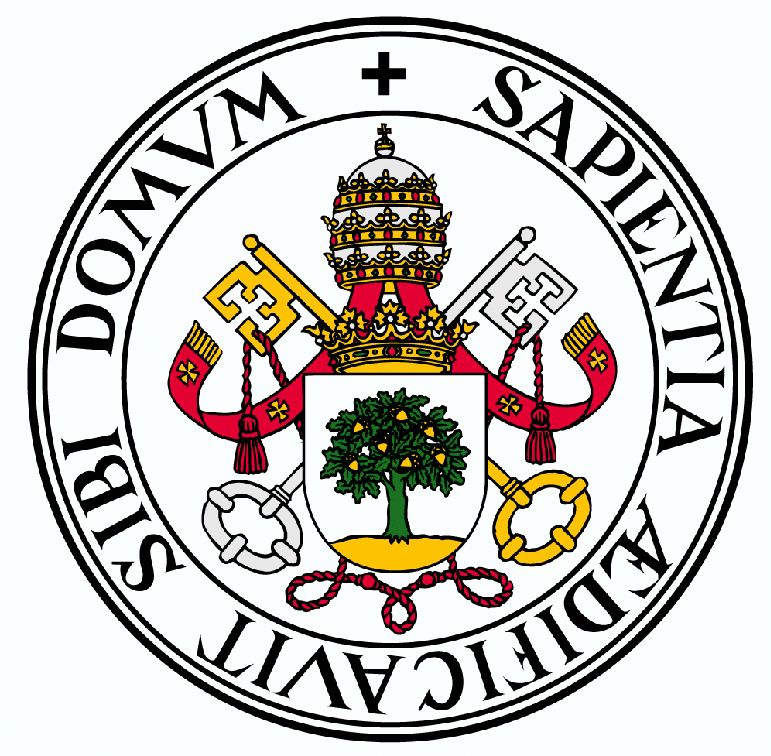
****

**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**

**TITULO**: 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,

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

Telecomunicación, Universidad de Valladolid

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

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

Telecomunicación, Universidad de Valladolid

**DEPARTAMENTO**: Departamento de Teoría de la Señal, 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 cada elemento individual. El uso de redes definidas por software (SDN) permite disponer de in 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 configurar una red con el simulador GNS3 e incorporarle a un controlador diferentes aplicaciones que gestione de diferentes formas el comportamiento de dicha red, consiguiendo de esta forma una red auto-adaptativa.

**MATERIAL**

Entre los materiales que se han utilizado se encuentran: Un ordenador portátil que contiene una distribución Linux instalada , el entorno gráfico GNS3, utilizado para la configuración de la red, el *software* Eclipse, utilizado para programar las diferentes aplicaciones que se le mandarán al controlador de la red.

**METODOLOGIA**

**RESULTADOS**

**CONCLUSIONES**

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

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

1.2 Organización de la memoria

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 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 tambien 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 aparación del protocolo Openflow que es considerado uno de los primeros estándares de las redes SDN y surgió a raiz de separar los planos.

Openflow es una tecnologia 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:

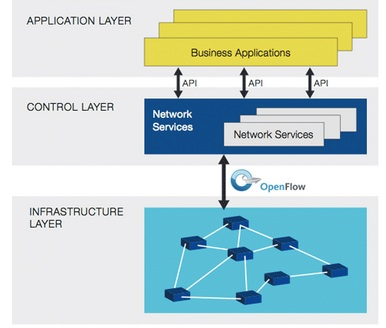
* Flexibilidad en el uso de la red, asi como en la forma de operar sobre ella.
* El ser de código abierto, por tanto puede ser modificado de forma muy sencilla.
* Reducir el gasto de operación.
* Mejorar la integración de la red. Permite que la red sea controlada con un solo punto de vista.

1.2 Arquitectura de las redes SDN

Todas las redes SDN están caracterizadas por tener estos elementos:

* Aplicación SDN: Son programas que comunican el comportamiento de la red al controlador SDN
* Controlador SDN: Es el cerebro de la red y encargado de traducir las peticiones de la aplicación y enviarlas a los switches.
* Interfaz hacia el norte: Es la API que comunica la aplicación SDN y los controladores.
* Capa de datos: Compuesta por los diversos elementos de red como los switches o sistemas finales.
* Interfaz hacia el sur: Es el protocolo que comunica el controlodor 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 (openVswitch) y el controlador.

Supongamos que el Host 1 envía una trama al Host 2. La trama en primer lugar llega al openVswitch. 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 como se debe conmutar dicha trama.

Capitulo 2: Primeros pasos

En este capítulo se van a detallar los primeros pasos previos a poder empezar a simular redes SDN.

2.1 Instalación del SO

Para poder realizar el proyecto correctamente es necesario la instalación de un sistema operativo Linux funcional. En este caso se ha elegido la distribución Linux Fedora 29 basada en GNOME 3, aunque cualquier distribución es válida.

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

Por su parte el sistema operativo Debian tiene problemas con los repositorios *non-free*

2.2 Instalación de GNS3

Una vez tenemos una distribución de Linux instalada, el siguiente paso es instalar el simulador gráfico GNS3. Este simulador es gratis y se puede descargar de forma sencilla desde la página web.

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 virtualizacion como VMWare y descargar la máquina virtual para GNS3 disponible en la página web.

2.3 Instalación de ONOS

NO RECUERDO, ONOS SE INSTALABA SOBRE GNS3? ERA INSTALANDO LA IMAGEN DIRECTAMENTE NO?

Y, TUVIMOS QUE INSTALARLO EN EL PC TAMBIEN NO?

2.4 Instalación de Eclipse

Una vez tenemos GNS3 instalado el siguiente paso es instalar Eclipse para poder programar las diferentes aplicaciones que querramos mandar al controlador y que veremos en capitulos posteriores.

2.5 Instalación de maven

BUSCAR, NO RECUERDO SI IBA CON ONOS O A PARTE

Capitulo 3: Creando la primera red con GNS3

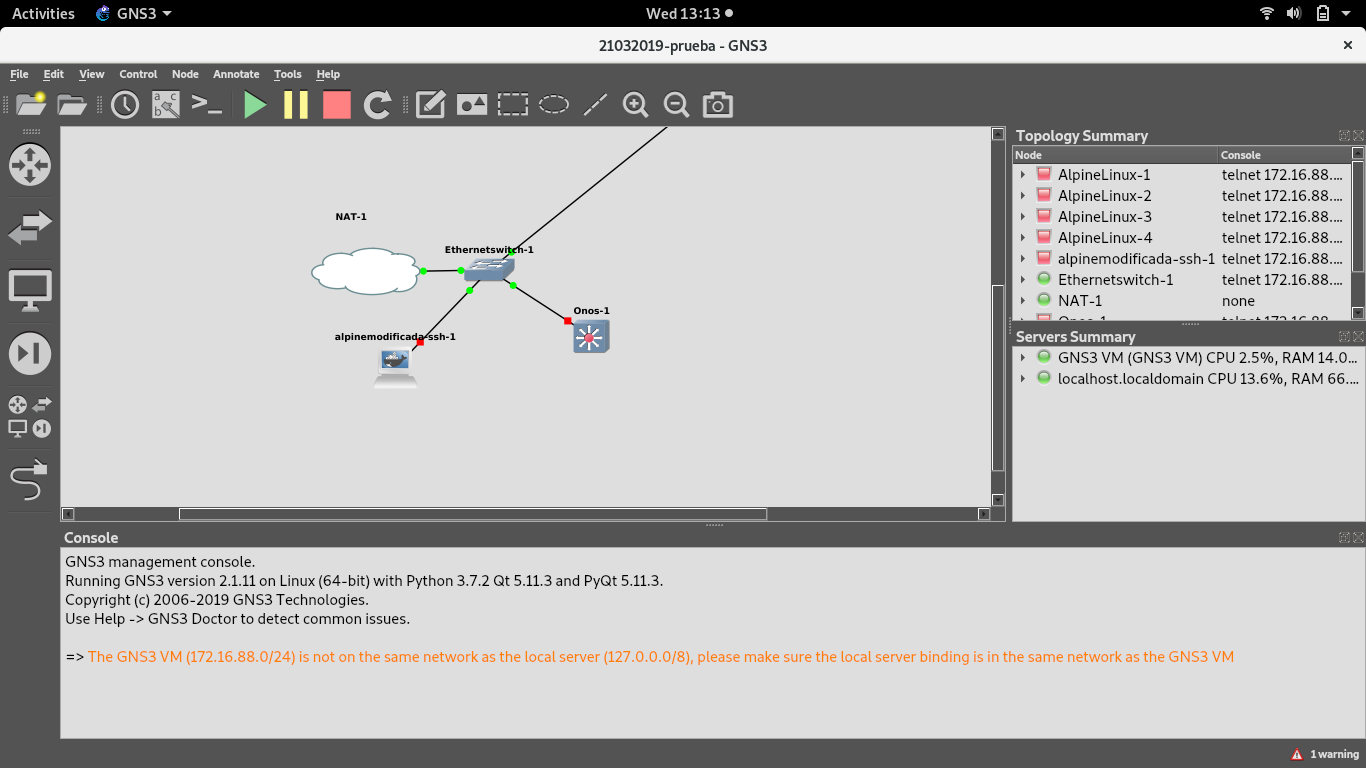
En este capitulo se va a detallar la red que se va a simular y sobre la cual instalaremos las diferentes aplicaciones que iran componiendo el proyecto.

La red 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 openVswitch

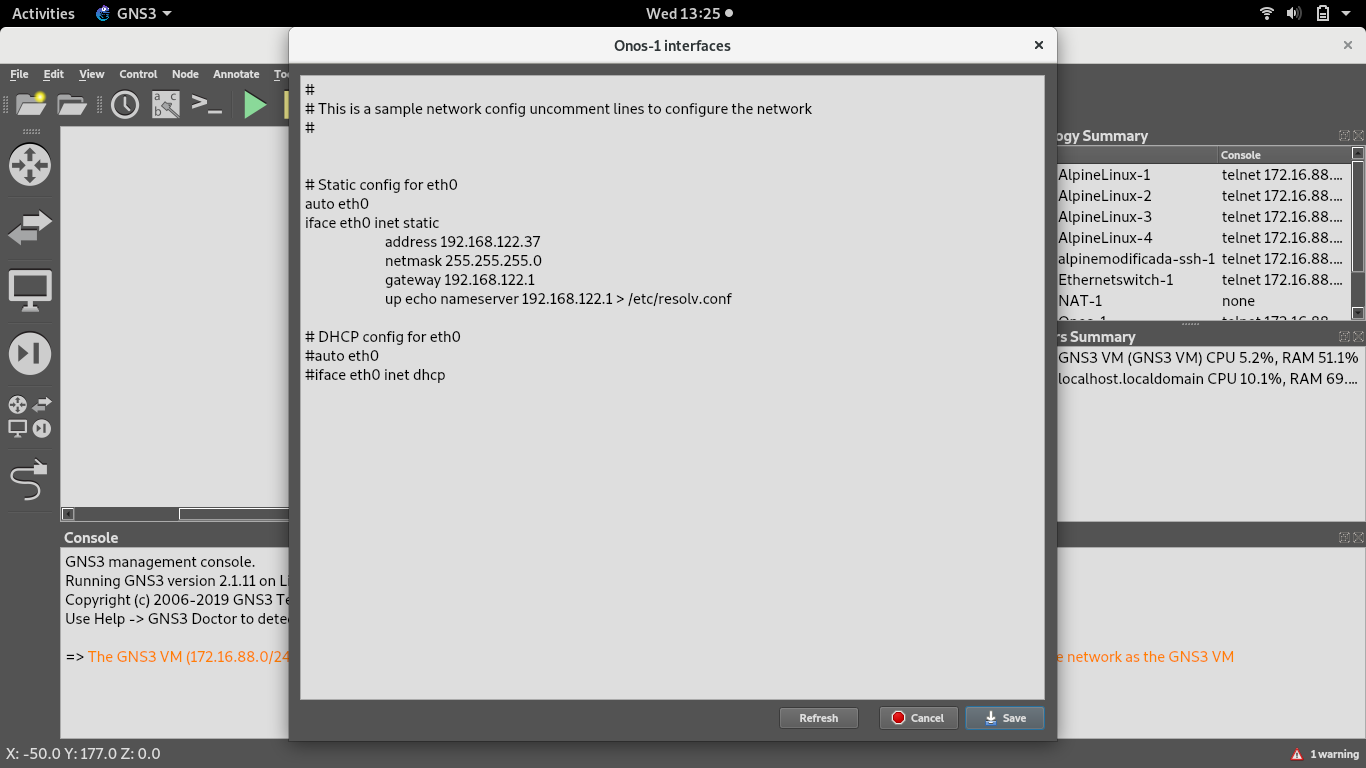
3.1 Conexión con el controlador y con el exterior

La parte de la red que concierne a este apartado se puede ver en la Ilustracion 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 NAT (Network Address Translation) que nos permita acceder a nuestra red privada a través de una IP pública.

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

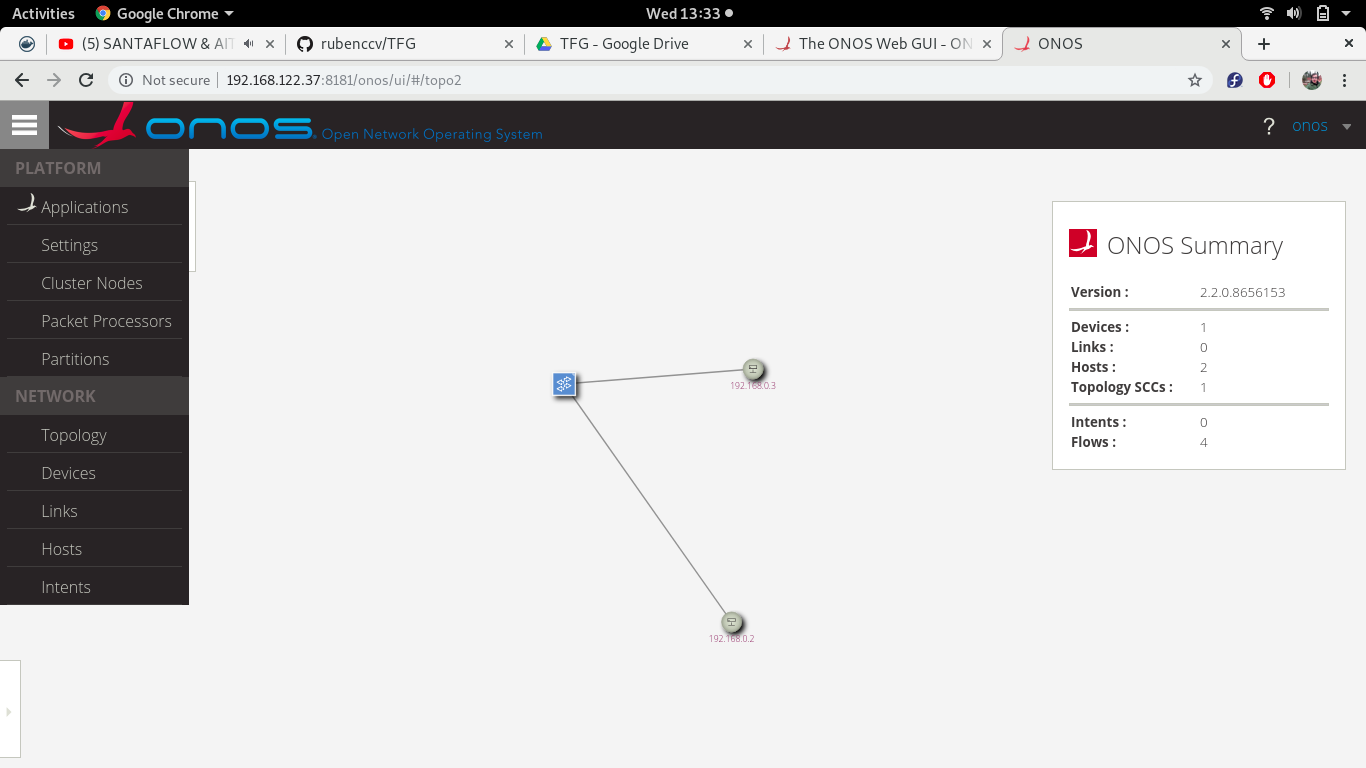


Le hemos elegido arbitrariamente la IP 192:168:122:37

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 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, asi 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 ssh (el Alpine por defecto no lo trae instalado), asi que lo que haremos será modificar la imagen almacenada en Docker.

Para empezar hacemos un pull de la imagen alpine almacenada

docker pull alpine

A continuación, ejecutamos la imagen en nuestro shell

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

Le instalamos las aplicaciones necesarias para nuestro proyecto

apk add openssh wget curl

Y la subimos de nuevo a docker con el nombre deseado

docker commit alpinemodificada2 alpinemodificada

docker rm alpinemodificada2

Ahora que ya tenemos un host con 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*

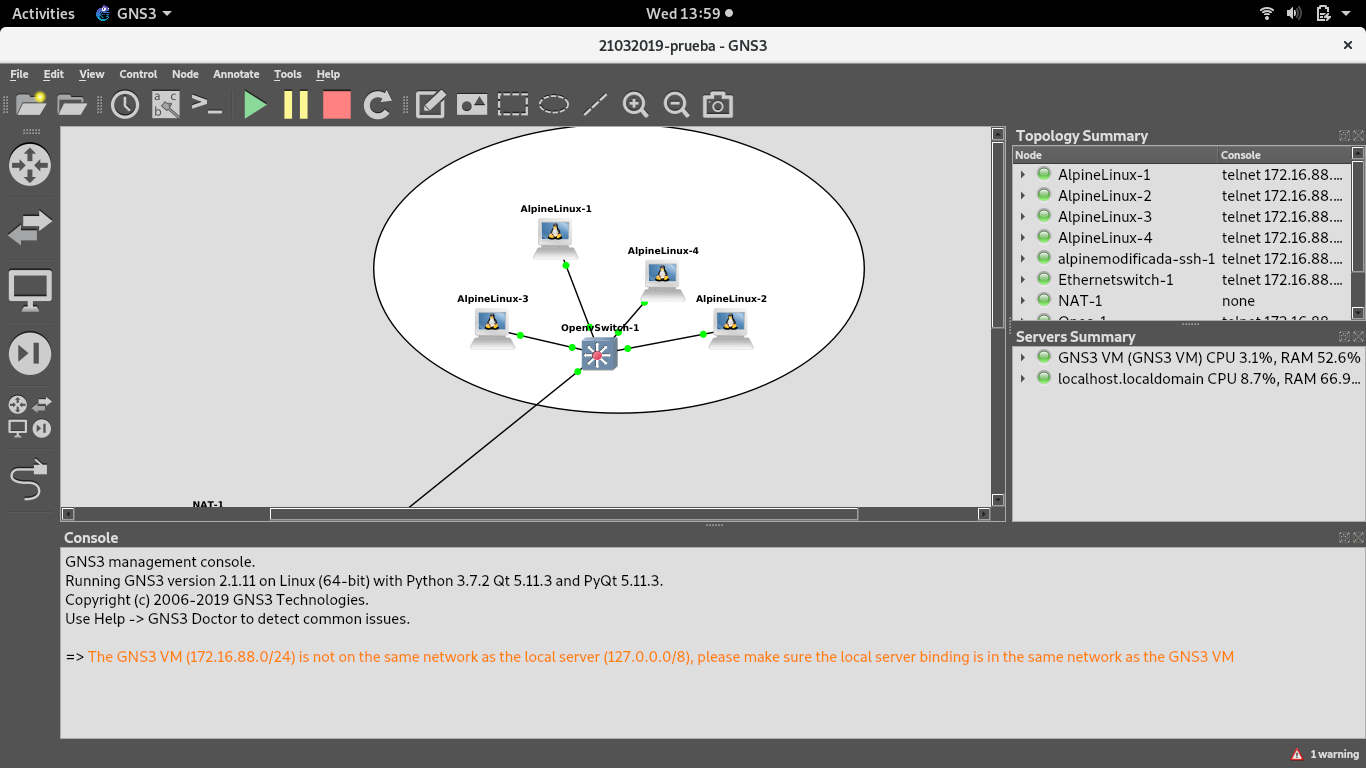
Algunos comandos de utilidad que son necesarios para comprobar el funcionamiento son:

* devices: Devuelve los dispositivos que son controlados por el controlador. En las redes que simularemos son los openVswitch.
* hosts: Devuelve los hosts que ha reconocido (aquellos que han enviado algún paquete y están conectados a los dispositivos)
* flows: Devuelve los flujos que han instalado sobre los dispositivos las diferentes aplicaciones y que gestionan el comportamiento de la red.
* log:set debug: Accedemos al modo debug 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 openVswitch

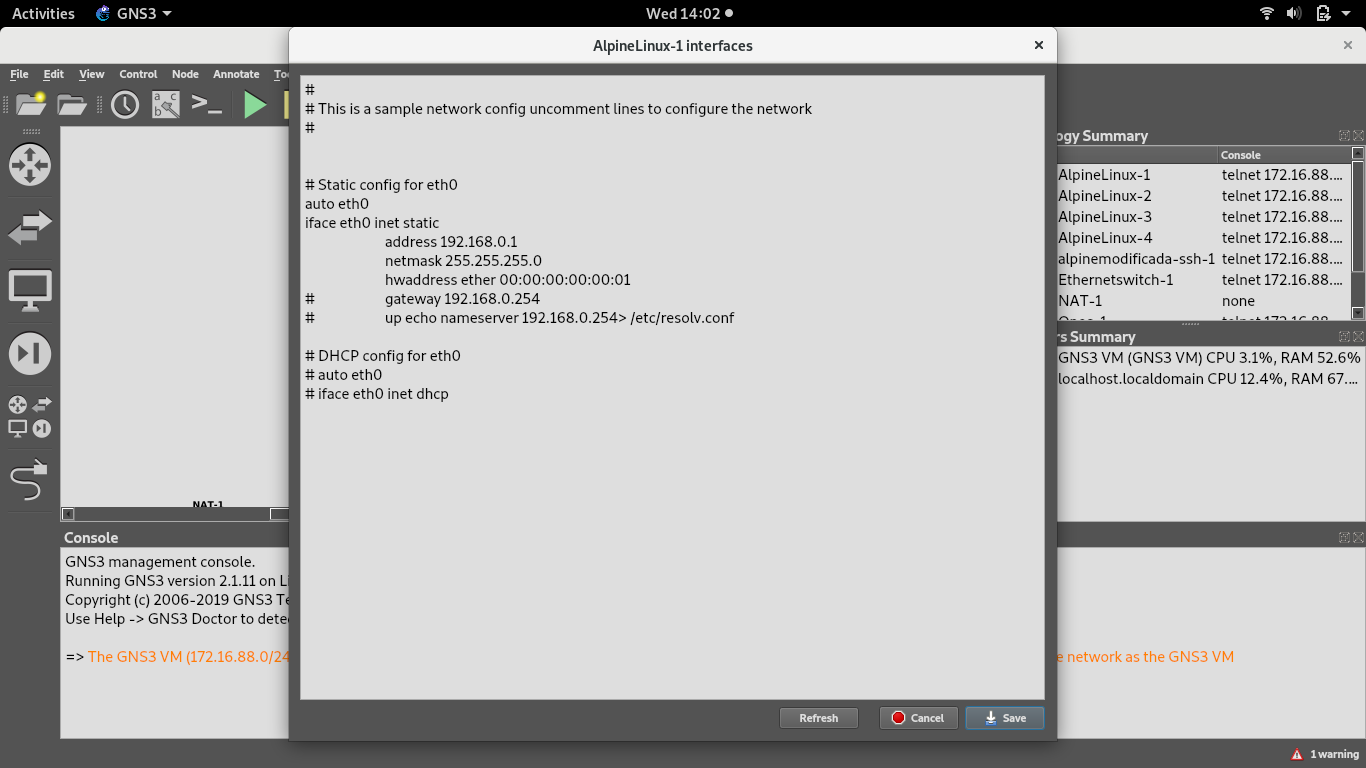
Esta parte de la red es la que nos permitirá probar el funcionamiento de nuestras aplicaciones y se irá modificando en función de los requerimientos de dichas aplicaciones.

Para las primeras tendrá la siguiente topología:



Como podemos ver, tenemos 4 hosts conectados a un openVswitch sobre el que van instaladas las aplicaciones. Para poder trabajar más comodamente 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 openVswitch es necesario indicarle quien es el controlador. Como la IP de este es estática este proceso solo tendremos que realizarle una vez, en caso de que se eligiese por DHCP habría que realizarlo cada vez que arrancamos el programa. Para poder indicarselo abriremos una consola desde el openVswitch y ejecutaremos los siguientes comandos:

ovs-vsctl set-manager ptcp:IP-ONOS:6640

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

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

Capitulo 4: Aplicaciones desarrolladas

En este capitulo 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

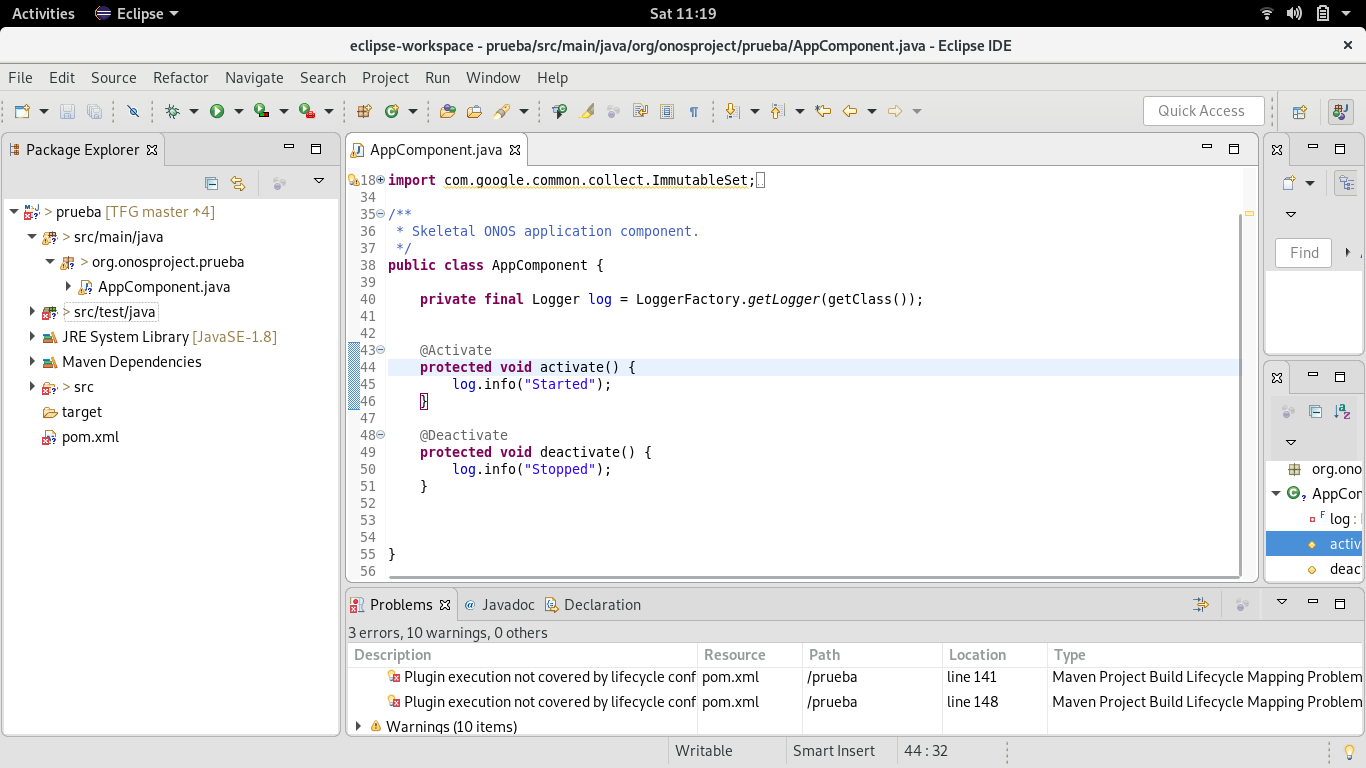
donde nombreApp será el nombre que decidamos dar a nuestra aplicación y la versión es la fase en la que estemos de la aplicación. Si es una aplicación que estamos empezando se pone 1.0-SNAPSHOT.

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 situan los código fuente que podremos modificar. Aparecere 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 enviarsela 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 enviandola 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 adapatación de la aplicación *oneping (PONER REFERENCIA)* cuyo objetivo principal es limitar el numero 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 limite 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 openVswitch de nuestra red para enviarlos al controlador de la manera que se indica a continuación:

**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 openVswitch 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

4.2 Analizando las estadísticas de tráfico

(...)

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>