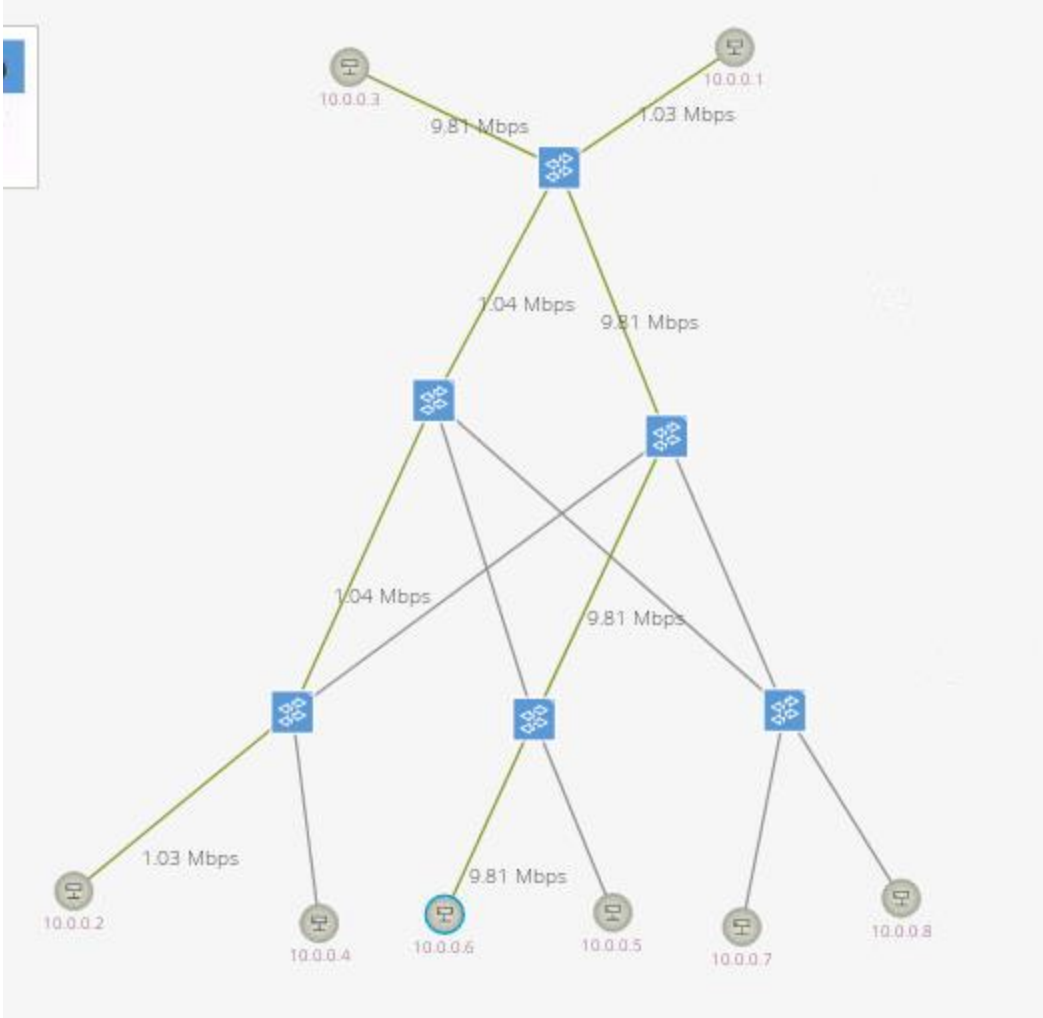

Figura 37. Destino vídeo 10.0.0.2

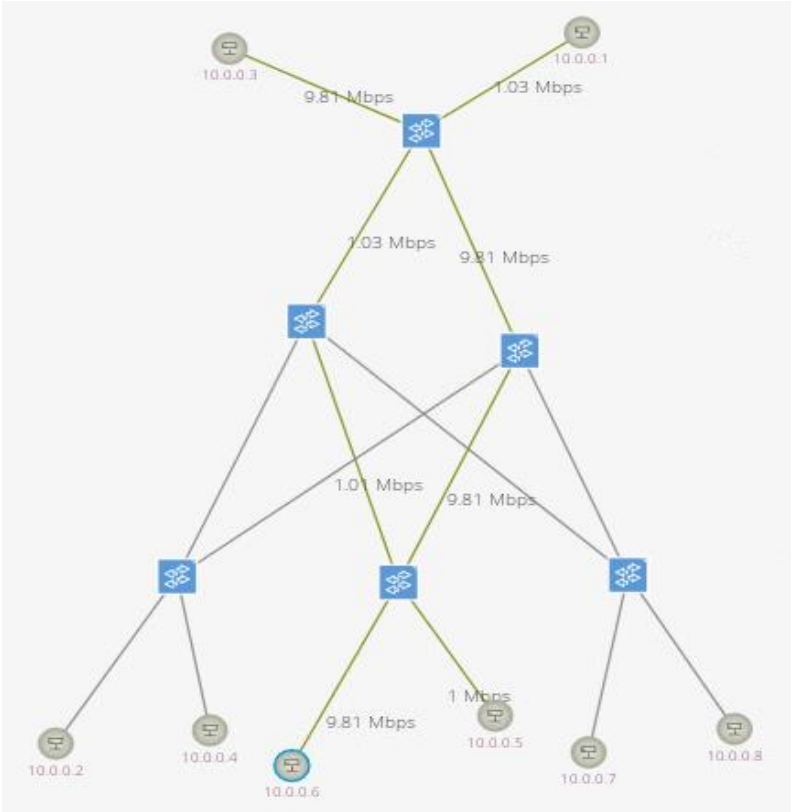

Figura 38. Destino vídeo 10.0.0.5

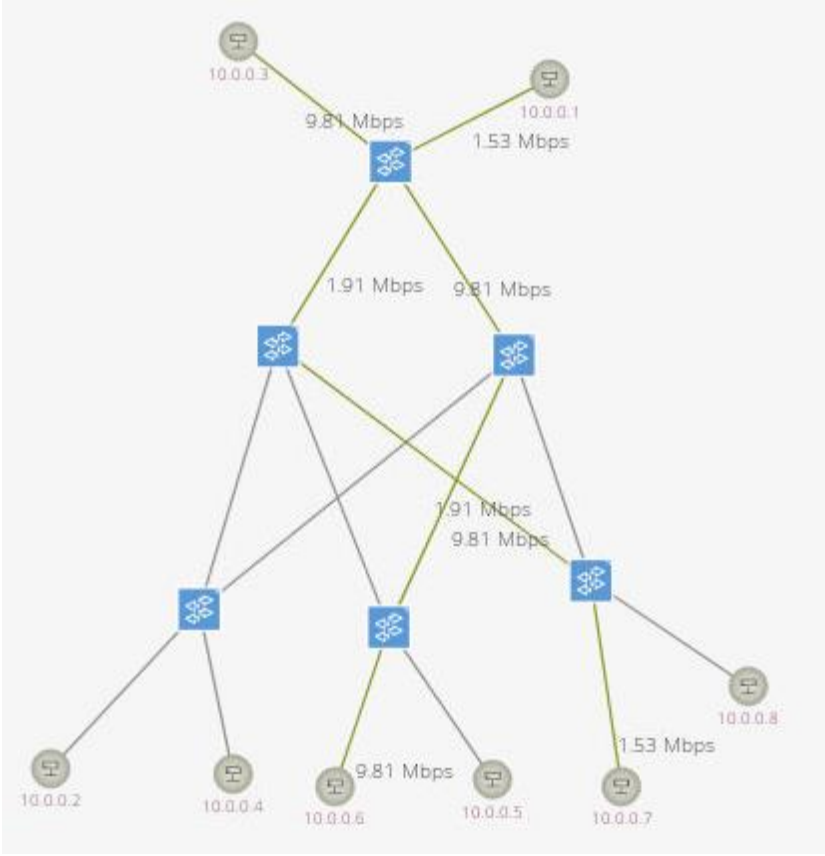

Figura 39. Destino vídeo 10.0.0.7

Tras la realización de las tres pruebas comprobamos el correcto funcionamiento de la aplicación, creando los dos caminos con intent y dirigiendo el tráfico a cada destino solicitado sin coincidir, ni congestionar la red.

5.3 Escenario 3

En este último escenario se va a realizar otra aplicación para recuperación de tráfico de videovigilancia en caso de fallo o caída de un switch, priorizando el vídeo sobre el resto de los flujos.

Esta aplicación detectará cuando hay un dispositivo que no está disponible y detendrá el tráfico UDP para que solo pase el vídeo por el otro camino, ya que una de las ventajas del intent *HostToHost* es que se reenruta automáticamente cuando unos de los dispositivos por los que ha creado el camino cae. En los *PointToPoint* este proceso no sucede.

El código de Python usará las funciones y configuraciones de la aplicación anterior, la cual es necesario que se haya ejecutado previamente para que circulen los flujos de tráfico.

Mediante el método GET irá obteniendo información de los dispositivos buscando la disponibilidad, cuando lea un *false* en el apartado *available* añadirá un flujo en el primer switch para eliminar el resto de tráfico.

```
while True:
        devices = str(json_get_req('http://192.168.122.100:8181/onos/v1/devices'))
        print("Leido")
        if "False" in devices:
                json_post_req('http://192.168.122.100:8181/onos/v1/flows/of:0000000000000001', flow)
                print("Flujo eliminado")
                break
        else:
                print("Esperando...")
                continue
```

Figura 40. Recovery.py

El flujo se añade en formato json y debe tener una prioridad superior a los flujos instalados anteriormente para que realice la coincidencia con él. Como parámetros debe aparecer el puerto del switch por el que entra el tráfico y la MAC del inyector. Al no poner nada en el apartado de *treatment*, realizará un DROP de los paquetes.

```
flow = {
        "priority": 200,
        "isPermanent": True,
        "deviceId": "of:0000000000000001",
        "treatment": {
        },
        "selector": {
                "criteria": [
                        {
                                "type": "ETH_TYPE",
                                "ethType": "0x800"
                        },
                        {
                                "type": "ETH_SRC",
                                "mac": macIperf
                        },
                        {
                                "type": "IN_PORT",
                                "port": "4"
                        }
                ]
        }
}
```

Figura 41. Flujo de Recovery.py

Mientras se envía tráfico desde los dos servidores, se apaga uno de los switches para simular un fallo. En este caso escogeremos de manera aleatoria el switch 2. ONOS tarda unos segundos en detectarlo y mostrarlo en la interfaz gráfica con un color diferente.

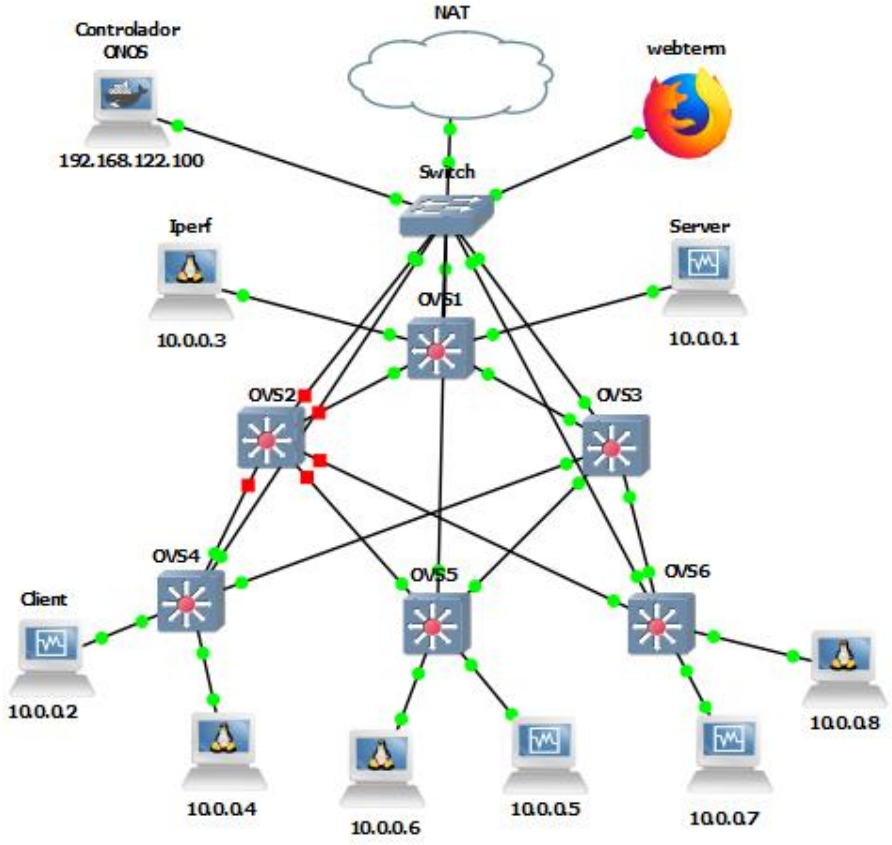

Figura 42. Switch caído en GNS3

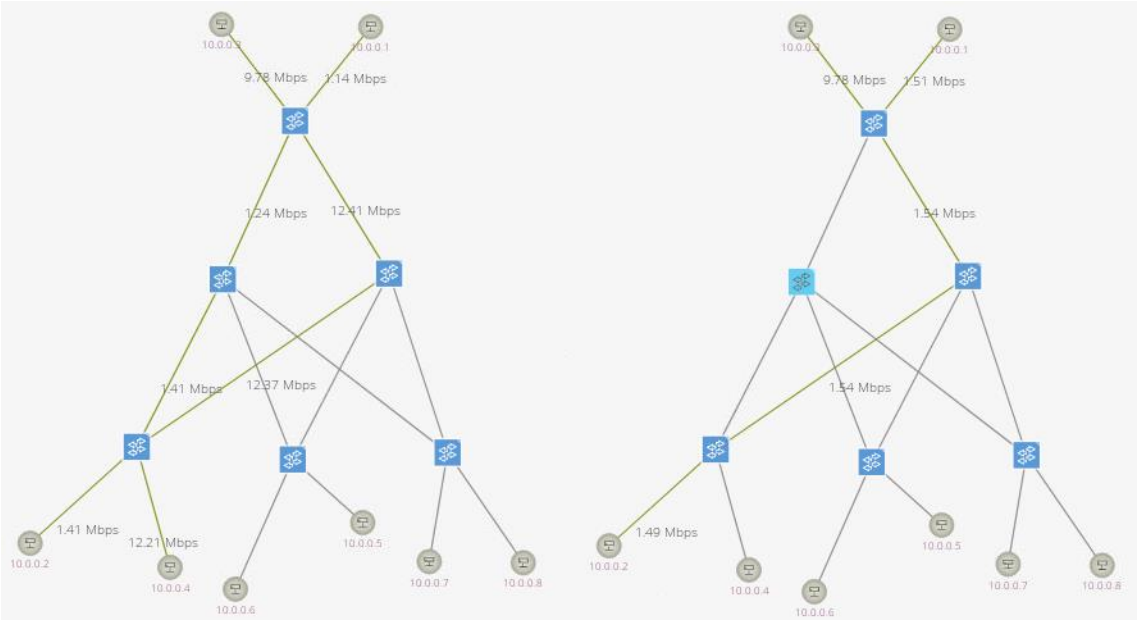

Figura 43. Flujos por la red normal vs Flujos con un switch caído

Como se ve en la figura 43, cuando en ONOS aparece el switch como no disponible, el flujo de vídeo (1,5 Mbps) se reencamina por el otro camino y el primer conmutador no deja pasar al resto de tráfico, por lo que la aplicación ha funcionado correctamente detectando el fallo e instalando el flujo.

Capítulo 6. Conclusiones y trabajo futuro

6.1 Conclusiones

GNS3 es un software que crece cada vez más y es capaz de ofrecer múltiples proveedores de servicios. El interés de las grandes empresas por el uso de GNS3 y la manera en la que colaboran o desarrollan aplicaciones es cada vez mayor. El uso de equipos de alto rendimiento de diferentes proveedores, para hacer pruebas y evaluaciones se está extendiendo en las empresas. Empresas que comienzan a ofrecer soluciones a problemas reales a través del software de GNS3. Además, como queda presente en este proyecto, la multitud de elementos que ofrece y el gran soporte de la comunidad y desarrolladores que actualizan bugs con cierta regularidad.

Se ha podido constatar, cómo el controlador SDN ONOS ha cumplido con las características necesarias de la red. Cumpliendo con todos los requisitos necesarios para este proyecto. Y ofreciendo unas características excepcionales centralizando y gestionando el control de la red, además de los recursos que dispone para crear directivas.

La interfaz gráfica de ONOS ha aportado datos relevantes para descubrir fallos de herramientas y comportamientos anómalos.

El uso de open vSwitch ha supuesto un elemento esencial en la red, ofreciendo una infraestructura programable y con muchos recursos. Además, ofrece la posibilidad de dar soporte a diferentes protocolos y a la interfaz de mantenimiento.

El generador de tráfico Iperf es una herramienta fiable que nos asegura el envío de los datos.

VLC sobre Ubuntu ha sido una aplicación estable para hacer el streaming del tráfico entre cliente-servidor.

El protocolo Openflow, que es el encargado de definir el camino de reenvío de paquetes, ha tenido un comportamiento estable y funcional.

Utilizar las redes SDN como herramienta de evaluación para dar soluciones al entorno real es lo que hace realmente atractivo este proyecto.

6.2 Trabajo futuro

El trabajo realizado ha sido todo un reto. No solo por el aspecto de la tecnología que se actualiza constantemente y el entorno de trabajo de las redes SDN que es cambiante, sino porque conforme se ha ido examinando, evaluando e implementando, había que configurar y comprobar si funcionaba todo correctamente. En caso de no resolver el problema, plantear otras alternativas. A pesar de haber requerido un tiempo para poder comprender y dar con una solución óptima, las posibilidades que plantea son muchas y muy variadas. Comenzando por aspectos como el de la seguridad, ya que disponemos de muchas herramientas para usar en el software GNS3, como análisis de vulnerabilidades, securizar el sistema ante posibles intrusiones, y muchas otras. Otro aspecto interesante es el de utilizar servidores, el cual puede ser muy útil para alojar en ellos servicios o aplicaciones. Un aspecto más podría ser el uso de Dockers para automatizar un servicio específico en la red o el uso de equipos de alto rendimiento.

Entre muchas de las posibilidades que se pueden hacer como trabajo futuro, se encuentra el uso de SDN para desarrollar otro tipo de servicios, y dar así soluciones a otros problemas de red. Donde las características y necesidades de la red difieran de las de este proyecto. Un ejemplo podría ser, desarrollar un servicio en el que los dispositivos estuviesen en diferentes redes y surge la problemática de que algunos tráficos de una red a otra pierdan paquetes. Con la particularidad de que los tráficos que circulen en ambas redes fuesen esenciales, y en ningún caso se pudiesen perder.

Capítulo 7. Bibliografía

[1] S. Subramanian and S. Voruganti, “Software-Defined Networking (SDN) with OpenStack”, Packt Publishing, October, 2016.

[2] B. K. Thapa and B. Dikici, “Reactive Forwarding Applications in ONOS,” no 2, 2016 IEEE Int. Conf. Commun. ICC, January, 2018.

[3] D. Sanvito, D. Moro, M. Gulli, I. Filippini, A. Capone, and A. Campanella, “ONOS Intent Monitor and Reroute service: Enabling plugplay routing logic,” 2018 4th IEEE Conf. Netw. Softwarization Work. NetSoft 2018

[4] Thomas D. Nadeau and K. Gray, “SDN: Software Defined Networks”, O'Reilly Media, Inc., August 2013.

[5] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, “Inter-controller traffic in ONOS clusters for SDN networks,” 2016 IEEE Int. Conf. Commun. ICC 2016.

[6] Getting Started with GNS3 | GNS3 Documentation. https://docs.gns3.com/docs/

[Online].

[7] I. Awan, M. Younas, and W. Naveed, “Modelling QoS in IoT applications” Proc. - 2014 Int. Conf. Network-Based Inf. Syst. NBiS 2014.

[8] ONOS - ONOS - Wiki. https://wiki.onosproject.org/

[Online].

[9] Open vSwitch. http://www.openvswitch.org/

[Online].

[10] What is GNS3 | CiscoPods. https://www.ciscopods.com/what-is-gns3/

[Online].

[11] Linux Advanced Routing & Traffic Control HOWTO. https://lartc.org/lartc.pdf

[Online].

[12] Cisco VIRL PE Online Documentation. http://virl.cisco.com/

[Online].

[13]AdvancedtrafficcontrolArchWiki.https://wiki.archlinux.org/index.php/advanced_traffic_control

[Online].

[14] Docs.openvswitch.org. https://docs.openvswitch.org/_/downloads/en/latest/pdf/

[Online].

[15] M. Putri, R. Muldina Negara and D. Dwi Sanjoyo, “Performance analysis of software defined network using intent monitor and reroute method on ONOS controller” pp. 2065~2073, Bulletin of Electrical Engineering and Informatics Vol. 9, No. 5, October 2020.

[16] Gns3.com. 2020. https://gns3.com/marketplace/appliances

[Online].

[17] Hub.docker.com. 2020. “Docker Hub” https://hub.docker.com/search?q=onos&type=image

[Online].

[18] GitHub. *GNS3*. https://github.com/GNS3

[Online].

[19] GitHub. *Open Networking Foundation*. https://github.com/opennetworkinglab

[Online].

[20] OpenDaylight. *Home - Opendaylight*. https://www.opendaylight.org/

[Online].

[22] GitHub. *Antlab-Polimi/Onos-Opa-Example*. https://github.com/ANTLab-polimi/onos-opa-example

[Online].