



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: Dpto. de Teoría de la Señal y Comunicaciones e Ing. Telemática
E.T.S.I. Telecomunicación, Universidad de Valladolid

COTUTOR: Jaime Gómez Gil
Dpto. de Teoría de la Señal y Comunicaciones e Ing. Telemática
E.T.S.I. Telecomunicación, Universidad de Valladolid

DEPARTAMENTO: Dpto. de Teoría de la Señal y Comunicaciones e Ing. 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:

A mis padres

Resumen

Las redes tradicionales requieren de cambios de configuración para cada elemento individual, lo que supone una sobrecarga sustancial para el administrador de red. El empleo de redes definidas por software (SDN) permite disponer de uno o varios controladores que gestionan de manera simultánea todos los elementos de la red. Este enfoque permite monitorizar y 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 adapte de manera automática 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.

El objetivo principal de este trabajo es desarrollar aplicaciones SDN para el controlador ONOS, que permitan gestionar diferentes escenarios y 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.

El principal material empleado en este TFG ha sido 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 serán elementos de red empleados en todas las aplicaciones desarrolladas en este TFG.

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, y (iv) evaluación. Para cada una de las aplicaciones desarrolladas, así como para las pruebas piloto realizadas, este proceso se repitió en bucle hasta alcanzar unos resultados que cumpliesen con los objetivos inicialmente marcados.

En el presente TFG se han desarrollado 8 aplicaciones relacionadas con las redes definidas por *software*: (i) *severalping*: 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 el número de *hosts* conectados a la red gestionada por ONOS, así como información de las direcciones MAC e IP de los *hosts* junto con el dispositivo y el puerto al que está conectado cada uno de ellos; (iv) *detectHostBan*: permite deshabilitar el envío de tráfico por parte de cualquier *host* de la red cuando supera un cierto umbral; (v) *VLAN*: permite asignar a los *hosts* de la red una VLAN para poder crear subredes virtuales con independencia del puerto al que se conectan; (vi) *fwdBalanceo*: permite realizar balanceo de carga cuando hay más de un enlace que comunica 2 dispositivos; (vii) *FakeDHCP*: permite detectar servidores DHCP falsos conectados a la red cuyo objetivo es interceptar el tráfico de los *hosts* con fines maliciosos; y (viii) *diffserv*: permite crear colas asociadas a diferentes flujos de tráfico, asignándoles diferentes tipos de calidad de servicio.

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 que se pueden extraer del presente TFG son: (i) El tiempo entre sondeos es un factor crítico a la hora de limitar el tráfico entre *hosts*, puesto que, si este tráfico es enviado a gran velocidad, se puede superar ampliamente el umbral establecido. (ii) La velocidad a la hora de administrar una red se ha visto incrementada mediante la utilización de las diversas aplicaciones desarrolladas. (iii) El puerto al cual se conecta el *host* a la hora de asignarle una VLAN es indiferente, ya que la asociación se realiza mediante MAC. (iv) El hecho de limitar el número de *pings* de una aplicación, se puede comprobar, que en caso de ataque DoS reduce de forma significativa el ancho de banda circulante por la red. (v) Dado que existe una comunicación constante entre el Open vSwitch y el controlador, es importante no contar el puerto que los comunica a la hora de analizar las estadísticas de tráfico, ya que no sería realista al incorporar el tráfico de gestión.

Palabras clave: Redes definidas por *software* (SDN), aplicación, controlador, ONOS, red, simulación, GNS3.

Abstract

Traditional networks require configuration changes for each individual element, what implies a heavy overload for network administrators. The use of software defined networks (SDN) allows to have a controller that simultaneously manages all the elements of the network. This approach allows the elements of the network adapting in a dynamic way, enabling the real-time optimization of the network to the varying needs.

This work intends to simulate, through GNS3 network simulator, different SDN schemes that automatically react to the current conditions in order to move forward in the deployment of self-adaptive networks.

The main objective of this work is to develop SDN applications, for ONOS controller, that allow various network topologies, created with GNS3 network simulator, to be managed in different scenarios. ONOS controller will be able to load different applications that manage the behavior of this network in different ways, achieving a self-adaptive network.

The main material used to conduct the underlying work is a laptop. This computer has a GNU/Linux operating system distribution installed (Fedora 29). In this operating system the following software is required: (i) GNS3 graphic network simulator, used for network configuration; (ii) Eclipse integrated development environment (IDE), used for coding and programming the different applications that will be sent to the network controller. In turn, within the GNS3 simulator, it is necessary to have installed both the image of Open vSwitch and ONOS controller, since these network elements will be used in all the applications developed in this work.

The methodology followed in this work to reach the proposed objectives has consisted in the iterative realization of the following phases: (i) documentation, (ii) design, (iii) programming, and (iv) evaluation. For each of the applications developed, as well as for the pilot tests carried out, this process was repeated in a loop until reaching results that met the initially marked objectives.

In the present work 8 applications related to software defined networks have been developed: (i) *severalpings*: it allows to limit the number of pings that can be exchanged by any 2 hosts in the network; (ii) *statsshow*: it allows to know the amount of data that are sending and receiving the hosts connected to the network; (iii) *detectHost*: it allows to know the number of hosts connected to the network managed by ONOS controller, as well as the device and the port to which each host is connected; (iv) *detectHostBan*: it allows banning the sending of traffic for any host in the network when it exceeds certain threshold; (v) *VLAN*: it allows a VLAN to be dynamically assigned to the hosts of the network in order to create virtual subnets; (vi) *fwdBalanceo*: it allows load balancing when there is more than one link that communicates 2 devices; (vii) *FakeDHCP*: it allows the detection of fake DHCP servers whose objective is to intercept the traffic of hosts for malicious purposes; and (viii) *difsserv*: it allows the creation of queues and assigning to them different qualities of service.

The main conclusion, that can be draw from this work, is that management (centralized or distributed) through a controller can solve network problems that are not easily resolvable without using a software defined networks approach.

The main secondary conclusions that can be extracted from the present work are: (i) The sampling period between statistics polls for each switch is a critical factor when limiting traffic between hosts, since, if this traffic is sent at high speed, it can exceed the limit threshold by far. (ii) The speed at which to manage a network has been increased by using several of the applications developed. (iii) The port to which the host connects, when assigning a VLAN, is irrelevant since the association is performed on a MAC address basis. (iv) The fact of limiting the number of *pings*, significantly reduces the traffic through the network in case of a Denial of Service (DoS) attack. (v) Since there is constant communication between Open vSwitch and the

controller, it is important to not include the port that communicates them where analyzing the traffic statistics, since it would not be realistic to take into account management traffic as if it was data traffic.

Keywords: Software Defined Networks (SDN), application, controller, ONOS, network, simulation, GNS3.

Agradecimientos

Este Trabajo de Fin de Grado no sería posible sin todas las personas que me rodean y apoyan día a día.

En primer lugar, quiero agradecer a los tutores de este proyecto, por su ayuda e implicación durante todos los meses de trabajo, sin los cuales habría sido imposible poder realizar el trabajo.

Seguidamente, agradecer a mis padres y hermanos, ya que sin ellos no habría podido llegar hasta este punto.

Y, finalmente, aunque no menos importante, a todos y cada uno de mis amigos, por conseguir sacarme una sonrisa todos los días.

Lista de acrónimos

A continuación, se listan una serie de siglas utilizadas a lo largo de la memoria:

| | |
|--------------|--|
| API | <i>Application Programming Interface</i> |
| BA | <i>Behavior Aggregate</i> |
| BDDP | <i>Broadcast Domain Discovery Protocol</i> |
| CBS | <i>Committed Burst Size</i> |
| CIR | <i>Committed Information Rate</i> |
| CLI | <i>Command-Line Interface</i> |
| DDR4 | <i>Double Data Rate 4</i> |
| DHCP | <i>Dynamic Host Configuration Protocol</i> |
| DoS | <i>Denial of Service</i> |
| DS | <i>Differentiated Services Field</i> |
| DSCP | <i>Differentiated Service Code Point</i> |
| EBS | <i>Excess Burst Size</i> |
| FIFO | <i>First In First Out</i> |
| GNOME | <i>GNU Network Object Model Environment</i> |
| GNS3 | <i>Graphical Network Simulator 3</i> |
| GNU | <i>GNU is Not Unix</i> |
| GUI | <i>Graphical User Interface</i> |
| HDD | <i>Hard Disk Drive</i> |
| ICMP | <i>Internet Control Message Protocol</i> |
| IDE | <i>Integrated Development Environment</i> |
| IETF | <i>Internet Engineering Task Force</i> |
| IGMP | <i>Internet Group of Management Protocol</i> |
| IP | <i>Internet Protocol</i> |
| LAN | <i>Local Area Network</i> |
| LLDP | <i>Link Layer Discovery Protocol</i> |
| MAC | <i>Media Access Control</i> |
| MF | <i>MultiField</i> |
| MitM | <i>Man in the Middle</i> |
| NAT | <i>Network Address Translation</i> |
| ONOS | <i>Open Network Operating System</i> |
| OvS | <i>Open vSwitch</i> |
| PC | <i>Personal Computer</i> |

| | |
|-------------|--|
| POO | <i>Programación Orientada a Objetos</i> |
| PPP | <i>Point-to-Point Protocol</i> |
| QoS | <i>Quality of Service</i> |
| RAM | <i>Random Access Memory</i> |
| RED | <i>Random Early Detection</i> |
| SDN | <i>Software Defined Network</i> |
| SLA | <i>Service Level Agreement</i> |
| SLS | <i>Service Level Specification</i> |
| SQP | <i>Strict Priority Queueing</i> |
| SSH | <i>Secure Shell</i> |
| TCP | <i>Transmission Control Protocol</i> |
| TOS | <i>Type of Service</i> |
| UDP | <i>User Datagram Protocol</i> |
| VID | <i>VLAN ID</i> |
| VLAN | <i>Virtual Local Area Network</i> |
| VoIP | <i>Voice over IP (Internet Protocol)</i> |
| WRED | <i>Weighted Random Early Detection</i> |
| WRR | <i>Weighted Round Robin</i> |

Índice

| | |
|---|----|
| <i>Resumen</i> | 4 |
| <i>Abstract</i> | 6 |
| <i>Agradecimientos</i> | 8 |
| <i>Lista de acrónimos</i> | 9 |
| <i>Índice</i> | 11 |
| <i>Índice de figuras y tablas</i> | 14 |
| <i>Capítulo 1. Introducción</i> | 17 |
| 1.1 Contexto | 17 |
| 1.2 Objetivos del proyecto | 17 |
| 1.3 Fases | 17 |
| 1.4 Medios empleados | 18 |
| 1.5 Organización de la memoria | 18 |
| <i>Capítulo 2: Estado del arte</i> | 20 |
| 2.1 Historia de las redes SDN | 20 |
| 2.2 Definición y arquitectura de las redes SDN | 20 |
| 2.3 Funcionamiento básico de una red SDN | 22 |
| 2.4 Protocolo OpenFlow | 23 |
| 2.4.1 Componentes de un <i>switch</i> que soporta OpenFlow. | 23 |
| 2.4.2 Puertos OpenFlow | 24 |
| 2.4.3 Tablas de OpenFlow | 25 |
| 2.4.3.1 Proceso de pipeline | 25 |
| 2.4.3.2 Tablas de flujo y entradas | 26 |
| 2.4.3.3 Matching | 26 |
| 2.4.3.4 Acciones | 27 |
| 2.4.4 Mensajes de OpenFlow | 27 |
| 2.4.4.1 Mensajes del controlador al <i>switch</i> | 28 |
| 2.4.4.2 Mensajes asíncronos | 28 |
| 2.4.4.3 Mensajes simétricos | 28 |
| 2.4.5 Cambios en la versión 1.5 | 29 |
| 2.5 Open vSwitch | 29 |
| 2.6 El controlador ONOS | 31 |
| <i>Capítulo 3: Preparación del entorno de trabajo</i> | 33 |
| 3.1 Instalación del SO | 33 |
| 3.2 Instalación de GNS3 | 33 |
| 3.3 Instalación de ONOS | 38 |
| 3.3.1 Instalación en el PC | 38 |
| 3.3.2 Instalación en el simulador gráfico GNS3 | 39 |

| | |
|---|-----------|
| 3.4 Instalación de Eclipse | 42 |
| Capítulo 4: Creando la primera red con GNS3 | 44 |
| 4.1 Conexión con el controlador y con el exterior | 44 |
| 4.1.1 Acceso al controlador desde la página web | 45 |
| 4.1.2 Acceso al controlador desde línea de comandos | 49 |
| 4.2 Hosts conectados a Open vSwitch | 53 |
| Capítulo 5: Aplicaciones desarrolladas | 56 |
| 5.1 Indicaciones previas | 56 |
| 5.2 Aplicación severalping | 58 |
| 5.2.1 Motivación y explicación teórica. | 58 |
| 5.2.2 Componentes de la aplicación | 58 |
| 5.2.2.1 Interceptar los paquetes | 58 |
| 5.2.2.2 Procesando los paquetes | 59 |
| 5.2.2.3 Escuchando los flujos | 60 |
| 5.2.2.4 Parámetros configurables | 60 |
| 5.2.3 Banco de pruebas | 61 |
| 5.2.3.1 Envío de pings entre 2 hosts | 63 |
| 5.2.3.2 Enviando <i>pings</i> entre 2 parejas de <i>hosts</i> | 66 |
| 5.2.3.3 Enviando <i>pings</i> desde 2 <i>hosts</i> a un tercero | 67 |
| 5.2.3.4 Modificando los parámetros configurables | 68 |
| 5.2.3.5 Enviando otro tipo de tráfico | 70 |
| 5.3 Analizando las estadísticas de tráfico | 71 |
| 5.3.1 Aplicación <i>statsshow</i> | 71 |
| 5.3.1.1 Motivación | 71 |
| 5.3.1.2 Componentes de la aplicación | 71 |
| 5.3.1.2.1 Tareas repetidas | 71 |
| 5.3.1.3 Banco de pruebas | 72 |
| 5.3.2 Aplicación <i>detectHost</i> | 73 |
| 5.3.2.1 Motivación y explicación teórica | 73 |
| 5.3.2.2 Componentes de la aplicación | 74 |
| 5.3.2.2.1 Tarea repetida | 74 |
| 5.3.2.2.2 Listener | 75 |
| 5.3.2.3 Banco de pruebas | 75 |
| 5.3.3 Aplicación <i>detectHostBan</i> | 76 |
| 5.3.3.1 Motivación | 76 |
| 5.3.3.2 Componentes de la aplicación | 76 |
| 5.3.3.3 Banco de pruebas | 77 |
| 5.4 Aplicación VLAN | 84 |
| 5.4.1 Motivación y explicación teórica | 84 |
| 5.4.2 Componentes de la aplicación | 86 |
| 5.4.2.1 Fichero <i>VlanByMac.java</i> | 86 |
| 5.4.2.1.1 Host añadido a la red | 86 |
| 5.4.2.1.2 Host eliminado de la red | 87 |
| 5.4.2.2 Ficheros auxiliares | 87 |
| 5.4.3 Banco de pruebas | 88 |
| 5.4.3.1 Prueba de la aplicación sin utilizar comandos ni <i>routers</i> | 88 |
| 5.4.3.2 Prueba con el <i>router</i> | 90 |
| 5.4.3.3 Probando los comandos | 95 |
| 5.5 Aplicación <i>fwdBalanceo</i> | 98 |
| 5.5.1 Motivación y explicación teórica | 98 |
| 5.5.2 Componentes de la aplicación | 101 |

| | |
|--|------------|
| 5.5.2.1 Método <i>activate</i> | 101 |
| 5.5.2.2 Método <i>installRule</i> | 102 |
| 5.5.3 Banco de pruebas | 103 |
| 5.6 Aplicación FakeDHCP | 106 |
| 5.6.1 Explicación teórica | 106 |
| 5.6.2 Componentes de la aplicación | 107 |
| 5.6.3 Banco de pruebas | 107 |
| 5.7 Calidad de servicio | 114 |
| 5.7.1 Motivación y explicación | 114 |
| 5.7.2 Componentes de la aplicación | 122 |
| 5.7.3 Banco de pruebas | 123 |
| Capítulo 6: Conclusiones y líneas futuras | 129 |
| Bibliografía | 131 |
| Anexo 1: Introducción a la programación por objetos | 134 |
| 1. ¿Qué es la programación orientada a objetos? | 134 |
| 2. Elementos de la programación orientada por objetos | 134 |
| 3. Propiedades de la orientación a objetos | 137 |
| 3.1 Herencia | 138 |
| 3.2 Abstracción | 138 |
| 3.3 Polimorfismo | 139 |
| 3.4 Encapsulamiento | 139 |
| Anexo 2: Introducción a git y al repositorio GitHub | 140 |

Índice de figuras y tablas

| | |
|--|----|
| Figura 1: Arquitectura red SDN. | 22 |
| Figura 2: Topología de la red ejemplo. | 23 |
| Figura 3: Arquitectura Open vSwitch. | 23 |
| Figura 4: Proceso de pipeline en un Switch OpenFlow. | 25 |
| Figura 5: Proceso de matching en el switch OpenFlow. | 27 |
| Figura 6: Proceso de pipeline en OpenFlow 1.5. | 29 |
| Figura 7: Tecnologías soportadas por Open vSwitch. | 30 |
| Figura 8: Estructura del Open vSwitch. | 30 |
| Figura 9: Arquitectura del controlador ONOS. | 32 |
| Figura 10: Página web de VMware. | 34 |
| Figura 11: Página principal de la web del simulador gráfico GNS3. | 35 |
| Figura 12: Pestaña de descargas disponibles de GNS3. | 35 |
| Figura 13: Interfaz de inicio del programa VMware. | 36 |
| Figura 14: Explorador de archivos con el que se selecciona la máquina virtual. | 36 |
| Figura 15: Pestaña de selección de máquina virtual en VMware. | 37 |
| Figura 16: Ventana de información de la máquina virtual instalada. | 37 |
| Figura 17: Ventana mostrada al ejecutar la máquina virtual. | 38 |
| Figura 18: Selección del controlador ONOS para iniciar su instalación en GNS3. | 39 |
| Figura 19: Información sobre el dispositivo que vamos a instalar. | 40 |
| Figura 20: Selección de servidor en el que se va a instalar el dispositivo. | 40 |
| Figura 21: Instalación completada de ONOS. | 41 |
| Figura 22: Configuración del dispositivo. | 42 |
| Figura 23: Aplicación software desde la que se instala el programa Eclipse. | 42 |
| Figura 24: Add-ons necesarios para el funcionamiento del proyecto. | 43 |
| Figura 25: Red básica utilizada durante el transcurso del proyecto. | 44 |
| Figura 26: Parte de la red referente al controlador y la conexión con el exterior. | 44 |
| Figura 27: Fichero de configuración de las interfaces del controlador ONOS. | 45 |
| Figura 28: Página principal GUI ONOS. | 46 |
| Figura 29: Menú con las opciones de la GUI de ONOS. | 46 |
| Figura 30: Vista desde la GUI de ONOS de los dispositivos conectados a la red. | 47 |
| Figura 31: Botones con las opciones para ver en detalle el comportamiento de los dispositivos. | 47 |
| Figura 32: Pestaña de análisis de los flujos del dispositivo. | 48 |
| Figura 33: Pestaña de análisis de los puertos del dispositivo. | 48 |
| Figura 34: Vista desde la GUI de ONOS de los hosts conectados a la red. | 49 |
| Figura 35: Comandos de acceso a la máquina virtual de GNS3. | 49 |
| Figura 36: Acceso a la máquina virtual de GNS3. | 50 |
| Figura 37: Muestra de los contenedores de Docker instalados. | 51 |
| Figura 38: Selección de la máquina virtual al añadir un nuevo contenedor Docker. | 51 |
| Figura 39: Selección del contenedor Docker. | 52 |
| Figura 40: Parte de la red referida al Open vSwitch y los hosts. | 53 |
| Figura 41: Fichero de configuración de los hosts de la red. | 53 |
| Figura 42: Comandos de configuración iniciales del Open vSwitch. | 54 |
| Figura 43: Estado del Open vSwitch tras configurarlo. | 55 |
| Figura 44: Vista desde el Eclipse de una aplicación recién importada. | 57 |
| Figura 45: Interfaz para la gestión del controlador. | 62 |
| Figura 46: Aplicaciones necesarias para activar la aplicación. | 62 |
| Figura 47: Vista desde el controlador de la activación de la aplicación severalping. | 63 |
| Figura 48: Envío de 10 pings. | 63 |
| Figura 49: Resultado del envío de los 10 pings. | 64 |
| Figura 50: Vista desde el controlador del envío de los pings. | 64 |
| Figura 51: Vista de los flujos al activar la aplicación. | 65 |
| Figura 52: Vista de los flujos al banearse el enlace. | 65 |
| Figura 53: Envío de 2 pings entre 2 parejas de hosts diferentes. | 66 |
| Figura 54: Flujos creados en la aplicación. | 67 |

| | |
|--|-----|
| Figura 55: Ejecución del comando que envía pings desde 2 hosts a un tercero. | 67 |
| Figura 56: Vista desde el controlador de la ejecución del comando. | 68 |
| Figura 57: Comandos que modifican los parámetros configurables. | 68 |
| Figura 58: Vista desde el controlador de la modificación de los parámetros configurables. | 69 |
| Figura 59: Resultado del envío de 10 pings con los parámetros cambiados. | 69 |
| Figura 60: Vista desde el controlador del resultado del envío de los pings. | 70 |
| Figura 61: Comandos para comunicar 2 hosts mediante la herramienta netcat. | 70 |
| Figura 62: Envío de tráfico TCP a través de la herramienta netcat. | 71 |
| Figura 63: Envío de pings para comprobar el funcionamiento de la aplicación. | 72 |
| Figura 64: Vista desde el controlador de las estadísticas recogidas. | 73 |
| Figura 65: Fichero BUILD de la aplicación hostProbingProvider. | 74 |
| Figura 66: Vistas desde el controlador al activar la aplicación. | 75 |
| Figura 67: Vista desde el controlador de la detección de los hosts. | 75 |
| Figura 68: Vista desde el controlador de la desconexión de un host. | 76 |
| Figura 69: Vista desde el controlador de la activación de la aplicación detectHostBan. | 78 |
| Figura 70: Vista desde el controlador de los datos acumulados. | 78 |
| Figura 71: Captura de Wireshark del protocolo LLDP. | 79 |
| Figura 72: Vista desde el controlador del baneo de los hosts al superar el umbral. | 80 |
| Figura 73: Vista de los flujos creados al banear hosts. | 80 |
| Figura 74: Comprobación del baneo de los hosts al superar el umbral. | 81 |
| Figura 75: Vista de la eliminación del baneo de los hosts. | 82 |
| Figura 76: Comparativa de los pings enviados frente a los recibidos. | 82 |
| Figura 77: Modificación de los parámetros configurables. | 83 |
| Figura 78: Comprobación de la modificación de los parámetros. | 83 |
| Figura 79: Comprobación del baneo con los nuevos parámetros. | 84 |
| Figura 81: Diferencias entre una trama Ethernet sin etiquetado VLAN (802.1Q) y otra con él. | 85 |
| Figura 82: Detalles acerca de la trama 802.1Q. | 85 |
| Figura 83: Aplicaciones necesarias para el funcionamiento de VLAN. | 89 |
| Figura 84: Envío de pings entre hosts de la misma VLAN. | 89 |
| Figura 85: Envío de pings entre hosts de diferentes VLAN. | 89 |
| Figura 86: Red con el router añadido. | 90 |
| Figura 87: Nueva configuración de los hosts. | 91 |
| Figura 88: Salida del comando ifconfig ejecutado en el router. | 92 |
| Figura 89: Salida del comando route. | 92 |
| Figura 90: Captura de tráfico del router al enviar pings entre 2 hosts de la misma VLAN. | 93 |
| Figura 91: Envío del ping entre hosts de 2 VLAN diferentes. | 94 |
| Figura 92: Captura de tráfico del router al enviar tráfico entre 2 hosts de diferentes VLAN. | 94 |
| Figura 93: Captura de tráfico del router al enviar tráfico entre 2 hosts de diferentes VLAN. | 95 |
| Figura 94: Salida del comando show-Vlan-Mac. | 95 |
| Figura 95: Salida del comando add-Mac-Vlan. | 96 |
| Figura 96: Envío de pings con el host añadido. | 96 |
| Figura 97: Eliminación de la VLAN del host. | 97 |
| Figura 98: Ejecución del comando show-Vlan-Mac. | 97 |
| Figura 99: Comprobación de que los pings no llegan al eliminar el host. | 98 |
| Figura 100: Red utilizada para la aplicación fwdBalanceo. | 99 |
| Figura 101: Modificaciones del fichero pom.xml. | 99 |
| Figura 102: Grupos en OpenFlow. | 100 |
| Figura 103: Envío de los pings entre los 2 Open vSwitch. | 103 |
| Figura 104: Salida del comando groups. | 103 |
| Figura 105: Envío de pings de mayor tamaño. | 104 |
| Figura 106: Muestra de los grupos creados. | 105 |
| Figura 107: Vista del controlador de los buckets añadidos al grupo. | 105 |
| Figura 108: Configuración de un host mediante DHCP. | 108 |
| Figura 109: Red utilizada para la aplicación FakeDHCP. | 108 |
| Figura 110: Vista desde el Wireshark de los DHCP Offer enviados. | 109 |
| Figura 111: Red con el router falso añadido. | 110 |
| Figura 112: Vista de las direcciones IP de los routers. | 110 |

| | |
|---|-----|
| <i>Figura 113: Configuración del servidor DHCP.</i> | 111 |
| <i>Figura 114: Vista del controlador de la activación de la aplicación FakeDHCP.</i> | 111 |
| <i>Figura 115: Comprobación de la dirección IP recibida por el host.</i> | 112 |
| <i>Figura 116: Captura de wireshark del enlace entre el host y el Open vSwitch.</i> | 112 |
| <i>Figura 117: Captura de wireshark del enlace entre el router bueno y el Open vSwitch.</i> | 113 |
| <i>Figura 118: Captura de wireshark del enlace entre el router falso y el Open vSwitch.</i> | 113 |
| <i>Figura 119: Vista del controlador del bloqueo de los DHCP OFFER falsos.</i> | 114 |
| <i>Figura 120: Cabecera IPv4 añadiendo la calidad de servicio.</i> | 115 |
| <i>Figura 121: Detalle de los campos añadidos a la cabecera.</i> | 115 |
| <i>Figura 122: Clasificación de los servicios en función de la sensibilidad a retardos.</i> | 115 |
| <i>Figura 123: Valores del campo DSCP.</i> | 117 |
| <i>Figura 124: Representación de Leaky Bucket.</i> | 117 |
| <i>Figura 125: Representación de Token Bucket.</i> | 118 |
| <i>Figura 126: Representación de Dual Token Bucket.</i> | 119 |
| <i>Figura 127: Ejemplo de curvas WRED.</i> | 119 |
| <i>Figura 128: Gestión de buffers.</i> | 120 |
| <i>Figura 129: Vista de la configuración de los dispositivos necesarios.</i> | 121 |
| <i>Figura 130: Vista de los drivers cargados.</i> | 121 |
| <i>Figura 131: Vista de las colas creadas al activar la aplicación en el controlador.</i> | 124 |
| <i>Figura 132: Vista de las colas creadas en el Open vSwitch.</i> | 124 |
| <i>Figura 133: Vista en detalle de las colas creadas.</i> | 125 |
| <i>Figura 134: Vista en detalle de las colas creadas en el Open vSwitch.</i> | 126 |
| <i>Figura 135: Prueba del envío de datos a través de las colas.</i> | 126 |
| <i>Figura 136: Comprobación de la cola por la cual se ha enviado el tráfico.</i> | 127 |
| <i>Figura 137: Comprobación de los flujos creados por la aplicación.</i> | 127 |
| <i>Figura 138: Representación de un objeto.</i> | 135 |
| <i>Figura 139: Representación esquemática de un objeto con sus métodos.</i> | 135 |
| <i>Figura 140: Representación del envío de un mensaje entre 2 objetos.</i> | 136 |
| <i>Figura 141: Ejemplo del envío de un mensaje.</i> | 136 |
| <i>Figura 142: Representación de la herencia.</i> | 138 |

Capítulo 1. Introducción

1.1 Contexto

El concepto de redes definidas por *software* (*Software Defined Network*, SDN) es un enfoque arquitectónico de la red que permite que la red sea controlada de manera inteligente y central, o que su comportamiento se vea programado utilizando aplicaciones de *software* [1].

Es en este ámbito en el que se encuadra el presente TFG. La motivación que da lugar a este TFG radica fundamentalmente en la necesidad de crear diversas aplicaciones que abarquen requisitos de las redes actuales que supongan un avance a la hora de lograr la auto-adaptatividad de una red.

1.2 Objetivos del proyecto

El objetivo principal del proyecto consiste en la propuesta, desarrollo y evaluación de mecanismos SDN para avanzar hacia redes auto-adaptativas. Para ello se realizarán diversas aplicaciones que satisfagan necesidades actuales de las redes cuya implementación sería mucho más costosa en las redes tradicionales.

Para alcanzar el objetivo principal se plantean los siguientes objetivos específicos:

- Desarrollar y evaluar una aplicación que permita limitar el número de *pings* entre *hosts*.
- Desarrollar y evaluar una aplicación que permite mostrar las estadísticas de tráfico enviado y recibido entre *hosts*.
- Desarrollar y evaluar una aplicación que permita bloquear enlaces cuando uno o varios *hosts* superen un cierto umbral de datos.
- Desarrollar y evaluar una aplicación que permita crear redes virtuales (VLANs).
- Desarrollar y evaluar una aplicación que permita evitar la conexión de servidores DHCP falsos en los puertos de acceso del *switch*.
- Desarrollar y evaluar una aplicación que permita crear colas en los *switches*, y asignarles diferentes calidades de servicio.

1.3 Fases

Para simplificar el desarrollo del proyecto se han seguido diversas fases, las cuales se procede a detallar a continuación:

- Documentación acerca del funcionamiento y características de las redes SDN.
- Documentación acerca del funcionamiento del protocolo OpenFlow.
- Estudio sobre el funcionamiento de los *switches* que soportan el protocolo OpenFlow, en concreto el *switch* virtual Open vSwitch.
- Documentación en relación con el software GNS3.
- Estudio de los principales casos de uso de las redes SDN.

- Estudio del estado del arte en referencia a las aplicaciones desarrolladas para redes SDN.
- Estudio de las diferentes tecnologías de redes utilizadas, concretamente VLAN, ICMP, DHCP y QoS.
- Documentación acerca de los diferentes controladores disponibles para las redes SDN con el objetivo de elegir el que mejor se adecuaba, en este caso el controlador ONOS.
- Programación de las diferentes aplicaciones desarrolladas sobre el controlador ONOS ubicado en las redes creadas con el *software* GNS3.
- Ejecución de las aplicaciones para comprobar el correcto funcionamiento de las mismas, exponiéndolas a diferentes pruebas y escenarios que puedan ocurrir.
- Identificación de las líneas futuras del presente Trabajo de Fin de Grado.
- Redacción de la memoria final del Trabajo de Fin de Grado.

1.4 Medios empleados

Durante el transcurso del TFG fueron necesario los siguientes materiales:

- Ordenador portátil Lenovo *IdeaPad* 320S que consta de un procesador Intel® Core™ i5-8250U a 1,6GHz, 8GB de memoria RAM de tipo DDR4 y 1TB de almacenamiento a través de un disco duro de tipo magnético (HDD). El sistema operativo instalado es una distribución GNU/Linux Fedora 29 con un entorno de escritorio GNOME3. Este dispositivo se ha utilizado tanto para el desarrollo del proyecto como para la redacción de la memoria.
- *Software* GNS3 usado como simulador gráfico de redes para diseñar las topologías necesarias y poner en marcha simulaciones sobre ellas.
- *Software* Eclipse utilizado para desarrollar las diferentes aplicaciones en código Java realizadas a lo largo del presente TFG.
- Controlador ONOS utilizado para gestionar la red de forma centralizada.
- *Switch* Open vSwitch utilizado para interconectar los *hosts* de la red de acuerdo con la arquitectura SDN, ya que soporta el protocolo OpenFlow.
- *Software* VMware Workstation Pro™. utilizado para la máquina virtual de GNS3, donde se guardarán las diferentes imágenes *Docker* utilizadas.

1.5 Organización de la memoria

La presente memoria está organizada en 6 capítulos.

En el Capítulo 1, se realiza una introducción a este Trabajo Fin de Grado, en el que se establece el contexto, los objetivos, las fases y los medios empleados para su realización.

En el Capítulo 2, se ha realizado un breve estudio del arte, en el que se detalla el funcionamiento de las diferentes tecnologías realizadas, como SDN u OpenFlow.

En el Capítulo 3, se detallan los primeros pasos, que fundamentalmente se refiere a la instalación de todo el *software* necesario para la realización del proyecto, tales como GNS3 y ONOS.

En el Capítulo 4, se explica cómo implementar y configurar la red que se va a utilizar durante todo el transcurso del proyecto en el simulador de redes GNS3.

En el Capítulo 5, se explican en detalle cada una de las aplicaciones realizadas. Cada una de las aplicaciones consta de la siguiente estructura: (i) Motivación y explicación teórica, en aquellas que lo requieran, acerca de los conocimientos previos necesarios para realizar y comprender la aplicación; (ii) Componentes de la aplicación; y (iii) Banco de pruebas al que se somete a la aplicación para comprobar la correcta funcionalidad de la misma.

Finalmente, en el Capítulo 6, se encuentran las conclusiones y líneas futuras extraídas tras la realización del trabajo expuesto en el resto de la memoria de este TFG.

Capítulo 2: Estado del arte

2.1 Historia de las redes SDN

Las redes definidas por *software* (SDN, *Software Defined Networking*) permiten a los ingenieros y administradores de redes 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 [2] [3] [4].

En primer lugar, el periodo denominado **redes activas**, que abarca entre el año 1995 hasta 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*) [5], ya que era un proceso muy lento y no permitía avanzar al ritmo deseado en esa época. Este tipo de redes están orientadas hacia el control de la red incluyendo una interfaz de programación (API, *Application Programming Interface*) 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 entre 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 aquél 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 la depuración de errores (*debugging*).

La tercera etapa fue la aparición del protocolo **OpenFlow** [6], que es considerado uno de los primeros estándares de las redes SDN y surgió de raíz de separar los planos de control y de datos. Su funcionamiento más en detalle se mostrará en el apartado 2.4. 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 *routing*, y los *switches* que dispone la red, que llevan a cabo el *forwarding*. Cabe resaltar que los *switches* tradicionales no soportan este protocolo y, por tanto, fue necesario desarrollar unos nuevos que lo soportaran. Actualmente la mayoría de *switches* comerciales ya incorporan soporte a este protocolo, entre ellos los de la marca *Cisco* [7].

Algunos de los beneficios que introdujo OpenFlow son:

- Flexibilidad en el uso de la red, así como en la forma de operar sobre ella.
- Es un protocolo abierto que hace que, por tanto, pueda ser estudiado y modificado de forma accesible.
- Reducir el gasto de operación.
- Mejorar la integración de la red. Permite que la red sea controlada con un solo punto de vista.

2.2 Definición y arquitectura de las redes SDN

Las redes definidas por *software* (SDN) 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 [8]. 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 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 [9]:

- Programabilidad: La red se puede configurar a través de aplicaciones escritas en lenguajes de programación 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 concentrada en el/los controlador/es, que tiene/n una visión completa de la red y proporciona/n las instrucciones de control de flujo a los diferentes conmutadores o *switches*.
- 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 disponer de estos elementos [6]:

- Plano de aplicaciones (Application Plane): Las aplicaciones SDN son programas que comunican el comportamiento deseado de la red al controlador SDN. Todas las aplicaciones juntas forman el plano de aplicaciones.
- Plano de control (Control Plane): 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 (SDN Controller): 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. La forma de comunicación entre los diversos controladores que pueda poseer una misma red se realiza a través de las interfaces hacia el este (*eastbound API*) y hacia el oeste (*westbound API*).
- Plano de datos (Data plane): Compuesta por los diversos elementos de red como los *switches* o sistemas finales (*hosts*).
- Interfaz hacia el norte (northbound API): Es la API que comunica la aplicación SDN y los controladores.
- Interfaz hacia el sur (southbound API): Es el protocolo que comunica el plano de control con la capa de datos, generalmente es OpenFlow.

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

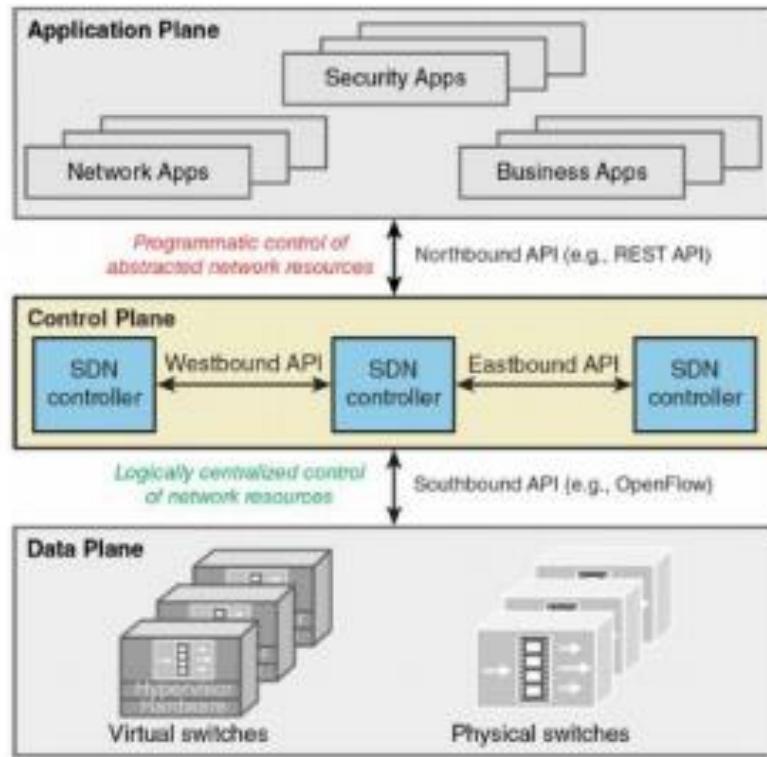


Figura 1: Arquitectura red SDN.

2.3 Funcionamiento básico 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 3 *hosts*, que denominaremos PC-A, PC-B y PC-C; conectados cada uno de ellos a un *switch* (Open vSwitch), que soporta el protocolo OpenFlow; y el controlador, conectado a uno de los tres *switches* de la red. En la Figura 2 se puede observar la topología descrita [10].

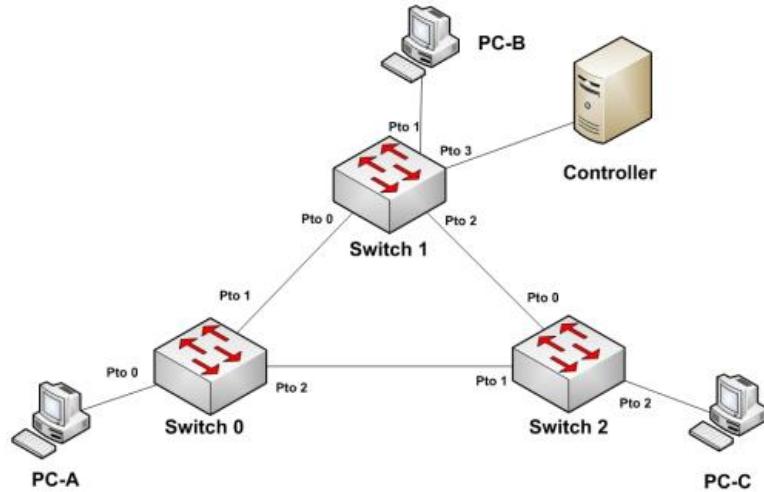


Figura 2: Topología de la red ejemplo.

Supongamos que el *host* PC-A envía una trama al *host* PC-B. La trama, en primer lugar, llega al Open vSwitch (Switch 0). Si dicho *switch* posee una entrada en sus tablas de flujos con correspondencia para dicha trama para algún campo de capa 2, 3 o 4, se actúa según dicha regla, mientras que, si el *switch* no posee una entrada con correspondencia en sus tablas, envía el paquete al controlador para que éste detalle cómo se debe commutar dicha trama. Esto mismo ocurre en cada *switch* que tengamos en la red con soporte a OpenFlow.

2.4 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 el plano de datos y el controlador. Hay varios modelos de *switches* que soportan OpenFlow, aunque en este TFG nos vamos a centrar en el denominado Open vSwitch. Cabe destacar que la versión utilizada del protocolo OpenFlow es la 1.4. En el apartado 2.4.5 se explican brevemente los cambios que supone trabajar con la versión 1.5.

2.4.1 Componentes de un *switch* que soporta OpenFlow.

Un *switch* OpenFlow tiene los componentes mostrados en la Figura 3 [11].

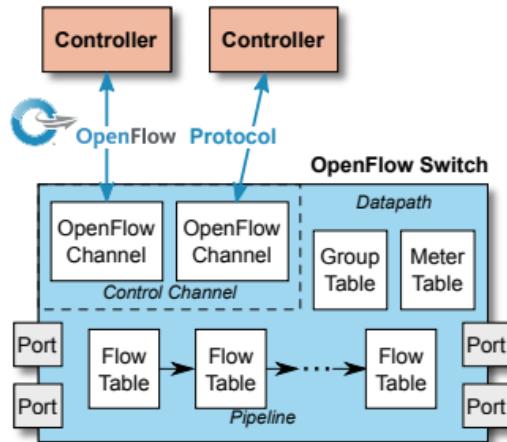


Figura 3: Arquitectura Open vSwitch.

Tal y como se puede observar en la anterior figura, los componentes son fundamentalmente los siguientes:

- Puertos (*Ports*).
- Tablas de flujo (*Flow Tables*).
- Tablas de grupo (*Group Tables*).
- Canales de OpenFlow (*OpenFlow Channels*), utilizados para la comunicación con el controlador.

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

2.4.2 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 envía al controlador.
- Puerto reservado: Utilizado para la comunicación con el controlador o para la conmutación tradicional (sin hacer uso del protocolo OpenFlow). A continuación, se detallan los puertos reservados que obligatoriamente debe poseer un *switch* con soporte OpenFlow.
 - **ALL**: Representa todos los puertos que un *switch* puede usar para reenviar un paquete. Sólo 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 del *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 en algunas peticiones cuando no se especifica ningún puerto. Algunas peticiones OpenFlow contienen referencias a puertos específicos a los que aplica la susodicha petición. Usar este valor (ANY) como número de puerto en esas peticiones permite que se aplique a cualquiera (todos) los puertos. Este valor no puede emplearse como puerto de entrada o salida en las tablas.
 - **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 ni como puerto de

entrada. Este valor tampoco puede emplearse como puerto de entrada ni salida en las tablas.

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

- **LOCAL:** Representa la pila de protocolos de la red local del *switch*.
- **NORMAL:** Representa el reenvío 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).

2.4.3 Tablas de OpenFlow

En este apartado se detalla, en primer lugar, el proceso de *pipeline* de OpenFlow, así como los componentes de las tablas de flujo y de grupo. A continuación, se explica el mecanismo de *matching* y finalmente el gestor de acciones.

2.4.3.1 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 procesados por el *pipeline* de OpenFlow, mientras que los segundos admiten tanto las funcionalidades OpenFlow como las normales que aporta la commutación Ethernet.

El *pipeline* de OpenFlow de todos los *switches* contiene tablas de flujo, y cada tabla de flujo contiene múltiples entradas. Éste define como interaccionan los paquetes con las tablas de flujo. En la Figura 4 se muestra como es el flujo de los paquetes cuando tenemos más de una tabla de flujo.

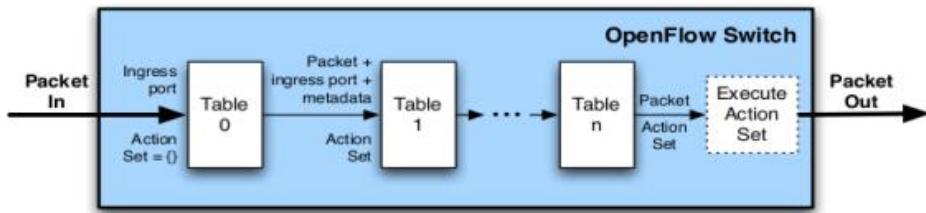


Figura 4: Proceso de pipeline en un Switch OpenFlow.

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

El proceso de *pipeline* siempre comienza por la primera tabla de flujo, en primer lugar, se comprueba el *match* (ver apartado 2.4.3.3) 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 están detalladas en el apartado 2.4.3.4, aunque se adelanta 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.

2.4.3.2 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 |
|--------------|----------|----------|--------------|----------|--------|-------|
|--------------|----------|----------|--------------|----------|--------|-------|

Tabla 1: Estructura de una entrada de una tabla de flujo.

Donde:

- Match Fields: campos que permiten hacer la operación de *matching*. Consiste en el puerto de entrada y las diversas cabeceras 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.

2.4.3.3 Matching

Cuando el *switch* OpenFlow recibe un paquete se activa el diagrama de flujo que se puede ver en la Figura 5:

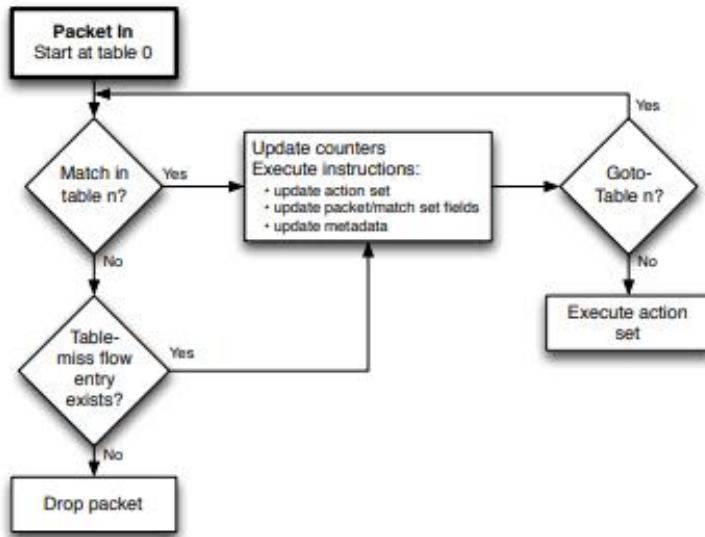


Figura 5: Proceso de matching en el switch OpenFlow.

Tal y como se puede observar, el *switch* comienza realizando una búsqueda en la primera tabla de flujo, y basado en el procesamiento *pipeline* explicado anteriormente, en caso de no haber coincidencia en la primera tablas realiza búsquedas en el resto de las tablas.

Finalmente, en caso de no haber coincidencia se busca la existencia de la *table-miss*. En caso de haberla se ejecutan las acciones que la incluyan, mientras que, en caso de no coincidencia se descarta el paquete.

2.4.3.4 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 por `group_id`.
- Drop: Descarta el paquete.
- Push VLAN header: Añade el campo VLAN a la que pertenece el paquete
- Pop VLAN header: Elimina el campo VLAN de la cabecera del paquete.
- GOTO-TABLE: Busca coincidencias en la tabla que se le indique.

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

2.4.4 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 del controlador al *switch*, asíncronos y síncronos.

2.4.4.1 Mensajes del controlador al *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 consiguen 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.

2.4.4.2 Mensajes asíncronos

Estos mensajes se envían entre el *switch* y el controlador al llegar un paquete. Entre ellos se encuentran 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.

2.4.4.3 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.

2.4.5 Cambios en la versión 1.5

Tal y como se ha comentado, en este TFG, se ha utilizado la versión 1.4 del protocolo OpenFlow. Sin embargo, ya está desarrollada la versión 1.5 que introduce una serie de cambios. Entre las modificaciones más importantes destacan las siguientes [12]:

- Se introducen las tablas de salida, posibilitando el procesado en el contexto del puerto de salida. Cuando llega un flujo, puede comenzar a ser procesado en las tablas de entrada y ser redirigido a las tablas de salida. Tal y como se ve en la Figura 6 el proceso de *pipeline* se modifica sustancialmente, permitiendo una versatilidad mucho mayor.

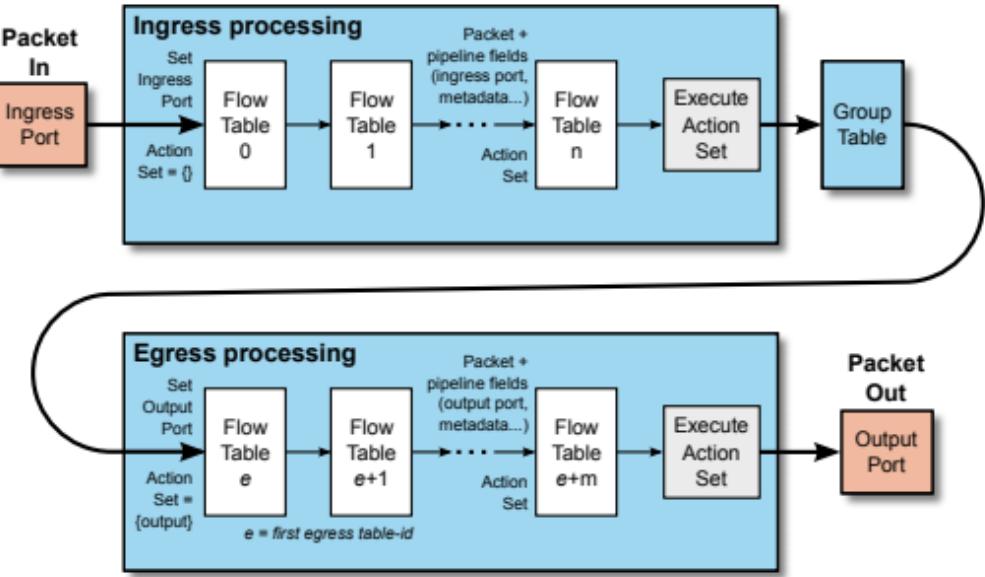


Figura 6: Proceso de pipeline en OpenFlow 1.5.

- Se definen tipos de paquetes diferentes a *Ethernet*, tales como paquetes PPP.
- Posibilidad de realizar el proceso de *matching* a partir de las *flags* de los paquetes TCP como SYN, ACK y FIN. Una utilidad es detectar el inicio y fin de las comunicaciones TCP.
- En versiones anteriores, las operaciones sobre los grupos sólo podían cambiar todas las acciones de los *group buckets*. A partir de esta versión permite insertar y/o eliminar acciones a los *group buckets* de manera aislada.

2.5 Open vSwitch

Tal y como se ha comentado, hay varios *switches* que soportan el protocolo OpenFlow, sin embargo, a lo largo del TFG se va a utilizar el Open vSwitch [13], que se procede a explicar a continuación.

Open vSwitch es un *software* de código abierto diseñado para actuar como un conmutador virtual en un escenario con máquinas virtuales. Es el encargado de conmutar tráfico entre máquinas localizadas en un mismo equipo físico, así como la comunicación entre máquinas virtuales y la red física. Es compatible además con diferentes tecnologías y protocolos tales como VLAN, NetFlow, QoS... En la Figura 7 se muestra con detalle las tecnologías soportadas por Open vSwitch.

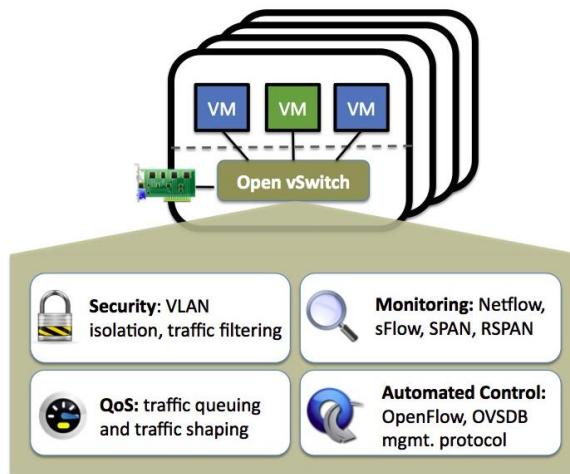


Figura 7: Tecnologías soportadas por Open vSwitch.

Open vSwitch puede funcionar en una red que contenga controladores, o en modo *stand-alone*, en el que actúa como un comutador *Ethernet* basado en aprendizaje (similar al funcionamiento de los *switches* físicos) o configurándolo a través de órdenes transmitidas por consola de comandos.

La estructura del Open vSwitch se puede ver en la Figura 8.

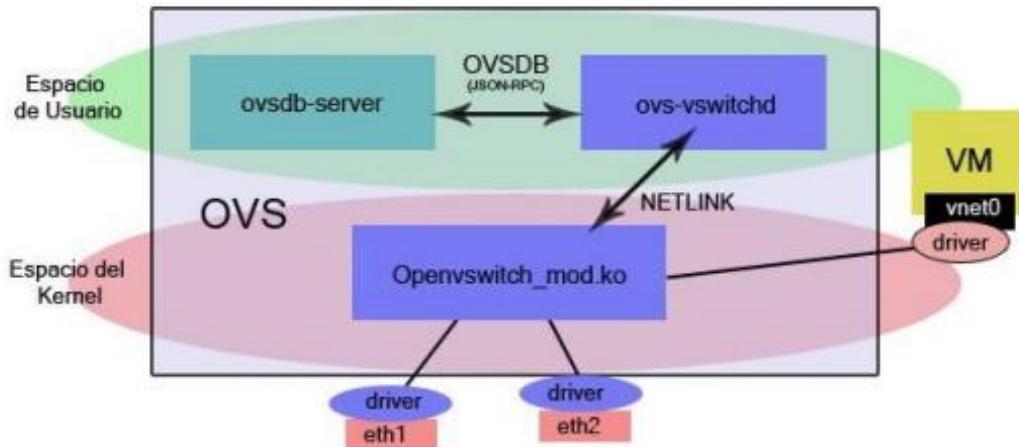


Figura 8: Estructura del Open vSwitch.

Tal y como se puede ver, consta de 3 módulos diferentes:

- Ovs-vswitchd: Componente que recibe paquetes procedentes de un flujo nuevo en la red y determina cómo se van a tratar. En el caso de haber un controlador, se comunica con él.
- Ovsdb-server: Servidor en el que se almacena la configuración del comutador. Se comunica con el ovs-vswitchd y con el controlador (en caso de existir) a través del protocolo OVSDB.

- Openvswitch_mod.ko: Módulo que opera en el *kernel* cuya función es encaminar paquetes pertenecientes a un flujo que ya ha sido definido. En caso de que no haya coincidencia con algún flujo, se envía al módulo ovs-vswitchd que es el encargado de encaminarlo y establecer una nueva entrada en la tabla de flujos. La comunicación entre los 2 módulos se realiza a través del protocolo NETLINK. Al ser un módulo ejecutado en el *kernel*, su velocidad de operación es más elevada.

Otros elementos y herramientas que contiene el Open vSwitch y que serán de utilidad son:

- ovs-dpctl: Herramienta que permite la configuración del *kernel* del *switch*.
- ovs-vsctl: Utilidad que permite consultar y actualizar la configuración del módulo ovs-vswitchd.
- ovs-appctl: Utilidad que envía comandos para ejecutar los demonios de Open vSwitch.
- ovs-ofctl: Utilidad para consultar y controlar los *switches* OpenFlow y controladores.
- ovs-testcontroller: Controlador OpenFlow sencillo que permite ser empleado para testar.

2.6 El controlador ONOS

ONOS (*Open Network Operating System*) [14] [15] es un proyecto de código abierto, surgido en el año 2014 gracias a *Open Networking Lab*, aunque en el año 2015 pasó a formar parte de *The Linux Foundation*.

Las características fundamentales que proporciona este controlador son las siguientes:

- Alta disponibilidad y resiliencia: ONOS soporta las redes de operadores más exigentes, garantizando la fiabilidad de las conexiones para que los clientes no experimenten tiempos de inactividad en la red.
- Software modular: Las funciones *software* se implementan en módulos independientes mediante la definición de interfaces, lo que facilita la lectura, compresión y mantenibilidad. Existe una gran variedad de aplicaciones y extensiones.
- Rendimiento a escala: ONOS se adapta al crecimiento de la red, ya que puede atender millones de solicitudes en un tiempo menor a 50 milisegundos. Para conseguirlo agrega nuevas instancias del controlador de forma automática cuando requiere más capacidad en el plano de control.
- Separación en tres capas de los módulos de ONOS:
 - Módulos que interactúan con la red (*southbound APIs*)
 - Núcleo del sistema que recibe la información sobre el estado de la red
 - Aplicaciones que reciben la información por parte del núcleo del sistema y le comunican cambios que debe transmitir a la red (*northbound APIs*)
- Independencia del protocolo: A pesar de que en general se utilice el protocolo OpenFlow, la arquitectura de ONOS no está vinculada al uso de éste, sino que se pueden diseñar módulos que interactúen con las *southbound APIs* para soportar nuevos protocolos.
- Aplicaciones escritas en lenguaje de alto nivel: Las aplicaciones se escriben en código Java. También existe una API (ONOS Java API) [16].

La arquitectura del controlador se puede ver en la Figura 9.

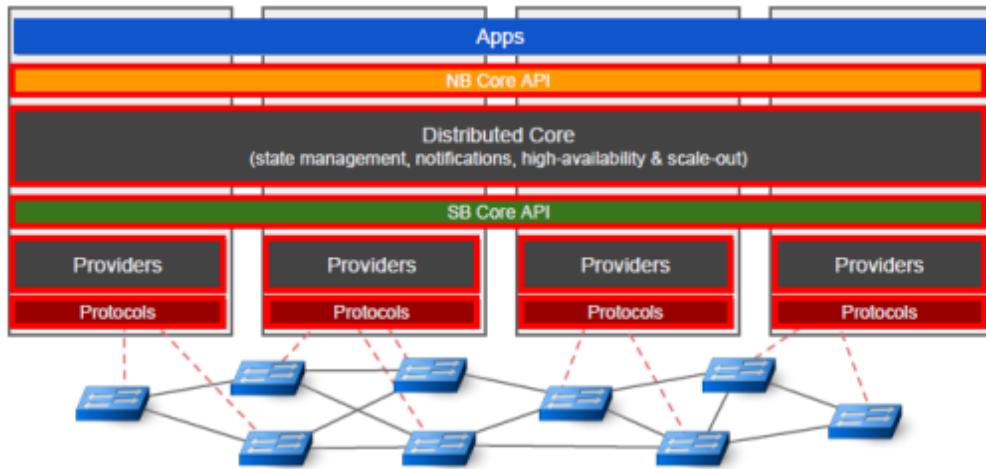


Figura 9: Arquitectura del controlador ONOS.

Tal y como se puede ver, es una arquitectura similar a la de una red SDN existiendo módulos que conforman las 3 capas principales: aplicación, control y datos.

Capítulo 3: 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.

3.1 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 en principio.

El motivo de haber elegido este sistema operativo se debió a diferentes problemas de compatibilidad con el ordenador acaecido 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 [17].

3.2 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 a continuación [18].

Instalamos dependencias que necesitamos, como Python o Wireshark [19]:

- `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-netifaces python3-devel python-pip`
- `python-pip python3-setuptools python3-PyQt4 python3-zmq`
- `sudo dnf -y install wireshark`

A continuación, 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™ [20]. 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 Figura 10.

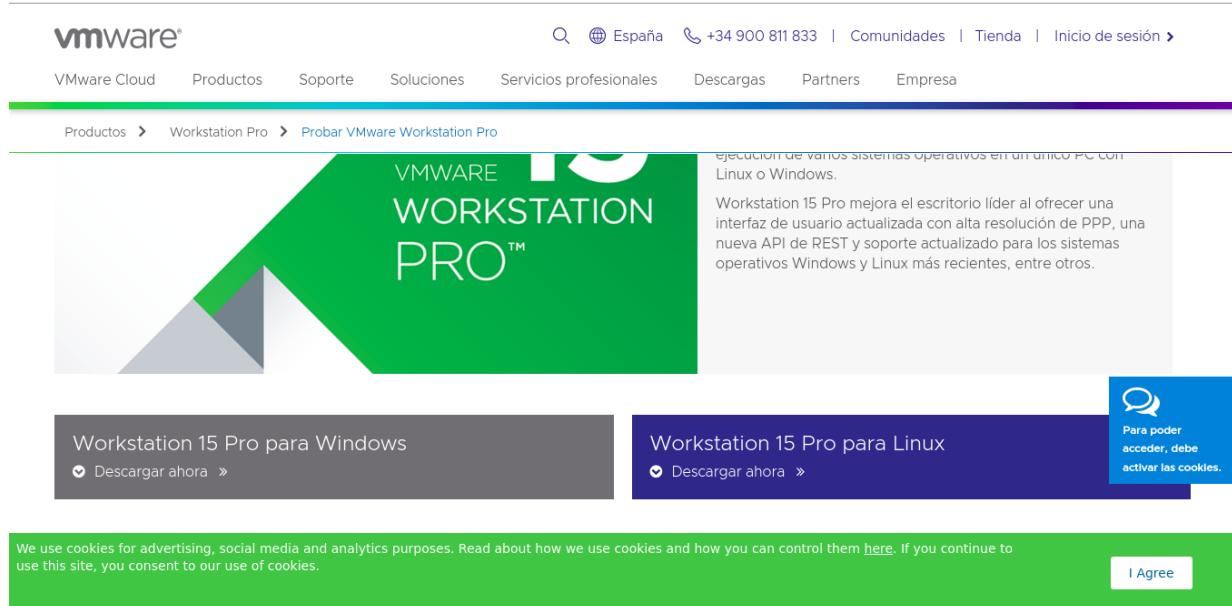


Figura 10: Página web de VMware.

Seguidamente, se descarga la máquina virtual para GNS3 disponible en la página web, tal y como se ve en la Figura 11 y la Figura 12.

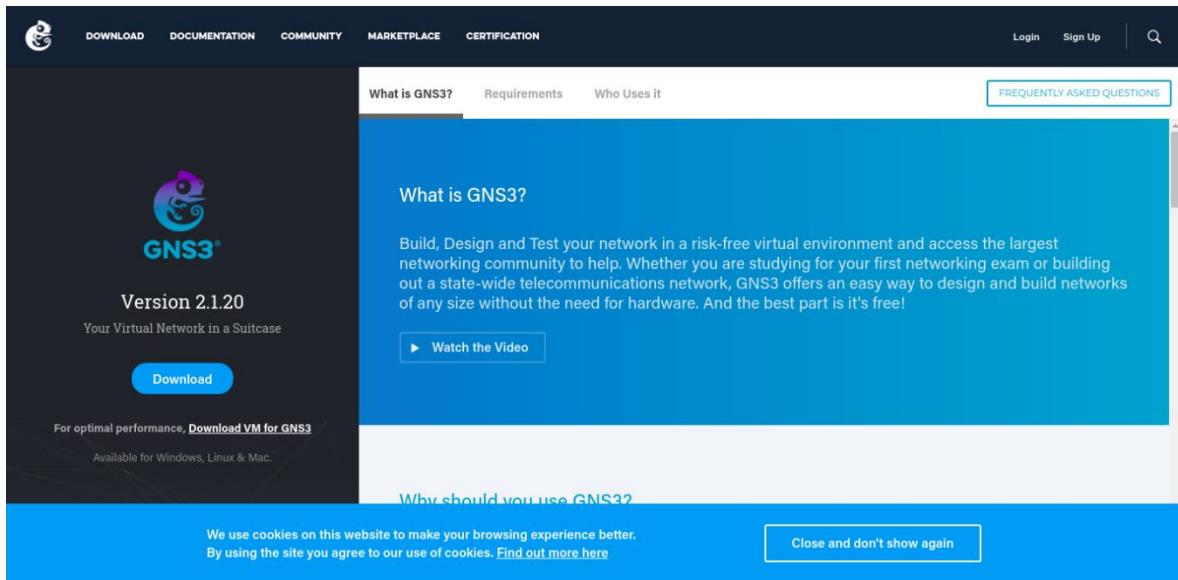


Figura 11: Página principal de la web del simulador gráfico GNS3.

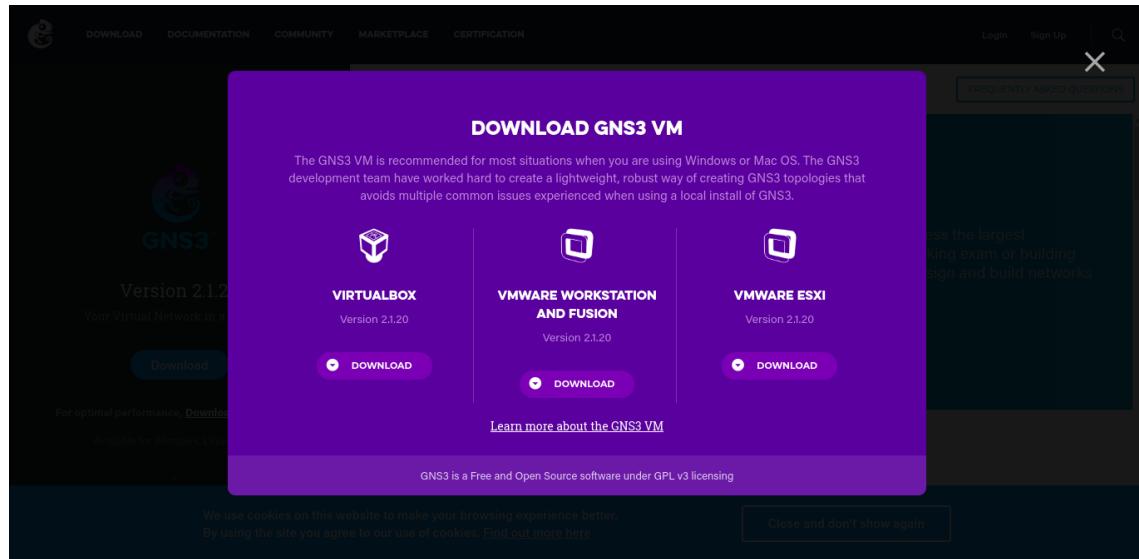


Figura 12: Pestaña de descargas disponibles de GNS3.

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

- Abrimos el programa, mostrándose la interfaz que se puede ver en la Figura 13.

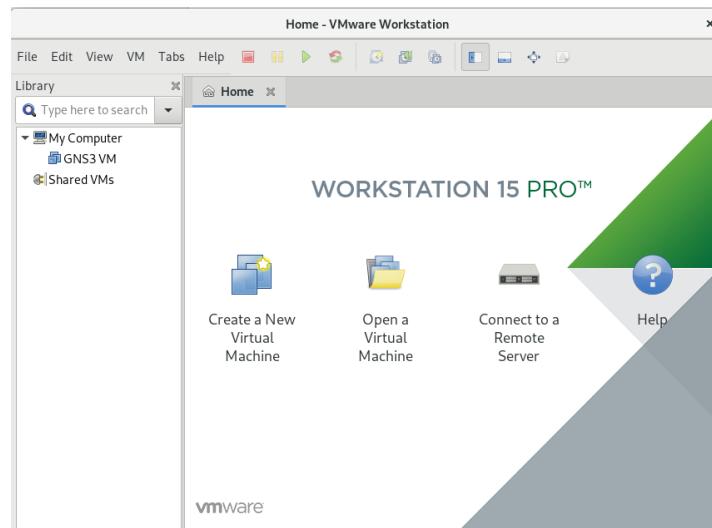


Figura 13: Interfaz de inicio del programa VMware.

- Clicamos en *File* → *Open*
- Seleccionamos la máquina Virtual recién descargada (Figura 14).

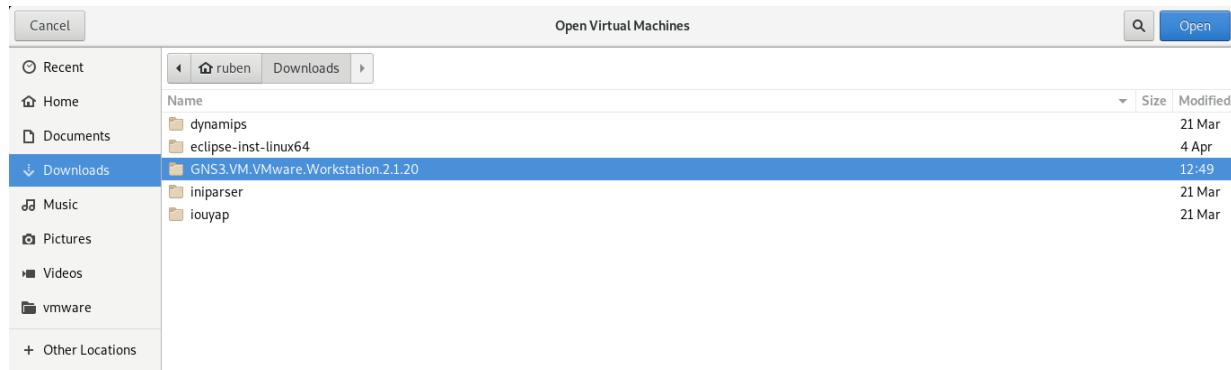


Figura 14: Explorador de archivos con el que se selecciona la máquina virtual.

- Clicamos en *Import* en la siguiente pantalla (Figura 15).



Figura 15: Pestaña de selección de máquina virtual en VMware.

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

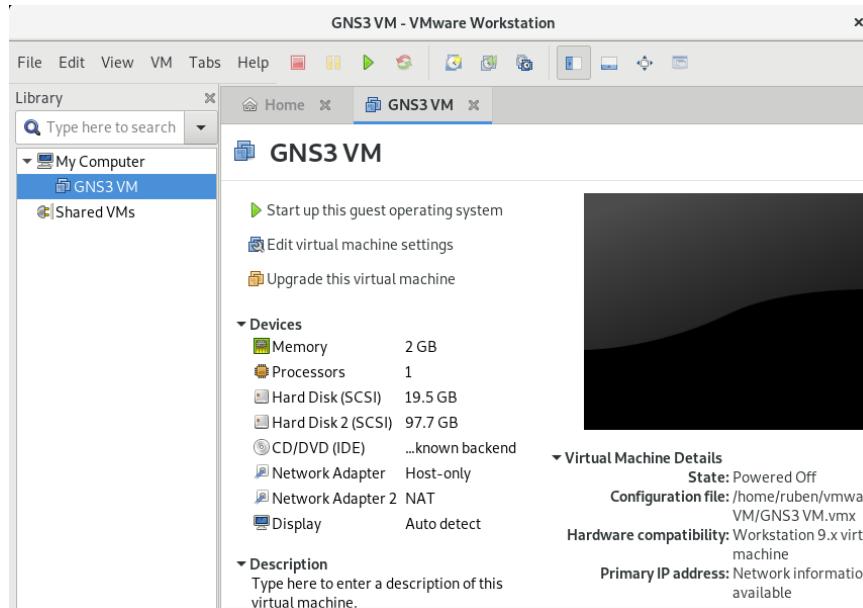


Figura 16: Ventana de información de la máquina virtual instalada.

- Y nos aparecerá algo como lo mostrado en la Figura 17.

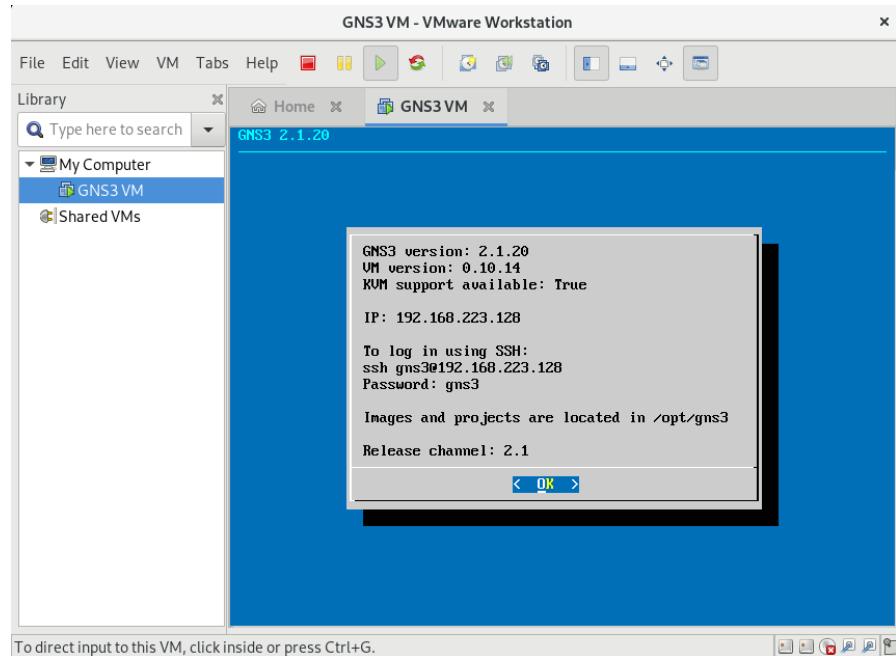


Figura 17: Ventana mostrada al ejecutar la máquina virtual.

3.3 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.

3.3.1 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 mediante *git* (ver Anexo 2 para funcionamiento de *git*) 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`

3.3.2 Instalación en el simulador gráfico 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 y lo arrastramos a la ventana principal (ver Figura 18).

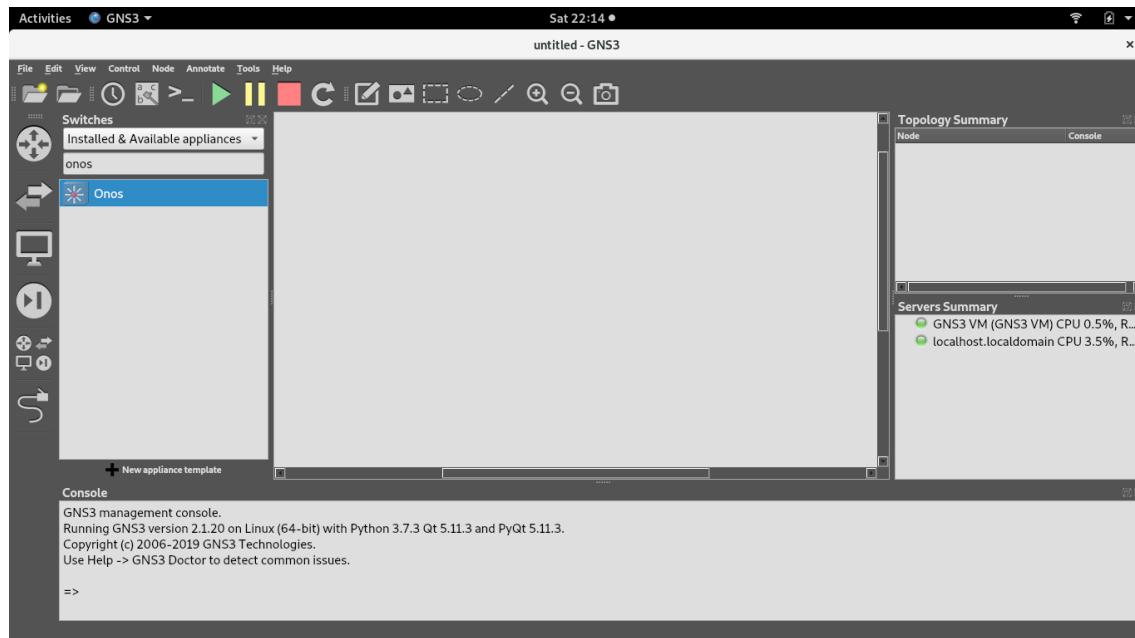


Figura 18: Selección del controlador ONOS para iniciar su instalación en GNS3.

Nos aparece la ventana mostrada en la Figura 19 donde aparece la información del dispositivo, clicamos en *Next*.

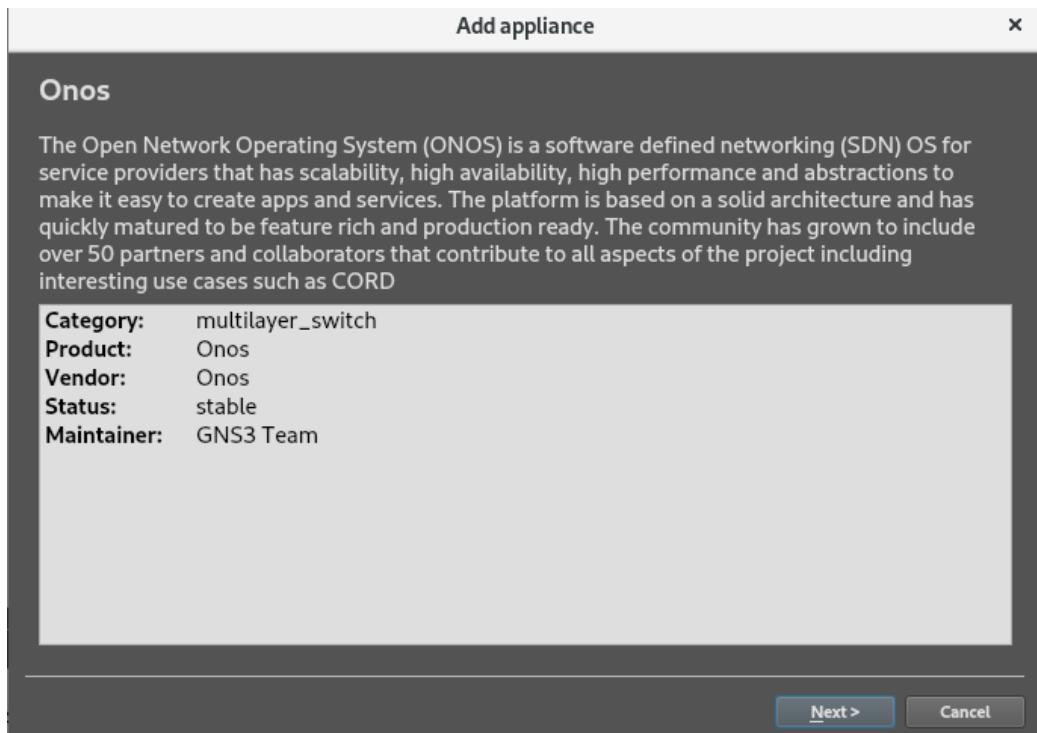


Figura 19: Información sobre el dispositivo que vamos a instalar.

A continuación, aparece una ventana en la que indicamos dónde se va a ejecutar el dispositivo (ver Figura 20).

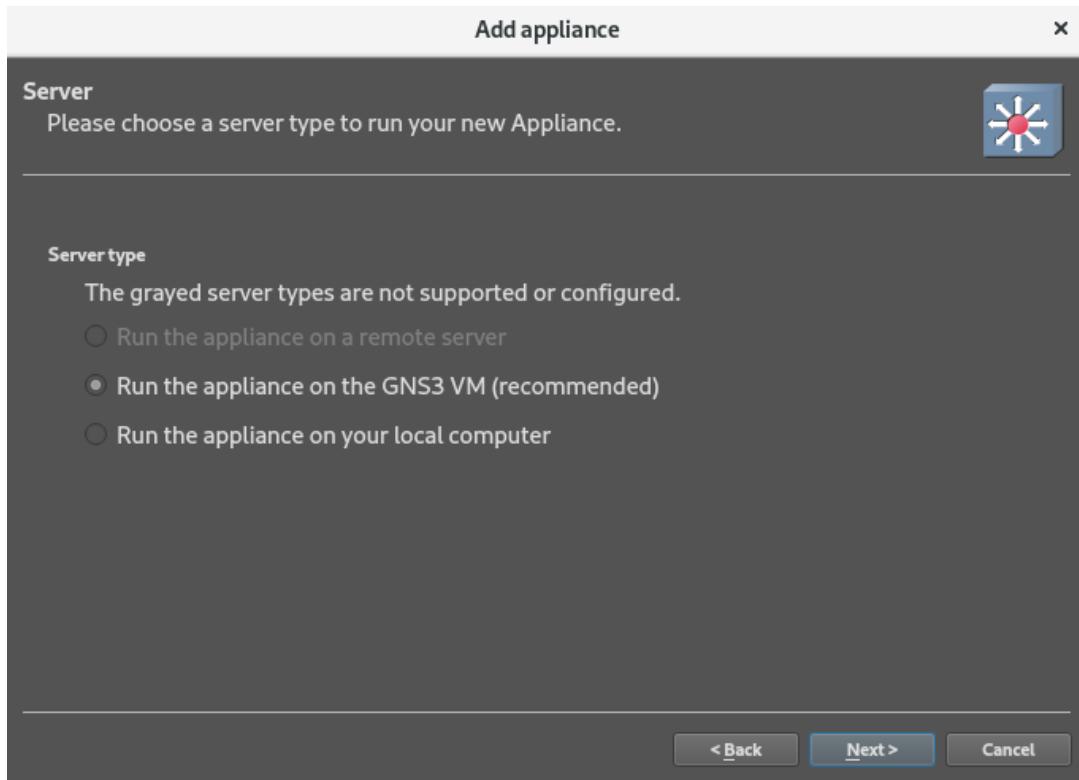


Figura 20: Selección de servidor en el que se va a instalar el dispositivo.

Seleccionamos que se ejecute en la máquina virtual de GNS3 y vamos avanzando en la instalación. Finalmente nos saldrá la ventana de que ONOS está instalado correctamente (ver Figura 22) y ya podremos utilizarlo en el programa.

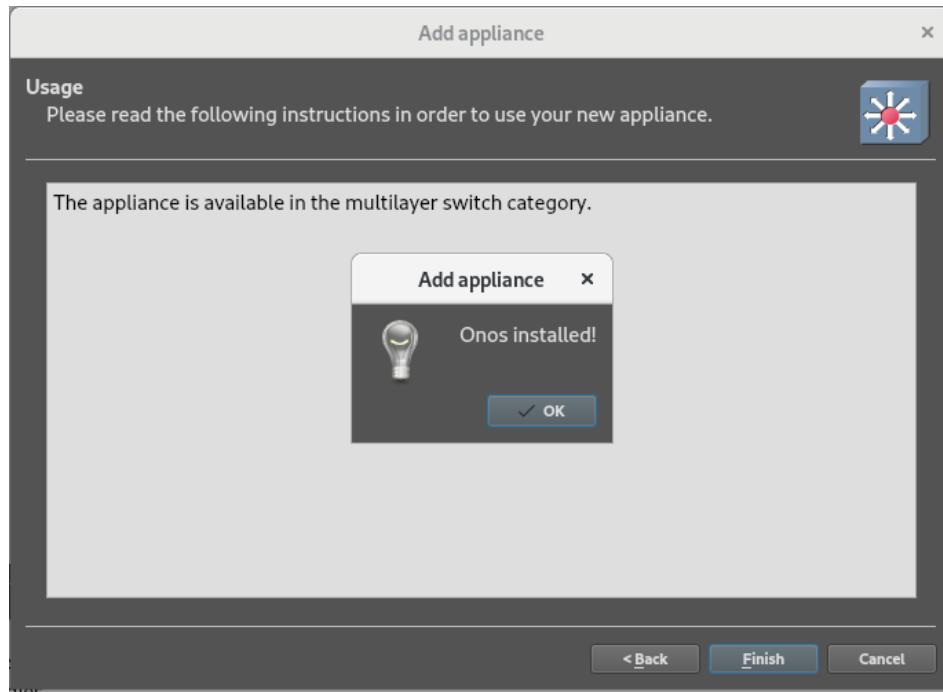


Figura 21: Instalación completada de ONOS.

Como se puede comprobar, el controlador ONOS, por defecto trae únicamente una interfaz, si queremos que tenga más, debemos clicar botón derecho sobre ONOS → *Configure template* y aparecerá lo mostrado en la Figura 22 y simplemente modificamos el parámetro *Adapters* por el número de interfaces que deseemos que tenga el controlador.

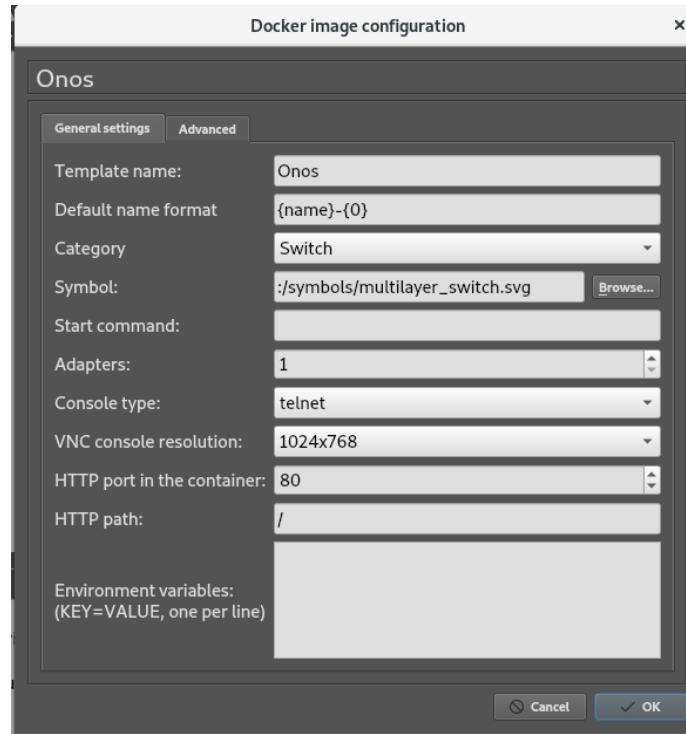


Figura 22: Configuración del dispositivo.

3.4 Instalación de Eclipse

Una vez tenemos GNS3 instalado, el siguiente paso es instalar Eclipse [21] para poder programar las diferentes aplicaciones que queramos mandar al controlador. 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 tal y como se ve en la Figura 23.

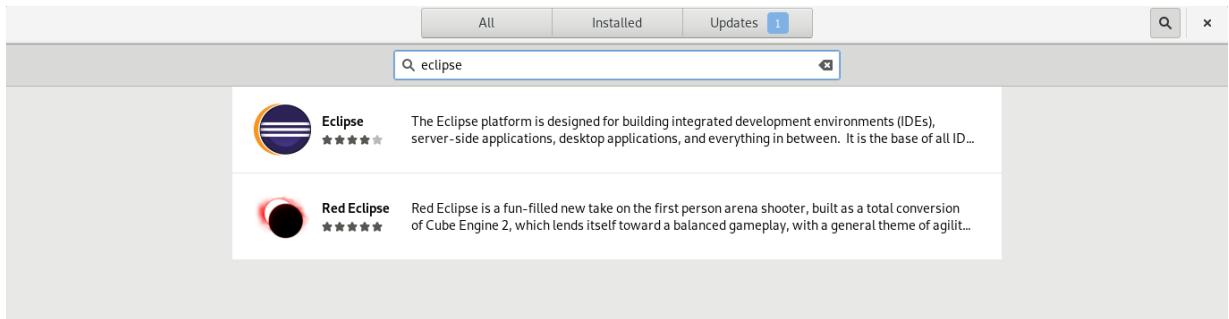


Figura 23: Aplicación software desde la que se instala el programa Eclipse.

Seleccionamos la primera opción que indica 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 Environment*) 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 como se puede observar en la Figura 24.

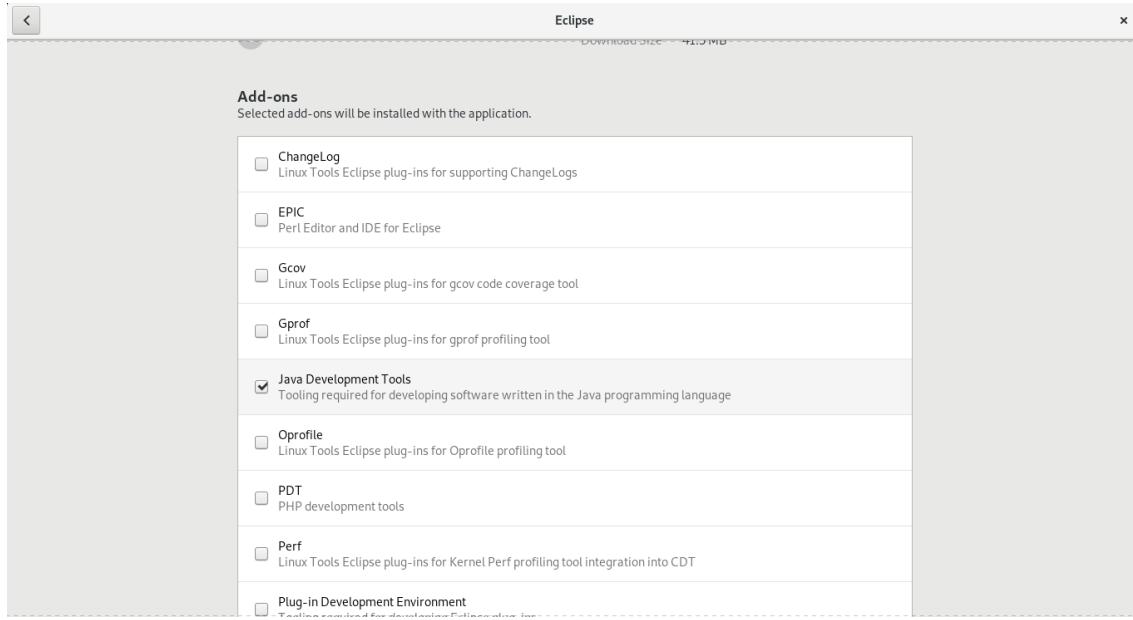


Figura 24: Add-ons necesarios para el funcionamiento del proyecto.

Finalmente, damos a instalar y esperamos a que finalice el proceso.

Capítulo 4: Creando la primera red con GNS3

En este capítulo se va a explicar en detalle la red que se va a simular y sobre la cual (o ligeras modificaciones de la misma) instalaremos las diferentes aplicaciones que irán componiendo el proyecto.

La red, mostrada en la Figura 25, se puede dividir en 2 partes diferenciadas: por un lado, la conexión con el controlador y con el exterior (NAT) y, por otro, los *hosts* conectados por un Open vSwitch que representa la red controlada.

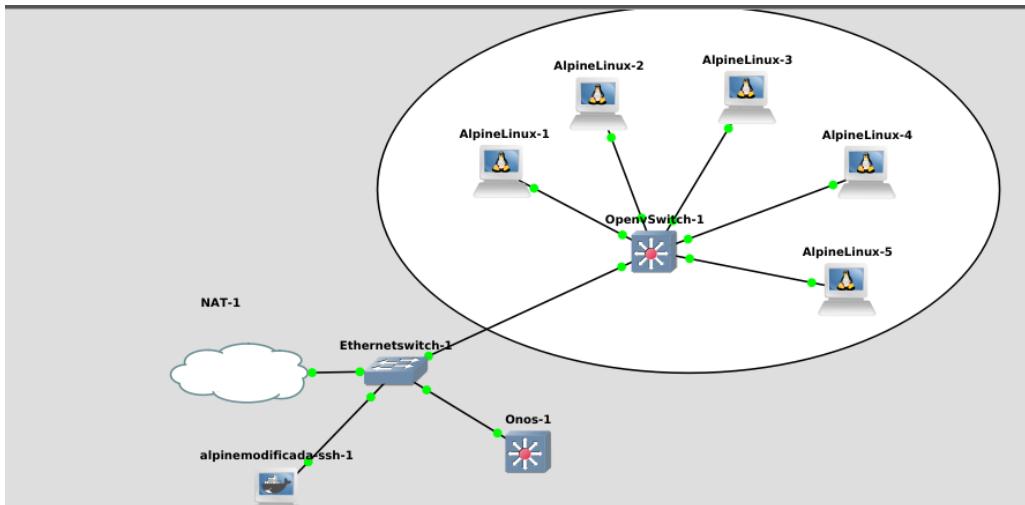


Figura 25: Red básica utilizada durante el transcurso del proyecto.

4.1 Conexión con el controlador y con el exterior

La parte de la red que concierne a este apartado se puede ver en la Figura 26.

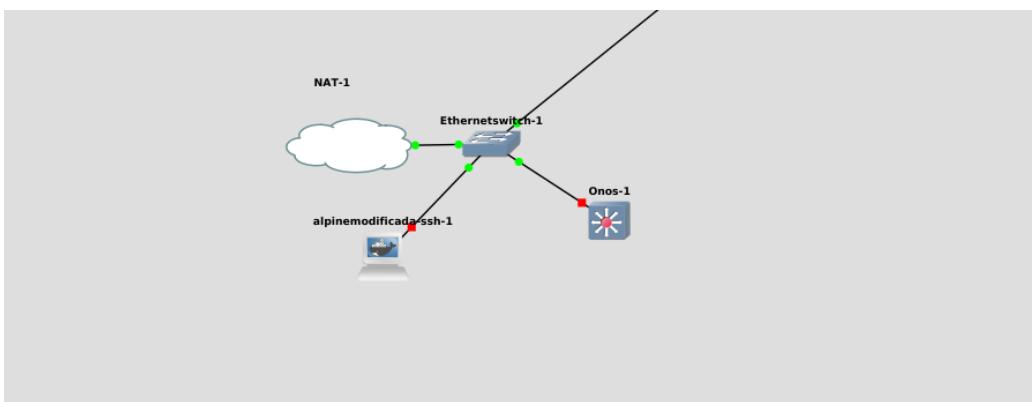


Figura 26: Parte de la red referente al controlador y la conexión con el exterior.

Para poder configurar esta parte de la red, 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 en GNS3 un elemento NAT (*Network Address Translation*) que nos permita acceder desde nuestra red privada interna en el GNS3 (huésped) a través de la red a la que se conecta el ordenador (anfitrión).

Otro paso previo necesario 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 (Figura 27).

```
#  
# This is a sample network config uncomment lines to configure the network  
#  
  
# Static config for eth0  
auto eth0  
iface eth0 inet static  
    address 192.168.122.37  
    netmask 255.255.255.0  
    gateway 192.168.122.1  
    up echo nameserver 192.168.122.1 > /etc/resolv.conf  
  
# DHCP config for eth0  
#auto eth0  
#iface eth0 inet dhcp
```

Figura 27: Fichero de configuración de las interfaces del controlador ONOS.

Hemos elegido arbitrariamente la dirección IP 192.168.122.37, perteneciente a 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. Existen dos formas de acceder a él [22], que veremos en los apartados 4.1.1 y 4.1.2.

4.1.1 Acceso al controlador desde la página web

Una de las formas de acceder al controlador es, desde un *host* conectado a él, a través de la interfaz gráfica de usuario (*Graphical User Interface*, GUI) mediante la URL `http://IP-ONOS:8181/onos/ui` desde un navegador web. Donde IP-ONOS es la dirección IP que hemos definido previamente en el fichero de configuración. Para poder acceder correctamente es necesario que en el controlador esté previamente activada la aplicación `org.onosproject.gui`. Esta aplicación viene se activa por defecto al arrancar el ONOS, pero sin ella sería imposible acceder de esta forma. Las credenciales de acceso son:

- user: onos
- password: rocks.

Lo que vemos una vez que accedemos es lo siguiente (Figura 28).

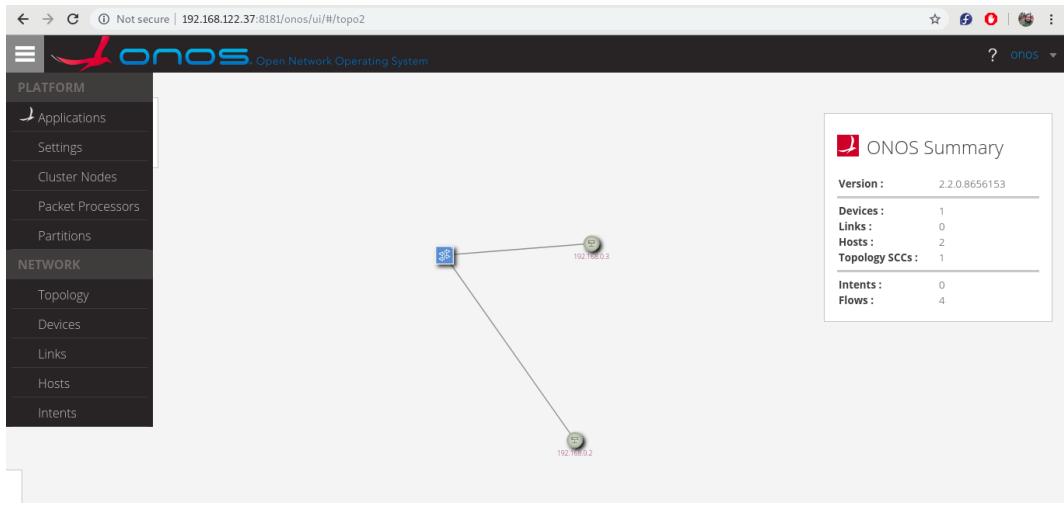


Figura 28: Página principal GUI ONOS.

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* conectando 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, visible en la Figura 29 podemos ver las aplicaciones disponibles, ver en detalle diferentes aspectos de la red, como las aplicaciones instaladas, los dispositivos conectados, los *hosts*...

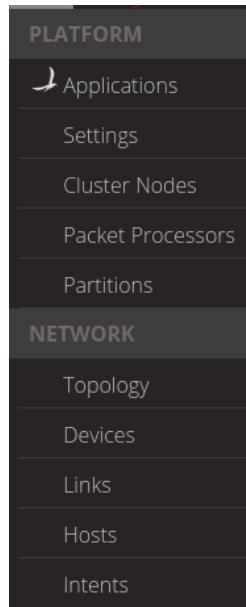


Figura 29: Menú con las opciones de la GUI de ONOS.

Lo primero que vamos a ver son los dispositivos conectados, en la red ejemplo que estamos usando únicamente hay, para ello clicamos en *Devices*. En la Figura 30 podemos ver el resultado.

| FRIENDLY NAME | DEVICE ID | MASTER | PORTS | VENDOR | H/W VERSION | S/W VERSION | PROTOCOL |
|---------------------|---------------------|----------------|-------|--------------|--------------|-------------|----------|
| of:00007e157c24884f | of:00007e157c24884f | 192.168.122.37 | 10 | Nicira, Inc. | Open vSwitch | 2.8.1 | OF_14 |

Figura 30: Vista desde la GUI de ONOS de los dispositivos conectados a la red.

Tal y como se puede ver, se muestran los siguientes detalles:

- ID del dispositivo: Cadena de caracteres que identifica al dispositivo.
- Master: Se refiere a la dirección IP del controlador, en este caso: 192.168.122.37.
- Ports: Número de puertos que tiene el Open vSwitch.
- Vendor: Compañía fabricante del dispositivo.
- H/W versión: Indica que el modelo es un Open vSwitch.
- S/W versión: Indica la versión del Open vSwitch, en este caso la 2.8.1.
- Protocolo: Se refiere a la versión de protocolo OpenFlow utilizado. En este caso se trabaja con la versión 1.4

Haciendo clic en los botones que se encuentran situados en la parte superior derecha, que se pueden ver en detalle en la Figura 31, se pueden ver más detalles acerca del dispositivo.



Figura 31: Botones con las opciones para ver en detalle el comportamiento de los dispositivos.

De izquierda a derecha, los botones tienen el siguiente significado:

- Vista de las entradas de las diferentes tablas de flujos del dispositivo. En este caso, tal y como se puede ver en la Figura 32, se pueden ver 4 entradas. Los campos concretos, como son el *selector* o la prioridad se explicarán en el Capítulo 4, en el momento de desarrollar las aplicaciones.

Flows for Device of:000012baaaa38544 (4 Total)

| STATE | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|-------|---------|----------|---------------|------------|---------------|--------------------------------------|----------|
| Added | 0 | 980 | 40000 | 0 | ETH_TYPE:bddp | lmm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 0 | 980 | 40000 | 0 | ETH_TYPE:lldp | lmm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 3 | 980 | 5 | 0 | ETH_TYPE:ipv4 | lmm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 4 | 980 | 40000 | 0 | ETH_TYPE:arp | lmm[OUTPUT:CONTROLLER], cleared:true | *core |

Figura 32: Pestaña de análisis de los flujos del dispositivo.

- Estadísticas de todos los puertos de los que consta el dispositivo en cuestión, entre las que se encuentran los paquetes enviados, o los descartados (Figura 33).

Ports for Device of:000012baaaa38544 (9 Total)

| PORT ID | PKTS RECEIVED | PKTS SENT | BYTES RECEIVED | BYTES SENT | PKTS RX DROPPED | PKTS TX DROPPED | DURATION (SEC) |
|---------|---------------|-----------|----------------|------------|-----------------|-----------------|----------------|
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 1384 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1384 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1384 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1384 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1384 |
| 4 | 2 | 874 | 140 | 118,160 | 0 | 0 | 1384 |
| 3 | 2 | 874 | 140 | 118,160 | 0 | 0 | 1384 |
| 2 | 19 | 881 | 1,614 | 118,846 | 2 | 0 | 1384 |
| 1 | 17 | 883 | 1,434 | 119,118 | 0 | 0 | 1384 |

Figura 33: Pestaña de análisis de los puertos del dispositivo.

- Tabla de grupos. Idéntica la tabla de flujos, pero, en este caso, contiene los datos acerca de los grupos creados.
- Vista de los *meters*. En este TFG no se van a usar, por tanto, no se hace mayor hincapié sobre ellos.
- Configuración del *pipeline*: Al igual que en el caso anterior, no se va a trabajar sobre ello.

Finalmente, si clicamos desde el menú de la Figura 29 en *Hosts*, vemos la información referente a los hosts detectados en la red.

| FRIENDLY NAME | HOST ID | MAC ADDRESS | VLAN ID | CONFIGURED | IP ADDRESSES | LOCATION |
|---------------|------------------------|-------------------|---------|------------|--------------|-----------------------|
| 192.168.0.2 | 00:00:00:00:00:02/None | 00:00:00:00:00:02 | None | false | 192.168.0.2 | of:000012baaaa38544/2 |
| 192.168.0.1 | 00:00:00:00:00:01/None | 00:00:00:00:00:01 | None | false | 192.168.0.1 | of:000012baaaa38544/1 |

Figura 34: Vista desde la GUI de ONOS de los hosts conectados a la red.

Los datos más importantes mostrados son:

- Host ID: El *host* ID está formado por la direcciónMAC/VLAN a la que pertenece.
- Location: La localización del *host* está formada por el dispositivo al que está conectado/Puerto al que está conectado.

4.1.2 Acceso al controlador desde línea de comandos

La segunda opción para acceder al controlador, y la que más usaremos, es mediante CLI [23]. 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 acceder a *Docker*, accederemos a un terminal desde nuestro ordenador y realizaremos un *ssh* a la dirección IP que tiene la máquina virtual de GNS3. Esta dirección se puede ver en la ventana mostrada en la Figura 17.

```

gns3@gns3vm: ~
File Edit View Search Terminal Help
[ruben@localhost ~]$ ssh gns3@192.168.223.128
gns3@192.168.223.128's password:
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 4.4.0-31-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
New release '16.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Jun 27 15:11:23 2019
INFO: /dev/kvm exists
KVM acceleration can be used

```

Figura 35: Comandos de acceso a la máquina virtual de GNS3.

La contraseña de acceso en este caso es: *gns3*

A continuación, nos aparece la ventana que se puede ver en la Figura 36.

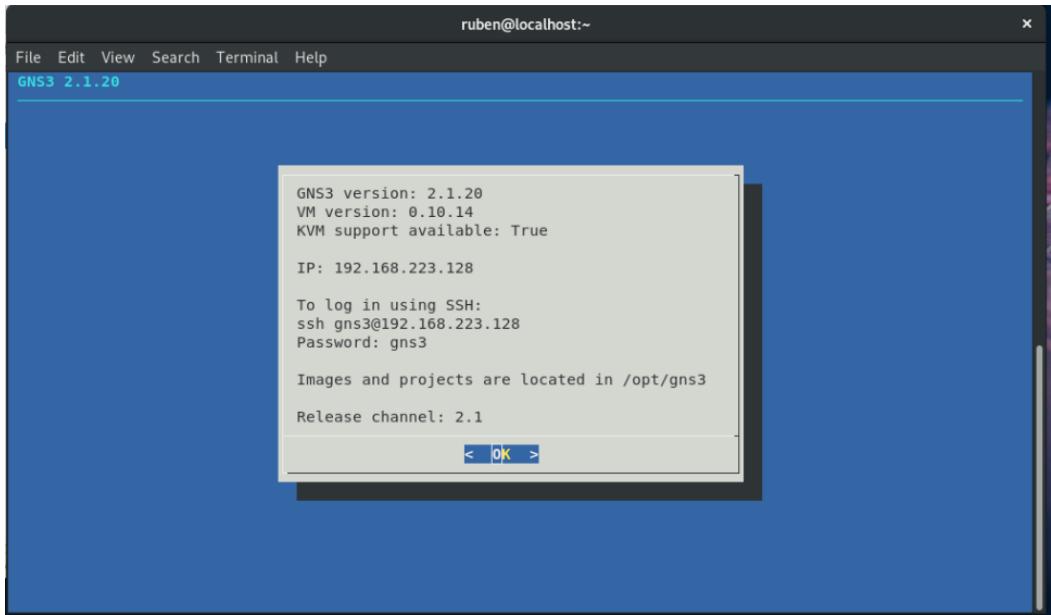


Figura 36: Acceso a la máquina virtual de GNS3.

Damos clic a *OK* y en la siguiente ventana abrimos un *shell*.

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

- `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. Esto lo que implica es que estamos modificando la imagen original, añadiéndole los paquetes que necesitaremos para el resto del proyecto:

- `apk add openssh wget nano curl netcat-openbsd`

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`

El siguiente paso es incluir la nueva imagen modificada en nuestra red GNS3. Para ello, realizamos los siguientes pasos:

- Desde el GNS3, clicamos en *Edit* → *Preferences* → *Docker Containers* obteniendo el resultado mostrado en la Figura 37.

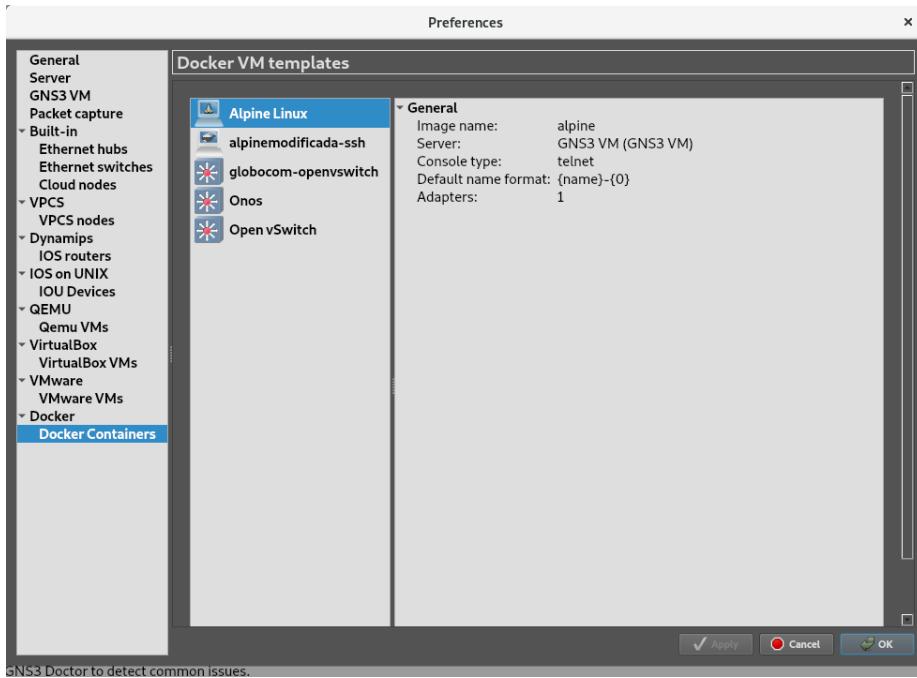


Figura 37: Muestra de los contenedores de Docker instalados.

- En esta ventana se pueden ver los diferentes contenedores *Docker* que están ya instalados. Si queremos añadir uno nuevo, bajamos el *scroll* y clicamos en *New*.
- Seleccionamos que se ejecuta sobre la máquina virtual, tal y como se ve en la Figura 38.

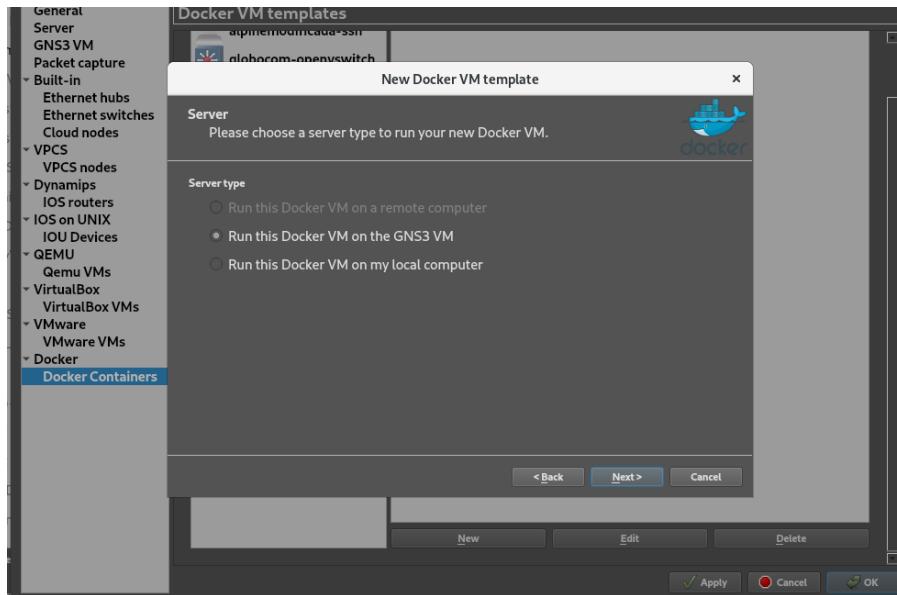


Figura 38: Selección de la máquina virtual al añadir un nuevo contenedor Docker.

- Clicamos en *Existing image* y seleccionamos la imagen modificada. En nuestro caso será la que hemos denominado como *alpinemodificada-ssh*. En la Figura 39 se puede ver lo descrito.

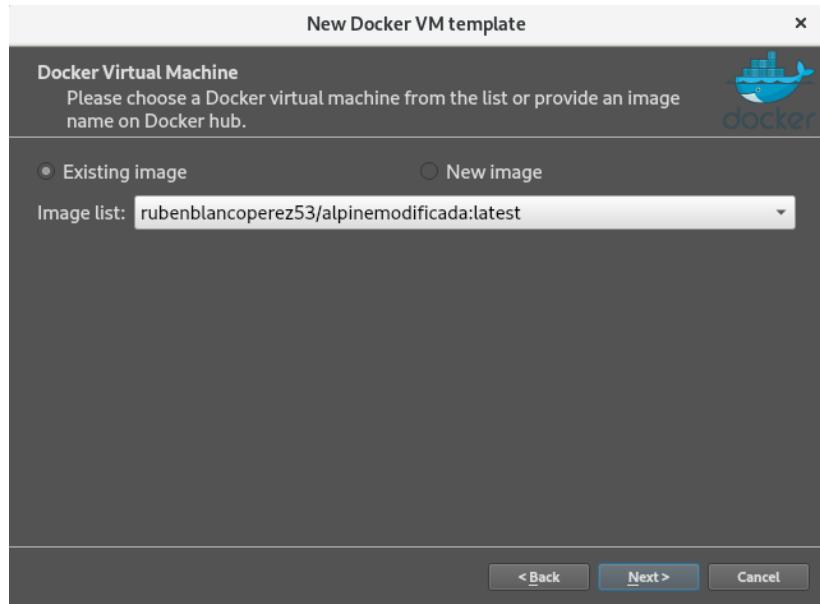


Figura 39: Selección del contenedor Docker.

- Finalmente, damos a *Next* hasta acabar el proceso.

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`

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 *nombreAplicacion* para que se inicie su funcionamiento.
- `app deactivate nombreAplicacion`: Desactiva la aplicación *nombreAplicacion* en el controlador para que se detenga su funcionamiento.

4.2 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 requisitos de dichas aplicaciones.

Para las primeras aplicaciones tendrá la topología mostrada en la Figura 40, que posteriormente se irá modificando en función de los requisitos de la aplicación desarrollada.

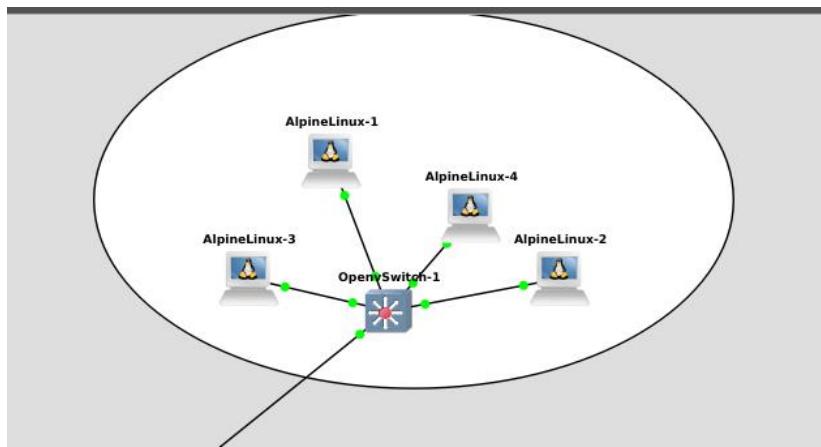


Figura 40: Parte de la red referida al Open vSwitch y los hosts.

Como podemos ver, tenemos 4 *hosts* conectados a un Open vSwitch que recibirá información del controlador para conocer cómo 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 forma mostrada en la Figura 41.

The screenshot shows a terminal window titled "AlpineLinux-1 interfaces" containing the following configuration file content:

```
#  
# This is a sample network config uncomment lines to configure the network  
#  
  
# Static config for eth0  
auto eth0  
iface eth0 inet static  
    address 192.168.0.1  
    netmask 255.255.255.0  
    hwaddress ether 00:00:00:00:00:01  
    # gateway 192.168.0.254  
    # up echo nameserver 192.168.0.254> /etc/resolv.conf  
  
# DHCP config for eth0  
# auto eth0  
# iface eth0 inet dhcp
```

Figura 41: Fichero de configuración de los hosts de la red.

Tal y como se puede ver, se le ha asignado una dirección IP y una dirección MAC estática (en orden ascendente por cada *host* que utilicemos, es decir, el *host* numerado por 1 tendrá la MAC finalizada en :01, el *host* 2 tendrá la MAC finalizada en :02 y así sucesivamente).

Finalmente, para configurar el Open vSwitch es necesario indicarle quién es el controlador. Como la dirección IP de éste es estática, este proceso sólo tendremos que realizarlo 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 [24].

Creamos el bridge br0:

- `ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev`

Añadimos todas las interfaces 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* [23]:

- `ovs-vsctl set-manager ptcp:6640`
- `ovs-vsctl set-controller br0 tcp:IP-ONOS:6633`

En la Figura 42 se pueden observar los comandos ejecutados sobre el Open vSwitch.

```

alpinemodificada-ssh-2
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^]'.

globocom@openvswitch-1:~# ovs-vsctl show
b51c85de-3968-463b-8943-2c69d0b3a075
globocom@openvswitch-1:~# ovs-vsctl add-br br0 -- set bridge br0 datapath_type=n
etdev
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth1
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth2
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth3
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth4
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth5
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth6
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth7
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth8
globocom@openvswitch-1:~# ovs-vsctl add-port br0 eth9
globocom@openvswitch-1:~# ovs-vsctl set-manager ptcp:6640
globocom@openvswitch-1:~# ovs-vsctl set-controller br0 tcp:192.168.122.37:6633
globocom@openvswitch-1:~#

```

Figura 42: Comandos de configuración iniciales del Open vSwitch.

En la Figura 43 se muestra la salida del comando, para comprobar cómo ha quedado la configuración del *switch*, cuando ejecutamos el siguiente comando:

- ovs-vsctl show

```
globocom-openvswitch-1:~# ovs-vsctl show
06df1dal-fda0-4b89-946c-503dfbb8bc89
  Manager "ptcp:6640"
  Bridge "br0"
    Controller "tcp:192.168.122.37:6633"
      is_connected: true
    Port "eth5"
      Interface "eth6"
    Port "eth2"
      Interface "eth2"
    Port "br0"
      Interface "br0"
        type: internal
    Port "eth7"
      Interface "eth7"
    Port "eth9"
      Interface "eth9"
    Port "eth5"
      Interface "eth5"
    Port "eth1"
      Interface "eth1"
    Port "eth3"
      Interface "eth3"
    Port "eth4"
      Interface "eth4"
    Port "eth8"
      Interface "eth8"
globocomm-openvswitch-1:~# ■
```

Figura 43: Estado del Open vSwitch tras configurarlo.

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

Capítulo 5: Aplicaciones desarrolladas

En este capítulo se van a detallar las aplicaciones que se han realizado, así como las utilidades que tienen. Estas aplicaciones, se han realizado de la forma más sencilla posible y, vienen acompañadas de comentarios y guías presentes tanto en este TFG, como en el propio código, así como una batería de pruebas, permitiendo al lector su comprensión, y la posibilidad de crear otras aplicaciones similares que pueda tener en mente. Todos estos códigos están incluidos en un repositorio de GitHub públicamente accesible [25].

Cabe destacar también que las explicaciones referentes al diferente *software* utilizado, tanto ONOS, Open vSwitch... están explicados a lo largo de la presente memoria facilitando nuevamente la comprensión.

5.1 Indicaciones previas

Antes de entrar en detalles sobre cada una de las aplicaciones desarrolladas, 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 versión`
`org.onosproject.nombreApp`

donde:

- *nombreApp* será el nombre que decidamos dar a nuestra aplicación.
- *versión* es la fase en la que estemos de la aplicación. En caso de ser una aplicación que estamos empezando, lo normal suele ser poner la versión 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 códigos 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. Es recomendable modificar el nombre de esta clase a uno más ilustrativo acerca del funcionamiento de la aplicación.

Nos aparecerá por defecto un código similar al de la Figura 44.

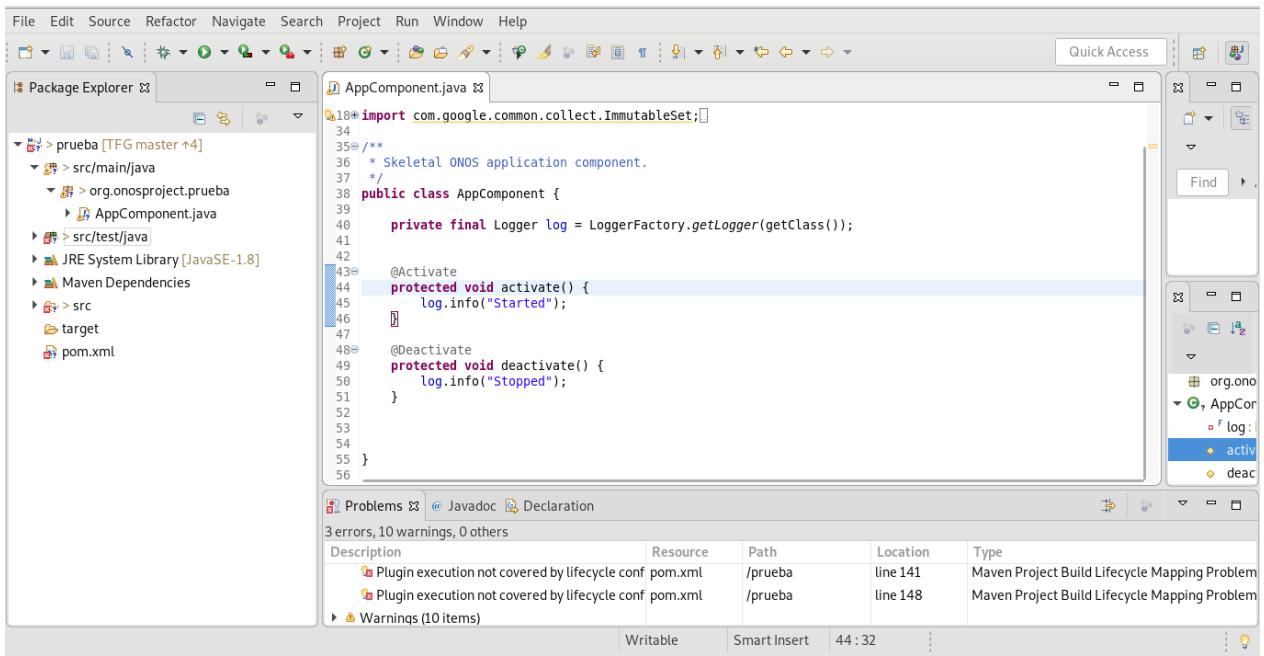


Figura 44: Vista desde el Eclipse de una aplicación recién importada.

En él se destacan los 2 métodos principales: *activate* y *deactivate*. El método *activate* será aquél 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 envímos al controlador con el comando:

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

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

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:

Este comando automáticamente desactiva la aplicación del ONOS y la vuelve a enviar al controlador. Al igual que en el caso anterior, si mantenemos la exclamación la activa automáticamente, mientras que si la suprimimos simplemente la envía sin activarla.

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

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

5.2 Aplicación *severalping*

Esta aplicación es una adaptación de la aplicación *oneping* [26], cuyo objetivo principal es limitar el número de *pings* que se pueden enviar entre 2 *hosts* cualesquiera.

5.2.1 Motivación y explicación teórica.

En este apartado se va a detallar la motivación por la cual se ha considerado realizar esta aplicación, así como una explicación teórica, en caso de ser necesario, que permita clarificar la explicación referente al código programado.

Las motivaciones referentes a la aplicación *severalping* son fundamentalmente dos. En primer lugar realizar un primer acercamiento al protocolo OpenFlow modificando una aplicación existente y, en segundo lugar, el hecho de limitar el número máximo de *pings* aumenta la seguridad en el sentido de que se evitan ataques por inundación de tráfico, sobre todo si al *ping* se le incluye la opción *-s* que permite aumentar el tamaño del mismo consiguiendo que los hosts que desean enviar otro tipo de tráfico se encuentren sin ancho de banda disponible ya que el mismo está siendo ocupado por los *pings* indeseados.

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 envía 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 ejecutar el comando en GNU/Linux es:

- ping [opciones] IP_HOST

Y las opciones más utilizadas que usaremos son:

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

5.2.2 Componentes de la aplicación

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.

5.2.2.1 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 al controlador los paquetes que cumplan un determinado selector:

```
packetService.requestPackets(intercept, PacketPriority.CONTROL, appId,
Optional.empty());
```

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:

```
private final TrafficSelector intercept =
DefaultTrafficSelector.builder()
.matchEthType(Ethernet.TYPE_IPV4)
.matchIPProtocol(IPv4.PROTOCOL_ICMP)
.matchIcmpType(ICMP.TYPE_ECHO_REQUEST)
.build();
```

Como se puede ver, es necesario ir “pasando” por todos los protocolos de capas inferiores, en este caso únicamente IPv4 hasta poder trabajar con las cabeceras del protocolo de capa superior, en este caso ICMP. Por tanto, el selector recoge todos los paquetes IPv4 que pasen por el Open vSwitch que pertenezcan al protocolo ICMP y, finalmente sean del tipo ECHO *REQUEST*.

5.2.2.2 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 sólo los que nos interesan, esto se consigue procesando el paquete que se está recibiendo en cada momento. Para ello definimos el método *isIcmpPing* 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:

1. Se aumenta en 1 el valor de *pings* enviados en el *hashMap*.
2. 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(TIME_BAN)
    .add());

```

Fundamentalmente una regla de flujo se compone de 2 elementos:

1. 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*).
2. 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.

5.2.2.3 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 del *switch* (porque ha pasado su tiempo de vida determinado). De esta forma, se crea un mensaje en el *log* del controlador que nos informa de que el enlace entre los 2 *hosts* baneados vuelve a estar disponible para mandar otra vez *pings*.

```

private final FlowRuleListener flowListener =new
InternalFlowListener();

flowRuleService.addListener(flowListener);

```

En este caso creamos un método que ejecuta a su vez el método *evento* cuya finalidad es reaccionar a un evento concreto. En este caso será el siguiente:

```
event.type() == RULE_REMOVED && flowRule.appId() == appId.id()
```

Es decir, que se haya eliminado una regla de flujo de nuestra aplicación. Cuando ocurre esto, obtenemos algunos datos de la regla eliminada como son las direcciones MAC origen y destino que estaban baneadas y lo imprimimos.

Cabe destacar que la finalidad de este método es meramente informativa, para conocer cuando la regla se ha eliminado y a que efectos prácticos de la aplicación no es necesario.

5.2.2.4 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 [27]. 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 más *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 la siguiente:

```
@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* con el fin de poder realizar modificaciones desde la consola de ONOS en tiempo real.

Cuando hayamos declarado los parámetros declaramos un método que incluye la anotación `@Modified` cuya implementación básica es la siguiente:

```
Dictionary<?, ?> properties = context.getProperties();  
  
String s = Tools.get(properties, "MAX_PINGS");  
MAX_PINGS = Strings.isNullOrEmpty(s) ? MAX_PINGS_DEFAULT :  
Integer.parseInt(s.trim());
```

Este código se encarga 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

En el apartado 5.2.3.4 se puede ver algún ejemplo de uso de estos comandos descritos.

5.2.3 Banco de pruebas

En este apartado se van a realizar diferentes pruebas para comprobar el funcionamiento de la aplicación, así 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*.

```

alpinemodificada-ssh-1
x

Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^]'.

/ # ssh -p 8101 onos@192.168.122.37
Password authentication
Password:
Welcome to Open Network Operating System (ONOS)!

Documentation: wiki.onosproject.org
Tutorials: tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

onos@root >

```

Figura 45: Interfaz para la gestión del controlador.

A continuación, activamos las aplicaciones necesarias tal y como se ve en la Figura 46.

```

alpinemodificada-ssh-1
x

onos@root > app activate org.onosproject.drivers.ovsdb
Activated org.onosproject.drivers.ovsdb
onos@root > app activate org.onosproject.ovsdb
Activated org.onosproject.ovsdb
onos@root > app activate org.onosproject.openflow
Activated org.onosproject.openflow
onos@root > app activate org.onosproject.fwd
Activated org.onosproject.fwd
onos@root > app activate org.onosproject.severalping
Activated org.onosproject.severalping
onos@root >

```

Figura 46: Aplicaciones necesarias para activar la aplicación.

Y, comprobamos, tal y como se ve en la Figura 47, en el controlador que se han activado correctamente.

```

Onos-1
ge=null]
16:53:40.421 INFO [ApplicationManager] Application org.onosproject.severalping
has been installed
16:53:40.432 INFO [FeaturesServiceImpl] Adding features: severalping/[1.0.0.SNA
PSHOT,1.0.0,SNAPSHOT]
16:53:41.332 INFO [FeaturesServiceImpl] Changes to perform:
16:53:41.338 INFO [FeaturesServiceImpl] Region: root
16:53:41.339 INFO [FeaturesServiceImpl] Bundles to install:
16:53:41.339 INFO [FeaturesServiceImpl] mvn:org.onosproject/severalping/1
.0-SNAPSHOT
16:53:41.341 INFO [FeaturesServiceImpl] Installing bundles:
16:53:41.341 INFO [FeaturesServiceImpl] mvn:org.onosproject/severalping/1.0-S
NAPSHOT
16:53:41.371 INFO [FeaturesServiceImpl] Starting bundles:
16:53:41.372 INFO [FeaturesServiceImpl] org.onosproject.severalping/1.0.0.SNA
PSHOT
16:53:41.412 INFO [AppComponent] Propiedades cambiadas a: 7 pings y 60 segundos
16:53:41.414 INFO [AppComponent] Activada aplicación severalpings
16:53:41.418 INFO [FeaturesServiceImpl] Done.
16:53:41.423 INFO [ApplicationManager] Application org.onosproject.severalping
has been activated
16:53:41.668 INFO [AppComponent] Propiedades cambiadas a: 7 pings y 60 segundos

```

Figura 47: Vista desde el controlador de la activación de la aplicación severalping.

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.

5.2.3.1 Envío 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 Figura 48.

```

AlpineLinux-1
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^]'.
AlpineLinux-1 console is now available... Press RETURN to get started.
/ #
/ # ping -c 10 192.168.0.4

```

Figura 48: Envío de 10 pings.

Ejecutamos el comando, pudiendo ver su salida en la Figura 49.

```

AlpineLinux-1
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^['.
AlpineLinux-1 console is now available... Press RETURN to get started.
/ #
/ # ping -c 10 192.168.0.4
PING 192.168.0.4 (192.168.0.4): 56 data bytes
64 bytes from 192.168.0.4: seq=0 ttl=64 time=48.051 ms
64 bytes from 192.168.0.4: seq=1 ttl=64 time=24.329 ms
64 bytes from 192.168.0.4: seq=2 ttl=64 time=22.130 ms
64 bytes from 192.168.0.4: seq=3 ttl=64 time=27.064 ms
64 bytes from 192.168.0.4: seq=4 ttl=64 time=20.148 ms
64 bytes from 192.168.0.4: seq=5 ttl=64 time=21.523 ms
64 bytes from 192.168.0.4: seq=6 ttl=64 time=4.557 ms

--- 192.168.0.4 ping statistics ---
10 packets transmitted, 7 packets received, 30% packet loss
round-trip min/avg/max = 4.557/23.971/48.051 ms
/ #

```

Figura 49: Resultado del envío de los 10 pings.

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.

```

Onos-1
15:43:32.371 INFO [FeaturesServiceImpl] Done.
15:43:32.378 INFO [ApplicationManager] Application org.onosproject.severalping has been deactivated
15:43:38.246 INFO [ApplicationManager] Application org.onosproject.severalping has been uninstalled
15:43:38.300 INFO [ApplicationManager] Application org.onosproject.severalping has been installed
15:43:38.308 INFO [FeaturesServiceImpl] Adding features: severalping/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
15:43:38.977 INFO [FeaturesServiceImpl] Changes to perform:
15:43:38.978 INFO [FeaturesServiceImpl]   Region: root
15:43:38.979 INFO [FeaturesServiceImpl]     Bundles to install:
15:43:38.979 INFO [FeaturesServiceImpl]       mvn:org.onosproject/severalping/1.0-SNAPSHOT
15:43:38.982 INFO [FeaturesServiceImpl]     Installing bundles:
15:43:38.984 INFO [FeaturesServiceImpl]       mvn:org.onosproject/severalping/1.0-SNAPSHOT
15:43:39.014 INFO [FeaturesServiceImpl]     Starting bundles:
15:43:39.014 INFO [FeaturesServiceImpl]       org.onosproject.severalping/1.0.0.SNAPSHOT
15:43:39.039 INFO [AppComponent] Propiedades cambiadas a: 7 pings y 60 segundos
15:43:39.043 INFO [AppComponent] Activada aplicacion severalpings
15:43:39.054 INFO [FeaturesServiceImpl] Done.
15:43:39.057 INFO [ApplicationManager] Application org.onosproject.severalping has been activated
15:43:39.290 INFO [AppComponent] Propiedades cambiadas a: 7 pings y 60 segundos
15:44:29.789 INFO [AppComponent] Ping 1/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:30.762 INFO [AppComponent] Ping 2/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:31.763 INFO [AppComponent] Ping 3/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:32.765 INFO [AppComponent] Ping 4/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:33.765 INFO [AppComponent] Ping 5/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:34.770 INFO [AppComponent] Ping 6/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:35.767 INFO [AppComponent] Ping 7/7 received from 00:00:00:00:01 to 00:00:00:00:04
15:44:36.771 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00:00:01 to 00:00:00:00:04 h
as already been received, 60 seconds ban
15:44:37.775 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00:00:01 to 00:00:00:00:04 h
as already been received, 60 seconds ban
15:44:38.771 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00:00:01 to 00:00:00:00:04 h
as already been received, 60 seconds ban
15:45:40.722 WARN [AppComponent] Re-enabled ping from 00:00:00:00:01 to 00:00:00:00:04

```

Figura 50: Vista desde el controlador del envío de los pings.

Vemos, en la Figura 50, como se notifica la llegada 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.

También se observa, viendo en la parte izquierda de la imagen los tiempos en los que se mandan los mensajes, 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. En la Figura 51 se ven los flujos cuando se ha activado la aplicación, mientras que en la Figura 52 se ven los *flows* en el momento en el que se ha baneado el enlace.

```
alpinemodificada-ssh-1
onos@root > flows
deviceId=of:000008697641ba245, flowRuleCount=5
    id=100001d5cae13, state=ADDED, bytes=0, packets=0, duration=10, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4, IP_PROTO:1, ICMPV4_TYPE:8], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=10000248410ea, state=ADDED, bytes=0, packets=0, duration=1824, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=100008895df10, state=ADDED, bytes=0, packets=0, duration=1824, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=1000093dda8d3, state=ADDED, bytes=480, packets=8, duration=1824, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=100003ef4daea, state=ADDED, bytes=196, packets=2, duration=1824, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
onos@root >
```

Figura 51: Vista de los flujos al activar la aplicación.

```
alpinemodificada-ssh-1
onos@root > flows
deviceId=of:000008697641ba245, flowRuleCount=7
    id=100001d5cae13, state=ADDED, bytes=980, packets=10, duration=725, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4, IP_PROTO:1, ICMPV4_TYPE:8], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=10000248410ea, state=ADDED, bytes=0, packets=0, duration=1279, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=1000093dda8d3, state=ADDED, bytes=480, packets=8, duration=1279, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=c00000d64c818a, state=ADDED, bytes=0, packets=0, duration=4, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.severalping, selector=[ETH_DST:00:00:00:00:00:04, ETH_SRC:00:00:00:00:00:01, ETH_TYPE:ipv4, IP_PROTO:1, ICMPV4_TYPE:8], treatment=DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[], cleared=false, StatTrigger=null, metadata=null}
    id=3100003780a5da, state=ADDED, bytes=588, packets=6, duration=13, liveType=UNKNOWN, priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:3, ETH_DST:00:00:00:00:01, ETH_SRC:00:00:00:00:04], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:11], deferred[], transition=None, meter[], cleared=false, StatTrigger=null, metadata=null}
    id=100003ef4daea, state=ADDED, bytes=196, packets=2, duration=1279, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
onos@root >
```

Figura 52: Vista de los flujos al banearse el enlace.

Se puede ver que se han añadido 2 flujos respecto al momento en el que se activó la aplicación:

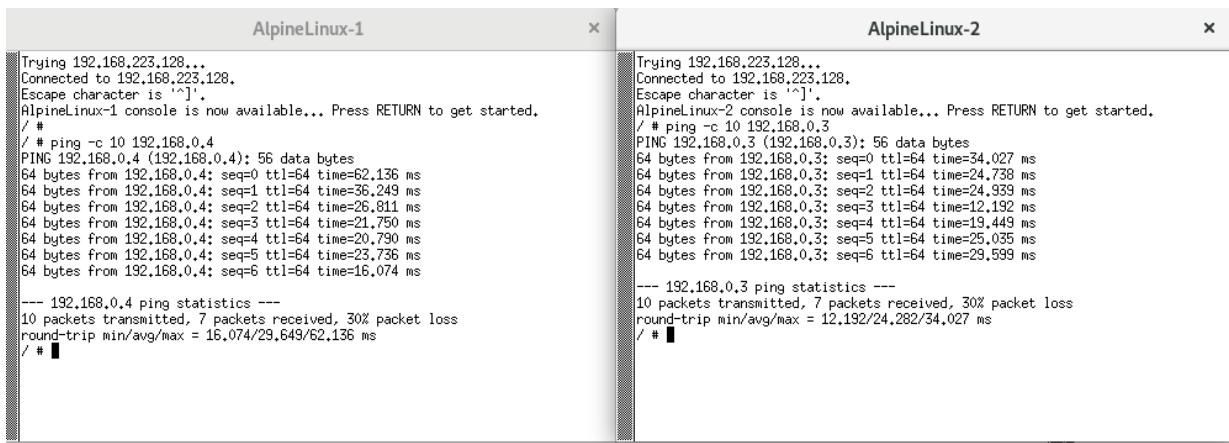
- El primero de ellos, creado por la aplicación *severalping* indica que cuando llegue tráfico con la MAC de origen :01 (referente al Alpine-1) y destino :04 (referente al Alpine-4) que sea de tipo ICMP tipo *REQUEST* no realice ninguna acción (equivalente a descartar el paquete, dado que no va a llegar al destino).
- El segundo de ellos, creado por la aplicación *fwd* actúa sobre las mismas parejas de *hosts*, pero en este caso seleccionando todo el tráfico que pasa por los enlaces en los que están conectados. Tiene como objetivo que aquel tráfico que no sea ICMP sí llegue al destino. Hay que tener en cuenta las prioridades, ya que, si esta regla tuviese mayor prioridad que la anterior, al llegar tráfico ICMP sí se enviaría puesto que se ejecutaría esta regla (que selecciona todo el tráfico) en vez de la definida anteriormente. Al tener menor prioridad, el tráfico ICMP va por la regla anterior bloqueándose, mientras que el resto del tráfico va por ésta.

5.2.3.2 Enviando *pings* entre 2 parejas de *hosts*

El objetivo de esta prueba es comprobar que, aunque se manden *pings* entre varios pares de *hosts*, sólo 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. La ejecución puede verse en la Figura 53.

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



The screenshot shows two terminal windows side-by-side. Both windows have a title bar 'AlpineLinux-1' and 'AlpineLinux-2' respectively. The left window (AlpineLinux-1) contains the following text:

```

Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is ']'.
AlpineLinux-1 console is now available... Press RETURN to get started.
/ #
/ # ping -c 10 192.168.0.4
PING 192.168.0.4 (192.168.0.4): 56 data bytes
64 bytes from 192.168.0.4: seq=0 ttl=64 time=62.136 ms
64 bytes from 192.168.0.4: seq=1 ttl=64 time=36.249 ms
64 bytes from 192.168.0.4: seq=2 ttl=64 time=26.811 ms
64 bytes from 192.168.0.4: seq=3 ttl=64 time=21.750 ms
64 bytes from 192.168.0.4: seq=4 ttl=64 time=20.790 ms
64 bytes from 192.168.0.4: seq=5 ttl=64 time=23.736 ms
64 bytes from 192.168.0.4: seq=6 ttl=64 time=16.074 ms
--- 192.168.0.4 ping statistics ---
10 packets transmitted, 7 packets received, 30% packet loss
round-trip min/avg/max = 16.074/29.649/62.136 ms
/ #

```

The right window (AlpineLinux-2) contains the following text:

```

Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is ']'.
AlpineLinux-2 console is now available... Press RETURN to get started.
/ #
/ # ping -c 10 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: seq=0 ttl=64 time=34.027 ms
64 bytes from 192.168.0.3: seq=1 ttl=64 time=24.738 ms
64 bytes from 192.168.0.3: seq=2 ttl=64 time=24.939 ms
64 bytes from 192.168.0.3: seq=3 ttl=64 time=12.192 ms
64 bytes from 192.168.0.3: seq=4 ttl=64 time=19.449 ms
64 bytes from 192.168.0.3: seq=5 ttl=64 time=25.035 ms
64 bytes from 192.168.0.3: seq=6 ttl=64 time=29.599 ms
--- 192.168.0.3 ping statistics ---
10 packets transmitted, 7 packets received, 30% packet loss
round-trip min/avg/max = 12.192/24.282/34.027 ms
/ #

```

Figura 53: Envío de 2 *pings* entre 2 parejas de *hosts* diferentes.

Los flujos que se crean son los siguientes (ver Figura 54):

```
onos@root > flows
deviceId=0:0000e59578ccb49, flowRuleCount=9
    id=00000d4bf20, state=ADDED, bytes=0, packets=0, duration=305, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultT
rafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=1000078cebd9, state=ADDED, bytes=1260, packets=30, duration=305, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=Defau
ltTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=1000094122673, state=ADDED, bytes=0, packets=0, duration=305, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:l1dp], treatment=DefaultT
rafficTreatment{immediate=[OUTPUT:UNTRIGGER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=10000d46d8f1, state=ADDED, bytes=664, packets=68, duration=303, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4, IP_PROTO:1, ICM
PV4_TYPE:8], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
    id=000009d138b5a, state=ADDED, bytes=0, packets=0, duration=1, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.severalping, selector=[ETH_DST:00:00:00:00:00:03, ETH
_SRC:00:00:00:00:00:02, ETH_TYPE:ip4, IP_PROTO:1, ICMPV4_TYPE:8], treatment=DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[], cleared=false, StatTrig
ger=null, metadata=null}
    id=00000074c044e, state=ADDED, bytes=0, packets=0, duration=2, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.severalping, selector=[ETH_DST:00:00:00:00:00:04, ETH
_SRC:00:00:00:00:00:01, ETH_TYPE:ip4, IP_PROTO:1, ICMPV4_TYPE:8], treatment=DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[], cleared=false, StatTrig
ger=null, metadata=null}
    id=310000b1dd280, state=ADDED, bytes=588, packets=6, duration=10, liveType=UNKNOWN, priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:3, ETH_DST:00:00:00:00:00:02
, ETH_SRC:00:00:00:00:00:03], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:2], deferred[], transition=None, meter[], cleared=false, StatTrigger=null, metadata=null}
    id=310000fa97964, state=ADDED, bytes=588, packets=6, duration=11, liveType=UNKNOWN, priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:4, ETH_DST:00:00:00:00:00:01
, ETH_SRC:00:00:00:00:00:04], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], deferred[], transition=None, meter[], cleared=false, StatTrigger=null, metadata=null}
    id=1000030e3829a, state=ADDED, bytes=586, packets=7, duration=305, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=DefaultTra
fficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[], cleared=true, StatTrigger=null, metadata=null}
onos@root >
```

Figura 54: Flujos creados en la aplicación.

Como se puede ver, en este caso se han creado 2 reglas de flujo que realizan el baneo (una por cada par de envío) y, además, también hay 2 reglas (de menor prioridad) creadas por la aplicación *fwd* que permiten el envío de tráfico normal.

5.2.3.3 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.

```
AlpineLinux-1
/ # ping -c 10 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: seq=0 ttl=64 time=41.211 ms
64 bytes from 192.168.0.3: seq=1 ttl=64 time=30.956 ms
64 bytes from 192.168.0.3: seq=2 ttl=64 time=22.284 ms
64 bytes from 192.168.0.3: seq=3 ttl=64 time=25.369 ms
64 bytes from 192.168.0.3: seq=4 ttl=64 time=31.549 ms
64 bytes from 192.168.0.3: seq=5 ttl=64 time=25.753 ms
64 bytes from 192.168.0.3: seq=6 ttl=64 time=27.924 ms
...
--- 192.168.0.3 ping statistics ---
10 packets transmitted, 7 packets received, 30% packet loss
round-trip min/avg/max = 22,284/29,292/41.211 ms
/ #

AlpineLinux-2
/ # ping -c 10 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: seq=0 ttl=64 time=30.036 ms
64 bytes from 192.168.0.3: seq=1 ttl=64 time=17.431 ms
64 bytes from 192.168.0.3: seq=2 ttl=64 time=22.018 ms
64 bytes from 192.168.0.3: seq=3 ttl=64 time=31.907 ms
64 bytes from 192.168.0.3: seq=4 ttl=64 time=32.544 ms
64 bytes from 192.168.0.3: seq=5 ttl=64 time=27.848 ms
64 bytes from 192.168.0.3: seq=6 ttl=64 time=27.120 ms
...
--- 192.168.0.3 ping statistics ---
10 packets transmitted, 7 packets received, 30% packet loss
round-trip min/avg/max = 17.431/26.986/32.544 ms
/ #
```

Figura 55: Ejecución del comando que envía pings desde 2 hosts a un tercero.

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 sí puedan interactuar.

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

```

:00:00:00:00:03
11:03:01.856 INFO [AppComponent] Ping 5/7 received from 00:00:00:00:02 to 00
:00:00:00:00:03
11:03:02.457 INFO [AppComponent] Ping 6/7 received from 00:00:00:00:01 to 00
:00:00:00:00:03
11:03:02.869 INFO [AppComponent] Ping 6/7 received from 00:00:00:00:02 to 00
:00:00:00:00:03
11:03:03.459 INFO [AppComponent] Ping 7/7 received from 00:00:00:00:01 to 00
:00:00:00:00:03
11:03:03.860 INFO [AppComponent] Ping 7/7 received from 00:00:00:00:02 to 00
:00:00:00:00:03
11:03:04.463 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:03 has already been received. 60 seconds ban
11:03:04.859 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00
:00:00:02 to 00:00:00:00:03 has already been received. 60 seconds ban
11:03:05.457 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:03 has already been received. 60 seconds ban
11:03:05.860 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00
:00:00:02 to 00:00:00:00:03 has already been received. 60 seconds ban
11:03:06.460 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:03 has already been received. 60 seconds ban
11:03:06.862 WARN [AppComponent] Limit of 7 pings reached!!! Ping from 00:00:00
:00:00:02 to 00:00:00:00:03 has already been received. 60 seconds ban
11:04:04.985 WARN [AppComponent] Re-enabled ping from 00:00:00:00:01 to 00:0
0:00:00:00:03
11:04:04.997 WARN [AppComponent] Re-enabled ping from 00:00:00:00:02 to 00:0
0:00:00:00:03
11:04:45.407 WARN [ServerSessionImpl] exceptionCaught(ServerSessionImpl[onos@/1
92.168.122.122;36606])[state=Opened] InterruptedByTimeoutException: null

```

Figura 56: Vista desde el controlador de la ejecución del comando.

5.2.3.4 Modificando los parámetros configurables

En esta 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 Figura 57, los parámetros nuevos van a permitir únicamente 3 *pings* y que el tiempo de baneo pase a ser de 20 segundos.

```

onos@root > cfg set org.onosproject.severalping.AppComponent MAX_PINGS 3
onos@root > cfg set org.onosproject.severalping.AppComponent TIME_BAN 30
onos@root >

```

Figura 57: Comandos que modifican los parámetros configurables.

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

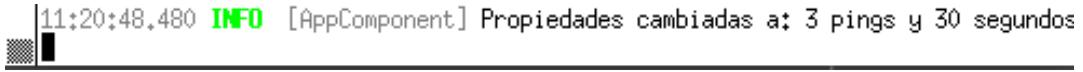


Figura 58: Vista desde el controlador de la modificación de los parámetros configurables.

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

```
AlpineLinux-1
x
/ # ping -c 10 192.168.0.2
PING 192.168.0.2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: seq=0 ttl=64 time=38.865 ms
64 bytes from 192.168.0.2: seq=1 ttl=64 time=20.575 ms
64 bytes from 192.168.0.2: seq=2 ttl=64 time=28.275 ms

--- 192.168.0.2 ping statistics ---
10 packets transmitted, 3 packets received, 70% packet loss
round-trip min/avg/max = 20.575/29.238/38.865 ms
/ #
```

Figura 59: Resultado del envío de 10 pings con los parámetros cambiados.

Vemos en la Figura 59 como, efectivamente, de los 10 *pings* enviados en esta ocasión, sólo llegan 3 mientras que los otros 7 han sido descartados. Lo comprobamos también viendo el *log* en el controlador.

```

Onos-1
x
0:00:00:00:03
11:04:45.407 [WARN] [ServerSessionImpl] exceptionCaught(ServerSessionImpl[onos@/192.168.122.122:36606])[state=Opened] InterruptedByTimeoutException: null
11:20:20.733 [INFO] [ServerUserAuthService] Session onos@/192.168.122.122:36618 a
uthenticated
11:20:40.997 [INFO] [AppComponent] Propiedades cambiadas a: 3 pings y 60 segundos
11:20:48.480 [INFO] [AppComponent] Propiedades cambiadas a: 3 pings y 30 segundos
11:25:24.461 [INFO] [AppComponent] Ping 1/3 received from 00:00:00:00:01 to 00
:00:00:00:02
11:25:25.449 [INFO] [AppComponent] Ping 2/3 received from 00:00:00:00:01 to 00
:00:00:00:02
11:25:26.447 [INFO] [AppComponent] Ping 3/3 received from 00:00:00:00:01 to 00
:00:00:00:02
11:25:27.448 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:28.456 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:29.452 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:30.452 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:31.456 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:32.457 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:33.471 [WARN] [AppComponent] Limit of 3 pings reached!!! Ping from 00:00:00
:00:00:01 to 00:00:00:00:02 has already been received. 60 seconds ban
11:25:59.987 [WARN] [AppComponent] Re-enabled ping from 00:00:00:00:01 to 00:0
:00:00:00:02

```

Figura 60: Vista desde el controlador del resultado del envío de los pings.

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

5.2.3.5 Enviando otro tipo de tráfico

Tal y como se ha comentado, la aplicación permite banear el envío de *pings*, sin embargo, el restante tráfico debería poder enviarse de forma normal, pues no es el objetivo de esta aplicación banear otro tipo de tráfico.

Para poder enviar otro tipo de tráfico se utilizará la herramienta *netcat* [28]. Para ello, comunicaremos 2 *hosts*, en nuestro caso Alpine-1 y Alpine-2, el primero hará las veces de receptor, mientras que el segundo será aquel que envía el tráfico. En la Figura 61 se ejecutan los comandos que permite realizar esta comunicación.

```

AlpineLinux-1
x
/ # nc -l 1234

AlpineLinux-2
x
/ # nc 192.168.0.1 1234
/ #

```

Figura 61: Comandos para comunicar 2 hosts mediante la herramienta netcat.

Para probarlo lo que se va a realizar es enviar tráfico TCP, a través de esta herramienta, cuando el *host* se encuentra baneado. Para ello, antes de realizar la comunicación, hemos aumentado el tiempo de baneo mediante los parámetros configurables a 5 minutos con el objetivo de que dé

tiempo a realizar la comunicación con *netcat*. Una vez el enlace está baneado, enviamos cualquier tipo de texto para comprobar si llega al destino. El resultado se puede ver en la Figura 62.

The screenshot shows two terminal windows side-by-side. Both are titled "AlpineLinux-1" and "AlpineLinux-2". In the left terminal window, the command "/ # nc -l 1234" is run. In the right terminal window, the command "/ # nc 192.168.0.1 1234" is run. Both terminals show the message "Prueba de que el envío de otro tráfico (en este caso TCP) no afecta a la hora de limitar el tráfico de pings existente en la red cuando el enlace se encaneado". This indicates that traffic other than pings (in this case, TCP) is not affected by the link being disabled.

Figura 62: Envío de tráfico TCP a través de la herramienta netcat.

Tal y como se puede comprobar, la comunicación se ha realizado correctamente, lo que indica que el hecho de que el enlace esté baneado afecta, tal y como se desea, únicamente a los *pings* enviados y no al resto de tráfico.

5.3 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 a continuación.

5.3.1 Aplicación statsshow

5.3.1.1 Motivación

El objetivo de esta aplicación es informativo, ya que muestra las estadísticas de tráfico de todos los puertos de los dispositivos que controlamos. En este caso los puertos del Open vSwitch.

5.3.1.2 Componentes de la aplicación

En este caso la aplicación es muy sencilla y únicamente tiene un apartado.

5.3.1.2.1 Tareas repetidas

Para llevar a cabo el funcionamiento correcto de la tarea es necesario crear una tarea que se repita periódicamente. 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 en nuestro caso.

- 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, en la aplicación desarrollada, es un parámetro configurable, por tanto, puede modificarse desde la CLI de ONOS tal y como se ha explicado en el apartado 5.2.4.4.

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

```
timer.cancel();
```

Dentro de la tarea, el objetivo, como se ha comentado, es obtener las estadísticas 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*.

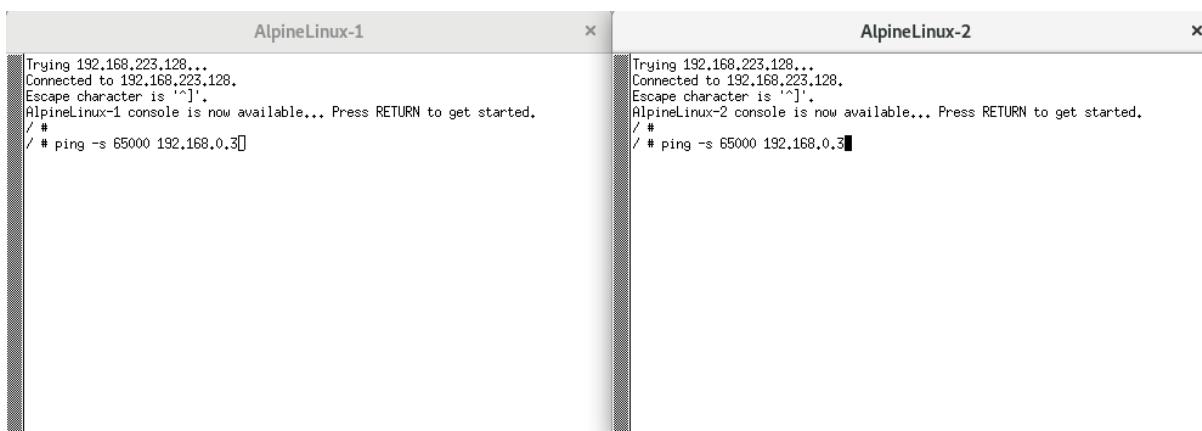
Una vez creados los 2 bucles necesarios obtenemos las siguientes estadísticas, en concreto obtendremos las siguientes estadísticas:

- *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

5.3.1.3 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, tal y como se ve en la Figura 63.



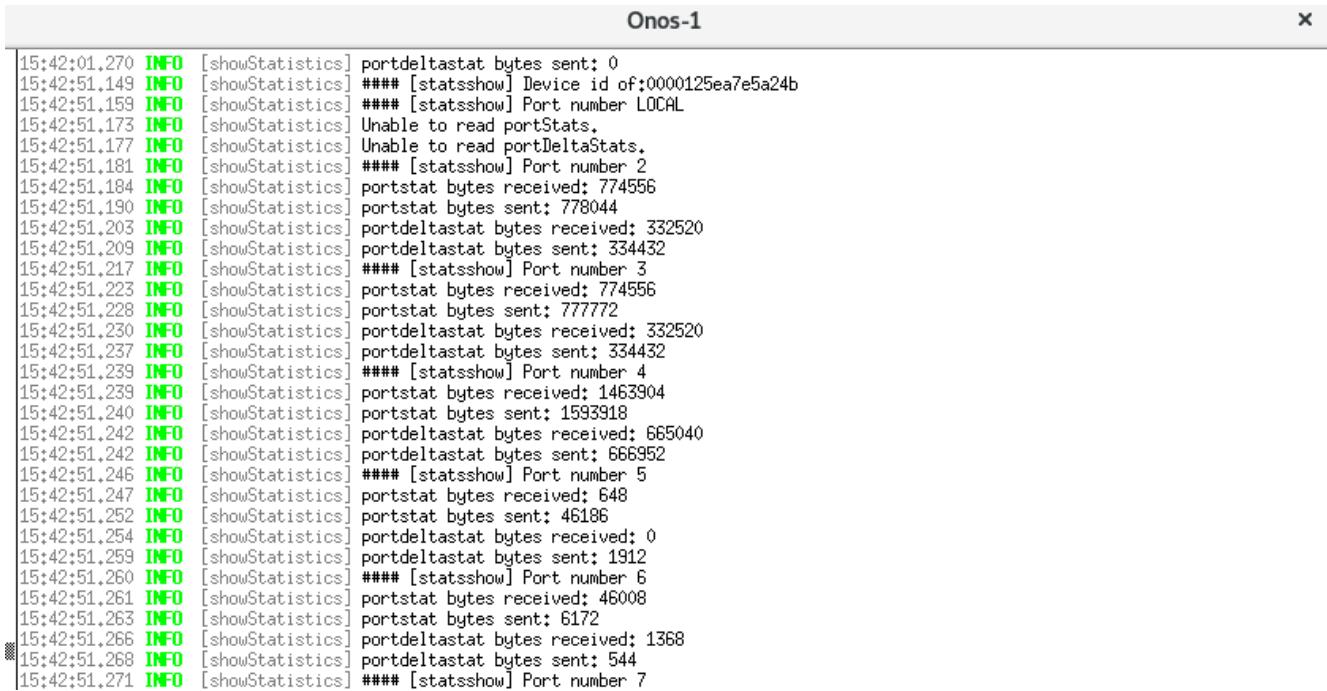
The screenshot shows two terminal windows side-by-side. The left window is titled "AlpineLinux-1" and the right one is titled "AlpineLinux-2". Both windows display a command-line interface with the following text:

```
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^]'.
AlpineLinux-1 console is now available... Press RETURN to get started.
/ #
/ # ping -s 65000 192.168.0.3
```

The command `ping -s 65000 192.168.0.3` is being typed at the AlpineLinux prompt in both windows.

Figura 63: Envío de pings para comprobar el funcionamiento de la aplicación.

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



```

Onos-1
x
15:42:01.270 INFO [showStatistics] portdeltastat bytes sent: 0
15:42:51.149 INFO [showStatistics] ##### [statsshow] Device id of:0000125ea7e5a24b
15:42:51.159 INFO [showStatistics] ##### [statsshow] Port number LOCAL
15:42:51.173 INFO [showStatistics] Unable to read portStats.
15:42:51.177 INFO [showStatistics] Unable to read portDeltaStats.
15:42:51.181 INFO [showStatistics] ##### [statsshow] Port number 2
15:42:51.184 INFO [showStatistics] portstat bytes received: 774556
15:42:51.190 INFO [showStatistics] portstat bytes sent: 778044
15:42:51.203 INFO [showStatistics] portdeltastat bytes received: 332520
15:42:51.209 INFO [showStatistics] portdeltastat bytes sent: 334432
15:42:51.217 INFO [showStatistics] ##### [statsshow] Port number 3
15:42:51.223 INFO [showStatistics] portstat bytes received: 774556
15:42:51.228 INFO [showStatistics] portstat bytes sent: 777772
15:42:51.230 INFO [showStatistics] portdeltastat bytes received: 332520
15:42:51.237 INFO [showStatistics] portdeltastat bytes sent: 334432
15:42:51.239 INFO [showStatistics] ##### [statsshow] Port number 4
15:42:51.239 INFO [showStatistics] portstat bytes received: 1463904
15:42:51.240 INFO [showStatistics] portstat bytes sent: 1593918
15:42:51.242 INFO [showStatistics] portdeltastat bytes received: 665040
15:42:51.242 INFO [showStatistics] portdeltastat bytes sent: 666952
15:42:51.246 INFO [showStatistics] ##### [statsshow] Port number 5
15:42:51.247 INFO [showStatistics] portstat bytes received: 648
15:42:51.252 INFO [showStatistics] portstat bytes sent: 46186
15:42:51.254 INFO [showStatistics] portdeltastat bytes received: 0
15:42:51.259 INFO [showStatistics] portdeltastat bytes sent: 1912
15:42:51.260 INFO [showStatistics] ##### [statsshow] Port number 6
15:42:51.261 INFO [showStatistics] portstat bytes received: 46008
15:42:51.263 INFO [showStatistics] portstat bytes sent: 6172
15:42:51.266 INFO [showStatistics] portdeltastat bytes received: 1368
15:42:51.268 INFO [showStatistics] portdeltastat bytes sent: 544
15:42:51.271 INFO [showStatistics] ##### [statsshow] Port number 7

```

Figura 64: Vista desde el controlador de las estadísticas recogidas.

Tal y como se puede observar en la Figura 64, 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 envía el doble de tráfico, esto es debido a la suma agregada de los otros 2 *hosts* que le están enviando *pings*. El hecho de que sea el doble de tráfico es lo que indica que la aplicación recoge correctamente las estadísticas.

Por otro lado, también 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.

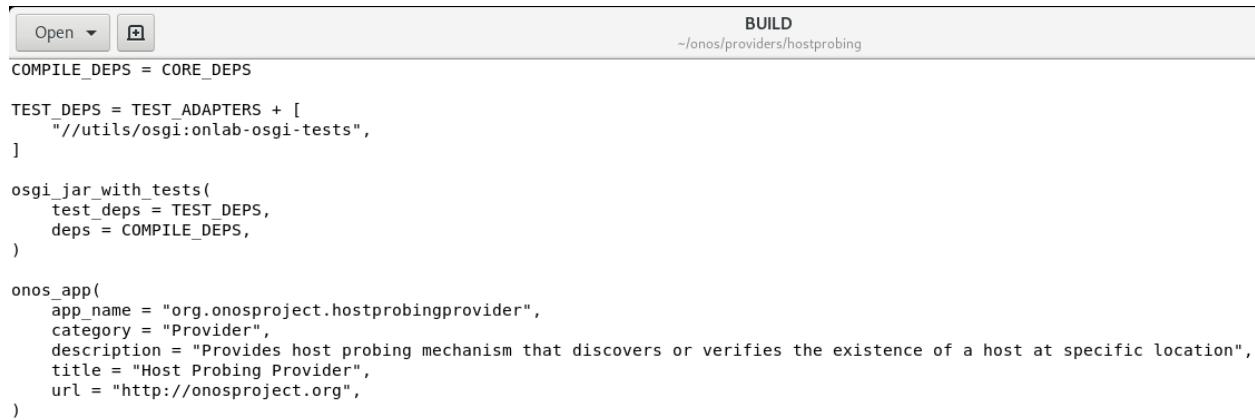
5.3.2 Aplicación *detectHost*

5.3.2.1 Motivación y explicación teórica

El objetivo de esta aplicación es conocer, por parte del controlador, todos los *hosts* que tenemos conectados a los dispositivos controlados por él mismo, en nuestro caso, 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. Ésta es la app *org.onosproject.hostprobingprovider*. El motivo de necesitar de una aplicación más 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 controlador ONOS. Es esta aplicación la que registra este servicio. El código de la aplicación puede encontrarse en la siguiente ruta *onos/providers/hostprobing*.

En el fichero BUILD se registra la aplicación tal y como se ve en la Figura 65.



```

Open ▾ + BUILD
~/onos/providers/hostprobing

COMPILE_DEPS = CORE_DEPS

TEST_DEPS = TEST_ADAPTERS + [
    "//utils/osgi:onlab-osgi-tests",
]

osgi_jar_with_tests(
    test_deps = TEST_DEPS,
    deps = COMPILE_DEPS,
)

onos_app(
    app_name = "org.onosproject.hostprobingprovider",
    category = "Provider",
    description = "Provides host probing mechanism that discovers or verifies the existence of a host at specific location",
    title = "Host Probing Provider",
    url = "http://onosproject.org",
)

```

Figura 65: Fichero BUILD de la aplicación hostProbingProvider.

Por otro lado, en el fichero *DefaultHostProbingProvider.java* se ve cómo se activa el servicio y la implementación de los métodos que se van a utilizar posteriormente.

5.3.2.2 Componentes de la aplicación

Centrándonos ya en la aplicación *detectHost*, vemos que se puede dividir en 2 partes.

5.3.2.2.1 Tarea repetida

En este caso, a diferencia de la aplicación anterior, se utilizan 2 tareas repetidas. Dentro de la primera, se utiliza el servicio *hostService* definido previamente. 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 el caso de que las direcciones IP sean asignadas estáticamente, como es nuestro caso, debemos forzarlos a que manden algún paquete (por ejemplo, *pings* entre ellos). Sin embargo, si las direcciones IP estuviesen asignadas mediante DHCP se detectaría automáticamente. Esto se debe a que al conectarse envía paquetes para obtener una dirección IP, por tanto, ya provocarían que el controlador fuese notificado de la aparición de un nuevo *host*.

También cabe destacar que los detalles de los *hosts* dependen del número de dispositivos (Open vSwitch) existentes en la red:

- Si sólo hay 1 dispositivo reconocido en la red, se muestra de cada *host* su MAC y el puerto al que están conectados.
- Si hay más de 1 dispositivo reconocido en la red, se muestra de cada *host* su MAC, el id del dispositivo y el puerto al que están conectados.

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

Finalmente, hay que recordar que es necesario que en el método *deactivate* se cancelen los 2 temporizadores creados, para evitar que se queden residuales en ejecución continua tras la desactivación de la aplicación.

5.3.2.2 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 evento y presentar un mensaje informativo que aparecerá en la consola del controlador ONOS.

He de destacar que, como el *HostListener* viene definido como una interfaz, es necesario instanciarlo previamente de la forma que procede:

```
private final HostListener hostListener = new InternalHostListener();
```

Finalmente, comentar que, al igual que en el momento en el que se añade un nuevo *host*, el mensaje informativo que indica la desconexión de uno varía en función del número de dispositivos conectados. En caso de ser 1, no se indica el dispositivo al que pertenece, mientras que, en caso de ser más, sí se indica.

5.3.2.3 Banco de pruebas

Al igual que en la aplicación anterior, en este apartado simplemente se va a comprobar el funcionamiento de la misma, haciendo especial hincapié en los momentos en los cuales se detecta tanto la aparición de un nuevo *host* como la desconexión del mismo. Para ello ejecutamos las aplicaciones habituales auxiliares, añadiendo como se ha comentado la aplicación *org.onosproject.hostprobingprovider*.

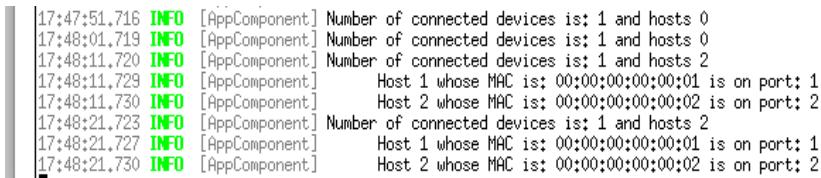
Una vez activadas, tanto las aplicaciones auxiliares como la aplicación *detectHost*, observamos que en el controlador se notifican los *hosts* conectados, que tal y como se ve, al inicio son 0, a pesar de que en la red tenemos todos los *hosts* conectados al Open vSwitch. Esto se debe, tal y como se ha comentado previamente, a que las direcciones IP han sido definidas estéticamente y no mediante el protocolo DHCP, y no se ha generado tráfico que permita su detección.



```
17:46:20,718 INFO [ApplicationManager] Application org.onosproject.detectHost has been activated
17:46:21,718 INFO [AppComponent] Number of connected devices is: 1 and hosts 0
17:46:31,719 INFO [AppComponent] Number of connected devices is: 1 and hosts 0
```

Figura 66: Vistas desde el controlador al activar la aplicación.

Realizamos un *ping* entre los hosts Alpine-1 y Alpine-2 y observamos en la Figura 67 como se notifica la aparición de los 2 *hosts* que intervienen en ese *ping*. En concreto, se notifica del número de *hosts* que se han detectado, así como su MAC y el puerto en el que están. Se recuerda que el ID del dispositivo sólo se muestra en caso de que haya más de 1 dispositivo conectado a la red.



```
17:47:51,716 INFO [AppComponent] Number of connected devices is: 1 and hosts 0
17:48:01,719 INFO [AppComponent] Number of connected devices is: 1 and hosts 0
17:48:11,720 INFO [AppComponent] Number of connected devices is: 1 and hosts 2
17:48:11,729 INFO [AppComponent] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
17:48:11,730 INFO [AppComponent] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
17:48:21,723 INFO [AppComponent] Number of connected devices is: 1 and hosts 2
17:48:21,727 INFO [AppComponent] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
17:48:21,730 INFO [AppComponent] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
```

Figura 67: Vista desde el controlador de la detección de los hosts.

Finalmente paramos el *ping* (lo que no provoca que se detecte como eliminado el *host*) y apagamos el *host* Alpine-1 para ver si bajo esta situación si se deja de detectar el dispositivo.

Comprobamos que, efectivamente, al apagar el *host* Alpine-1 se notifica su desconexión, mientras que el *host* Alpine-2 se mantiene conectado.

```
17:48:41,718 INFO [HppComponent]      Host 2 whose MAC is: 00:00:00:00:00:02 is on port: 2
17:48:47,111 INFO [AppComponent] Host with MAC 00:00:00:00:00:01 has been removed from port 1
17:48:51,719 INFO [AppComponent] Number of connected devices is: 1 and hosts 1
17:48:51,722 INFO [AppComponent]      Host 1 whose MAC is: 00:00:00:00:00:02 is on port: 2
17:49:01,727 INFO [AppComponent] Number of connected devices is: 1 and hosts 1
17:49:01,734 INFO [AppComponent]      Host 1 whose MAC is: 00:00:00:00:00:02 is on port: 2
17:49:11,730 INFO [AppComponent] Number of connected devices is: 1 and hosts 1
17:49:11,731 INFO [AppComponent]      Host 1 whose MAC is: 00:00:00:00:00:02 is on port: 2
```

Figura 68: Vista desde el controlador de la desconexión de un host.

5.3.3 Aplicación *detectHostBan*

5.3.3.1 Motivación

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 tiempo. La finalidad más clara es evitar que los *hosts* envíen una cantidad muy elevada de datos, al estilo de lo comentado con la aplicación *severalpings*, pero, en esta ocasión, restringiendo todo tipo de tráfico, no solamente mensajes ICMP.

5.3.3.2 Componentes de la aplicación

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 *deltaStatistics* dentro de un temporizador. En cada iteración del bucle iremos sumando el valor e introduciéndolo 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. Hay que 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 instala 2 reglas de flujo en el Open vSwitch que se pueden ver a continuación:

```
TrafficSelector selector = DefaultTrafficSelector.builder()
    .matchEthSrc(src)
    .build();
TrafficTreatment drop = DefaultTrafficTreatment.builder()
    .drop().build();

FlowRule rule1 = DefaultFlowRule.builder()
    .fromApp(appId)
    .forDevice(deviceId)
    .makePermanent()
```

```

.withSelector(selector)
.withPriority(DROP_PRIORITY)
.withTreatment(drop)
.build();

selector = DefaultTrafficSelector.builder()
.matchEthDst(src)
.build();

FlowRule rule2 = DefaultFlowRule.builder()
.fromApp(appId)
.forDevice(deviceId)
.makePermanent()
.withSelector(selector)
.withPriority(DROP_PRIORITY)
.withTreatment(drop)
.build();

```

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 esté baneada de la siguiente forma:

```
timer.schedule(new PingPruner(rule1,rule2), TIME_BAN * 1000);
```

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.

Si mantenemos el *ping*, una vez se ha superado el límite, el *timeout* asociado a la regla se resetea cada vez que hay un *matching* con un paquete ICMP, de esta forma, nunca vencería el temporizador, y, por tanto, nunca se borraría el flujo.

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

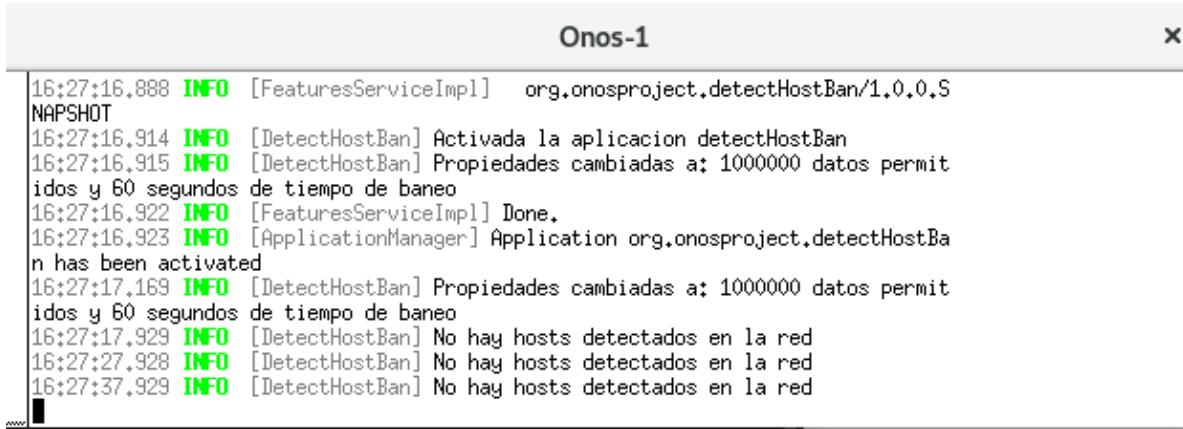
Dentro del este método se ejecuta a su vez método *run()*, que lo que hace es eliminar las 2 reglas y actualizar el *HashMap* volviendo a reiniciar el valor de los datos acumulados para la MAC que ya ha dejado de ser baneada.

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 y sobrecargando de manera innecesaria al controlador.

5.3.3.3 Banco de pruebas

En este apartado se va a probar el funcionamiento de esta aplicación para comprobar si se realizan correctamente los baneos cuando se supera el umbral.

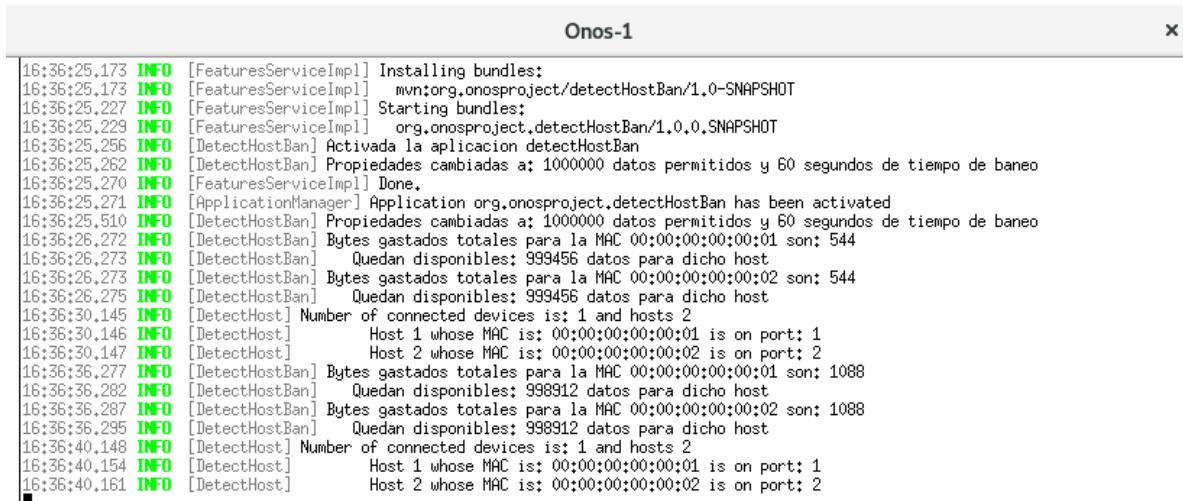
Para ello, en primer lugar, activamos las aplicaciones necesarias, entre las cuales se incluyen la aplicación *detectHost* explicada anteriormente, que nos permite ver cuando los *hosts* se desconectan de la red. Comprobamos también que, al principio, la red no tiene ningún *host* detectado. Además, tal y como se puede ver en la Figura 69, se ha establecido el límite de datos en 1MB y el tiempo de baneo en 60 segundos.



```
Onos-1
16:27:16,888 INFO [FeaturesServiceImpl] org.onosproject.detectHostBan/1.0.0-SNAPSHOT
16:27:16,914 INFO [DetectHostBan] Activada la aplicacion detectHostBan
16:27:16,915 INFO [DetectHostBan] Propiedades cambiadas a: 1000000 datos permitidos y 60 segundos de tiempo de baneo
16:27:16,922 INFO [FeaturesServiceImpl] Done.
16:27:16,923 INFO [ApplicationManager] Application org.onosproject.detectHostBan has been activated
16:27:17,169 INFO [DetectHostBan] Propiedades cambiadas a: 1000000 datos permitidos y 60 segundos de tiempo de baneo
16:27:17,929 INFO [DetectHostBan] No hay hosts detectados en la red
16:27:27,928 INFO [DetectHostBan] No hay hosts detectados en la red
16:27:37,929 INFO [DetectHostBan] No hay hosts detectados en la red
```

Figura 69: Vista desde el controlador de la activación de la aplicación *detectHostBan*.

A continuación, al igual que en aplicaciones anteriores, enviamos un *ping* desde el *host* Alpine-1 al *host* Alpine-2, para que los *hosts* sean reconocidos. Esto supondrá también que el tamaño de estos *pings* se empiece a descontar del límite establecido.



```
Onos-1
16:36:25,173 INFO [FeaturesServiceImpl] Installing bundles:
16:36:25,173 INFO [FeaturesServiceImpl] mvn:org.onosproject/detectHostBan/1.0-SNAPSHOT
16:36:25,227 INFO [FeaturesServiceImpl] Starting bundles:
16:36:25,229 INFO [FeaturesServiceImpl] org.onosproject.detectHostBan/1.0.0-SNAPSHOT
16:36:25,256 INFO [DetectHostBan] Activada la aplicacion detectHostBan
16:36:25,262 INFO [DetectHostBan] Propiedades cambiadas a: 1000000 datos permitidos y 60 segundos de tiempo de baneo
16:36:25,270 INFO [FeaturesServiceImpl] Done.
16:36:25,271 INFO [ApplicationManager] Application org.onosproject.detectHostBan has been activated
16:36:25,510 INFO [DetectHostBan] Propiedades cambiadas a: 1000000 datos permitidos y 60 segundos de tiempo de baneo
16:36:26,272 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:01 son: 544
16:36:26,273 INFO [DetectHostBan] Quedan disponibles: 999456 datos para dicho host
16:36:26,273 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:02 son: 544
16:36:26,275 INFO [DetectHostBan] Quedan disponibles: 999456 datos para dicho host
16:36:30,145 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
16:36:30,146 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:00:01 is on port: 1
16:36:30,147 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:00:02 is on port: 2
16:36:36,277 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:01 son: 1088
16:36:36,282 INFO [DetectHostBan] Quedan disponibles: 998912 datos para dicho host
16:36:36,287 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:02 son: 1088
16:36:36,295 INFO [DetectHostBan] Quedan disponibles: 998912 datos para dicho host
16:36:40,148 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
16:36:40,154 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:00:01 is on port: 1
16:36:40,161 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:00:02 is on port: 2
```

Figura 70: Vista desde el controlador de los datos acumulados.

En este caso se observa que se han enviado y recibido un total de 1088 bytes. Asimismo, se informa también de los datos restantes para cada *host*.

Si paramos el envío de *pings*, manteniendo las aplicaciones activadas, vemos que los datos aumentan “ellos solos”. Esto se debe a que, internamente, el Open vSwitch tiene activado el protocolo LLDP (*Link Layer Discovery Protocol*) [29] cuya funcionalidad consiste en conocer los dispositivos vecinos y los puertos a los que está conectados los diversos dispositivos que conforman la red.

Para comprobarlo, capturamos el tráfico que va entre el Open vSwitch y cualquiera de los *hosts*. Para ello, clicamos desde el GNS3 con el botón derecho y clicamos en *Start Capture*, lo que provoca que se abra el programa *Wireshark*, tal y como se ve en la Figura 71.

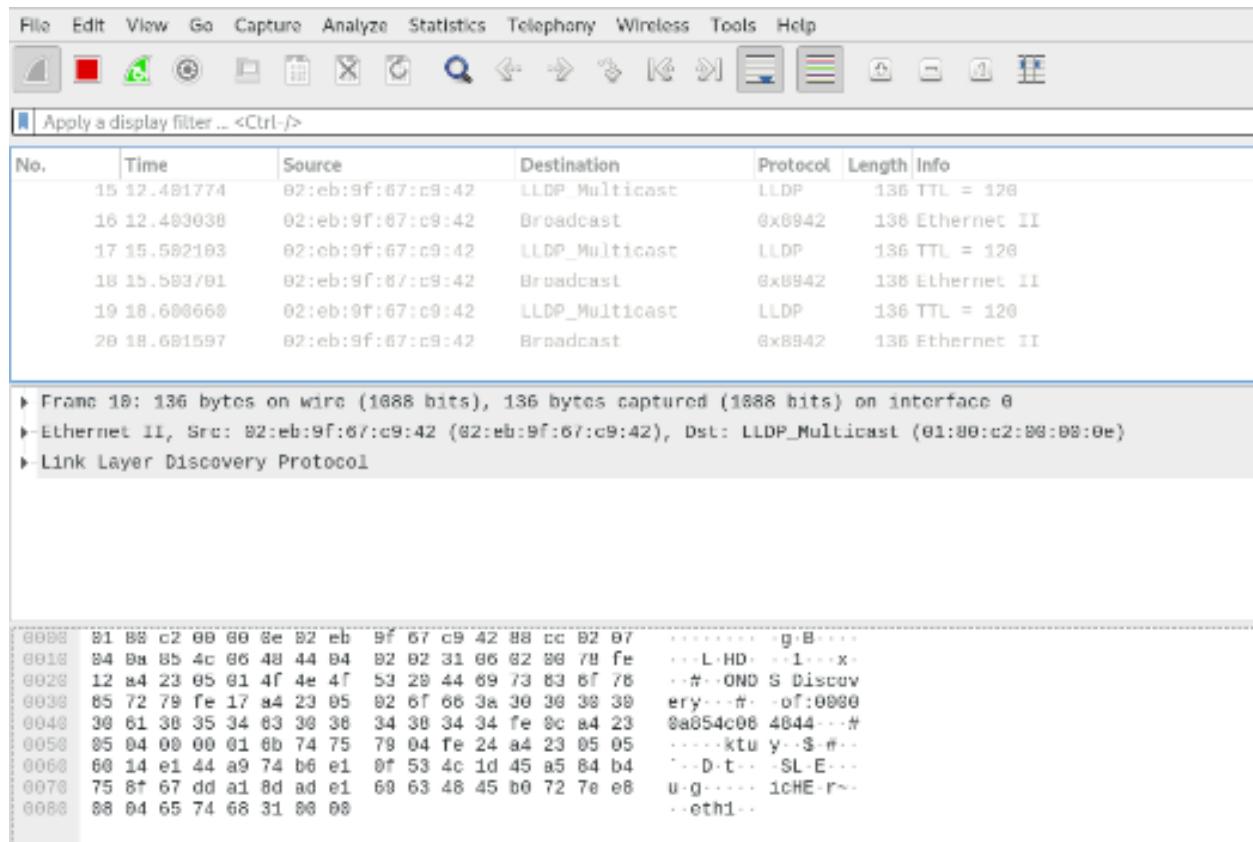


Figura 71: Captura de Wireshark del protocolo LLDP.

Tal y como se ve, cada 3 segundos se envía un paquete LLDP y un paquete BDDP, que hace que las estadísticas aumenten en aproximadamente 900 bytes por cada 10 segundos [30].

A continuación, comprobaremos el sistema de baneo. Para ello enviaremos *pings* de tamaño elevado con la opción -s con el objetivo de alcanzar el límite rápidamente.

Tal y como se puede ver, en la Figura 72, se notifican los 2 baneos a las 2 MAC (tanto la que envía el *ping*, como la que lo recibe) notificándose también la cantidad de bytes por la que se ha superado respecto al umbral permitido.

```

Onos-1
16:38:40.145 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
16:38:40.146 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
16:38:40.147 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
16:38:46.277 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 6800
16:38:46.283 INFO [DetectHostBan] Quedan disponibles: 993200 datos para dicho host
16:38:46.289 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 6800
16:38:46.293 INFO [DetectHostBan] Quedan disponibles: 993200 datos para dicho host
16:38:50.150 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
16:38:50.153 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
16:38:50.158 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
16:38:56.277 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 672384
16:38:56.282 INFO [DetectHostBan] Quedan disponibles: 327616 datos para dicho host
16:38:56.288 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 672384
16:38:56.293 INFO [DetectHostBan] Quedan disponibles: 327616 datos para dicho host
16:39:00.149 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
16:39:00.155 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
16:39:00.161 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
16:39:06.277 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 1337780
16:39:06.282 INFO [DetectHostBan] Has superado por 337780 bytes el total de datos permitido
16:39:06.304 ERROR [DetectHostBan] Baneo aplicado a la MAC: 00:00:00:00:01
16:39:06.308 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 1337780
16:39:06.313 INFO [DetectHostBan] Has superado por 337780 bytes el total de datos permitido
16:39:06.318 ERROR [DetectHostBan] Baneo aplicado a la MAC: 00:00:00:00:02

```

Figura 72: Vista desde el controlador del baneo de los hosts al superar el umbral.

Si ejecutamos el comando *flows* desde el controlador ONOS, mientras el baneo persiste, vemos, en la Figura 73 las reglas de flujo que se han creado.

```

alpinemodificada-ssh-1
onos@root > flows
deviceId=of:00004a709eaec48, flowRuleCount=10
  id=10000070c854a, state=ADDED, bytes=0, packets=0, duration=475, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:11dp], treatment=[DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}]
  id=100006fb1b1b6, state=ADDED, bytes=0, packets=0, duration=475, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=[DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}]
  id=10000f67209a2, state=ADDED, bytes=4788, packets=114, duration=475, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=[DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}]
  id=c000001967e39b, state=ADDED, bytes=0, packets=0, duration=1, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.detectHostBan, selector=[ETH_DST:00:00:00:00:01], treatment=[DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[]}, cleared=false, StatTrigger=null, metadata=null]
  id=c00000aceb25e2, state=ADDED, bytes=0, duration=1, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.detectHostBan, selector=[ETH_DST:00:00:00:00:02], treatment=[DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[]}, cleared=false, StatTrigger=null, metadata=null]
  id=c00000bb936432, state=ADDED, bytes=0, duration=1, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.detectHostBan, selector=[ETH_SRC:00:00:00:02], treatment=[DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[]}, cleared=false, StatTrigger=null, metadata=null]
  id=c00000bf5045ee, state=ADDED, bytes=66504, packets=44, duration=1, liveType=UNKNOWN, priority=129, tableId=0, appId=org.onosproject.detectHostBan, selector=[ETH_SRC:00:00:00:00:01], treatment=[DefaultTrafficTreatment{immediate=[NOACTION], deferred[], transition=None, meter[]}, cleared=false, StatTrigger=null, metadata=null]
  id=3100005be50a75, state=ADDED, bytes=1526564, packets=1010, duration=23, liveType=UNKNOWN, priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:1, ETH_DST:00:00:00:00:02, ETH_SRC:00:00:00:00:01], treatment=[DefaultTrafficTreatment{immediate=[OUTPUT:2], deferred[], transition=None, meter[]}, cleared=false, StatTrigger=null, metadata=null]
  id=31000073a62c35, state=ADDED, bytes=1526564, packets=1010, duration=23, liveType=UNKNOWN, priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:2, ETH_DST:00:00:00:00:01, ETH_SRC:00:00:00:00:02], treatment=[DefaultTrafficTreatment{immediate=[OUTPUT:1], deferred[], transition=None, meter[]}, cleared=false, StatTrigger=null, metadata=null]
  id=100002c6b5d05, state=ADDED, bytes=15336, packets=12, duration=475, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=[DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred[], transition=None, meter[]}, cleared=true, StatTrigger=null, metadata=null]

```

Figura 73: Vista de los flujos creados al banear hosts.

Las 2 penúltimas reglas vemos que se refieren al envío normal, que indica que el Open vSwitch ha aprendido por qué puerto debe enviar los datos que llegan cuando el destino es Alpine-1 y Alpine-2. Estas reglas persisten cuando el baneo haya finalizado y son las que permiten que los *pings* lleguen en situaciones normales. Sin embargo, vemos como también se han creado reglas de mayor prioridad, con el mismo selector (por tanto, se ejecutarán estas), pero que descartan el paquete impidiendo que lleguen los *pings*.

Comprobamos ahora que, si enviamos desde un tercer *host* a cualquiera de los que están baneados, no se reciben esos *pings*.

```

Onos-1
16:46:56,302 ERROR [DetectHostBan] Baneo aplicado a la MAC: 00:00:00:00:00:01
16:46:56,303 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:02 son: 1331252
16:46:56,315 INFO [DetectHostBan] Has superado por 331252 bytes el total de datos permitido
16:46:56,319 ERROR [DetectHostBan] Baneo aplicado a la MAC: 00:00:00:00:00:02
16:47:00,157 INFO [DetectHost] Number of connected devices is: 1 and hosts 3
16:47:00,161 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:00:03 is on port: 3
16:47:00,166 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:00:01 is on port: 1
16:47:00,171 INFO [DetectHost] Host 3 whose MAC is: 00:00:00:00:00:02 is on port: 2
16:47:06,277 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:03 son: 6022
16:47:06,287 INFO [DetectHostBan] Quedan disponibles: 993978 datos para dicho host
16:47:10,148 INFO [DetectHost] Number of connected devices is: 1 and hosts 3
16:47:10,150 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:00:03 is on port: 3
16:47:10,156 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:00:01 is on port: 1
16:47:10,162 INFO [DetectHost] Host 3 whose MAC is: 00:00:00:00:00:02 is on port: 2
16:47:16,273 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:03 son: 6294
16:47:16,274 INFO [DetectHostBan] Quedan disponibles: 993706 datos para dicho host
16:47:20,153 INFO [DetectHost] Number of connected devices is: 1 and hosts 3
16:47:20,164 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:00:03 is on port: 3
16:47:20,165 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:00:01 is on port: 1
16:47:20,168 INFO [DetectHost] Host 3 whose MAC is: 00:00:00:00:00:02 is on port: 2
16:47:26,276 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:03 son: 6566
16:47:26,278 INFO [DetectHostBan] Quedan disponibles: 993434 datos para dicho host

AlpineLinux-3
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is ']'.
/ # ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
^C
--- 192.168.0.1 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
/ # ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2): 56 data bytes
^C
--- 192.168.0.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
/ #

```

Figura 74: Comprobación del baneo de los hosts al superar el umbral.

Tal y como se esperaba, en el controlador ONOS aparecen como baneados los *hosts* 1 y 2 y desde el 3 enviamos a estos 2 y podemos comprobar que ni se envían ni se reciben *pings*, lo cual indica que las reglas de flujo están bien creadas.

Transcurrido 1 minuto, tal y como se puede ver en la Figura 75, el baneo queda eliminado, se restablece la cantidad de datos al máximo, y, por tanto, se permite nuevamente el envío de datos entre los *hosts* que estaban baneados.

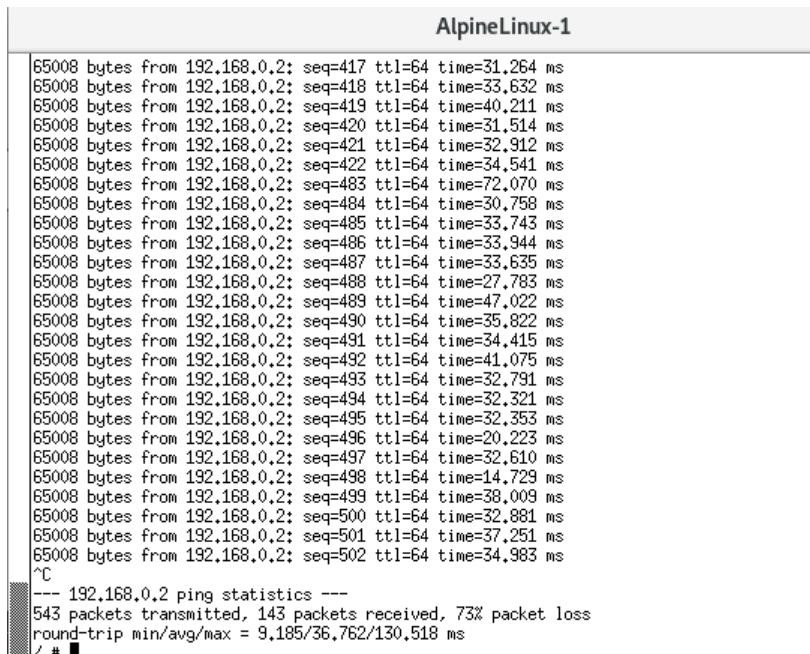
```

16:49:16.289 INFO [DetectHostBan] Quedan disponibles: 988266 datos para dicho host
16:49:16.302 WARN [DetectHostBan] Baneo eliminado a la MAC: 00:00:00:00:00:01
16:49:16.315 WARN [DetectHostBan] Baneo eliminado a la MAC: 00:00:00:00:00:02
16:49:20.144 INFO [DetectHost] Number of connected devices is: 1 and hosts 3
16:49:20.144 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:00:03 is on port: 3
16:49:20.144 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:00:01 is on port: 1
16:49:20.145 INFO [DetectHost] Host 3 whose MAC is: 00:00:00:00:00:02 is on port: 2
16:49:26.279 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:03 son: 12006
16:49:26.280 INFO [DetectHostBan] Quedan disponibles: 987994 datos para dicho host
16:49:26.300 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:00:01 son: 665396
16:49:26.301 INFO [DetectHostBan] Quedan disponibles: 334604 datos para dicho host

```

Figura 75: Vista de la eliminación del baneo de los hosts.

Como detalle, se ha dejado un *ping* de tamaño 65000 bytes durante un tiempo prolongado para comprobar la relación entre los mensajes enviados y los descartados obteniendo el siguiente resultado:



```

AlpineLinux-1
=====
65008 bytes from 192.168.0.2: seq=417 ttl=64 time=31.264 ms
65008 bytes from 192.168.0.2: seq=418 ttl=64 time=33.632 ms
65008 bytes from 192.168.0.2: seq=419 ttl=64 time=40.211 ms
65008 bytes from 192.168.0.2: seq=420 ttl=64 time=31.514 ms
65008 bytes from 192.168.0.2: seq=421 ttl=64 time=32.912 ms
65008 bytes from 192.168.0.2: seq=422 ttl=64 time=34.541 ms
65008 bytes from 192.168.0.2: seq=423 ttl=64 time=72.070 ms
65008 bytes from 192.168.0.2: seq=484 ttl=64 time=30.758 ms
65008 bytes from 192.168.0.2: seq=485 ttl=64 time=33.743 ms
65008 bytes from 192.168.0.2: seq=486 ttl=64 time=33.944 ms
65008 bytes from 192.168.0.2: seq=487 ttl=64 time=33.635 ms
65008 bytes from 192.168.0.2: seq=488 ttl=64 time=27.783 ms
65008 bytes from 192.168.0.2: seq=489 ttl=64 time=47.022 ms
65008 bytes from 192.168.0.2: seq=490 ttl=64 time=35.822 ms
65008 bytes from 192.168.0.2: seq=491 ttl=64 time=34.415 ms
65008 bytes from 192.168.0.2: seq=492 ttl=64 time=41.075 ms
65008 bytes from 192.168.0.2: seq=493 ttl=64 time=32.791 ms
65008 bytes from 192.168.0.2: seq=494 ttl=64 time=32.321 ms
65008 bytes from 192.168.0.2: seq=495 ttl=64 time=32.353 ms
65008 bytes from 192.168.0.2: seq=496 ttl=64 time=20.223 ms
65008 bytes from 192.168.0.2: seq=497 ttl=64 time=32.610 ms
65008 bytes from 192.168.0.2: seq=498 ttl=64 time=14.729 ms
65008 bytes from 192.168.0.2: seq=499 ttl=64 time=38.009 ms
65008 bytes from 192.168.0.2: seq=500 ttl=64 time=32.881 ms
65008 bytes from 192.168.0.2: seq=501 ttl=64 time=37.251 ms
65008 bytes from 192.168.0.2: seq=502 ttl=64 time=34.983 ms
^C
--- 192.168.0.2 ping statistics ---
543 packets transmitted, 143 packets received, 73% packet loss
round-trip min/avg/max = 9.185/36.762/130.518 ms
/ #

```

Figura 76: Comparativa de los pings enviados frente a los recibidos.

Como se puede ver de 543 *pings* que se han intentado enviar, solo 143 han llegado a su destino, lo que supone una reducción muy elevada del ancho de banda consumido en la red.

Para finalizar este apartado, comprobamos el funcionamiento del parámetro configurable DATA_LIMIT que limita el máximo de datos permitidos. Tal y como se puede ver en la Figura 77, primero mostramos los parámetros presentes y, a continuación, modificamos el valor de DATA_LIMIT a 20000 bytes, lo que supone una reducción a 1/5 del valor original.

```

alpinemodificada-ssh-1

onos@root > cfg get org.onosproject.detectHostBan.DetectHostBan
org.onosproject.detectHostBan.DetectHostBan
  name=DATA_LIMIT, type=integer, value=1000000, defaultValue=1000000, description=Límite de datos permitido en cada enlace
  name=TIME_BAN, type=integer, value=60, defaultValue=60, description=Tiempo que esta baneado un enlace cuando supera el límite de datos establecido
onos@root > cfg set org.onosproject.detectHostBan.DetectHostBan DATA_LIMIT 20000
onos@root > 

```

Figura 77: Modificación de los parámetros configurables.

Como podemos ver en la Figura 77, se notifica el cambio del parámetro al valor introducido, y se observa que los datos restantes se han modificado también en función del valor de la variable.

Figura 78: Comprobación de la modificación de los parámetros.

Asimismo, cuando se superan los 20000 bytes ocurre lo mismo que antes, el baneo se produce correctamente y no se permite ni el envío ni la recepción de dato alguno.

Onos-1

```

17:12:31.203 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
17:12:31.205 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
17:12:31.211 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
17:12:31.261 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 16088
17:12:31.261 INFO [DetectHostBan] Quedan disponibles: 3912 datos para dicho host
17:12:31.262 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 16088
17:12:31.262 INFO [DetectHostBan] Quedan disponibles: 3912 datos para dicho host
17:12:41.202 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
17:12:41.204 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
17:12:41.212 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
17:12:41.262 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 17340
17:12:41.262 INFO [DetectHostBan] Quedan disponibles: 2660 datos para dicho host
17:12:41.263 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 17340
17:12:41.263 INFO [DetectHostBan] Quedan disponibles: 2660 datos para dicho host
17:12:51.200 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
17:12:51.205 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
17:12:51.213 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
17:12:51.264 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 18592
17:12:51.264 INFO [DetectHostBan] Quedan disponibles: 1408 datos para dicho host
17:12:51.265 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 18592
17:12:51.265 INFO [DetectHostBan] Quedan disponibles: 1408 datos para dicho host
17:13:01.200 INFO [DetectHost] Number of connected devices is: 1 and hosts 2
17:13:01.204 INFO [DetectHost] Host 1 whose MAC is: 00:00:00:00:01 is on port: 1
17:13:01.210 INFO [DetectHost] Host 2 whose MAC is: 00:00:00:00:02 is on port: 2
17:13:01.265 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:01 son: 20116
17:13:01.266 INFO [DetectHostBan] Has superado por 116 bytes el total de datos permitido
17:13:01.280 ERROR [DetectHostBan] Baneo aplicado a la MAC: 00:00:00:00:01
17:13:01.284 INFO [DetectHostBan] Bytes gastados totales para la MAC 00:00:00:00:02 son: 20116
17:13:01.284 INFO [DetectHostBan] Has superado por 116 bytes el total de datos permitido
17:13:01.291 ERROR [DetectHostBan] Baneo aplicado a la MAC: 00:00:00:00:02

```

Figura 79: Comprobación del baneo con los nuevos parámetros.

5.4 Aplicación VLAN

5.4.1 Motivación y explicación teórica

La siguiente aplicación desarrollada permite asignar diferentes VLAN a los *hosts*. Una red de área local virtual (VLAN, *Virtual Local Area Network*) es una forma de crear redes lógicas independientes dentro de una misma red física [31]. Es decir, poder tener diferentes redes dentro de un mismo enrutador. Esto permite simplificar la administración de la red ya que permite separar los *hosts* que no deseamos que comparten el dominio de colisión. Por tanto, en resumen, las ventajas que aporta son las siguientes:

- Mayor flexibilidad, ya que se facilita el cambio y movimiento 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 dispositivos están separados en diferentes VLAN, por tanto, su comunicación está más 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.

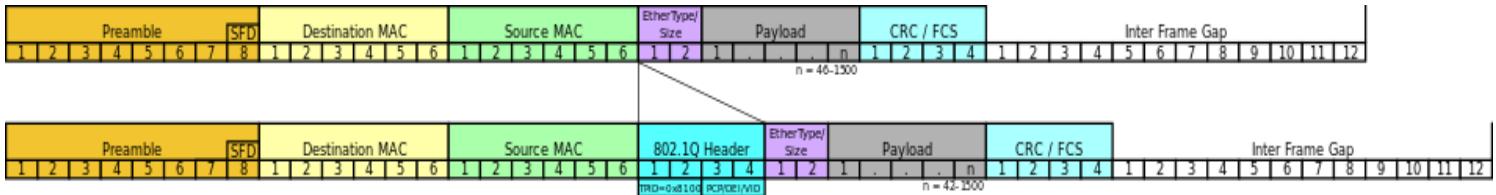


Figura 80: Diferencias entre una trama Ethernet sin etiquetado VLAN (802.1Q) y otra con él.

La propuesta principal de ésta es añadir 4 bytes a la trama *Ethernet*, en lugar de encapsularla, y manteniendo el resto de la trama igual. En la Figura 80 se puede observar en primer lugar, el formato de una trama Ethernet y en la parte inferior el formato de una trama que incluye la cabecera 802.1Q.

Seguidamente, en la Figura 81, se muestra con más detalle los bytes que se añaden:

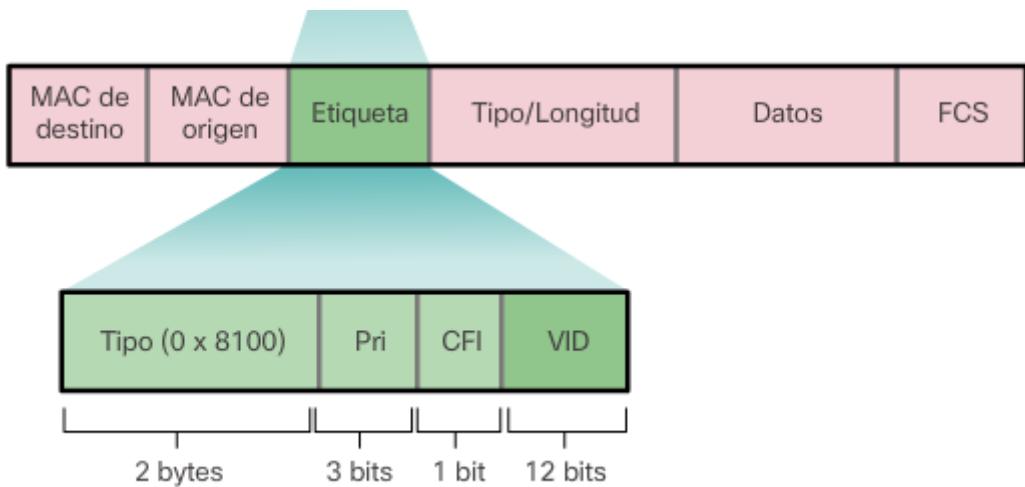


Figura 81: Detalles acerca de la trama 802.1Q.

- Tipo: Valor de 2 bytes cuyo valor por defecto se establece en 0x8100 en hexadecimal.
- Prioridad de usuario (Pri): Valor de 3 bits que admite la implementación de nivel o calidad de servicio.
- Identificador de formato canónico (CFI): Identificador de 1 bit que habilita las tramas Token Ring que se van a transportar a través de los enlaces Ethernet.
- ID de VLAN (VID): Conjunto de 12 bits que forman el número de identificación de la VLAN.

Este protocolo permite identificar una trama perteneciente a una VLAN. De esta forma, el tráfico se mandará únicamente por su VLAN. Para interconectar VLAN se añadió un tipo de puerto en los *switches*. Por un lado, están los puertos de acceso. Éstos son a los que se conectan 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 (sin etiquetar).

Por otro lado, están los puertos troncales o *trunk ports*, 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.

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

5.4.2 Componentes de la aplicación

Esta aplicación consta de 4 ficheros. Un fichero principal, en el que se encuentra el código de la aplicación y otros 3 en los que se implementan comandos para la consola CLI de ONOS que ayudan a trabajar con la aplicación.

5.4.2.1 Fichero VlanByMac.java

El código de esta aplicación se va a ir explicando paulatinamente. Antes de nada, declaramos 3 HashMap que serán necesarios:

- macVlanMap: HashMap que relaciona la dirección 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 específicamente para este fin.
- vlanRuleMap: Este HashMap devuelve las reglas de flujo que tiene cada VLAN creada. Más adelante se verá su finalidad.
- macRuleMap: Este HashMap devuelve para cada dirección MAC todas las reglas de flujo que tiene asociadas.

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

Seguidamente, añadimos también un *HostListener*, que tendrá como objetivo notificarnos cuando se añade un *host* nuevo a la red.

Para que este *HostListener* funcione, es necesario activar previamente las aplicaciones *detectHost* y *hostprobingprovider*, ya que sino la siguiente línea no tendría funcionalidad:

```
event.type() == HostEvent.Type.HOST_ADDED
```

5.4.2.1.1 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, obteniéndola a partir del HashMap *macVlanMap*, incluyendo una excepción para el caso de que esa MAC no tenga asociada en cuyo caso se notifica, no se crea regla de flujo alguna y, por tanto, dicho *host* no puede enviar tráfico.

En el caso de que sí 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 conectado. Sólo 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 también que se haga una transición a la tabla 2 cuyo contenido se verá posteriormente.
- Si el valor de la VLAN 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 que, en el momento en el que el *host* se desconecte, poder 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 puertos 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.

5.4.2.1.2 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 también aquella referida al tráfico *broadcast*, por tanto, es necesario volverla a crear, pero esta vez sin incluir el *host* que acabamos de eliminar. El procedimiento para realizar esta regla es idéntico al explicado en la sección anterior.

5.4.2.2 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 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 codifica el 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 a la que queremos que pertenezca.

Un ejemplo de uso podría ser el siguiente:

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

En este caso se le asigna, 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.

5.4.3 Banco de pruebas

5.4.3.1 Prueba de la aplicación sin utilizar comandos ni routers

En primer lugar, se va a probar el funcionamiento de la aplicación entre *hosts*. Para ello, previamente hemos definido en el código unas VLAN asociadas a los 4 *hosts* de la siguiente forma:

```
macVlanMap.put(MacAddress.valueOf("00:00:00:00:00:01"),VlanId.vlanId((short)1)
macVlanMap.put(MacAddress.valueOf("00:00:00:00:00:02"),VlanId.vlanId((short)1)
macVlanMap.put(MacAddress.valueOf("00:00:00:00:00:03"),VlanId.vlanId((short)2)
macVlanMap.put(MacAddress.valueOf("00:00:00:00:00:04"),VlanId.vlanId((short)2))
```

Tal y como se puede ver, los *hosts* 1 y 2 tendrán asociada la VLAN 1, por tanto, podrán comunicarse entre ellos, mientras que los *hosts* 3 y 4 tendrán asociada la VLAN 2 y, por tanto, también sólo podrán comunicarse entre ellos.

Para comprobarlo, activamos todas las aplicaciones necesarias que se pueden ver en la Figura 82.

```

Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^J'.
alpinemodificada-ssh-1 console is now available... Press RETURN to get started.
udhcpc (v1.24.2) started
Sending discover...
Sending discover...
Sending discover...
onos@root > app activate org.onosproject.drivers.ovsdb
Activated org.onosproject.drivers.ovsdb
onos@root > app activate org.onosproject.ovsdb
Activated org.onosproject.ovsdb
onos@root > app activate org.onosproject.openflow
Activated org.onosproject.openflow
onos@root > app activate org.onosproject.fwd
Activated org.onosproject.fwd
onos@root > app activate org.onosproject.detectHost
Activated org.onosproject.detectHost
onos@root > app activate org.onosproject.hostprobingprovider
Activated org.onosproject.hostprobingprovider
onos@root >

```

Figura 82: Aplicaciones necesarias para el funcionamiento de VLAN.

El siguiente paso es enviar *pings* entre los *hosts* de la misma VLAN, que deberían llegar a su destino correctamente tal y como se observa en la Figura 83

```

AlpineLinux-1
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^J'.
AlpineLinux-1 console is now available... Press RETURN to get started.
/ #
/ # ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: seq=0 ttl=64 time=125.614 ms
64 bytes from 192.168.0.2: seq=1 ttl=64 time=2.284 ms
64 bytes from 192.168.0.2: seq=2 ttl=64 time=1.645 ms
64 bytes from 192.168.0.2: seq=3 ttl=64 time=2.345 ms
64 bytes from 192.168.0.2: seq=4 ttl=64 time=2.478 ms
64 bytes from 192.168.0.2: seq=5 ttl=64 time=1.794 ms
64 bytes from 192.168.0.2: seq=6 ttl=64 time=1.796 ms
64 bytes from 192.168.0.2: seq=7 ttl=64 time=0.448 ms
64 bytes from 192.168.0.2: seq=8 ttl=64 time=1.857 ms
64 bytes from 192.168.0.2: seq=9 ttl=64 time=1.737 ms
64 bytes from 192.168.0.2: seq=10 ttl=64 time=1.736 ms
64 bytes from 192.168.0.2: seq=11 ttl=64 time=0.908 ms
64 bytes from 192.168.0.2: seq=12 ttl=64 time=1.585 ms
[...]

```

```

AlpineLinux-3
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^J'.
AlpineLinux-3 console is now available... Press RETURN to get started.
/ #
/ # ping 192.168.0.4
PING 192.168.0.4 (192.168.0.4): 56 data bytes
64 bytes from 192.168.0.4: seq=0 ttl=64 time=43.775 ms
64 bytes from 192.168.0.4: seq=1 ttl=64 time=0.480 ms
64 bytes from 192.168.0.4: seq=2 ttl=64 time=0.627 ms
64 bytes from 192.168.0.4: seq=3 ttl=64 time=2.043 ms
64 bytes from 192.168.0.4: seq=4 ttl=64 time=0.652 ms
64 bytes from 192.168.0.4: seq=5 ttl=64 time=2.076 ms
64 bytes from 192.168.0.4: seq=6 ttl=64 time=1.898 ms
[...]

```

Figura 83: Envío de pings entre hosts de la misma VLAN.

A continuación, probamos el envío entre *hosts* de diferentes VLAN. Por ejemplo, que el *host 1* envíe al *host 3* y que el *host 2* envíe al *host 4*. Tal y como se puede ver en la Figura 84, no llegan, al pertenecer a diferentes subredes virtuales.

```

AlpineLinux-1
/ # ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
[...]

```

```

AlpineLinux-2
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^J'.
AlpineLinux-2 console is now available... Press RETURN to get started.
/ #
/ # ping 192.168.0.4
PING 192.168.0.4 (192.168.0.4): 56 data bytes
[...]

```

Figura 84: Envío de pings entre hosts de diferentes VLAN.

5.4.3.2 Prueba con el *router*

En esta prueba se va a utilizar un *router* que conecte las 2 subredes. Por tanto, la nueva red queda de la forma mostrada en la Figura 85.

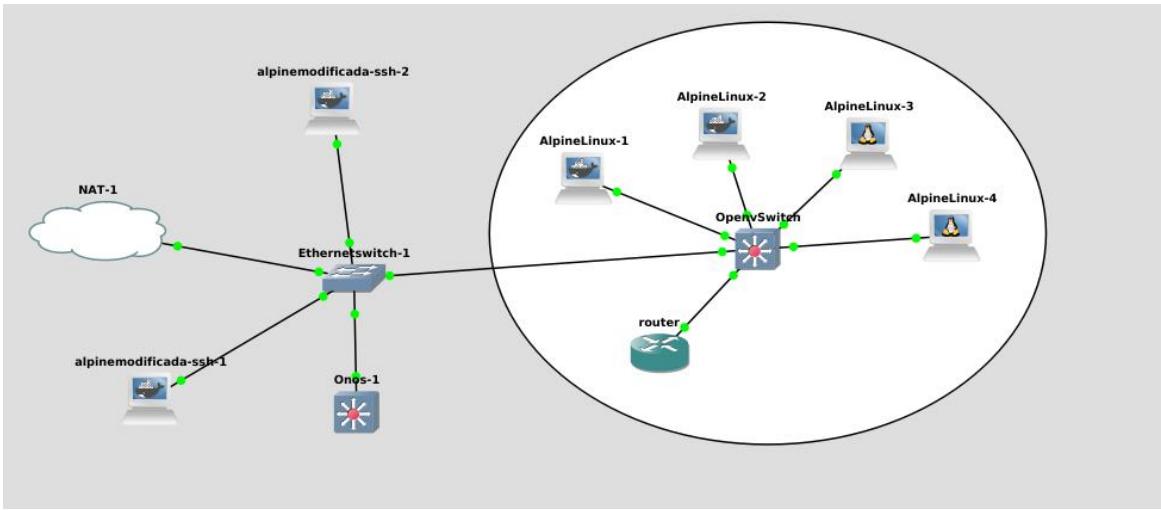


Figura 85: Red con el *router* añadido.

En primer lugar, hay que configurar el *router* añadido para que sea capaz de redirigir tráfico entre las 2 subredes. Para ello ejecutamos los siguientes comandos:

- `ifconfig br-lan down`
- `ip link add link eth0 name eth0.1 type vlan id 1`
- `ip link add link eth0 name eth0.2 type vlan id 2`
- `ifconfig eth0.1 192.168.0.254 up`
- `ifconfig eth0.2 192.168.1.254 up`
- `ifconfig eth0 hw ether 00:00:00:00:00:05`

Asimismo, también es necesario modificar la configuración de los *hosts* dejándola de la forma mostrada en la Figura 86.

Lo que se ha hecho es dividir los *hosts* en 2 subredes. Aquellos que pertenecen a la VLAN 1 se corresponde con la subred 192.168.0.x, mientras que aquellos que pertenecen a la VLAN 2 se corresponden con la subred 192.168.1.x. De la misma forma es necesario actualizar la pasarela (*gateway*) de cada *host* en el fichero de configuración (ver Figura 86).

```

#
# This is a sample network config uncomment lines to configure the network
#


# Static config for eth0
auto eth0
iface eth0 inet static
    address 192.168.1.4
    netmask 255.255.255.0
    gateway 192.168.1.254
#           up echo nameserver 192.168.0.1 > /etc/resolv.conf
hwaddress ether 00:00:00:00:00:04

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp

```

Figura 86: Nueva configuración de los hosts.

A la hora de añadir el *router* es necesario desactivar el *firewall*, ya que sin él no es capaz de interconectar las subredes existentes. Esto se hace de la siguiente forma:

- Modificando el fichero `/etc/config/firewall`, la parte que indica aceptar el tráfico *FORWARD*.
- Reiniciando el servicio a partir del comando: `/etc/init.d/firewall reload`

Una vez realizado este proceso, añadiremos un *router* que se conectará a un puerto troncal del *switch*. Este proceso lo haremos de idéntica forma al apartado anterior, cuando se añadieron los *hosts* manualmente desde el código (posteriormente se realizará mediante los comandos).

```
macVlanMap.put(MacAddress.valueOf("00:00:00:00:00:05"),VlanId.NONE);
```

Si ejecutamos el comando *ifconfig* en el *router* añadido, vemos que ésta es su MAC (Figura 87).

```

router
x

eth0.1      collisions:0 txqueuelen:1000
             RX bytes:50262 (49,0 KiB)  TX bytes:4971 (4,8 KiB)

eth0.1      Link encap:Ethernet HWaddr 0C:A5:CC:F3:9B:00
             inet addr:192.168.0.254 Bcast:192.168.0.255 Mask:255.255.255.0
             inet6 addr: fe80::ea5:ccff:fef3:9b00/64 Scope:Link
               UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:0 (0,0 B)  TX bytes:746 (746,0 B)

eth0.2      Link encap:Ethernet HWaddr 0C:A5:CC:F3:9B:00
             inet addr:192.168.1.254 Bcast:192.168.1.255 Mask:255.255.255.0
             inet6 addr: fe80::ea5:ccff:fef3:9b00/64 Scope:Link
               UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:0 (0,0 B)  TX bytes:746 (746,0 B)

eth1        Link encap:Ethernet HWaddr 0C:A5:CC:F3:9B:01
             inet6 addr: fe80::ea5:ccff:fef3:9b01/64 Scope:Link
               UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

```

Figura 87: Salida del comando `ifconfig` ejecutado en el router.

Finalmente, si ejecutamos el comando `route`, podemos ver cómo está la tabla de encaminamiento del *router*.

```

router
x

root@OpenWrt:/# ifconfig br-lan down
root@OpenWrt:/# route
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
192.168.0.0     *              255.255.255.0 U     0      0      0 eth0.1
192.168.1.0     *              255.255.255.0 U     0      0      0 eth0.2
root@OpenWrt:/# ■

```

Figura 88: Salida del comando `route`.

En origen salía una tercera entrada correspondiente a la interfaz *br-lan* sin embargo, en esta aplicación no es necesario y podría llegar a dar problemas con las entradas existentes, por tanto, es necesario quitar esa entrada. Es por ello por lo que fue necesario ejecutar el comando `ifconfig br-lan down`.

A continuación, vamos a empezar a enviar *pings* entre los *hosts* para comprobar el funcionamiento y capturaremos tráfico en el enlace que conecta el Open vSwitch con el *router* añadido para ver 2 cosas:

- Si pasa tráfico por el *router* cuando realizamos los *pings*.
- Si este tráfico va etiquetado con alguna VLAN o no.

Para ello, en primer lugar, probaremos a realizar *ping* entre 2 *hosts* pertenecientes a la misma VLAN, por ejemplo, entre los *hosts* Alpine-1 y Alpine-2 y capturaremos tráfico.

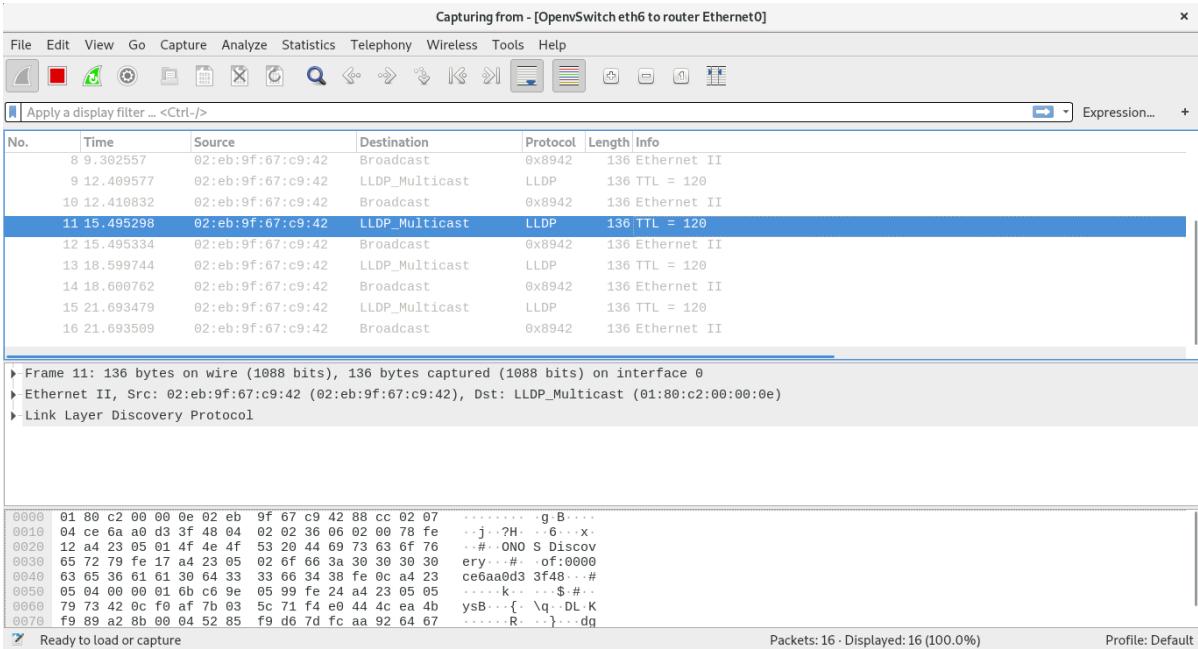


Figura 89: Captura de tráfico del router al enviar pings entre 2 hosts de la misma VLAN.

Tal y como se ve en la Figura 89, en este caso el tráfico no circula a través del *router*, sino que el Open vSwitch es capaz de redirigirlo sin problemas ya que pertenecen a la misma VLAN. Por tanto, hemos creado reglas de flujo que permiten encaminar el tráfico directamente.

Seguidamente, vamos a enviar tráfico entre 2 *hosts* no pertenecientes a la misma VLAN y nuevamente capturaremos tráfico en el mismo enlace para ver qué ocurre. Por ejemplo, desde el *host* Alpine-1 con destino Alpine-3 tal y como se observa en la Figura 90.

```

Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^]'.
AlpineLinux-1 console is now available... Press RETURN to get started.
/ #
/ # ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3): 56 data bytes
64 bytes from 192.168.1.3: seq=0 ttl=63 time=1020.016 ms
64 bytes from 192.168.1.3: seq=1 ttl=63 time=13.826 ms
64 bytes from 192.168.1.3: seq=2 ttl=63 time=4.224 ms
64 bytes from 192.168.1.3: seq=3 ttl=63 time=4.891 ms

```

Figura 90: Envío del ping entre hosts de 2 VLAN diferentes.

En el enlace se ha capturado el tráfico mostrado en la Figura 91. En ella se ve que el tráfico está circulando a través del *router* y que es éste el que lo redirige al destino.

De la misma forma, podemos observar en la parte inferior de la figura que este tráfico va etiquetado. En concreto, en *ICMP Request* lleva la VLAN 1, mientras que, como se observa en la Figura 92, el paquete *ICMP Reply* lleva etiqueta de la VLAN 2. Por tanto, la aplicación funciona correctamente.

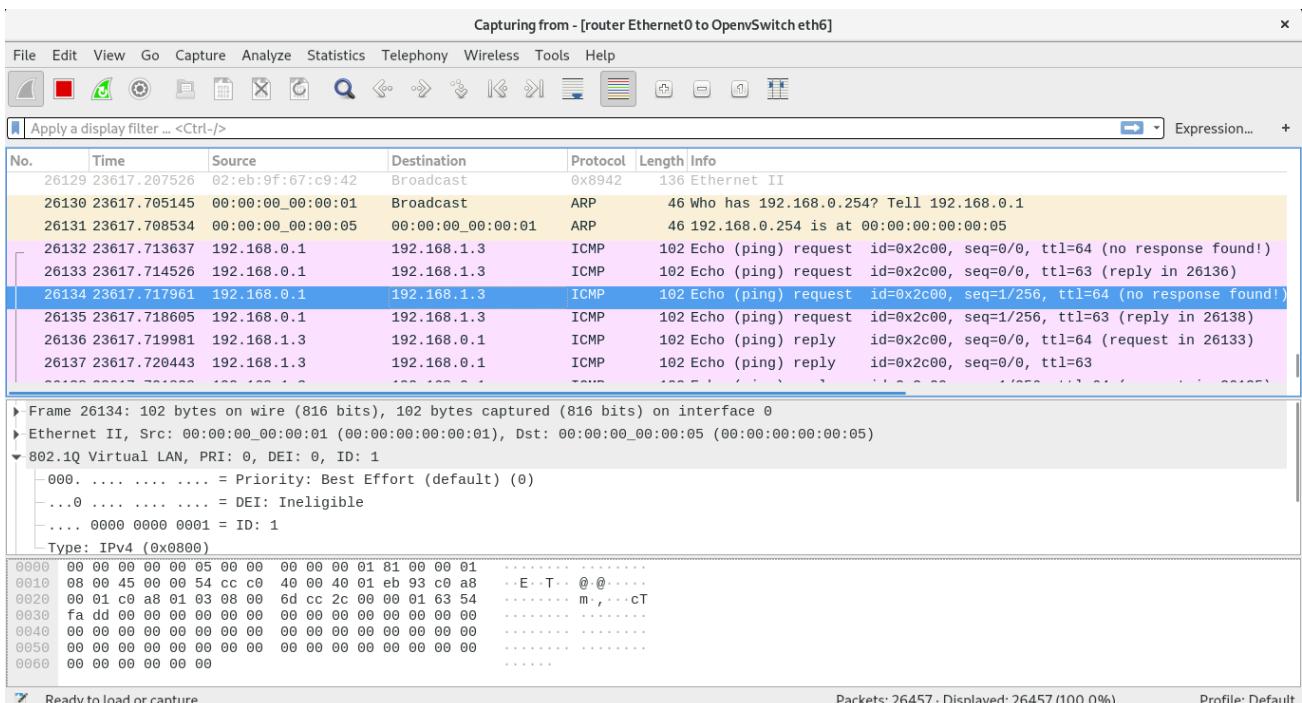


Figura 91: Captura de tráfico del router al enviar tráfico entre 2 hosts de diferentes VLAN.

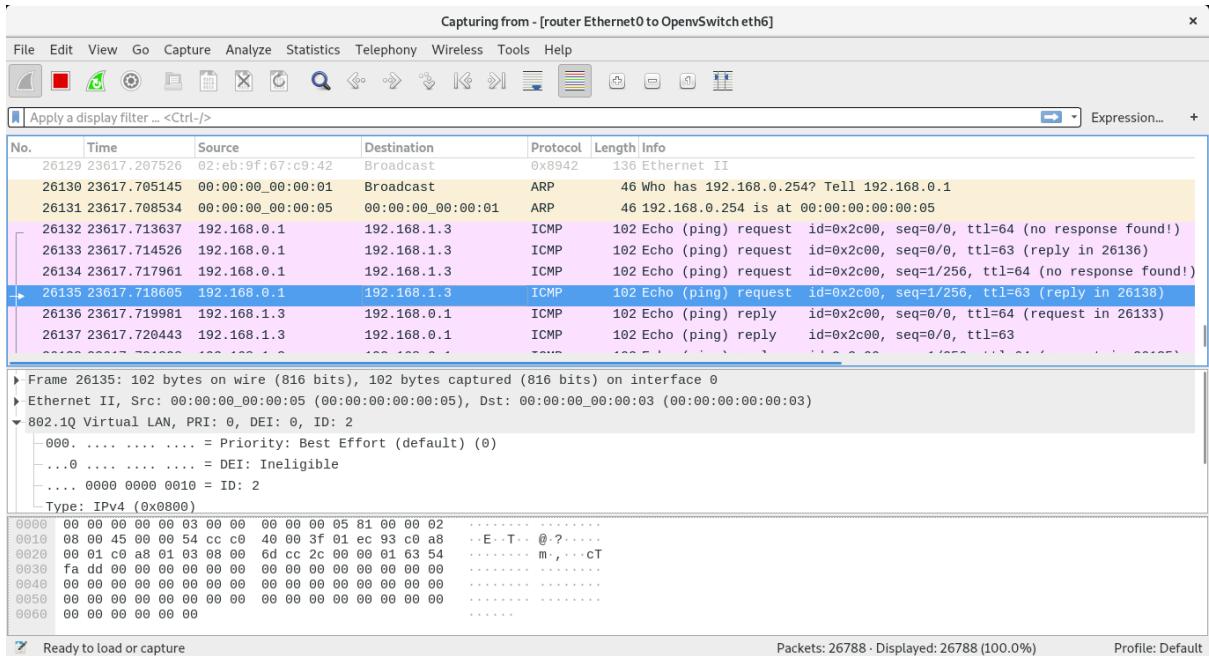


Figura 92: Captura de tráfico del router al enviar tráfico entre 2 hosts de diferentes VLAN.

5.4.3.3 Probando los comandos

En este apartado se va a probar el funcionamiento de los 3 comandos creados, que simplificarán el proceso de incorporación de *hosts*. Para ello, añadimos un nuevo *host*, Alpine-6 y recuperamos la red sin el router de interconexión de VLANs.

En primer lugar, mostramos las correspondencias que tenemos creadas en estos momentos con el comando *show-Vlan-Mac*.

```
alpinemodificada-ssh-1
onos@root > show-Vlan-Mac
macVlanMap es: {00:00:00:00:00:01=1, 00:00:00:00:00:02=1, 00:00:00:00:00:03=2
00:00:00:00:00:04=2, 00:00:00:00:00:05=None}
onos@root >
```

Figura 93: Salida del comando *show-Vlan-Mac*.

Tal y como se aprecia en la Figura 93, se ve la relación entre las MAC de los dispositivos que tenemos y las VLAN que contienen.

A continuación, añadimos al Alpine-6 la VLAN 1, por ejemplo, mediante el uso del comando *Add-Mac-Vlan*.

```
onos@root > add-Mac-Vlan 00:00:00:00:00:06 1
MacAddress-----Vlan
00:00:00:00:00:06      1
onos@root >
```

Figura 94: Salida del comando add-Mac-Vlan.

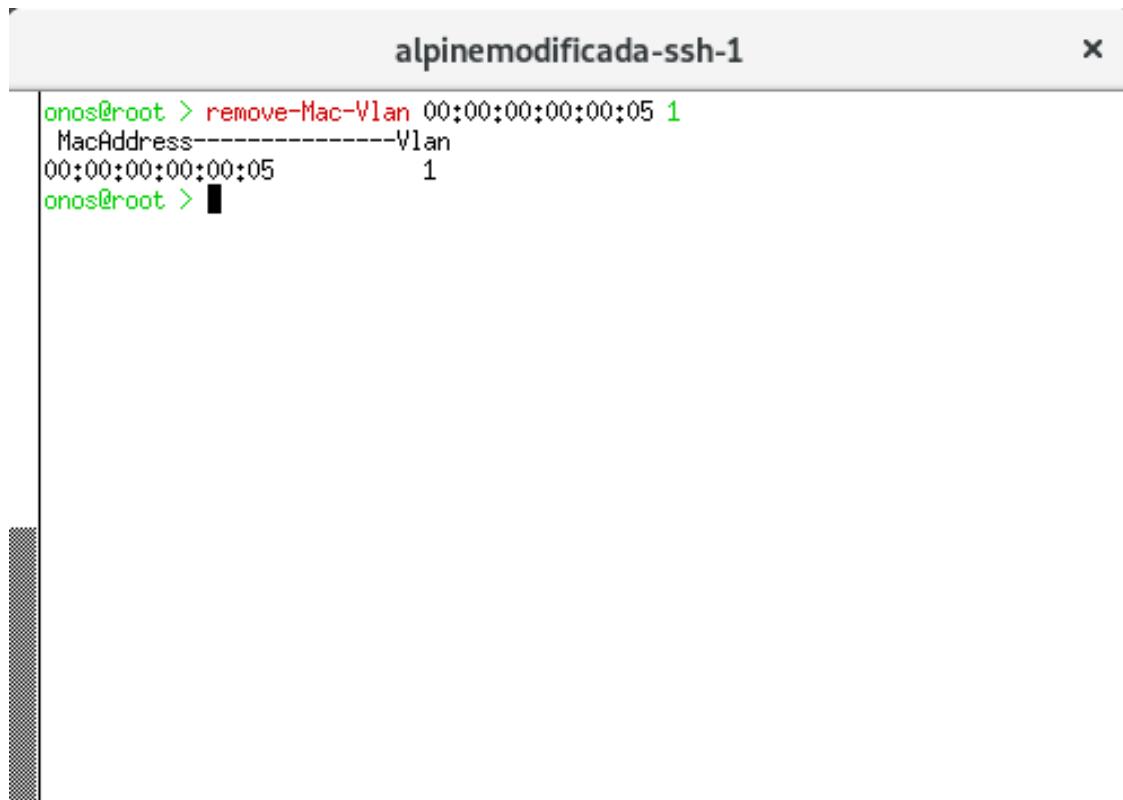
Como se puede observar, en la Figura 94, se ha introducido para la MAC :06 la VLAN 1. Por tanto, ahora, desde los *hosts* 1 y 2 podremos enviar tráfico, pero no desde los 4 y 5. Lo comprobamos:

```
Trying 192.168.223.128...
Connected to 192.168.223.128.
Escape character is '^]'.
AlpineLinux-6 console is now available... Press RETURN to get started.
/ #
/ # ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: seq=0 ttl=64 time=955.238 ms
64 bytes from 192.168.0.1: seq=1 ttl=64 time=1.100 ms
64 bytes from 192.168.0.1: seq=2 ttl=64 time=1.708 ms
64 bytes from 192.168.0.1: seq=3 ttl=64 time=1.796 ms
64 bytes from 192.168.0.1: seq=4 ttl=64 time=0.849 ms
64 bytes from 192.168.0.1: seq=5 ttl=64 time=1.793 ms
64 bytes from 192.168.0.1: seq=6 ttl=64 time=1.026 ms
64 bytes from 192.168.0.1: seq=7 ttl=64 time=1.631 ms
```

Figura 95: Envío de pings con el host añadido.

Tal y como podemos comprobar en la Figura 95 los *pings* llegan correctamente, lo que indica que se ha añadido bien el *host*.

A continuación, probaremos el comando restante y borraremos el *host* que acabamos de crear.



```
alpinemodificada-ssh-1
onos@root > remove-Mac-Vlan 00:00:00:00:00:05 1
MacAddress-----Vlan
00:00:00:00:00:05           1
onos@root >
```

Figura 96: Eliminación de la VLAN del host.

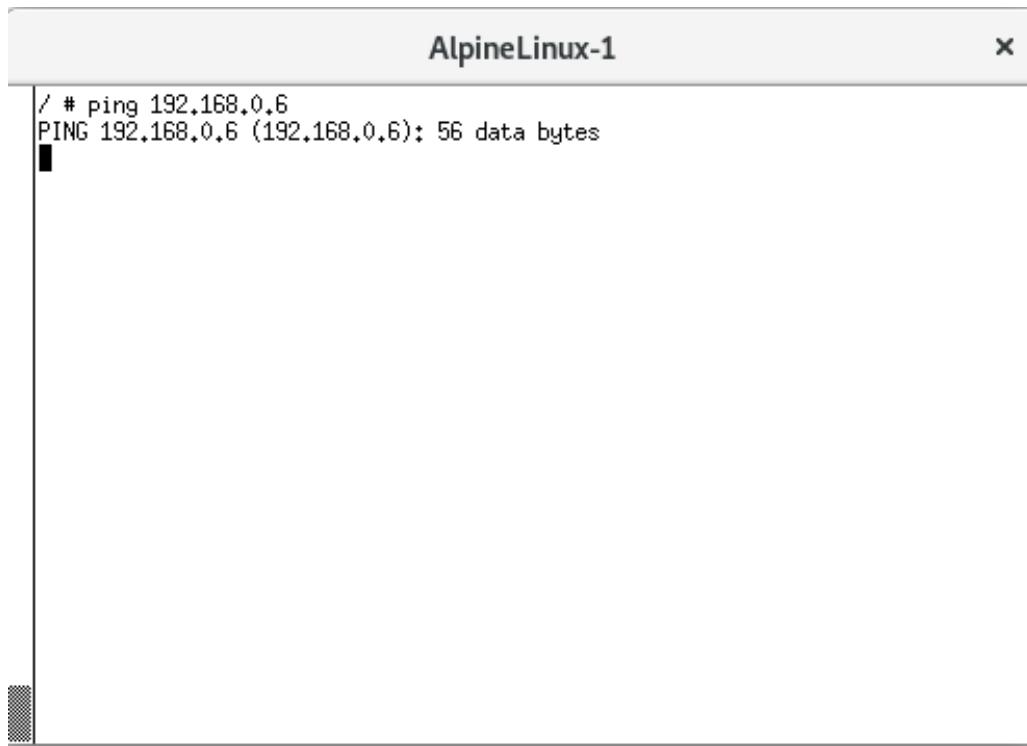
Si ejecutamos el comando anterior, podemos comprobar que del HashMap se ha eliminado el *host*.



```
alpinemodificada-ssh-1
onos@root > show-Vlan-Mac
macVlanMap es: {00:00:00:00:00:01=1, 00:00:00:00:00:02=1, 00:00:00:00:00:03=2, 00:00:00:00:00:04=2, 00:00:00:00:00:05=None}
onos@root >
```

Figura 97: Ejecución del comando *show-Vlan-Mac*.

A continuación, probamos a enviar desde el *host* Alpine-1, que antes pertenecía a la misma VLAN. Como podemos ver en la Figura 98 no llegan, por lo tanto, este comando también funciona correctamente.



The screenshot shows a terminal window titled "AlpineLinux-1". The terminal displays the following command and its output:

```
/ # ping 192.168.0.6
PING 192.168.0.6 (192.168.0.6): 56 data bytes
```

Figura 98: Comprobación de que los pings no llegan al eliminar el host.

5.5 Aplicación *fwdBalanceo*

5.5.1 Motivación y explicación teórica

Esta aplicación es diferente, puesto que es una modificación de la app *fwd* que contiene el controlador ONOS por defecto. El objetivo es realizar un balanceo de carga cuando tenemos varios enlaces que conectan 2 *switches* Open vSwitch. Para ello, en primer lugar, modificamos la red que tenemos en el GNS3 sustituyéndola por la mostrada en la Figura 99.

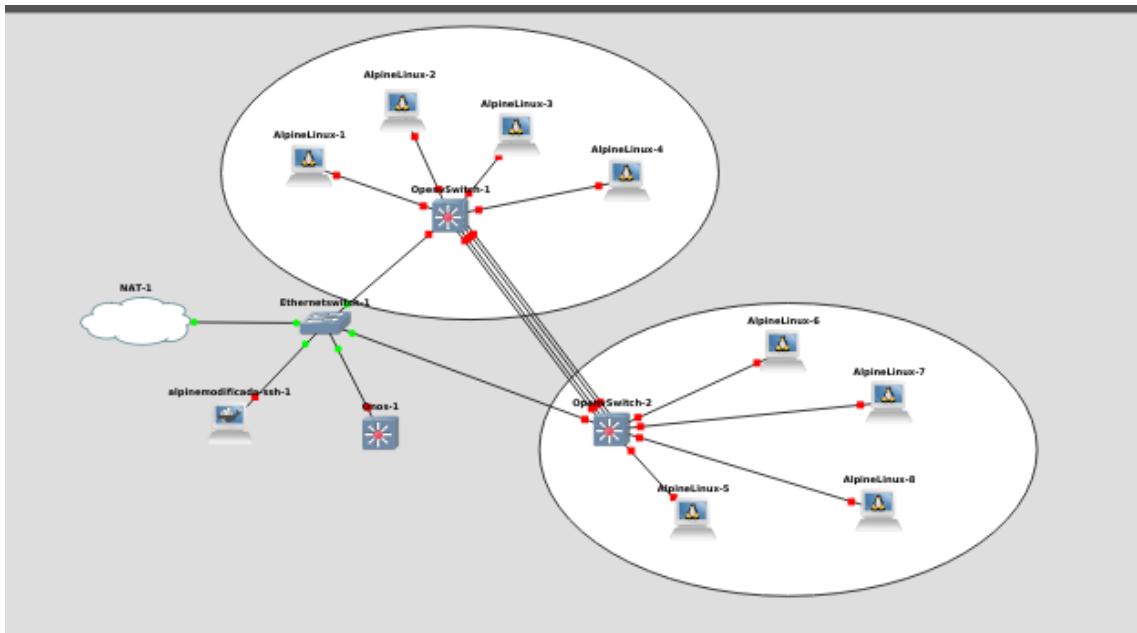


Figura 99: Red utilizada para la aplicación *fwdBalanceo*.

Como vemos, tenemos 2 *switches* del tipo Open vSwitch conectados por 4 enlaces, que será por 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 *onos/app* que copiaremos y llevaremos a nuestro directorio de trabajo. Seguidamente modificamos el fichero *pom.xml* para cambiar el nombre de la aplicación dejándolo de la forma que se ve en la Figura 100.

```

-
-      http://www.apache.org/licenses/LICENSE-2.0
-
- Unless required by applicable law or agreed to in writing, software
- distributed under the License is distributed on an "AS IS" BASIS,
- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
- See the License for the specific language governing permissions and
- limitations under the License.
-->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/
POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.onosproject</groupId>
<artifactId>fwdBalanceo</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>bundle</packaging>

<description>ONOS OSGi bundle archetype</description>
<url>http://onosproject.org</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <onos.version>2.0.0</onos.version>
    <onos.app.name>org.onosproject.fwdBalanceo</onos.app.name>
    <onos.app.title>Aplicación auxiliar para balanceoSwitch</onos.app.title>
    <onos.app.origin>UVA</onos.app.origin>
    <onos.app.category>default</onos.app.category>
    <onos.app.url>http://onosproject.org</onos.app.url>
    <onos.app.readme>ONOS OSGi bundle archetype.</onos.app.readme>
</properties>

<dependencies>
    <dependency>
        <groupId>org.onosproject</groupId>
        <artifactId>onos-api</artifactId>
        <version>${onos.version}</version>
    
```

Figura 100: Modificaciones del fichero *pom.xml*.

Finalmente, modificamos el nombre de las carpetas que tienen el nombre de la aplicación sustituyéndolas 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*.

A continuación, se va a detallar la explicación acerca de cómo realizar el balanceo de carga [32].

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 ésta), 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.

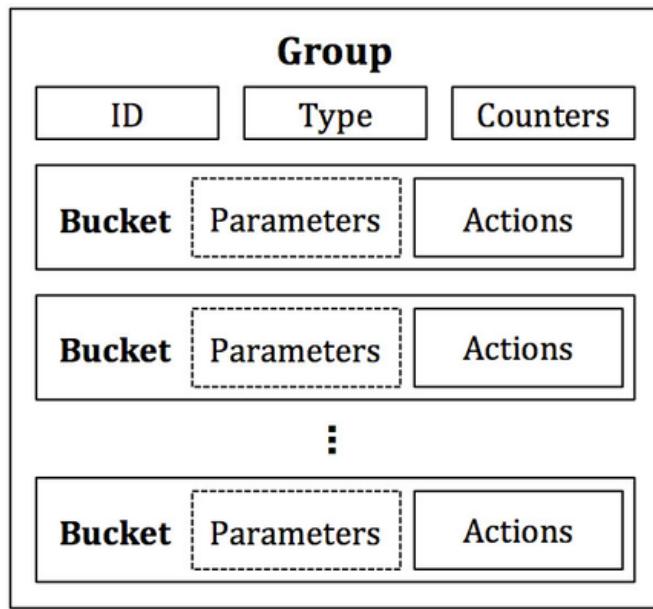


Figura 101: Grupos en OpenFlow.

Tal y como se ve, 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 en cada *bucket* del grupo. 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 algoritmo de selección depende de la implementación del *switch*. En el apartado 5.7.3 se indica cómo se realiza la selección de los *buckets* del grupo para el *switch* que manejamos, ya que ésta 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 difícil 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 hay que indicar que sólo se puede usar un *bucket* y que éste no cambia, a menos que se detecte que la vida del mismo desciende.

5.5.2 Componentes de la aplicación

5.5.2.1 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 modificarlos en caso de que se modifique la topología de la red, 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 vSwitch. Depende de la versión del *switch* utilizado, hay que fijarse si a la hora de programar hay que sumar 1 a estos valores ya que, en función de la versión, 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 dg = new DefaultGroup(new GroupId(137), degd);`

A su vez, un *DefaultGroupDescription* tiene varios constructores. Usaremos el utilizado para las aplicaciones (*northbound API*). 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(puerto1))
 .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.

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 troncal.

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

$$\text{bitsPorSegundo} = \text{bitsPorSegundo} + \frac{\text{traffic.bytesSent()} * 8}{\text{temporizadorTarea}}$$

Para añadir *buckets* simplemente creamos un nuevo tratamiento con el puerto correspondiente que esté almacenado en un nuevo *GroupBucket* y lo añadiremos al *ArrayList* ya creado. Por ejemplo, para añadir el segundo puerto hacemos lo siguiente:

- ```
TrafficTreatment sendPort2 = DefaultTrafficTreatment.builder()
    .setOutput(PortNumber.portNumber(puerto2)).build();
```
- ```
GroupBucket gb2=DefaultGroupBucket.createSelectGroupBucket(sendPort2)
```
- ```
listGroup.add(gb2);
```

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*.

5.5.2.2 Método *installRule*

Dentro de este método, el objetivo es crear una regla que indique que 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 envíe por el grupo de la siguiente forma:

```
if(portNumber.toLong() == puerto1 || portNumber.toLong() == puerto2 ||
    portNumber.toLong() == puerto3 || portNumber.toLong() == puerto4) {
```

```
    TrafficTreatment treatment = DefaultTrafficTreatment.builder()
        .group(new GroupId(137))
        .build();
```

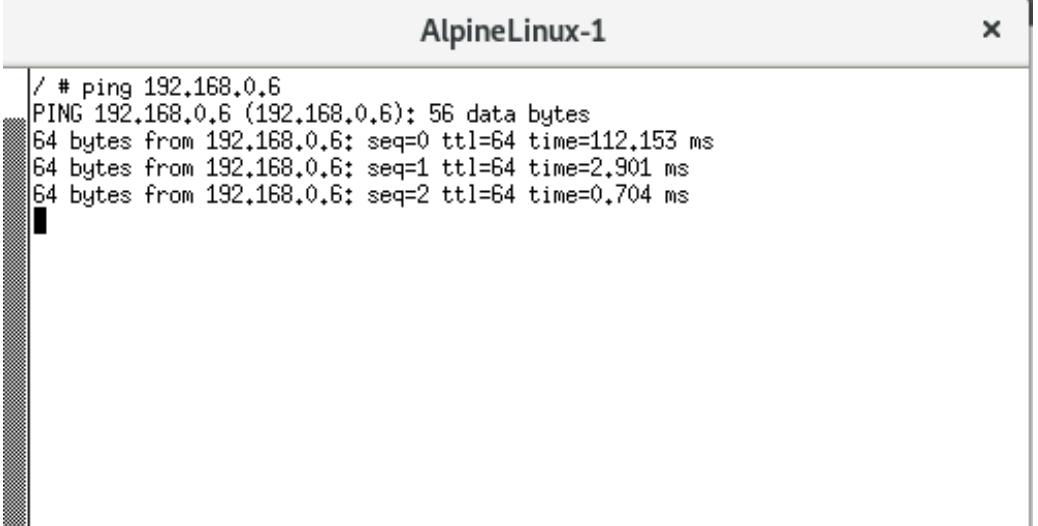
Esto, como se ha comentado, lo que hace es enviar por el grupo previamente creado y que sea éste el que escoja según el tipo y los *buckets* que contenga.

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

5.5.3 Banco de pruebas

En este apartado se va a probar el funcionamiento de la aplicación. Para ello se van a enviar diferentes *pings* entre las 2 subredes y comprobar si se realiza el balanceo correctamente.

Por ejemplo, enviamos un *ping* entre Alpine-1 y Alpine-6.



```
AlpineLinux-1
/ # ping 192.168.0.6
PING 192.168.0.6 (192.168.0.6): 56 data bytes
64 bytes from 192.168.0.6: seq=0 ttl=64 time=112.153 ms
64 bytes from 192.168.0.6: seq=1 ttl=64 time=2.901 ms
64 bytes from 192.168.0.6: seq=2 ttl=64 time=0.704 ms
```

Figura 102: Envío de los pings entre los 2 Open vSwitch.

Vemos como el *ping* efectivamente llega, sin embargo, falta comprobar que se esté enviando por el grupo que hemos creado, así que vamos a la CLI de ONOS y ejecutamos el comando *groups*:



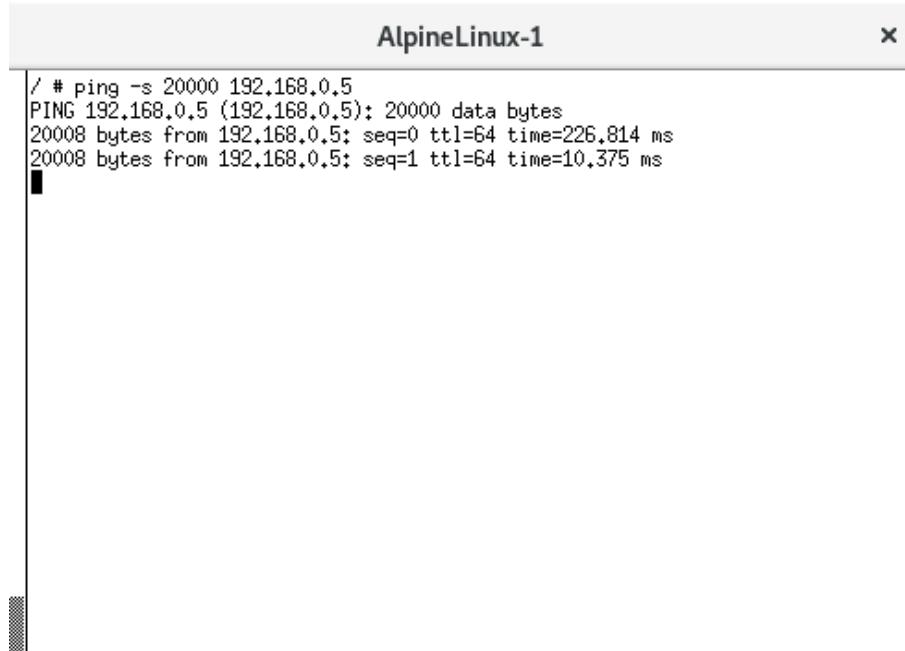
```
alpinemodificada-ssh-1
onos@root > groups
deviceId=of:00001e4d5fd35d4c, groupCount=1
    id=0x89, state=ADDED, type=SELECT, bytes=648, packets=7, appId=org.onosproject.fwdBalanceo
    , referenceCount=1
        id=0x89, bucket=1, bytes=648, packets=7, weight=1, actions=[OUTPUT:11]
deviceId=of:00009a50cf463946, groupCount=1
    id=0x89, state=ADDED, type=SELECT, bytes=844, packets=9, appId=org.onosproject.fwdBalanceo
    , referenceCount=1
        id=0x89, bucket=1, bytes=844, packets=9, weight=1, actions=[OUTPUT:11]
onos@root >
```

Figura 103: Salida del comando *groups*.

Como se observa en la Figura 103, los *buckets* están recibiendo tráfico, lo que indica que los *pings* realizados se están enviando a través de los grupos implementados de forma correcta.

Además, como el tamaño de un *ping* es muy pequeño (64 bytes) sólo se activa el primer *bucket*, por tanto, todo el tráfico va únicamente por 1 de los 4 puertos.

Por tanto, el siguiente paso consiste en enviar un *ping* de más tamaño, por ejemplo, el mostrado en la Figura 104, para comprobar el funcionamiento y ver si se añaden correctamente los *buckets* al grupo.



The terminal window is titled "AlpineLinux-1". The command entered is "/ # ping -s 20000 192.168.0.5". The output shows two ping responses: "PING 192.168.0.5 (192.168.0.5): 20000 data bytes" and "20008 bytes from 192.168.0.5: seq=0 ttl=64 time=226.814 ms", followed by "20008 bytes from 192.168.0.5: seq=1 ttl=64 time=10.375 ms".

Figura 104: Envío de pings de mayor tamaño.

Ejecutamos el comando *groups* al igual que en el caso anterior. La salida se puede ver en la Figura 105, en la que se ve como efectivamente, se han creado los 4 *buckets*. El controlador ONOS, tal como se puede ver en la Figura 106, también indica que los puertos se han ido añadiendo, dado que el tráfico circulante por la red superaba los umbrales predefinidos.

```
alpinemodificada-ssh-1
x
onos@root > groups
deviceId=of:00001e4d5fd35d4c, groupCount=1
    id=0x89, state=ADDED, type=SELECT, bytes=163872, packets=112, appId=org.onosproject.fwdBalanceo, referenceCount=1
        id=0x89, bucket=1, bytes=0, packets=0, weight=1, actions=[OUTPUT:11]
        id=0x89, bucket=2, bytes=0, packets=0, weight=1, actions=[OUTPUT:12]
        id=0x89, bucket=3, bytes=0, packets=0, weight=1, actions=[OUTPUT:13]
        id=0x89, bucket=4, bytes=163872, packets=112, weight=1, actions=[OUTPUT:14]
deviceId=of:00009a50cf463946, groupCount=1
    id=0x89, state=ADDED, type=SELECT, bytes=204840, packets=140, appId=org.onosproject.fwdBalanceo, referenceCount=1
        id=0x89, bucket=1, bytes=0, packets=0, weight=1, actions=[OUTPUT:11]
        id=0x89, bucket=2, bytes=0, packets=0, weight=1, actions=[OUTPUT:12]
        id=0x89, bucket=3, bytes=0, packets=0, weight=1, actions=[OUTPUT:13]
        id=0x89, bucket=4, bytes=204840, packets=140, weight=1, actions=[OUTPUT:14]
onos@root >
```

Figura 105: Muestra de los grupos creados.

```
Onos-1
x
19:23:54,835 INFO [ReactiveForwarding] Unable to read portStats
19:23:54,839 INFO [ReactiveForwarding] Bits: 83467
19:23:54,856 WARN [ReactiveForwarding] Anadido puerto 2
19:23:54,857 WARN [ReactiveForwarding] Anadido puerto 3
19:23:54,859 WARN [ReactiveForwarding] Anadido puerto 4
19:23:54,868 WARN [ReactiveForwarding] Device id of:00009a50cf463946
19:23:54,869 INFO [ReactiveForwarding] Unable to read portStats
19:23:54,869 INFO [ReactiveForwarding] Bits: 83250
19:23:54,869 WARN [ReactiveForwarding] Anadido puerto 2
19:23:54,869 WARN [ReactiveForwarding] Anadido puerto 3
19:23:54,870 WARN [ReactiveForwarding] Anadido puerto 4
19:24:14,832 WARN [ReactiveForwarding] Device id of:00001e4d5fd35d4c
19:24:14,834 INFO [ReactiveForwarding] Unable to read portStats
19:24:14,840 INFO [ReactiveForwarding] Bits: 83027
19:24:14,844 WARN [ReactiveForwarding] Anadido puerto 2
19:24:14,847 WARN [ReactiveForwarding] Anadido puerto 3
19:24:14,857 WARN [ReactiveForwarding] Anadido puerto 4
19:24:14,867 WARN [ReactiveForwarding] Device id of:00009a50cf463946
19:24:14,867 INFO [ReactiveForwarding] Unable to read portStats
19:24:14,868 INFO [ReactiveForwarding] Bits: 83298
19:24:14,868 WARN [ReactiveForwarding] Anadido puerto 2
19:24:14,868 WARN [ReactiveForwarding] Anadido puerto 3
19:24:14,869 WARN [ReactiveForwarding] Anadido puerto 4
```

Figura 106: Vista del controlador de los buckets añadidos al grupo.

Sin embargo, tal y como se puede ver, únicamente se envía por 1 de los 4 puertos, a pesar de que los *buckets* están bien creados. Esto se debe a que, por la forma en la que están implementados los grupos, internamente, se realiza un *HashMap* en el que se asigna 1 de todos los puertos disponibles en función de parámetros tales como la MAC origen o destino, y, para esa conexión siempre se va a usar el mismo *bucket*. Si hacemos otro *ping* entre 2 *hosts* distintos se realizaría otra entrada de *HashMap* según la cual se elegiría otro *bucket* (o podría ser el mismo) de entre los disponibles [33].

Durante las pruebas realizadas se realizaron *pings* con todas las combinaciones posibles entre las 2 redes creadas para comprobar esto resultando que efectivamente, cada conexión cogía 1 *bucket* y siempre se envía por el mismo.

5.6 Aplicación FakeDHCP

5.6.1 Explicación teórica

En primer lugar, se va a explicar el funcionamiento del protocolo DHCP (*Dynamic Host Configuration Protocol*). Éste 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 solicite otra.

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

- 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 los parámetros, tales como el 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 servidor tiene almacenados para el cliente. Éste 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.

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 legítimos. 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 servidor DHCP falso, puede detectar todo el tráfico que envía violando todo tipo de privacidad y pudiendo acceder a datos comprometidos o realizar ataques del tipo denegación de servicio (*Denial of Service*, DoS) o *man-in-the-middle* (MitM).

5.6.2 Componentes de la aplicación

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 interceptarán 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 *DHCPPProcesor* 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 acaba 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, efectivamente, el paquete que se está procesando es un *DHCPOffer*, comprobamos el puerto por el que venía ese paquete. Si es el mismo que aquél 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, entonces se bloquea y el *host* no recibe ese paquete. Por tanto, no acepta la oferta y se evita que cualquiera intente realizar un ataque a nuestra red.

5.6.3 Banco de pruebas

En este apartado se va a comprobar el funcionamiento de la aplicación. Para ello, en primer lugar, se va a mostrar cómo es el funcionamiento del protocolo DHCP cuando únicamente hay un servidor, y a continuación se probará el funcionamiento de la aplicación.

Para ello, tenemos la red mostrada en la Figura 108, donde se puede ver que se ha añadido un *router* de tipo *OpenWrt*, que incluye un servidor DHCP encargado de proporcionar direcciones IP a los *hosts* cuya configuración así lo permite. En nuestro caso, el Alpine-5 tiene por defecto

obtener una IP a través de este protocolo de manera dinámica. La forma de indicárselo se puede ver en la Figura 107.

```

AlpineLinux-5 interfaces

#
# This is a sample network config uncomment lines to configure the network
#


# Static config for eth0
#auto eth0
#iface eth0 inet static
#    address 192.168.0.2
#    netmask 255.255.255.0
#    gateway 192.168.0.1
#    up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
auto eth0
iface eth0 inet dhcp

```

Figura 107: Configuración de un host mediante DHCP.

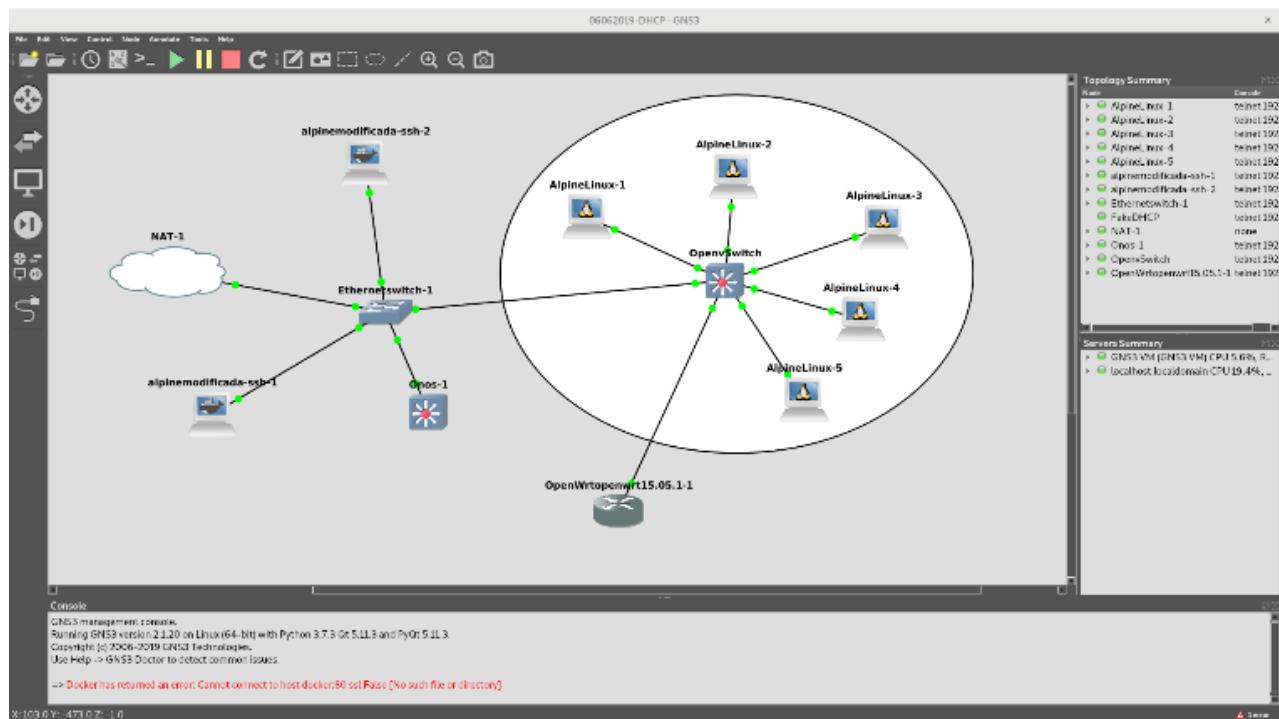


Figura 108: Red utilizada para la aplicación FakeDHCP.

A continuación, arrancamos el *host* y capturamos tráfico en el enlace que comunica el Open vSwitch con el *router* que hemos añadido, con el objetivo de ver el funcionamiento del protocolo DHCP.

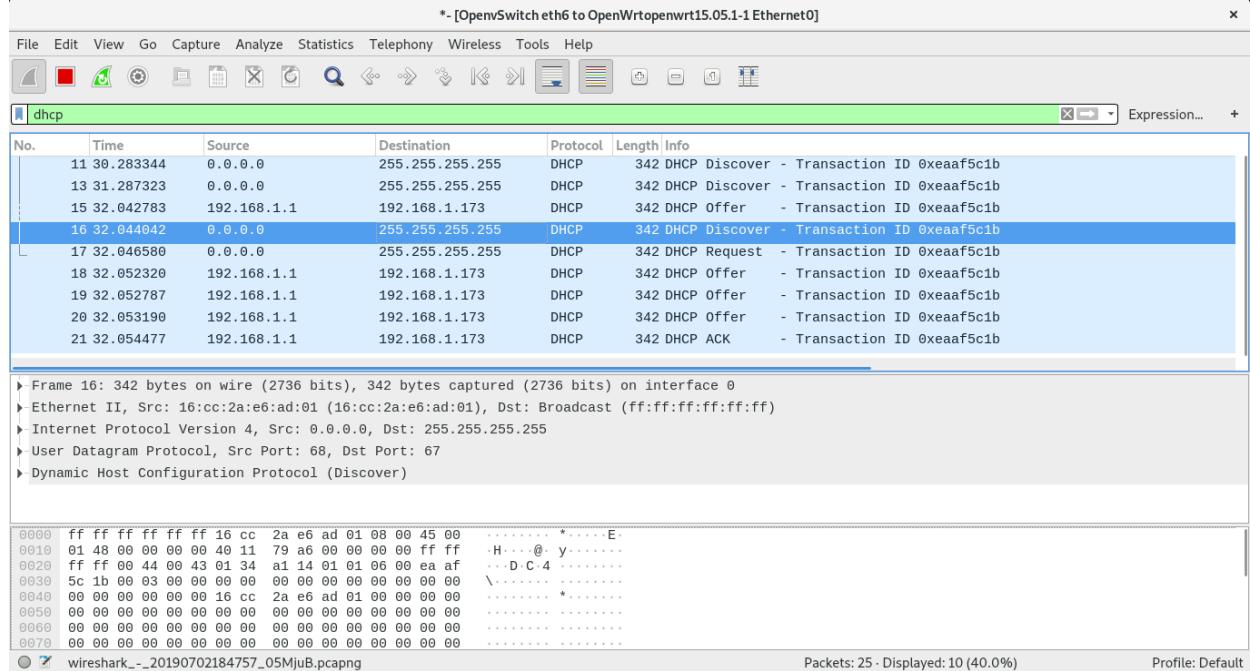


Figura 109: Vista desde el Wireshark de los DHCP Offer enviados.

Tal y como se puede observar, el *router* envía varios *DHCP OFFER*, pues el protocolo UDP no es fiable y podrían perderse los mensajes enviados.

Una vez comprobado que el *router* funciona correctamente como servidor DHCP, el siguiente paso es activar la aplicación e incluir un segundo *router* que consideraremos falso, quedando como resultado la siguiente red (Figura 110).

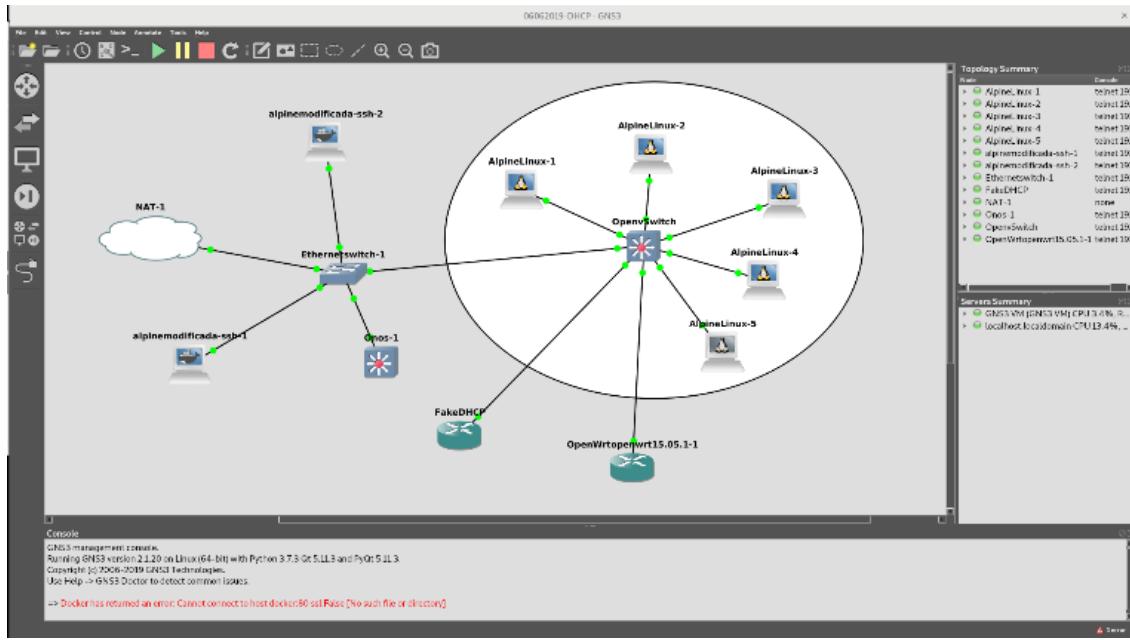


Figura 110: Red con el router falso añadido.

La diferencia entre ambos, que nos permita poder identificarlos, está en el rango de direcciones IP que ofrece cada *router*. El *router* que consideramos legítimo, ofrece rangos de direcciones IP pertenecientes a la subred 192.168.1.x, mientras que el *router* falso ofrece direcciones IP pertenecientes a la subred 192.168.2.x. De esta forma, cuando veamos la IP que tiene el *host*, sabremos a qué *router* ha hecho caso. Esto podemos verlo ejecutando el comando *ifconfig* sobre los 2 *routers* y viendo la subred a la que pertenecen.

Cabe destacar que, en este caso, el *router* legítimo se conecta en la interfaz eth8, mientras que el *router* falso está conectado en cualquier otra interfaz, por ejemplo, la interfaz eth7 del Open vSwitch.

```
FakeDHCP
[ 15.753429] br-lan: port 1(eth0) entered forwarding state
[ 15.818699] 8021q: adding VLAN 0 to HW filter on device eth1
[ 17.755341] br-lan: port 1(eth0) entered forwarding state
root@OpenWrt:/# ifconfig
br-lan  Link encap:Ethernet HWaddr 0C:A5:CC:E0:86:00
        inet addr:192.168.1.1 Bcast:192.168.1.255 Mask:255.255.255.0
        inet6 addr: fe80::ea5:ccff:fe00:8600/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:11 errors:0 dropped:0 overruns:0 frame:0
        TX packets:354 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:3308 (3.2 KiB) TX bytes:33572 (32.7 KiB)
RX bytes:0 (0.0 B) TX bytes:29718 (29.0 KiB)

eth0     Link encap:Ethernet HWaddr 0C:A5:CC:D3:80:00
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:121 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:9326 (9.1 KiB)

eth1     Link encap:Ethernet HWaddr 0C:A5:CC:D3:80:01
        inet6 addr: fe80::ea5:ccff:fed3:8001/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

OpenWrtopenwrt15.05.1-1
root@OpenWrt:/# ifconfig
root@OpenWrt:/# ifconfig
br-lan  Link encap:Ethernet HWaddr 0C:A5:CC:E0:86:00
        inet addr:192.168.1.1 Bcast:192.168.1.255 Mask:255.255.255.0
        inet6 addr: fe80::ea5:ccff:fe00:8600/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:11 errors:0 dropped:0 overruns:0 frame:0
        TX packets:354 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:3308 (3.2 KiB) TX bytes:33572 (32.7 KiB)
RX bytes:0 (0.0 B) TX bytes:29718 (29.0 KiB)

eth0     Link encap:Ethernet HWaddr 0C:A5:CC:E0:86:00
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        root@OpenWrt:/# ifconfig
br-lan  Link encap:Ethernet HWaddr 0C:A5:CC:E0:86:00
        inet addr:192.168.1.1 Bcast:192.168.1.255 Mask:255.255.255.0
        inet6 addr: fe80::ea5:ccff:fe00:8600/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:11 errors:0 dropped:0 overruns:0 frame:0
        TX packets:354 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:3308 (3.2 KiB) TX bytes:33572 (32.7 KiB)
```

Figura 111: Vista de las direcciones IP de los routers.

El rango concreto de direcciones IP que ofrece dentro de la subred en la que se encuentra el *router* se puede ver en el fichero /etc/config/dhcp visualizable con el editor vi.

```

option rebind_localhost '1'
option local '/lan/'
option domain 'lan'
option expandhosts '1'
option nonegcache '0'
option authoritative '1'
option readethers '1'
option leasefile '/tmp/dhcp.leases'
option resolvfile '/tmp/resolv.conf.auto'
option localservice '1'

config dhcp 'lan'
    option interface 'lan'
    option start '100'
    option limit '150'
    option leasetime '12h'
    option dhcpcv6 'server'
    option ra 'server'

config dhcp 'wan'
    option interface 'wan'
    option ignore '1'

- /etc/config/dhcp 30/35 85%

```

Figura 112: Configuración del servidor DHCP.

Como se puede ver, el *router*, cuando actúa como servidor DHCP, ofrece direcciones IP para la LAN en la que se encuentra entre la .100 y la .150.

El siguiente paso es activar la aplicación *org.onosproject.FakeDHCP*, que nos notifica el puerto al que tenemos que conectar el *router* legítimo.

```

OT
16:59:08.368 INFO [TockenBucket] Desactivada aplicacion diffserv
16:59:08.386 INFO [FeaturesServiceImpl] Uninstalling bundles:
16:59:08.386 INFO [FeaturesServiceImpl] org.onosproject.diffServ/1.0.0.SNAPSHOT
OT
16:59:08.418 INFO [FeaturesServiceImpl] Refreshing bundles:
16:59:08.419 INFO [FeaturesServiceImpl] org.onosproject.diffServ/1.0.0.SNAPSHOT
SHOT (Bundle will be uninstalled)
16:59:08.427 INFO [FeaturesServiceImpl] Done.
16:59:08.432 INFO [ApplicationManager] Application org.onosproject.diffserv has
been deactivated
16:59:08.518 INFO [ApplicationManager] Application org.onosproject.FakeDHCP has been installed
16:59:08.555 INFO [FeaturesServiceImpl] Adding features: FakeDHCP/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
17:00:00.067 INFO [FeaturesServiceImpl] Changes to perform:
17:00:00.070 INFO [FeaturesServiceImpl] Region: root
17:00:00.070 INFO [FeaturesServiceImpl] Bundles to install:
17:00:00.071 INFO [FeaturesServiceImpl] mvn:org.onosproject/FakeDHCP/1.0-SNAPSHOT
17:00:00.074 INFO [FeaturesServiceImpl] Installing bundles:
17:00:00.075 INFO [FeaturesServiceImpl] mvn:org.onosproject/FakeDHCP/1.0-SNAPSHOT
17:00:00.125 INFO [FeaturesServiceImpl] Starting bundles:
17:00:00.126 INFO [FeaturesServiceImpl] org.onosproject.FakeDHCP/1.0.0.SNAPSHOT
17:00:00.156 INFO [FakeDHCP] Puerto al que se conecta el router se ha cambiado al 9
17:00:00.156 INFO [FakeDHCP] Activada aplicacion DHCP
17:00:00.158 INFO [FeaturesServiceImpl] Done.
17:00:00.160 INFO [ApplicationManager] Application org.onosproject.FakeDHCP has been activated
17:00:00.405 INFO [FakeDHCP] Puerto al que se conecta el router se ha cambiado al 9

```

Figura 113: Vista del controlador de la activación de la aplicación *FakeDHCP*.

A continuación, capturamos tráfico, tanto en los 2 enlaces que unen el Open vSwitch con los *routers* como con el enlace que une el Open vSwitch con el *host*, y encendemos el *host* Alpine-5 para ver qué ocurre. Antes de nada, comprobamos que el *host* ha recibido una dirección IP de la subred adecuada.

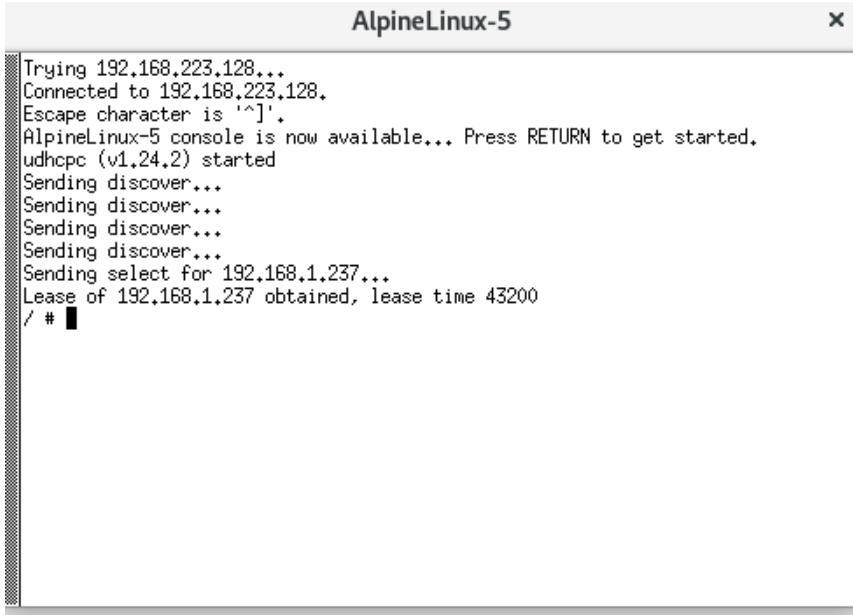


Figura 114: Comprobación de la dirección IP recibida por el host.

Como se ve en la Figura 114, es la subred adecuada, es decir, la subred 192.168.1.0/24.

En el enlace que une el *host* con el Open vSwitch se ha capturado el siguiente tráfico (Figura 115).

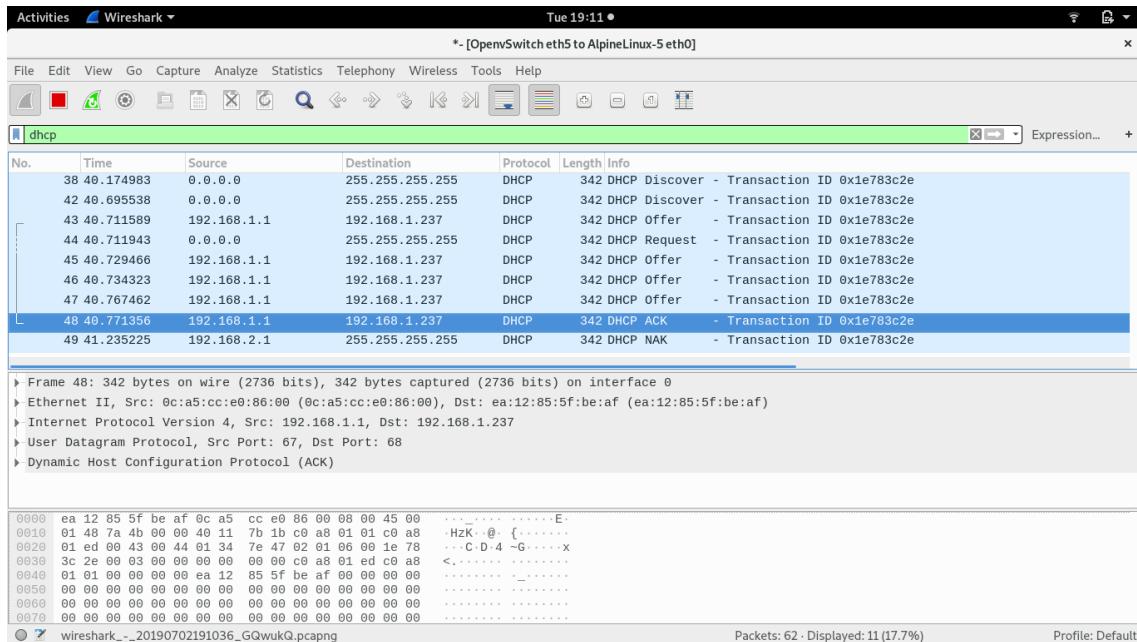


Figura 115: Captura de wireshark del enlace entre el host y el Open vSwitch.

Como se puede ver, recibe ofertas, pero aquellas que vienen del *router* falso (dirección IP 192.168.2.1) son filtradas, mientras que se aceptan las que provienen del *router* legítimo.

En el enlace que une el *router* legítimo con el Open vSwitch se ha capturado el siguiente tráfico (Figura 116).

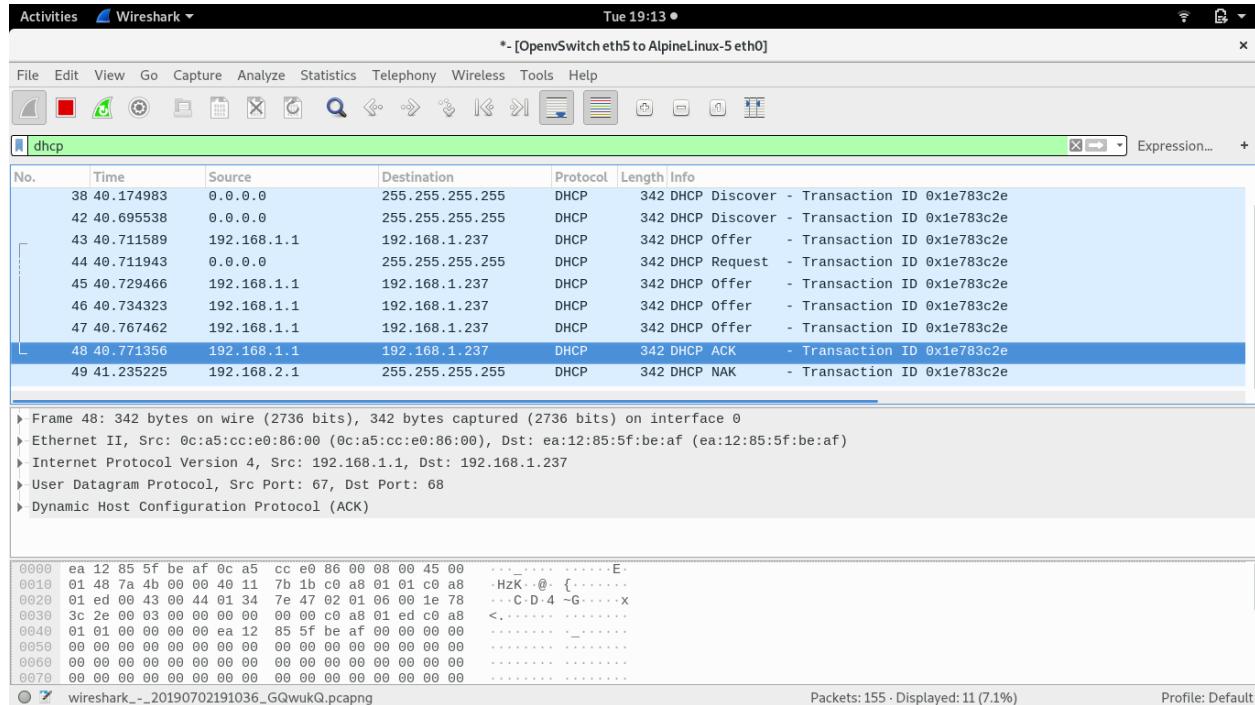


Figura 116: Captura de wireshark del enlace entre el router bueno y el Open vSwitch.

En el enlace que une el *router* falso con el Open vSwitch se ha capturado el siguiente tráfico (Figura 117).

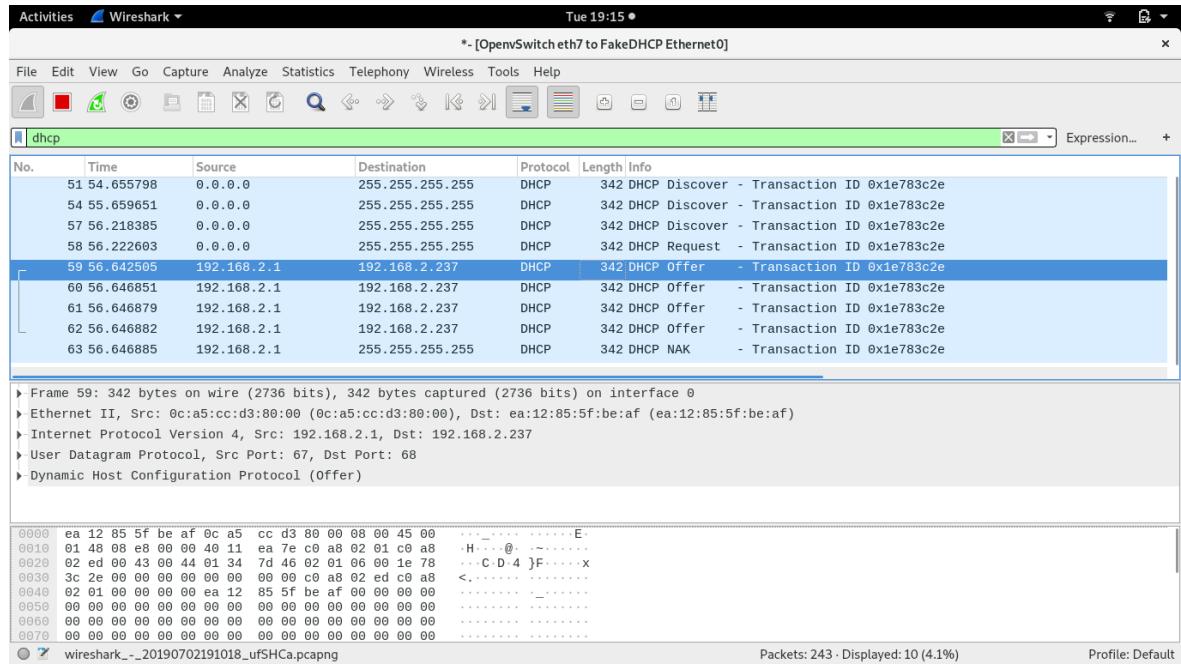
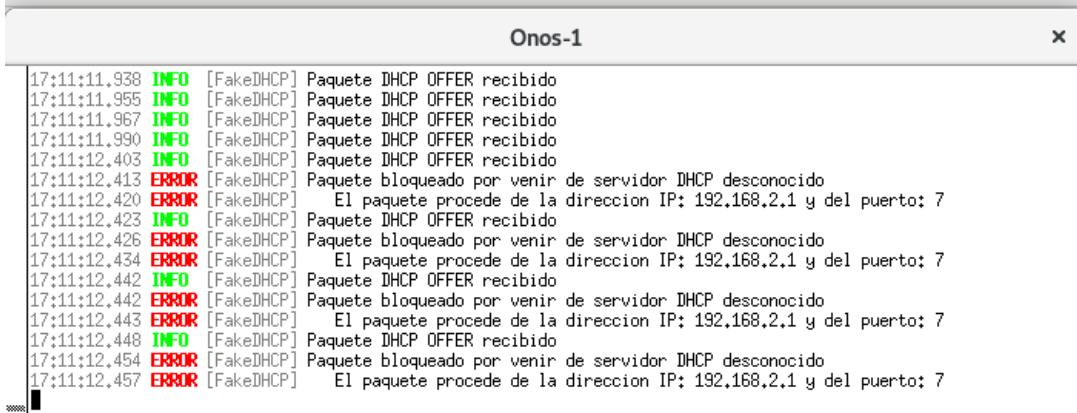


Figura 117: Captura de wireshark del enlace entre el router falso y el Open vSwitch.

Finalmente, en el controlador, vemos cómo se indica que se han bloqueado mensajes del tipo *DHCP OFFER* cuando llegan al Open vSwitch. Tal y como se puede ver en la Figura 118, se muestra que ha habido un mensaje bloqueado y se indica también la dirección IP del servidor falso que ha proporcionado la IP falsa y el puerto al que está conectado.



```

Onos-1
x
17:11:11.938 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:11.955 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:11.967 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:11.990 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:12.403 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:12.413 ERROR [FakeDHCP] Paquete bloqueado por venir de servidor DHCP desconocido
17:11:12.420 ERROR [FakeDHCP] El paquete procede de la direccion IP: 192.168.2.1 y del puerto: 7
17:11:12.423 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:12.426 ERROR [FakeDHCP] Paquete bloqueado por venir de servidor DHCP desconocido
17:11:12.434 ERROR [FakeDHCP] El paquete procede de la direccion IP: 192.168.2.1 y del puerto: 7
17:11:12.442 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:12.442 ERROR [FakeDHCP] Paquete bloqueado por venir de servidor DHCP desconocido
17:11:12.443 ERROR [FakeDHCP] El paquete procede de la direccion IP: 192.168.2.1 y del puerto: 7
17:11:12.448 INFO [FakeDHCP] Paquete DHCP OFFER recibido
17:11:12.454 ERROR [FakeDHCP] Paquete bloqueado por venir de servidor DHCP desconocido
17:11:12.457 ERROR [FakeDHCP] El paquete procede de la direccion IP: 192.168.2.1 y del puerto: 7

```

Figura 118: Vista del controlador del bloqueo de los DHCP OFFER falsos.

5.7 Calidad de servicio

5.7.1 Motivación y explicación

En este apartado se van a realizar diferentes políticas de calidad de servicio (QoS, *Quality of Service*). La calidad de servicio se puede definir como la capacidad que tiene una red de proveer diferentes niveles de servicio para asegurar distintos perfiles de tráfico [34] [35].

En un principio Internet era un servicio *best effort*, es decir, se trataba por igual todo el tráfico existente en la red. Sin embargo, hoy en día, esto no es suficiente ya que existen multitud de aplicaciones con diferentes necesidades de recursos. Por tanto, la motivación para aplicar calidad de servicio se puede resumir de la siguiente forma: priorizar ciertas aplicaciones en la red que requieren una elevada cantidad de recursos como VoIP (voz sobre protocolo de internet):

- Maximizar el uso de la infraestructura de red, manteniendo un margen de flexibilidad.
- Mejorar las prestaciones para servicios en tiempo real.
- Proporcionar mecanismos para priorizar tráfico.
- Responder a los cambios en el perfil de tráfico establecido.

La cabecera IPv4 incluye el campo denominado TOS (*Type of Service*), también llamado campo DS (*Differentiated Services Field*), tal y como se ve en la Figura 119.

| Version | Lon.Cab. | DS | Longitud total | | | | |
|----------------------|-----------|----|----------------|---|---|--|--------------------------|
| | | | X | D | M | | Desplazamiento fragmento |
| Tiempo de vida | Protocolo | | Checksum | | | | |
| Dirección de origen | | | | | | | |
| Dirección de destino | | | | | | | |
| Opciones | | | | | | | |

Figura 119: Cabecera IPv4 añadiendo la calidad de servicio.

El campo DS contiene los siguientes campos (Figura 120).

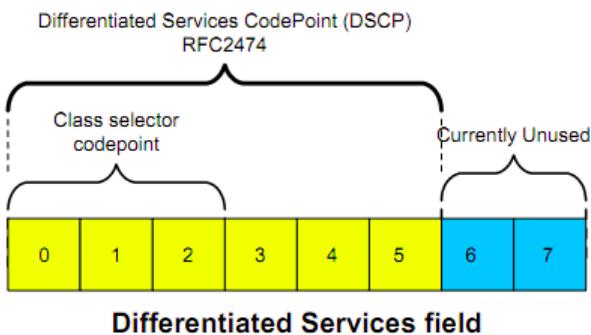


Figura 120: Detalle de los campos añadidos a la cabecera.

En la Figura 121 se puede ver un resumen de los diferentes servicios y los requisitos que deberían cumplir.

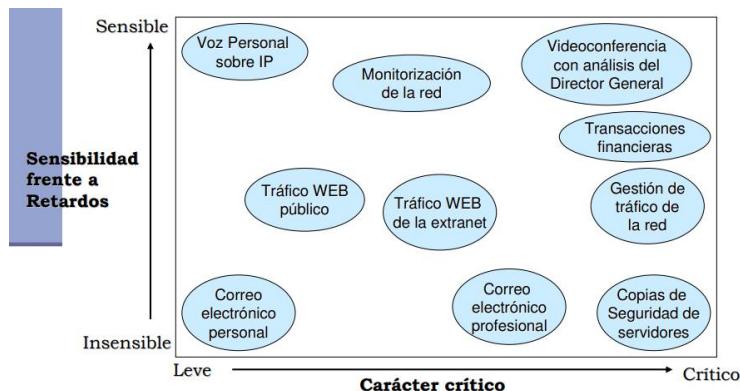


Figura 121: Clasificación de los servicios en función de la sensibilidad a retardos.

Los parámetros más importantes que definen la calidad de servicio son:

- Caudal (throughput): Medido en bits por segundo (bps).
- Retardo: que experimenta un paquete en su recorrido por la red. Medido en segundos (s).
- Pérdidas de paquetes: Porcentaje de paquetes perdidos en la red con respecto al total.
- Jitter: Variabilidad de la latencia en paquetes con el mismo origen, destino y calidad de servicio asociada. Se mide en segundos.

- Disponibilidad: Porcentaje de tiempo en el que funciona el servicio.
- Resiliencia: Capacidad de recuperación ante fallos en la red.

Estos parámetros se reflejan en contratos existentes entre los proveedores de servicio y los clientes denominados SLS (*Service Level Specification*) y SLA (*Service Level Agreement*) en el que se añaden connotaciones legales y económicas en relación con las bases del contrato.

A la hora de realizar calidad de servicio se pueden establecer 3 niveles diferentes en función de la exigencia requerida:

- Nivel best effort: En este nivel no se ofrece ninguna garantía para la recepción de tráfico. Generalmente se utilizan técnicas FIFO (*First in First out*) cuyo funcionamiento consiste en que aquellos paquetes que llegan antes son los primeros en salir, sin realizar ninguna distinción entre los diferentes tipos de flujos.
- Nivel para servicios diferenciados (diffserv): Se basa en la división de tráfico en diferentes clases y en la asignación de prioridades.
- Nivel garantizado: Destinado para aplicaciones con requerimientos exigentes en tiempo real, se asegura un ancho de banda, un límite en el retardo y que no va a haber ninguna pérdida en las colas.

Finalmente, para implementar QoS existen diversas técnicas:

- Clasificación de tráfico: Separación de los paquetes en diferentes categorías, se puede hacer de 2 formas:
 - *MultiField (MF) Classifier*: La clasificación se realiza en función de los siguientes parámetros:
 - Dirección IP origen
 - Dirección IP destino
 - Protocolo
 - Puerto TCP/UDP origen
 - Puerto TCP/UDP destino
 - *Behavior Aggregate (BA)*: En función del DSCP (en Diffserv). Los paquetes han sido previamente etiquetados con el DSCP en la red del cliente. Tal y como se han visto son 6 bits. Los valores que puede tomar son los siguientes (Figura 122) [36].

| Codepoint..... | Descripción |
|----------------|-------------|
| 000000..... | CS0 |
| 001000..... | CS1 |
| 010000..... | CS2 |
| 011000..... | CS3 |
| 100000..... | CS4 |
| 101000..... | CS5 |
| 110000..... | CS6 |
| 111000..... | CS7 |
| 001010..... | AF 11 |
| 001100..... | AF 12 |
| 001110..... | AF 13 |
| 010010..... | AF 21 |
| 010100..... | AF 22 |
| 010110..... | AF 23 |
| 011010..... | AF 31 |
| 011100..... | AF 32 |
| 011110..... | AF 33 |
| 100010..... | AF 41 |
| 100100..... | AF 42 |
| 100110..... | AF 43 |
| 101110..... | EF |

Figura 122: Valores del campo DSCP.

Esta operación, al ser muy costosa, se realiza únicamente en los extremos de la red.

- Marcado de paquetes realizado por el primer nodo de la red para identificar el tipo de tráfico.
- Conformado de tráfico: Controla el volumen de tráfico que entra a la red, es decir, que sean conformes a una determinada especificación. Fundamentalmente se plantean 2 alternativas:
 - *Leaky Bucket*: Es un algoritmo que tiene una similitud con un cubo ya que plantea que, si tenemos un cubo, y la velocidad promedio de gotas que recibe ese cubo es mayor que aquella que es capaz de sacar la cubeta se desbordará. En el caso de redes, la cubeta hace las veces de capacidad máxima del servidor y las gotas de agua entrantes el tráfico entrante al servidor. La Figura 123 ilustra lo explicado [37].

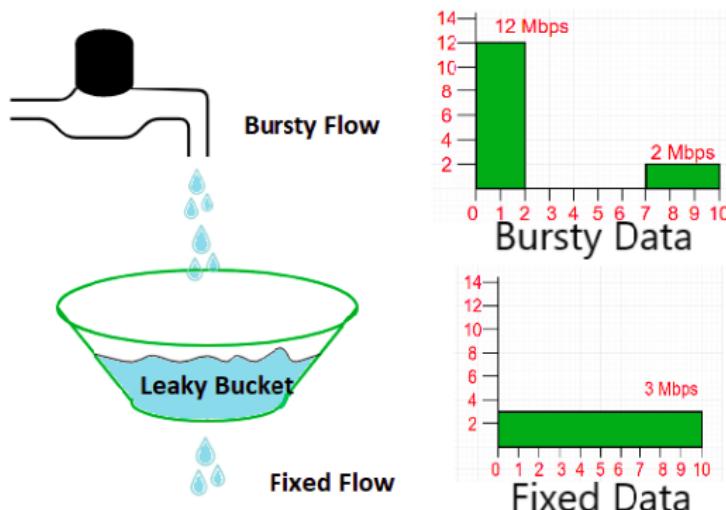


Figura 123: Representación de Leaky Bucket.

- *Token Bucket*: El funcionamiento de este algoritmo es el siguiente:
 - Se generan *tokens* con una tasa de R tokens por unidad de tiempo. Este parámetro se denomina CIR (*Committed Information Rate*) y se mide en bps.
 - Se almacenan en una cubeta (*bucket*) de tamaño máximo B tokens. La capacidad máxima se denomina CBS (*Committed Burst Size*).
 - Cuando llega un paquete de n bytes se retiran n tokens de la cubeta. En caso de que no haya se aplican políticas de *policing*, que es el siguiente paso.

En la Figura 124 se puede ver un esquema de este algoritmo.

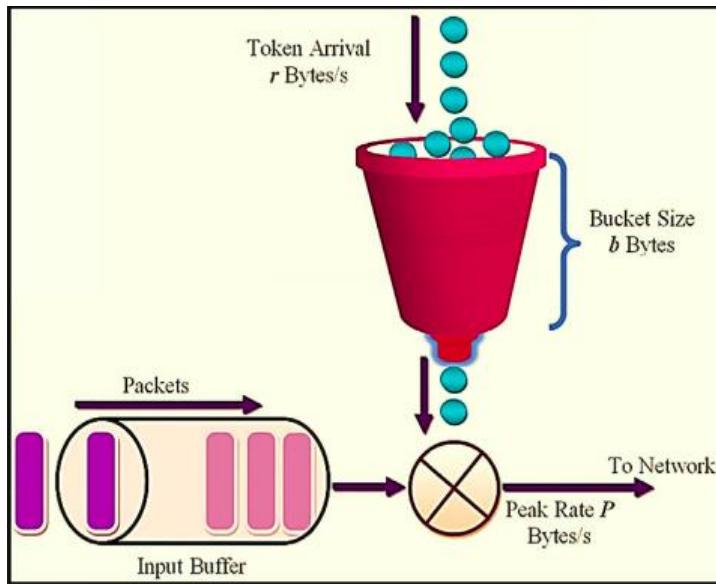


Figura 124: Representación de Token Bucket.

La fase de conformado incluye otra parte, que es la de *policing*. Esto se debe a que, ¿qué se debe hacer con los paquetes que exceden las características de las políticas anteriores? Habría 2 opciones: o se descartan esos paquetes, o se etiquetan de forma que sea más probable su descarte.

Un algoritmo muy eficiente que permite etiquetar paquetes de forma muy sencilla es el *Dual Token Bucket*: Este algoritmo está basado en la utilización de 2 *buckets* y su funcionamiento es el siguiente:

- Se tienen 2 *buckets* llenos de tamaño CBS y EBS (*Excess Burst Size*) bytes.
- El tamaño del *bucket* 1, en cada momento determinada es T_c , mientras que el del segundo es T_e . Lógicamente, en el instante inicial equivalen a CBS y EBS respectivamente.
- El *bucket* 1 se llena según una tasa CIR.
- Cuando llega un paquete de tamaño B bytes:
 - Si $B < T_c$, entonces el paquete se marca como verde y se quitan los B tokens que ocupa ese paquete.
 - Si $T_c < B < T_e$ entonces, el paquete se marca como amarillo y se quitan B tokens del 2º *bucket*.
 - Si $B > T_e$ el paquete se marca como rojo y no se quita ningún token.
- El significado de los colores dependerá de la gestión de *buffers* realizada, pero en general, indican preferencia de descarte. Los paquetes rojos tendrán una probabilidad de descarte mucho más alta que los amarillos o verdes.

En la Figura 125 se esquematiza este algoritmo.

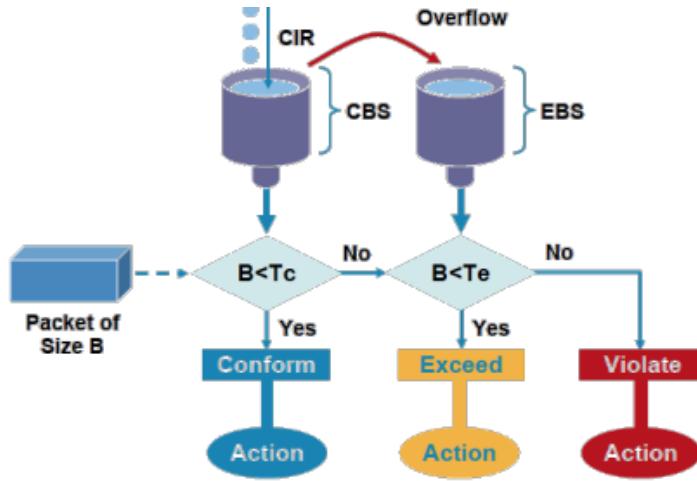


Figura 125: Representación de Dual Token Bucket.

El último paso del conformado de tráfico consiste en la Gestión de los *buffers*. Es decir, decidir cuando un paquete que está en una cola se ha de descartar: En general se usan 2:

- Algoritmo RED (*Random Early Detection*): Si el tamaño medio de la cola excede un umbral, el paquete se descarta con una cierta probabilidad que depende del tamaño medio de la cola.
- Algoritmo WRED (*Weighted Random Early Detection*): Similar a RED, salvo que se realizan curvas de probabilidad de descarte en función del flujo al que pertenezca el paquete.

Por ejemplo, unas posibles curvas WRED son las siguientes (Figura 126):

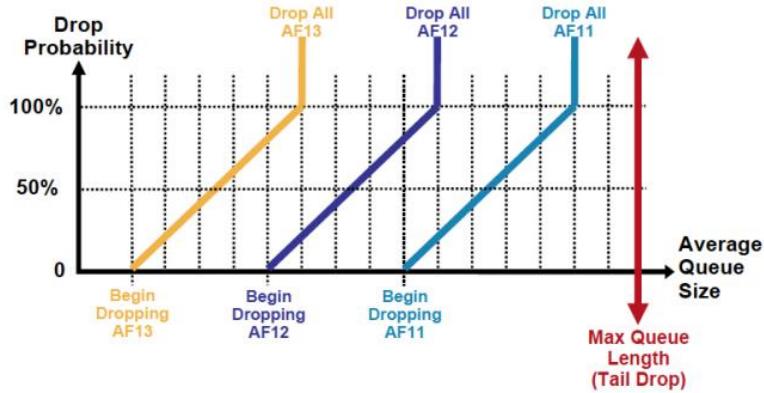


Figura 126: Ejemplo de curvas WRED.

- Vigilancia de tráfico: Realizado por los operadores para marcar o descartar el tráfico cursado por su red.
- Planificación de tráfico: Algoritmos que determinan que paquete se envía en cada momento por el enlace. Como se ha comentado, un mismo enlace tiene varias colas, por tanto, a la hora de enviar paquetes todas esas colas hay que tener un orden, por ejemplo, dando más prioridad a diferentes colas. Las alternativas más usadas son:

- *SPQ (Strict Priority Queueing)*: Se asigna una prioridad a cada cola. Siempre se transmiten primero los paquetes de colas más prioritarias. El problema que tiene es que las colas más prioritarias pueden hacerse con toda la capacidad de transmisión.
- *WRR (Weighted Round Robin)*: Se asigna un peso a cada cola y esta tendrá al menos ese porcentaje de utilización de la capacidad de transmisión.

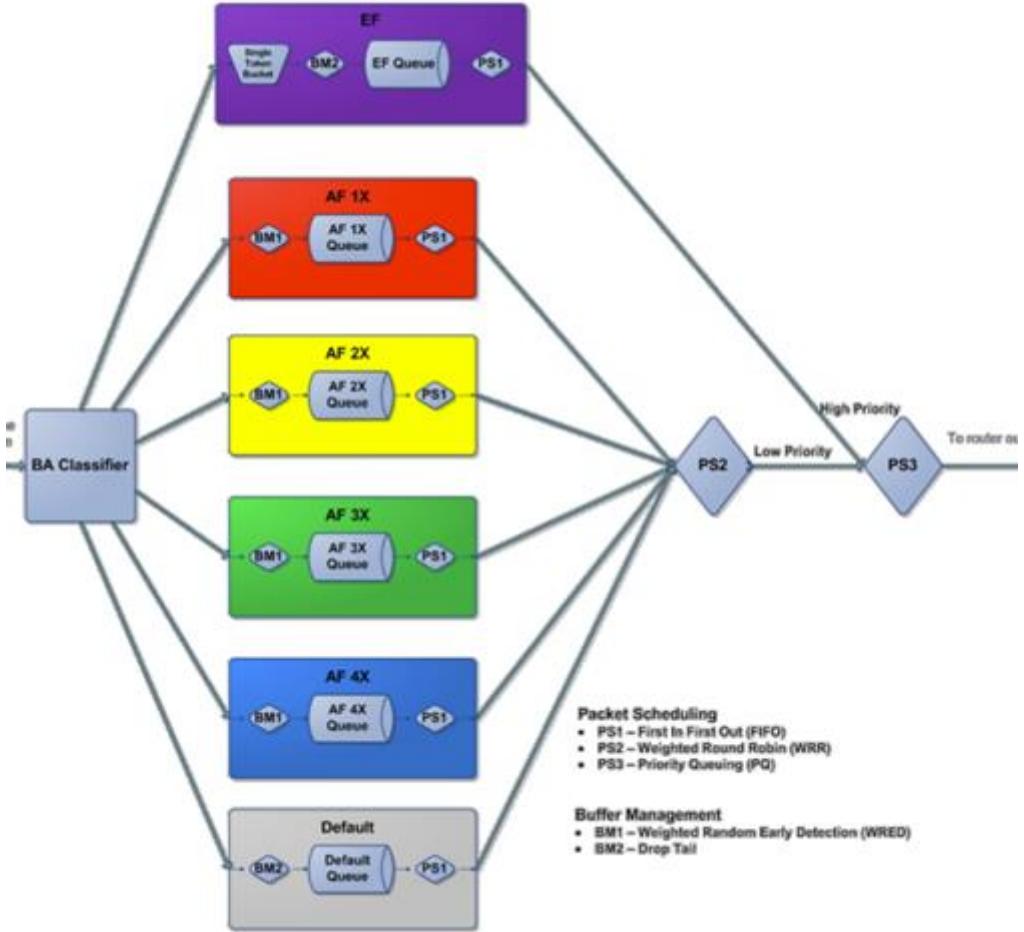


Figura 127: Gestión de buffers.

Centrándonos en la parte referente a las redes SDN, y en concreto, al controlador ONOS, es necesario configurar un par de pasos más. Para ello, volvemos al controlador ONOS, una vez hemos finalizado de configurar el Open vSwitch y realizamos lo siguiente sobre él [38]:

- devices
- device-controllers ID-dispositivo

```

Password:
Welcome to Open Network Operating System (ONOS)!

onos@root > devices
onos@root > devices
id=of:0000d2007aa1514a, available=true, local-status=connected 1m38s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.8.1, serial=None, chassis=d2007aa1514a, driver=ovs, channelId=192.168.122.2:58002, managementAddress=192.168.122.2, protocol=OF_14
onos@root > device-controllers of:0000d2007aa1514a
10:35:43.944 INFO [Controller] Handle new node connection
10:35:43.949 INFO [Controller] Get connection from ip address 192.168.122.2 : 6
640
10:35:44.221 INFO [DeviceManager] Local role is MASTER for ovsdb:192.168.122.2
10:35:44.242 INFO [DeviceManager] Device ovsdb:192.168.122.2 connected
10:35:44.934 WARN [DefaultOvsdbClient] type of controller column
10:35:44.957 WARN [DefaultOvsdbClient] type of controller column
tcp:192.168.122.37:6633
onos@root > devices
id=of:0000d2007aa1514a, available=true, local-status=connected 2m12s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.8.1, serial=None, chassis=d2007aa1514a, driver=ovs, channelId=192.168.122.2:58002, managementAddress=192.168.122.2, protocol=OF_14
id=ovsdb:192.168.122.2, available=true, local-status=connected 6s ago, role=MASTER, type=CONTROLLER, mfr=unknown, hw=unknown, sw=unknown, serial=unknown, chassis=0, driver=default, ipAddress=192.168.122.2
onos@root >

```

Figura 128: Vista de la configuración de los dispositivos necesarios.

Esto activará una parte del Open vSwitch que será necesaria para realizar cualquier tipo de calidad de servicio. Tal y como se ve en la Figura 128, se ha activado otro dispositivo que es el controlador ovsdb:direccionIP. Es con éste con el que trabajaremos.

Si nos fijamos un poco más, vemos que los *drivers* que contiene son los predefinidos por defecto, en lugar de los *drivers ovs* que tiene el otro dispositivo. Esto hay que cambiarlo, ya que serán necesarios (en el siguiente apartado se explica el motivo).

Para cambiarlos se va a utilizar la aplicación qosSample [39], que realiza el cometido de cambiarlos a los del *ovs*. Para ello, clonamos a nuestro repositorio esta aplicación y la instalamos en el controlador ONOS, finalmente desde la CLI del controlador ONOS ejecutamos el siguiente comando (Figura 129):

```

id=of:0000fa0f8d371c4b, available=true, local-status=connected 10s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.8.1, serial=None, chassis=fa0f8d371c4b, driver=ovs, channelId=192.168.122.2:56238, managementAddress=192.168.122.2, protocol=OF_14
onos@root > device-controllers of:0000fa0f8d371c4b
11:33:19.772 INFO [Controller] Handle new node connection
11:33:19.780 INFO [Controller] Get connection from ip address 192.168.122.2 : 6
640
11:33:20.053 INFO [DeviceManager] Local role is MASTER for ovsdb:192.168.122.2
11:33:20.080 INFO [DeviceManager] Device ovsdb:192.168.122.2 connected
11:33:20.752 WARN [DefaultOvsdbClient] type of controller column
11:33:20.777 WARN [DefaultOvsdbClient] type of controller column
tcp:192.168.122.37:6633
onos@root > qos-loading-driver ovsdb:192.168.122.2
onos@root > devices
id=of:0000fa0f8d371c4b, available=true, local-status=connected 31s ago, role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.8.1, serial=None, chassis=fa0f8d371c4b, driver=ovs, channelId=192.168.122.2:56238, managementAddress=192.168.122.2, protocol=OF_14
id=ovsdb:192.168.122.2, available=true, local-status=connected 13s ago, role=MASTER, type=CONTROLLER, mfr=unknown, hw=unknown, sw=unknown, serial=unknown, chassis=0, driver=ovs, ipAddress=192.168.122.2, locType=none, name=ovsdb:192.168.122.2
2

```

Figura 129: Vista de los drivers cargados.

Como se puede ver en la Figura 129, los *drivers* se han cambiado satisfactoriamente.

5.7.2 Componentes de la aplicación

Como primera aplicación desarrollada, se va a intentar crear una serie de colas en el Open vSwitch y aplicarles una política de *token bucket*.

Para ello, en primer lugar, hay que comprobar que nuestro dispositivo contenga las características necesarias para poder llevar a cabo calidad de servicio (ya que en función de la versión utilizada o de los *drivers* que tenga el Open vSwitch podría no funcionar).

Para ello obtenemos los dispositivos de la red, en nuestro caso ahora serán dos, y aplicamos el siguiente código:

```
if(d.is(QueueConfigBehaviour.class) && d.is(PortConfigBehaviour.class))
```

Este método devuelve *true* si estas clases son configurables en nuestro dispositivo.

En este punto es cuando cobra importancia la configuración realizada anteriormente, ya que sin ella estos servicios no funcionarían. Una vez comprobado que nuestro Open vSwitch contiene estas configuraciones, las utilizamos como servicio de la siguiente forma:

```
QueueConfigBehaviour queueConfig = d.as(QueueConfigBehaviour.class);  
QosConfigBehaviour qosConfig = d.as(QosConfigBehaviour.class);  
PortConfigBehaviour portConfig = d.as(PortConfigBehaviour.class);
```

Esto lo que hace es devolvernos la abstracción de la clase en la variable utilizada, es decir, acceder a los métodos de la misma.

A continuación, creamos una cola con el siguiente código:

```
QueueDescription queueDesc = DefaultQueueDescription.builder()  
    .queueId(QueueId.queueId("1"))  
    .maxRate(Bandwidth.bps(4000000))  
    .minRate(Bandwidth.bps(3000000))  
    .burst(20000L)  
    .build();
```

Lo que hemos hecho es crear una cola de id="1", que tenga una velocidad oscilante entre 3Mbps (mínimo garantizado) y 4Mbps (máximo) y que el tamaño de ráfaga sea de 20000 bytes.

Seguidamente, añadimos la cola:

```
queueConfig.addQueue(queueDesc);
```

El siguiente paso es crear una calidad de servicio sobre esa cola. En este caso la calidad de servicio será implementada con una disciplina *token bucket*, para que sea éste el que descarte tráfico en caso de que la cola no pueda con el volumen de datos recibido:

```
QosDescription qosDesc = DefaultQosDescription.builder()  
    .qosId(QosId.qosId("qos1"))  
    .type(QosDescription.Type.HTB)  
    .cir(cir)  
    .cbs(cbs)
```

```
.queues(queues)
.build();
```

La forma de asignarle esta calidad de servicio a una cola determinada es a partir de *queues*. La API exige que el argumento sea un *HashMap* con el ID de la cola y la descripción hecha (notar que de esta forma se permite tener más de 1 cola con la misma calidad de servicio):

```
Map<Long, QueueDescription> queues = new HashMap<>();
queues.put(1L, queueDesc);
```

Finalmente, hay que indicar a qué puerto se le va a aplicar la cola y la calidad de servicio. Esto se hace mediante el siguiente código:

```
PortDescription portDesc = DefaultPortDescription.builder()
    .isEnabled(true)
    .withPortNumber(PortNumber.portNumber(2, "eth2"))
    .build();
```

Y lo aplicamos:

```
qosConfig.addQoS(qosDesc);
portConfig.applyQoS(portDesc, qosDesc);
```

Para poder probar el código se ha creado en el dispositivo una regla que haga que el tráfico vaya por la cola. Se le añade un puerto TCP a esta regla para poder probarlo con la herramienta *netcat*.

Esto se verá con más detalle en el siguiente apartado.

5.7.3 Banco de pruebas

En este apartado vamos a probar la aplicación, Una vez la activamos se ve la información acerca de la/s cola/s creadas.

```

14:46:37.034 INFO [FeaturesServiceImpl] org.onosproject.diffserv/1.0.0.SNAPSHOT
14:46:37.038 INFO [TockenBucket] Desactivada aplicación diffserv
14:46:37.068 INFO [FeaturesServiceImpl] Uninstalling bundles:
14:46:37.078 INFO [FeaturesServiceImpl] org.onosproject.diffserv/1.0.0.SNAPSHOT
14:46:37.095 INFO [FeaturesServiceImpl] Refreshing bundles:
14:46:37.095 INFO [FeaturesServiceImpl] org.onosproject.diffserv/1.0.0.SNAPSHOT (Bundle will be uninstalled)
14:46:37.098 INFO [FeaturesServiceImpl] Done.
14:46:37.101 INFO [ApplicationManager] Application org.onosproject.diffserv has been uninstalled
14:46:37.140 INFO [FeaturesServiceImpl] Adding features: diffserv/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
14:46:37.846 INFO [FeaturesServiceImpl] Changes to perform:
14:46:37.847 INFO [FeaturesServiceImpl] Region: root
14:46:37.847 INFO [FeaturesServiceImpl] Bundles to install:
14:46:37.848 INFO [FeaturesServiceImpl] mvn:org.onosproject/diffserv/1.0-SNAPSHOT
14:46:37.849 INFO [FeaturesServiceImpl] Installing bundles:
14:46:37.852 INFO [FeaturesServiceImpl] mvn:org.onosproject/diffserv/1.0-SNAPSHOT
14:46:37.875 INFO [FeaturesServiceImpl] Starting bundles:
14:46:37.879 INFO [FeaturesServiceImpl] org.onosproject.diffserv/1.0.0.SNAPSHOT
14:46:37.889 INFO [TockenBucket] Activada aplicación diffserv
14:46:37.893 INFO [TockenBucket] Dispositivo: ovsdb:192.168.122.2
14:46:37.941 INFO [TockenBucket] Colas: DefaultQueueDescription{name=1, type=[MIN, MAX, BURST], dscp=0, maxRate=4000000, minRate=3000000, burst=20000, priority=0}
14:46:37.944 INFO [TockenBucket] Dispositivo: of:00008628e50d2849
14:46:37.948 ERROR [TockenBucket] El id dispositivo no empieza por ovsdb
14:46:37.954 INFO [FeaturesServiceImpl] Done.
14:46:37.955 INFO [ApplicationManager] Application org.onosproject.diffserv has been activated

```

Figura 130: Vista de las colas creadas al activar la aplicación en el controlador.

Como podemos ver, se recorren todos los dispositivos que tenemos, sin embargo, el único en el que nos centramos es el ovsdb:192.168.122.2, ya que el otro no soporta las configuraciones necesarias. Además, se muestra información de la cola creada con los parámetros introducidos que se pueden ver en el apartado anterior.

El siguiente paso, es comprobar si en el Open vSwitch se han creado también las colas correctamente. Para ello nos dirigimos a él y ejecutamos el comando mostrado en la Figura 131.

```

OpenvSwitch:~# ovs-vsctl list queue
_uuid : 5d512697-96be-4fd5-9d86-1d199ceeeeeec
_dscp : []
_external_ids : {onos-queue-id="1"}
_other_config : {burst="20000", max-rate="4000000", min-rate="3000000"}
OpenvSwitch:~# ovs-vsctl list qos
_uuid : 6530b295-27a0-4575-8f04-ff759ea1a6ec
_external_ids : {onos-qos-id="qos1"}
_other_config : {max-rate="5000000"}
_queues : {1=5d512697-96be-4fd5-9d86-1d199ceeeeeec}
_type : linux-htb
OpenvSwitch:~#

```

Figura 131: Vista de las colas creadas en el Open vSwitch.

Tal y como vemos, se muestran correctamente tanto las colas como la calidad de servicio implementadas, lo cual parece indicar que están correctamente implementadas.

Otra forma de ver con más detalle estos aspectos, es a través del comando mostrado en la Figura 132. La diferencia, es que en esta ocasión nos centramos en la interfaz concreta a la cual le implementamos la cola, por tanto, muestra detalles como los bytes enviados o recibidos.

```

alpinemodificada-ssh-2
x
OpenvSwitch:~# ovs-appctl -t ovs-vswitchd qos/show eth2
QoS: eth2 linux-htb
max-rate: 5000000

Default:
    burst: 12512
    min-rate: 12000
    max-rate: 5000000
    tx_packets: 110
    tx_bytes: 14960
    tx_errors: 0

Queue 1:
    burst: 20000
    burst: 12512
    min-rate: 3000000
    min-rate: 12000
    max-rate: 4000000
    max-rate: 5000000
    tx_packets: 0
    tx_bytes: 0
    tx_errors: 0
OpenvSwitch:~#

```

Figura 132: Vista en detalle de las colas creadas.

Tal y como se puede apreciar, en esta versión de Open vSwitch, este comando tiene un *bug* y es que, por cada cola creada, los parámetros aparecen duplicados. Por lo que se ha podido comprobar es únicamente un *bug* visual e internamente lo hace correctamente. Tal y como se comprobará posteriormente, en versiones más recientes se ha solucionado este *bug*. No obstante, aún no ha sido actualizada la imagen en *Docker* y, por tanto, en GNS3 no se puede comprobar (de momento).

Para demostrar el *bug*, se han creado 3 colas (la naturaleza de las mismas es indiferente) y se ha ejecutado el mismo comando, obteniendo la siguiente salida (Figura 133):

```
alpin
max-rate: 5000000
tx_packets: 54
tx_bytes: 7344
tx_errors: 0

Queue 234:
burst: 12512
burst: 12512
min-rate: 12000
min-rate: 12000
max-rate: 5000000
tx_packets: 0
tx_bytes: 0
tx_errors: 0

Queue 1:
burst: 12512
burst: 12512
burst: 20000
max-rate: 5000000
max-rate: 4000000
min-rate: 12000
min-rate: 12000
min-rate: 3000000
tx_packets: 0
tx_bytes: 0
tx_errors: 0

Queue 345:
burst: 12512
burst: 12512
burst: 12512
burst: 20000
max-rate: 19000
max-rate: 5000000
max-rate: 4000000
min-rate: 12000
min-rate: 12000
min-rate: 3000000
tx_packets: 0
tx_bytes: 0
tx_errors: 0

OpenvSwitch:~# █
```

Figura 133: Vista en detalle de las colas creadas en el Open vSwitch.

Como se observa, en cada iteración los parámetros salen repetidos.

El siguiente paso, consiste en enviar tráfico que vaya por esa cola. Es en este momento en el que entra en juego la regla de flujo creada en el código. Esta regla de flujo se recuerda, capturaba tráfico que fuese por el puerto 1230 TCP y lo enviaba por la cola.

Para que la cola hiciese algo distinto al envío normal, se ha decidido bajar ostensiblemente las tasas de envío de la cola, para comprobar si el envío tardaba más de lo habitual, lo que implicaría que el tráfico va por la cola correctamente.

Como la cola está situada en el puerto 2, que corresponde a la máquina Alpine-2, se va a enviar tráfico de este *host* a cualquier otro, en este caso el Alpine-1, a través de la herramienta *netcat* (ver Figura 134).

```
AlpineLinux-1 x AlpineLinux-2 x
/ # nc -l 1230
prueba acerca del funcionamiento de la cola creada por la aplicacion diffserv que incluye una politica de calidad de servicio basada en token bucket
[0]
/ # nc 192.168.0.1 1230
prueba acerca del funcionamiento de la cola creada por la aplicacion diffserv que incluye una politica de calidad de servicio basada en token bucket
[1]
```

Figura 134: Prueba del envío de datos a través de las colas.

Como se puede ver, el mensaje ha llegado correctamente. Sin embargo, ¿se habrá enviado por la cola que hemos creado o por la cola habitual?

Para comprobarlo enviamos más mensajes e invocamos el comando mostrado en la Figura 132. La salida se puede ver en la Figura 135.

```
alpinemodificada-ssh-2
OpenvSwitch:~# ovs-appctl -t ovs-vswitchd qos/show eth2
QoS: eth2 linux-htb
max-rate: 5000000

Default:
    burst: 12512
    min-rate: 12000
    max-rate: 5000000
    tx_packets: 217
    tx_bytes: 29014
    tx_errors: 0

Queue 1:
    burst: 20000
    burst: 12512
    min-rate: 3000000
    min-rate: 12000
    max-rate: 4000000
    max-rate: 5000000
    tx_packets: 0
    tx_bytes: 0
    tx_errors: 0
OpenvSwitch:~#
```

Figura 135: Comprobación de la cola por la cual se ha enviado el tráfico.

Como se puede ver, a pesar de que la regla de flujo está bien creada (ver Figura 136), el tráfico se envía por la cola por defecto.

```
alpinemodificada-ssh-1
onos@root > flows
deviceId=of:00008628e50d2849, flowRuleCount=5
  id=10000e5e10695, state=ADDED, bytes=168, packets=4, duration=1774, liveType=UNKNOWN, priority=40000, appId=org.onosproject.onosflowrule, selector=[ETH_TYPE:17bf], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[]}, clearAtTrigger=null, metadata=null}
  id=1000090de17bf, state=ADDED, bytes=0, packets=0, duration=2349, liveType=UNKNOWN, priority=40000, appId=org.onosproject.onosflowrule, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[]}, clearAtTrigger=null, metadata=null}
  id=10000bcc31d31, state=ADDED, bytes=0, packets=0, duration=2349, liveType=UNKNOWN, priority=40000, appId=org.onosproject.onosflowrule, selector=[ETH_TYPE:11dp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[]}, clearAtTrigger=null, metadata=null}
  id=10000e5e10695, state=ADDED, bytes=336, packets=8, duration=2349, liveType=UNKNOWN, priority=40000, appId=org.onosproject.onosflowrule, selector=[ETH_TYPE:a], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[]}, clearAtTrigger=null, metadata=null}
  id=c20000fdde7c4c, state=ADDED, bytes=769, packets=9, duration=425, liveType=UNKNOWN, priority=129, appId=org.onosproject.onosflowrule, selector=[IN_PORT:2, ETH_TYPE:ipv4, IP_PROTO:6, TCP_DST:1230], treatment=DefaultTrafficTreatment{immediate=[QUEUE{queueId=1}], output=of:00008628e50d2849, deferred=[], transition=None, meter=[]}, cleared=false, StatTrigger=null, metadata=null}
  id=10000da14818c, state=ADDED, bytes=362, packets=5, duration=2349, liveType=UNKNOWN, priority=5, appId=org.onosproject.onosflowrule, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[]}, cleared=false, StatTrigger=null, metadata=null}
deviceId=ovsdb:192.168.122.2, flowRuleCount=0
onos@root >
```

Figura 136: Comprobación de los flujos creados por la aplicación.

En este punto se revisó en multitud de ocasiones el código y la implementación de las colas (probando diversas opciones tanto creando colas como calidades de servicio). Sin embargo, el tráfico siempre se dirigía por la cola por defecto. Por tanto, se concluyó que el problema estaba en la implementación de Open vSwitch y que en futuras versiones funcionaría correctamente.

Sin embargo, tal y como se ha comentado, en *Docker* aún no se encuentra disponible, por lo tanto, no se ha podido comprobar en este TFG y queda pendiente como línea futura.

Posteriormente, se pensó que el problema pudiese ser en la comunicación entre ONOS y el Open vSwitch y se decidió crear las colas y calidad de servicio directamente en el Open vSwitch sin utilizar el controlador. Los comandos utilizados fueron los siguientes:

- `ovs-vsctl set port eth2 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=15000000 queues:0=@newqueue1 queues:1=@newqueue2 -- --id=@newqueue1 create queue other-config:min-rate=3000000 other-config:max-rate=4000000 other-config:burst=20000 -- --id=@newqueue2 create queue other-config:min-rate=2000000 other-config:max-rate=2500000 other-config:burst=22000`
- `ovs-ofctl add-flow br0 in_port=2,ip_proto=6,tcp_dst=1230,actions=set_queue:1`
- `ovs-ofctl add-flow br0 in_port=1,actions=normal`

Sin embargo, tampoco funcionó, obteniendo el mismo resultado que el anterior: el tráfico iba por la cola por defecto y no por las que creaba.

Lo que se hizo a continuación, fue, dado que en *Docker* no está disponible, pero sí en cualquier distribución GNU/Linux, instalar una máquina virtual con una versión más nueva del Open vSwitch, para comprobar al menos si estaba solucionado el *bug* visual comentado anteriormente, lo cual efectivamente ocurrió. Se espera que también se solucionará el otro error, sin embargo, por el momento no se ha podido comprobar.

Capítulo 6: Conclusiones y líneas futuras

En este último capítulo se detallan las conclusiones alcanzadas y su relación con los objetivos marcados, así como una serie de líneas futuras derivadas del presente trabajo.

Como conclusión general principal, se puede afirmar que el enfoque de las redes definidas por software permite dar solución a problemas comunes a los que se enfrenta un administrador de red y permiten avanzar hacia redes auto-adaptativas, evitando así la necesidad de aplicar cambios continuamente en la configuración de la red. Esto se consigue gracias a una gestión centralizada mediante el controlador ONOS. Una aplicación donde se puede ver esto comentado fácilmente es en la de *VLAN* ya que permite asimilar el cambio de ubicación de un usuario, sin necesidad de asignar un puerto de acceso a una VLAN de manera estática. De esta forma, un administrador de red podría conectarse a cualquier roseta RJ45 y acceder a la VLAN asignada a la dirección MAC de su tarjeta de red. Asimismo, otra aplicación donde esta conclusión se ve reflejada es en la aplicación *severalping*, donde para limitar paquetes concretos sin utilizar el enfoque que aportan las SDN sería necesario monitorizar en todo momento la red para detectar patrones de tráfico y prohibirlos.

Como conclusiones secundarias cabe destacar las siguientes:

- En primer lugar, que el tiempo entre sondeos es un factor crítico a la hora de limitar el tráfico entre *hosts* ya que si el tráfico es enviado a gran velocidad (por ejemplo, un enlace a 100Mbps) se puede superar ampliamente el umbral establecido. Esto se puede observar claramente en la aplicación *detectHostBan* (apartado 5.3.2) ya que si establecemos un temporizador de la tarea muy elevado, como puede ser de varios minutos, es posible que un *host* supere el umbral momentos después de superar una iteración, pero, hasta la siguiente iteración podría estar enviando sin restricción alguna. Esto tiene como posible solución poner un temporizador corto.
- Una segunda conclusión que se puede extraer es que la velocidad a la hora de administrar una red se ve incrementada utilizando las aplicaciones desarrolladas. Esto se puede comprobar en la aplicación *VLAN*, por ejemplo, ya que es muy sencillo utilizar los comandos desarrollados para mostrar, añadir o eliminar *hosts* a cualquier VLAN (ver apartado 5.5.3). Otras aplicaciones donde se puede comprobar son *severalping* o *detectHostBan*, ya que se puede modificar de forma muy sencilla el número de *pings* máximos permitidos en el caso de la primera (ver apartado 5.2.3.4) y el límite de datos máximo permitido en el caso de la segunda (ver apartado 5.3.3). Además, también se puede modificar de forma sencilla el tiempo que los enlaces se encontrarán bloqueados.
- Otra conclusión importante es que las redes SDN permiten aumentar la seguridad de las mismas. Esto se puede ver de forma clara en las aplicaciones *statsshow*, *detectHostBan* y *FakeDHCP*. En las dos primeras se evitan ataques de denegación de servicio que puedan impedir que el tráfico legítimo circule por la red, tal y como se ve en el apartado 5.3.3.3, mientras que la tercera de ellas evita ataques del tipo *rogue DHCP*, impidiendo que servidores DHCP falsos puedan interceptar el tráfico y obtener nuestros datos.
- Una última conclusión importante es remarcar la importancia que tiene, a la hora de analizar estadísticas de tráfico, el hecho de no tener en consideración el enlace que conecta el controlador con el Open vSwitch, ya que por este enlace circula tráfico de gestión y, por tanto, las estadísticas dejarían de ser realista. Por tanto, aplicaciones como *statsshow* o *detectHostBan* perderían funcionalidad. Es por esto, por lo que a la hora de configurar la red se decide quitar la interfaz *eth0* del *bridge br0*, conectada al controlador ONOS.

En cuanto a las líneas futuras, al ser un campo tan abierto y en auge, evidente son muchas. Sin embargo, algunas de las que se han identificado y consideramos más útiles se muestran a continuación:

- En primer lugar, la más directa es solucionar los problemas acarreados en la aplicación *difsserv* utilizando *software* más reciente (la versión 2.11 del Open vSwitch es posible que solucione los errores). Una vez estén solucionados, el siguiente paso sería realizar diferentes experimentos en referencia tanto a crear nuevas colas con diferentes parámetros como, sobre todo, probando diversas técnicas de *policing* como *Leaky Bucket*, *Token Bucket*...
- En segundo lugar, toda aplicación es mejorable, bien porque se puedan ocurrir comandos a añadir (al estilo de los creados en la aplicación VLAN) o bien porque se pueden añadir nuevos parámetros configurables que simplifiquen aún más la utilización de las aplicaciones.
- Utilización de *meters*. Una de las opciones que permite ONOS es utilizar *meters* para controlar la velocidad a la que sale el tráfico de las diferentes reglas de flujo (en nuestro caso, ninguna de las reglas de flujo creadas tenía *meters* asociados).
- Realización de una aplicación para la creación de un *firewall* (cortafuegos) dinámico basado en patrones de tráfico. Por ejemplo, en el momento en el que se observe que un *host* determinado está enviando tráfico que pueda ser considerado “sospechoso” de ser un ataque de denegación de servicio (DoS), se descartará todo el tráfico asociado.
- Realización de una aplicación para la creación dinámica de un árbol para distribución del tráfico *multicast*. Cuando un *host* quiere añadirse a un grupo *multicast*, se envía un paquete *join* IGMP (*Internet Group Management Protocol*), por lo que el *switch* detecta dicha notificación y crea una regla de OpenFlow que añade el puerto al que se conecta el *host* que desea unirse al grupo a la lista de puertos a la que realizar el *multicast*.
- También cabe destacar que, dado que las redes SDN están pensadas para satisfacer necesidades de las redes no satisfechas actualmente, diferentes líneas futuras podrían obtenerse tras la identificación de casos de uso extraídos a partir de una conversación con administradores de red.
- Finalmente, el siguiente paso lógico en la escala de aprendizaje consistiría en la utilización del protocolo P4 para programar el plano de datos de la arquitectura SDN (recordar que las aplicaciones pertenecen a la capa de aplicación y que OpenFlow es el protocolo de comunicación entre el plano de control y el plano de datos o *switch*, por tanto, no se ha trabajado en este TFG con la programación del plano de datos).

Con todo lo dicho, cabe destacar que se puede asegurar que las redes SDN son el futuro, y el presente, por su versatilidad y escalabilidad, así como por la cantidad de aplicaciones desarrollables, que pueden permitir controlar las redes de una forma eficaz y sencilla. Las aplicaciones desarrolladas, se pueden considerar como exitosas, ya que, como se ha ido comprobando, todas ellas solucionan problemas de muy diversa índole, pero a la vez, problemas presentes en la vida diaria de los administradores de la red y que permiten simplificar su trabajo.

Bibliografía

- [1] J. H y R. R, «¿Qué es SDN?», 9 septiembre 2017. [En línea]. Available: https://www.ciena.com.mx/insights/what-is/What-is-SDN_es_LA.html.
- [2] P. Goransson, C. Black y T. Culver, Software Defined Networks: A Comprehensive Approach, Morgan Kaufmann, 2016.
- [3] Z. Garden, «Historia del SDN», 12 febrero 2015. [En línea]. Available: <https://openzen.wordpress.com/2015/02/12/historia-del-sdn/>.
- [4] N. Feamster, «SDN Lectures: SDN Course», 4 julio 2015. [En línea]. Available: <https://www.youtube.com/playlist?list=PLpherdrLyny8YN4M24iRJBMCXkLcGbmhY>.
- [5] «IETF,» Internet Society, [En línea]. Available: <https://www.ietf.org/>.
- [6] «Open Networking Foundation: SDN Definition,» [En línea]. Available: <https://www.opennetworking.org/sdn-definition/>.
- [7] Cisco, «Cisco for OpenFlow Configuration Guide,» [En línea]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus/openflow/b_openflow_agent_nxos_n3kn9k/b_openflow_native_agent_nxos_7x_chapter_01.html.
- [8] I. Ccoyllo Sulca, «Redes Definidas por Software (SDN),» Universidad Complutense de Madrid, 2018. [En línea]. Available: <https://informatica.ucm.es/data/cont/media/www/pag-103596/transparencias/redes-por-software-SDN.pdf>.
- [9] SDxCentral, «What is Software Defined Networking?», [En línea]. Available: <https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>.
- [10] M. Ramírez Giraldo y A. M. López Echeverry, «Redes de datos definidas por software - SDN, arquitectura, componentes y funcionamiento,» 9 enero 2018. [En línea]. Available: <https://jci.uniautonoma.edu.co/2018/2018-7.pdf>.
- [11] Open Networking Foundation, «Especificación OpenFlow v1.4,» [En línea]. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf>.
- [12] Open Networking Foundation, «Capítulo B.19 Especificación 1.5 OpenFlow,» [En línea]. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf>.
- [13] Linux Foundation, «Página oficial Open vSwitch,» 2016. [En línea]. Available: <https://www.openvswitch.org/>.
- [14] Open Networking Foundation, ONOS Project, «Página oficial ONOS,» [En línea]. Available: <https://onosproject.org>.
- [15] T. Vachuska, «ONOS Developer Workshop at ONOS,» ONOS Project, 15 junio 2015. [En línea]. Available: <https://speakerdeck.com/onosproject/onos-developer-workshop-at-ons-thomas-vachuska?slide=6>.

- [16] ONOS Project, «API Oficial de ONOS,» [En línea]. Available: <http://api.onosproject.org/2.0.0/apidocs/>.
- [17] Solvetic, «Características y como instalar Fedora 29,» 18 octubre 2018. [En línea]. Available: <https://www.solvetic.com/tutoriales/article/6199-caracteristicas-y-como-instalar-fedora-29/>.
- [18] J. Mutai, «How to Install GNS3 on Fedora 29,» Computing for Geeks, 12 noviembre 2018. [En línea]. Available: <https://computingforgeeks.com/how-to-install-gns3-on-fedora-29-fedora-28/>.
- [19] The Wireshark Team, «Página oficial de Wireshark,» 1999. [En línea]. Available: <https://www.wireshark.org>.
- [20] «Página oficial VMware,» EMC Corporation, [En línea]. Available: <https://vmware.com>.
- [21] Eclipse Foundation, «Página oficial Eclipse,» 2001. [En línea]. Available: <https://eclipse.org>.
- [22] A. Koshibe, «The ONOS Web GUI,» ONOS Project, 17 mayo 2017. [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/The+ONOS+Web+GUI>.
- [23] K. Ayaka, «The ONOS CLI,» ONOS Project, 26 enero 2017. [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/The+ONOS+CLI>.
- [24] Open vSwitch, «Open vSwitch manual ovs-vsctl,» [En línea]. Available: www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt.
- [25] R. Blanco Pérez y R. Ruiz González, «Códigos de las aplicaciones desarrolladas,» [En línea]. Available: <https://github.com/rubencv/TFG>.
- [26] T. Vachuska (tomikazi), «Aplicaciones simples de onos,» 2017. [En línea]. Available: <https://github.com/opennetworkinglab/onos-app-samples/tree/master/oneping>.
- [27] T. Vachuska, «Component Configuration,» ONOS Project, 07 julio 2016. [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/Componet+Configuration>.
- [28] EcuRed, «Netcat,» [En línea]. Available: <https://www.ecured.cu/Netcat>.
- [29] networkingcontrol, «LLDP (Link Layer Discovery Protocol),» 2014. [En línea]. Available: <https://networkingcontrol.wordpress.com/2014/06/10/lldp/>.
- [30] A. Koshibe, «Network Discovery in ONOS,» 25 octubre 2016. [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/Network+Discovery>.
- [31] J. V. Capella Hernández, «Características y configuración básica de VLANs,» [En línea]. Available: <https://riunet.upv.es/bitstream/handle/10251/16310/Art%C3%ADculo%20docente%20configuraci%C3%B3n%20b%C3%ADscica%20VLANs.pdf>.
- [32] R. Izard, «How to Work with OpenFlow Groups,» 22 agosto 2018. [En línea]. Available: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups>.
- [33] Open vSwitch, «Open vSwitch manual ovs-ofctl,» [En línea]. Available: www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt.

- [34] G. Armitage, *Quality of service in IP Networks: foundations for a mult-service internet*, Indianapolis: Macmillan Technical Pub, 2000.
- [35] EcuRed, «Calidad de servicio,» [En línea]. Available: https://www.ecured.cu/Calidad_de_servicio.
- [36] Nicolas, «Tipos de servicio con DSCP,» 8 junio 2009. [En línea]. Available: ccie-en-espanol.blogspot.com/2009/06/tipos-de-servicio-con-dscp.html.
- [37] GeeksforGeeks, «Algoritmo leaky bucket,» [En línea]. Available: www.geeksforgeeks.org/leaky-bucket-algorithm/.
- [38] A. Campanella, «ONOS,» 6 marzo 2017. [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/OVSDB+interaction+and+ONOS+cli+example>.
- [39] JackSunshine, «GitHub (Aplicación de muestra de QoS en ONOS),» 2017. [En línea]. Available: <https://github.com/JackSunshine/qos-sample>.
- [40] J. D. Meza, «Curso Java,» 2018. [En línea]. Available: <https://www.programarya.com/Cursos/Java/>.
- [41] A. Campanella, «OVSDB interaction and ONOS CLI example,» 06 marzo 2017. [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/OVSDB+interaction+and+ONOS+cli+example>.

Anexo 1: Introducción a la programación por objetos

1. ¿Qué es la programación orientada a objetos?

La programación orientada a objetos (POO) es un paradigma de programación que intenta innovar con la forma de obtener resultados facilitando la modificación de las aplicaciones y el modelado del mundo real acercándose a la perspectiva del usuario, la interacción con los entornos y la reutilización de *software*.

Los problemas más importantes que hicieron despuntar este paradigma son los siguientes:

- La creciente complejidad de las aplicaciones, que provoca el aumento del coste que será superior al presupuestado, o son de una calidad y utilidad limitadas.
- La difícil reutilización del *software*, ya que es complicado que las funciones desarrolladas para un proyecto sirvan para otros y, por tanto, existía una baja productividad.
- Las limitaciones de la programación estructurada, ya que este tipo de programación tiende a ser poco potente en lo que respecta a la capacidad de representación abstracta de datos.
- El mantenimiento de las aplicaciones supone la parte más costosa de su ciclo de vida, ya que en la programación estructurada no se desarrollan pensando en una futura posible modificación y/o crecimiento.
- La limitación que tenían los lenguajes de programación estructurados de crear interfaces gráficas.

2. Elementos de la programación orientada por objetos

Este tipo de programación se basa fundamentalmente en un elemento, que es el objeto [40]:

Un objeto es una entidad utilizada para modelar objetos de la vida real, por tanto, consta de un estado y comportamiento. El estado se almacena en variables que se denominan atributos, mientras que el comportamiento se representa mediante métodos.

Por tanto, un objeto es un conjunto de atributos (o variables) y de métodos.

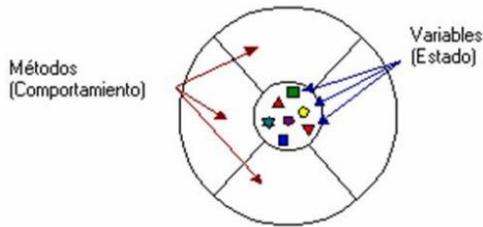


Figura 137: Representación de un objeto.

Pongamos un ejemplo, un objeto que represente una bicicleta de la vida real. Este objeto, tal y como se ha comentado, tendría unas características que definen el estado recogidas en variables. En este ejemplo podrían ser la velocidad, la cadencia de pedaleo o la marcha. Además de estas variables, un objeto puede tener métodos. En el ejemplo que nos concierne podrían ser cambiar la cadencia, cambiar el plato, cambiar la marcha, o acelerar. En la Figura 138 se puede observar como se podría modelar el objeto bicicleta según la estructura anterior.

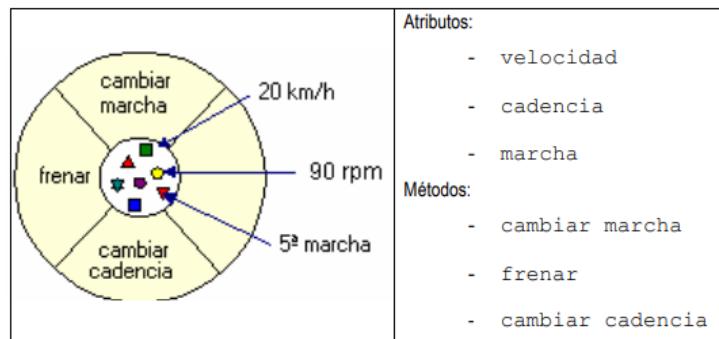


Figura 138: Representación esquemática de un objeto con sus métodos.

Sin embargo, aunque el objeto (junto con las clases) es sin duda la entidad más importante de la POO, no es la única, ya que normalmente un objeto por sí solo no es demasiado útil, sino que, en general, los programas constan de otros muchos objetos. Estos objetos interactúan y se comunican entre ellos por medio de mensajes.

En la Figura 139 se puede ver la representación de un envío de un mensaje.

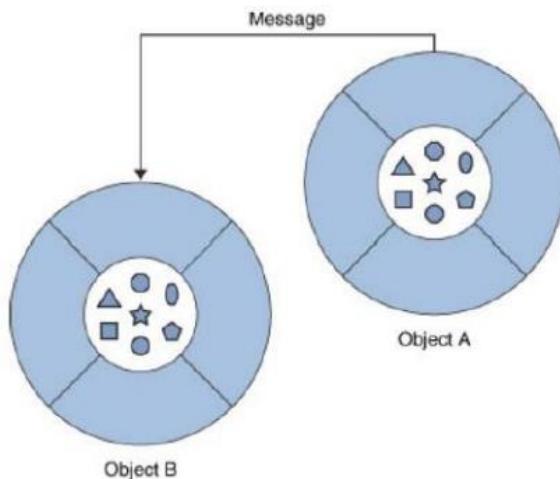


Figura 139: Representación del envío de un mensaje entre 2 objetos.

Continuando con el ejemplo de la bicicleta expuesto anteriormente, una interacción posible sería añadir un objeto persona (aquel que va montado en la bicicleta) y decida en un momento puntual acceder al método Cambiar de marcha con el objetivo de modificar el estado de la bicicleta.

En la Figura 140 se puede ver la representación del ejemplo expuesto:

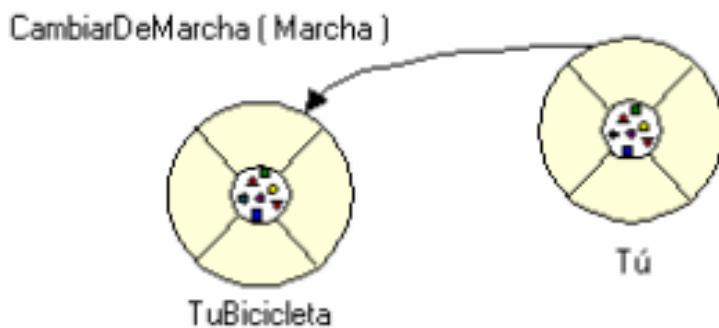


Figura 140: Ejemplo del envío de un mensaje.

Tal y como se mencionó previamente, el otro elemento fundamental de la POO son las clases. Esto se debe a que en general, en el mundo real existen varios objetos de un mismo tipo. Por ejemplo, la bicicleta definida previamente, es un objeto que representa una única entidad, sin embargo, en el mundo hay miles de bicicletas diferentes. Se dice, por tanto, que nuestra bicicleta es una instancia de la clase Bicicletas.

Todas las bicicletas tienen una serie de atributos comunes como pueden ser color, marcha actual... y de métodos, como frenar o cambiar de marcha. Sin embargo, el estado particular de cada bicicleta es independiente. Por ejemplo, una bicicleta puede ser roja o azul, sin embargo, todas tienen un color, sea cual sea. Por tanto, la clase se puede definir como un molde para la creación de objetos.

A la hora de definir un objeto, es obligatorio el indicar a que clase pertenece. En el ejemplo de las bicicletas una posible definición de 2 bicicletas sería la siguiente:

- `Bicicletas bicicletaDeRuben, bicicletaDeJuan;`

Lo que hemos hecho ha sido definir 2 bicicletas concretas pertenecientes a la clase Bicicletas, el siguiente paso es instanciarlas. Para ello se accede al constructor que tenga la clase para inicializar los objetos.

Un constructor es un método especial que permite instanciar e inicializar objetos, aunque no es obligatorio, la mayoría de las clases suelen llevar al menos uno.

Para instanciar por tanto un objeto se hace de la siguiente forma:

- `bicicletaDeRuben = new Bicicletas(azul, 30);`

En el ejemplo hemos supuesto que el constructor de la clase tiene 2 atributos, el color de la bicicleta y la velocidad actual, aunque este podría contener cualquier atributo.

Si queremos acceder a los métodos de una clase a través de un objeto la sintaxis es la siguiente:

- `objetoDeLaClase.metodo(argumentosDelMetodo);`

En el ejemplo de las bicicletas un ejemplo podría ser acceder al método velocidad actual para poder imprimirla posteriormente. Se haría de la siguiente forma:

- `int velocidadActual = bicicletaDeRuben.dameVelocidadActual();`

Este método devolvería el valor 30 que hemos definido previamente al instanciar el objeto.

Para finalizar esta sección se va a hablar de otro elemento importante y muy utilizado durante el transcurso del presente TFG, son las interfaces:

El concepto de interfaz es, en términos generales, la forma de ver una clase desde el exterior, es decir, no se puede ver el estado directamente, sin embargo, si se puede ver la declaración de los métodos (aunque no la implementación de los mismos). De esta forma, cualquier clase que implemente una interfaz debe codificar los métodos de dicha forma respetando la forma en que estos se han declarado, en concreto, el mismo orden y tipo de argumentos y/o mismo tipo de dato devuelto.

Un ejemplo podría ser con el método guardar que puede estar definido en una interfaz. Cualquier clase que desee implementar la interfaz en la que está definido el método guardar debe codificar este método, aunque esta implementación puede variar en función de cada programador.

Es decir, cualquiera que quiera implementar la interfaz debe saber que tiene que crear un método que guarde, la forma de realizarlo se deja libre. Puede decirse, por tanto, que las interfaces permiten al programador, definir un conjunto de funcionalidades sin saber cómo se implementarán la misma.

3. Propiedades de la orientación a objetos

La POO consta de 4 propiedades que merece la pena destacar:

3.1 Herencia

La herencia es un mecanismo que permite definir nuevas clases partiendo de otras ya existentes, es decir, que las clases que derivan de otras heredan automáticamente todos sus elementos, pero además pueden introducir elementos particulares que las diferencien.

Volviendo al ejemplo de las bicicletas, es sabido que hay varios tipos de las mismas, como, por ejemplo, bicicletas de montaña o los tandem. Por tanto, la Bicicleta es la superclase, mientras que las otras 2 son las subclases. En la Figura 141 se puede ver una representación esquemática de lo mencionado.

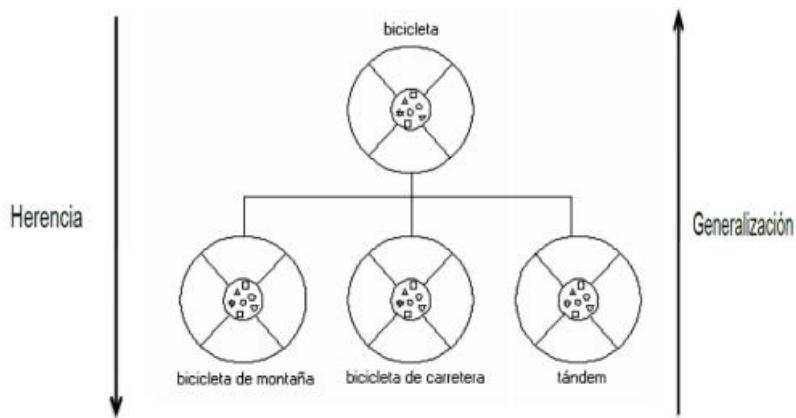


Figura 141: Representación de la herencia.

Por tanto, toda subclase hereda tanto el estado, en el ejemplo cadencia, velocidad... y el comportamiento, es decir, los métodos de la superclase, en este caso, frenar, acelerar, cambiar de marcha.... Pero, además, puede añadir atributos y/o métodos adicionales. Por ejemplo, en el caso del tandem se añadiría un asiento más. De igual forma, también podría sobrescribir métodos de la superclase.

Para indicar en código Java que una clase hereda de otra se utiliza la sentencia *extends* en la declaración de la clase, y la sentencia *super* realiza una llamada al constructor de la super clase para recibir los atributos que tuviera.

3.2 Abstracción

Otro aspecto que permite la POO es definir clases abstractas que definen comportamientos genéricos. Las clases abstractas son clases que no pueden tener instancias que no lo sean también de sus clases derivadas o subclases. Por tanto, una clase abstracta no se puede instanciar ya que solo se usa para definir subclases.

Además, toda clase abstracta tendrá algún método abstracto, obligando de esta forma a las subclases a ofrecer una implementación del mismo.

En el ejemplo de las bicicletas una clase abstracta podría ser vehículos. Vehículos es una clase genérica a partir de la cual se obtienen subclases como la bicicleta, pero también coches, barcos...

3.3 Polimorfismo

El polimorfismo se refiere al hecho de que comportamientos diferentes, asociados a objetos distintos, comparten el mismo nombre, la misma declaración en realidad. Al hacer uso de dicho nombre se invocará el comportamiento correspondiente al objeto que se esté usando.

Esta propiedad permite que un mismo mensaje enviado a objetos de clases distintas haga que estos se comporten también de forma distinta. En el polimorfismo, las declaraciones de las funciones polimórficas en las diferentes clases deben coincidir en cuanto a número, tipo de parámetros, valor devuelto y tipo.

Como ejemplo de Polimorfismo se puede suponer que tenemos 2 objetos, piso1 perteneciente a la clase Pisos y chalet1 perteneciente a la clase Chalets.

Ambas clases pertenecen a la superclase Casas que tiene un método abstracto pintar. Si se llama al método pintar desde la clase chalets se ejecuta el código de la clase Chalet y no el de Pisos.

3.4 Encapsulamiento

El encapsulamiento consiste en la propiedad que tienen los objetos de ocultar sus atributos y/o métodos a otros objetos del programa. La forma natural de construir una clase es la de definir una serie de atributos, que, en general, no serán accesibles fuera del propio objeto, sino que únicamente podrán manipularse a través de los métodos que sean definidos como accesibles desde el exterior de esa clase.

El encapsulamiento permite, por tanto, reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.

Esta propiedad permite proteger la información o estado de un objeto, proporcionando a la par una interfaz pública perfectamente definida que otros objetos podrán usar para comunicarse con él.

Anexo 2: Introducción a git y al repositorio GitHub

GitHub es una plataforma de desarrollo colaborativo que permite alojar proyectos utilizando un sistema de control de versiones (git).

Este *software* (git) ha sido fundamental para poder trabajar durante el presente TFG, ya que todas las aplicaciones desarrolladas se han ido subiendo a un repositorio de GitHub. Una de las grandes ventajas que tiene es que en cualquier momento se puede recuperar una versión anterior de la aplicación, en caso de que en las siguientes hayamos cometido un error. Además, entre versiones, se pueden ver cuáles han sido las líneas añadidas y suprimidas, lo cual, en caso de trabajar entre varios programadores es muy útil para ver los cambios que han podido hacer los compañeros.

Durante el transcurso del proyecto se creó un repositorio en GitHub al que se le añadió la carpeta TFG que contienen todo el código de las aplicaciones desarrolladas. Esta carpeta se puede descargar fácilmente al ordenador mediante el comando *git clone* y el nombre del repositorio. En el caso concreto de la carpeta utilizada en este TFG sería el siguiente comando:

- `git clone http://github.com/rubenccv/TFG.git`

Otros comandos necesarios para el buen funcionamiento de GitHub son los siguientes:

- `git status`: Nos devuelve la lista de ficheros que han sido modificados desde la última vez.
- `git add .`: Añade todos los ficheros modificados a seguimiento de ficheros. En caso de que queramos añadir a seguimiento ficheros concretos el comando sería `git add nombreFichero1 nombreFichero2` y así sucesivamente.
- `git commit`: Añade los ficheros que se han añadido con el comando anterior en el fichero de seguimientos de nuestra carpeta, pero no en el repositorio. Esto se hace para poder añadir algún mensaje informativo con la opción `-m` mensaje de tal forma que cualquiera puede ver el *commit* con la descripción escrita cuando se haya actualizado el repositorio.
- `git push`: Este comando envía los cambios que se encuentran en el HEAD.