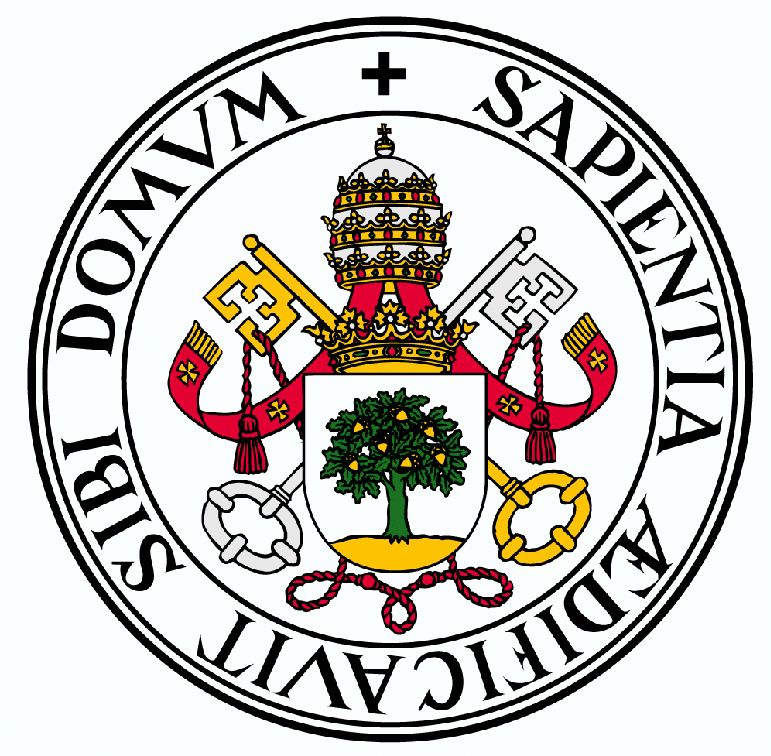
****

**Universidad de Valladolid**

Escuela Técnica Superior de Ingenieros de Telecomunicación (Valladolid)

Grado de Ingeniería en Tecnologías de Telecomunicación

**Avanzando hacia una red auto-adaptativa: simulación de redes definidas por software (SDN) mediante el simulador GNS3**

Alumno: Rubén Blanco Pérez

Tutores: Rubén Ruiz González

Jaime Gómez Gil

**DESCRIPCIÓN DEL TFG**

**TÍTULO**: Avanzando hacia una red auto-adaptativa: simulación de redes definidas por software (SDN) mediante el simulador GNS3

**AUTOR**: Rubén Blanco Pérez

**TUTOR**: Rubén Ruiz González, Departamento de Teoría de la Señal y

Comunicaciones e Ingeniería Telemática, E.T.S.I.

Telecomunicación, Universidad de Valladolid

**COTUTOR**: Jaime Gómez Gil, Departamento de Teoría de la Señal y

Comunicaciones e Ingeniería Telemática, E.T.S.I.

Telecomunicación, Universidad de Valladolid

**DEPARTAMENTO**: Departamento de Teoría de la Señal y Comunicaciones e

Ingeniería Telemática

**MIEMBROS DEL TRIBUNAL**

**PRESIDENTE**: José Fernando Díez Higuera

**VOCAL**: Jaime Gómez Gil

**SECRETARIO**: Isabel de la Torre Díez

**SUPLENTE**: Javier Manuel Aguiar Pérez

**FECHA** **DE** **LECTURA**:

**CALIFICACIÓN**:

Resumen

**INTRODUCCIÓN**

Las redes tradicionales requieren de cambios de configuración para cada elemento individual. El uso de redes definidas por software (SDN) permite disponer de un controlador que gestiona de manera simultánea todos los elementos de la red. Este enfoque permite adaptar los elementos de la red de manera dinámica, habilitando la optimización de la red a las necesidades concretas según varían en tiempo real. En este TFG se pretende simular, mediante el simulador gráfico de redes GNS3, diferentes esquemas SDN que permitan que la red se auto-adapte a las condiciones actuales. Con este trabajo se espera conseguir desarrollar nuevos mecanismos que puedan implementarse en entornos reales, avanzando así hacia el despliegue de redes auto-adaptativas.

**OBJETIVOS**

El objetivo principal de este trabajo es desarrollar aplicaciones SDN, para el controlador ONOS, que permitan gestionar diversas topologías de red creadas con el simulador GNS3. El controlador ONOS podrá cargar diferentes aplicaciones que gestionen de diferentes formas el comportamiento de dicha red, consiguiendo de esta forma una red auto-adaptativa.

**MATERIAL**

El principal material empleado en este TFG es un ordenador portátil. Dicho ordenador tiene instalada una distribución del sistema operativo GNU/Linux (Fedora 29). En dicho sistema operativo se encuentra instalado el siguiente *software*: (*i*) el simulador gráfico de redes GNS3, utilizado para la configuración de la red; (*ii*) el entorno de desarrollo integrado(IDE)Eclipse, utilizado para programar las diferentes aplicaciones que se le mandarán al controlador de la red. A su vez, dentro del simulador GNS3 es necesario disponer tanto de la imagen del switch Open vSwitch como del controlador ONOS, pues será elementos de red empleados en todas las aplicaciones desarrolladas en este TFG.

**METODOLOGÍA**

La metodología seguida en este TFG para alcanzar los objetivos propuestos ha consistido en la realización de manera iterativa de las siguientes fases: (i) documentación, (ii) diseño, (iii) programación, (iv) evaluación. Para cada una de las aplicaciones desarrolladas como para las pruebas piloto realizadas, este proceso se repitió en bucle hasta alcanzar unos resultados que cumpliesen con los objetivos inicialmente marcados.

**RESULTADOS**

En el presente TFG se han desarrollado 7 aplicaciones relacionadas con las redes definidas por *software*: (i) *severalping*, esta aplicación permite limitar el número de pings que pueden intercambiarse 2 hosts cualesquiera de una red. (ii) *statsshow:* Permite conocer la cantidad de datos que están enviando y recibiendo los hosts conectados a la red. (iii) *detectHost:* Permite conocer la MAC, el número de hosts y el puerto al que están conectados los hosts a los dispositivos controlados por el controlador. (iv) *detectHostBan*: Aplicación que permite banear el envío de tráfico por parte de cualquier host de la red cuando supera un cierto umbral. (v) *Vlan:* Aplicación que permite asignar a los hosts de la red una VLAN para poder crear subredes virtuales. (vi) *fwdBalanceo:* Esta aplicación permite realizar balanceo de carga cuando hay más de un enlace que comunica 2 dispositivos.(vii) *FakeDHCP:* Esta aplicación permite detectar servidores DHCP falsos cuyo objetivo es interceptar el tráfico de los hosts con fines maliciosos.

**CONCLUSIONES**

La principal conclusión que se puede extraer del trabajo realizado es que la gestión (centralizada o distribuida) mediante un controlador consigue resolver de manera eficaz problemas de redes que no son fácilmente resolubles sin usar un enfoque de redes definidas por software.

Las principales conclusiones secundarias son…

**Palabras clave**: red, GNS3, controlador, ONOS, aplicación, SDN, simulación

Abstract

**INTRODUCTION**

**OBJECTIVES**

**MATERIALS**

**METHODOLOGY**

**RESULTS**

**CONCLUSIONS**

**Keywords**:

Agradecimientos

*A nadie 😊*

Índice

Índice de figuras y tablas

Capítulo 1. Introducción

1.1 Objetivos del proyecto

Los objetivos del proyecto son los siguientes:

* Introducción a las redes SDN y principales protocolos asociados
* Estudio de los principales casos de uso de las redes SDN
* Revisión del estado de la técnica referente a SDN para lograr redes auto-adaptativas
* Propuesta de mecanismos SDN para avanzar hacia redes auto-adaptativas
* Simulación de los mecanismos diseñados en el simulador de redes GNS3

1.2 Organización de la memoria

La memoria está organizada en 6 capitulos. En el primero de ellos se detallan los objetivos del mismo, así como una introducción a las redes SDN, que incluye la historia y la arquitectura de las redes.

A continuación, en el capitulo 2 se detallan los primeros pasos que fundamentalmente se refiere a la instalación del diverso software necesario como GNS3 u ONOS.

En el capítulo 3, se explica como implementar y configurar la red con el GNS3 que se va a utilizar durante todo el transcurso del proyecto

En los siguientes capitulos que abarcan desde el 4 hasta el PONER NUMERO, se explican en detalle las aplicaciones realizadas. Para ello, en primer lugar se realiza una explicación teórica en aquellas que lo requieran, que introduzca los contenidos a tratar. A continuación, se divide la aplicación en diferentes partes que lo componen para poder clarificar mejor el funcionamiento de la aplicación en cuestión.

Finalmente, en el último capítulo se encuentran las conclusiones y líneas futuras.

1.3 Historia de las redes SDN

Las redes SDN *(Software Defined Networking)* permiten a los ingenieros responder más rápidamente a diferentes requisitos a través de un control centralizado. Esta tecnología surgió hace aproximadamente 20 años y su historia se puede dividir en 3 etapas.

En primer lugar, el periodo denominado redes activas que abarca entre el año 1995 y el año 2000. Las redes activas surgieron como una forma de evitar los procesos de estandarización por parte de la IETF (*Internet Engineering Task Force)* ya que era un mecanismo muy lento y no permitía avanzar tanto como se deseaba en esa época. Este tipo de redes están orientadas hacia el control de la red incluyendo una API (interfaz de programación) que expone recursos en los nodos de la red individuales, como por ejemplo el almacenamiento o las colas de paquetes y permite ejecutar código sobre ellos con el objetivo de poder procesar el flujo de datos. Fue el primer intento de hacer redes programables.

Una segunda etapa que transcurre durante los años 2001 y 2007 consiste en la separación de los planos de control y datos. El plano de control está destinado al tráfico que ocupan los equipos para gestionar, mantener y modificar el estado de la red, por ejemplo los protocolos de enrutamiento o la configuración de firewall están en este plano, mientras que el plano de datos se refiere a aquel destinado a los servicios, básicamente a hacer efectivo lo establecido en el plano de control, como por ejemplo el IP forwarding. El motivo fundamental de esta separación se debió a que el volumen de tráfico empezó a aumentar exponencialmente y los protocolos de enrutamiento convencionales dejaron de ser eficientes. Además, separar ambos planos permite también una independencia a la hora de desarrollo y, realizar un control desde un programa de alto nivel de toda una red lo que simplifica el debug.

La tercera etapa fue la aparición del protocolo Openflow que es considerado uno de los primeros estándares de las redes SDN y surgió a raíz de separar los planos. Su funcionamiento en detalle se mostrará en el apartado 1.4

Openflow es una tecnología de switching que empezó en la Universidad de Stanford y consiste básicamente en un protocolo de comunicación entre los controladores de la red, que deciden el encaminamiento y los switches que dispone la red. Cabe resaltar que los switches tradicionales no soportan este protocolo, por tanto fue necesario desarrollar unos nuevos que lo soportaran.

Algunos beneficios que introdujo Openflow son:

* Flexibilidad en el uso de la red, así como en la forma de operar sobre ella.
* El ser de código abierto, por tanto puede ser modificado de forma muy sencilla.
* Reducir el gasto de operación.
* Mejorar la integración de la red. Permite que la red sea controlada con un solo punto de vista.

1.2 Definición y arquitectura de las redes SDN

Las redes definidas por software se pueden definir como un paradigma en desarrollo con el objetivo de obtener un mayor rendimiento, flexibilidad y escalabilidad en la implantación de servicios de red. Todo ello conseguido, tal y como se ha comentado, gracias a la separación de los planos de control y datos.

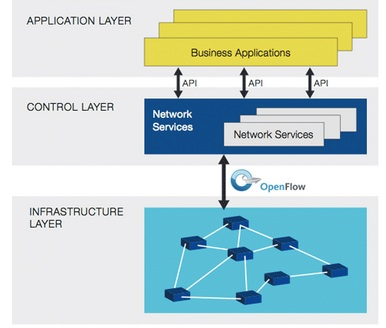
En este tipo de redes, no es necesario configurar separadamente cada dispositivo de encaminamiento (router o switch), sino que a través de un servidor central (controlador de la red) se proporcionan las reglas a través de las cuales se tratan los flujos de datos. Las características más importantes de este tipo de redes son:

* Programabilidad: La red se puede configurar a través de aplicaciones escritas en lenguajes de alto nivel como Java o Python.
* Gestión centralizada: A diferencia de una arquitectura tradicional en la cual cada router posee su propia inteligencia, la inteligencia de la red SDN está centralizada en el controlador que tiene una visión completa de la red y proporciona las instrucciones de control de flujo a los diferentes conmutadores.
* Agilidad: Las redes SDN proporcionan una respuesta rápida ante cambios en la red. Además, por otro lado, las aplicaciones están en constante desarrollo e implementación lo que permite cumplir con los objetivos comerciales muy rápidamente.

Todas las redes SDN están caracterizadas por tener estos elementos:

* Aplicación SDN: Son programas que comunican el comportamiento deseado de la red al controlador SDN.
* Plano de control: Es la capa intermedia en la que se implementan los controladores SDN. Su cometido es comunicar los eventos que ocurren en la red a las aplicaciones para que sean tratados y traducir la respuesta a instrucciones entendibles por los conmutadores.
* Controlador SDN: Es el cerebro de la red y encargado de traducir las peticiones de la aplicación y enviarlas a los switches. Todos los controladores de la red juntos forman el plano de control.
* Plano de datos: Compuesta por los diversos elementos de red como los switches o sistemas finales.
* Interfaz hacia el norte: Es la API que comunica la aplicación SDN y los controladores.
* Interfaz hacia el sur: Es el protocolo que comunica el plano de control con la capa de datos, generalmente es Openflow.

En la Ilustración 1 se puede ver un esquema sencillo de la arquitectura de una red SDN.



1.3 Funcionamiento de una red SDN

Para comprender mejor el funcionamiento de una red SDN vamos a poner un pequeño ejemplo que clarificará los conceptos. Para ello supongamos que tenemos una red formada por 2 hosts que denominaremos Host 1 y Host 2, intercomunicados por un switch que soporta el protocolo OpenFlow (Open vSwitch) y el controlador.

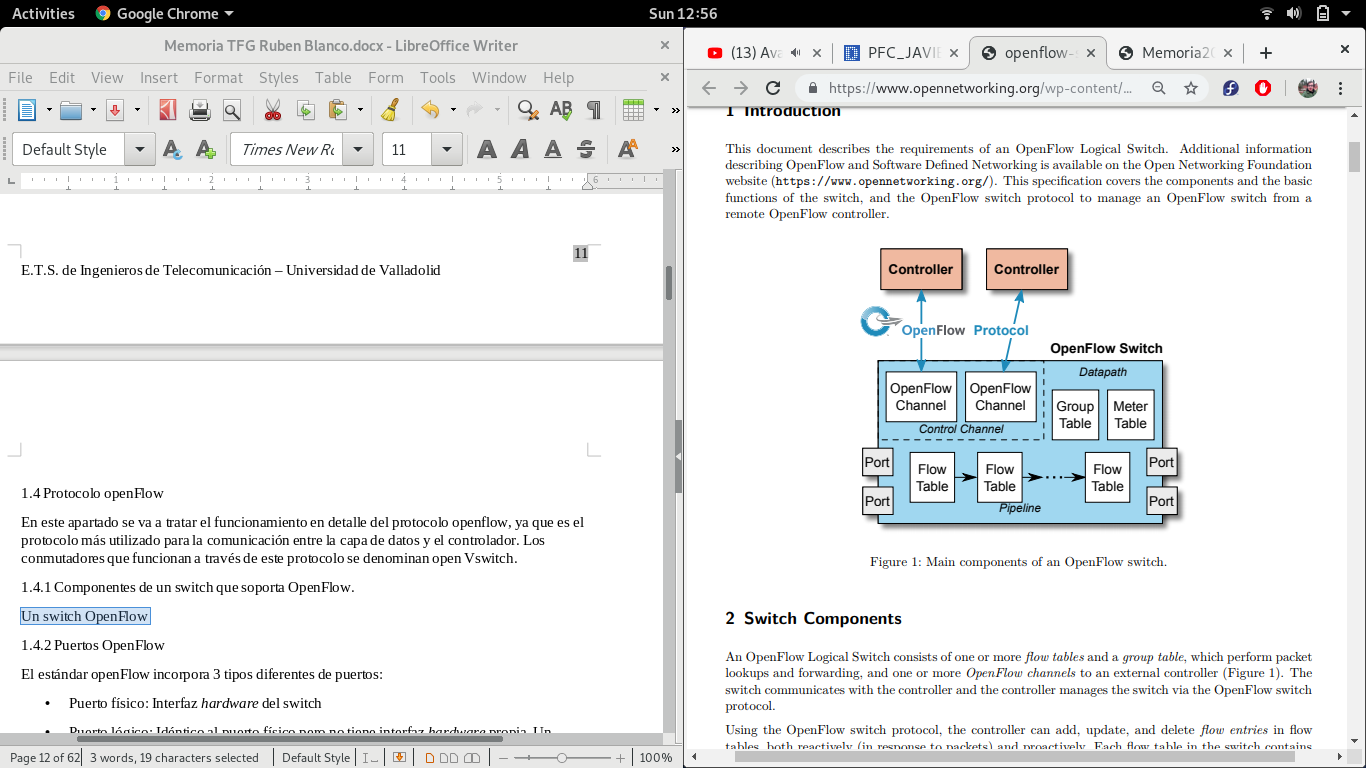
Supongamos que el Host 1 envía una trama al Host 2. La trama en primer lugar llega al open vSwitch. Si dicho switch posee una entrada de flujo para algún campo de capa 2, 3 o 4 se actúa según dicha regla, mientras que si el switch no posee una entrada en sus tablas envía el paquete al controlador para que este detalle cómo se debe conmutar dicha trama.

1.4 Protocolo openFlow

En este apartado se va a tratar el funcionamiento en detalle del protocolo openflow, ya que es el protocolo más utilizado para la comunicación entre la capa de datos y el controlador. Los conmutadores que funcionan a través de este protocolo se denominan open Vswitch.

1.4.1 Componentes de un switch que soporta OpenFlow.

Un switch OpenFlow tiene los componentes mostrados en la Ilustración 1



Estos elementos se van a ir detallando en los siguientes apartados:

1.4.2 Puertos OpenFlow

El estándar openFlow incorpora 3 tipos diferentes de puertos:

* Puerto físico: Interfaz *hardware* del switch
* Puerto lógico: Idéntico al puerto físico pero no tiene interfaz *hardware* propia. Un puerto físico puede contener varios puertos lógicos. La diferencia fundamental entre los puertos físicos y lógicos consiste en que los paquetes asociados a los puertos lógicos contienen un campo a mayores denominado *Tunnel-Id*. Cuando un paquete se recibe en un puerto lógico se envia al controlador.
* Puerto reservado: Utilizado para la comunicación con el controlador o para la conmuntación tradicional (sin hacer uso del protocolo openflow). A continuación se detallan los puertos reservados que obligatoriamente debe poseer un open Vswitch.
  + ALL: Representa todos los puertos que un switch puede usar para reenviar un paquete. Solo se puede utilizar como puerto de salida (no puede recibir tráfico)
  + CONTROLLER: Representa el puerto que conecta con el controlador. Puede ser usado tanto como puerto de entrada como de salida
  + TABLE: Representa el inicio de las *pipeline* de OpenFlow. Envía el paquete a la primera tabla de flujo para que pueda ser procesado.
  + IN\_PORT: Representa el puerto de entrada del paquete. Puede ser usado como puerto de salida con el objetivo de enviar el paquete por el mismo puerto por el que entró.
  + ANY: Valor especial utilizado cuando no se especifica ningún puerto NO CONSIGO TRADUCIR ESTO DE LA ESPECIFICACION
  + UNSET: Valor especial que indica que el puerto de salida no debe incluirse en el *Action-Set*. No puede ser usado ni como puerto de salida no como puerto de entrada.

Finalmente se detallan los puertos reservados que opcionalmente puede poseer un switch que soporte OpenFlow.

* + LOCAL: Representa la red local del switch
  + NORMAL: Representa el reenvio utilizando la forma tradicional (sin utilización del protocolo OpenFlow)
  + FLOOD: Representa todos los puertos menos el de entrada. Utilizado cuando se desea inundar una red (por ejemplo a la hora de utilizar el protocolo ARP)

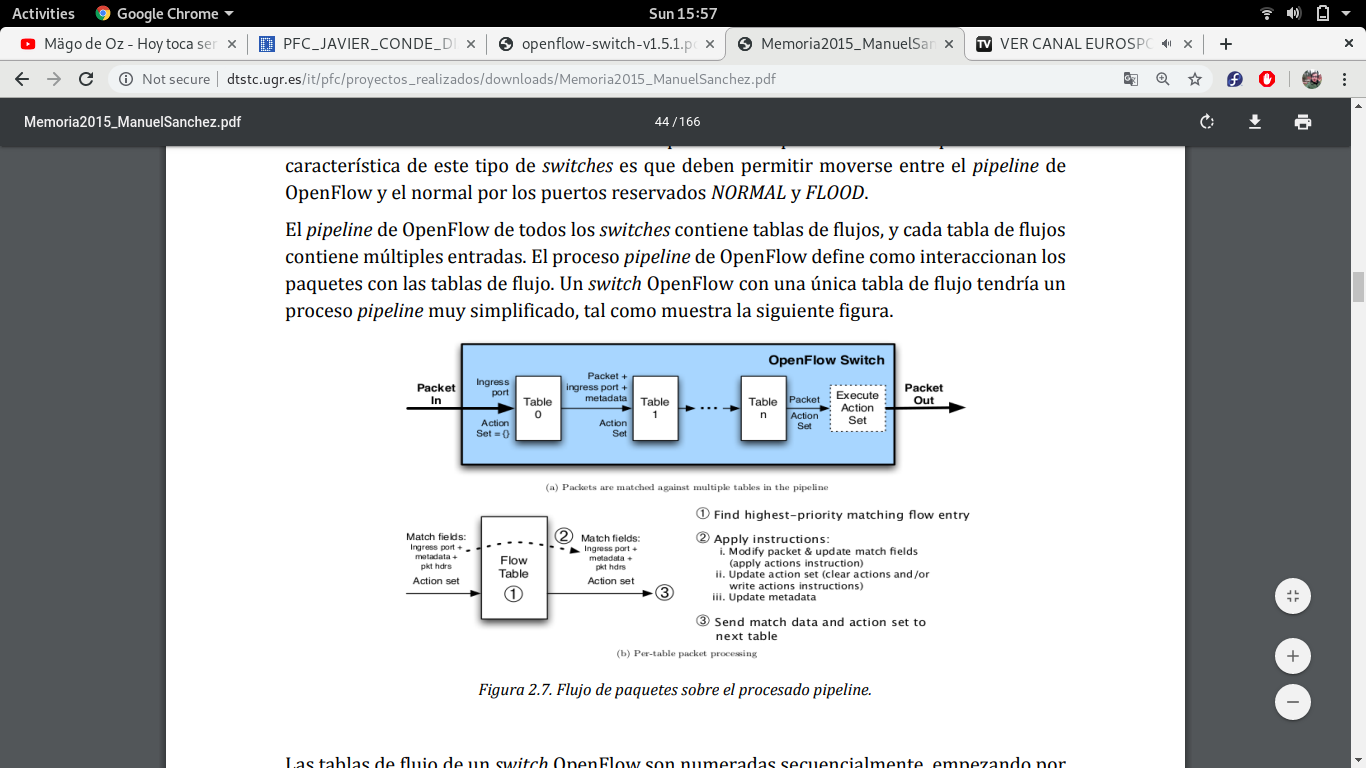
1.4.3Tablas de OpenFlow

En este apartado se detallan en primer lugar el proceso de pipeline de OpenFlow, los componentes de las tablas de flujo y de grupo, a continuación el mecanismo de *matching* y finalmente el gestionador de acciones.

1.4.3.1 Proceso de *pipeline*

Los switches OpenFlow pueden dividirse en 2 tipos: *OpenFlow-only* y *OpenFlow-hybrid.* Los primeros soportan únicamente las operaciones relacionadas como el protocolo OpenFlow y, por tanto, todos los paquetes son procesador por el *pipeline* de OpenFlow, mientras que los segundos admiten tanto las funcionalidades OpenFlow como las normales que aporta la conmutación Ethernet.

El *pipeline* de OpenFlow de todos los switches contiene tablas de flujo, y cada tabla de flujo contiene multiples entradas. Este define como interaccionan los paquetes con las tablas de flujo. En la Ilustración X se muestra como es el flujo de los paquetes cuando tenemos más de una tabla de flujo.



Estas tablas, como se puede observar, son numeradas secuencialmente empezando por el índice 0.

El proceso de *pipeline* siempre comienza por la primera tabla de flujo del procesamiento de entrada: en primer lugar se comprueba el *match* (ver apartado XXXX) entre el paquete y las entradas de flujo de la tabla 0. El resto de tablas de entrada se utiliza en función del resultado de la primera tabla.

Cuando un paquete es procesado por una tabla de flujo, se hace el proceso de *matching* con las entradas de dicha tabla. En caso de existir coincidencia se realizan las instrucciones correspondientes que contuviera dicha entrada. Estas instrucciones estan detalladas en el apartado XXXX, aunque se detalla que una de ellas es enviar el paquete a otra tabla de flujo, que ha de ser de índice mayor, en cuyo caso el procedimiento se repite. En caso de que no se encuentre esta instrucción el proceso de *pipeline* finaliza, y el paquete es enviado por el puerto correspondiente.

Otro caso que puede ocurrir es que ninguna entrada de la tabla de flujo coincida con el paquete, en cuyo caso se denomina *table miss*. El comportamiento de esta tabla depende de la configuración, pero algunos comportamientos habituales son descartar el paquete, enviarlo a otra tabla o directamente al controlador.

1.4.3.2 Tablas de flujo y entradas

Como ya se ha comentado, una tabla de flujo contiene diferentes entradas, cuya estructura es la siguiente:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Match Fields | Priority | Counters | Instructions | Timeouts | Cookie | Flags |

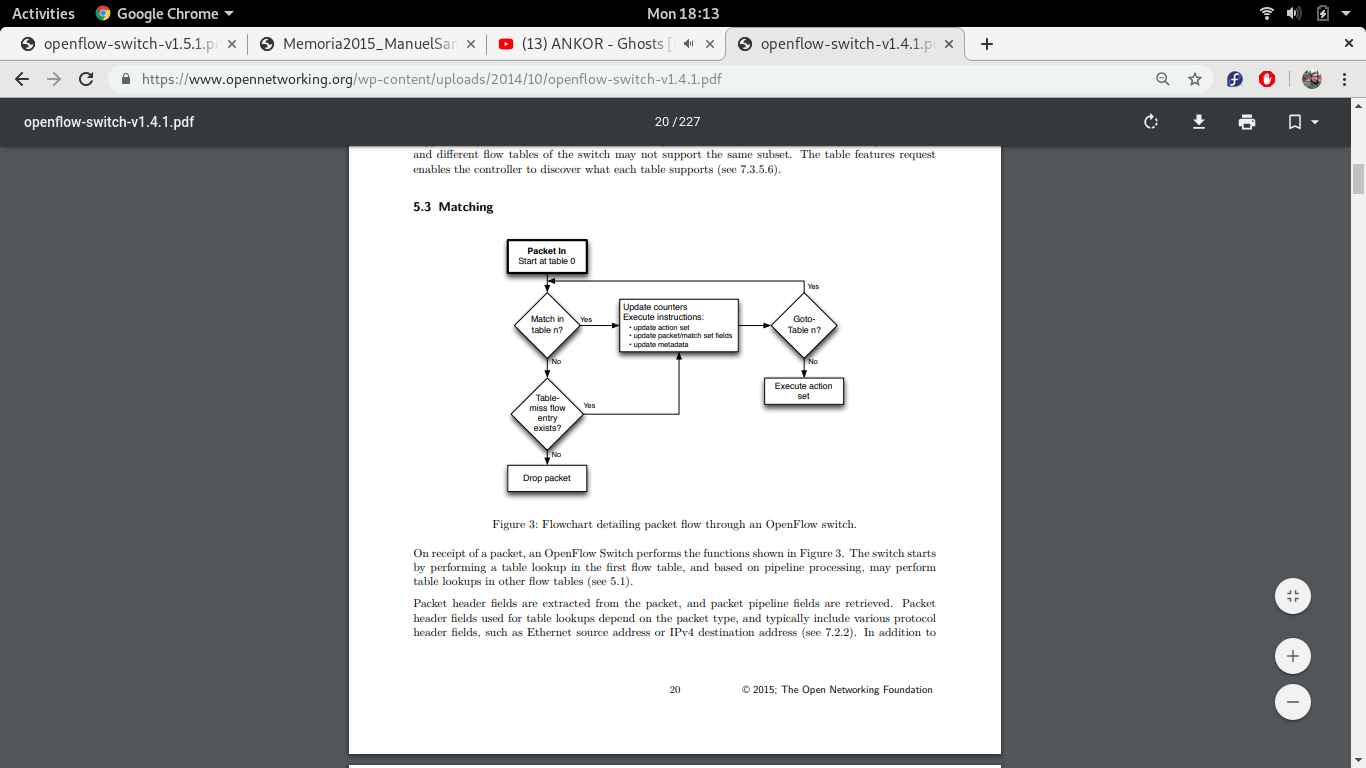
PONER REFERENCIA DE TABLA CON WORD

Donde:

* Match Fields: Campos que permiten hacer la operación de *matching*. Consiste en el puerto de entrada y las cabecera del paquete.
* Priority: precedencia de *matching* de una entrada de flujo
* Counters: Actualizados cuando el paquete tiene una coincidencia
* Instructions: Modifican el conjunto de acciones o el procesamiento *pipeline*.
* Timeouts: Tiempo máximo antes de que un flujo expire en el switch
* Cookie: Dato seleccionado por el controlador. Su objetivo es filtrar las estadísticas de flujos, la modificación de ellos y su eliminación. No se usa cuando se procesan paquetes.
* Flags: Permiten alterar la forma en que se gestionan las entradas de flujo

1.4.3.3 Matching

Cuando el OpenFlow Switch recibe un paquete se activa el diagrama de flujo que se puede ver en la Ilustración X.



Y EL MATCHING CON LOS GRUPOS???

El switch comienza realizando una búsqueda en la primera tabla de flujo del procesamiento de entrada, y basado en el procesamiento *pipeline,* realiza búsquedas en otras tablas si fuera necesario. A continuación, se comprueba si hay tabla de grupo que modifique las acciones

1.4.3.4 Acciones

Cada paquete procesado tiene una serie de acciones a procesar que pueden ser modificadas por las entradas de las tablas de flujo, algunas de las más importantes son:

* Output *port\_no*: Envía el paquete por el puerto indicado en el campo *port\_no.*
* Group *group\_id:* Procesa el paquete a través del grupo especificado
* Drop: Descarta el paquete
* Push VLAN header: Añade el campo VLAN a la que pertenece el paquete (ver apartado 4.3.1)
* Pop VLAN header: Elimina el campo VLAN de la cabecera del paquete

Estas acciones serán utilizadas a la hora de crear reglas de flujo en nuestras aplicaciones.

1.4.4 Mensajes de OpenFlow

A la hora de comunicar el switch OpenFlow con el controlador se utiliza el canal de control. A través de este canal el controlador configura y gestiona el switch, recibe los eventos del switch y le envía los paquetes. Todo esto se hace a través de 3 tipos de mensajes: mensajes controlador a switch, asíncronos y síncronos.

1.4.4.1 Mensajes controlador a switch

Estos mensajes se generan en el controlador y el switch puede, o no, responder a ellos. Dentro de este tipo se encuentran, entre otros, los siguientes mensajes:

* *Features:* Se envía cuando el controlador solicita la identidad y características básicas del switch. Enviado cuando se establece el canal
* *Configuration:* Mensajes de consulta de los parámetros de configuración
* *Modify-state:* Mensajes que gestionan el estado del switch. Por ejemplo para añadir o eliminar flujos.
* *Read-State:* Mensajes que consigue información acerca del switch, como estadísticas o configuración actual.
* *Packet-out:* Mensajes mediante los cuales el controlador puede enviar paquetes por un puerto concreto del switch o redireccionar paquetes que han llegado modificando las acciones que se le aplican.

1.4.4.2 Mensajes asíncronos

Estos mensajes se envían entre el switch y el controlador al llegar un paquete. Entre ellos se encuentan los siguientes:

* *Packet-in:* Mensaje enviado cuando al recibir un paquete no tiene una entrada de flujos para dicho paquete. El controlador procesa el paquete y responde con un mensaje de tipo *Packet-out*
* *Flow-Removed:* Mensaje que notifica que se ha eliminado una entrada de flujo de una tabla.
* *Port-status:* Informa al controlador del cambio en la configuración en un puerto.
* *Role-status:* Informa al controlador del cambio de rol.
* *Controller-status:* Informa al controlador del estado de cambios en el canal de comunicación OpenFlow
* *Flow-monitor:* Informa al controlador de cambios en las tablas de flujo.

1.4.4.3 Mensajes simétricos

Estos mensajes se envían desde cualquier dispositivo sin solicitud previa. Son los siguientes:

* *Hello:* Mensajes que se intercambian en el momento de establecer la conexión entre los conmutadores y el controlador.
* *Echo:* Mensajes que permiten medir la latencia o el ancho de banda para comprobar que un dispositivo esté activo.

1.5 Open vSwitch

DE LOS PUNTOS 1.2 Y 1.3 HAY QUE PONER BASTANTE MAS, COMO CASOS DE USO O COSAS DEL ESTILO QUE TENGO QUE BUSCAR

AÑADIR TAMBIEN UN APARTADO ACERCA DEL OPEN V SWITCH

Capítulo 2: Preparación del entorno de trabajo

En este capítulo se van a detallar los primeros pasos previos a poder empezar a simular redes SDN. Para ello se detalla en cada epígrafe, de manera separada, todas las instalaciones y configuraciones que fueron necesarias para dejar un entorno de trabajo operativo en el que desarrollar la parte esencial de este TFG.

2.1 Instalación del SO

Para poder realizar el TFG correctamente es necesaria la instalación de un sistema operativo GNU/Linux funcional. En este caso se ha elegido la distribución GNU/Linux Fedora 29 con entorno de escritorio GNOME 3, aunque cualquier otra distribución es válida.

El motivo de haber elegido este sistema operativo se debió a diferentes problemas con distribuciones más conocidas como Ubuntu o Debian. En concreto, cabe resaltar que el sistema operativo Ubuntu sufría de cuelgues aleatorios esporádicos, cada vez más habituales, que impedían una continuidad a la hora de trabajar con él.

Por su parte el sistema operativo Debian presentó ciertos problemas con algunos repositorios *non-free* requeridos por el *hardware* del ordenador utilizados, lo que dificultó también su trabajo, y por tanto, se desechó la idea de trabajar con él.

Estos problemas no surgieron con el sistema operativo Fedora y, por tanto, se decidió trabajar con él.

En relación con el usuario principiante que esté interesado en conocer más detalles acerca de la instalación de Fedora 29 es instado a leer el manual de instalación poner referencia con word bien con el enlace de abajo

<https://www.solvetic.com/tutoriales/article/6199-caracteristicas-y-como-instalar-fedora-29/>

2.2 Instalación de GNS3

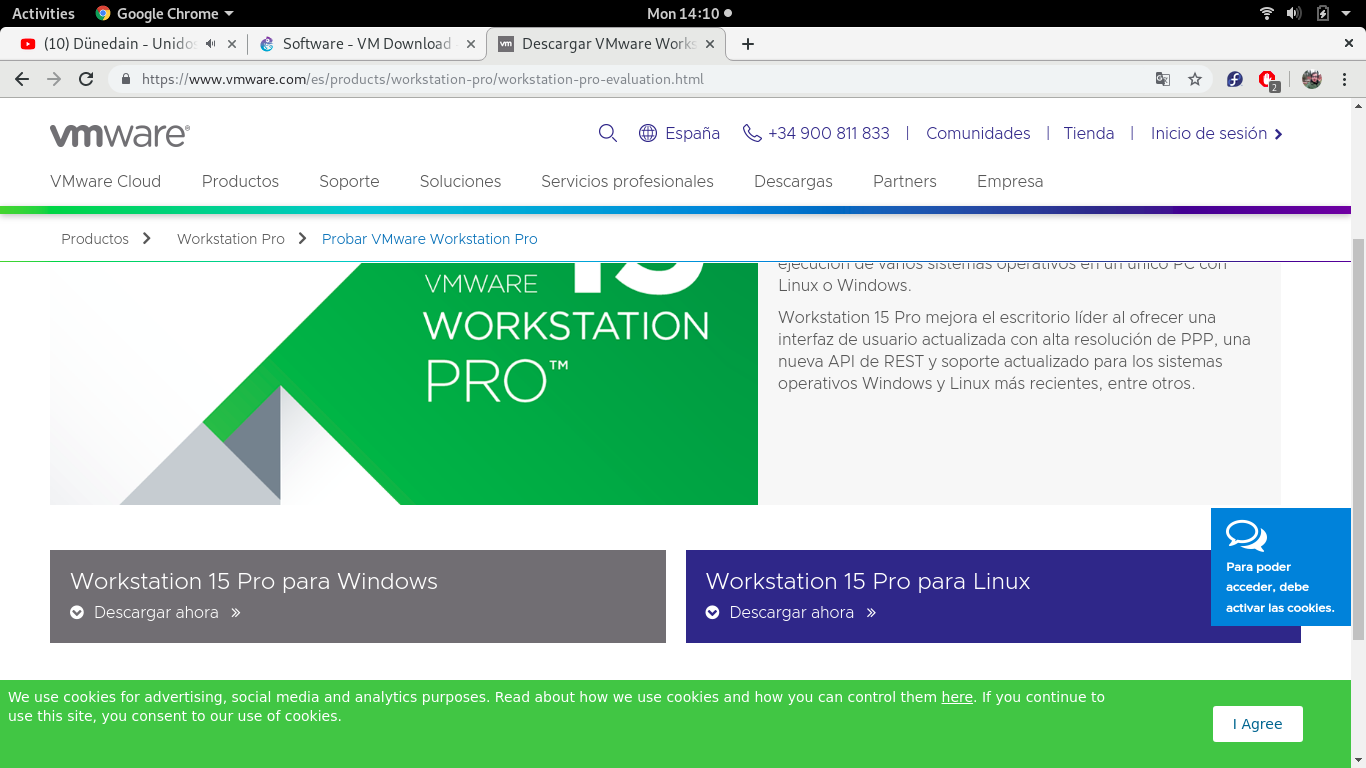
Una vez tenemos una distribución de GNU/Linux instalada, el siguiente paso es instalar el simulador gráfico de redes GNS3. Este simulador es gratis y se puede descargar mediante repositorios siguiendo los pasos que se describen:

* Instalamos dependencias que necesitamos, como python o Wireshark:
  + sudo dnf -y install git gcc cmake flex bison
  + sudo dnf -y install elfutils-libelf-devel libuuid-devel libpcap-devel
  + sudo dnf -y install python3-tornado python3-neifaces python3-devel pyhton-pip

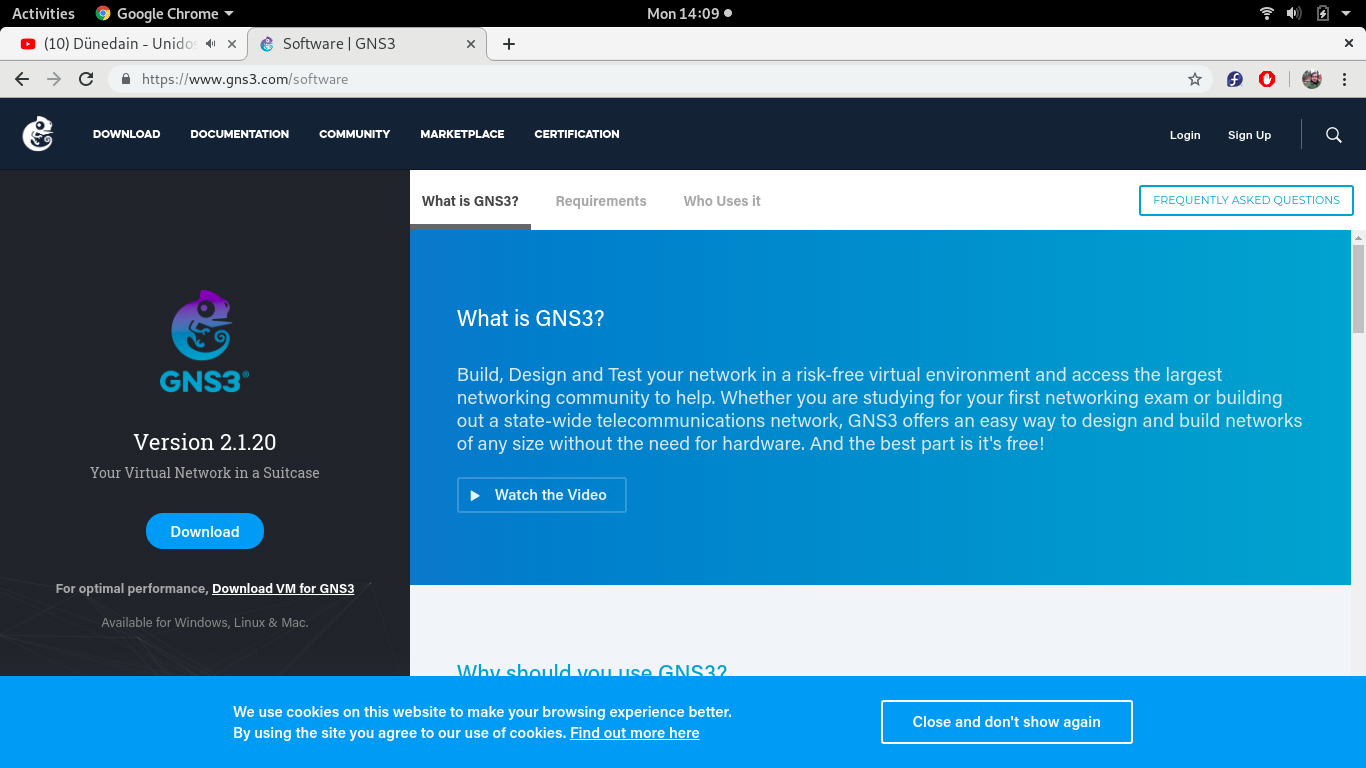
python-pip python3-setuptools python3-PyQt4 python3-zmq

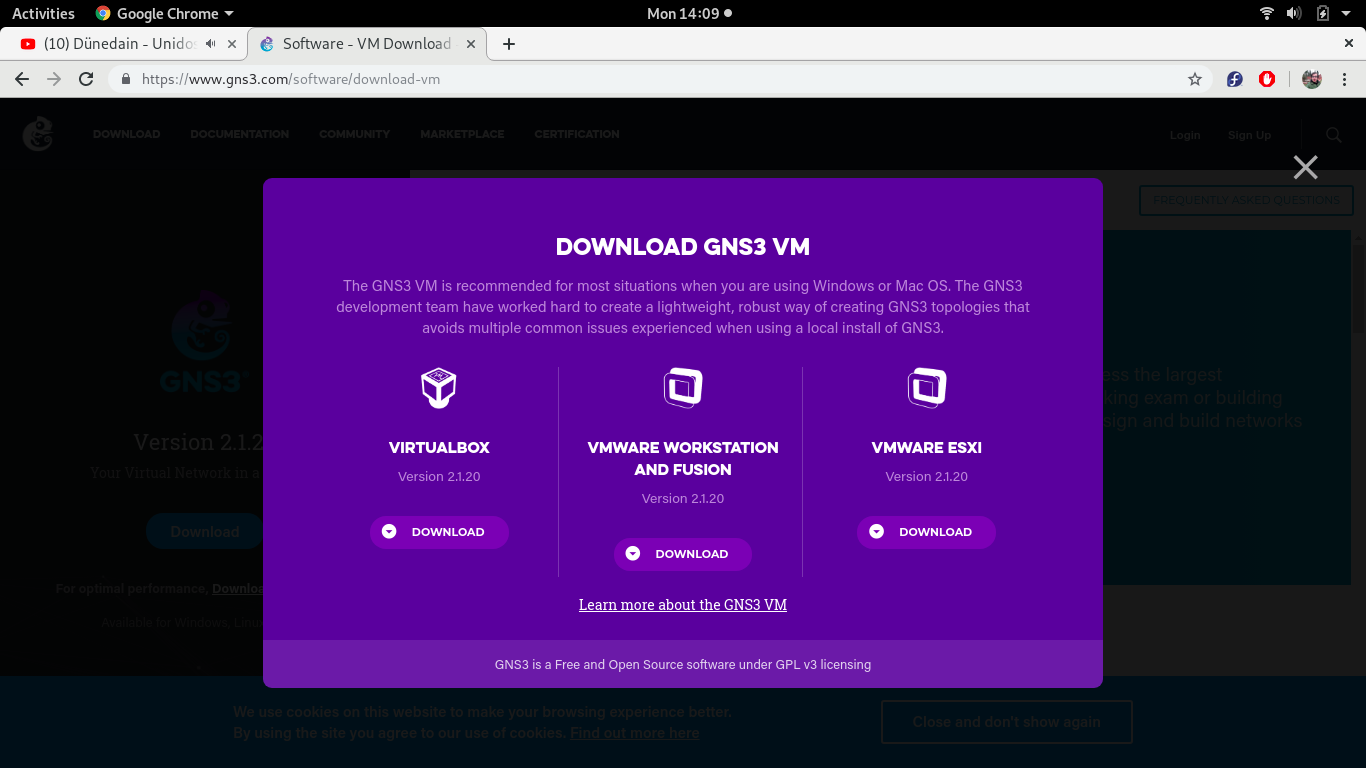
* + sudo dnf -y install wireshark
* Instalamos GNS3:
  + sudo dnf -y install gns3-server gns3-gui
* Añadimos soporte para Docker:
  + sudo dnf -y install dnf-plugins-core
  + sudo dnf config-manager –add-repo https://download.docker.com/linux/fedora/docker-ce.repo
  + sudo dnf config-manager –set-enabled docker-ce-nightly
  + sudo dnf config-manager –set-enabled docker-ce-test
  + sudo dnf install docker-ce docker-ce-cli containerd.io
  + sudo systemctl enable –now docker
  + sudo usermod -aG docker $(whoami)

Una de las opciones que da el programa es utilizar una máquina virtual para que todo el proceso de simulación de las redes quede virtualizado. Para ello es recomendable instalar una herramienta de virtualización como VMware Workstation Pro™ Vmware. En caso de utilizar este software es un requisito que sea la versión Pro. En este TFG se ha empleado la versión 15, como puede verse en la siguiente figura.

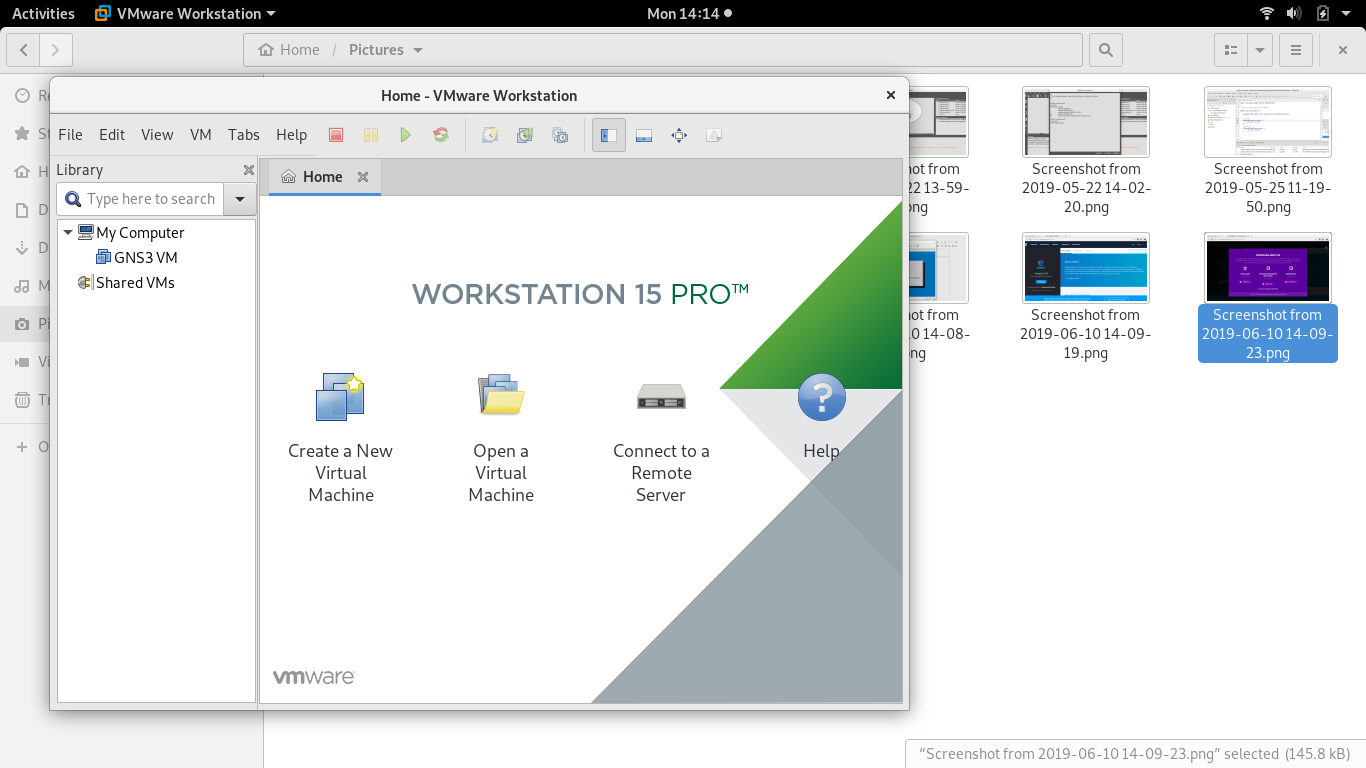


Seguidamente se descarga la máquina virtual para GNS3 disponible en la página web, tal y como se ve en las Ilustraciones siguientes:

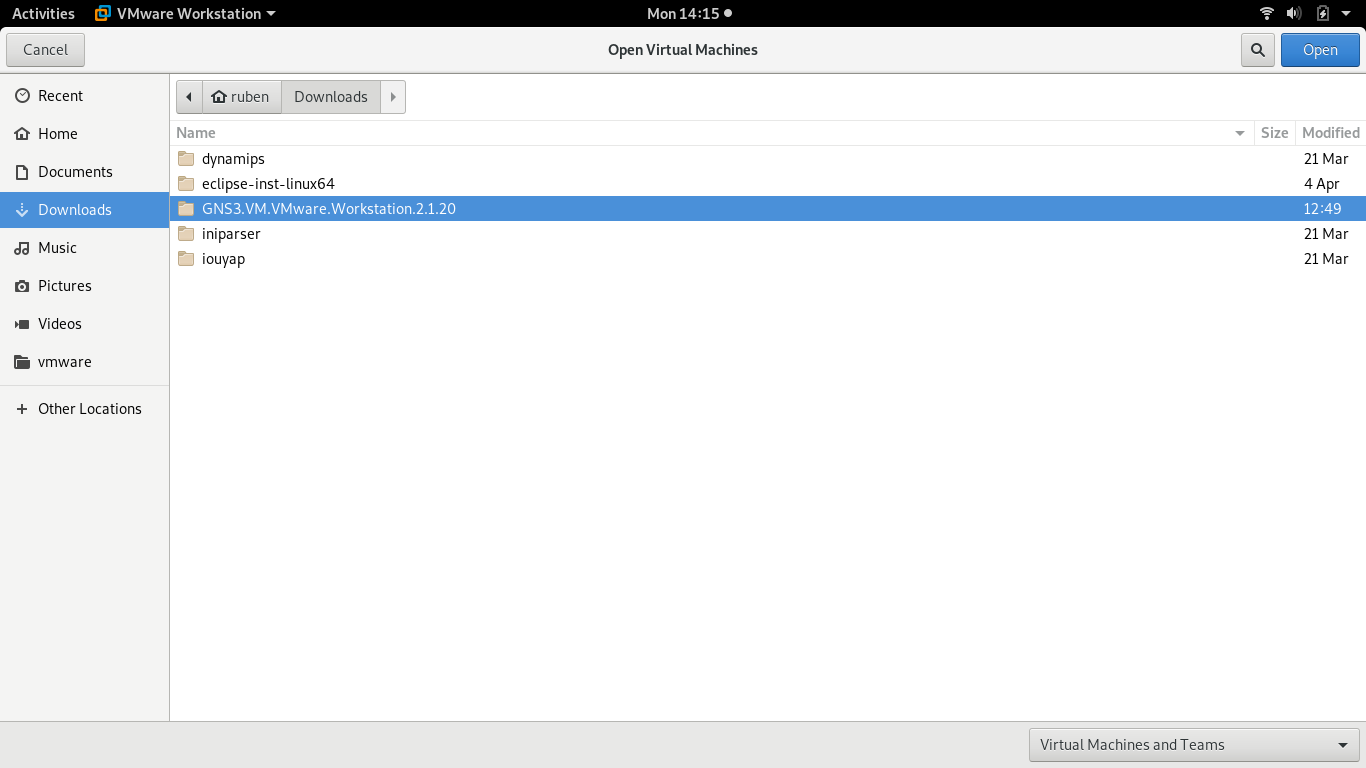


Finalmente importamos la máquina virtual GNS3 al software de virtualización VMware siguiendo los pasos siguientes:

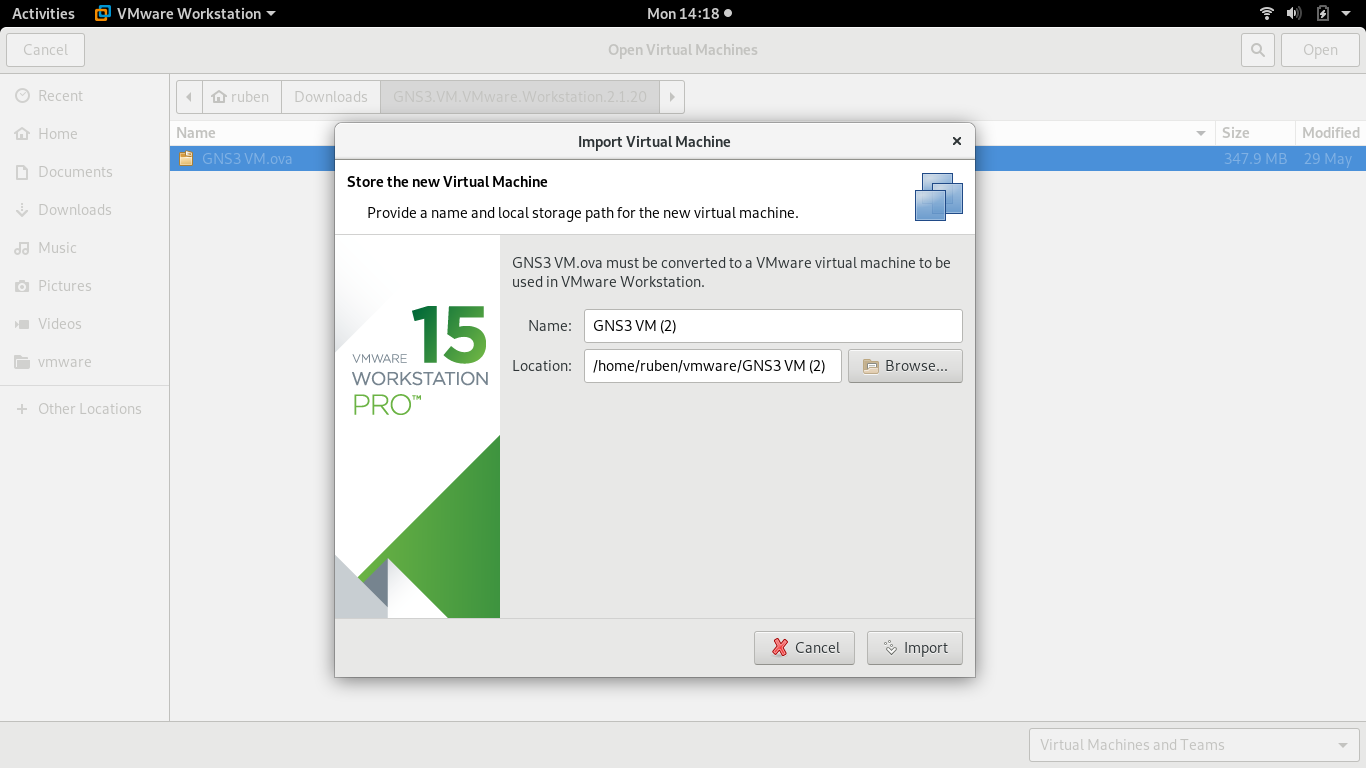
* Abrimos el programa



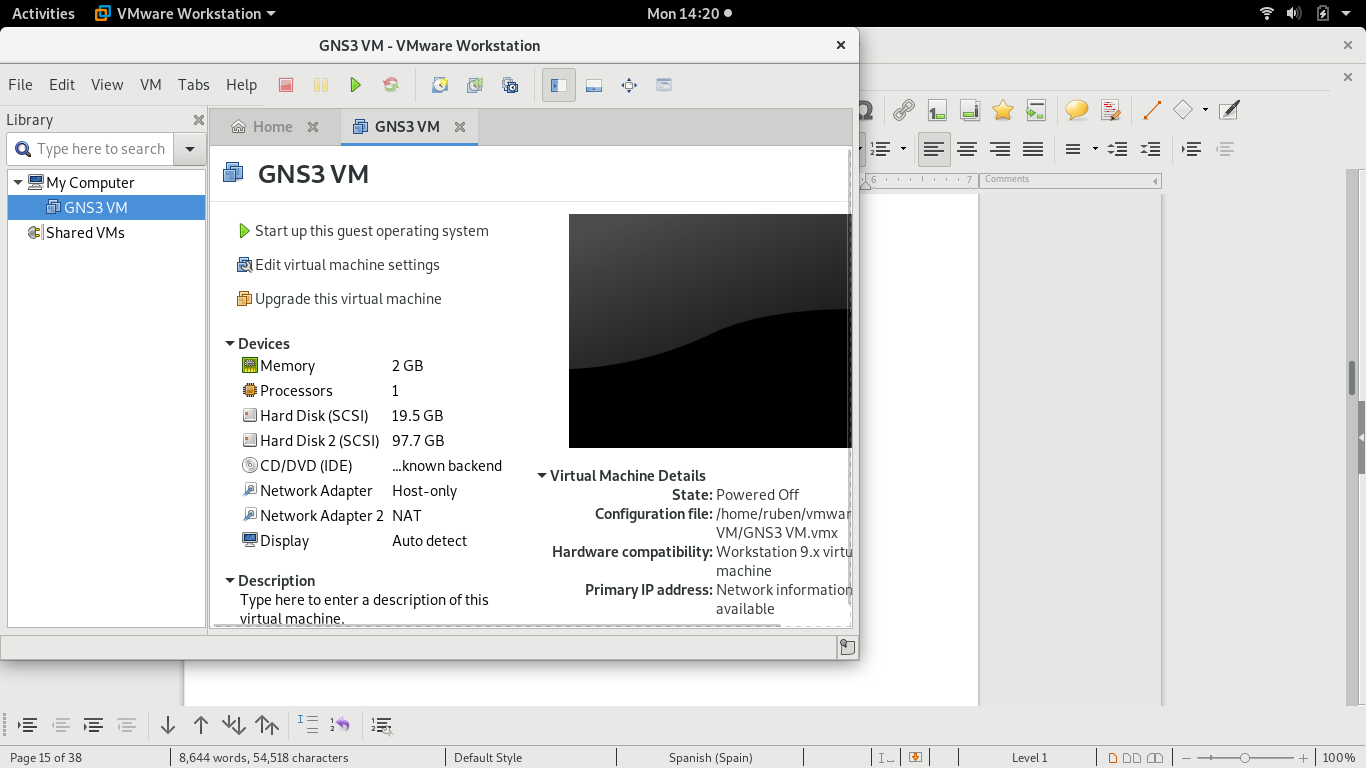
* Clicamos en File  Open
* Seleccionamos la máquina Virtual recién descargada:



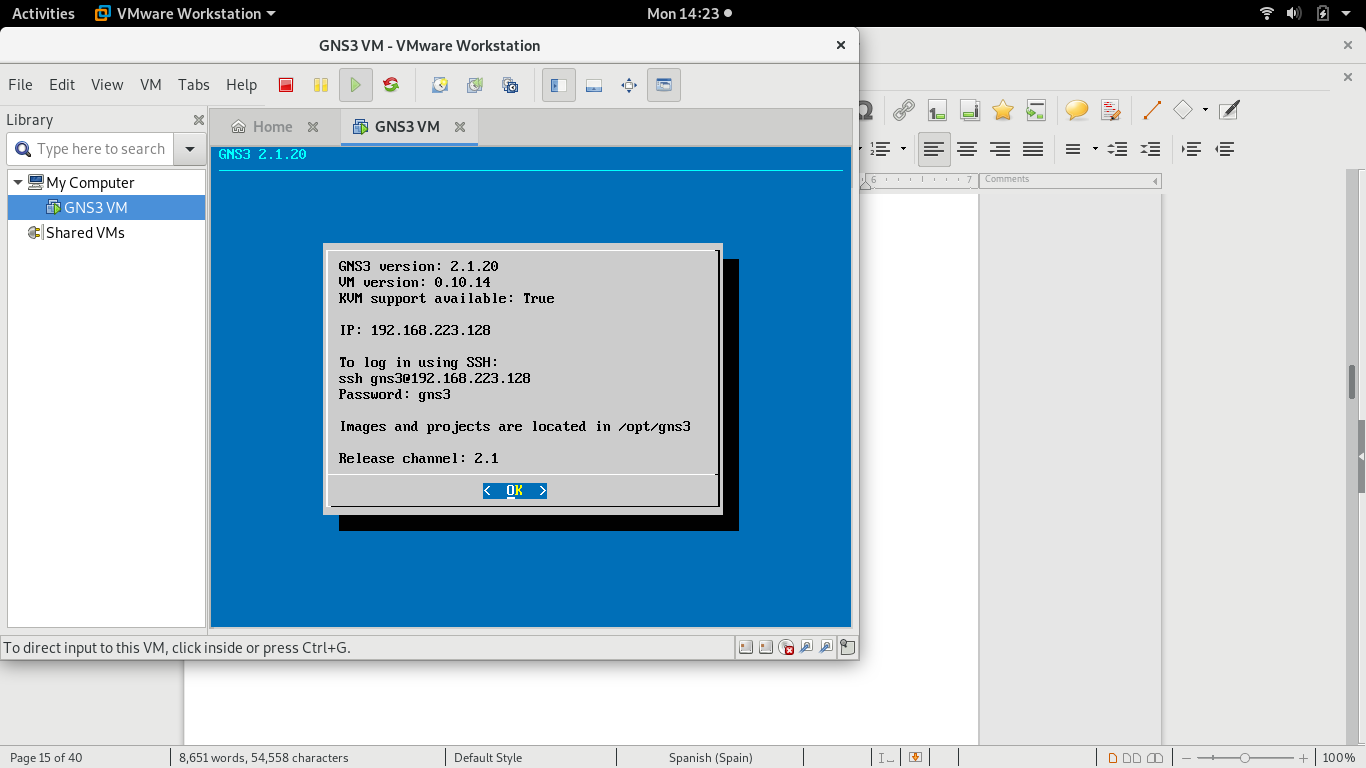
* Clicamos en importar



* Finalmente vamos dando a “Siguiente” en el resto de pasos y ya tendremos la máquina virtual GNS3 instalada. Para arrancarla clicamos sobre la máquina y damos a “*Start up this guest operating system”*



Y nos aparecerá algo como lo siguiente:



2.3 Instalación de ONOS

El siguiente paso en el proceso es la instalación de ONOS (Open Network Operating System) tanto en nuestro PC como en el GNS3. El sentido de tenerlo en nuestro ordenador es poder compilar, depurar y enviar aplicaciones a la máquina virtual.

2.3.1 Instalación en el PC

En primer lugar instalamos dependencias previas que vamos a necesitar, para ello ejecutamos el siguiente comando:

* dnf install git zip curl unzip python bzip2 build

A continuación clonamos desde git (ver Anexo 2 para funcionamiento de GitHub) el código fuente de ONOS

* git clone https://gerrit.onosproject.org/onos
* cd onos
* bazel build onos

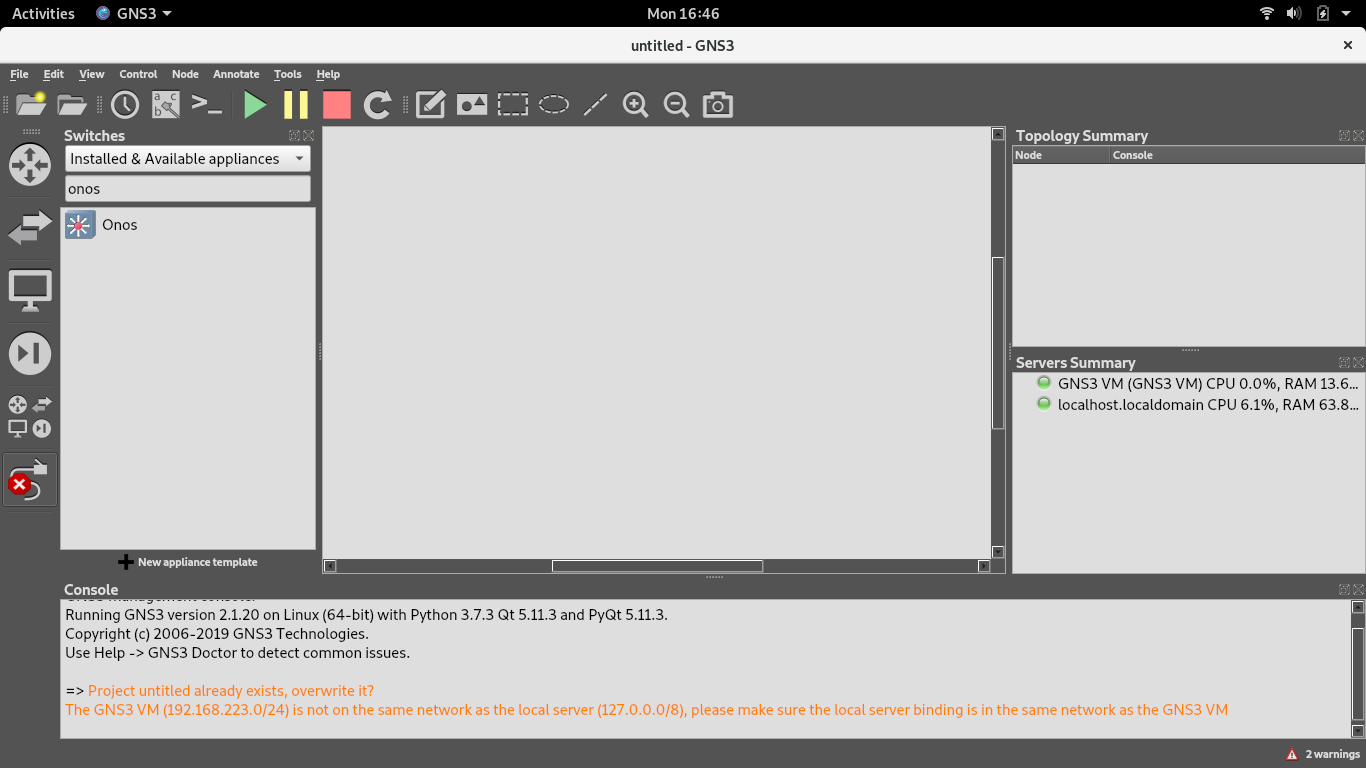
Finalmente instalamos Maven, que es una herramienta que permite simplificar el proceso de compilación y generación de ejecutables a partir del código fuente. Será esta herramienta la que usaremos para compilar y crear los ficheros .oar que enviaremos al controlador de la red. Para descargarlo simplemente bastará con instalar el paquete:

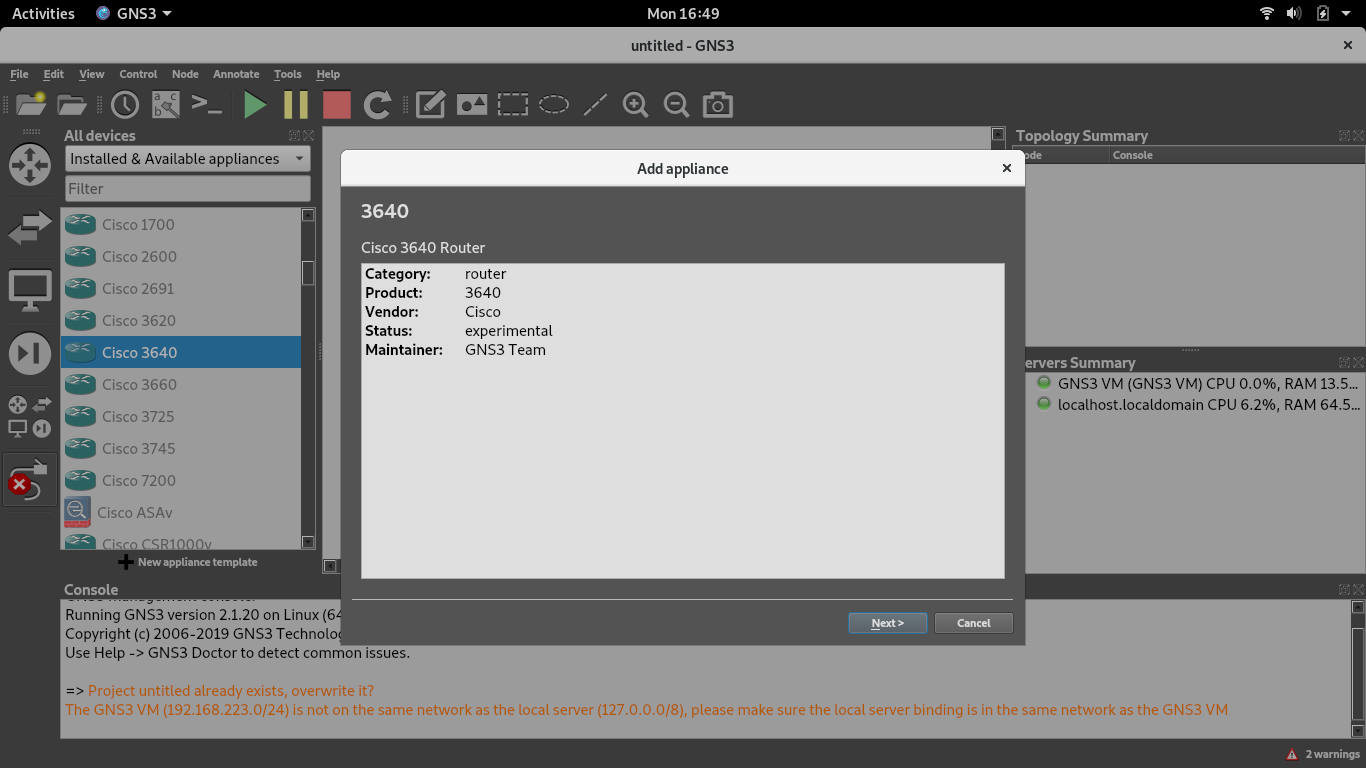
* dnf install maven

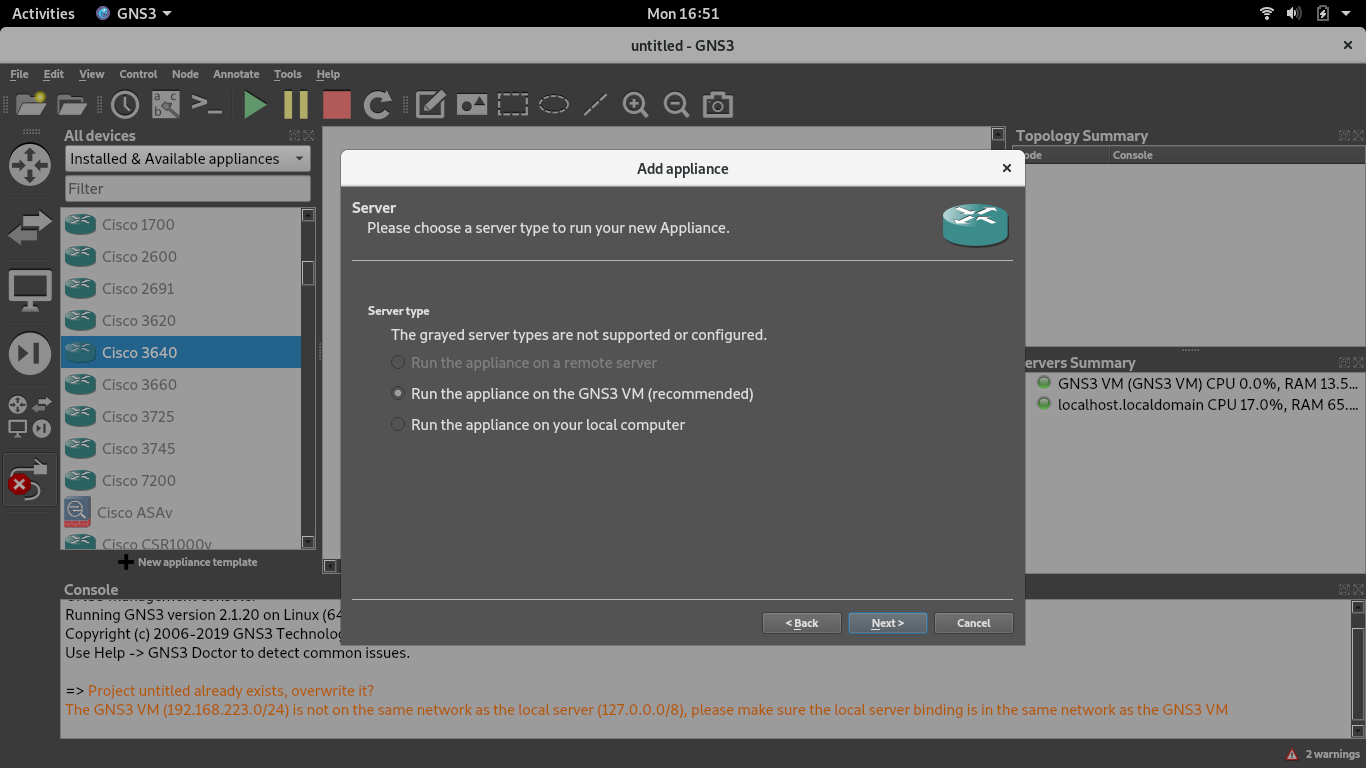
2.3.2 Instalación el GNS3

Para instalar cualquier dispositivo en el GNS3, y en concreto el controlador ONOS, el procedimiento es siempre el mismo:

* Seleccionamos el dispositivo que vamos a instalar, en nuestro caso ONOS (Nota: El resto de explicación se va a realizar con un router Cisco debido a que ONOS ya está instalado, sin embargo el procedimiento es idéntico)

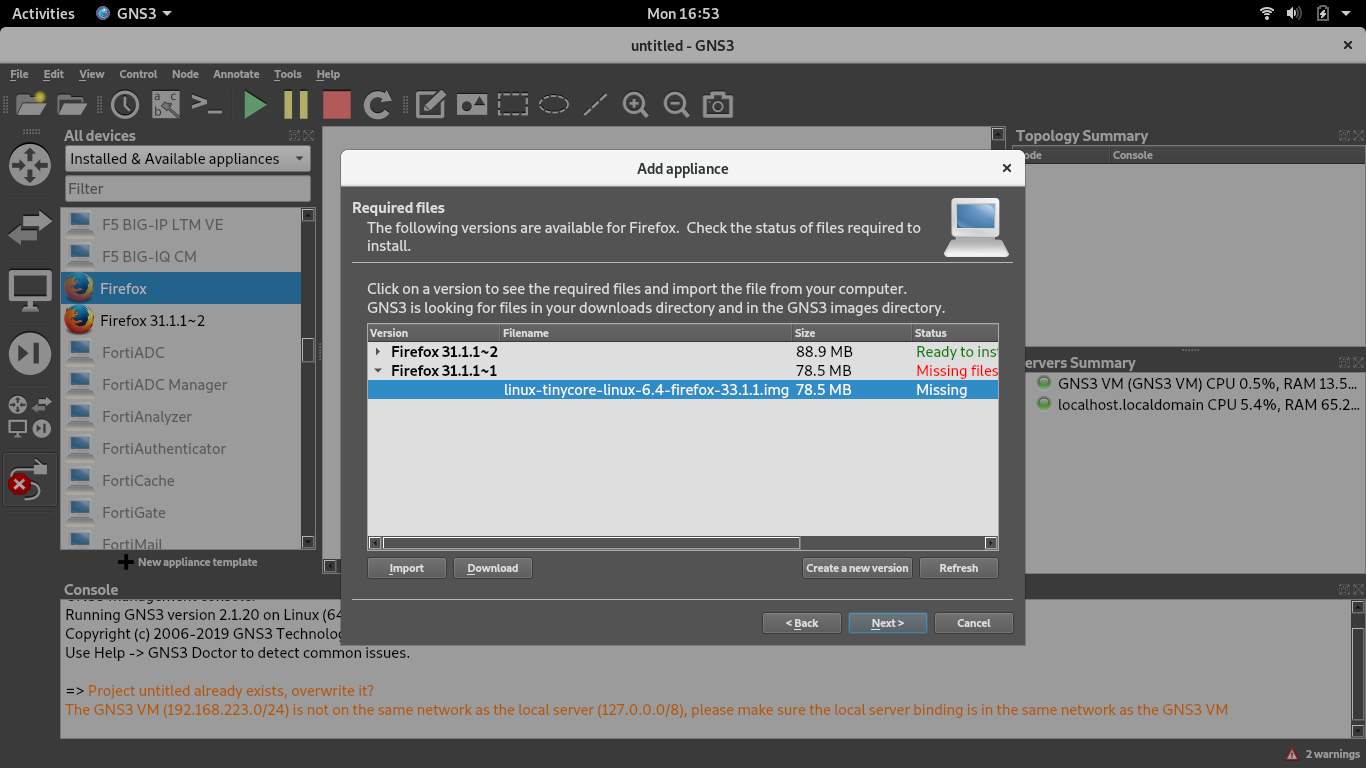


* Nos aparece esta ventana donde aparece la información del dispositivo, clicamos en “Next”
* Aparece una ventana en la que indicamos dónde se va a ejecutar el dispositivo



Seleccionamos que se ejecute en la máquina virtual de GNS3

* A continuación, nos aparece una lista con las versiones del dispositivo están disponibles, tanto las instaladas como las que no lo están.

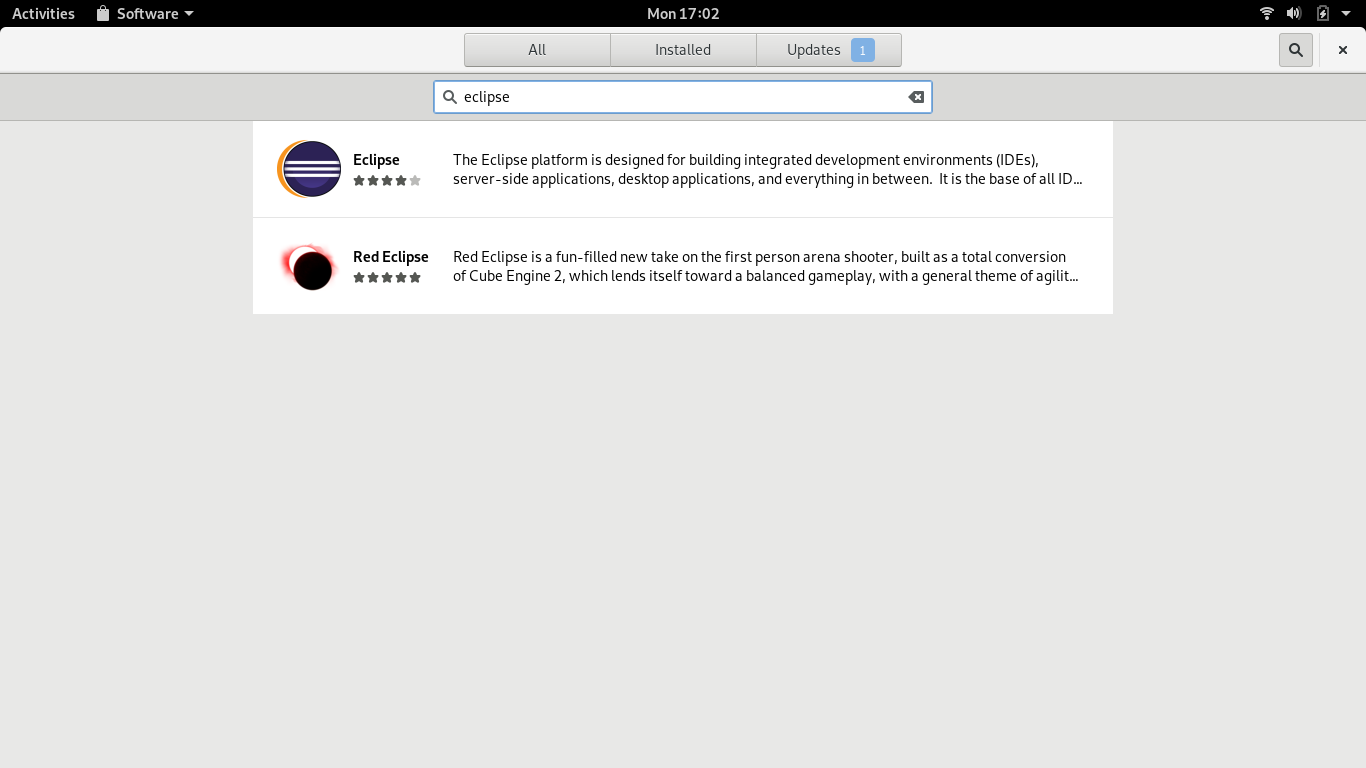


En el ejemplo se observa como la versión 2 de Firefox está instalada, mientras que la 1 no lo está. Si queremos instalarla seleccionamos la imagen que no ha podido encontrar y clicamos en *Download*. Esto hace que se descargue la imagen automáticamente.

Cuando se haya descargado la importamos accediendo al directorio en el que se haya descargado y ya tendremos instalado nuestro dispositivo

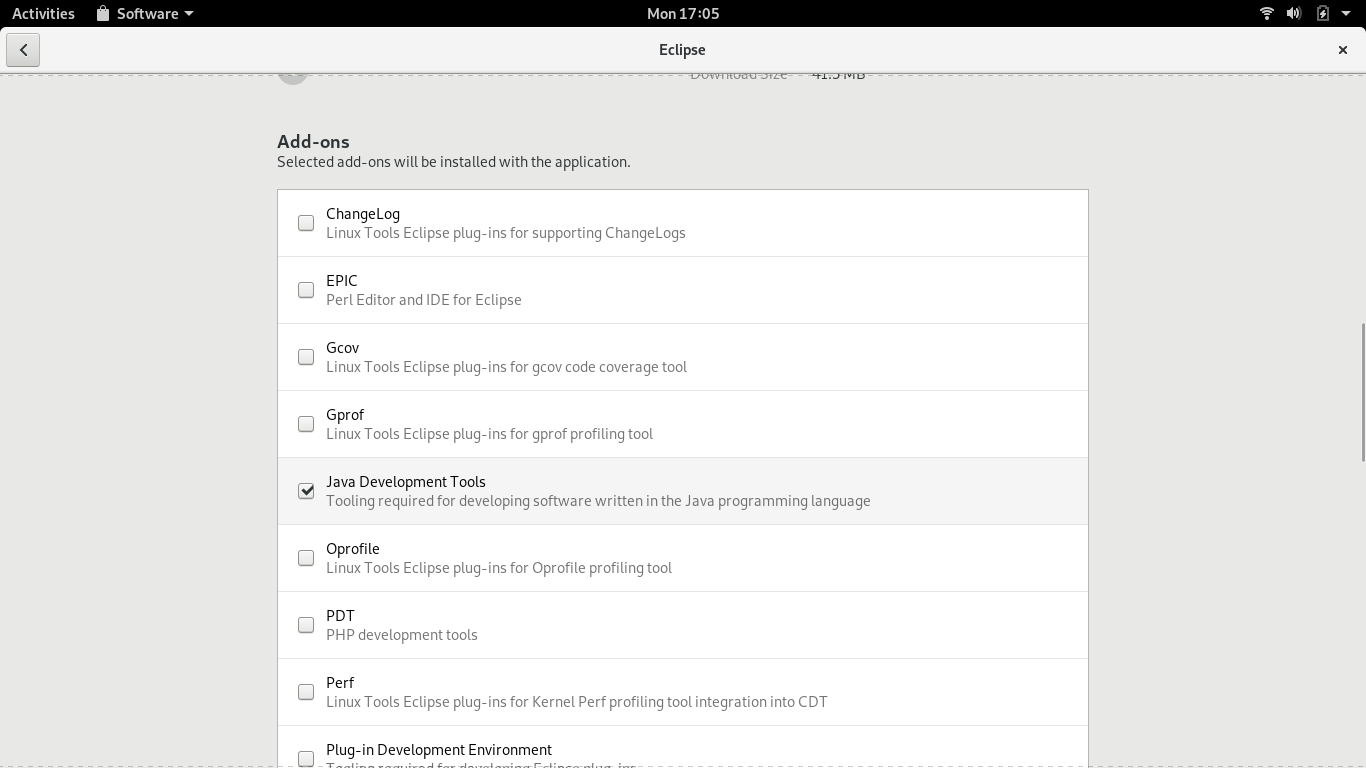
2.4 Instalación de Eclipse

Una vez tenemos GNS3 instalado el siguiente paso es instalar Eclipse para poder programar las diferentes aplicaciones que queramos mandar al controlador y que veremos en capítulos posteriores. En este caso la instalación es muy sencilla ya que consiste únicamente con acceder a la aplicación *Software* en nuestro sistema Fedora 29, buscar Eclipse e instalarlo.



Seleccionamos Eclipse y bajamos el scroll hasta la parte que indica los add-ons, porque si lo instalamos únicamente se instala la base del IDE (*Integrated Development Environmentt*) sin ningún tipo de herramienta adicional.

Dado que el proyecto se va a basar en el código Java instalamos los add-ons referidos a este lenguaje.

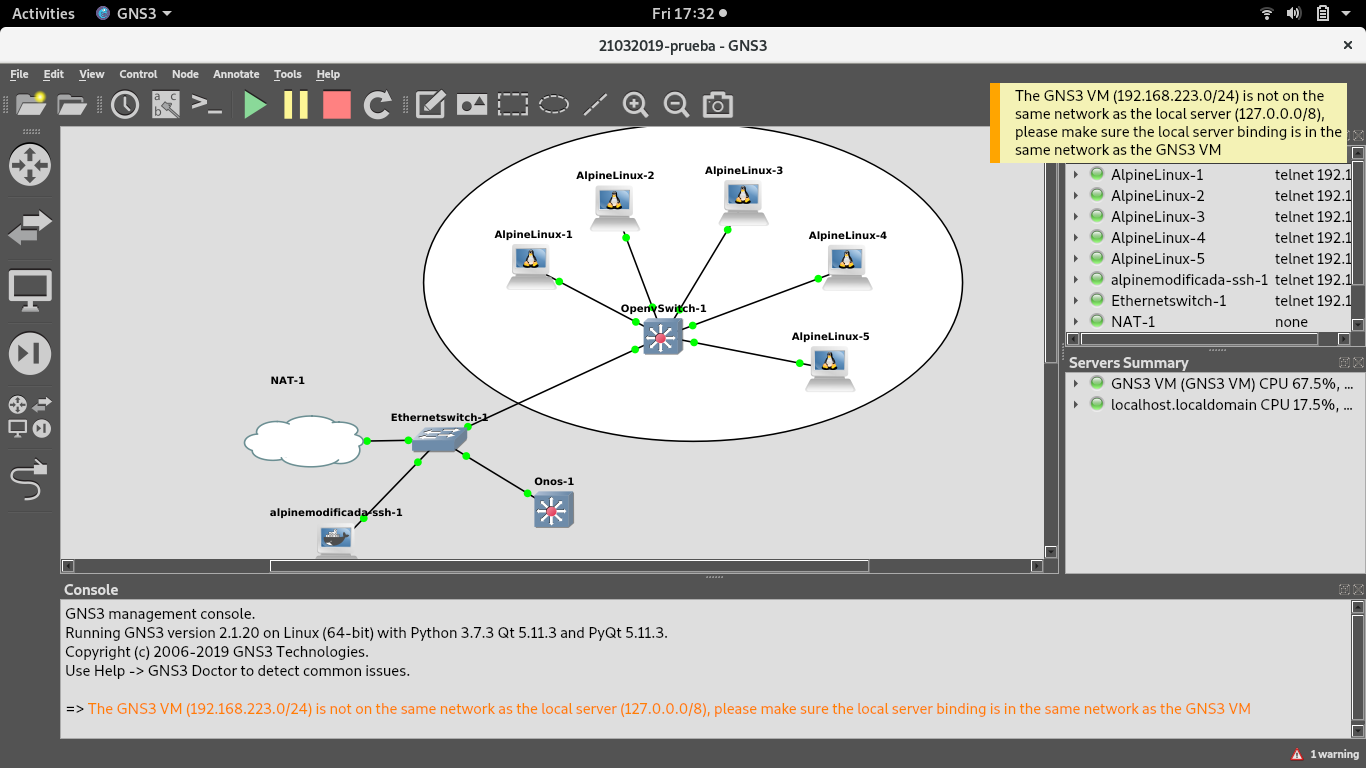


Finalmente damos a instalar y esperamos a que finalice el proceso

Capítulo 3: Creando la primera red con GNS3

En este capítulo se va a detallar la red que se va a simular y sobre la cual instalaremos las diferentes aplicaciones que irán componiendo el proyecto.

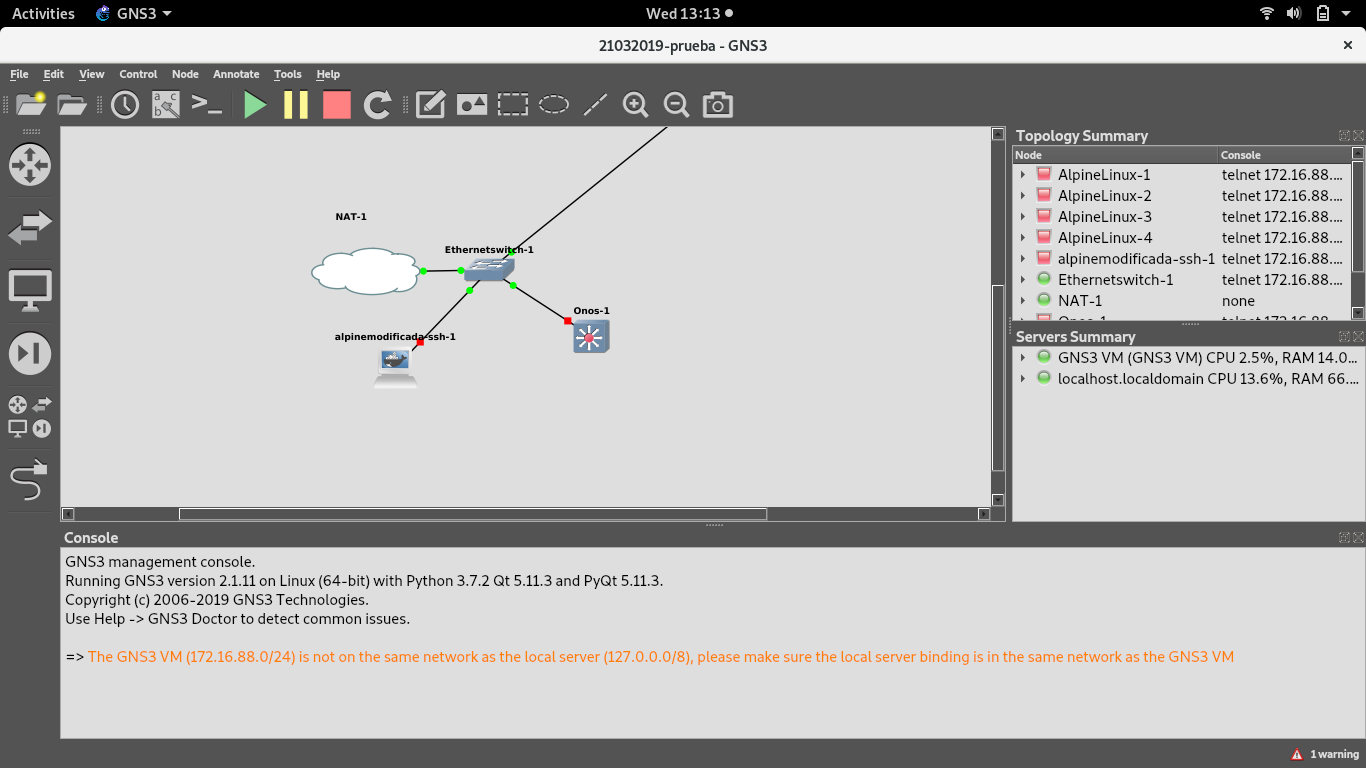
La red visible en la Ilustración X se puede dividir en 2 partes diferenciadas:



Por un lado la conexión con el controlador y con el exterior y por otro los hosts conectados por un Open vSwitch.

3.1 Conexión con el controlador y con el exterior

La parte de la red que concierne a este apartado se puede ver en la Ilustración siguiente:



Para poder realizar esta parte es necesario tener acceso a Internet desde nuestro propio portátil (recordemos que todo este proceso se está realizando de forma virtual), para ello es necesario instalar una regla NAT (Network Address Translation) que nos permita acceder desde nuestra red privada interna a GNS3 (huésped) a través de la red a la que se conecta el ordenador (anfitrión).

Otro paso previo es asignar una dirección IP a nuestro controlador. Ello se puede hacer en GNS3 clicando botón derecho sobre nuestro controlador, editando la configuración del mismo y asignándole una dirección IP de manera estática (también se le puede asignar por DHCP pero ello implica que cada vez que ejecutemos el proyecto la dirección IP va a cambiar). En nuestro caso el fichero de configuración queda de la siguiente forma:

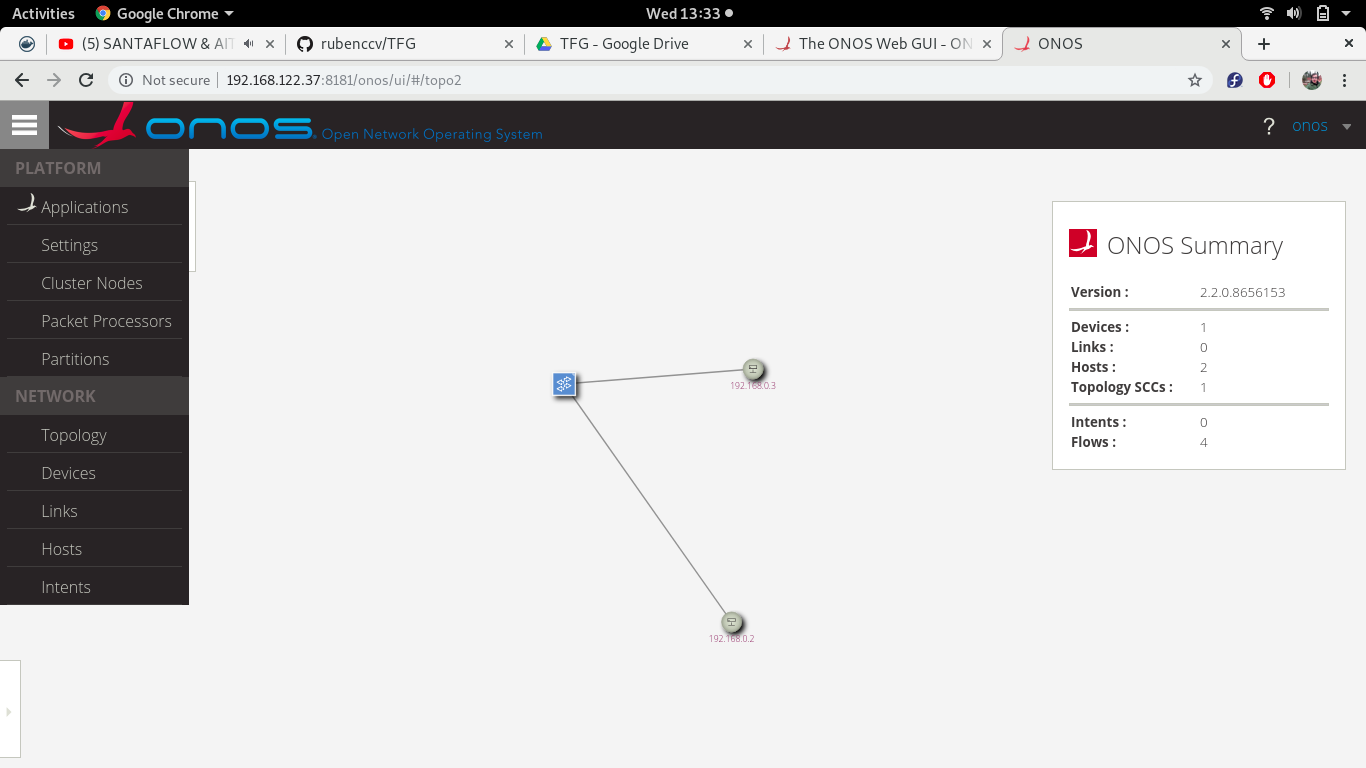


Hemos elegido arbitrariamente la dirección IP 192.168.122.37, dentro de la subred 192.168.122.0/24

Una vez realizados estos pasos previos, ya podemos acceder al controlador para acceder a sus comandos y funciones. Hay 2 formas de acceder a él:

* Una de ellas es accediendo desde un host conectado a él a la interfaz gráfica mediante la página web http://IP-ONOS:8181/onos/ui. Donde IP-ONOS es la dirección IP que hemos definido previamente en el fichero de configuración. Las credenciales de acceso son user: *onos* y password: *rocks*.

Lo que vemos una vez que accedemos es lo siguiente:

En primer lugar vemos la topología que el controlador ha detectado. En nuestro ejemplo podemos ver una red muy simple que consta de un switch conectado a 2 hosts.

En el recuadro de la derecha podemos ver un resumen acerca de la red, así como los flujos que han generado diferentes aplicaciones.

Finalmente en el menú de la izquierda podemos ver las aplicaciones disponibles, ver en detalle los dispositivos o los hosts...

* La segunda opción para acceder al controlador, y la que más usaremos, es mediante consola. Para ello debemos instalar un host que tenga un cliente SSH (el Alpine por defecto no lo trae instalado), así que lo que haremos será modificar la imagen almacenada en Docker.

Para empezar hacemos un *pull* de la imagen alpine original:

docker pull alpine

A continuación, ejecutamos una instancia de dicha imagen en nuestro shell:

docker run --name alpinemodificada2 -it alpine /bin/sh

Una vez dentro de la instancia que acabamos de ejecutar, vamos a instalar las aplicaciones necesarias para nuestro proyecto:

apk add openssh wget curl

Y guardamos la imagen modificada con los paquetes instalados con el nombre deseado:

docker commit alpinemodificada2 alpinemodificada

Por último podemos borrar la instancia ejecutada para liberar recursos:

docker rm alpinemodificada2

Ahora que ya tenemos un host con un cliente SSH instalado, procedemos a acceder desde él al controlador mediante consola con el comando:

ssh -p 8101 [onos@IP-ONOS](mailto:onos@IP-ONOS)

Nuevamente la contraseña de acceso es *rocks*

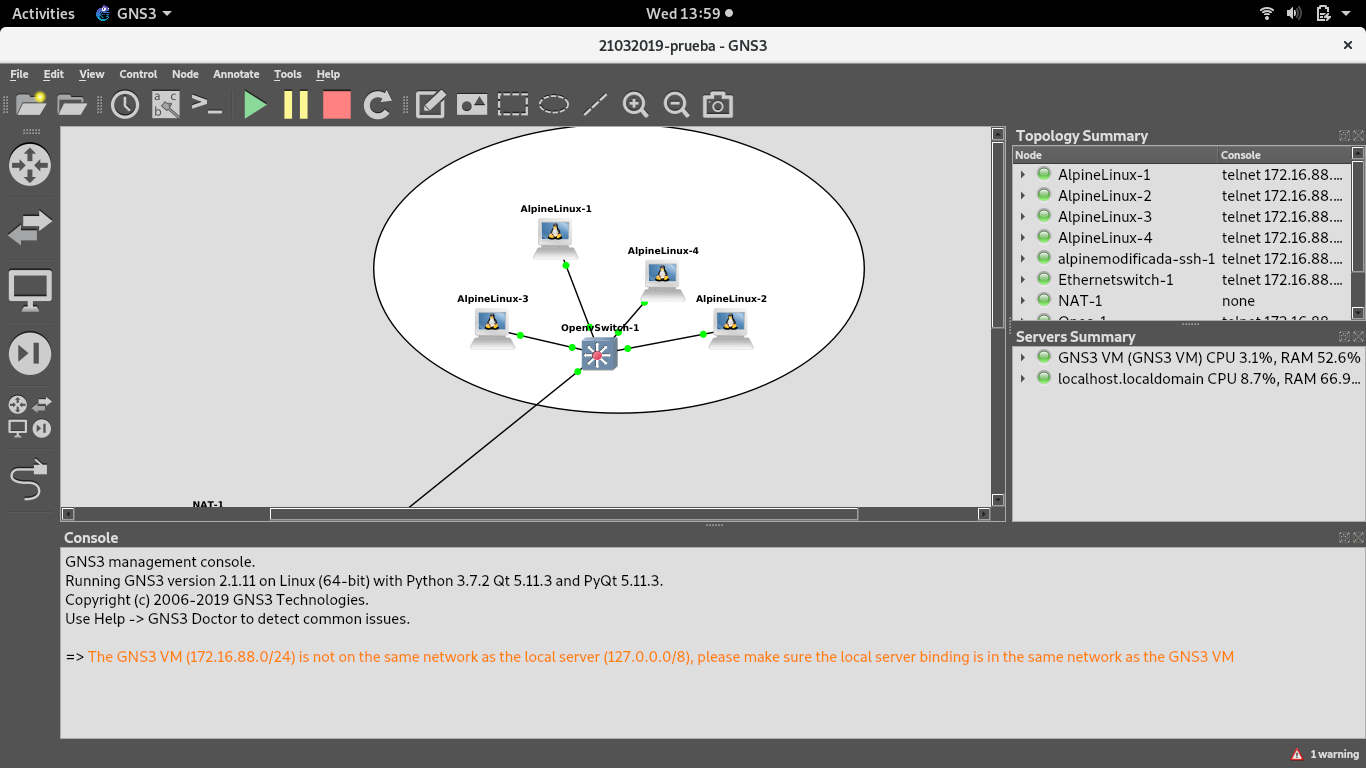
Algunos comandos de utilidad que son necesarios para comprobar el funcionamiento son:

* devices: Devuelve los dispositivos que son controlados por el controlador. En las redes que simularemos son los switches Open vSwitch.
* hosts: Devuelve los hosts que ha reconocido (aquellos que han enviado algún paquete y están conectados a los dispositivos).
* flows: Devuelve los flujos OpenFlow que se han instalado sobre los dispositivos las diferentes aplicaciones y que gestionan el comportamiento de la red.
* log:set debug: Accedemos al modo debug del registro (log) que nos devuelve más información acerca de lo que está ocurriendo en el controlador.
* app activate *nombreAplicacion*: Activa la aplicación en el controlador para que se inicie su funcionamiento.
* app deactivate *nombreAplicacion*: Desactiva la aplicación en el controlador para que se finalice su funcionamiento.

3.2 Hosts conectados a Open vSwitch

Esta parte de la red es la que nos permitirá probar el funcionamiento de nuestras aplicaciones y se irá modificando en función de los requerimientos de dichas aplicaciones.

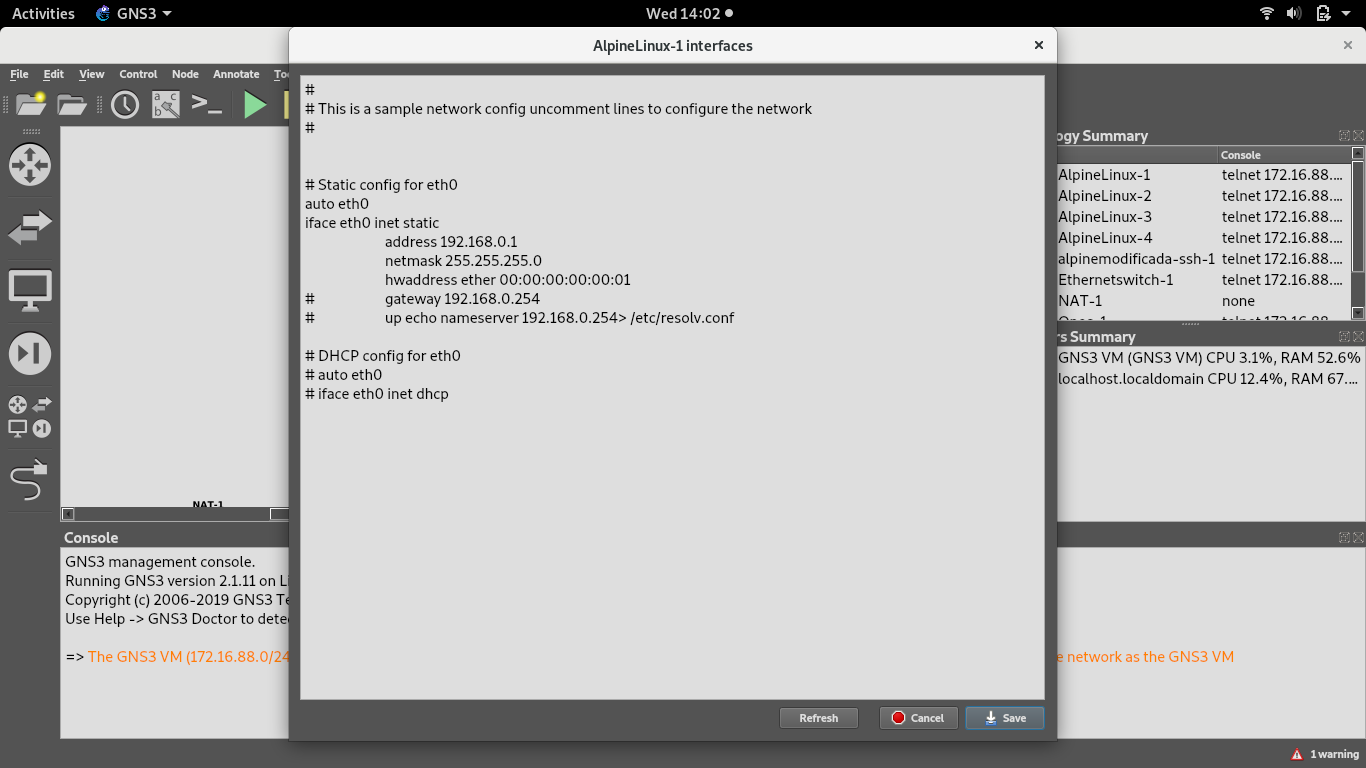
Para las primeras tendrá la siguiente topología:



Como podemos ver, tenemos 4 hosts conectados a un Open vSwitch que recibirá información del controlador para conocer como debe reaccionar frente al tráfico que le llegue. Es en este dispositivo en el que se instalan los flujos que cree el controlador.

Para poder trabajar más cómodamente sobre cada uno de los hosts se ha modificado la configuración creando una subred propia.

El fichero de configuración queda de la siguiente forma:



Tal y como se puede ver se le ha asignado una IP y una MAC estática (en orden ascendente por cada host que utilicemos).

Finalmente, para configurar el Open vSwitch es necesario indicarle quién es el controlador. Como la IP de éste es estática, este proceso sólo tendremos que realizarle una vez, en caso de que se eligiese por DHCP habría que realizarlo cada vez que arrancamos el programa. Para poder indicárselo abriremos una consola desde el Open vSwitch y ejecutaremos los siguientes comandos:

Creamos el bridge br0

ovs-vsctl add-br br0 -- set bridge br0 datapath\_type=netdev

Añadimos los puertos que tiene el router excepto eth0 que lo mantendremos fuera del *bridge*

ovs-vsctl add-port br0 eth1

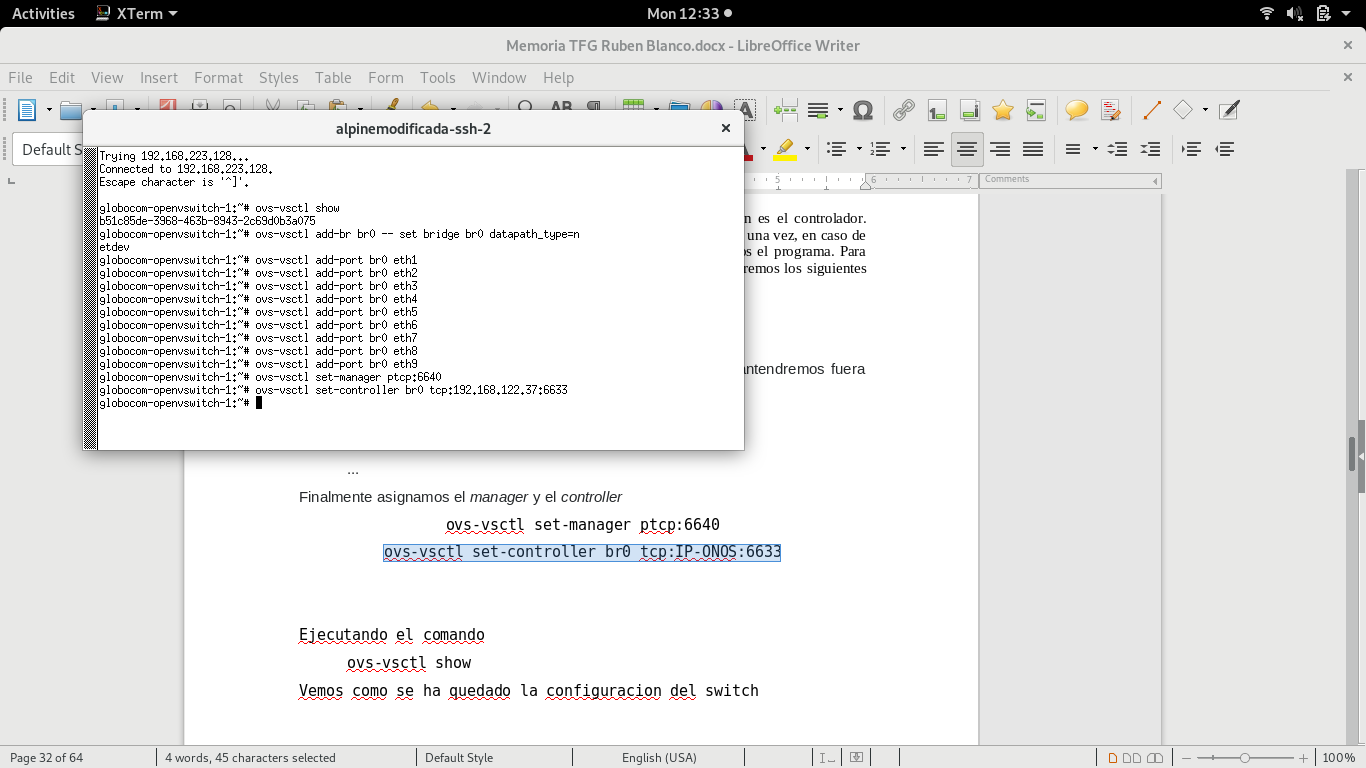
ovs-vsctl add-port br0 eth2

...

Finalmente asignamos el *manager* y el *controller*

ovs-vsctl set-manager ptcp:6640

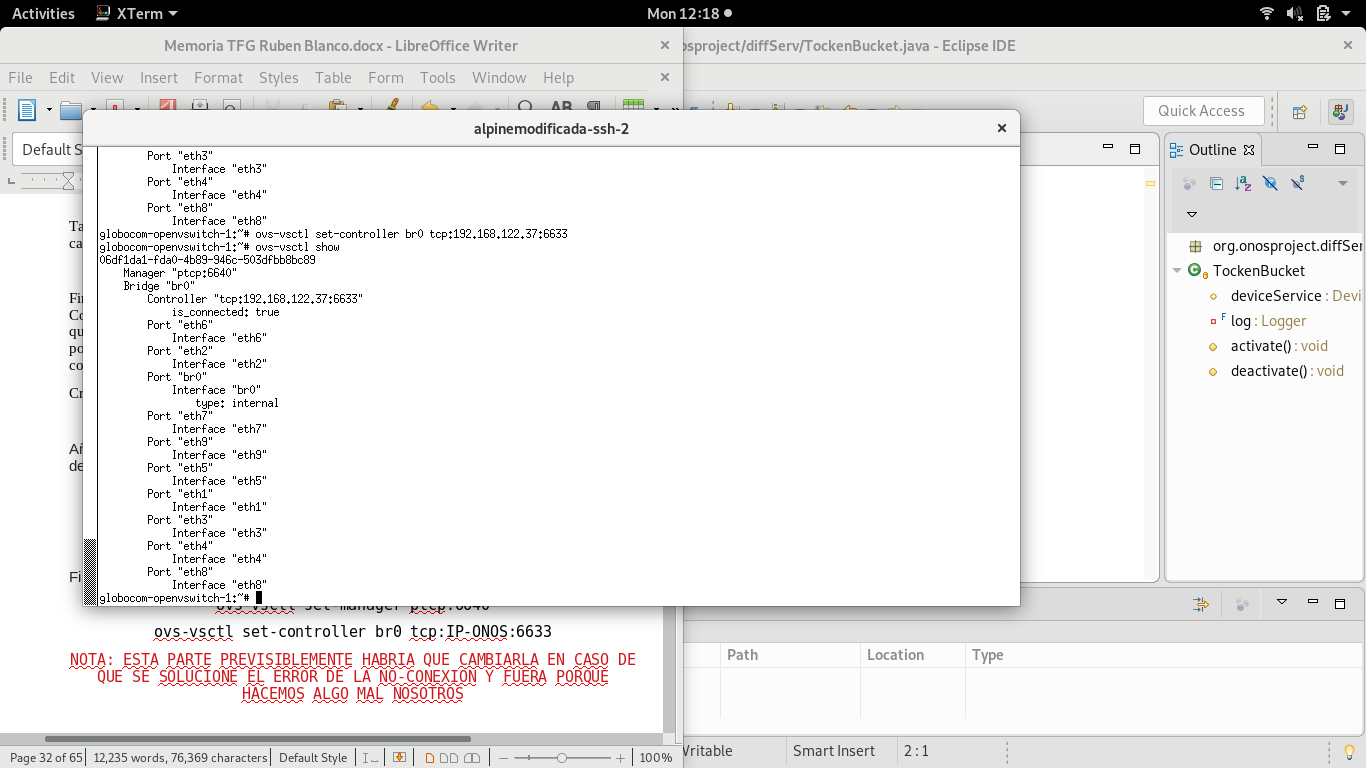
ovs-vsctl set-controller br0 tcp:IP-ONOS:6633



Ejecutando el comando

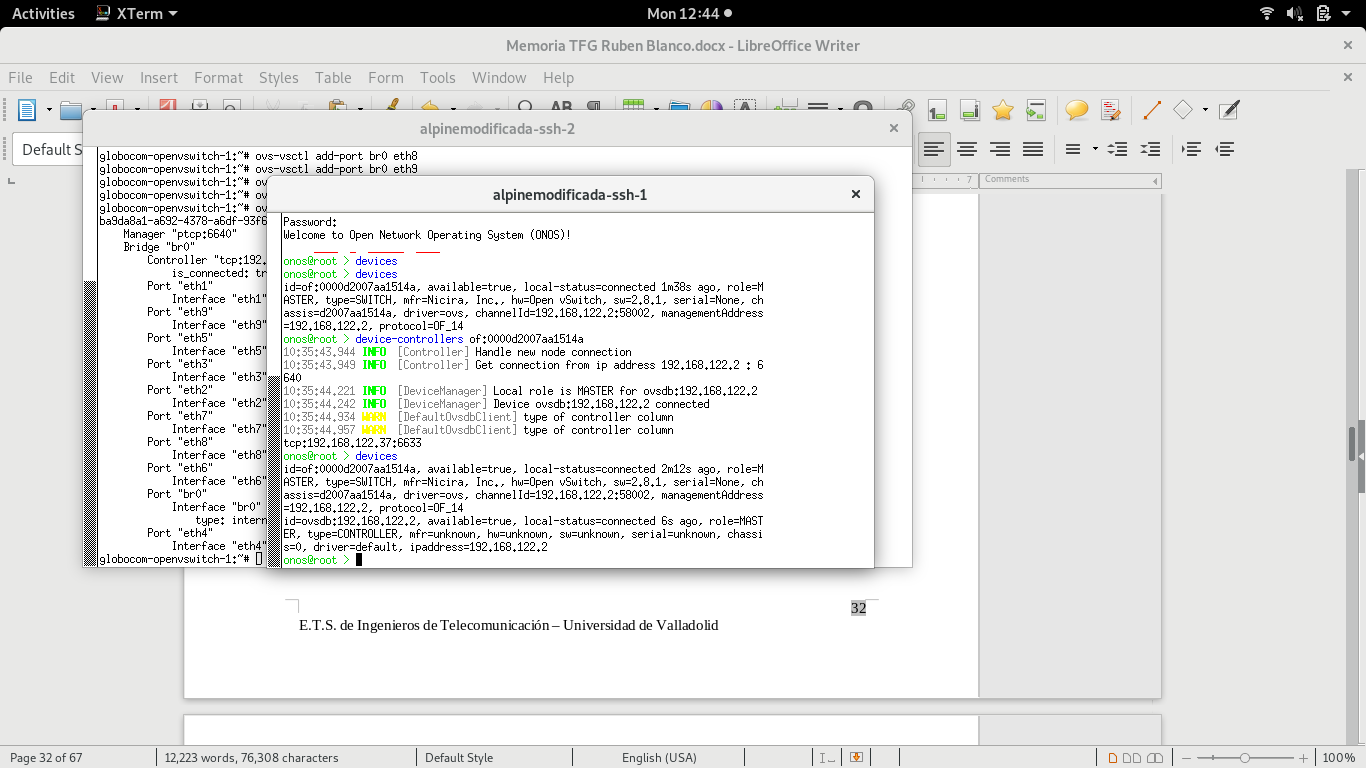
ovs-vsctl show

Vemos como se ha quedado la configuracion del switch



En el controlador ONOS realizamos lo siguiente:

DETALLAR MAS LOS PASOS Y VER PARA QUE SIRVE EL COMANDO DEVICE-CONTROLLERS



Una vez realizado todo esto podremos proceder a realizar nuestras propias aplicaciones y ejecutarlas en esta red.

Capítulo 4: Aplicaciones desarrolladas

En este capítulo se van a detallar las aplicaciones que se han realizado asi como las utilidades que tienen.

Antes de ello, tenemos que crear el proyecto para luego poder modificarlo. Esto se realiza desde nuestra terminal ejecutando el comando.

onos-create-app app org.onosproject nombreApp version org.onosproject.nombreApp

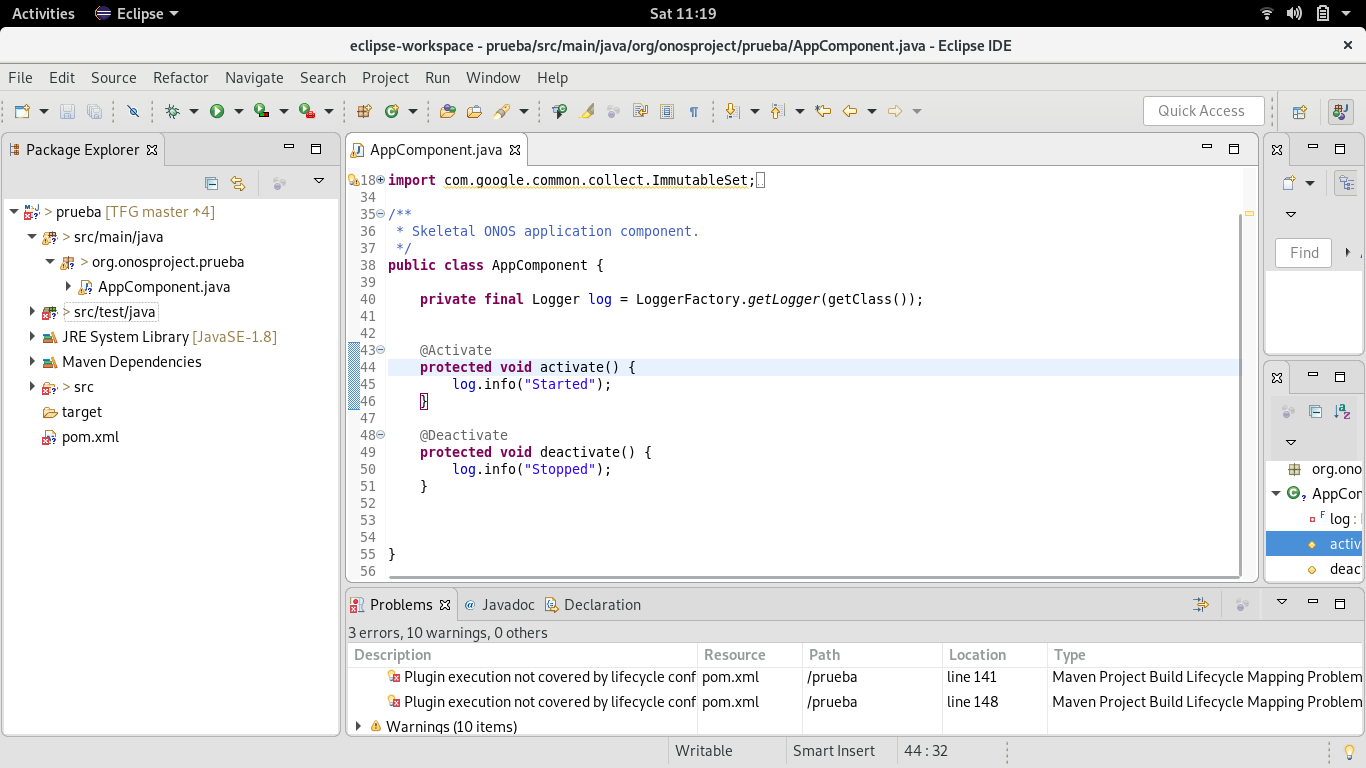
donde nombreApp será el nombre que decidamos dar a nuestra aplicación y la versión es la fase en la que estemos de la aplicación. Si es una aplicación que estamos empezando se pone 1.0-SNAPSHOT.

Una vez ejecutado el comando se crea una carpeta con nuestro proyecto y en primer lugar vemos el fichero *pom.xml*. En este fichero tenemos toda la información acerca de las versiones que estamos utilizando. Si en algún momento se desea cambiar alguna configuración de este estilo se debe modificar en este fichero.

A continuación vamos a la carpeta *src* donde se sitúan los código fuente que podremos modificar. Aparecerá el fichero *AppComponent.java* que es el que vamos a editar en nuestras sucesivas aplicaciones.

Para ello, modificamos el fichero *AppComponent.java* importando el proyecto desde el programa Eclipse.

Nos aparecerá por defecto un código similar al de la Ilustración

En él se destacan los 2 métodos principales: *activate* y *desactivate.* El método *activate* será aquel que se ejecute cuando activamos la aplicación, mientras que el *deactivate* se ejecuta cuando desactivamos la aplicación en ONOS.

Una vez tengamos las aplicaciones finalizadas, o simplemente queramos compilar y enviársela al controlador ONOS lo que debemos hacer es situarnos en la carpeta principal de nuestro proyecto y compilar con el comando:

*mvn clean install*

Esto lo que nos genera es una carpeta *target* con un fichero .oar que será el que enviemos al controlador con el comando:

*onos-app IP-ONOS install! target/fichero.OAR*

La exclamación es opcional e implica que la aplicación se active automáticamente en la aplicación. Si no queremos que ocurra esto simplemente la quitamos manteniendo el resto del comando.

Como detalle, cabe destacar que si la aplicación ya ha sido instalada previamente y estamos enviándola otra vez el comando se modifica ligeramente quedando de la siguiente forma:

*onos-app IP-ONOS reinstall! target/fichero.OAR*

Una vez detallado el proceso previo, nos centramos en las aplicaciones desarrolladas

4.1 Aplicacion *severalpings*

Esta aplicación es una adaptación de la aplicación *oneping (PONER REFERENCIA)* cuyo objetivo principal es limitar el número de pings que se pueden enviar entre 2 hosts cualesquiera.

Esto tiene como principal función el no saturar la red o evitar ataques por inundación.

En primer lugar vamos a detallar el funcionamiento de los *pings*.

4.1.1 Explicación teórica

El *ping* es un comando que permite comprobar si existe conexión entre 2 hosts cualesquiera. Para ello se basa en el envío de paquetes ICMP. El host que ejecuta el comando envia un paquete ICMP de tipo REQUEST al destino. En caso de que el host reciba este paquete responde con un paquete ICMP de tipo REPLY.

La forma de ejecuta el comando en Linux es:

ping -opciones IP -HOST

Y las opciones más utilizadas que usaremos son:

* -c *count:* Indica en la variable *count* el número de pings que se van a enviar.
* -s *size:* Modifica el tamaño del paquete a enviar. Por defecto son 84 bytes

Una vez explicado el funcionamiento teórico se procede a detallar la programación de la aplicación. Para ello se va a dividir la misma en 3 partes diferenciadas que se proceden a detallar.

4.1.2 Interceptar los paquetes

En primer lugar para poder realizar la aplicación es necesario que todos los pings del tipo ICMP REQUEST sean enviados al controlador. Esto tiene como objetivo poder contabilizar los que se han enviado y saber si son más o menos del límite prefijado.

Para poder interceptar estos paquetes es necesario definir un servicio de paquetes (*packetService*) de la siguiente forma:

@Reference(cardinality = ReferenceCardinality.**MANDATORY**)

**protected** PacketService packetService;

Notar que todos los servicios que iremos definiendo tanto en esta como en las aplicaciones sucesivas se definirán de la misma forma.

Con este packetService accedemos al método requestPackets cuya finalidad es enviar los paquetes que cumplan un determinado selector al controlador.

Este selector tiene que identificar los paquetes ICMP tipo REQUEST que lleguen al Open vSwitch de nuestra red para enviarlos al controlador de la manera que se indica a continuación:

LA PARTE DE CODIGO DEJARLA CON EL FORMATO QUE SE TRAE DESDE ECLIPSE (CAMBIARLO EN WORD POSTERIORMENTE)

**private** **final** TrafficSelector intercept = DefaultTrafficSelector.builder()

.matchEthType(Ethernet.**TYPE\_IPV4**).matchIPProtocol(IPv4.**PROTOCOL\_ICMP**).matchIcmpType(ICMP.**TYPE\_ECHO\_REQUEST**)

.build();

4.1.3 Procesando los paquetes

Una vez tenemos controlado que todos los ICMP van a llegar al controlador es momento de procesarlos. Para ello definimos una clase que implemente a *PacketProcessor* y comprobamos que de todos los paquetes que llegan al controlador cogemos solo los que nos interesan, esto se accediendo al paquete que se está procesando en cada momento. Para ello definimos el método que is*IcmpPing* que tiene como argumento el paquete Ethernet procesado. Una vez realizada la comprobación se llama al método *processPing*. Este método obtiene la dirección MAC origen y destino del paquete procesado y accede a un hashMap definido previamente que contiene el número de pings realizados previamente entre esa correspondencia MAC origen-destino. Si el número de pings es menor que el máximo predefinido se realizan 2 operaciones:

* Se aumenta en 1 el valor de pings enviados en el hashMap.
* Se crea una tarea utilizando la clase *schedule* programada para el tiempo de baneo máximo prefijado que quite del hashMap el ping enviado.

Por contra, si el número de pings es mayor que el máximo predefinido lo que se hace es bloquear ese paquete y llamar al método banPings.

Este método crea una regla en el controlador que este enviará al Open vSwitch que regula el tráfico. La parte de código correspondiente a esta parte se presenta a continuación:

TrafficSelector selector = DefaultTrafficSelector.builder()

.matchEthSrc(src).matchEthDst(dst).matchEthType(Ethernet.**TYPE\_IPV4**).matchIPProtocol(IPv4.**PROTOCOL\_ICMP**).matchIcmpType(ICMP.**TYPE\_ECHO\_REQUEST**).build();

TrafficTreatment drop = DefaultTrafficTreatment.builder()

.drop().build();

flowObjectiveService.forward(deviceId, DefaultForwardingObjective.builder()

.fromApp(appId)

.withSelector(selector)

.withTreatment(drop)

.withFlag(ForwardingObjective.Flag.**VERSATILE**)

.withPriority(**DROP\_PRIORITY**)

.makeTemporary(**TIMEOUT\_SEC**)

.add());

Fundamentalmente una regla de flujo se compone de 2 elementos.

* El selector de tráfico que en este caso lo que hace es coger todos los paquetes tipo ICMP REQUEST que vayan entre ese origen y destino, permitiendo de esta forma poder enviar el resto de tráfico (incluidos los ICMP REPLY que se puedan originar de otros pings)
* El tratamiento a realizar para ese tráfico seleccionado previamente. En este caso se trata simplemente de descartarlo, pero otras acciones que pueden ocurrir son por ejemplo enviar el tráfico por otro puerto.

A mayores de estos 2 elementos seleccionamos la prioridad que tendrá la regla (que será superior al de un envío normal) y la temporalidad de la regla que vendrá predefinida previamente.

4.1.4 Escuchando los flujos

Una vez creamos la regla que sabemos que tiene una duración limitada creamos un *Listener* cuya implementación se puede ver más abajo con el objetivo de que esté pendiente de cuando una regla es eliminada (porque ha pasado su tiempo de vida determinado) para, de esta forma, crear un mensaje en el *log* que nos informe que el enlace entre los 2 hosts baneados vuelve a estar disponible para mandar otra vez pings.

flowRuleService.addListener(flowListener);

4.1.5 Parámetros configurables

Finalmente dentro de esta aplicación se han creado 2 parámetros configurables mediante la CLI de onos para evitar tener que modificar el código en caso de querer variar su valor. En concreto los parámetros configurables son el número máximo de pings que permitiremos entre 2 hosts y el tiempo que mantendremos el enlace baneado para el envío de mas pings. Los valores por defecto se pueden encontrar en el fichero *OsgiPropertyConstants.java* y son 7 pings como máximo y 60 segundos de baneo.

La forma de declarar parámetros configurables es así:

@Component(

immediate = **true**,

service = AppComponent.**class**,

property = {

**MAX\_PINGS** + ":Integer=" + **MAX\_PINGS\_DEFAULT**,

**TIME\_BAN** + ":Integer=" + **TIME\_BAN\_DEFAULT**,

}

)

Resaltando que el servicio que hemos de declarar es cfgService

Cuando hayamos declarado los parametros declaramos un método modified cuya implementación puede verse en el código que se ocupa de actualizar el valor de los parámetros una vez los hayamos cambiado.

Para poder ver los parámetros que tiene una aplicación concreta y modificarlos es necesario ir a la CLI de ONOS. Los comandos fundamentales son:

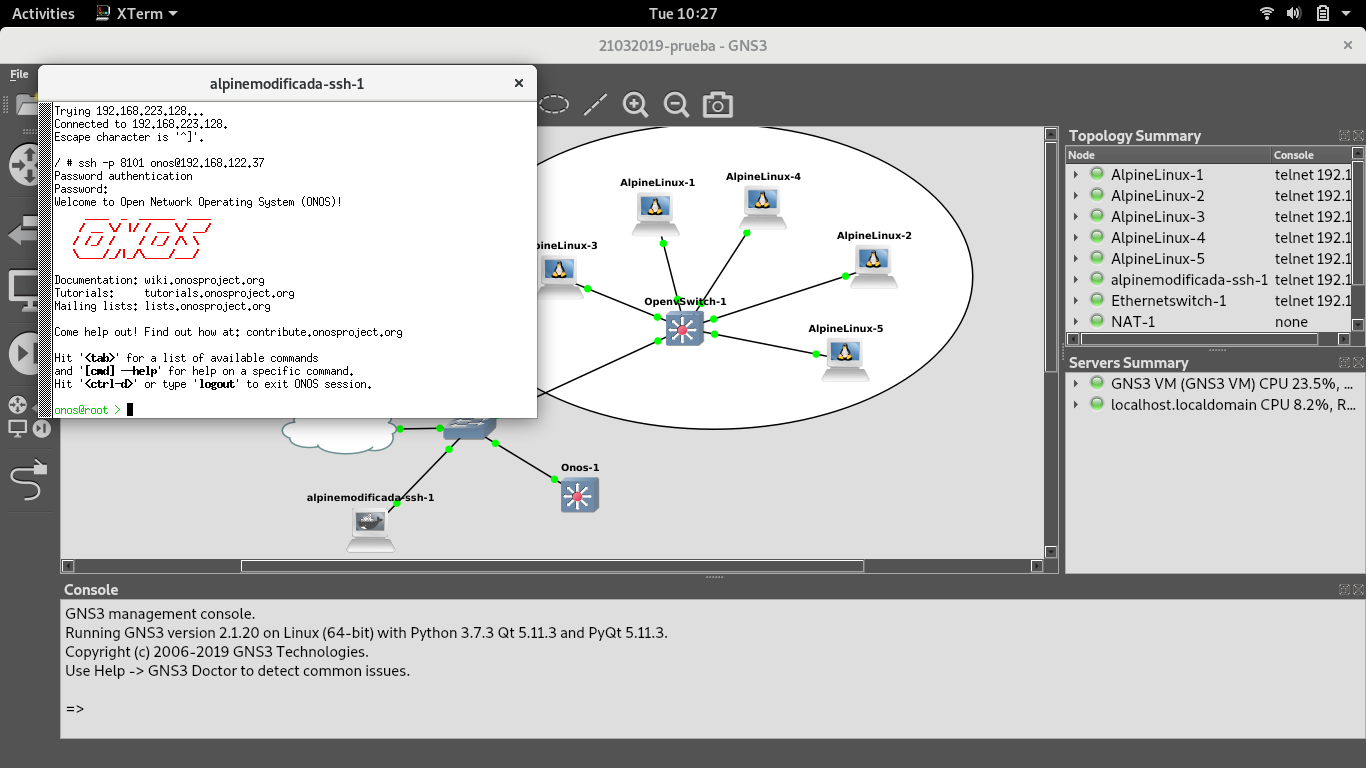
* cfg: Lista todos los nombres de las clases que tienen parámetros configurables
* cfg get componentClass: Lista todas las propiedades de la clase especificada
* cfg get componentClass name: Lista el valor de la propiedad especificada
* cfg set componentClass name value: Modifica el valor de la propiedad especificada
* cfg set componentClass name: Reestablece el valor de la propiedad especificada a su valor por defecto

PONER CAPTURA DE LA ALPINEMODIFICADA CON ESTOS COMANDOS PARA QUE QUEDE MAS CLARO

4.1.6 Banco de pruebas

En este apartado se van a realizar diferentes pruebas para comprobar el funcionamiento de la aplicación, asi como incluir diversas capturas que permitan clarificar el funcionamiento de la misma.

En primer lugar, nos conectamos mediante ssh al controlador ONOS desde el host *alpinemodificada*

**

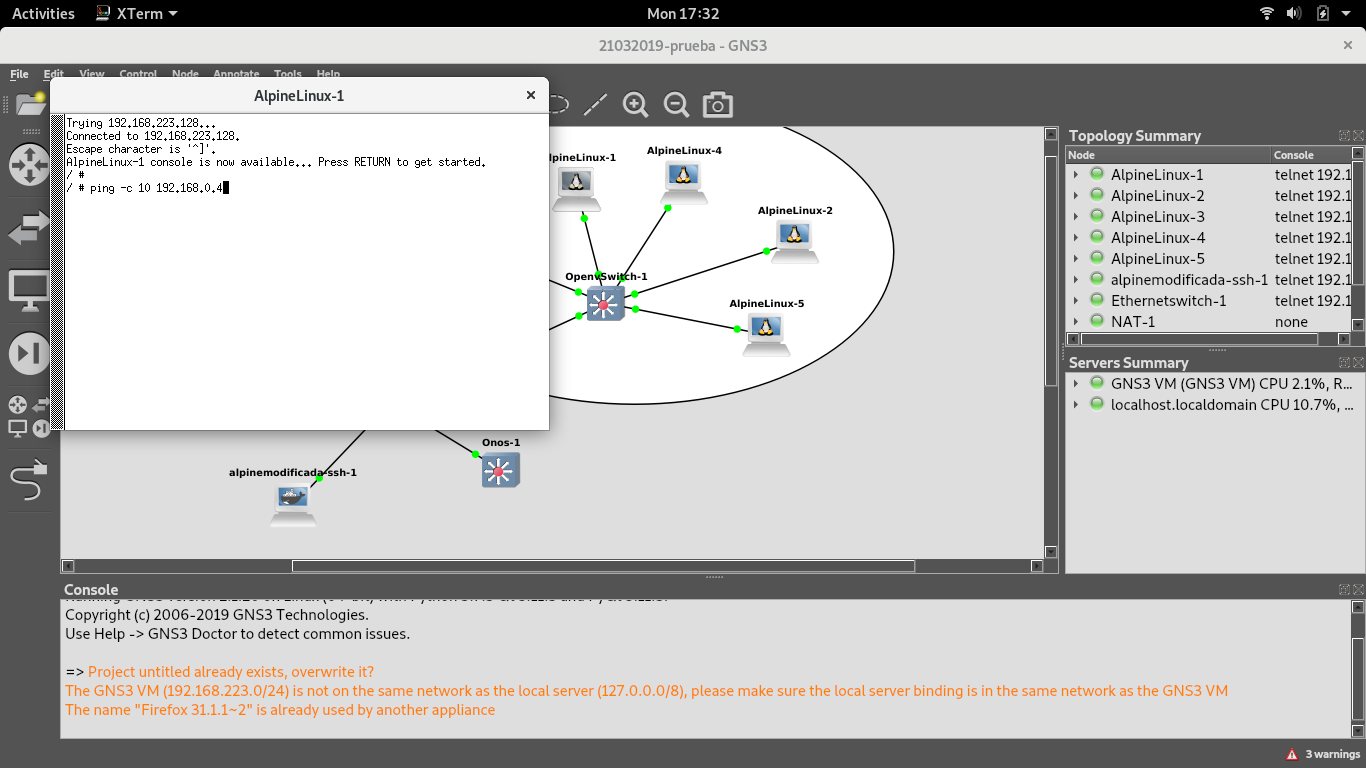
A continuación, activamos las aplicaciones necesarias y comprobamos en el controlador que se han activado correctamente:

Tal y como se puede ver se muestra el mensaje informativo de que la aplicación se ha activado correctamente, así como el valor de los parámetros configurables que tiene la aplicación. En este caso se observa como el número máximo de pings permitidos son 7 y que, en caso de superarlo, el tiempo de baneo es de 60 segundos.

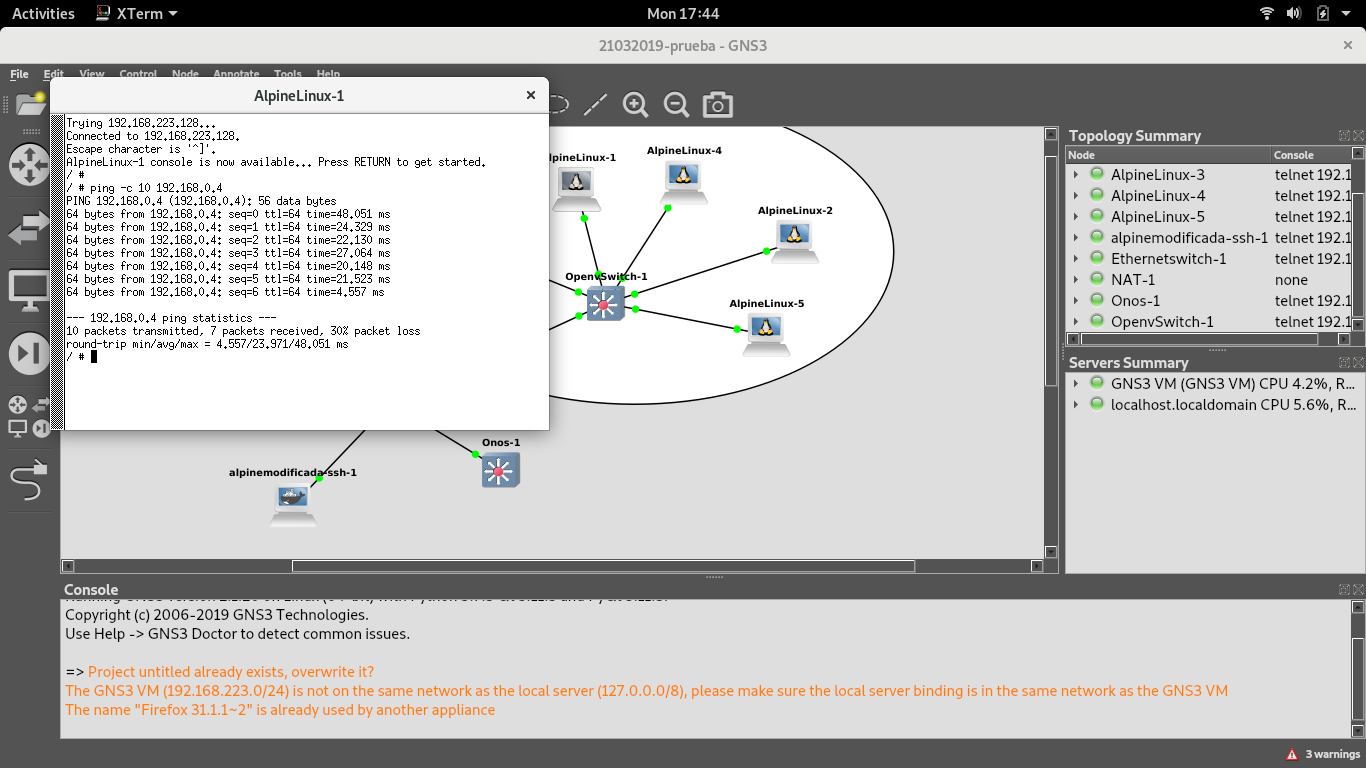
4.6.1.1 Envio de pings entre 2 hosts

Como primera prueba, vamos a realizar *pings* entre 2 hosts cualesquiera de nuestra red, por ejemplo enviaremos desde la máquina Alpine-1 a la máquina Alpine-4 y comprobaremos si una vez superado el umbral se realiza el baneo.

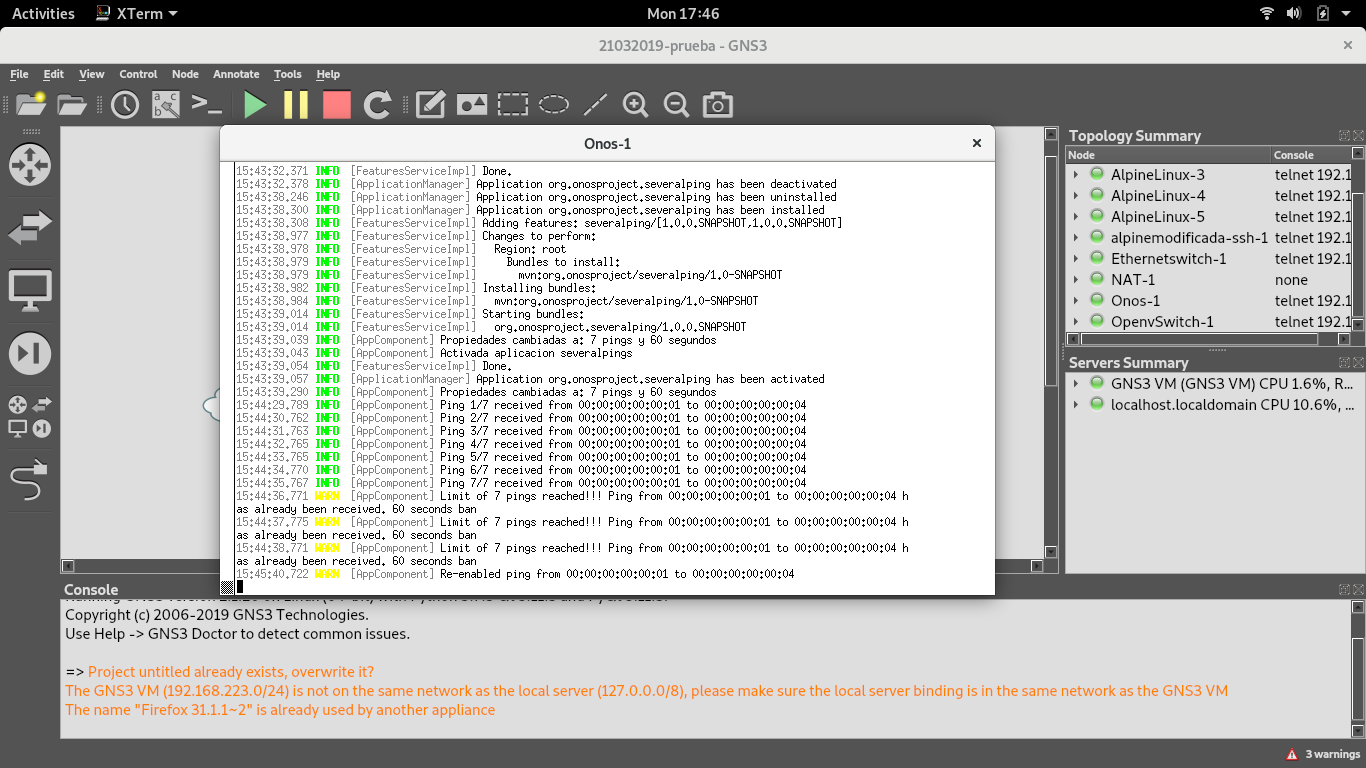
Para ello enviamos 10 pings tal y como se puede ver en la Ilustración



Ejecutamos el comando:



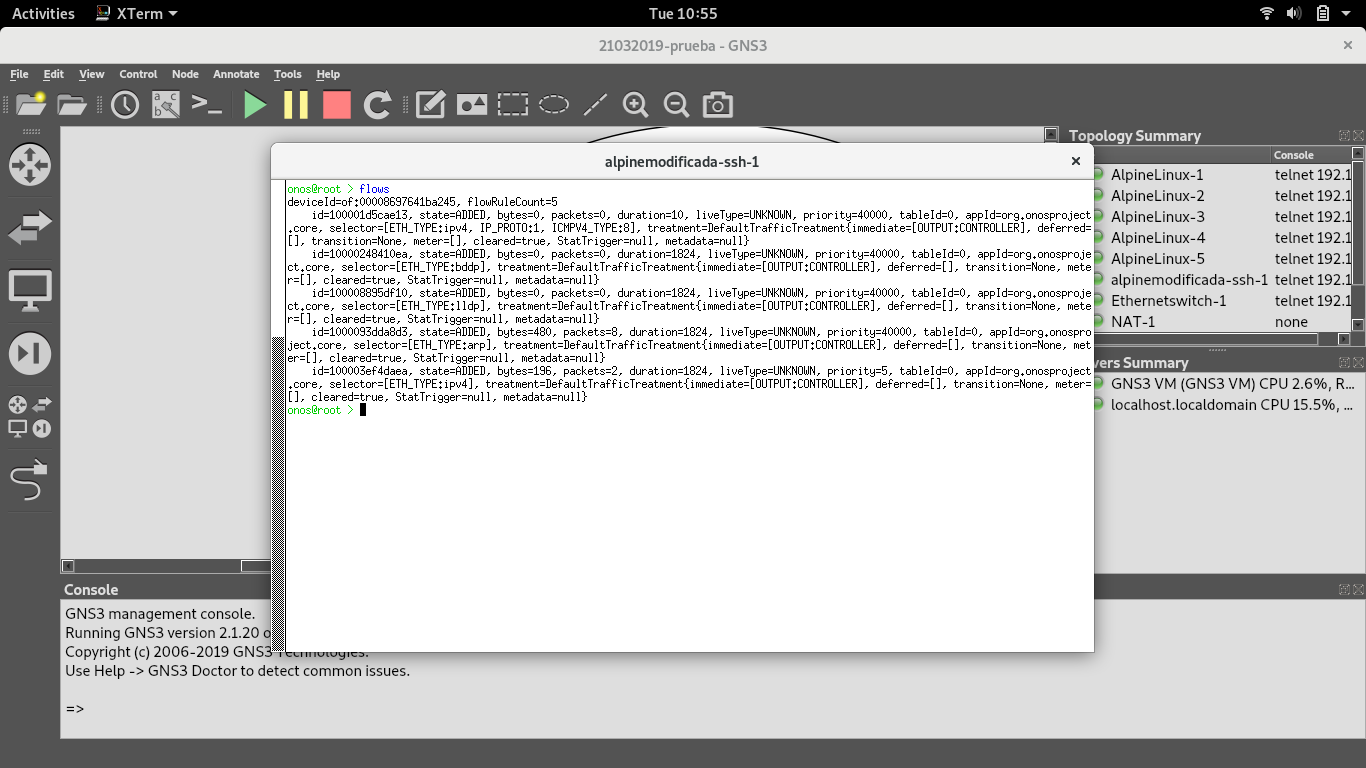
Vemos como efectivamente, de los 10 pings que hemos enviado 7 han llegado a su destino, mientras que los otros 3 han sido bloqueados por el controlador. Si vamos a la ventana del controlador podemos comprobarlo

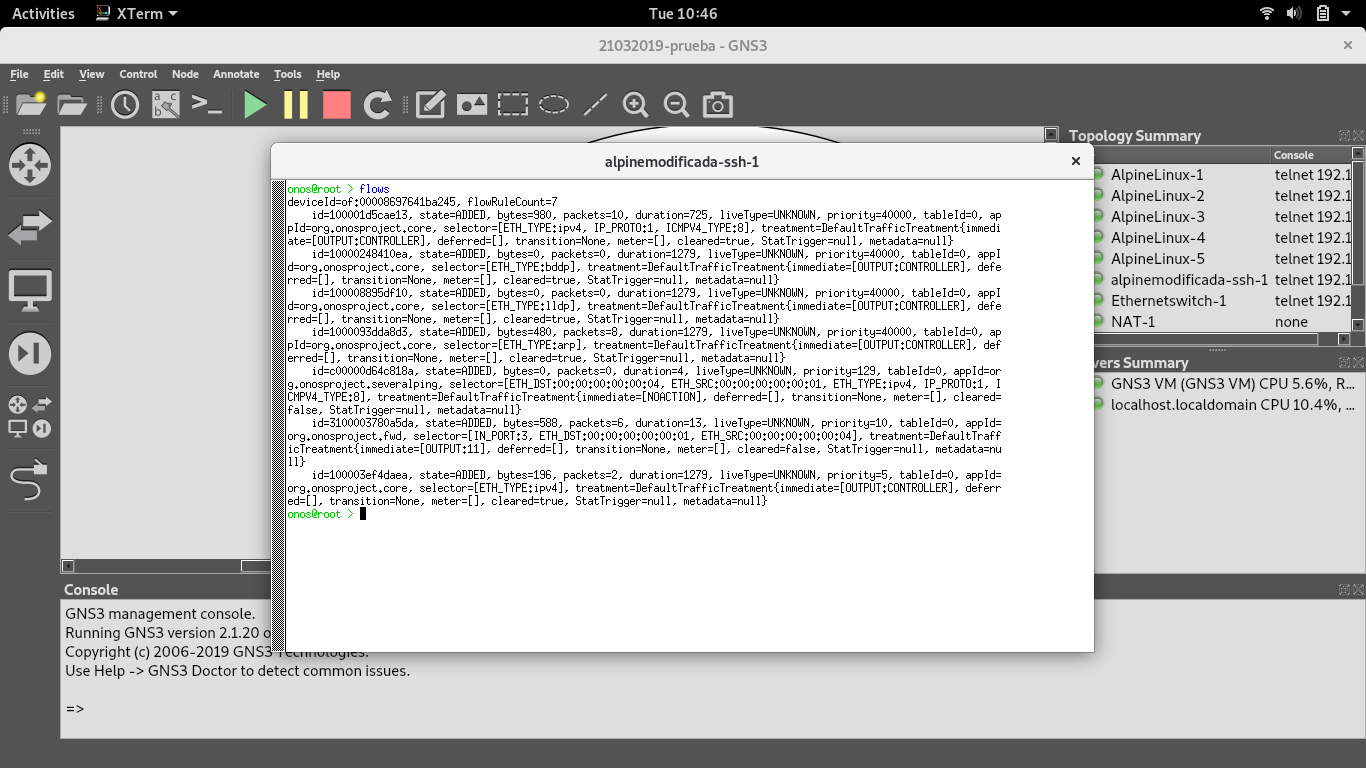


Vemos como se notifica la llegade de los pings desde el Alpine-1 al Alpine-4. En el momento en el que se supera el umbral máximo permitido se notifica con un *warning* por cada *ping* de más que llega. En este caso aparece el mensaje 3 veces.

Tambien se observa que 1 minuto después se notifica que el enlace ha sido reestablecido permitiendo de nuevo el envío de los 7 pings.

Finalmente comprobamos que el controlador ha instalado los flujos en el open Vswitch. Para ello vamos a la máquina *alpinemodificada* y ejecutamos el comando *flows* en 2 momentos, cuando se ha activado la aplicación y cuando se ha baneado el enlace:

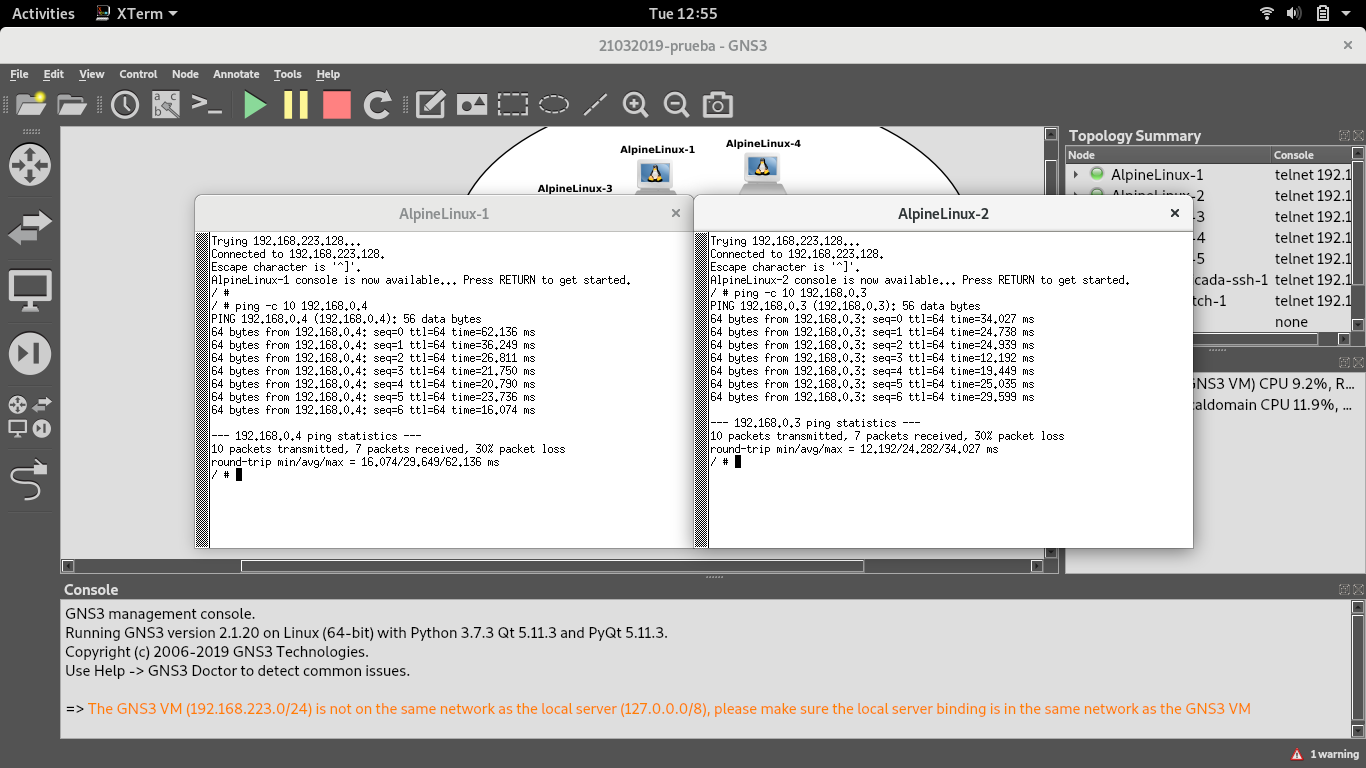




EXPLICAR DIFERENCIAS ENTRE LAS 2 CAPTURAS (CUANDO LAS RECORTE QUE NO VEO QUE PONE JAJA)

4.1.6.2 Enviando pings entre 2 parejas de hosts

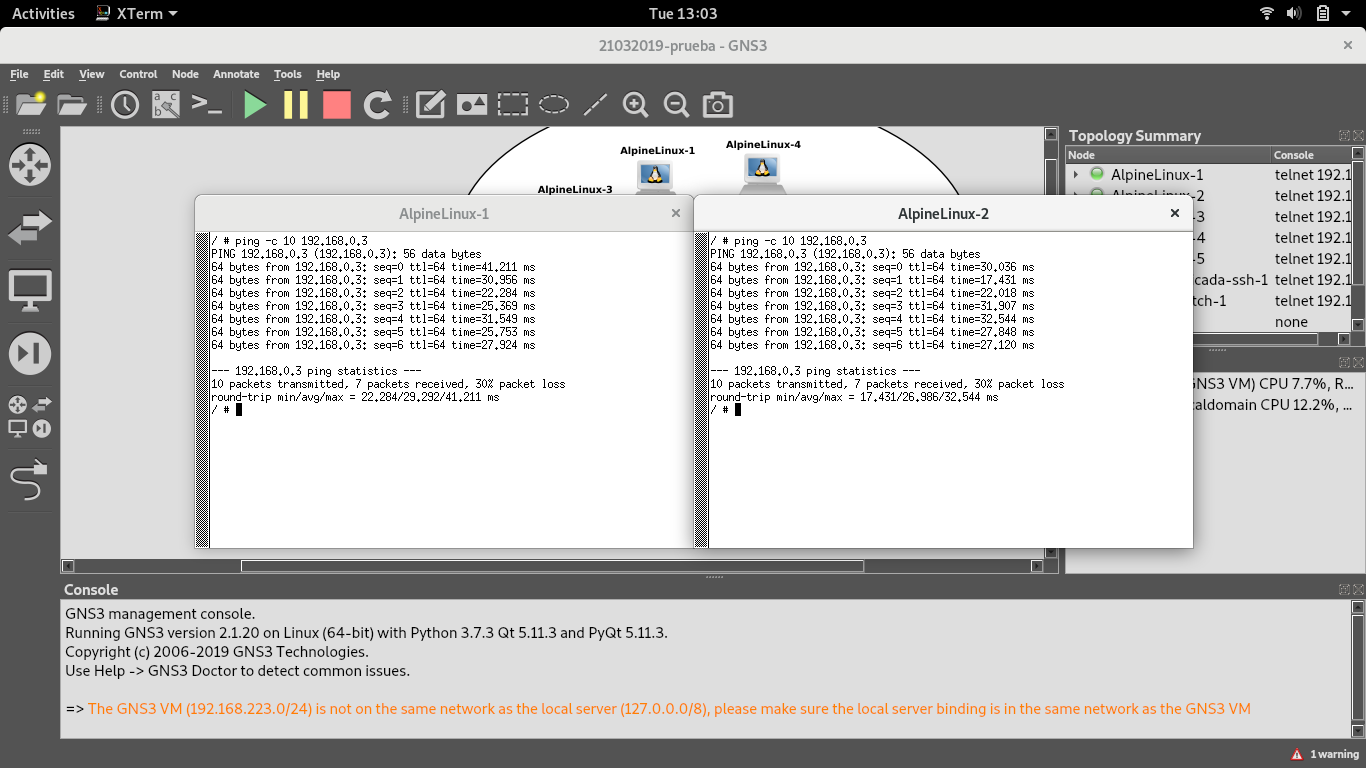
El objetivo de esta prueba es comprobar que aunque se manden pings entre varios pares de hosts solo se cuentan para el baneo aquellos que coinciden en MAC origen y destino en vez de contar los pings totales que circulan por la red.

Por tanto, enviamos 2 pings entre 2 pares de hosts diferente, por ejemplo enviamos 10 pings entre la Alpine-1 y la Alpine-4 y otros 10 pings simultáneamente entre la máquina Alpine-2 y la máquina Alpine-3

Se observa como se permite el envío de 7 pings por cada pares de hosts, mientras que los otros 6 (3 por cada par) han sido baneados.

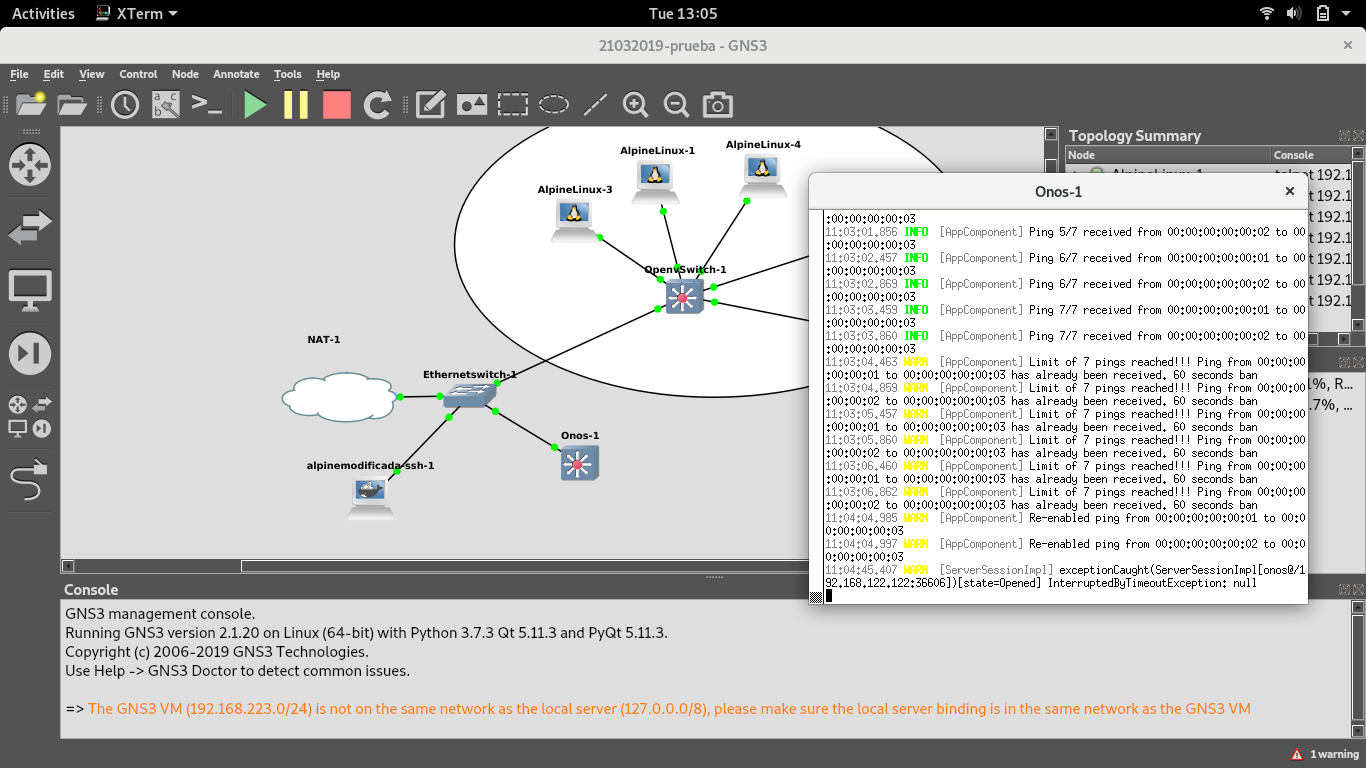
4.1.6.3 Enviando pings desde 2 hosts a un tercero

En este caso el objetivo será enviar 10 pings desde 2 hosts a otro, por ejemplo desde Alpine-1 y Alpine-2 a Alpine-3, con la finalidad de comprobar si el baneo se realiza a los 7 pings de cada host, o se realiza la suma de ambos.



Como vemos, y tal como era de esperar, funciona correctamente el hecho de que el baneo se ha producido en función del par MAC origen-destino, por tanto, se permite que otras correspondencias si puedan interactuar.

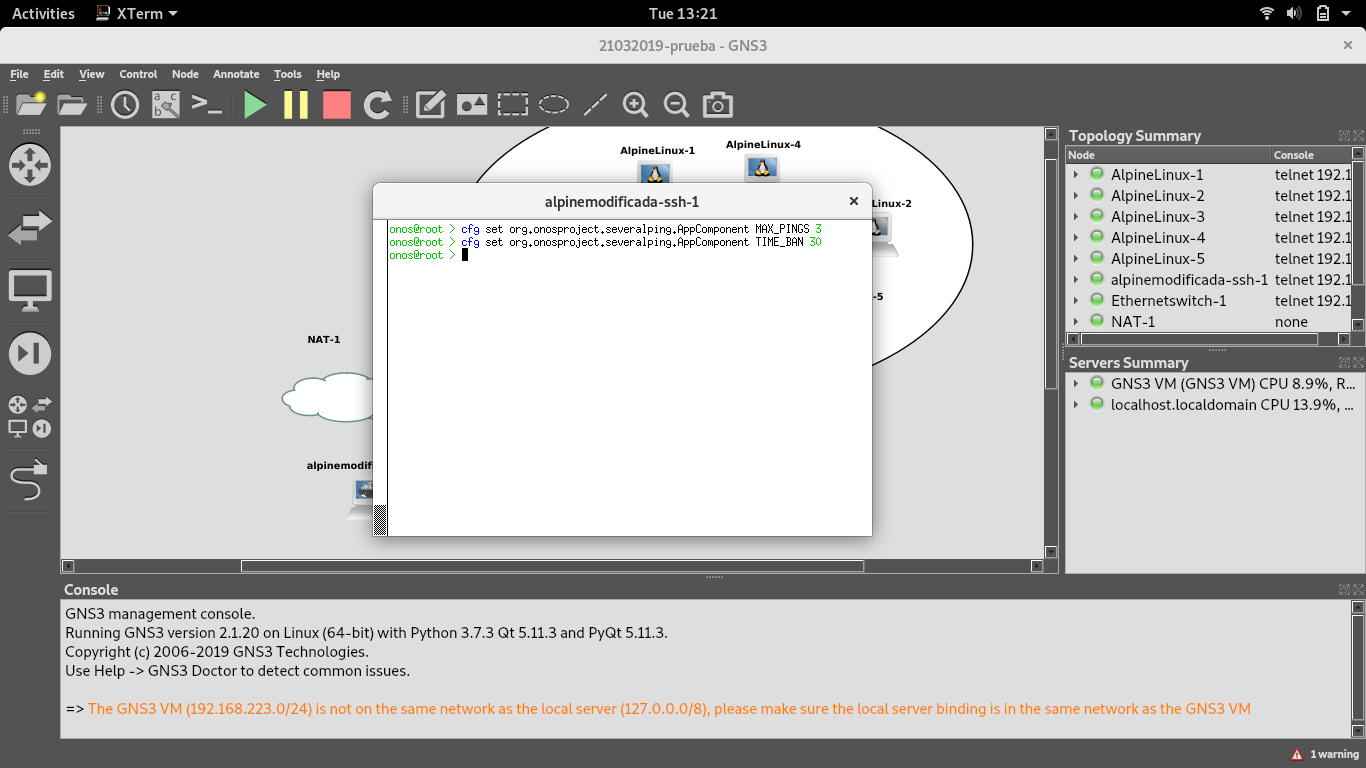
Asimismo, vemos como transcurrido el tiempo de baneo se desbloquean los 2 enlaces baneados.



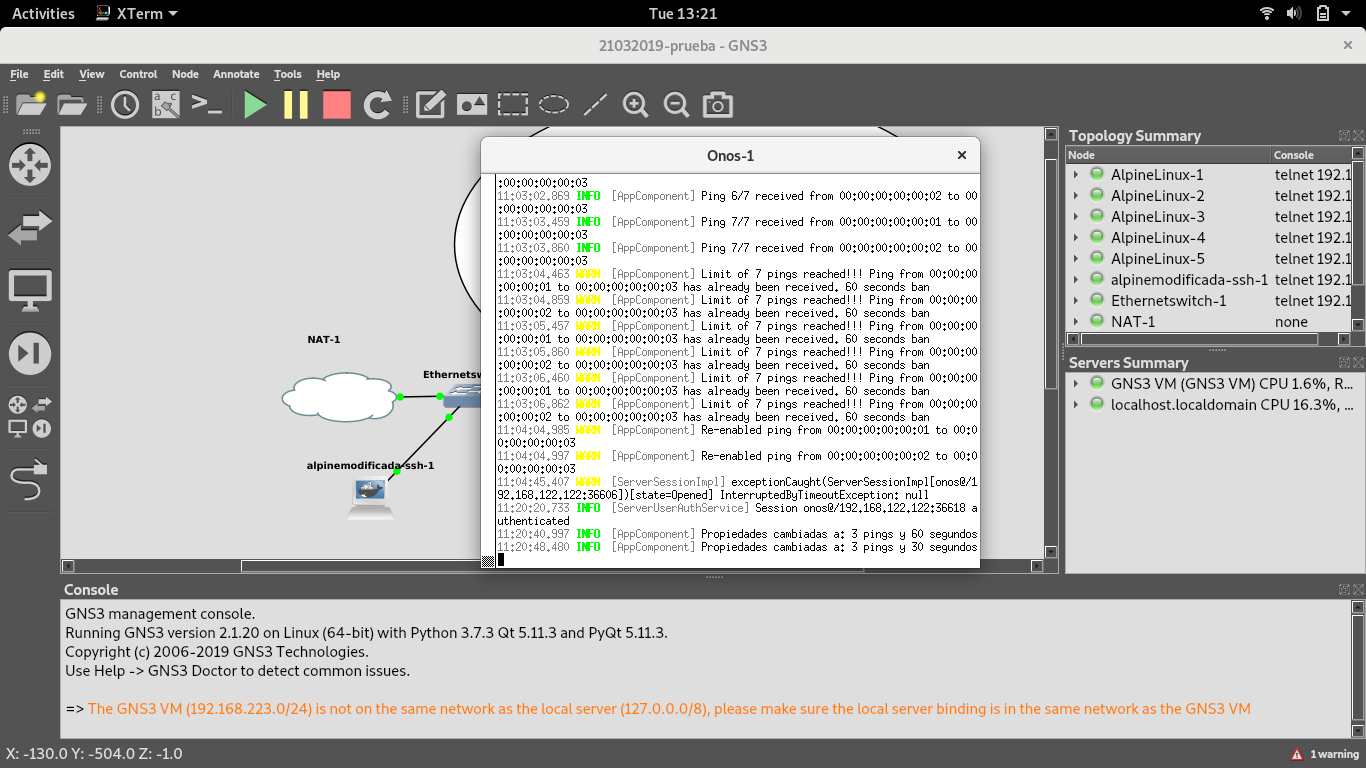
4.1.6.4 Modificando los parámetros configurables

En esta última prueba se va a comprobar el funcionamiento de los parámetros configurables, que se recuerda que son el número de pings máximos permitidos, cuya variable se denomina MAX\_PINGS, y el tiempo que permanece el enlace baneado, cuya variable se llama TIME\_BAN

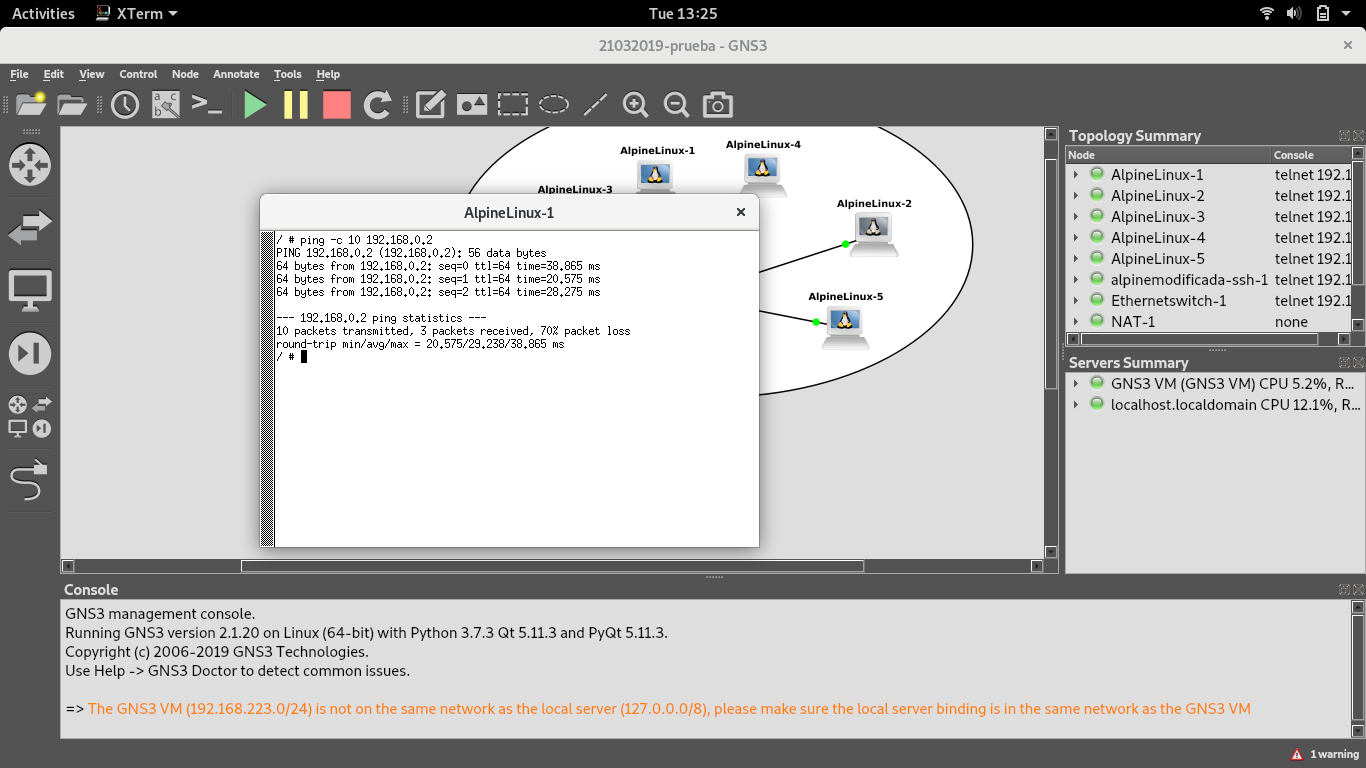
Por ejemplo, tal y como se ve en la Ilustración se ve que los parámetros nuevos van a ser permitir únicamente 3 pings y que el tiempo de baneo pase a ser de 20 segundos.



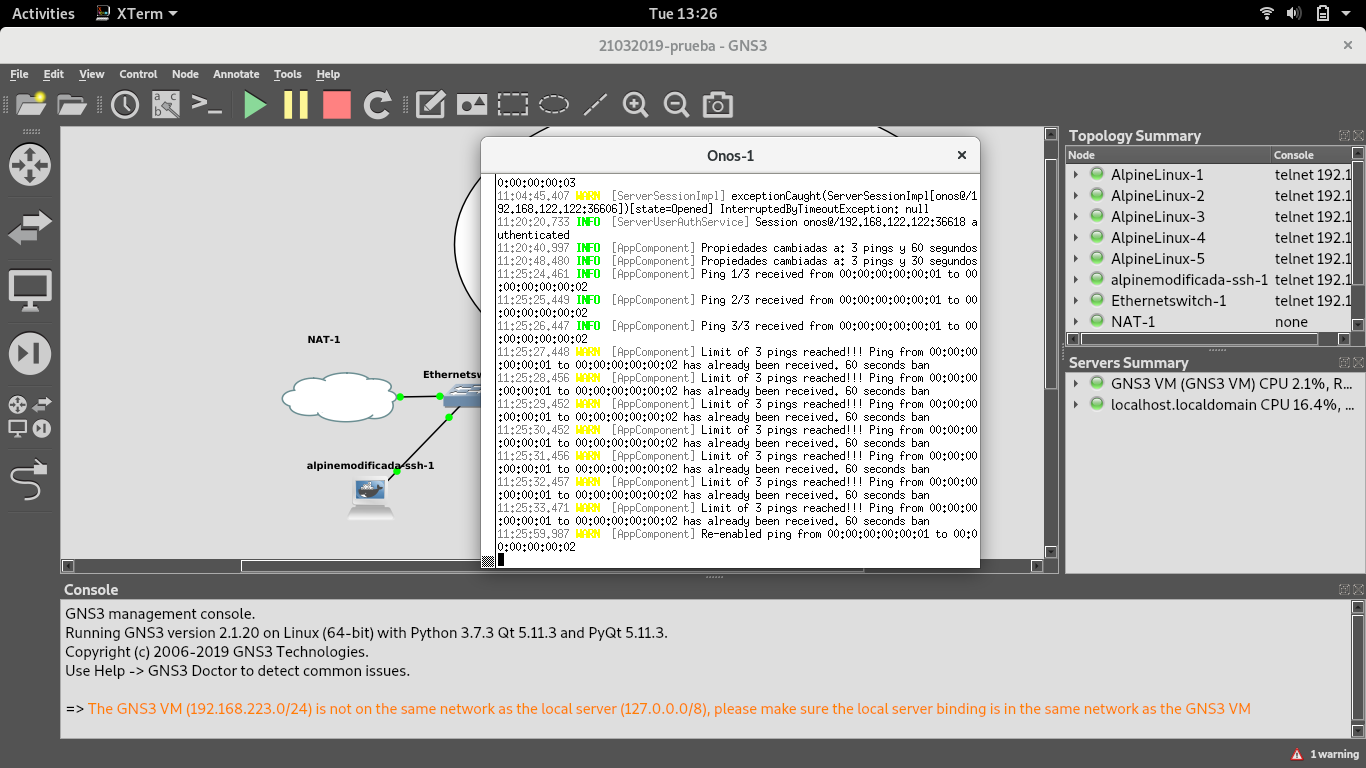
Si vemos el controlador, comprobamos que, efectivamente, se han producido correctamente los cambios.



Una vez modificados los parámetros, a continuación, comprobamos que funcionan, para ello mandamos nuevamente 10 pings desde Alpine-1 a Alpine-2



Vemos como efectivamente de los 10 en esta ocasión solo llegan 3 mientras que los otros 7 han sido baneados. Lo comprobamoso viendo el log en el controlador



Vemos como automaticamente se ha modificado el limite a 3 pings y, observando los tiempos en los que se produce cada evento comprobamos tambien que el tiempo de baneo se ha reducido hasta 20 segundos aproximadamente.

4.2 Analizando las estadísticas de tráfico

Con el objetivo de analizar las estadísticas de tráfico y poder usarlas para determinados objetivos se han realizado 3 aplicaciones que se proceden a explicar:

4.2.1 Aplicación *statsshow*

El objetivo de esta aplicación es informativo, ya que muestra las estadísticas de tráfico de todos los puertos de los dispositivos que controlemos. En este caso los puertos del Open vSwitch. Esta aplicación es muy sencilla y únicamente tiene un apartado.

4.2.1.1 Tareas repetidas

Para llevar a cabo el funcionamiento correcto de la tarea es necesario crear una tarea que se repita periodicamente. Esto se hace creando un temporizador de la siguiente forma:

timer = new Timer("Timer");

Y le definimos una periodicidad y un retardo:

timer.scheduleAtFixedRate(repeatedTask, delay, period);

Donde:

* repeatedTask: es el nombre de la tarea y dentro de ella se ejecuta el método *run* donde está implementado el código que se debe repetirse.
* Delay: Es un atributo de tipo long que indica el retraso desde que se activa la aplicación hasta que se ejecuta por primera vez la tarea. Está definido a un segundo.
* Period: Es otro atributo de tipo long que indica la periodicidad de la tarea, es decir, cada cuanto tiempo se repite la tarea. Este atributo es un parámetro configurable, por tanto puede modificarse desde la CLI de ONOS tal y como se ha explicado en el apartado 4.1.5

Es importante destacar que en el método *deactivate* hay que cancelar el temporizador, de lo contrario la tarea se quedará ejecutandose indefinidamente. Para ello se ejecuta la orden:

timer.cancel();

Dentro de la tarea el objetivo, como se ha comentado, es obtener las estadisticas de tráfico de los puertos de cada dispositivo. Por tanto es necesario implementar 2 bucles *for*; el primero que recorra todos los dispositivos y el segundo que recorra todos los puertos de ese dispositivo. Tanto los dispositivos existentes como los puertos se pueden obtener a través de *deviceService*

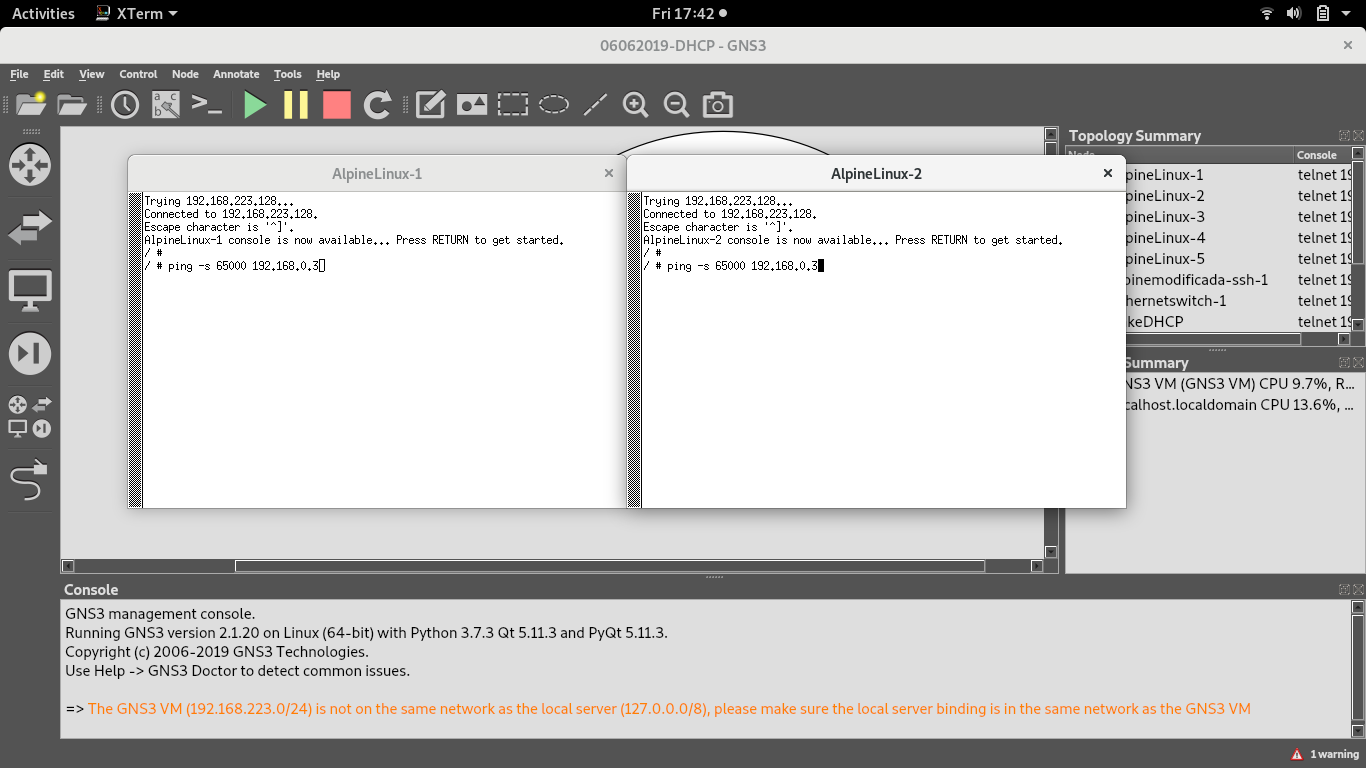
Una vez creados los 2 bucles necesarios obtenemos las siguientes estadisticas, en concreto obtendremos las siguientes estadísticas:

* bytes enviados por cada puerto desde que se activó la aplicación
* bytes recibidos por cada puerto desde que se activó la aplicación
* bytes enviados por cada puerto en cada ejecución de la tarea
* bytes recibidos por cada puerto en cada ejecución de la tarea

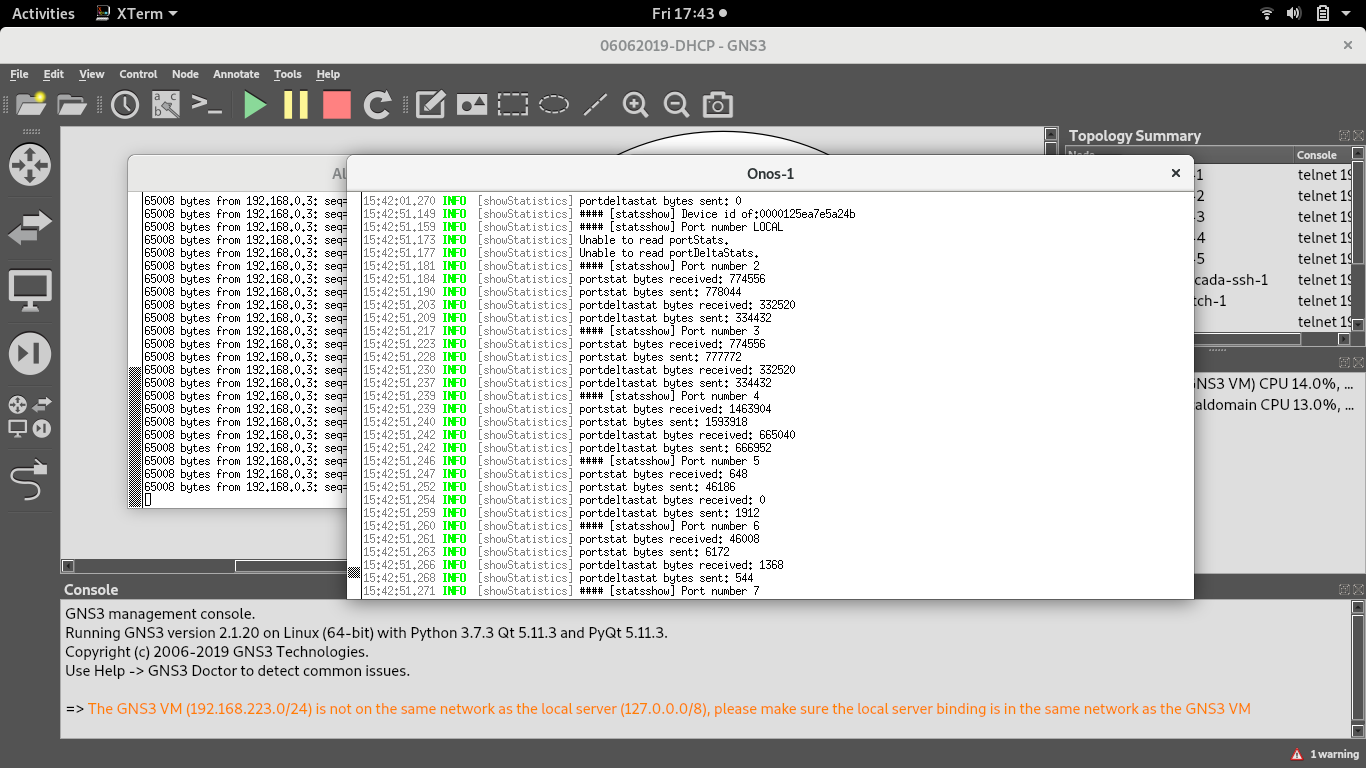
4.2.1.2 Banco de pruebas

En este caso, dado que la aplicación es muy sencilla únicamente se va a poner en marcha la aplicación y enviar una serie de pings para comprobar que se recogen las estadísticas correctamente.

En concreto, 2 hosts, Alpine-1 y Alpine-2, van a enviar pings a Alpine-3.



Esperamos que transcurran un par de iteraciones de la tarea programada y comprobamos en el controlador los parámetros recogidos por la aplicación



Tal y como se puede observar, en primer lugar se notifica el dispositivo y a continuación se recorren todos los puertos que lo conforman.

A continuación, vemos que los puertos con más actividad son los 2,3 y 4 que es donde están conectados los *hosts* que han enviado o recibido los diferentes paquetes ping.

En concreto, vemos como, efectivamente, los puertos 2 y 3 en total han enviado y recibido aproximadamente la misma cantidad de datos, mientras que el puerto 4 recibe y envia el doble de tráfico, esto es debido a la suma agregada de los 2 enlaces que le están enviando pings. El hecho de que sea el doble de tráfico es lo que indica que la aplicación recoge correctamente las estadísitcas.

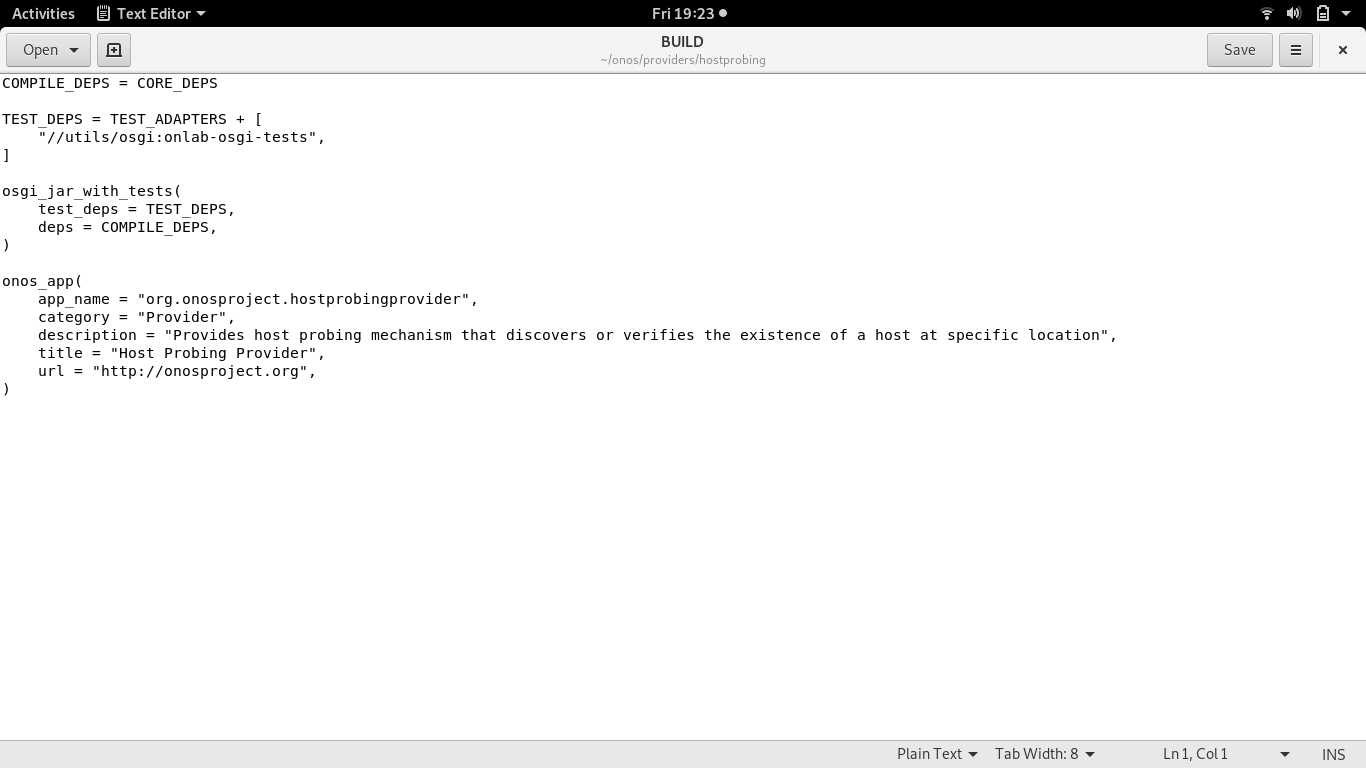
Por otro lado, tambien se ven las *deltaStatistics* que tal y como se ha mencionado recogen los datos que se han enviado o recibido en la última iteración. Como se han dejado 2 iteraciones de la aplicación, y el ping se ha mantenido durante todo el tiempo estas estadísticas deben ser la mitad de las totales, algo que se cumple, por tanto se concluye que la aplicación funciona correctamente.

4.2.2 Aplicación detectHost

El objetivo de esta aplicación es conocer todos los hosts que tenemos conectados a los Open vSwitch de la red.

Hay que resaltar que para el buen funcionamiento de la aplicación es necesario instalar una aplicación predefinida en el controlador a parte de las habituales ya comentadas, esta es la app org.onosproject.*hostprobingprovider.* El motivo de necesitar una aplicación mas es porque se va a utilizar el servicio *hostProbing* y a diferencia de los utilizados hasta ahora no se activa por defecto al arrancar el dispositivo ONOS. Es esta aplicación la que registra este servicio. El código de la aplicación puede encontrarse en la siguiente ruta *home/onos/providers/hostprobing*

En el fichero BUILD se registra la aplicación tal y como se ve en la siguiente Ilustración



mientras que en el fichero *DefaultHostProbingProvider.java*  se ve como se activa el servicio y la implementación de los métodos que se van a utilizar posteriormente.

Centrandonos ya en la aplicación *detectHost* vemos que se puede dividir en 2 partes:

4.2.2.1 Tarea repetida

En este caso, a diferencia de la aplicación anterior, se utilizan 2 tareas repetidas, dentro de la primera se utiliza el servicio definido previamente *hostService*. Este servicio nos devuelve todos los hosts que procedemos a detallar mediante un bucle que los recorra. Cabe destacar que para que un host sea detectado por este servicio es necesario que haya mandado algún paquete al controlador.

En la segunda tarea, lo que se hace es, mediante el servicio *hostProbing* mandar un paquete de prueba a todos los hosts para ver si siguen activos, o por el contrario se han desconectado. Nuevamente destacar que para que un hots se reconozca como desconectado debe estarlo físicamente, es decir, no estar conectado mediante un cable al Open vSwitch.

Finalmente, recordar que es necesario en el método *deactivate* cancelar los 2 temporizadores creados para evitar que se queden residuales.

4.2.2.2 Listener

Además de las tareas repetidas, el código incluye un *HostListener* cuyo objetivo es estar pendiente de cuando un host sea eliminado notificarlo mediante un mensaje informativo al usuario que aparecerá en el controlador ONOS.

Destacar que como HostListener viene definido como una interfaz es necesario instanciarlo previamente de la forma que procede:

private final HostListener hostListener = new InternalHostListener();

4.2.2.3 Banco de pruebas

4.2.3 Aplicación detectHostBan

En esta aplicación el objetivo es realizar un seguimiento de los hosts conectados a la red actual y banear a aquellos que superen un determinado umbral durante un cierto umbral. La finalidad más clara es evitar que los hosts envíen una cantidad muy elevada de datos.

Para el desarrollo de la aplicación se utilizarán aspectos de las 2 aplicaciones mencionadas anteriormente ya que, en primer lugar se tienen que detectar los hosts que hay en la red para lo cual se usará lo explicado en la aplicación detectHost.

En segundo lugar, es necesario, para todos los hosts detectados conocer sus estadísticas, para lo cual se usará código de la aplicación statsshow, utilizando las delta estadísticas dentro de un temporizador. En cada iteración del bucle iremos sumando el valor e introduciendolo en un HashMap que relaciona la MAC del host con los datos acumulados que lleva desde que se activó la aplicación.

Una vez se ha superado el umbral, todo el tráfico es baneado. Destacar, que a diferencia de la aplicación *severalpings* se banea todo tipo de tráfico bidireccionalmente, es decir, el host no puede ni enviar ni recibir dato alguno durante un tiempo que está definido como un parámetro configurable del estilo a aplicaciones anteriores, para ello se llama al método *banDatas.*

Este método crea 2 reglas de flujo en el Open vSwitch,:

El objetivo del selector es capturar todo el tráfico que pase por el enlace; En la primera regla se captura el tráfico cuyo origen es la MAC baneada, mientras que en el segundo se captura todo el tráfico cuyo destino es la MAC baneada, de esta forma baneamos bidireccionalmente

El tratamiento de estas reglas es idéntico y consiste simplemente en descartar ese tráfico seleccionado.

Finalmente, hacemos que estas reglas sean permanentes y ponemos una tarea programada para el tiempo que hemos definido que la MAC este baneada.

El motivo principal por el que no se hace temporal se explica fácilmente con un breve ejemplo:

Supongamos que tenemos 2 hosts: host A y host B. El host A está enviando una serie de pings y supera el límite predefinido. Se llamaría al método *banDatas* y se crearían las reglas que descartan todo el tráfico.

Supongamos tambien que el tiempo de baneo es mayor que el tiempo de la tarea que actualiza las estadísticas y que el comando *ping* se sigue manteniendo en ejecución (aunque no se esté recibiendo respuesta). Por cada iteración de la tarea se volvería a ejecutar el método *banDatas* y se volverían a crear los flujos, otra vez con el temporizador de baneo reiniciado, por tanto nunca se eliminarían estos flujos y el baneo pasaría a ser a todos los efectos permanente.

Para evitar este problema, directamente creamos las reglas de flujo permanentes y creamos una tarea programada para el tiempo de baneo con el comando timer.schedule().

Dentro del este método se ejecuta a su vez método *run* que lo que hace es eliminar las 2 reglas y actualizar el HashMap volviendo a reiniciar el valor de los datos acumulados para la MAC que ya ha dejado de ser baneada.

Una opción alternativa para este proceso sería enviar todo el tráfico al controlador, hacer la regla temporal y bloquear los paquetes mediante un procesador del mismo con la orden *block,* pero esto aumentaría el flujo de datos enviado al controlador haciendo que se ralentizase la aplicación.

4.3 Aplicación *Vlan*

La siguiente aplicación desarrollada permite asignar diferentes Vlan a los hosts.

4.3.1 Explicación teórica

Una Vlan (red de área local virtual) es una forma de crear redes lógicas independientes dentro de una misma red física. Es decir, poder tener diferentes redes dentro de un mismo enrutador. Esto permite simplificar la administración de la red ya que permite separar que hosts no deben intercambiar datos. Por tanto, en resumen las ventajas que aporta son las siguientes:

* Mayor flexibilidad ya que se facilita el cambio y moviemiento de los dispositivos de la red. Simplemente con cambiar la Vlan en la que se encuentra un host cambiamos por completo la topología de la red virtual
* Mayor seguridad ya que los dispostivos están separados en diferentes Vlan, por tanto su comunicación está mas restringida
* Control del tráfico broadcast, ya que entre subredes no se produce.

A la hora de configurar las Vlan se utiliza el protocolo IEEE 802.1Q. Su propuesta principal es añadir 4 bytes a una trama Ethernet (en lugar de encapsularla)

PONER IMAGEN FORMATO DE TRAMA ETHERNET Y 802.1Q y ver diferencias

Este protocolo permite identificar a una trama perteneciente a una Vlan, de esta forma el tráfico se mandará unicamente por su Vlan.

Para interconectar Vlan se añadió un tipo de puerto en los switches:

* Por un lado están los puertos de acceso, estos conectan con los hosts directamente. Cuando entra una trama Ethernet se le añade la etiqueta de acuerdo con el protocolo 802.1Q, mientras que cuando sale una trama 802.1Q se le quita la etiqueta de tal forma que al host final le llega la trama Ethernet original.
* Por otro lado, están los puerto 1Q Trunk que son los que se utilizan para conectar los switches entre sí y que el tráfico de varias VLANs circule a través de él. En este caso, tanto las tramas entrantes al switch como las salientes van etiquetadas según 802.1Q

EXPLICAR FUNCIONAMIENTO TABLAS OPENFLOW AQUI

Por tanto, el objetivo de esta aplicación será implementar el funcionamiento de este protocolo estableciendo reglas de flujo.

4.3.2 Desarrollo de la aplicación

4.3.2.1 Fichero VlanByMac.java

El código de esta aplicación se va a ir explicando paulatinamente.

Antes de nada, declaramos 3 HashMaps que serán necesarios:

* *macVlanMap*: HashMap que relaciona la Mac de un host con la Vlan a la que pertenece. A la hora de introducir los datos lo haremos con un comando que crearemos especificamente para este fin (ver sección PONER SECCION DONDE SE EXPLICA CUANDO ESTE)
* *vlanRuleMap*: Este HashMap devuelve las reglas de flujo que tiene cada Vlan creada. Más adelante se verá el fin de él
* *macRuleMap*: Este HashMap devuelve para cada Mac todas las reglas de flujo que tiene asociadas.

En primer lugar, nada mas activar la aplicación capturamos todo el tráfico ARP para mandarlo al controlador y que sea este el que decida que hacer con él. Esta regla será fundamental como se verá posteriormente y se instalará en todos los dispostivos que tengamos.

Seguidamente, añadimos tambien un *HostListener,* que tendrá como objetivo notificarnos cuando se añade un host nuevo a la red. FALTA EXPLICAR QUE HACE FALTA ACTIVAR DETECT HOST Y HOSTPROBINGPROVIDER

4.3.2.1.1 Host añadido a la red

Cuando un host se añade a la red lo primero que se hace es obtener la VLAN a la que está asociada obteniendola a partir del *macVlanMap*, incluyendo una excepción para el caso de que esa MAC no tenga asociada en cuyo caso se notifica y se finaliza la ejecución de la aplicación.

En el caso de que si tenga una VLAN se hace una distinción en función del valor recibido.

* Si el valor es cualquier entero distinto a 0, significa que quien se ha conectado es un host. El procedimiento a seguir es el siguiente:
  + Creamos una regla de flujo en la tabla 0 que añada la VLAN que contenía el host y que a continuación haga una transición a la tabla 1
  + En la tabla 1 creamos otra regla de flujo cuando se envíe tráfico al host que se ha conetado. Solo en el caso de que coincida la VLAN del host que envió tráfico y la del host que se acaba de añadir enviamos por el puerto correspondiente y a continuación, desetiquetamos la VLAN (se recuerda que el tráfico que sale y llega a los hosts va sin ningún tipo de etiquetado VLAN). Se indica tambien que se haga una transición a la tabla 2 cuyo contenido se verá posteriormente
* Si el valor es 0 significa que quien se ha conectado es un router. El procedimiento en este caso varía siendo el siguiente:
  + Se crea una regla de flujo en la tabla 0 para cuando el router envía tráfico al open Vswitch cuyo tratamiento es simplemente hacer una transición a la tabla 1 ya que este tráfico ya viene etiquetado con la VLAN.
  + En la tabla 1 nos centramos en el tráfico cuyo destino es el router, en este caso se le indica al open vSwitch que lo envíe por el puerto en el que está conectado el router sin desetiquetarlo ya que el router a diferencia de un host si necesita saber la VLAN para poder enviar el tráfico nuevamente.
  + Finalmente añadimos las reglas a *macRuleMap*  para en el momento en el que el host se desconecta eliminarlas

Otro caso que hay que contemplar es para el tráfico broadcast. Hay que tener en cuenta que en un caso que no contemple VLANs el tráfico broadcast se difunden por todos los puertos. En este caso hay que hacer que se mande solo por aquellos que pertenecen a la VLAN y por los routers que tengamos.

Hay que recordar que en la tabla 0 ya se ha realizado la regla que asigna la VLAN, por tanto lo que hacemos es obtener del HashMap *macVlanMap* aquellas MAC cuya VLAN es la 0 haciendo uso del método *getKeys* e indicar modificando el tratamiento de la regla que el tráfico se envíe por esos puertos etiquetado.

A continuación, desetiquetamos el paquete y utilizando nuevamente el método *getKeys* obtenemos los hosts que coinciden con la VLAN y enviarlo por los puertos correspondientes.

Finalmente, instalamos la regla en la tabla 1 con el tratamiento previo.

FALTA EXPLICAR LAS 2 REGLAS QUE SELECCIONAN EL TRAFICO ARP, PERO TENGO QUE DARLE UNA VUELTA QUE AUN NO LAS ENTIENDO

4.3.2.1.2 Host eliminado de la red

Cuando un host es eliminado de la red lo primero que hacemos es eliminar todas las reglas de flujo que contengan esa MAC. Esto hace que se borre tambien aquella referida al tráfico broadcast, por tanto es necesario volverla a crear esta vez sin incluir el host que acabamos de eliminar. El procedimiento para realizar esta regla es idéntico al explicado en la sección anterior.

4.3.2.2.2 Ficheros auxiliares

Para que la interactividad con la aplicación sea más sencilla se han creado 3 comandos ejecutables mediante la CLI de ONOS para evitar tener que modificar el código de la aplicación cada vez que se quiera cambiar de VLAN algún host o cada vez que se quiera añadir o eliminar una correspondencia.

En primer lugar, en el fichero ShowVlanCommand se crea un el comando *show-Vlan-Mac* cuyo objetivo es mostrar las correspondencias MAC-VLAN presentes en la aplicación.

Para crear un comando se hace de la siguiente forma:

@Command(scope = "onos", name = "show-Vlan-Mac",

description = "shows the Vlan matches to an existing Mac")

Y, a continuación se llama al método doExecute(). En este método creamos un servicio de la clase principal de la aplicación (en este caso el fichero VlanByMac.java) que nos permitirá acceder a los métodos. Llamando al método showVlanMac definido en el fichero principal y cuyo código es simplemente imprimir por pantalla el HashMap macVlanMap ya tenemos realizado el comando.

El siguiente comando, presente en el fichero AddVlanCommand.java nos permite añadir nuevas correspondencias MAC-VLAN. Para ello se debe ejecutar el comando add-Mac-Vlan y añadir 2 argumentos. El primero es la MAC del host y el segundo la VLAN.

Un ejemplo de uso podría ser el siguiente:

add-Mac-Vlan 00:00:00:00:00:03 2

En este caso se le añade al host cuya MAC acaba en :03 la VLAN 2.

Cabe destacar que en caso de que se quiera modificar una correspondencia ya existente no es necesario eliminarla y volverla a crear, basta con utilizar este comando y ya se actualiza sin necesidad de haberla borrado previamente.

Finalmente, con el comando remove-Mac-Vlan eliminamos permanentemente una correspondencia. La sintaxis es idéntica al caso anterior teniendo que insertar en primer lugar la MAC del host y continuación la VLAN que tuviera ese host.

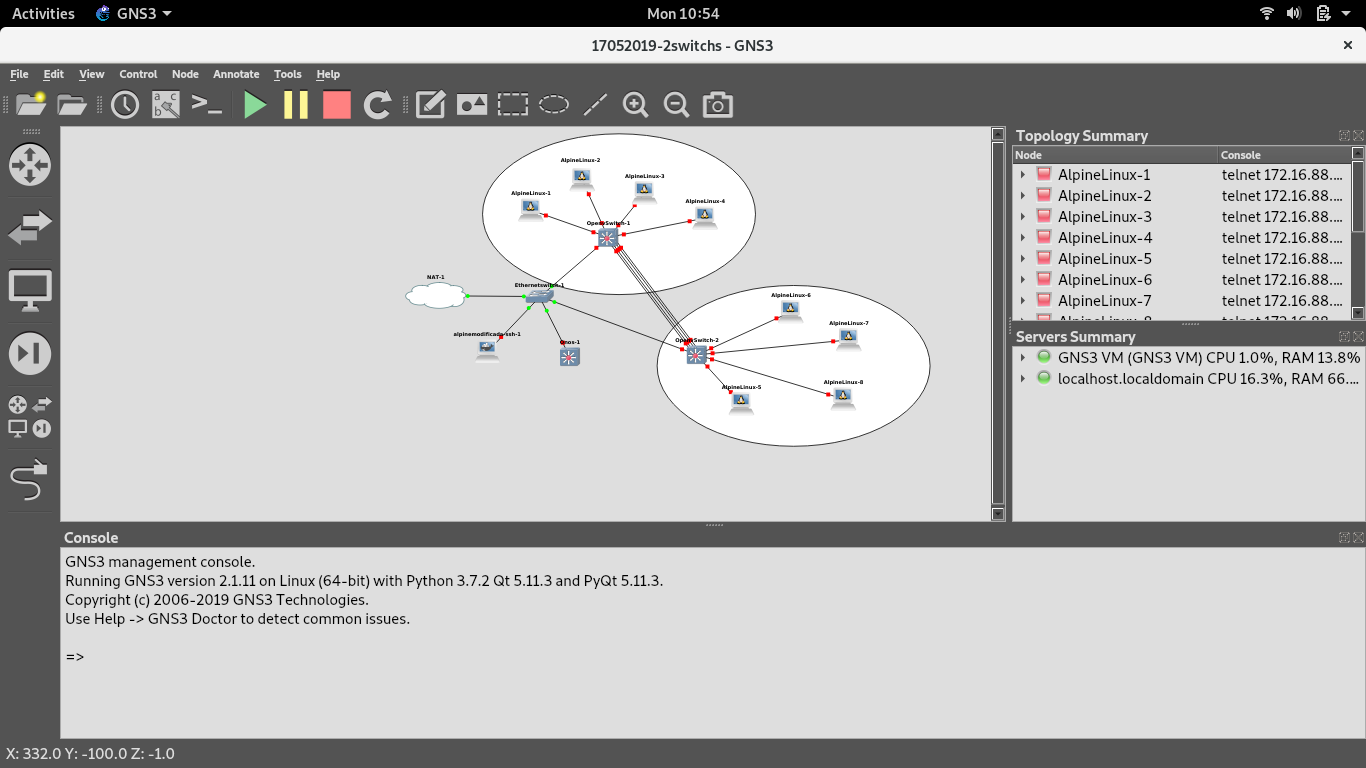
Un ejemplo de uso es:

remove-Mac-Vlan 00:00:00:00:00:03 2

En este caso hemos borrado la correspondencia creada en el ejemplo anterior.

4.4 Aplicación fwdBalanceo

Esta aplicación es diferente, puesto que es una modificación de la fwd que contiene el controlador ONOS por defecto. El objetivo es realizar un balanceo de carga cuando tenemos varios enlaces que conectan 2 open V switch. Para ello, en primer lugar modificamos la red que tenemos en el GNS3 sustituyendola por la mostrada en la Ilustración



Como vemos, tenemos 2 open V switch conectados por 4 enlaces que será el punto donde realizaremos el balanceo de carga entre las 2 subredes que conectan 4 hosts cada una

Como se ha comentado, lo que se va a hacer es modificar la aplicación fwd. Esta aplicación la podemos encontrar dentro de la carpeta Home/onos/app que copiaremos y llevaremos a nuestro directorio de trabajo. Seguidamente modificamos el pom.xml para cambiar el nombre de la aplicación dejandolo de la siguiente forma:

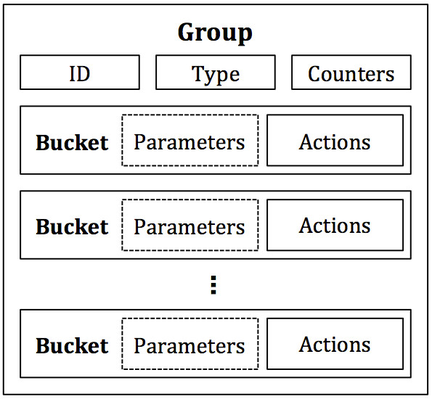


REMARCAR LOS CAMBIOS MODIFICANDO LA CAPTURA

Finalmente, modificamos el nombre de las carpetas que tienen el nombre de la aplicación sustituyendolas por fwdBalanceo (lo que modificará el paquete en el que está contenido), y una vez hecho esto ya podemos empezar a realizar nuestra aplicación, en concreto se van a modificar el *activate* y el método *installRule* del fichero java *ReactiveForwarding*

4.4.1 Explicación teórica

En este apartado se va a realizar la explicación acerca de como realizar el balanceo de carga. Para ello es necesario que los puertos que conecten ambos open Vswitch (puertos troncales) estén asociados un grupo. Un grupo es una abstracción que permite realizar operaciones más complejas (como lo es esta) que no se pueden realizar mediante las reglas de flujo que se han visto hasta el momento. Tampoco se pueden enviar a otras tablas. Cada grupo contiene una lista separada de acciones (*bucket list*). En nuestro caso el bucket serán los puertos por los que se enviará tráfico en cada momento en función del tráfico existente en la red.



Tal y como se vem un grupo está formado por un tipo. En concreto, hay 4 tipos diferentes que puede tener que son:

* ALL group: Si seleccionamos esta opción cuando un paquete llegue como entrada al grupo se duplicará para que se opere sobre él independientemente. De esta forma podemos operar en copias separadas del paquete original. Estas operaciones estarán definidas en las acciones en cada grupo. En este caso, el bucket solo está compuesto de acciones y no hay parámetros especiales.
* SELECT group: Esta opción está diseñada para equilibrar la carga. En este caso el parámetro consiste al peso que tendrá cada *bucket.* El algorimo de selección depende de la implementación del switch. En el apartado APARTADO BANCO DE PRUEBAS se indica como se realiza la selección de los buckets del grupo para el switch que manejamos, ya que esta es la sentencia que vamos a usar para nuestra aplicación
* INDIRECT group: Este caso es un poco especial, ya que solo contiene un único *bucket ,* por lo tanto siendo estrictos es díficil verlo como un grupo. Su fin es encapsular un conjunto de acciones utilizadas en muchos flujos. Por ejemplo, si tenemos diferentes tipos de flujos (por ejemplo que sean tipos diferentes de tráfico) pero las acciones que deben llevar son las mismas, se pueden enviar al grupo en vez de indicar la acción para cada flujo independiente. De esta forma creamos menos reglas de flujo y se ahorra en memoria.
* FAST-FAILOVER group: Este tipo de grupo está diseñado para detectar y superar fallos en los puertos. Cada *bucket* tiene un puerto de vigilancia que controla el “estado de vida” del puerto indicado. Si se considera que la vida está baja no se utilizará ese *bucket,* mientras que si su nivel de vida está alto se usa ese *bucket.* Como detalle indicar que solo se puede usar un *bucket* y que este no cambia a menos que se detecte que la vida del mismo desciende.

4.4.2 Método activate

En este método en primer lugar definimos como parámetros configurables los puertos que van a ser troncales. Se ponen como configurables para que sea más sencillo en caso de que se modifique la topología de la red modificarlos, y así mantener el código. En nuestro caso, los puertos seleccionados para ser troncales son los 10, 11, 12 y 13 de la topología del open V switch. A la hora de programar hay que sumar 1 a estos valores ya que no contabiliza el 0 como puerto y empieza en el 1 directamente.

Por tanto, nada más activar la aplicación creamos el primer grupo. Como el proceso puede parecer un poco complejo se va a explicar detalladamente.:

Un grupo se crea a partir de un constructor que contiene un GroupId (arbitrariamente hemos elegido el 137, y un *DefaultGroupDescription*:

DefaultGroup dgd = **new** DefaultGroup(**new** GroupId(137),degd);

A su vez, un *DefaultGroupDescription* tiene varios constructores. Usaremos el utilizado para las aplicaciones de la API del norte. Está formado por el dispositivo, el tipo de grupo que va a ser, los *GroupBuckets*, una clave para el grupo, el groupId definido previamente y la aplicación en la que se va a instanciar.

DefaultGroupDescription degd = **new** DefaultGroupDescription(d.id(), GroupDescription.Type.***SELECT***,gbs,k,137,appId);

Nuevamente, a su vez, los *GroupBuckets* está formado por un ArrayList que contiene todos los buckets individuales que tiene el grupo

gbs = **new** GroupBuckets(listGroup);

List<GroupBucket> listGroup = **new** ArrayList<GroupBucket>();

Finalmente, a este ArrayList de GroupBucket creado le iremos añadiendo cada bucket individual:

GroupBucket gb = DefaultGroupBucket.*createSelectGroupBucket*(sendPort);

Cada GroupBucket tendrá como argumento la acción a llevar por él, en nuestro caso enviarlo por el primer puerto troncal que tenga la red:

TrafficTreatment sendPort = DefaultTrafficTreatment.*builder*()

.setOutput(PortNumber.*portNumber*(puerto))).build();

Por tanto, lo que hemos hecho es crear un grupo que por el momento tiene un único bucket, que consiste en enviar por el puerto 10 cuando se utilice.

El siguiente paso es determinar 3 umbrales de tráfico acordes (al igual que los puertos son parámetros configurables). Cuando el tráfico agregado en toda la red supere un determinado umbral añadiremos un nuevo bucket que contenga el siguiente puerto.

Para poder obtener el tráfico agregado creamos una tarea y en ella obtenemos las delta estadísticas de cada puerto que iremos almacenando de acuerdo a la siguiente ecuación:

bitsPorSegundo=bitsPorSegundo+(traffic.bytesSent()\*8)/temporizadorTarea;

PONER MODO ECUACION EN WORD

Para añadir *buckets* simplemente creamos un nuevo tratamiento con el puerto correspondiente que esté almacenado en un nuevo *GroupBucket* ylo añadiremos al ArrayList ya creado.

Una vez tenemos actualizado en cada iteración del temporizador predefinido los *buckets* por los que se mandará el tráfico entre los 2 open Vswitch hay que crear las reglas de flujo que indiquen que el tráfico se mande por el grupo en vez de por los puertos propiamente dicho. Esto se tiene que realizar en el método *installRule.*

4.4.3 Método installRule

Dentro de este método el objetivo es crear una regla que indique que cuando el tráfico tenga como destino alguno de los puertos troncales, lo cual haremos dentro de una sentencia *if.*

El tratamiento cuando entremos en esta sentencia condicional será indicar al tráfico que se envie por el grupo de la siguiente forma:

TrafficTreatment treatment= DefaultTrafficTreatment.*builder*()

.group(**new** GroupId(137))

.build();

Esto, como se ha comentado, lo que hace es enviar por al grupo previamente creado y que sea este el que según el tipo y los *buckets* que contenga.

Por otro lado, en el caso de que el tráfico tenga como destino cualquier otro puerto el tratamiento a seguir será enviarlo por el puerto de salida que haya aprendido simplemente.

4.5 Aplicación FakeDHCP

4.5.1 Explicación teórica

En primer lugar se va a explicar el funcionamiento del protocolo DHCP (*Dynamic Host Configuration Protocol)*  es un protocolo de la capa de aplicación en el cual un servidor DHCP asigna de forma dinámica direcciones IP a los dispositivos de una red para poder comunicarse. Este protocolo permite 3 formas de asignación de direcciones IP:

* Asignación manual: Se asigna una IP a una máquina de forma estática, es decir, cada vez que se conecte recibirá la misma dirección
* Asignación automática: Se asigna una IP a una máquina la primera vez que hace la solicitud al servidor y se mantiene hasta que se desconecta
* Asignación dinámica: En este caso el administrador determina un rango de direcciones IP y cada máquina solicita su dirección IP al servidor cuando la tarjeta de interfaz de red se inicializa y durante un tiempo limitado. Cuando este tiempo finaliza la IP es eliminada y la máquina no puede funcionar hasta que pida otra.

El funcionamiento del protocolo se basa en el envío de 4 tramas diferentes intercambiadas entre cliente y servidor. Cabe destacar que el servidor utiliza el puerto 67 UDP y el cliente utiliza el puerto 68 UDP

* DHCP Discovery: Es una solicitud DHCP realizada por un cliente con la dirección destino MAC broadcast para que el servidor DHCP le asigne una dirección IP y el resto de parámetros como servidor DNS
* DHCP Offer: Es el paquete de respuesta del servidor DHCP ante la petición previa en el que le ofrece una dirección IP. Contiene una oferta de configuración con todos los detalles que el servidot tiene almacenados para el cliente. Este puede aceptarla o ignorarla
* DHCP Request: El cliente confirma la oferta recibida con sus detalles
* DCHPACK: El servidor confirma la asignación y una vez recibida el cliente empieza a usar la red.

PONER CAPTURAS WIRESHARK DEL INTERCAMBIO DE TRAMAS

Una vez explicado el funcionamiento del protocolo se va a detallar el de la aplicación en concreto:

Esta aplicación está basada en aumentar la seguridad de la red con el objetivo de evitar conectarnos a servidores DHCP no fiables.

Este fenómeno conocido como *rogue DHCP* tiene el siguiente funcionamiento:

En primer lugar el cliente se conecta a la red, en este momento tanto el servidor DHCP legítimo como el falso que está conectado a la red les ofrecen direcciones IP entre otros. En el momento en el que algún cliente acepta la dirección IP del DHCP falso puede detectar todo el tráfico que envia violando todo tipo de privacidad y pudiendo acceder a datos comprometidos o realizar ataques del tipo *man in the middle*.

4.5.2 Procesador de paquetes

Para el funcionamiento de la aplicación es necesario crear un procesador de paquetes muy similar al utilizado en la aplicación *severalpings*.

En primer lugar se interceptaran todos los paquetes que sean del tipo DHCP. Como es un protocolo de la capa de aplicación ONOS no llega a sus cabeceras ya que se queda en el nivel 3, por tanto, lo que haremos será interceptar todo el tráfico que tenga como origen el puerto UDP 67 y destino el puerto UDP 68

TrafficSelector selector = DefaultTrafficSelector.*builder*()

.matchEthType(Ethernet.***TYPE\_IPV4***)

.matchIPProtocol(IPv4.***PROTOCOL\_UDP***)

.matchUdpDst(TpPort.*tpPort*(UDP.***DHCP\_CLIENT\_PORT***))

.matchUdpSrc(TpPort.*tpPort*(UDP.***DHCP\_SERVER\_PORT***))

.build();

Cuando el paquete está en el controlador añadimos un procesador de paquetes que será instanciado:

**private** **final** PacketProcessor packetProcessor = **new** DHCPProcessor();

Dentro del método DHCPProcesor lo primero es comprobar que de todo el tráfico que llega al controlador estamos seleccionando el DHCP. Para ello habrá que obtener el paquete que acaca de llegar y obtener la carga útil e igualarlo al mensaje que queremos comprobar, DHCPOffer

((DHCP)(((UDP)((IPv4) eth.getPayload()).getPayload())).getPayload()).getPacketType() == DHCP.MsgType.***DHCPOFFER***;

Cuando se ha comprobado que efecivamente, el paquete que se está procesando es un DHCPOffer comprobamos el puerto por el que venía ese paquete. Si e el mismo que aquel que tenemos prefijado (el puerto del router es un parámetro configurable) permitimos el envío del paquete, en caso de que ese DHCPOffer venga por cualquier otro puerto se bloquea, el host no recibe ese paquete, por tanto no acepta la oferta y se evita que cualquiera que intente realizar un ataque a nuestra red.

Referencias

<https://es.wikipedia.org/wiki/Redes_definidas_por_software>

[https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html#~stickynav=1](https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html" \l "~stickynav=1)

<https://openzen.wordpress.com/2015/02/12/historia-del-sdn/>

<https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>

<https://blogthinkbig.com/sdn-software-defined-networking-cambiando-de-paradigma-en-la-red>

<https://computingforgeeks.com/how-to-install-gns3-on-fedora-29-fedora-28/>

[https://riunet.upv.es/bitstream/handle/10251/16310/Art%C3%ADculo%20docente%20configuraci%C3%B3n%20b%C3%A1sica%20VLANs.pdf](https://riunet.upv.es/bitstream/handle/10251/16310/Artículo docente configuración básica VLANs.pdf)

<https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups>