

UNIVERSIDAD DE VALLADOLID



E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE
TELECOMUNICACIÓN

Integración de Tecnologías SDN a nivel experimental en redes de acceso GPON

Autor:

D. Carlos Manuel Sangrador Núñez

Tutor:

Dña. Noemí Merayo Álvarez

TÍTULO: Integracion de Tecnologías SDN a nivel experimental en redes de acceso GPON

AUTOR: D. Carlos Manuel Sangrador Núñez

TUTOR: Dña. Noemí Merayo Álvarez

DEPARTAMENTO: Teoría de la Señal y Comunicaciones e Ingeniería Telemática

TRIBUNAL

PRESIDENTE: Ignacio de Miguel Jiménez

SECRETARIO: Noemí Merayo Álvarez

VOCAL: J. Carlos Aguado Manzano

SUPLENTE: Ramón J. Durán Barroso

SUPLENTE: Patricia Fdez del Reguero

FECHA:

CALIFICACIÓN:

Resumen

Este Trabajo Fin de Grado se ha realizado sobre una maqueta de red de acceso GPON (*Gigabit Passive Optical Network*) sobre la que está implementada una solución SDN (*Software Defined Networking*) empleando el protocolo OpenFlow.

En primer lugar, se llevará a cabo la activación y configuración de una pasarela IoT (*Internet of Things*) de modo que esta se pueda integrar fácilmente en la red de acceso GPON. Posteriormente, se procederá con la activación y puesta en marcha del puerto 10G de la OLT, tras lo cual será necesario actualizar el escenario SDN-GPON desplegado.

Para el nuevo escenario SDN-GPON se desplegarán en un servidor externo todas las herramientas *software* necesarias para realizar la gestión SDN de la red, dado que este servidor se conecta al puerto 10G de la OLT mediante un cable de fibra óptica. Para ello, se empleará Docker como herramienta de virtualización, de modo que cada una de las herramientas se desplegará en un contenedor docker distinto.

Por último, también se realizará en el servidor el despliegue de la aplicación de gestión de la red GPON desarrollada en anteriores TFGs, para lo cual se empleará una nueva herramienta denominada Docker-Compose y se realizarán ciertas modificaciones en algunas de sus funciones con el objetivo de mejorar su funcionamiento.

Finalmente, tras la actualización del escenario SDN-GPON, se realizarán distintas pruebas para analizar el rendimiento y poder determinar la correcta funcionalidad del puerto 10G de la OLT.

Palabras clave

SDN (*Redes Diseñadas por Software*), GPON (*Red Óptica Pasiva con Capacidad de Gigabit*), OLT (*Terminación Óptica de Línea*), ONU/ONT (*Unidad de Red Óptica/Terminal de Red Óptica*), OVS (*Open Virtual Switch*), IoT (*Internet de las cosas*), OpenFlow, Open Network Operating System (ONOS), Docker, Python, Gigoló.

Abstract

This Final Degree Project has been carried out on a GPON (*Gigabit Passive Optical Network*) access network model on which an SDN (*Software Defined Networking*) solution is implemented using the OpenFlow protocol.

First, the activation and configuration of an IoT (*Internet of Things*) gateway will be carried out so that it can be easily integrated into the GPON access network. Subsequently, the activation and start-up of the OLT's 10G port will proceed, after which it will be necessary to update the deployed SDN scenario.

For the new SDN-GPON scenario, all the software tools necessary to carry out the SDN management of the network will be deployed on a server, since this server is connected to the 10G port of the OLT through a fiber optic cable. To do this, Docker will be used as a virtualization tool, so that each of the tools will be deployed in a different Docker container.

Certainly, the deployment of the GPON network management application developed in previous TFGs will also be carried out on the server, for which a new tool called Docker-Compose will be used and certain modifications will be made to some of its functions with the aim of improving its performance.

Finally, after updating the SDN-GPON scenario, different tests will be carried out to analyze its performance and to determine the correct functionality of the OLT's 10G port.

Keywords

SDN (*Software Defined Networks*), GPON (*Gigabit-capable Passive Optical Networks*), OLT (*Optical Line Termination*), ONU/ONT (*Optical Network Unit/Optical Network Terminal*), OVS (*Open Virtual Switch*), IoT (*Internet of Things*), OpenFlow, Open Network Operating System (ONOS), Docker, Python, Gigol6.

Agradecimientos

A mis padres y mi hermano, porque gracias a ellos soy lo que soy hoy en día.

A Noemí y Juan Carlos, por su ayuda, compromiso y comprensión en todo momento.

A David y Alfredo, por estar siempre dispuestos a ayudar.

La investigación desarrollada en este Trabajo Fin de Grado ha sido financiada por el Ministerio de Ciencia, Innovación y Universidades en el marco del proyecto ONOFRE-2 (TEC2017-84423-C3-1-P) y la red de investigación Go2Edge (RED2018-102585-T), por la Consejería de Educación de la Junta de Castilla y León en el marco del proyecto ROBIN (VA085G19), y por el Fondo Europeo de Desarrollo Regional FEDER a través del proyecto DISRUPTIVE del Programa Interreg V-A España-Portugal (POCTEP) 2014-2020. Las opiniones son de exclusiva responsabilidad del autor que las emite.

Índice

Agradecimientos	vii
Índice	viii
Índice de figuras	xi
Índice de tablas	xv
1 Introducción	1
1.1 Motivación.....	1
1.2 Objetivos.....	2
1.2.1 Objetivo General.....	2
1.2.2 Objetivos Específicos	2
1.3 Fases y Métodos	3
1.3.1 Fase de Análisis	3
1.3.2 Fase de Implementación	4
1.3.3 Fase de Pruebas.....	4
1.3.4 Fase de Realización de los Informes	4
1.4 Estructura de la Memoria del TFG	4
2 Metodología y herramientas de trabajo	6
2.1 Introducción.....	6
2.2 Redes de Acceso Ópticas Pasivas (PON, <i>Passive Optical Networks</i>).....	6
2.3 Introducción a SDN (<i>Software Defined Networking</i>).....	8
2.4 Docker.....	9
2.4.1 Imágenes docker	10
2.4.2 Contenedores docker	11
2.5 Herramientas y software SDN utilizados	12
2.5.1 Protocolo OpenFlow 1.3.....	12

2.5.2	Software OVS (Open Virtual Switch)	12
2.5.3	Open Network Operating System (ONOS)	13
2.6	Lenguaje de programación Python	15
2.7	Metodología de trabajo	15
2.7.1	Activación e integración de nuevos componentes PON.....	15
2.7.2	Actualización del escenario SDN-GPON.....	16
2.8	Conclusiones.....	16
3	Activación e integración de componentes en la maqueta GPON	17
3.1	Introducción.....	17
3.2	Descripción de la maqueta GPON y del entorno SDN-GPON inicial.....	17
3.3	Activación de nuevos puertos PON en la OLT	23
3.3.1	Registro de las ONTs en puertos PON mediante TGMS	23
3.3.2	Prueba de funcionamiento y conectividad del nuevo puerto	28
3.4	Integración de una pasarela IoT en la red GPON	31
3.5	Activación del puerto 10G de la OLT	41
3.6	Conclusiones.....	43
4	Implementación del nuevo escenario SDN-GPON	45
4.1	Introducción.....	45
4.2	Introducción a Docker	45
4.2.1	Obtención de imágenes y creación de contenedores docker	46
4.2.2	Redes en Docker	48
4.3	Implementación de un servidor DHCP y un <i>router</i> en contenedores docker ...	51
4.3.1	Despliegue y prueba de funcionamiento del servidor DHCP.....	52
4.3.2	Despliegue y prueba de funcionamiento del router	57
4.4	Instalación y despliegue de un OVS en un contenedor docker	60
4.5	Despliegue del controlador ONOS en un contenedor docker.....	65

4.6	Configuración de <i>flows</i> y <i>meters</i> en ONOS.....	71
4.6.1	Configuración de flows en ONOS	72
4.6.2	Configuración de meters en ONOS	75
4.6.3	Configuración de flows con meters en ONOS	77
4.7	Actualizaciones del escenario SDN en el lado del cliente.....	78
4.7.1	Instalación de Docker en los ordenadores del laboratorio.....	78
4.7.2	Instalación y despliegue de nuevos OVSs en contenedores docker en los ordenadores del laboratorio	80
4.7.3	Separación de la red del usuario final de la red de acceso en dos subredes distintas.....	84
4.8	Pruebas de rendimiento del nuevo escenario SDN-GPON mediante la creación de disintos <i>flows</i> y <i>meters</i> en ONOS	88
4.9	Conclusiones.....	95
5	Despliegue de la aplicación de gestión de la red GPON en el servidor <i>chron2</i>.....	96
5.1	Introducción.....	96
5.2	Docker-Compose	96
5.3	Despliegue de la aplicación empleando un archivo Dockerfile y Docker-Compose	97
5.4	Actualización de la aplicación de gestión de la red GPON	104
5.4.1	Instalación de Gigoló en un ordenador con Linux.....	104
5.4.2	Descripción de las modificaciones realizadas en la aplicación de gestión de la red GPON	108
5.5	Creación de <i>flows</i> y <i>meters</i> en ONOS empleando el lenguaje de programación Python	112
5.6	Conclusiones.....	116
6	Conclusiones y líneas futuras.....	118
6.1	Conclusiones.....	118
6.2	Líneas futuras.....	120
7	Bibliografía	121

Índice de figuras

Figura 1: Ejemplo de topología de redes SDN	9
Figura 2: Proceso para la creación de una imagen mediante un archivo Dockerfile.....	10
Figura 3: GUI del controlador ONOS	14
Figura 4: Aspecto real de la red de acceso GPON desplegada en el laboratorio.....	19
Figura 5: Topología en árbol de la red desplegada en el laboratorio.....	19
Figura 6: Esquema del escenario SDN-GPON desplegado sobre la red GPON del laboratorio.....	21
Figura 7: Imagen frontal de la OLT con los puertos PON y los puertos Ethernet	23
Figura 8: Acceso en TGMS a los detalles del puerto 3 de la OLT	24
Figura 9: Pestaña con las ONUs que están conectadas a la red GPON (<i>Disabled ONUs</i>) pero no asociadas al puerto Port 3	25
Figura 10: Ventana para configurar la ONU “ONU Configuration”	25
Figura 11: Estado inicial en el que aparecen las ONUs registradas (estado <i>Pending</i>).....	26
Figura 12: OLT pendiente de sincronización debido al registro de nuevas ONUs	27
Figura 13: Imagen en la que se observan todas las ONUs que están Online.....	27
Figura 14: Topología para la prueba del Puerto 3 de la OLT	28
Figura 15: Captura de pantalla de la interfaz de configuración del router de la ONT (<i>WAN Info</i>).....	29
Figura 16: Prueba de conectividad <i>end-to-end</i> empleando una conexión TCP.....	31
Figura 17: Pasarela IoT BabelGate G5002	32
Figura 18: Conexión mediante <i>ssh</i> con la pasarela IoT	33
Figura 19: Código del programa que permite encender y apagar un led.....	35
Figura 20: Contenido del archivo Makefile.....	35
Figura 21: Visualización de la ejecución del programa.....	36
Figura 22: Código del programa que permite encender y apagar un led y medir la temperatura y la humedad del ambiente	37

Figura 23: Contenido del archivo Makefile.....	38
Figura 24: Visualización de la ejecución del programa.....	38
Figura 25: Estación de contaminación conectada a la pasarela a través de la placa de Arduino.....	39
Figura 26: Contenido del archivo Makefile.....	40
Figura 27: Visualización de la ejecución del programa.....	40
Figura 28: Conexión a la OLT mediante CLI.....	41
Figura 29: Ejecución de los comandos necesarios para conmutar al puerto 10G	42
Figura 30: Activación del puert 10G de la OLT.....	42
Figura 31: Topología de red con cuatro contenedores docker.....	43
Figura 32: Interfaz de red <i>docker0</i>	48
Figura 33: Modelo de red de contenedores docker.....	49
Figura 34: Modelo de red de un contenedor en modo <i>host</i>	50
Figura 35: Nueva interfaz de red creada.....	50
Figura 36: Dirección IP de la ONT de nivel 3 en la red de acceso.....	56
Figura 37: Prueba de conexión entre el <i>chron2</i> y la ONT de nivel 3	56
Figura 38: Topología de la configuración del router en el <i>chron2</i>	58
Figura 39: Prueba de conexión a Intenet desde la red de acceso	60
Figura 40: OVS inicializado	62
Figura 41: Puente OVS.....	62
Figura 42: Implementación del puente OVS en el <i>chron2</i>	63
Figura 43: Configuración del puente <i>br0</i>	63
Figura 44: Prueba de conexión entre el <i>chron2</i> y la ONT de nivel 3	64
Figura 45: Página de inicio de ONOS	66
Figura 46: Página principal de ONOS	67
Figura 47: Activación de la aplicación <i>org.onosproject.openflow</i>	68
Figura 48: Activación de la aplicación <i>org.onosproject.fwd</i>	68
Figura 49: Topología de la red con un OVS activo	69

Figura 50: Prueba de conexión entre la red de acceso y el contenedor de ONOS	70
Figura 51: Topología de la red con varios OVS activos, incluidos los asociados a las ONTs de nivel 2 del laboratorio	71
Figura 52: Lista con las funciones disponibles de la API de REST de ONOS	72
Figura 53: Opciones disponibles dentro de la función <i>flows</i>	72
Figura 54: Campos para la creación de un <i>flow</i>	73
Figura 55: Visualización del <i>flow</i> creado	75
Figura 56: Opciones disponibles dentro de la función <i>meters</i>	75
Figura 57: Campos para la creación de un <i>meter</i>	76
Figura 58: Visualización del <i>meter</i> creado	77
Figura 59: OVS inicializado	81
Figura 60: Implementación del puente OVS en el ordenador del laboratorio	82
Figura 61: Configuración del puente <i>br0</i>	82
Figura 62: Topología de la configuración del <i>router</i> del ordenador del laboratorio	86
Figura 63: Prueba de conexión a Internet desde el ordenador del cliente	88
Figura 64: Prueba de conectividad <i>end-to-end</i> inicial con el ordenador antiguo	89
Figura 65: Prueba de conectividad <i>end-to-end</i> con los <i>flows</i> por defecto empleando el ordenador antiguo	91
Figura 66: Prueba de conectividad <i>end-to-end</i> con los <i>flows</i> por defecto empleando el ordenador nuevo	91
Figura 67: Prueba de conectividad <i>end-to-end</i> con un <i>flow</i> de 150 Mbps empleando el ordenador antiguo	94
Figura 68: Contenido del directorio <i>Pruebas_Aplicacion</i>	97
Figura 69: Archivo Dockerfile con las instrucciones necesarias	98
Figura 70: Contenido del fichero requirements.txt	99
Figura 71: Contenido del fichero docker-compose.yml	100
Figura 72: Menú principal de la aplicación de gestión de la red GPON	103
Figura 73: Opciones del menú <i>Service configuration</i>	104
Figura 74: Tabla con los distintos servicios configurados en la base de datos	104

Figura 75: Interfaz del programa Gigoló	105
Figura 76: Configuración de la conexión con el <i>chron2</i>	106
Figura 77: Ventana para introducir la contraseña.....	106
Figura 78: Interfaz principal del programa con la carpeta que nos permite acceder a nuestros archivos situados en <i>chron2</i>	107
Figura 79: Contenido del directorio <i>/home/csannun</i> del <i>chron2</i>	107
Figura 80: Editor de código <i>Sublime Text</i>	108
Figura 81: Código para comprobar si el servicio seleccionado esta siendo empleado por alguna ONT de la red de acceso	109
Figura 82: Opciones del menú <i>CLI configuration</i>	109
Figura 83: Ejecución de los comandos necesarios para conmutar al modo de gestión CLI	110
Figura 84: Fragmento de código correspondiente al algoritmo DBA	111
Figura 85: Fragmento de código actualizado correspondiente al algoritmo DBA	111
Figura 86: Código para la creación de los <i>flows</i> por defecto.....	113
Figura 87: Visualización del <i>flow</i> creado asociado al OVS del <i>chron2</i>	114
Figura 88: Código para la creación de los <i>meters</i>	114
Figura 89: Visualizacion del <i>meter</i> creado asociado al OVS del cliente.....	114
Figura 90: Código para la creación de los <i>flows</i> con <i>meters</i>	115
Figura 91: Visualizacion del <i>flow</i> creado asociado al OVS del cliente.....	115
Figura 92: Prueba de conectividad <i>end-to-end</i> con un flow de 50 Mbps empleando el ordenador antiguo	116

Índice de tablas

Tabla 1: Resultados obtenidos de las distintas pruebas con el ordenador nuevo 94

1

Introducción

1.1 Motivación

Este Trabajo Fin de Grado (TFG), desarrollado sobre una red de acceso GPON (*Gigabit-capable Passive Optical Networks*) situada en el laboratorio 2L007 de la escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid, se divide en tres partes bien diferenciadas.

En la primera parte, se describirá el proceso de integración de una pasarela IoT (*Internet of Things*) en la red de acceso GPON, así como la activación del puerto 10G de la OLT (*Optical Line Terminal*) que permita llevar a cabo una actualización del escenario SDN (*Software Defined Networking*) desplegado en trabajos previos sobre el testbed GPON.

En la segunda parte, se abordará el despliegue del nuevo escenario SDN-GPON llevado a cabo en un servidor de la escuela, tras la activación del nuevo puerto 10G de la OLT, para lo cual se empleará Docker como herramienta de virtualización. Así mismo, se describirán las correspondientes actualizaciones SDN llevadas a cabo en la parte de la red del cliente para su correcto funcionamiento en el nuevo escenario.

En la tercera parte del TFG, se describirá el proceso de despliegue en dicho servidor de la aplicación desarrollada en Python en trabajos anteriores, la cual controla la gestión de la solución SDN-GPON implementada en este escenario de red. Para dicha implementación, se empleará una nueva herramienta denominada Docker-Compose.

Una vez configurado el nuevo escenario SDN-GPON, se realizarán distintas pruebas para analizar su rendimiento y comportamiento con el objetivo de determinar la correcta funcionalidad del puerto 10G de la OLT.

1.2 Objetivos

1.2.1 Objetivo General

Los objetivos principales de este Trabajo Fin de Grado son tres. En primer lugar, activar e integrar nuevos componentes en la red de acceso del laboratorio. En segundo lugar, implementar un nuevo escenario SDN-GPON tras la activación del puerto 10G de la OLT, para lo cual se empleará un servidor de la escuela (fuera del laboratorio) en el cual se van a desplegar todas las herramientas necesarias. Por último, desplegar la aplicación de gestión de la red GPON desarrollada en anteriores TFGs en el servidor externo.

Para la consecución de los objetivos que se acaban de describir, previamente es necesario analizar la configuración de la red de acceso disponible en el laboratorio, así como estudiar los métodos y herramientas necesarias para lograr los objetivos deseados. Además, es necesario entender el funcionamiento del programa que gestiona el escenario de red SDN-GPON desplegado.

Para alcanzar estos objetivos generales, es necesario establecer otros objetivos más específicos, que se van a describir a continuación.

1.2.2 Objetivos Específicos

Durante este Trabajo Fin de Grado se han desarrollado los objetivos específicos que se enumeran a continuación:

1. Análisis de la topología y configuración del escenario SDN-GPON desplegado al comienzo de la realización de este trabajo,
2. Activación e integración de una pasarela IoT (*Internet of Things*) en la red de acceso del laboratorio y conectarla a la red GPON a través de una de las ONTs.
3. Activación y prueba de un nuevo puerto PON 1G y del puerto 10G de la OLT (*Optical Line Terminal*).

4. Análisis y estudio de herramientas de virtualización para el despliegue del nuevo escenario SDN-GPON en el servidor. Estas nuevas herramientas serán Docker y Docker-Compose.
5. Despliegue y configuración de un *router* y un servidor DHCP, tanto en el servidor como en los ordenadores del laboratorio, mediante tecnología Docker.
6. Despliegue y configuración de nuevos *switches* virtuales mediante Docker, tanto en el servidor como en los ordenadores del laboratorio, compatibles con el estándar SDN para la gestión de las ONTs de nivel 2 disponibles en la maqueta GPON.
7. Despliegue de un controlador SDN que permita gestionar los mensajes *OpenFlow* intercambiados entre los *switches* virtuales desplegados en la maqueta GPON para gestionar los dispositivos PON (OLT, ONTs).
8. Análisis de las diferentes funcionalidades que se encuentran en el programa de gestión global SDN-GPON desarrollado en Python.

1.3 Fases y Métodos

La metodología a seguir para el desarrollo de los objetivos ha conestado fundamentalmente de las fases que se explicarán a continuación.

1.3.1 Fase de Análisis

La finalidad de esta fase es aprender de forma básica cómo funcionan los principales componentes de este Trabajo de Fin de Grado:

- *Análisis de la topología de red GPON del laboratorio 2L007*: estudio de los distintos componentes que constituyen la red y sus modos de gestión.
- *Análisis de las herramientas software utilizadas para el despliegue SDN*: estudio de las diferentes herramientas implementadas sobre el escenario de red SDN-GPON desplegado.
- *Análisis de la herramienta de virtualización Docker*: estudio del funcionamiento de Docker, tipos de redes en docker, obtención de imágenes y creación de contenedores.

- *Análisis del programa de gestión global de la red SDN-GPON*: estudio del programa desarrollado en Python que gestiona la maqueta GPON bajo el protocolo OpenFlow.

1.3.2 Fase de Implementación

En esta fase se llevará a cabo la integración de la pasarela IoT en la red del laboratorio, así como la actualización del escenario SDN-GPON desplegado sobre la maqueta de red GPON. Para ello, se usarán las herramientas *software* y de virtualización analizadas en la fase anterior.

En esta fase también se realizará el despliegue de la aplicación de gestión global de la red GPON en el servidor, modificando las funciones en Python necesarias para mejorar su funcionamiento.

1.3.3 Fase de Pruebas

Fase final en la que se probará el nuevo escenario de red SDN-GPON desplegado en el servidor con el objetivo de comprobar su funcionamiento y prestaciones, así como la funcionalidad del puerto 10G de la OLT. Para ello, se empleará la herramienta *iper/iperf3* que nos va a permitir analizar niveles el ancho de banda de conexiones establecidas entre dos puntos dentro de la red.

1.3.4 Fase de Realización de los Informes

En esta fase, se procedió a realizar los informes del proyecto:

- Informe de Prácticas de Empresa, ya que este Trabajo Fin de Grado tenía asociadas unas prácticas de empresa dentro del grupo de investigación.
- Memoria del Trabajo Fin de Grado.

1.4 Estructura de la Memoria del TFG

El Capítulo 2 describe los fundamentos teóricos de las redes PON y de la tecnología SDN, así como las principales herramientas de virtualización y *software* utilizadas en este trabajo, acabando con la descripción de la metodología a seguir.

El Capítulo 3 contiene la descripción de la solución SDN inicial desplegada sobre la red GPON del laboratorio, continuando con la activación y prueba de un nuevo puerto

PON de la OLT y de la pasarela IoT disponible en el laboratorio. Finalmente, se describe el proceso de activación del puerto 10G de la OLT y la configuración del nuevo escenario SDN-GPON en el servidor situado fuera del laboratorio.

El Capítulo 4 describe la implementación del nuevo escenario SDN tras la activación del puerto 10G de la OLT, para lo cual se realiza, en primer lugar, una introducción a la herramienta de virtualización Docker, para posteriormente continuar con el proceso de creación de los distintos contenedores en los que se van a desplegar cada una de las herramientas necesarias. Por último, se describen una serie de pruebas para analizar el rendimiento del nuevo escenario SDN-GPON.

El Capítulo 5 describe el proceso de despliegue de la aplicación de gestión de la red GPON desarrollada en anteriores TFGs en el servidor, empleando para ello la herramienta Docker-Compose. Finalmente, se comentan algunos cambios realizados en el código con el objetivo de mejorar el funcionamiento de la aplicación y se describe el proceso de configuración de la solución SDN mediante el empleo del lenguaje de programación Python.

El Capítulo 6 incluye las conclusiones finales de este Trabajo Fin de Grado e introduce posibles líneas futuras a desarrollar a partir de este proyecto.

Al final de la Memoria se incluye la bibliografía correspondiente empleada para la realización de este trabajo.

2

Metodología y herramientas de trabajo

2.1 Introducción

En este capítulo de la memoria se va a realizar, en primer lugar, una descripción de los principales fundamentos teóricos de las redes PON y de la tecnología SDN. Posteriormente, se describirán las distintas herramientas software que se van a utilizar a lo largo del desarrollo de este Trabajo Fin de Grado, así como el lenguaje de programación Python, usado para la creación de la aplicación de gestión de la red GPON mediante tecnologías SDN. Además, también se describirá la metodología empleada para la consecución de este trabajo.

Para la realización de este proyecto se utilizará la maqueta red de acceso GPON situada en un armario en el laboratorio de Comunicaciones Ópticas (2L007) de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid. En dicho armario se encuentra la OLT (*Optical Line Termination*), encargada de dar servicio a los clientes mediante el uso de una interfaz web TGMS (*TELNET GPON Management System*) o bien mediante línea de comandos por CLI (*Command Line Interface*), dos divisores ópticos 1:8, un conjunto de ONUs/ONTs (*Optical Network Unit/Terminal*) situadas en el lado del cliente y 25 kilómetros de fibra óptica separados en varias bobinas que conectan todos los dispositivos activos y pasivos.

2.2 Redes de Acceso Ópticas Pasivas (PON, *Passive Optical Networks*)

Una Red Óptica Pasiva o red PON (*Passive Optical Network*) está formada por una serie de elementos ópticos pasivos unidos mediante fibra óptica. Dichos elementos,

tales como divisores o *splitters*, reciben este nombre al no necesitar de alimentación para su funcionamiento. Las redes PON presentan una topología en árbol con una configuración punto-multipunto para transmitir datos hacia y desde los usuarios finales. Los principales componentes que forman una red PON son:

- **OLT (*Optical Line Terminal*)**: elemento activo que se sitúa en la oficina central del proveedor de servicios. Constituye el punto de partida de la red de acceso. Su función es la gestión de los servicios de los usuarios y la conversión óptico-eléctrica entre las señales eléctricas empleadas por los equipos de los proveedores de servicios y las señales ópticas empleadas por la red PON.
- ***Splitters***: elementos pasivos que dividen la potencia óptica de la señal que proviene de la OLT en diferentes ramas hacia los usuarios finales. Del mismo modo, estos divisores ópticos pasivos combinan las señales en el sentido inverso, es decir, desde los usuarios hacia la OLT.
- **ONT/ONU (*Optical Network Terminal/Unit*)**: elemento activo que se encuentra ubicado en el domicilio del cliente. Se encarga de la conversión al dominio eléctrico de las señales ópticas recibidas a través de la fibra (y viceversa) y permite enviar datos en sentido ascendente desde los usuarios hacia la OLT.

En las redes PON se emplean 3 longitudes de onda para el intercambio de información. Para el canal descendente, es decir, desde la OLT hacia las ONTs se emplea la longitud de onda de 1490 nm, mientras que para el canal ascendente (desde las ONTs hacia la OLT) se emplea la longitud de onda de 1310 nm. Por último, es posible separar la transmisión de video utilizando la longitud de onda dedicada de 1550 nm.

Para la multiplexación de los datos y su distribución a los usuarios finales se emplea TDMA (*Time Division Multiple Access*) como protocolo de control de acceso al medio en ambos canales. Para el canal descendente, la OLT envía la información destinada a cada ONT en modo *broadcast*, de manera que cada una de las ONTs recibe el tráfico destinado al resto de las ONTs conectadas al mismo puerto PON de la OLT, por lo que tendrán que seleccionar únicamente los paquetes dirigidos a ellas y descartar los demás. Por ello, es necesario implementar mecanismos de seguridad, ya que a todos los usuarios les llega el tráfico dirigido a otras ONTs. Para el canal ascendente, TDMA separa la información de cada ONT en *slots* o intervalos de tiempo diferentes para evitar

colisiones de datos de diferentes usuarios. Además, para evitar dichas colisiones es necesario que la OLT conozca los retardos entre las distintas ONTs [1], ya que cada una estará a una distancia diferente de la OLT.

2.3 Introducción a SDN (Software Defined Networking)

Las Redes Definidas por Software o SDN (*Software Defined Networking*) constituyen una tecnología en la que se separa el plano de datos del plano de control, por lo que el control se desprende del hardware y se le otorga a una aplicación software, denominada controlador, responsable de gestionar la información de reenvío de los *switches* de las redes cableadas. Las redes SDN proporcionan una mayor flexibilidad, capacidad de programación, gestión y rentabilidad [2].

Los controladores, dispositivos basados en software en los que se encuentra centralizada la inteligencia de la red, mantienen una visión global de la misma. Como resultado, la red aparece frente a las aplicaciones y a las decisiones de política como un conmutador lógico y único. De esta manera, se adquiere el control sobre toda la red desde un único punto lógico, lo que simplifica en gran medida el diseño y operación de las redes. Las redes SDN también permiten simplificar los dispositivos de red, pues estos ya no tienen que entender y procesar varios protocolos estándares, sino simplemente aceptar instrucciones de los controladores SDN [2].

La separación del plano de control proporciona a los operadores de red la posibilidad de configurar de forma centralizada la red y sus servicios, modificándolos en tiempo real y pudiendo desplegar nuevas aplicaciones en un tiempo muy reducido comparándolo con las redes actuales convencionales [2].

Además de los controladores, los otros elementos fundamentales de una red SDN son las interfaces *Southbound* y *Northbound*. La interfaz *Southbound* permite conectar al controlador con la capa de red inferior, de modo que puede realizar cambios en tiempo real en función de las necesidades y demandas. Por otro lado, la interfaz *Northbound* facilita al controlador la conexión con las aplicaciones de la capa superior, siendo uno de los puntos más críticos de la red, pues soportan gran variedad de servicios y aplicaciones. Por último, también se definen las interfaces *East/West*, que permiten la interconexión

entre controladores que pertenecen a diferentes redes físicas, de modo que un controlador SDN de un entorno determinado puede acceder a los recursos de red y a las funciones de otro controlador [3]. Todo esto se ilustra en la Figura 1.

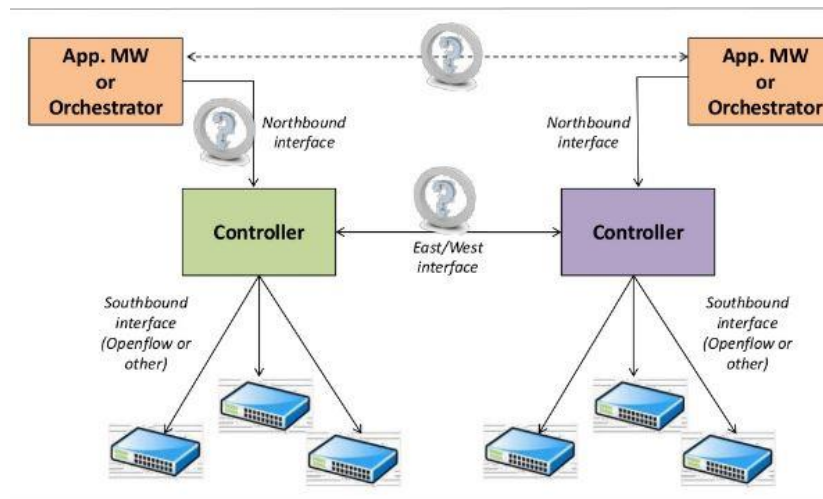


Figura 1: Ejemplo de topología de redes SDN

Por tanto, SDN permite una mayor flexibilidad y agilidad, aumentando la optimización de los recursos y simplificando las redes y sus dispositivos.

2.4 Docker

Docker es una plataforma de código abierto diseñada para desarrollar, implementar y ejecutar aplicaciones mediante el uso de contenedores [4]. Docker permite empaquetar una aplicación y sus dependencias en un contenedor que se puede ejecutar en cualquier servidor Linux, permitiendo una mayor flexibilidad y portabilidad.

En cierto modo, Docker constituye una herramienta de virtualización similar a una máquina virtual. No obstante, a diferencia de las máquinas virtuales, las cuales crean un sistema operativo virtual completo, Docker permite que las aplicaciones empleen el kernel Linux del *host* en el que se están ejecutando, evitando la sobrecarga de iniciar y mantener máquinas virtuales. Esto proporciona un aumento significativo del rendimiento y reduce el tamaño de la aplicación.

Hay dos componentes internos principales de Docker: imágenes y contenedores, y sus características se describen a continuación.

2.4.1 Imágenes docker

Las imágenes docker son plantillas únicamente de lectura con instrucciones para crear un contenedor docker [4]. Las imágenes se pueden obtener de Docker Hub. Docker Hub es un servicio de registro de repositorios que nos permite extraer y enviar imágenes desde y hacia Docker Hub. La mayoría de las comunidades de código abierto tienen imágenes docker precompiladas disponibles para los desarrolladores en Docker Hub.

No obstante, en lugar de usar imágenes predefinidas disponibles en Docker Hub, también se pueden crear imágenes docker usando un Dockerfile [4]. Un Dockerfile es un archivo de texto simple con un conjunto de comandos o instrucciones necesarias para crear y ejecutar una imagen. Para la creación de una imagen docker es necesario definir, al principio del archivo Dockerfile, la imagen base que se quiere emplear, por ejemplo *ubuntu:latest*. Cada una de las instrucciones definidas en el archivo Dockerfile crea una nueva capa en esta imagen base, obteniéndose como resultado la nueva imagen.

Estos comandos o instrucciones incluyen, entre otras opciones, la instalación de paquetes y dependencias, la transferencia de ficheros (por ejemplo: copiar el código de una aplicación que queremos ejecutar en un contenedor) o la ejecución de un comando cuando se cree el contenedor (por ejemplo: ejecutar un fichero *.py* para lanzar una aplicación desarrollada en Python).

En el caso de que la imagen que se va a emplear se genere mediante un archivo Dockerfile, los pasos son los que se pueden observar en la Figura 2.



Figura 2: Proceso para la creación de una imagen mediante un archivo Dockerfile

En este sentido, los pasos a seguir para crear la imagen mediante un archivo Dockerfile son los siguientes:

- Crear un archivo Dockerfile, con la imagen base que se quiere emplear y las instrucciones necesarias para crear la nueva imagen.
- Ejecutar el comando *docker build* para crear la nueva imagen.
- Una vez que la imagen ha sido creada, ejecutar el comando *docker run* para crear el contenedor.

En este proyecto se emplearán tanto imágenes predefinidas, disponibles en Docker Hub, como imágenes creadas mediante archivos Dockerfile.

2.4.2 Contenedores docker

Un contenedor docker es una instancia ejecutable de una imagen [4]. Se pueden crear tantos contenedores como se quieran a partir de la misma imagen docker, y cada uno de ellos puede tener una configuración distinta. Además, se pueden crear nuevas imágenes basadas en el estado actual de los contenedores.

Un contenedor constituye una unidad estándar de software que empaqueta una aplicación y todas sus dependencias, por lo que la aplicación se puede ejecutar de forma rápida y segura en cualquier otro entorno de programación. Una imagen docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones. Una imagen se convierte en un contenedor en tiempo de ejecución. Los contenedores siempre se ejecutan de la misma manera, independientemente de la infraestructura del *host*. Así mismo, comparten el núcleo del sistema operativo del *host* y, por lo tanto, no requieren un sistema operativo propio, lo que aumenta la eficiencia del servidor y reduce los costes.

Los contenedores y las máquinas virtuales tradicionales tienen beneficios similares en cuanto a asignación y aislamiento de recursos, pero funcionan de manera diferente. Los contenedores virtualizan el sistema operativo en lugar del hardware, haciendo que estos sean más portables y eficientes.

2.5 Herramientas y software SDN utilizados

2.5.1 Protocolo OpenFlow 1.3

El estándar que se utiliza en la red GPON del laboratorio para la implementación de SDN es OpenFlow 1.3 [5]. Como ya se ha comentado anteriormente, la gestión de la red se lleva a cabo a través de un controlador SDN. Dicho controlador se comunica con los *switches* virtuales a través del envío de mensajes *Openflow*, y es el encargado de tomar las decisiones de encaminamiento. Para ello, el controlador configura en los *switches* virtuales una serie de tablas *Openflow* que contienen *flows* o flujos de datos, que hacen que el *switch* ejecute diferentes operaciones (*instructions*) en caso de que se cumplan unas determinadas condiciones denominadas *match*.

En nuestro caso de uso, el controlador elegido para gestionar la red de acceso GPON del laboratorio es ONOS (*Open Network Operating System*) [6], cuyas características se explicarán en una de las secciones posteriores.

2.5.2 Software OVS (Open Virtual Switch)

Como se ha mencionado en el apartado anterior, el estándar OpenFlow utiliza *switches* virtuales que se comunican con el controlador SDN. Para la implementación de estos *switches* virtuales se han utilizado OVS (*Open Virtual Switch*) [7], un *software* de código abierto creado para la virtualización de puentes o *bridges*, que soporta y es soportado por OpenFlow, proporcionando encaminamiento en la capa 2 del modelo OSI (*Open System Interconnection*).

Por tanto, el OVS se encargará del plano de datos, dejando el plano de control al controlador SDN. El OVS consta de dos componentes principales, el espacio de usuario (*ovs-vsitchd*) y el módulo *kernel* (*datapath kernel module*). Los paquetes llegan al *kernel* del sistema operativo y es el espacio de usuario quien decide qué acciones se realizan sobre ellos. En este sentido, el OVS tiene dos tipos de procedimiento diferentes:

1. El espacio de usuario, donde el OVS funciona con cualquier sistema operativo y en cualquier entorno.
2. El espacio de *kernel*, donde el OVS está diseñado específicamente para el entorno en el que estamos trabajando.

En el primer caso, los paquetes que llegan al módulo del kernel se envían al espacio de usuario, donde se realizan los cambios correspondientes; sin embargo, en el segundo caso, el espacio de usuario únicamente decide que hacer con los datos y es el kernel quien lleva a cabo las instrucciones pertinentes. Empleando el espacio de kernel se aumenta en gran medida la velocidad de ejecución, eliminando el posible cuello de botella que pudiese crear el espacio de usuario, lo cual optimiza de forma considerable el ancho de banda soportado por el OVS.

En la red GPON del laboratorio, los OVSs se encuentran instalados a la entrada/salida de la OLT y de las ONTs, es decir, de los elementos activos, de forma que podamos controlar los paquetes que circulan por la red. Se crea, por tanto, una capa de abstracción a nivel software sobre la OLT y las ONTs que emula las capas SDN sobre dichos dispositivos activos. Así pues, el OVS que controla el flujo descendente (desde la OLT hacia las ONTs) se encuentra instalado en uno de los servidores de la escuela, denominado chron2 (justo antes del OLT), mientras que los OVSs que controlan el flujo ascendente de los usuarios (desde las ONTs hacia la OLT) se encuentran instalados dentro de contenedores docker en ordenadores situados justo después de las ONTs (hacia los dispositivos de la red residencial del usuario). El uso de estos OVSs nos permite configurar flows capaces de controlar las tasas máximas de transmisión utilizando los meters, que se encargan de imponer dichas tasas de transmisión máximas. Estos flujos o flows emularán la creación de servicios (tales como Internet) en ONTs en el sentido de subida y bajada de la red de acceso.

2.5.3 Open Network Operating System (ONOS)

Como ya se ha mencionado anteriormente, una de las piezas claves de las redes SDN es el controlador. Para el escenario presente en la red GPON del laboratorio, el controlador seleccionado es ONOS, un *software* de código abierto administrado por Linux Foundation [6]. Entre sus principales objetivos está ofrecer un controlador escalable de alto rendimiento con soporte para alta disponibilidad y compatible tanto con dispositivos tradicionales como con dispositivos OpenFlow. Está escrito en Java y construido sobre Apache Karaf [8].

Se trata de un controlador bastante bien documentado. Toda la información está correctamente clasificada y actualizada. Además, resulta muy sencillo configurar una red SDN simple con ONOS. Solo es necesario activar dos aplicaciones, por un lado

org.onosproject.openflow para poder comunicar los *switches* virtuales, en nuestro caso OVSs, con el controlador, y por otro lado la aplicación *org.onosproject.fwd* para habilitar el reenvío reactivo de paquetes. El reenvío reactivo consiste en crear *flows* dinámicamente cuando el controlador no tiene ningún *flow* específico para manejar ese tráfico.

ONOS se puede administrar a través de su API REST, mediante línea de comandos (CLI) o a través de una interfaz web gráfica (GUI). La línea de comandos está basada en la CLI de Apache Karaf y permite administrar las principales características de la red. Por ejemplo, permite crear, eliminar y modificar *flows*, así como administrar dispositivos de red. En cuanto a la interfaz gráfica, su finalidad es, principalmente, ofrecer una representación gráfica de la topología de la red y su estado, tal y como se muestra en la Figura 3 [9].

En este proyecto, el controlador ONOS se configurará mediante la API REST, accesible a través de la documentación de ONOS. Además, dado que la API REST emplea la herramienta CURL (*Curl URL Request Library*) para la configuración de los *flows* y *meters*, también existe la posibilidad de realizar la configuración del controlador a través de algún lenguaje de programación, en nuestro caso Python, a través de peticiones HTTP empleando los métodos GET, POST y DELETE.

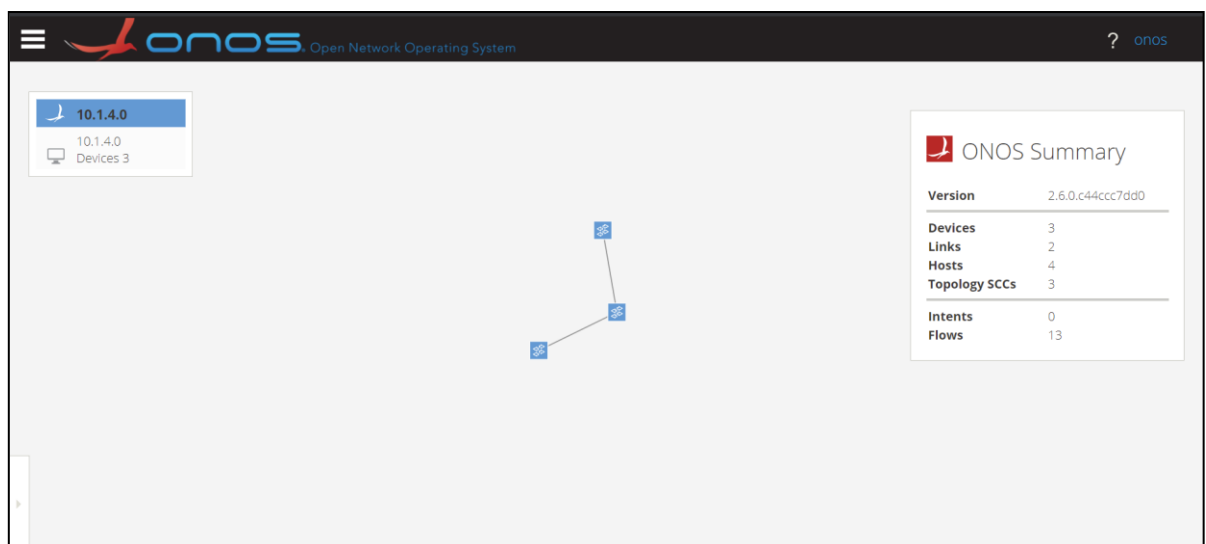


Figura 3: GUI del controlador ONOS

2.6 Lenguaje de programación Python

La aplicación desarrollada para la gestión de servicios en la red GPON del laboratorio está programada en el lenguaje de programación Python [10], por lo que a lo largo de este TFG se implementarán cambios en el código para adaptar la aplicación al nuevo escenario de red, ya que en este proyecto se va a emplear ONOS como controlador SDN en lugar de ODL (*OpenDayLight*), el cual era el que se había estado empleando hasta ahora.

Python es un lenguaje de programación de alto nivel e interpretado. Un lenguaje interpretado es aquel que no necesita una compilación previa del código completo antes de la ejecución, sino que a medida que se ejecuta, se va compilando línea a línea. Además, es multiparadigma, ya que soporta orientación a objetos, programación imperativa y programación funcional. Tiene una sintaxis muy simple y cuenta con un gran número de librerías apropiadas para la gestión de redes [10].

2.7 Metodología de trabajo

En este apartado del capítulo se va a describir la metodología y los pasos seguidos para la consecución de los objetivos propuestos para este trabajo.

2.7.1 Activación e integración de nuevos componentes PON

El objetivo en este caso es activar y configurar correctamente una pasarela IoT de manera que esta se pueda integrar fácilmente en la red de acceso GPON del laboratorio, conetándola a una de las ONTs de nivel 3 disponibles. Así mismo, se realizarán distintas pruebas mediante la conexión de distintos sensores a través de una placa de Arduino.

Por otro lado, se activará un nuevo puerto PON de la OLT, en el cual se registrarán posteriormente las distintas ONTs conectadas a la red de acceso con el objetivo de poder comprobar su correcto funcionamiento,

Por último, se va a activar el puerto 10G de la OLT, el cual es capaz de agregar en un único flujo 10G todo el tráfico de los 4 puertos PON, proporcionando el 100% del ancho de banda GPON. La activación de este nuevo puerto supone una gran actualización en el escenario SDN-GPON desplegado al comienzo de este trabajo.

2.7.2 Actualización del escenario SDN-GPON

En trabajos previos se ha implementado en la red GPON del laboratorio una solución SDN empleando el protocolo OpenFlow 1.3, por lo que es necesario emplear un controlador y varios *switches* virtuales para gestionar el tráfico de subida y de bajada de la red. El controlador empleado es *OpenDayLight* (ODL), el cual se encuentra instalado en una máquina virtual en el ordenador central del laboratorio. Para la implementación de los *switches* virtuales se han empleado OVSs, de modo que para controlar el tráfico de subida se utiliza un OVS en cada red local de los usuarios (entre la ONT y el usuario final) y para el tráfico de bajada se utiliza un OVS instalado en el ordenador central.

Tras la activación del puerto 10G de la OLT, el ordenador central del laboratorio se va a sustituir por un servidor, de modo que dicho servidor se conecta mediante un cable de fibra óptica al puerto 10G de la OLT. Por tanto, es necesario desplegar en el servidor todas las herramientas desplegadas en el ordenador central destinadas a la gestión SDN. Estas herramientas incluyen un *router*, un servidor DHCP, un OVS y un controlador SDN, que en este caso va a ser ONOS. Para el despliegue de todas estas herramientas se van a emplear distintos contenedores docker. Igualmente, se llevarán a cabo ciertas modificaciones SDN en el lado del cliente. Estas modificaciones incluyen el despliegue de nuevos OVSs en contenedores docker que sustituirán a los OVSs desplegados en máquinas virtuales, con el objetivo de actualizar la versión empleada.

Por último, para el despliegue de la aplicación de gestión de la red GPON en el servidor se crearán dos nuevos contenedores en el *chron2* mediante el empleo de una nueva herramienta denominada Docker-Compose, la cual permite crear y ejecutar aplicaciones en múltiples contenedores.

2.8 Conclusiones

En este primer capítulo de la memoria se ha presentado el entorno SDN que se encuentra desplegado sobre un testbed GPON situado en uno de los laboratorios de la escuela. Para ello, se han descrito las distintas herramientas *software* que se van a utilizar a lo largo de este proyecto, así como el lenguaje de programación con el que se va a trabajar y la metodología a seguir para la consecución de los objetivos propuestos, comenzando por la integración y activación de nuevos componentes PON y siguiendo con la actualización del escenario SDN-GPON de cara a futuros proyectos.

3

Activación e integración de componentes en la maqueta GPON

3.1 Introducción

En este capítulo de la memoria se va a realizar, en primer lugar, una descripción del entorno SDN-GPON inicial presente en la red del laboratorio. Posteriormente, se procederá a la activación de un nuevo puerto PON de la OLT y se realizarán pruebas para analizar su rendimiento y correcto funcionamiento. Además, se conectará a la red GPON una pasarela IoT, describiendo los pasos a seguir para su integración y configuración. Se realizarán también algunas pruebas con la misma mediante del empleo de distintos sensores conectados a través de una placa de Arduino. Por último, se procederá a la actualización de la red GPON mediante la activación del puerto de 10G de la OLT.

3.2 Descripción de la maqueta GPON y del entorno SDN-GPON inicial

A continuación, se va a describir la solución SDN desplegada en el laboratorio al comienzo de este proyecto. Para ello, se va a presentar la estructura y se van a describir los distintos componentes y los diferentes modos de gestión con los que cuenta la red GPON desplegada en el laboratorio.

La OLT de la red GPON del laboratorio da servicio a varias ONTs que están conectadas en una topología en árbol. El ordenador central, el cual está conectado a la OLT mediante un cable Ethernet a uno de sus 4 puertos *Gigabit Ethernet*, constituye la puerta de enlace del laboratorio con la red externa de la facultad. Este ordenador se conecta mediante otro cable Ethernet con el exterior y proporciona conexión a Internet a todos los dispositivos de la red de acceso, es decir, actúa como un *router*, para lo cual

hay creado un archivo de configuración ejecutable (*activar-enrutamiento.sh*) en el que se encuentran las instrucciones necesarias para tal efecto. En este archivo se definen también dos VLANs, la 833 y la 806.

Siguiendo con la topología hacia el interior de la red de acceso, el siguiente elemento que se encuentra es la OLT, la cual tiene dos conexiones de tipo Ethernet. Una de ellas, como se ha mencionado anteriormente, conecta la OLT con el ordenador central del laboratorio, dotando a la red de acceso de conexión a Internet. La segunda conexión también conecta la OLT con el ordenador central. No obstante, se trata de una subred diferente, la 172.26.128.0/24, usada para la gestión y configuración de la OLT mediante dos posibles métodos:

1. A través de una interfaz web proporcionada por el fabricante de los equipos y denominada TGMS (*TELNET GPON Management System*), un software que se ejecuta en una máquina virtual dentro del ordenador central y que permite la configuración de servicios para los clientes.
2. A través de la interfaz de línea de comandos CLI (*Command-Line Interface*), accesible mediante conexión *telnet* al puerto 4551 de la OLT.

La OLT tiene 4 puertos PON de salida, cada uno de ellos con una capacidad máxima de 64 ONTs, de modo que se pueden conectar hasta 256 ONTs simultáneamente, es decir, puede dar servicio a 256 clientes en total. En el caso que nos ocupa, el puerto que está en servicio es el PON 1, con 4 ONTs conectadas, siendo dos de ellas de nivel 2 (sin capacidad de enrutamiento) y las otras dos de nivel 3 (con capacidad de enrutamiento). De este puerto sale un cable de fibra óptica de 20 kilómetros de longitud, divididos en una bobina de 10 kilómetros y dos de 5 kilómetros. Seguidamente se encuentra un *splitter* 1:8 (se dispone de dos *splitters*, aunque actualmente solo se encuentra uno conectado), que divide la señal para dar servicio a las ONTs conectadas a través de una bobina de 5 kilómetros de longitud, compuesta por 48 fibras ópticas. A este *splitter* se conectan las cuatro ONTs anteriormente mencionadas. El aspecto general que presenta la red de acceso GPON es el que se muestra en la imagen de la Figura 4, mientras que la Figura 5 muestra la topología de la red desplegada.

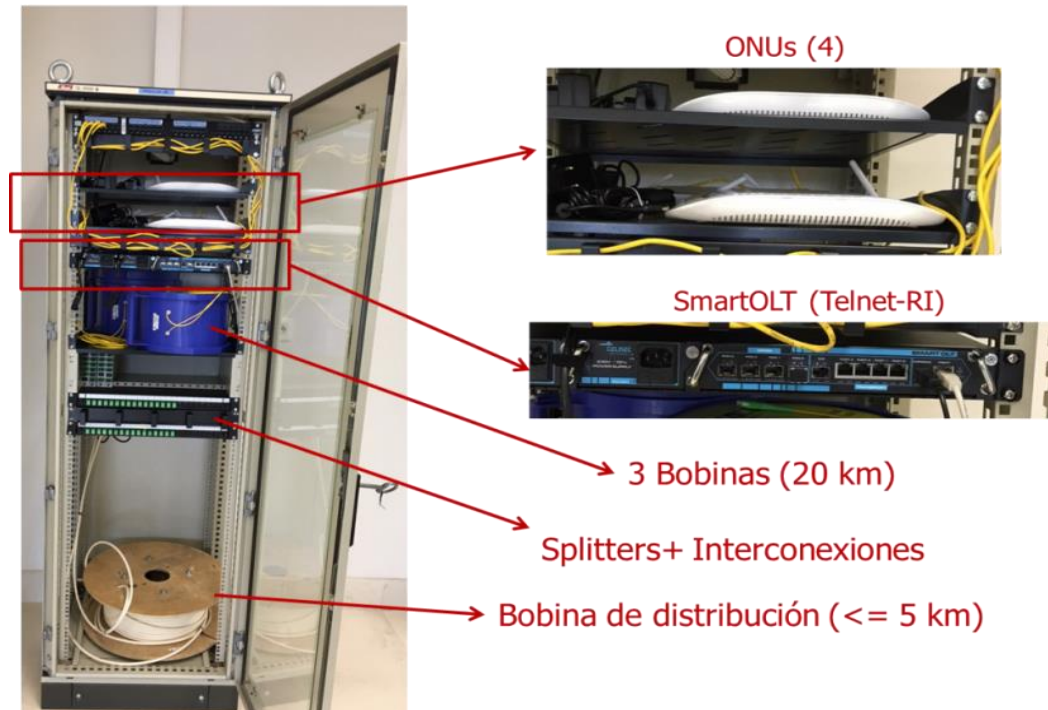


Figura 4: Aspecto real de la red de acceso GPON desplegada en el laboratorio

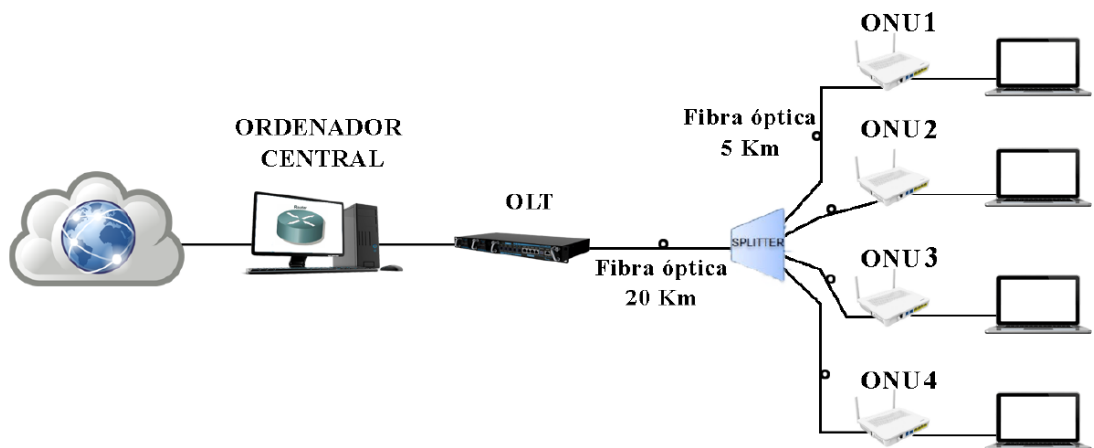


Figura 5: Topología en árbol de la red desplegada en el laboratorio

La interoperabilidad entre los diferentes dispositivos está garantizada, ya que todos ellos han sido fabricados por la empresa TELNET Redes Inteligentes [11]. Cada uno de los equipos que conforman la red GPON son los siguientes:

- **SmartOLT-350:** equipo utilizado en redes GPON que cuenta con 4 puertos PON que dan servicio a un total de 256 ONTs (64 por puerto). Cuenta con un puerto *Gigabit Ethernet* asociado a cada puerto PON, además de un puerto *FastEthernet* que soporta tasas de hasta 10 Gbps. Posee una gestión intuitiva a través de la interfaz web TGMS y utiliza el protocolo OMCI (*ONU Management and Control Interface*) para la gestión remota de las ONTs. Además, permite configurar servicios manualmente a través de CLI, empleando para ello comandos propios del estándar GPON
- **ONT WaveAccess 3021:** ONT de nivel 3 de alto rendimiento, con *router* integrado. Cuenta con 3 puertos *Gigabit Ethernet*, un puerto POTS destinado a la telefonía y servicio de Wifi b/g/n.
- **ONT WaveAccess 512:** ONT de nivel 2, es decir, sin capacidad de enrutamiento, por lo que necesita de un *router* a su salida que efectúe dicho trabajo. Fácilmente adaptable a cuadros eléctricos u oficinas gracias a su formato en carril DIN.

Los equipos descritos trabajan en dos canales diferentes, en función de la dirección de los datos que se transmiten a través de la red GPON:

- **Canal descendente:** denominado *downstream*, se trata de una comunicación punto-multipunto, ya que los datos viajan desde la OLT, situada en la Oficina Central del proveedor de servicios, hacia las ONTs, ubicadas en las viviendas de los clientes. Según las especificaciones del estándar GPON, emplea la longitud de onda de 1490 nm y permite una tasa máxima de 2.5 Gbps.
- **Canal ascendente:** denominado *upstream*, se trata de una comunicación punto-punto, ya que los datos viajan desde las diferentes ONTs hacia la OLT. Según las especificaciones del estándar GPON, emplea la longitud de onda de 1310 nm y permite una tasa máxima de 1.25 Gbps.

Para la configuración de servicios es importante tener siempre en cuenta la tasa máxima de cada uno de los canales, ya que la suma total de las tasas *upstream* y *downstream* dadas a cada uno de los servicios que se asignan a las diferentes ONTs no pueden exceder nunca las tasas máximas establecidas por el estándar GPON en ambos

canales de comunicación. En el caso de que se produzca esto, la red mostrará un error de configuración.

Sobre el testbed GPON descrito anteriormente hay desplegada una solución SDN, implementada mediante un controlador externo y varios *switches* virtuales. La solución SDN emplea como *switches* virtuales la tecnología OVS (*Open Virtual Switch*), los cuales están instalados en máquinas virtuales en varios ordenadores del laboratorio, y se usa como controlador *OpenDayLight* (ODL), el cual se encuentra instalado en una máquina virtual en el ordenador central del laboratorio. De este modo, el controlador SDN, gestionado por el operador de red, es capaz de modificar dinámicamente y en tiempo real servicios, adecuándolos a la capacidad de la red GPON, trabajando con el tráfico en ambos canales, tanto ascendente como descendente, y atendiendo a los diferentes requisitos de QoS contratados por los clientes, como puede ser un ancho de banda garantizado asociado a los clientes dados de alta en la red GPON.

Para la solución SDN-GPON desplegada en el laboratorio, el OVS central (COVS), que controla el tráfico del canal descendente, se encuentra instalado en el ordenador central, situado junto antes de la OLT, emulando la capa SDN de la OLT. Por su parte, en el lado de los usuarios, justo a la entrada de cada ONT se integra un OVS remoto (ROVS), capaz de controlar el tráfico en el canal ascendente que sale de la red residencial del usuario. La arquitectura SDN-GPON descrita se puede ver en la Figura 6.

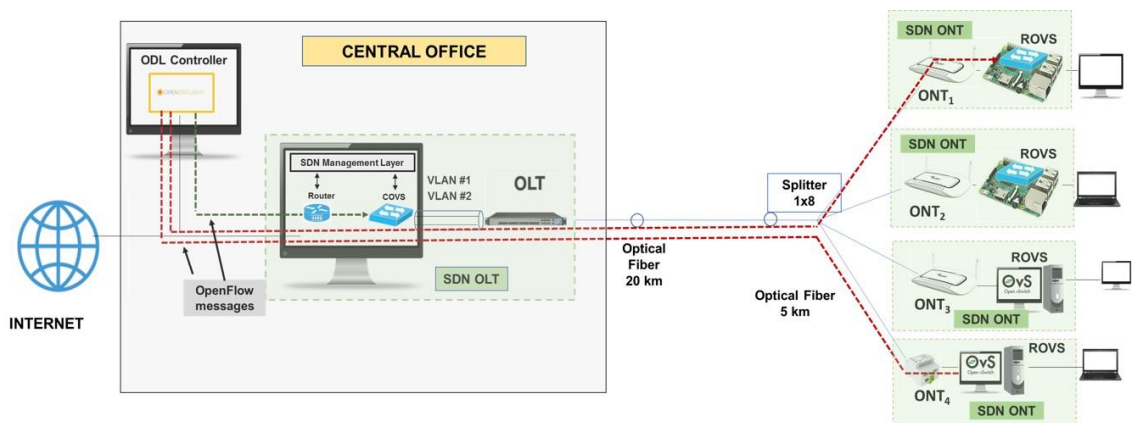


Figura 6: Esquema del escenario SDN-GPON desplegado sobre la red GPON del laboratorio

De este modo, el controlador ODL establece una configuración SDN a través del envío de mensajes *OpenFlow* a los diferentes ROVS y al COVS. En primer lugar, el controlador envía las tablas *OpenFlow* a los OVS con una configuración inicial, para posteriormente poder modificar sus entradas en tiempo real en función de los servicios demandados por los usuarios o cambios en sus requisitos de QoS. De este modo, para cada servicio desplegado se crearán dos *flows*, uno para el tráfico en el canal descendente y otro para el tráfico del canal ascendente.

Dentro del escenario de red SDN-GPON desplegado se puede controlar, por ejemplo, el ancho de banda máximo asignado a cada servicio/usuario a través de *meters*. Estos *meters* se asocian directamente con los *flows*, y permiten controlar la tasa máxima de los paquetes que tienen asignados dichos *flows*. Cada *meter* consta de un identificador y una banda, que especifica la velocidad asociada al *flow* y la forma de procesar los paquetes del *flow*. La velocidad de los datos es medida continuamente, de modo que si dicha tasa está por encima de la establecida para el *flow* se descartan paquetes, controlando, de este modo, el ancho de banda máximo asociado a dicho *flow*. Dado que los usuarios finales desconocen estos conceptos de bajo nivel, este proceso se resume en comandos de nivel superior entendibles para los usuarios.

Como ya se ha comentado anteriormente, la red GPON dispone, en el lado del cliente, de ONTs tanto de nivel 2 como de nivel 3. Para la solución SDN desplegada se han utilizado las ONTs de nivel 2, ya que proporcionan una mayor flexibilidad al no contar con *routers* integrados, permitiéndonos desarrollar capacidades de enrutamiento adaptadas a las necesidades y requisitos que se deseen desplegar. De este modo, cada ROVS necesita un servidor DHCP asociado que proporcione direccionamiento IP a los dispositivos conectados a la ONT correspondiente. Por ello, el servidor DHCP global, configurado en el ordenador central del laboratorio y controlado por el operador, asigna un rango de direcciones a cada servidor DHCP local para que puedan asignar una dirección IP a cada dispositivo conectado. De esta manera, cada dispositivo conectado a la red tendrá una dirección IP única.

3.3 Activación de nuevos puertos PON en la OLT

En este apartado de la memoria se va a describir el proceso de activación y puesta en marcha de uno de los puertos PON de la OLT, en concreto, el puerto PON 3. Además, se realizarán distintas pruebas para comprobar su rendimiento. Como se ha mencionado con anterioridad, el modelo de OLT empleado en la red de acceso SmartOLT 350 [12] consta de 4 puertos PON, y hasta ahora, únicamente se han activado dos de los puertos, concretamente el PON 0 y el PON 1. Si tenemos conectado el puerto PON 3 de la OLT, el cable que conecta la OLT con el ordenador central del laboratorio debe estar conectado en el puerto Ethernet correspondiente, es decir, en el puerto 3. En la Figura 7 podemos ver la parte de la OLT donde se encuentran los puertos PON y sus correspondientes puertos Ethernet.



Figura 7: Imagen frontal de la OLT con los puertos PON y los puertos Ethernet

3.3.1 Registro de las ONTs en puertos PON mediante TGMS

Una vez realizada correctamente la conectividad entre puertos, el siguiente paso es registrar las ONTs en dicho puerto PON a través del TGMS. El proceso de registro de las ONTs en TGMS es sencillo. Los pasos a se describen a continuación:

1. En el ordenador central, abrimos una terminal y nos identificamos como super-usuarios. Para ello, ejecutamos el comando `sudo su -` e introducimos la contraseña. A continuación, abrimos virtualbox desde la terminal, para lo cual ejecutamos el comando `virtualbox&`.
2. Una vez dentro de virtualbox, iniciamos la máquina virtual del TGMS. Esta tarda unos minutos en arrancar y una vez lo haga, se deja abierta. No es necesario introducir ni usuario ni contraseña.

3. Abrimos un navegador y escribimos la dirección 172.26.128.1. Se mostrará entonces el entorno web de TGMS, en el cual tenemos que introducir las credenciales de inicio, esto es, el usuario y la contraseña (*root/management*).
4. A continuación, aparece el menú principal del TGMS, donde podemos visualizar nuestra OLT. Seleccionamos *PON 0*, que es el nombre dado a nuestra OLT, seguidamente *Port: 3*, que en este caso es el puerto que tenemos conectado, y finalmente *Details*. Esto se puede ver en la Figura 8.

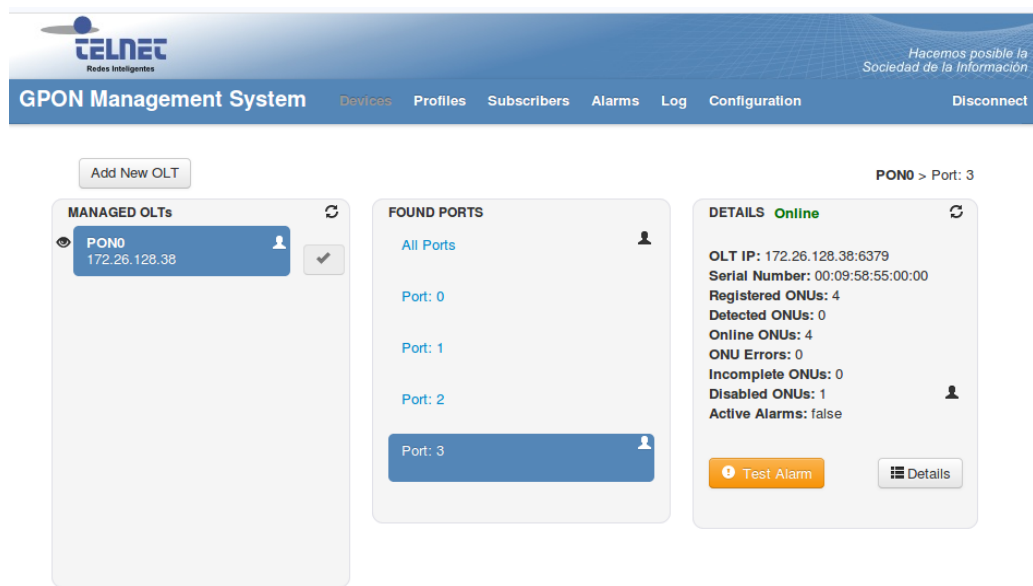


Figura 8: Acceso en TGMS a los detalles del puerto 3 de la OLT

5. A continuación, se muestra una pestaña donde nos aparecen las ONTs que tenemos registradas en el nuevo puerto. Dado que nunca se ha activado este puerto, esta pestaña aparece vacía. Las ONTs que tenemos conectadas a la red GPON, pero que aún no están asociadas al puerto, las podemos ver en la pestaña *Disabled ONUs*, como se observa en la Figura 9.

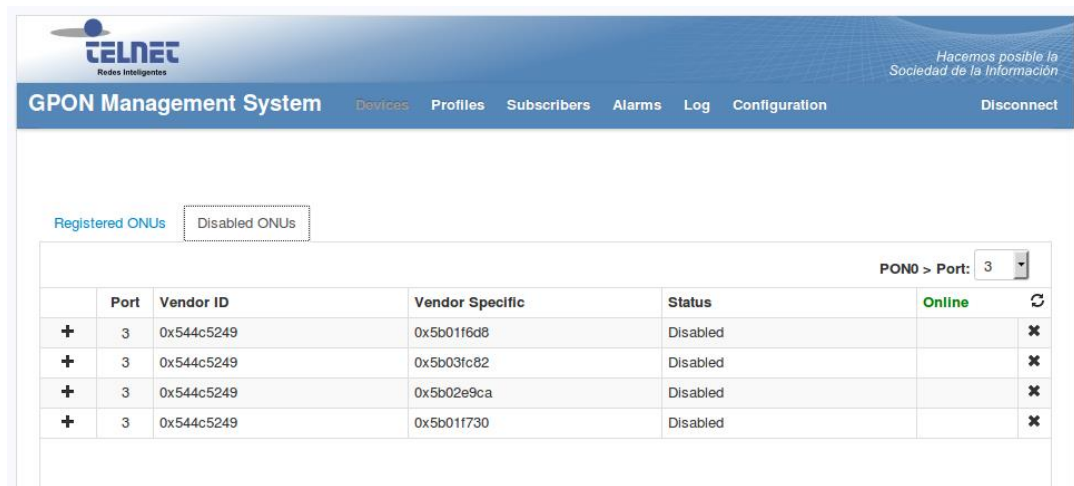


Figura 9: Pestaña con las ONUs que están conectadas a la red GPON (*Disabled ONUs*) pero no asociadas al puerto Port 3

Como se puede observar en la imagen superior, se asigna el mismo *Vendor ID* a todas las ONTs conectadas, pero el parámetro *Vendor Specific* es identificativo para cada una y se corresponde con parte de su dirección MAC. Posteriormente, se usará este parámetro para registrar cada ONT.

6. A continuación, volvemos a la pestaña *Registered ONUs* y pinchamos en el símbolo +, el cual se localiza arriba a la izquierda. Nos aparece entonces una ventana emergente como la que se muestra en la Figura 10.

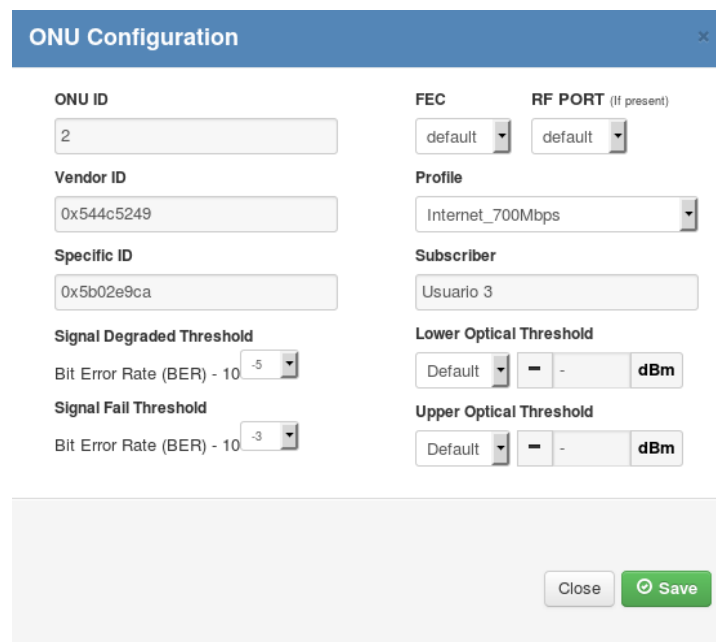
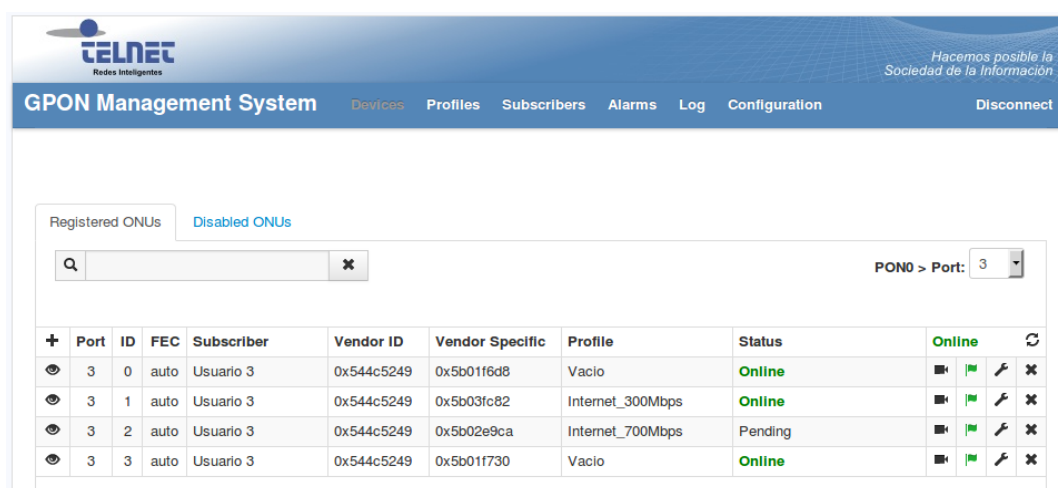


Figura 10: Ventana para configurar la ONU “ONU Configuration”

En esta ventana de configuración de la ONT seleccionamos un *Profile*, el cual se corresponde con el perfil de servicios que se desee asignar al usuario. En este caso concreto seleccionamos el perfil *Internet_700Mbps*. En el campo *Specific ID* introducimos el parámetro *Vendor Specific*, el cual se ha comentado anteriormente, y finalmente en el campo *Subscriber* le ponemos un nombre unívoco, en nuestro caso *Usuario 3*, aunque puede ser cualquier otro. El resto de parámetros se dejan como están, en caso de que no se quieran modificar las características de la tasa de error de bit (BER, *Bit Error Rate*) o FEC (*Forward Error Correction*). Finalmente, guardamos los cambios pulsando en el botón *Save*. Este proceso se repite para cada una de las ONTs detectadas por la red GPON.

- Una vez tengamos todas las ONTs registradas en el nuevo puerto activo, su estado aparece como *Pending*, tal y como podemos ver en la Figura 11.



+	Port	ID	FEC	Subscriber	Vendor ID	Vendor Specific	Profile	Status	Online	
👁	3	0	auto	Usuario 3	0x544c5249	0x5b0116d8	Vacio	Online	■	🔧 ✕
👁	3	1	auto	Usuario 3	0x544c5249	0x5b03fc82	Internet_300Mbps	Online	■	🔧 ✕
👁	3	2	auto	Usuario 3	0x544c5249	0x5b02e9ca	Internet_700Mbps	Pending	■	🔧 ✕
👁	3	3	auto	Usuario 3	0x544c5249	0x5b011730	Vacio	Online	■	🔧 ✕

Figura 11: Estado inicial en el que aparecen las ONUs registradas (estado *Pending*)

Para que su estado cambie a *Online*, tenemos que ir al menú principal del TGMS y sincronizar la OLT, de modo que se apliquen los cambios realizados, en este caso el registro de nuevas ONTs. Para ello, se pulsa sobre el botón que nos aparece a la derecha de *PON 0*, como se observa en la Figura 12, que es el nombre asignado a nuestra OLT.

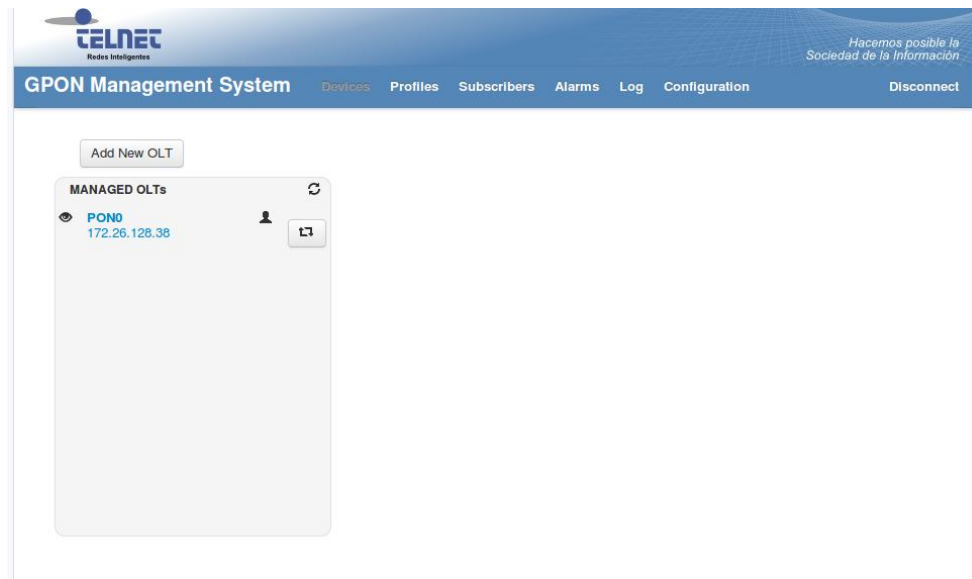


Figura 12: OLT pendiente de sincronización debido al registro de nuevas ONUs

8. Una vez realizado el proceso de sincronización de la OLT, volvemos a la pestaña donde aparecen las ONTs que tenemos registradas en el puerto para analizar su estado, siguiendo para ello los pasos descritos en el punto 4. Observamos que el estado de todas las ONTs que se han registrado es *Online*. Por tanto, ahora ya están registradas y operativas todas las ONTs que tenemos conectadas a la red GPON en el nuevo puerto. En la Figura 13 podemos ver el resultado.

+	Port	ID	FEC	Subscriber	Vendor ID	Vendor Specific	Profile	Status	Online	
👁	3	0	auto	Usuario 3	0x544c5249	0x5b01f6d8	Vacio	Online	🟢	🔧 ✖
👁	3	1	auto	Usuario 3	0x544c5249	0x5b03fc82	Internet_300Mbps	Online	🟢	🔧 ✖
👁	3	2	auto	Usuario 3	0x544c5249	0x5b02e9ca	Internet_700Mbps	Online	🟢	🔧 ✖
👁	3	3	auto	Usuario 3	0x544c5249	0x5b01f730	Vacio	Online	🟢	🔧 ✖

Figura 13: Imagen en la que se observan todas las ONUs que están Online

3.3.2 Prueba de funcionamiento y conectividad del nuevo puerto

Para comprobar el correcto funcionamiento del nuevo puerto (Port 3), se puede realizar una sencilla prueba *end-to-end* con cualquiera de las ONTs que están conectadas a la red del laboratorio y registradas en TGMS. En este caso concreto, se ha utilizado una ONT de nivel 3 para realizar la prueba. El proceso se describe a continuación.

Para la realización de esta prueba es necesario asignar previamente a la ONT correspondiente un perfil de Internet mediante TGMS. En este caso, el servicio seleccionado es *Internet_300Mbps*. Para simular una topología *end-to-end* en la red se han utilizado dos portátiles Lenovo disponibles en el laboratorio: uno conectado al ordenador central del laboratorio, actuando a modo de servidor, y otro conectado a la ONT de nivel 3, simulando un cliente. La topología resultante es la que se puede ver en la Figura 14.

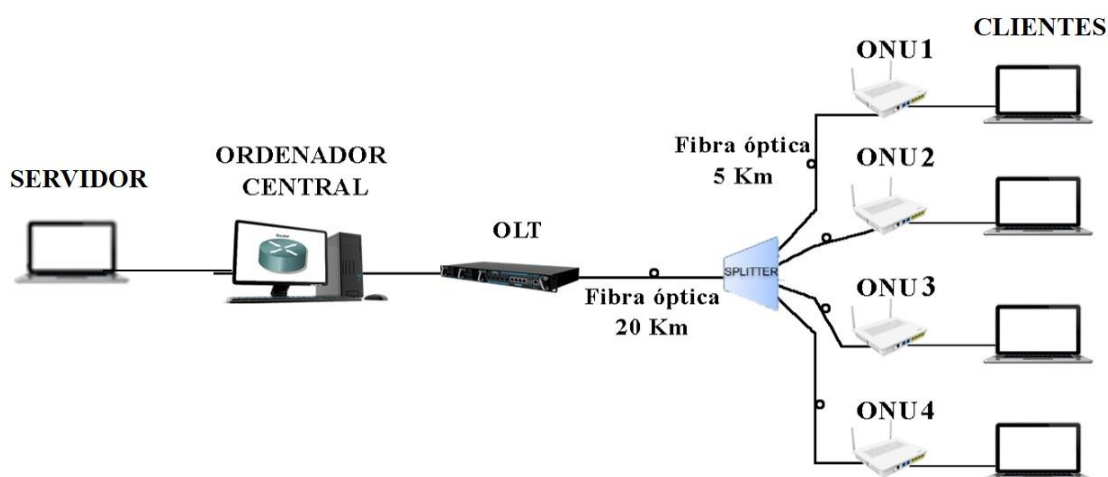


Figura 14: Topología para la prueba del Puerto 3 de la OLT

En el Lenovo empleado como servidor abrimos una terminal, nos identificamos como super-usuario y ejecutamos el script *activarServidor.sh* mediante el comando *sh activarServidor.sh*. De este modo se levanta la VLAN 833, se le asigna la dirección IP 10.0.103.60 a la interfaz que conecta el Lenovo con el ordenador central y se añade en la tabla de encaminamiento la ruta estática necesaria. A continuación, en el ordenador central, ejecutamos de la misma forma el script *activar-enrutamiento.sh*, que sirve para levantar la interfaz *enp4s1*, definir la VLAN 833 y hacer que el ordenador central funcione como un *router*. Además, también se inicializa el servidor DHCP que va a asignar direcciones IP a los distintos dispositivos de la red de acceso.

A continuación, se conecta el Lenovo que se emplea como cliente en la línea del laboratorio en la que está conectada la ONT elegida anteriormente, para que de este modo el portátil Lenovo esté conectado a internet a través de dicha ONT. Posteriormente, se debe configurar el *router* de la ONT de nivel 3 y sincronizar dicha configuración con la OLT a través del TGMS. Para ello, se abre un navegador en el portátil y se escribe la dirección IP del *router*, en este caso, 192.168.3.1 (el número marcado en negrita es distinto para cada ONT y se corresponde con el identificador del *router* de dicha ONT). Se muestra entonces la interfaz web del *router*. Para acceder introducimos el usuario y la contraseña. En el menú principal hay que seleccionar *Device Info* → *WAN* para comprobar si hay algún servicio de Internet configurado en el *router*. En caso de que no fuese así, sería necesario configurar uno. Como se puede ver en la Figura 15, ya tenemos un servicio configurado para el *router*. Es necesario que la propiedad *Status* aparezca en estado *Connected* para el interfaz asociado al servicio configurado.

WAN Info													
Interface	Description	Type	VlanMuxId	IPv6	Igmp Pxy	Igmp Enbl	MLD Pxy	MLD Src Enbl	NAT	Firewall	Status	IPv4 Address	IPv6 Address
veip0.1	ipoe_veip0.833	IPoE	833	Disabled	Disabled	Disabled	Disabled	Disabled	Enabled	Disabled	Connected	192.168.0.142	

Figura 15: Captura de pantalla de la interfaz de configuración del router de la ONT (*WAN Info*)

Dado que las ONTs de nivel 3 cuentan con un *router* integrado, este separa la red de acceso de la red del cliente en dos subredes distintas. Por ello, la conexión solo es viable en el sentido ascendente, es decir, desde el cliente (Lenovo conectado a la ONT) hacia el servidor (Lenovo conectado al ordenador central del laboratorio). Para poder tener conexión en el otro sentido, es decir, desde el servidor hacia el cliente y que, por lo tanto, este sea visible más allá del *router*, tenemos que agregar de manera manual una ruta estática en la tabla de encaminamiento del ordenador central del laboratorio. De esta manera, el ordenador central sabe por donde debe encaminar los paquetes que tengan como destino la dirección del cliente. Para ello, podemos emplear tanto el comando “*route add*” como el comando “*ip route add*”, tal y como se muestra a continuación:

1. `route add -net 192.168.3.0 netmask 255.255.255.0 gw 192.168.0.142`

2. `ip route add 192.168.3.0/24 via 192.168.0.142`

donde se indica la red que se quiere alcanzar, que en este caso es la que se encuentra al otro lado de la ONT, y a través de que dirección la vamos a alcanzar, que en este caso

sería la dirección IP del *router* en la red de acceso. Dicha dirección se encuentra en la *WAN Info*, como se puede observar en la Figura 15. Para ver si se ha añadido correctamente la entrada en la tabla de encaminamiento del ordenador central del laboratorio podemos ejecutar el comando *route -n*.

Para comprobar la conectividad *end-to-end* se puede hacer uso de la herramienta *iperf/iperf3* [13] en cualquiera de las dos direcciones. Se trata de una herramienta de código abierto empleada para analizar el rendimiento de las redes. Normalmente, en un entorno de prueba, el cliente *iperf* envía datos al servidor para la prueba. En nuestro caso, esto sería desde el Lenovo del cliente hacia el Lenovo que actúa como servidor.

Los resultados son variables en función del ancho de banda garantizado que esté definido para el perfil de Internet que se asigna a la ONT. Así pues, en caso de que la tasa que se transmite con *iperf/iperf3* sea inferior a dicho ancho de banda, la capa óptica de la red GPON permite que todo lo que estamos transmitiendo llegue al otro extremo de la red. Sin embargo, si transmitimos una tasa superior al ancho de banda asignado a la ONT, esta misma capa hace que la tasa de transmisión que llega al otro extremo de la red sea la máxima que tiene definida en su configuración.

Para la realización de la prueba *end-to-end* empleando la herramienta *iperf/iperf3* [14] podemos utilizar tanto una conexión UDP como una conexión TCP, en función de nuestras necesidades. Si empleamos una conexión TCP tenemos que seguir los siguientes pasos:

1. En primer lugar, tenemos que poner uno de los dos portátiles Lenovo en modo escucha. En nuestro caso, cual elijamos es indiferente, ya que tenemos conexión en ambos sentidos. No obstante, lo idóneo sería emplear el Lenovo que actúa como servidor, ya que de este modo no es necesario estar modificando la tabla de encaminamiento del ordenador central del laboratorio. Por tanto, abrimos una terminal en este portátil e introducimos el comando *iperf3 -s*.
2. Desde el portátil del cliente abrimos una terminal y ejecutamos el comando *iperf3 -c IP_servidor*. En nuestro caso, dado que la IP del Lenovo que actúa como servidor es 10.0.103.60 el comando sería “*iperf3 -c 10.0.103.60*”. Empleando *iperf3* podemos ver la ejecución en tiempo real. Si empleamos *iperf*,

tendremos que esperar unos segundos para poder visualizar el ancho de banda de la conexión

```

administrador@administrador-ThinkPad-L380:~$ iperf3 -c 10.0.103.60
Connecting to host 10.0.103.60, port 5201
[ 4] local 192.168.3.2 port 37092 connected to 10.0.103.60 port 5201
[ ID] Interval      Transfer    Bandwidth   Retr  Cwnd
[ 4]  0.00-1.00    sec  46.3 MBytes  388 Mbits/sec  16   187 KBytes
[ 4]  1.00-2.00    sec  43.9 MBytes  369 Mbits/sec   9   218 KBytes
[ 4]  2.00-3.00    sec  44.0 MBytes  369 Mbits/sec   7   185 KBytes
[ 4]  3.00-4.00    sec  43.9 MBytes  369 Mbits/sec   4   214 KBytes
[ 4]  4.00-5.00    sec  44.0 MBytes  369 Mbits/sec  10   178 KBytes
[ 4]  5.00-6.00    sec  43.9 MBytes  369 Mbits/sec   6   206 KBytes
[ 4]  6.00-7.00    sec  43.9 MBytes  369 Mbits/sec   6   173 KBytes
[ 4]  7.00-8.00    sec  44.0 MBytes  369 Mbits/sec   7   197 KBytes
[ 4]  8.00-9.00    sec  43.9 MBytes  368 Mbits/sec   5   209 KBytes
[ 4]  9.00-10.00   sec  44.0 MBytes  369 Mbits/sec   6   222 KBytes
-----
[ ID] Interval      Transfer    Bandwidth   Retr
[ 4]  0.00-10.00   sec  442 MBytes  371 Mbits/sec  76
[ 4]  0.00-10.00   sec  441 MBytes  370 Mbits/sec
sender
receiver

```

Figura 16: Prueba de conectividad *end-to-end* empleando una conexión TCP

Tal y como se puede observar en la Figura 16, el ancho de banda de la conexión empleando un perfil de Internet de 300 Mbps en TGMS es de 370 Mbps, ligeramente superior. No obstante, esto es algo normal debido al funcionamiento de *iperf3* en determinados escenarios, de modo que podemos concluir que el nuevo puerto funciona correctamente.

Si por el contrario queremos emplear una conexión UDP, los comandos a emplear son ligeramente distintos. Tenemos que indicar, tanto en el portátil que actúa como servidor como en el que utilizamos como cliente, que dicha conexión es UDP. En el servidor introducimos el comando *iperf -s -u*, y en el cliente *iperf -c 10.0.103.60 -u -b 500m*, siendo el valor del parámetro que sigue a la opción *-b* superior al ancho de banda que esté definido para el perfil de Internet asignado a la ONT. De esta manera, no se limita el ancho de banda de la conexión. Al emplear una conexión UDP, *iperf3* no funciona correctamente, ya que no marca el ancho de banda real que hay. En este caso es necesario emplear la versión *iperf* para solventar este problema. Los resultados obtenidos son similares a los de la conexión TCP que se muestra en la Figura 16.

3.4 Integración de una pasarela IoT en la red GPON

A continuación, se van a describir los pasos llevados a cabo para la activación y configuración de la pasarela IoT BabelGate G5002 [15] de la empresa TELNET Redes Inteligentes. También se mostrarán algunas pruebas realizadas mediante el empleo de

distintos sensores conectados a través de una placa de Arduino. La pasarela es la que se puede observar en la Figura 17



Figura 17: Pasarela IoT BabelGate G5002

En primer lugar, para acceder a la pasarela IoT es necesario establecer una conexión *ssh*. Dado que es la primera vez que nos conectamos a la pasarela, vamos a emplear uno de los Lenovos disponibles en el laboratorio conectando mediante un cable Ethernet al puerto 1 de la pasarela. No obstante, antes de acceder mediante *ssh* es necesario configurar la interfaz de red del Lenovo para que el portátil esté en la misma red que la pasarela. Por defecto, la dirección IP de la pasarela es 192.168.1.10. Por tanto, damos al Lenovo una dirección IP dentro de la red 192.168.1.0/24, para lo cual ejecutamos el comando `sudo ifconfig enp0s31f6 192.168.1.5 netmask 255.255.255.0`.

Mediante la ejecución de este comando asignamos la dirección IP 192.168.1.5 a la interfaz de red enp0s31f6 del Lenovo, de modo que ahora ambos dispositivos están en la misma red.

Una vez configurada la interfaz de red del portátil, ya podemos acceder a la pasarela mediante *ssh*, para lo cual ejecutamos el comando `ssh telnet@192.168.1.10` (contraseña: telnet). En la Figura 18 se puede ver el establecimiento de la conexión.

```
administrador@administrador-ThinkPad-L380:~$ sudo ifconfig enp0s31f6 192.168.1.5 netmask 255.255.255.0
administrador@administrador-ThinkPad-L380:~$ ssh telnet@192.168.1.10
telnet@192.168.1.10's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-209-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

19 packages can be updated.
9 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Mon May 31 10:02:55 2021 from 192.168.1.5
telnet@G5002:~$
```

Figura 18: Conexión mediante *ssh* con la pasarela IoT

Dentro de la pasarela está instalado *Ubuntu 16.04.3 LTS*. Lo primero que vamos a hacer es instalar las actualizaciones necesarias. No obstante, para poder hacer esto es necesario tener conexión a Internet desde la pasarela. Por tanto, damos previamente en TGMS un perfil de Internet a una de las ONTs de nivel 3 del laboratorio y nos conectamos con la pasarela mediante un cable Ethernet a dicha ONT, empleando para ello el segundo puerto. Para poder tener conexión con la ONT de nivel 3, y por tanto, acceso a Internet, tenemos que levantar la segunda interfaz de red de la pasarela. Para ello, ejecutamos el comando *sudo ifconfig enp3s0 up*. Una vez levantada la interfaz, le solicitamos al servidor DHCP de la ONT de nivel 3 una dirección IP en la red, para lo cual ejecutamos el comando *sudo dhclient enp3s0*.

Con esto, ya tenemos configurada esta segunda interfaz de red. Lo único que nos queda para poder tener conexión a Internet es configurar correctamente la puerta de enlace en la tabla de encaminamiento. Por defecto, la puerta de enlace está definida para la interfaz *enp2s0*, interfaz por la que hemos accedido a la pasarela mediante *ssh*. Por tanto, agregamos una nueva puerta de enlace mediante el comando *route add -net default gw 192.168.2.1* (el número marcado en negrita es distinto para cada ONT y se corresponde con el identificador del *router* de dicha ONT).

De esta manera, ya tenemos conexión a Internet. Finalmente, para actualizar la pasarela ejecutamos los siguientes comandos:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```


Primero actualizamos el repositorio y posteriormente instalamos las actualizaciones necesarias. Con esto, ya tenemos activada la pasarela IoT y actualizado el sistema operativo.

Por último, vamos a definir la segunda interfaz de red de la pasarela, la interfaz `enp3s0`, en el archivo `/etc/network/interfaces` para automatizar su proceso de configuración. De este modo, cada vez que conectemos la pasarela a una de las ONTs de nivel 3 mediante el segundo puerto Ethernet, la interfaz se configurará de manera automática. Esto permite integrar fácilmente la pasarela en la red GPON. Para realizar esta configuración, ejecutamos el comando `sudo nano /etc/network/interfaces` para poder editar el archivo y agregamos las siguientes líneas:

```
# The secondary network interface  
  
auto enp3s0  
  
iface enp3s0 inet dhcp
```

De esta manera, la ONT de nivel 3 asigna de manera automática una dirección IP a la pasarela al conectarla a la red. Al configurar esta segunda interfaz de esta manera, es decir, editando el archivo `/etc/network/interfaces`, podemos acceder a la pasarela mediante una conexión `ssh` a través de la dirección IP asignada por la ONT de nivel 3 a esta interfaz, siempre y cuando nosotros estemos conectados a la misma red, es decir, a la misma ONT de nivel 3. De este modo, solo es necesario tener conectado uno de los puertos Ethernet de la pasarela.

Para las pruebas realizadas con la pasarela IoT hemos empleado distintos sensores conectados a través de una placa de Arduino por el puerto USB del que dispone la pasarela. No obstante, para poder comunicarnos con la placa de Arduino necesitamos instalar el compilador de Arduino. Para ello, ejecutamos los siguientes comandos:

```
sudo apt-get update  
  
sudo apt-get instal -y arduino-mk
```

Con el primer comando actualizamos el repositorio y con el segundo instalamos el paquete `arduino-mk`. Este paquete nos va a permitir compilar un código escrito en Arduino y subirlo a la placa, de modo que los distintos sensores que tengamos conectados puedan funcionar correctamente y empezar a capturar información del ambiente.

Una vez instalado el compilador de Arduino, creamos una carpeta denominada *sketchbook* en nuestro directorio actual, por defecto */home/telnet*. En esta carpeta vamos a guardar los distintos programas que desarrollemos. Así mismo, creamos dentro de este nuevo directorio una carpeta denominada *libraries* en la cual vamos a almacenar las librerías necesarias para compilar nuestros programas.

Para la primera prueba hemos escrito un programa muy sencillo que nos permite encender y apagar un led. El objetivo de esta primera prueba es simplemente comprobar que el compilador de Arduino funciona correctamente y nos permite comunicarnos con la placa. Para ello, creamos el directorio */sketchbook/LED*, en la cual vamos a guardar el código del programa desarrollado y el archivo Makefile necesario para compilar el mismo. El código del programa se encuentra en un archivo denominado *led.ino*. Su contenido es el que se muestra en la Figura 19.

```
const int LED=13;
void setup()
{
  pinMode(LED,OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  digitalWrite(LED,HIGH);
  Serial.println("ON");
  delay(2000);

  digitalWrite(LED,LOW);
  Serial.println("OFF");
  delay(2000);
}
```

Figura 19: Código del programa que permite encender y apagar un led

Para poder compilar este programa y subirlo a la placa de Arduino creamos un archivo denominado *Makefile* cuyo contenido es el que se puede ver en la Figura 20.

```
ARDUINO_DIR = /usr/share/arduino

BOARD_TAG = uno
ARDUINO_PORT = /dev/ttyACM*

USER_LIB_PATH = /home/telnet/sketchbook/libraries

include /usr/share/arduino/Arduino.mk
```

Figura 20: Contenido del archivo Makefile

En este archivo definimos el directorio en el que se localizan los ficheros correspondientes al paquete *arduino-mk*, el modelo de la placa de Arduino que vamos a emplear, el puerto por el cual nos vamos a conectar a la placa de Arduino, en nuestro caso el puerto USB, y el directorio en donde se encuentran las librerías que se necesiten y que no vengan por defecto en el directorio de Arduino. En este caso concreto, no se necesita ninguna librería. Por último, incluimos el archivo *Arduino.mk* necesario para compilar el código.

Una vez definidos estos dos archivos, para poder compilar el código y subirlo a la placa de Arduino necesitamos dar permisos al puerto USB de la pasarela, para lo cual ejecutamos el comando `sudo chmod 666 /dev/ttyACM*`. Para poder visualizar la ejecución del programa en una pantalla es necesario instalar el paquete *screen*, para lo cual ejecutamos el comando `sudo apt-get install -y screen`.

Llegados a este punto, para compilar el programa, subirlo a la placa y visualizar su ejecución por pantalla ejecutamos, dentro del directorio */sketchbook/LED*, el comando `make upload monitor clean`. El resultado de la ejecución se muestra en la Figura 21.

```
ON
OFF
ON
OFF
ON
OFF
```

Figura 21: Visualización de la ejecución del programa

Para salir de la pantalla ejecutamos la secuencia CTRL-a CTRL-d. Sin embargo, esta pantalla aún se está ejecutando en segundo plano. Esto lo podemos ver empleando el comando `screen -list`. Si queremos volver a visualizar la pantalla ejecutamos el comando `screen -r`. Si por el contrario queremos cerrar esta pantalla ejecutamos el comando `screen -X quit`. Este último comando es necesario ejecutarlo siempre, ya que si no se cierra la pantalla no podremos subir un código distinto a la placa de Arduino.

Una vez visto que el compilador de Arduino funciona correctamente, vamos a añadir al montaje anterior el sensor DHT11 [16], un sensor que permite medir la temperatura y la humedad. Para poder emplear el sensor, tenemos que descargarnos la librería correspondiente. Para ello, nos situamos dentro del directorio */sketchbook/libraries* y ejecutamos los siguientes comandos:

```
wget -c https://github.com/adafruit/DHT-sensor-library/archive/1.2.3.zip  
unzip 1.2.3.zip  
mv DHT-sensor-library-1.2.3 DHT
```

Con el primer comando nos descargamos el archivo .zip correspondiente a la librería, con el segundo descomprimos el archivo y con el tercero renombramos el directorio, ya que los nombres de las librerías no pueden contener guiones. Una vez hecho esto, creamos el directorio `/sketchbook/LED_DHT11`, en el cual vamos a guardar el código del programa desarrollado y el archivo Makefile necesario para compilar el mismo. El código del programa se encuentra en un archivo denominado `Led_DHT11.ino`. Su contenido es el que se muestra en la Figura 22.

```
//Incluimos la libreria  
#include <DHT.h>  
  
//Definimos el pin digital donde se conecta el sensor  
#define DHTPIN 2  
  
//Dependiendo del tipo de sensor  
#define DHTTYPE DHT11  
  
//Inicializamos el sensor DHT11  
DHT dht(DHTPIN, DHTTYPE);  
const int LED=13;  
  
void setup()  
{  
  pinMode(LED,OUTPUT);  
  Serial.begin(9600);  
  dht.begin();  
}  
  
void loop()  
{  
  digitalWrite(LED,HIGH);  
  Serial.println("ON");  
  delay(3000);  
  
  //Leemos la humedad relativa  
  float h = dht.readHumidity();  
  
  //Leemos la temperatura en grados centigrados (por defecto)  
  float t = dht.readTemperature();  
  
  Serial.print("Humedad: ");  
  Serial.print(h);  
  Serial.println(" %\t ");  
  Serial.print("Temperatura: ");  
  Serial.print(t);  
  Serial.println(" *C ");  
  
  digitalWrite(LED,LOW);  
  Serial.println("OFF");  
  delay(2000);  
}
```

Figura 22: Código del programa que permite encender y apagar un led y medir la temperatura y la humedad del ambiente

Para poder compilar este programa y subirlo a la placa de Arduino creamos un archivo denominado *Makefile* cuyo contenido es el que se puede ver en la Figura 23.

```
ARDUINO_DIR = /usr/share/arduino

BOARD_TAG = uno
ARDUINO_PORT = /dev/ttyACM*

USER_LIB_PATH = /home/telnet/sketchbook/libraries/
ARDUINO_LIBS = DHT

include /usr/share/arduino/Arduino.mk
```

Figura 23: Contenido del archivo Makefile

A diferencia del archivo *Makefile* creado para el programa anterior, en este caso, dado que hemos incorporado el sensor DHT11 al montaje, tenemos que incluir la librería correspondiente. Para ello, es necesario indicar el nombre del directorio en el cual se encuentran los ficheros correspondientes a la librería. En nuestro caso, el nombre del directorio es *DHT*, ya que lo renombramos anteriormente. Además, este se localiza en el directorio */sketchbook/libraries*.

Para compilar el programa, subirlo a la placa y visualizar su ejecución por pantalla ejecutamos, dentro de la carpeta */sketchbook/LED_DHT11*, el comando *make upload monitor clean*. El resultado de la ejecución se muestra en la Figura 24.

```
ON
Humedad: 66.00 %
Temperatura: 23.00 *C
OFF
ON
Humedad: 66.00 %
Temperatura: 23.00 *C
OFF
ON
Humedad: 66.00 %
Temperatura: 23.00 *C
OFF
```

Figura 24: Visualización de la ejecución del programa

Por último, vamos a conectar a la pasarela una estación de contaminación. Esta estación emplea el sensor BME280 [17], por lo que nos tenemos que descargar la librería correspondiente a dicho sensor. En este caso, vamos a emplear el repositorio de GitHub. Para ello, nos situamos dentro del directorio */sketchbook/libraries* y clonamos el repositorio de GitHub que contiene la librería:

```
git clone https://github.com/adafruit/Adafruit_BME280_Library.git
```

Una vez clonado el directorio lo renombramos. Para ello ejecutamos el comando `mv Adafruit_BME280_Library BME280`. Además de esta librería, es necesario descargarse la librería `Adafruit_Sensor`, por lo que clonamos el repositorio de GitHub que contiene dicha librería:

```
git clone https://github.com/adafruit/Adafruit_Sensor.git
```

Una vez clonado el repositorio lo renombramos. Para ello ejecutamos el comando `mv Adafruit_Sensor AdafruitSensor`. Además de estas dos librerías, son necesarias otras dos que vienen por defecto con el paquete `arduino-mk`. Estas dos librerías son `SPI` y `Wire`. Sin embargo, la librería `SPI` da errores al compilar el código, de modo que también nos la descargamos. Para ello, clonamos el correspondiente repositorio de GitHub:

```
git clone https://github.com/PaulStoffregen/SPI.git
```

En este caso no es necesario renombrar el directorio. Una vez descargadas todas las librerías necesarias, creamos el directorio `/sketchbook/Estacion_Contaminacion`, en el cual vamos a guardar el código del programa desarrollado y el archivo Makefile necesario para compilar el mismo. El código del programa se encuentra en un archivo denominado `Medidor_Contaminacion.ino`. En este caso, su contenido es demasiado extenso como para mostrarlo.

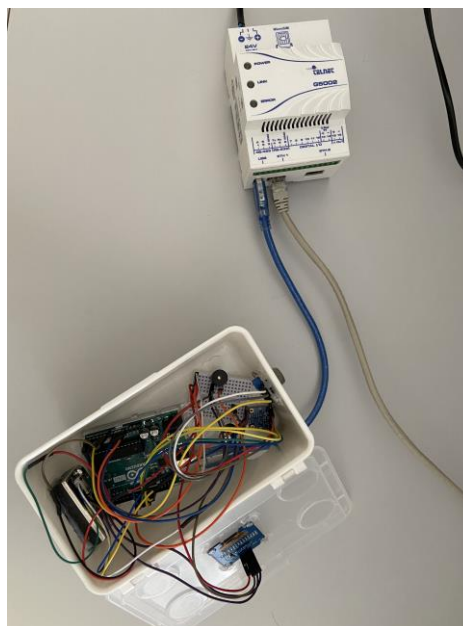


Figura 25: Estación de contaminación conectada a la pasarela a través de la placa de Arduino

Para poder compilar el programa y subirlo a la placa de Arduino creamos un archivo denominado *Makefile* cuyo contenido es el que se puede ver en la Figura 26.

```
ARDUINO_DIR = /usr/share/arduino

BOARD_TAG = uno
ARDUINO_PORT = /dev/ttyACM*

USER_LIB_PATH = /home/telnet/sketchbook/libraries
ARDUINO_LIBS = BME280 AdafruitSensor SPI Wire

include /usr/share/arduino/Arduino.mk
```

Figura 26: Contenido del archivo Makefile

En este caso, tenemos que incluir en el archivo *Makefile* cuatro librerías. Para ello, es necesario indicar el nombre de los directorios en los cuales se encuentran los ficheros correspondientes a cada una de ellas. Tres de estas librerías se encuentran en el directorio */sketchbook/libraries*, ya que nos las hemos descargado. La última librería, denominada *Wire*, se localiza en el directorio de Arduino definido al comienzo del archivo.

Para compilar el programa, subirlo a la placa y visualizar su ejecución por pantalla ejecutamos, dentro de la carpeta */sketchbook/Estacion_Contaminacion*, el comando *make upload monitor clean*. El resultado de la ejecución se muestra en la Figura 27.

```
P:933.11
H:46.60 %
T:26.40 C

Amoniaco:3532.88
NoX:5577.53
CO2:5352.86

CALIDAD DE AIRE BAJA
D.Polvo:77.5 ugxm3
Conc:121030.8 ppm

AIRE BUENO
```

Figura 27: Visualización de la ejecución del programa

Por último, cabe indicar que la pasarela IoT trae de fábrica un módulo *Xbee* que permite establecer comunicaciones inalámbricas empleando el estándar *GSM/GPRS*

mediante el uso de una tarjeta SIM. No obstante, para el escenario presente en el laboratorio, sería más adecuado emplear un módulo *ZigBee*, de manera que los distintos sensores conectados a través de una placa de Arduino (u otras de similares características) se pueden comunicar con la pasarela de manera inalámbrica, lo que permite desplegar estos sensores en distintas zonas.

3.5 Activación del puerto 10G de la OLT

Como se ha comentado anteriormente, el modelo de OLT empleado en la red de acceso SmartOLT 350 [12] consta de 4 puertos PON. En función de qué puerto PON tengamos activo, para que las ONTs registradas tengan conexión a Internet se debe conectar el correspondiente puerto *Gigabit Ethernet* al ordenador central del laboratorio, ya que este constituye la puerta de enlace del laboratorio con la red externa de la facultad.

No obstante, la OLT consta también de un puerto 10G. Para activar dicho puerto, debemos acceder a la OLT por CLI y cambiar el modo de transporte, para lo cual se emplea la interfaz que conecta el ordenador central del laboratorio con el puerto de gestión. El acceso se realiza a través de una conexión *telnet* al puerto 4555 de la OLT. Para ello, abrimos una terminal en el ordenador central y ejecutamos el comando *telnet 172.26.128.38 4555*. Una vez se establezca la conexión, debemos introducir el usuario y la contraseña (*root/root*). De esta manera, accedemos al CLI, tal y como se observa en la Figura 28.

```
tfglab7@tfglab7-B150M-D3H ~ $ telnet 172.26.128.38 4555
Trying 172.26.128.38...
Connected to 172.26.128.38.
Escape character is '^]'.
OLT Management CLI v1.0.154

Username: root
Password:

SmartOLT> █
```

Figura 28: Conexión a la OLT mediante CLI

Una vez dentro, accedemos al menú “*system*”. Podemos visualizar el modo de transporte actual introduciendo el comando *showTransportConfiguration*. El modo 4x1GbE se corresponde con el valor 2, mientras que el modo 10G se corresponde con el valor 1. En nuestro caso, para activar el modo de transporte 10G ejecutamos el comando *setTransportConfiguration 1*. Por último, actualizamos el modo de transporte mediante el

comando *updateTransportConfiguration*. De esta manera, si ahora volvemos a ejecutar el comando *showTransportConfiguration*, vemos que el modo de transporte es 10G. Todos estos pasos están recogidos en la Figura 29.

```
SmartOLT> system
SmartOLT/system> showTransportConfig
Current Configuration:
  IF_CONFIGURATION = 2
  SFPPLUS_TYPE = 0

SmartOLT/system> setTransportConfig 1
SmartOLT/system> update
updateNetwork updateTransportConfig

SmartOLT/system> updateTransportConfig

SmartOLT/system> showTransportConfig
Current Configuration:
  IF_CONFIGURATION = 1
  SFPPLUS_TYPE = 0
```

Figura 29: Ejecución de los comandos necesarios para conmutar al puerto 10G

Mediante la activación de este puerto, todo el tráfico de los 4 puertos PON de la OLT es agregado en un único flujo 10G, es decir, se sustituyen los cuatro puertos *Gigabit Ethernet* empleados para dotar a la red de acceso de conexión a Internet por un único puerto 10G. Esta interfaz es capaz de proporcionar el 100% del ancho de banda de GPON (4 x 2.5/1.25Gbps), es decir, permite una tasa máxima de 10 Gbps en el canal descendente y de 5 Gbps en el canal ascendente.



Figura 30: Activación del puert 10G de la OLT

Debido a la activación de este puerto 10G, el ordenador central del laboratorio, el cual se encuentra situado justo antes de la OLT dentro de la topología de la red GPON, va a ser reemplazado por un servidor de la escuela (fuera del laboratorio), denominado *chron2*, de modo que dicho servidor se conecta a la fibra que va conectada directamente al puerto 10G de la OLT situada en el laboratorio. Por tanto, para configurar de manera correcta este nuevo escenario SDN-GPON, es necesario configurar en el servidor los

servicios que implementaba el ordenador central: un *router*, un servidor DHCP, un OVS y un controlador SDN, en nuestro caso ONOS. Para llevar a cabo esta implementación en el *chron2*, se empleará Docker como herramienta de virtualización, a través de la creación de distintas imágenes y contenedores (consulte el Capítulo 2, Apartado 4 para obtener información sobre imágenes docker, contenedores y virtualización).

Por lo tanto, se crearán cuatro contenedores docker en el servidor, tal y como se muestra en la Figura 31. El primer contenedor se empleará para configurar el *router*, el segundo contenedor integrará el servidor DHCP, en el tercer contenedor se instalará el OVS y, por último, el cuarto contenedor implementará el controlador SDN.

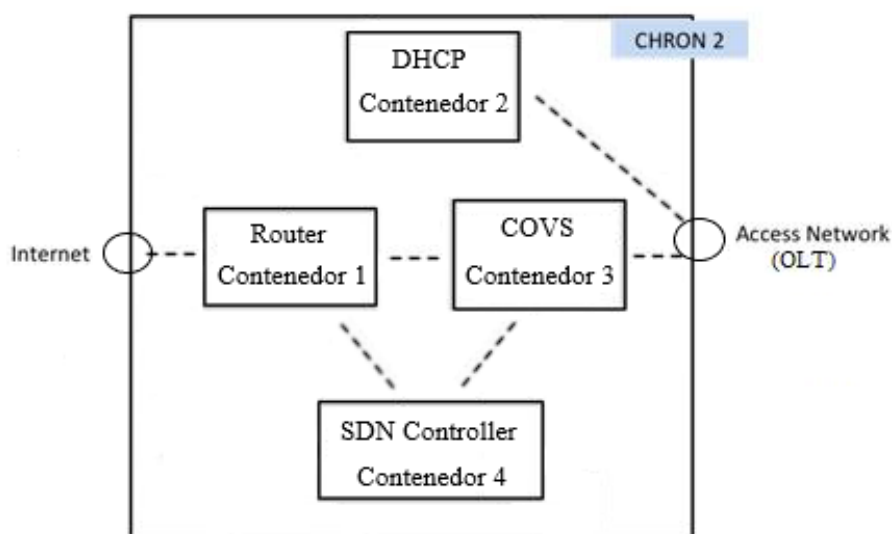


Figura 31: Topología de red con cuatro contenedores docker

En otro TFG anterior se realizó un despliegue similar en *chron2* al que se va a llevar a cabo en este proyecto. No obstante, en dicho TFG no se separaron cada uno de los servicios en distintos contenedores. Por ello, se va a volver a realizar toda la configuración empleando un contenedor distinto para cada herramienta, con el objetivo de mejorar el rendimiento y actualizar algunas de las imágenes docker empleadas.

3.6 Conclusiones

En este capítulo de la memoria se ha descrito, en primer lugar, el escenario SDN-GPON inicial que estaba implementado en la red del laboratorio. También se ha llevado a cabo la activación de un nuevo puerto PON de la OLT, y se ha podido comprobar que su funcionamiento es correcto. Respecto a la pasarela IoT, se han descrito los pasos

necesarios para su activación y configuración, haciendo que esta se pueda integrar fácilmente en la red GPON conectándola a una de las ONTs de nivel 3. Además, se han realizado varias pruebas empleando distintos sensores conectados a través de una placa de Arduino. Por último, se ha llevado a cabo la activación del puerto 10G de la OLT, lo cual supone una importante actualización en el escenario SDN-GPON desplegado en el laboratorio hacia el uso de tecnologías de virtualización ampliamente empleadas en la actualidad como es el caso de Docker, herramienta que se verá en el siguiente capítulo de este trabajo.

4

Implementación del nuevo escenario SDN-GPON

4.1 Introducción

En este capítulo de la memoria se va a realizar una descripción de la implementación del nuevo escenario SDN tras la activación del puerto 10G de la OLT. Como se ha mencionado al final del anterior capítulo, el ordenador central del laboratorio va a ser sustituido por un servidor, denominado *chron2*, de modo que todos los servicios que implementaba dicho ordenador se van a trasladar a este servidor. Para realizar esta tarea, se va a emplear Docker como herramienta de virtualización, por lo que es necesario comprender su funcionamiento.

Con tal objetivo se realizará, en primer lugar, un estudio sobre Docker y se presentarán los comandos más habituales. A continuación, se describirán los pasos a seguir para desplegar todas las herramientas necesarias en diferentes contenedores docker: un servidor DHCP, un *router*, un OVS y un controlador SDN, en nuestro caso ONOS. Posteriormente, se explicarán las actualizaciones llevadas a cabo en el lado de los usuarios, en concreto, el despliegue de nuevos OVSs empleando contenedores docker y la configuración de una subred independiente de la red de acceso. Una vez realizadas estas actualizaciones, se llevarán a cabo una serie de pruebas para analizar el rendimiento del nuevo escenario SDN-GPON mediante la creación de distintos *flows* y *meters* en el controlador ONOS.

4.2 Introducción a Docker

Docker es una herramienta que no viene instalada de serie en ningún sistema operativo. No obstante, la página oficial de Docker [18] proporciona toda la documentación necesaria para poder llevar a cabo el proceso de instalación. En nuestro

caso, Docker ya está instalado en el *chron2*, pero no está instalado en los ordenadores del laboratorio en lo que se encuentran las máquinas virtuales con los OVSS correspondientes a las ONTs de nivel 2. Por ello, posteriormente se documentarán los pasos necesarios para instalar Docker en sistemas Linux, concretamente en Linux Mint, ya que esta es la distribución con la que cuentan los ordenadores del laboratorio.

Los conceptos de imágenes y contenedores docker ya se definieron en el Capítulo 2 de este trabajo. Por ello, en este apartado nos centraremos en el uso de imágenes predefinidas de Docker Hub para crear distintos contenedores con el objetivo de ver los principales comandos y sus funcionalidades.

4.2.1 Obtención de imágenes y creación de contenedores docker

A lo largo de este proyecto se emplearán una serie de comandos básicos tanto para la manipulación de imágenes como de contenedores. Estos comandos son los siguientes [19]:

- *docker pull*: descarga una imagen de Docker Hub
- *docker images*: muestra las imágenes docker descargadas
- *docker rmi*: elimina una imagen
- *docker run*: ejecuta una imagen para crear un contenedor
- *docker ps*: muestra los contenedores que se están ejecutando
- *docker ps -a*: muestra todos los contenedores, tanto los que se están ejecutando como lo que están parados (stop)
- *docker stop*: detiene un contenedor que se está ejecutando
- *docker start*: inicia un contenedor que está en stop
- *docker attach*: accede al interior de un contenedor
- *docker exec*: ejecuta un comando en un contenedor
- *docker rm*: elimina un contenedor que está en stop

Además de estos comandos básicos existen otros comandos más avanzados que se comentarán según vayan apareciendo a lo largo del trabajo.

Al terminar de instalar Docker, si ejecutamos el comando *docker images* no aparecerá ninguna imagen. Por tanto, debemos descargarnos la imagen que necesitamos de Docker Hub. En este caso, vamos a descargarnos una imagen base de Ubuntu en su versión más reciente, para lo cual ejecutamos el comando *docker pull ubuntu:latest*. Si ejecutamos ahora el comando *docker images* nos aparecerá la imagen de Ubuntu que acabamos de descargar.

Antes de crear nuestro primer contenedor vamos a renombrar la imagen de Ubuntu. Para ello, empleamos el comando *docker tag <image-name:tag> <new-image-name:tag>*, por ejemplo, *docker tag ubuntu:latest pruebas:v1*. El *tag* solo es la versión de la imagen. Cuando esta se actualice, le daremos a la nueva imagen el mismo nombre con una etiqueta diferente, por ejemplo, *v2*.

Después de renombrar la imagen, ésta se puede eliminar con el comando *docker rmi <image-name:tag>*. No obstante, no vamos a eliminar la imagen base de Ubuntu dado que esta se empleará más adelante, pero este comando es útil para eliminar versiones anteriores de las imágenes.

Para crear un contenedor a partir de la imagen ejecutamos el comando *docker run -it <image-name: tag> bash*, en nuestro caso, *docker run -it pruebas:v1 bash*. Mediante este comando ejecutamos el programa *bash* contenido en la imagen de Ubuntu y además hacemos que esta ejecución sea interactiva con la opción *-it*, es decir, para que no tan solo ejecute *bash* y termine, si no que se quede mostrando por terminal la ejecución de este programa. En resumen, abrimos una terminal para poder interactuar con el contenedor.

También podemos darle al contenedor un nombre específico durante su creación agregando al comando anterior la opción *--name <container-name>*. De esta manera, el comando correspondiente sería de la siguiente forma: *docker run -it --name primer_contenedor pruebas:v1 bash*.

Una vez creado el primer contenedor, es necesario instalar algunos paquetes básicos de los que carece la imagen base de Ubuntu en función de lo que se quiera hacer. Estos paquetes se instalan del mismo modo que en cualquier ordenador/servidor Linux. Como ejemplo, vamos a instalar una serie de paquetes, para lo cual ejecutamos los siguientes comandos:

```
apt-get update
```

```
apt-get install -y net-tools nano iproute2 iputils-ping ifupdown iperf iperf3
```

De este modo, hemos creado nuestro primer contenedor, en el cual hemos ejecutado el programa *bash* de manera interactiva para poder interactuar con el contenedor. Además, hemos actualizado el repositorio y hemos instalado una serie de paquetes básicos como ejemplo.

4.2.2 Redes en Docker

Tras instalar Docker se crea, por defecto, una interfaz de red virtual denominada *docker0*, tal y como se puede observar en la Figura 32.

```
csannun@chron2:~$ /sbin/ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:d2ff:fe20:1542 prefixlen 64 scopeid 0x20<link>
    ether 02:42:d2:20:15:42 txqueuelen 0 (Ethernet)
    RX packets 14487218 bytes 10785703227 (10.0 GiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17870046 bytes 25817432902 (24.0 GiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 32: Interfaz de red *docker0*

Esta nueva interfaz de red es un puente al que Docker conecta de forma predeterminada todos los contenedores que se creen. Este puente se comporta como un *switch* tradicional, el cual realiza una traducción de la dirección NAT entre la red virtual *docker0* (por defecto 172.17.0.0/16) y la red a la que está conectado el *host*. La Figura 33 muestra cómo funciona el modelo de red cuando los contenedores están conectados entre sí a través de la red *docker0*.

De esta manera, los contenedores tienen conexión a Internet pero no están expuestos, ya que la dirección IP por la que se enruta el tráfico externo es la correspondiente a la interfaz de red del *host*. Esto vendría a ser algo similar al adaptador de red NAT de las máquinas virtuales. Si se desean exponer servicios, se le debe indicar al contenedor qué puertos se desean publicar/exponer, de manera que cuando los paquetes destinados a esos puertos lleguen al *host*, este los redireccione directamente al contenedor. Más adelante, cuando se ejecute el controlador ONOS dentro de un contenedor, se mostrará como se realiza esta redirección.

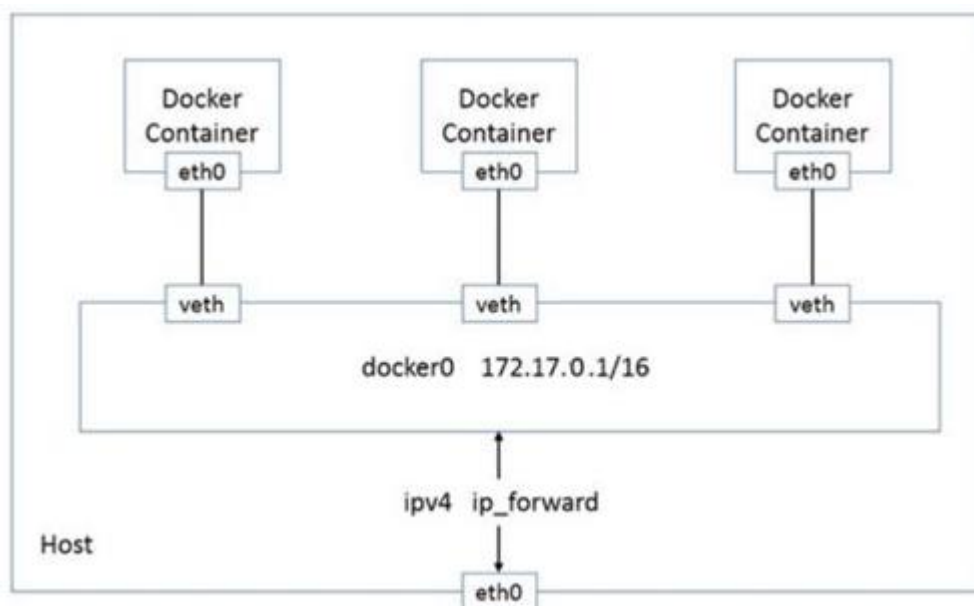


Figura 33: Modelo de red de contenedores docker

Cuando se crea un contenedor sin especificar ninguna información de red concreta, automáticamente este recibe una dirección IP de la red *docker0*. Esto es lo que se conoce como *bridge networking*. Los contenedores que se ejecutan en modo *bridge* adquieren una dirección IP de la red *docker0* y son solamente accesibles desde ese mismo *host*, ya sea desde otro contenedor definido en esta misma red *docker0* o desde el propio *host*. Por tanto, estos contenedores están aislados, en nuestro caso, del resto de la red de acceso GPON del laboratorio. No obstante, en este proyecto el objetivo es desplegar en este servidor *chron2* todas las herramientas necesarias para el nuevo escenario SDN, de manera que el controlador ONOS supervise la gestión de los mensajes *OpenFlow* intercambiados entre los distintos OVSs para la correcta gestión de los servicios proporcionados a los usuarios finales. Como consecuencia, tanto el controlador ONOS como el OVS que se van a desplegar en el servidor requieren de alguna otra configuración de red distinta al puente *docker0*, de modo que sean visibles desde la red de acceso del laboratorio.

Docker proporciona una solución ante esta situación, permitiendo que los contenedores empleen directamente las interfaces de red del *host*, eliminando de esta manera el aislamiento que supone el empleo del puente *docker0*. Esto es lo que se denomina *host networking*. De esta manera, si empleamos el modo *host* para crear un contenedor, este no adquiere una dirección IP propia, puesto que tiene acceso a todas las interfaces de red del *host*. La Figura 34 muestra como funciona este modo.

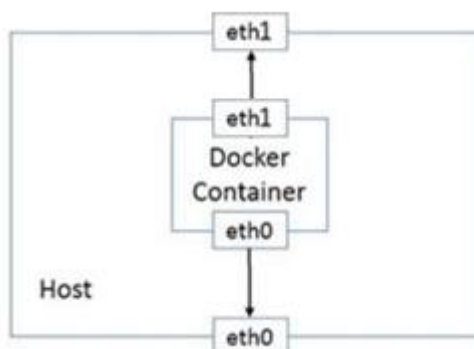


Figura 34: Modelo de red de un contenedor en modo *host*

Cabe destacar que ejecutar un contenedor en modo *host* conlleva una serie de riesgos si el usuario no tiene cuidado, puesto que los cambios realizados dentro del contenedor con respecto a la red pueden afectar al *host*. Para crear un contenedor empleando el modo *host* tenemos que agregar la opción `--network host` al comando `docker run`.

Para el despliegue del controlador ONOS en un contenedor vamos a emplear una interfaz de red virtual distinta de `docker0`, ya que el puente `docker0` tiene ciertas limitaciones. Para crear este nuevo puente ejecutamos el siguiente comando:

```
docker network create --driver bridge --subnet 10.1.0.0/16 --gateway 10.1.0.1 --ip-range 10.1.4.0/24 bridge_test
```

Mediante la ejecución de este comando creamos nuestra propia interfaz de red virtual denominada `bridge_test`, la cual se puede observar en la Figura 35. Con el resto de las opciones definimos la subred, la puerta de enlace y el rango de direcciones IPs disponibles para asignar a los distintos contenedores que se creen en esta nueva red. El modelo de red empleando este nuevo puente es similar al de la red `docker0`, el cual se puede observar en la Figura 33.

```
csannun@chron2:~$ /sbin/ifconfig
br-f73355c08a82: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 10.1.0.1 netmask 255.255.0.0 broadcast 10.1.255.255
    inet6 fe80::42:29ff:fef0:274f prefixlen 64 scopeid 0x20<link>
    ether 02:42:29:f0:27:4f txqueuelen 0 (Ethernet)
    RX packets 20408043 bytes 26136896883 (24.3 GiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 27514778 bytes 27516294903 (25.6 GiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 35: Nueva interfaz de red creada

En el caso de ONOS no empleamos el modo *host* para la creación del contenedor dado que la interfaz de red que conecta el *chron2* con la red de acceso va a ser empleada por el OVS. Si ejecutásemos ambos contenedores en modo *host*, los dos emplearían dicha interfaz para comunicarse con el resto de la red de acceso, lo cual supone un problema. Dado que el controlador ONOS se va a ejecutar en modo *bridge*, este contenedor estará aislado. Por ello, se modificarán las entradas *iptables* del servidor para dotar a este contenedor de conexión con el resto de la red de acceso.

Para crear un contenedor usando esta otra interfaz de red simplemente tenemos que agregar la opción `--network bridge_test` al comando `docker run`. Al igual que el puente `docker0`, este otro puente tiene conexión con todas las interfaces del *host*, y por lo tanto, también tiene conexión a Internet.

En resumen, se van a crear cuatro contenedores docker en el servidor. Los contenedores que van a implementar el servidor DHCP, el *router* y el OVS se van a crear en modo *host*, de modo que puedan tener acceso a las interfaces de red del *chron2*. En cambio, para el controlador ONOS se va a crear un contenedor en modo *bridge* empleando para ello el nuevo puente creado denominado `bridge_test`. Esta configuración está representada en la Figura 31.

4.3 Implementación de un servidor DHCP y un *router* en contenedores docker

El primer paso para la implementación del nuevo escenario SDN es configurar la interfaz de red que conecta el servidor *chron2* con la OLT a través del módulo 10G. Para ello, dado que en la red de acceso se emplea la VLAN 833 para la gestión de los servicios de Internet de las distintas ONTs, es necesario crear y configurar esta misma VLAN en la interfaz. La creación de la VLAN se realizará desde dentro del contenedor que implementará el servidor DHCP, dado que no tenemos permisos de super-usuario en el *chron2*. Esta es una de las ventajas que presenta el uso de Docker, tal y como se explicó en el apartado anterior.

Para la instalación del servidor DHCP se empleará el paquete `isc-dhcp-server`, configurándolo para que resuelva peticiones DHCP en la interfaz que conecta el *chron2* con la OLT, y por tanto, con el resto de la red de acceso, de manera que cada uno de los

dispositivos de la red adquiera una dirección IP. Para la configuración del *router*, dado que el servidor emplea Debian como sistema operativo, emplearemos la utilidad del reenvío de paquetes IP del *kernel* de Linux. De esta manera, mediante la configuración de *iptables*, se dotará de acceso a Internet a la red GPON.

4.3.1 Despliegue y prueba de funcionamiento del servidor DHCP

Para poder llevar a cabo la configuración del servidor DHCP el paso preliminar es crear un contenedor. En este caso, dado que haciendo pruebas hemos visto que el servidor DHCP no funciona correctamente empleando la imagen base de Debian, hemos utilizado la imagen de Ubuntu.

El primer paso es descargarse la imagen oficial de Ubuntu de Docker Hub. Nosotros emplearemos su versión más reciente, para lo cual ejecutamos el comando *docker pull ubuntu:latest*, tal y como vimos en el Apartado 2 de este capítulo sobre introducción a Docker.

Una vez descargada la imagen la renombramos para que se pueda identificar fácilmente. Para ello, empleamos el comando *docker tag <image-name:tag> <new-image-name:tag>*. En este caso será *docker tag ubuntu:latest dhcp_server:v1*. El nombre de la nueva imagen es *dhcp_server* y la versión *v1*. Después de haber renombrado la imagen, si se desea, esta se puede eliminar empleando el comando *docker rmi <image-name:tag>*, concretamente *docker rmi ubuntu:latest*.

Para crear el contenedor ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name servidor_dhcp --privileged dhcp_server:v1 bash
```

Mediante este comando creamos, a partir de la imagen que hemos renombrado, un contenedor en el cual vamos a ejecutar el programa *bash* contenido en la imagen de Ubuntu, y además hacemos que esta ejecución sea interactiva con la opción *-it*. Con la opción *--network host* hacemos que el contenedor se ejecute en modo *host*, de manera que este puede hacer uso de las interfaces de red del anfitrión. Con la opción *--name servidor_dhcp* damos un nombre al contenedor. Finalmente, con la opción *--privileged* otorgamos al contenedor permisos de super-usuario.

A partir de ahora, todos los comandos se ejecutarán dentro del contenedor. La imagen oficial de Ubuntu descargada de Docker Hub no contiene muchos de los paquetes necesarios para la configuración del *router* y del servidor DHCP, por lo que se instalarán manualmente. Los paquetes necesarios son los siguientes:

- net-tools
- nano
- isc-dhcp-server
- iproute2
- iputils-ping
- ifupdown
- vlan
- iperf
- iperf3

Para instalar estos paquetes en el contenedor primero actualizamos el repositorio y posteriormente procedemos a su instalación, para lo cual ejecutamos los siguientes comandos:

```
apt-get update
```

```
apt-get install -y net-tools nano isc-dhcp-server iproute2 iputils-ping ifupdown  
vlan iperf iperf3
```

Una vez instalados los paquetes necesarios podemos configurar la interfaz de red que conecta *chron2* con la OLT. Lo primero es crear la VLAN 833 asociada a esta interfaz, la *enp6s0f1*. Para ello, ejecutamos el siguiente comando dentro del contenedor:

```
vconfig add enp6s0f1 833
```

Si ejecutamos ahora el comando *ifconfig* veremos que se ha creado la VLAN 833 asociada a la interfaz (*enp6s0f1.833*). Finalmente, configuramos la VLAN, para lo cual es necesario editar el archivo */etc/network/interfaces* del servidor. Para escapar del contenedor, es decir, para salir de él sin detenerlo, ejecutamos la secuencia

CTRL-p CTRL-q. Si por el contrario queremos salir del contenedor y detenerlo, ejecutamos el comando *exit*.

Para editar el archivo */etc/network/interfaces* es necesario tener privilegios de super-usuario, de modo que nosotros no realizaremos este proceso. No obstante, los pasos a seguir son los siguientes:

- Ejecutamos el comando *sudo nano /etc/network/interfaces*
- Una vez dentro del archivo agregamos las siguientes líneas:

```
allow-hotplug enp6s0f1.833
iface enp6s0f1.833 inet static
address 192.168.100.1
netmask 255.255.255.0
network 192.168.100.0
broadcast 192.168.100.255
```

- Guardamos los cambios efectuados en el archivo mediante la secuencia CTRL-o y salimos del archivo mediante la secuencia CTRL-x.

De este modo configuramos la VLAN de manera estática, dándole una dirección IP fija (192.168.100.1). Al hacerlo de esta manera, es decir, editando el archivo */etc/network/interfaces*, si esta VLAN se elimina y se vuelve a crear se configurará de manera automática, lo cual resultará muy útil más adelante.

Una vez configurada la interfaz de red que conecta el *chron2* con la OLT procedemos con la configuración del servidor DHCP. Para ello, accedemos de nuevo al contenedor, si este se sigue ejecutando en segundo plano, mediante el comando *docker attach <container-ID>*. Si hemos otorgado un nombre al contenedor, como es nuestro caso, también es posible acceder a este mediante el comando *docker attach <container-name>*. Si se detuvo el contenedor, antes de acceder al mismo es necesario iniciarlo, para lo cual ejecutamos el comando *docker start <container-ID>* o *docker start <container-name>*.

Una vez de nuevo dentro del contenedor, para el despliegue del servidor DHCP hay que configurar una serie de archivos para que el DHCP funcione correctamente. El primer archivo se encuentra en el directorio */etc/dhcp/dhcpd.conf*, en el cual hay que

agregar una serie de reglas para la subred 192.168.100.0/24 que hemos definido anteriormente. En concreto, hay que definir:

- Un rango de direcciones IP
- Un servidor DNS
- Un nombre de dominio
- Un *router*
- Una dirección *broadcast*
- Un tiempo de arrendamiento predeterminado
- Un tiempo máximo de concesión

La configuración exacta que se ha realizado es la siguiente:

```
subnet 192.168.100.0 netmask 255.255.255.0 {  
    range 192.168.100.130 192.168.100.250;  
    option domain-name-servers 157.88.129.90;  
    option domain-name "RouterTFGLab7-833";  
    option routers 192.168.100.1;  
    option broadcast-address 192.168.100.255;  
    default-lease-time 600;  
    max-lease-time 7200;  
}
```

El segundo archivo se encuentra en el directorio */etc/default/isc-dhcp-server*. En este archivo hay que especificar la interfaz por la cual el servidor va a aceptar las peticiones DHCP. En nuestro caso, la interfaz correspondiente es la VLAN 833 creada anteriormente. Por tanto, agregamos la siguiente regla al archivo:

```
INTERFACESv4="enp6s0f1.833"
```

Con esto ya tenemos configurado nuestro servidor DHCP. Para inicializarlo, basta con ejecutar el comando *service isc-dhcp-server start*.

Para comprobar que el servidor DHCP asigna correctamente direcciones IP a los dispositivos de la red podemos hacer una prueba de conexión con una de las ONTs de nivel 3. Basta con hacer un *ping* a la dirección IP asignada por el DHCP a la ONT, la cual se puede obtener accediendo a la página web del *router*, dentro de la sección *Device Info* → *WAN* (Figura 36). Es importante que esta ONT tenga asignado un perfil de Internet en TGMS, si no, no recibirá ninguna dirección IP por parte del DHCP.

WAN Info													
Interface	Description	Type	VlanMuxId	IPv6	Igmp Pxy	Igmp Enbl	MLD Pxy	MLD Src Enbl	NAT	Firewall	Status	IPv4 Address	IPv6 Address
veip0.1	ipoe_veip0.833	IPoE	833	Disabled	Disabled	Disabled	Disabled	Disabled	Enabled	Disabled	Connected	192.168.100.140	

Figura 36: Dirección IP de la ONT de nivel 3 en la red de acceso

```
csannun@chron2:~$ ping 192.168.100.140
PING 192.168.100.140 (192.168.100.140) 56(84) bytes of data.
64 bytes from 192.168.100.140: icmp_seq=1 ttl=64 time=0.651 ms
64 bytes from 192.168.100.140: icmp_seq=2 ttl=64 time=0.638 ms
64 bytes from 192.168.100.140: icmp_seq=3 ttl=64 time=0.529 ms
64 bytes from 192.168.100.140: icmp_seq=4 ttl=64 time=0.530 ms
```

Figura 37: Prueba de conexión entre el *chron2* y la ONT de nivel 3

Una vez comprobado que el servidor DHCP funciona correctamente, vamos a crear un *script* denominado *configuracionVLAN.sh* dentro de la carpeta *root* del contenedor que nos va a permitir reconfigurar la interfaz *enp6s0f1.833*, ya que, como se verá posteriormente, cuando se inicializa el OVS del *chron2* la dirección IP de esta interfaz es asignada al puente *br0* creado por el OVS. El contenido de este *script* es el siguiente:

```
#!/bin/bash
ifconfig br0 0
ifconfig br0 down
vconfig rem enp6s0f1.833
vconfig add enp6s0f1 833
```

Mediante la ejecución de este *script* reiniciamos el puente *br0*, eliminamos la VLAN 833 asociada a la interfaz *enp6s0f1* y la volvemos a crear. Dado que esta VLAN se definió mediante el archivo */etc/network/interfaces*, una vez creada se configurará de manera automática. El comando necesario para ejecutar este *script* es *sh configuracionVLAN.sh*.

Por último, guardamos todos los cambios efectuados en este contenedor en una nueva imagen. Para ello, empleamos el comando `docker commit <container-ID> <new-image-name:tag>` o `docker commit <container-name> <new-image-name:tag>`. En nuestro caso, el nombre de la nueva imagen será `dhcp:v_final`. Para crear un contenedor a partir de esta nueva imagen, ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name
servidor_dhcp_vfinal --privileged dhcp:v_final bash
```

4.3.2 Despliegue y prueba de funcionamiento del router

Para poder llevar a cabo la configuración del *router*, dado que no tenemos permisos de super-usuario en el *chron2*, vamos a crear un contenedor empleando la imagen base de Debian, ya que el sistema operativo del servidor es Debian. De esta manera, podremos modificar las entradas *iptables* del *chron2* desde dentro del contenedor.

Al igual que en el caso del servidor DHCP, el primer paso es descargarse la imagen oficial de Debian de Docker Hub. Nosotros emplearemos su versión más reciente, para lo cual ejecutamos el comando `docker pull debian:latest`. Una vez descargada la imagen la renombramos para que se pueda identificar fácilmente. Para ello, empleamos el comando `docker tag <image-name:tag> <new-image-name:tag>`. En este caso será `docker tag debian:latest router:v1`. Después de que haber renombrado la imagen, si se desea, esta se puede eliminar empleando el comando `docker rmi <image-name:tag>`, concretamente `docker rmi debian:latest`.

Para crear el contenedor ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name router_iptables --
privileged router:v1 bash
```

Mediante este comando creamos, a partir de la imagen que hemos renombrado, un contenedor en el cual vamos a ejecutar el programa *bash* contenido en la imagen de Debian, y además hacemos que esta ejecución sea interactiva con la opción `-it`. Con la opción `--network host` hacemos que el contenedor se ejecute en modo *host*, de manera que este puede hacer uso de las interfaces de red del anfitrión. Con la opción `--name router_iptables` damos un nombre al contenedor. Finalmente, con la opción `--privileged`

otorgamos al contenedor permisos de super-usuario, de manera que podamos modificar las entradas *iptables* del *host* para la configuración del *router*.

No obstante, antes de llevar a cabo la configuración del *router* es necesario activar la utilidad del reenvío de paquetes IP del *kernel* de Linux, para lo cual escribimos un *1* en el archivo *ip_forward* que se encuentra en la ruta */proc/sys/net/ipv4/ip_forward*. Esto hace que el servidor se comporte como un enrutador, es decir, que pueda reenviar paquetes de una subred (la subred que conecta *chron2* con la red externa de la facultad) a otra subred (la subred que conecta *chron2* con la red de acceso del laboratorio), tal y como se puede observar en la Figura 38.

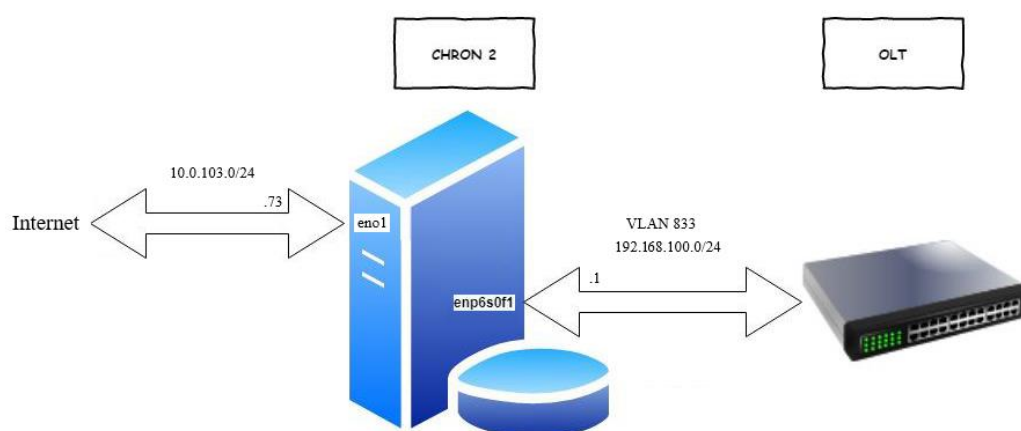


Figura 38: Topología de la configuración del router en el *chron2*

Para que el *router* actúe según lo previsto, vamos a modificar las entradas *iptables*. Para ello, previamente es necesario instalar el paquete correspondiente, ya que este no viene instalado por defecto en la imagen de Debian. Para instalar este paquete, primero actualizamos el repositorio y posteriormente procedemos con su instalación, para lo cual ejecutamos los siguientes comandos:

```
apt-get update
```

```
apt-get install -y iptables
```

Aunque es cierto que *iptables* se suele utilizar más como una utilidad para establecer reglas de *firewall*, también nos permite establecer reglas estáticas sobre cómo queremos que nuestro *kernel* reenvíe los paquetes. Estas reglas se especifican a través de la línea de comandos, utilizando los parámetros necesarios que veremos a continuación.

Generalmente se limitan a especificar de qué subred a qué subred se quieren reenviar los paquetes y como van a ser tratados. En concreto, vamos a aplicar una única regla:

```
iptables -t nat -I POSTROUTING -s 192.168.100.0/24 -o eno1 -j MASQUERADE
```

```
iptables -I FORWARD -i eno1 -j ACCEPT
```

```
iptables -I FORWARD -o eno1 -j ACCEPT
```

Las diferentes opciones de esta regla se describen a continuación:

- La opción `-t` especifica a qué tabla debe hacer referencia este comando, en este caso, a la tabla NAT. Esta tabla se consulta cuando un paquete entrante crea una nueva conexión, y tiene dos posibilidades: PREROUTING (para actuar sobre los paquetes según lleguen al *kernel*) y OUTPUT (para actuar sobre los paquetes que se generan localmente antes de ser enrutados).
- La opción `-I` especifica qué regla queremos agregar en la tabla previamente especificada. Esta regla se agrega al principio de la tabla, de manera que sea la primera que se aplica si es que existen otras reglas que tienen las mismas condiciones.
- La opción `-s` especifica el origen de los paquetes a los que queremos aplicar la regla. En este caso, agregaremos la subred 192.168.100.0/24, ya que esta es la subred que se ha configurado en la interfaz que conecta *chron2* con la red de acceso.
- La opción `-o` especifica la interfaz por la cual queremos sacar los paquetes que cumplan las reglas anteriores. En nuestro caso, la interfaz *eno1*, la cual conecta con la red externa de la facultad y proporciona conexión a Internet.
- La opción `-j` indica qué hacer con los paquetes que cumplan con todo lo especificado anteriormente.

Con esto, ya tenemos configurado nuestro *router* para dotar a la red de acceso de conexión a Internet.

Visto que el servidor DHCP funciona correctamente, puesto que tenemos conexión con la ONT de nivel 3, podemos realizar también una prueba para ver si el *router* funciona correctamente y, por lo tanto, tenemos conexión a Internet. Para ello,

basta con conectarse a alguna de las ONTs de nivel 3 del laboratorio con un perfil asignado en TGMS, y realizar un *ping* a *www.google.es*, por ejemplo. Para esta prueba, emplearemos uno de los Lenovos conectado a la misma ONT que la empleada para la prueba del servidor DHCP. El resultado de la conexión se muestra en la Figura 39.

```
administrador@administrador-ThinkPad-L380:~$ ping www.google.es
PING www.google.es (142.250.200.131) 56(84) bytes of data.
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=1 ttl=113 time=13.5 ms
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=2 ttl=113 time=13.9 ms
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=3 ttl=113 time=13.7 ms
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=4 ttl=113 time=13.8 ms
```

Figura 39: Prueba de conexión a Internet desde la red de acceso

Por último, guardamos todos los cambios efectuados en este contenedor en una nueva imagen. Para ello, empleamos el comando *docker commit <container-ID> <new-image-name:tag>* o *docker commit <container-name> <new-image-name:tag>*. En nuestro caso, el nombre de la nueva imagen será *router:v_final*. Para crear un contenedor a partir de esta nueva imagen, ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name router_iptables --privileged router:v_final bash
```

4.4 Instalación y despliegue de un OVS en un contenedor docker

Para la instalación y el despliegue del OVS en el *chron2* se creará un nuevo contenedor empleando la imagen base de Debian. Los pasos para crear este nuevo contenedor son similares a los descritos en el apartado anterior para los contenedores del *router* y el servidor DHCP, por lo que no se detallarán en profundidad.

Como se acaba de mencionar, la imagen base que emplearemos será la Debian en su versión más reciente. Dado que la descargamos anteriormente para realizar la configuración del *router*, simplemente la renombramos de modo que se pueda identificar fácilmente (*docker tag debian:latest ovs:v1*) y ejecutamos el siguiente comando para crear el contenedor:

```
docker run -it --cap-add SYS_ADMIN --network host --name ovs_v1 --privileged ovs:v1 bash
```

Una vez creado el contenedor instalamos manualmente todos los paquetes necesarios para poder desplegar el OVS. Para ello, primero actualizamos el repositorio y posteriormente procedemos con la instalación, ejecutando los siguientes comandos:

```
apt-get update
```

```
apt-get install -y wget net-tools nano iproute2 iputils-ping ifupdown vlan iperf  
iperf3 git curl python-simplejson python-qt4 python-twisted-conch python3.6  
automake autoconf gcc uml-utilities libtool build-essential pkg-config libssl-dev  
iproute2 tcpdump
```

Una vez instaladas todas las dependencias y herramientas necesarias para el despliegue del OVS procedemos con su instalación. La página oficial de Open vSwitch [20] proporciona toda la documentación necesaria para llevar a cabo el proceso de instalación. Existen distintas opciones para instalar Open vSwitch. En este proyecto realizaremos la instalación empleando el código fuente disponible en el repositorio oficial de GitHub. Para ello, clonamos en un directorio denominado “ovs” el repositorio mediante la ejecución del siguiente comando:

```
git clone https://github.com/openvswitch/ovs.git
```

El enlace clona los archivos de la rama *master* de GitHub. Durante este proyecto, la versión empleada del OVS es la 2.15.90. Una vez clonado el repositorio, el siguiente paso es situarnos dentro de la carpeta *ovs* mediante el comando *cd ovs*, para así poder continuar con el proceso de instalación.

Una vez dentro de la carpeta “ovs” ejecutamos el comando *./boot.sh*, mediante el cual se creará el script de configuración que posteriormente se ejecutará con el comando *./configure*. Por último, ejecutamos los comandos *make*, para realizar la compilación del código, y *make install*, para instalar el OVS.

Con la ejecución de estos cuatro comandos llevamos a cabo el proceso de instalación del OVS. El último paso es inicializarlo, para lo cual es necesario ejecutar los siguientes comandos:

```
$ mkdir -p /usr/local/etc/openvswitch
```

```
$ ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
```

```
vswitchd/vswitch.ovsschema

$ mkdir -p /usr/local/var/run/openvswitch

$ ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
  --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
  --private-key=db:Open_vSwitch,SSL,private_key \
  --certificate=db:Open_vSwitch,SSL,certificate \
  --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
  --pidfile --detach --log-file

$ ovs-vsctl --no-wait init

$ ovs-vswitchd --pidfile --detach --log-file

$ export PATH=$PATH:/usr/local/share/openvswitch/scripts
  ovs-ctl start
```

Una vez ejecutados todos los comandos anteriores ya tendremos inicializado nuestro OVS, tal y como se puede observar en la Figura 40.

```
[ ok ] Starting ovsdb-server.
[FAIL] system ID not configured, please use --system-id ... failed!
[ ok ] Configuring Open vSwitch system IDs.
[ ok ] Starting ovs-vswitchd.
[ ok ] Enabling remote OVSDB managers.
```

Figura 40: OVS inicializado

Ahora, para llevar a cabo la configuración del OVS es necesario comprender cómo funciona. Los *switches* virtuales actúan como puentes a los que se pueden conectar diferentes interfaces, y los paquetes que pasan a través de estos puentes se pueden administrar y manipular previamente, como se puede observar en la Figura 41.

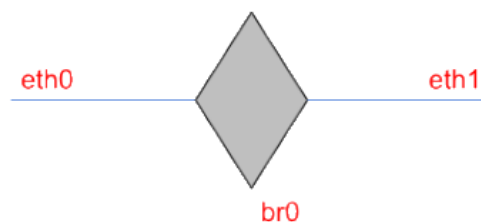


Figura 41: Puente OVS

En nuestro caso, solo necesitamos un único puente, el cual además va a tener un único puerto, correspondiente a la interfaz de red que conecta el *chron2* con la OLT. Los paquetes que lleguen al *chron2* por la otra interfaz, la cual conecta el servidor con la red externa de la facultad, serán encaminados por el *router* hacia el puente y, por lo tanto, hacia la red de acceso. Esta nueva configuración es la que se puede observar en la Figura 42.

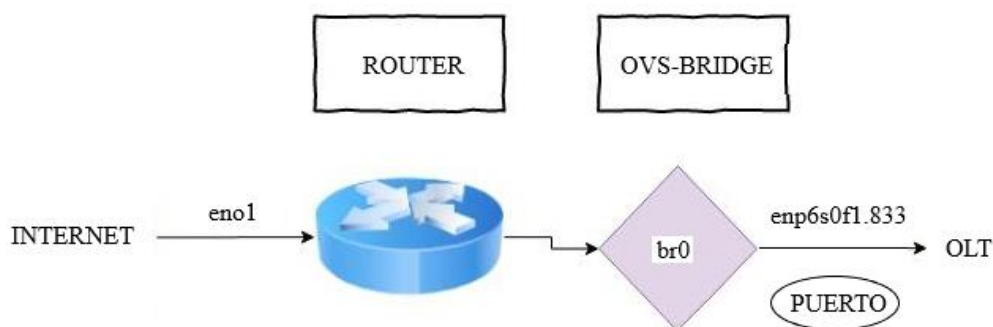


Figura 42: Implementación del puente OVS en el *chron2*

Para la creación del puente y el puerto correspondientes, con el objetivo de alcanzar la configuración anterior, ejecutamos los siguientes comandos:

```
ovs-vsctl add-br br0
```

```
ovs-vsctl add-port br0 enp6s0f1.833
```

De esta manera, creamos un puente en el OVS, denominado *br0*, y agregamos la interfaz *enp6s0f1.833* al puente a través de un puerto. Para asegurarnos de que se ha realizado correctamente la configuración de este nuevo puente ejecutamos el comando *ovs-vsctl show*, cuyo resultado se muestra en la Figura 43.

```
root@chron2:~# ovs-vsctl show
7f13723a-a51b-4489-b543-12c905f780de
  Bridge br0
    Controller "tcp:10.1.4.0:6633"
    Port br0
      Interface br0
        type: internal
      Port enp6s0f1.833
        Interface enp6s0f1.833
    ovs_version: "2.15.90"
root@chron2:~#
```

Figura 43: Configuración del puente *br0*

Una vez configurado el OVS, es necesario reiniciar la interfaz asociada a este (enp6s0f1.833) y dar una dirección IP a la interfaz *br0*, la correspondiente al puente creado. Para ello, ejecutamos los siguientes comandos:

```
ifconfig enp6s0f1.833 0
ifconfig br0 up
ifconfig br0 192.168.100.1 netmask 255.255.255.0
```

De este modo, asignamos al puente la dirección IP correspondiente a la interfaz enp6s0f1.833. Esto es necesario dado que, al agregar el puente a la salida de la interfaz, dicha interfaz se sustituye por el puerto al cual la hemos asociado. Para comprobar que se mantiene la conexión con la red de acceso, podemos hacer un *ping* a la ONT de nivel 3 del laboratorio empleada para las pruebas del servidor DHCP y del *router* (Figura 44).

```
csannun@chron2:~$ ping 192.168.100.140
PING 192.168.100.140 (192.168.100.140) 56(84) bytes of data.
64 bytes from 192.168.100.140: icmp_seq=1 ttl=64 time=0.977 ms
64 bytes from 192.168.100.140: icmp_seq=2 ttl=64 time=0.648 ms
64 bytes from 192.168.100.140: icmp_seq=3 ttl=64 time=0.580 ms
64 bytes from 192.168.100.140: icmp_seq=4 ttl=64 time=0.582 ms
```

Figura 44: Prueba de conexión entre el *chron2* y la ONT de nivel 3

Dado que cada vez que se inicializa el contenedor es necesario volver a ejecutar algunos de los comandos anteriores, se ha creado un *script* para automatizar este proceso. Este *script*, denominado *activarOpenVSwitch.sh*, se localiza en la carpeta *root* dentro del contenedor y su contenido es el siguiente:

```
#!/bin/bash
export PATH=$PATH:/usr/local/share/openvswitch/scripts
ovs-ctl start
ifconfig enp6s0f1.833 0
ifconfig br0 up
ifconfig br0 192.168.100.1 netmask 255.255.255.0
```

Mediante la ejecución de este *script* inicializamos el OVS, reinicamos la interfaz enp6s0f1.833 y configuramos la interfaz *br0*. El comando necesario para ejecutar el *script* es *sh activarOpenVSwitch.sh*.

Por último, para establecer la conexión con el controlador SDN, de modo que sea este quien controle al OVS, es necesario ejecutar el siguiente comando:

```
ovs-vsctl set-controller br0 tcp:10.1.4.0:6633
```

Con este comando, indicamos que queremos establecer conexión con el controlador SDN, cuya dirección IP es 10.1.4.0, a través del puerto 6633. De esta manera, el *switch* está listo para ser centralizado por el controlador.

No obstante, para que la conexión con el controlador funcione correctamente, es necesario indicarle al OVS la versión de *OpenFlow* que queremos emplear. Como se comentó en el Capítulo 2, el estándar empleado para el desarrollo de este trabajo es OpenFlow 1.3. Por tanto, para establecer esta versión ejecutamos el siguiente comando:

```
ovs-vsctl set bridge br0 protocols=OpenFlow13
```

Con esto, ya tendríamos completamente configurado nuestro OVS en el contenedor docker. Por último, guardamos todos los cambios efectuados en este contenedor en una nueva imagen. Para ello, empleamos el comando *docker commit <container-ID> <new-image-name:tag>* o *docker commit <container-name> <new-image-name:tag>*. En nuestro caso, el nombre de la nueva imagen será *ovs:v_final*. Para crear un contenedor a partir de esta nueva imagen, ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name ovs_vfinal --privileged ovs:v_final bash
```

4.5 Despliegue del controlador ONOS en un contenedor docker

Para el despliegue del controlador ONOS se creará un nuevo y último contenedor en el servidor. En este caso, en lugar de emplear una imagen base y realizar el proceso de instalación de manera manual, vamos a emplear una imagen precompilada disponible en Docker Hub, proporcionada por los desarrolladores oficiales de ONOS.

El primer paso es descargarse la imagen oficial de ONOS de Docker Hub [21]. Emplearemos la versión más reciente, para lo cual ejecutamos el comando *docker pull onosproject/onos*. Una vez descargada creamos el contenedor, para lo cual ejecutamos el siguiente comando [22]:


```
docker run -it -d --cap-add SYS_ADMIN --network bridge_test --name onos_vfinal -p 8101:8101 -p 5005:5005 -p 830:830 -p 10.0.103.73:8181 --privileged onosproject/onos:latest.
```

Mediante este comando creamos un contenedor a partir de la imagen de ONOS. Con la opción `-d` hacemos que el contenedor se ejecute en segundo plano. La opción `-p` permite establecer la correspondencia entre los puertos expuestos del servidor y los puertos internos del contenedor. Con la opción `--network bridge_test` hacemos que el contenedor se ejecute en modo *bridge*, pero en una red virtual creada con anterioridad distinta de la de *docker0*.

La dirección IP 10.0.103.73 se corresponde con la interfaz del servidor que conecta este con la red externa de la escuela. A través de esta interfaz se establece la conexión remota con el *chron2*. Por ello, empleamos la dirección IP correspondiente a esta interfaz para poder acceder posteriormente a la interfaz web gráfica (GUI) de ONOS desde un navegador. Por tanto, para acceder a la GUI de ONOS, abrimos un navegador y escribimos la siguiente dirección URL [23]:

```
http://10.0.103.73:8181/onos/ui/#/
```

Se mostrará entonces la interfaz web de ONOS, en la cual tenemos que introducir las credenciales de inicio, esto es, el usuario y la contraseña (*onos/rocks*), tal y como se puede observar en la Figura 45.



Figura 45: Página de inicio de ONOS

Una vez realizada la autenticación, se visualizará la página principal de ONOS, la cual se puede observar en la Figura 46.

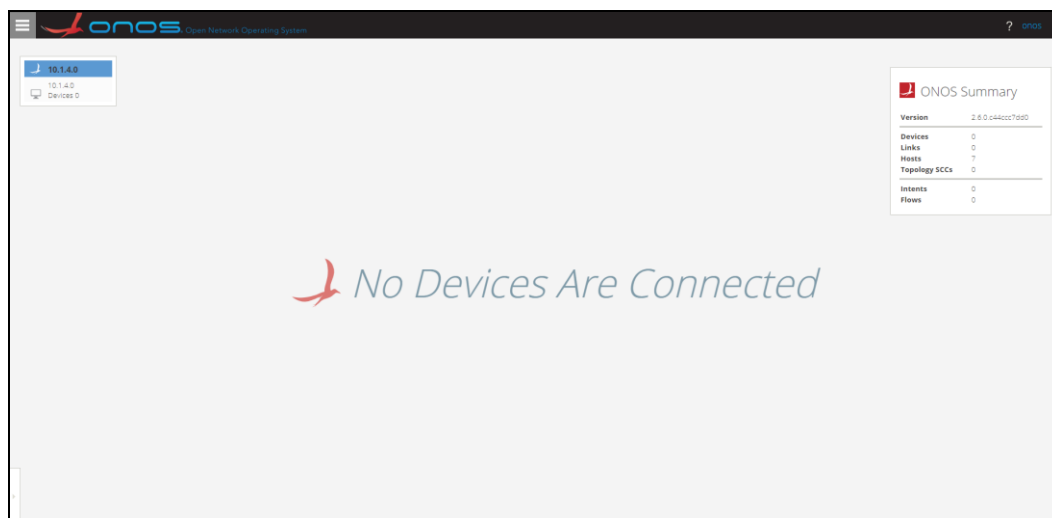


Figura 46: Página principal de ONOS

La interfaz web de ONOS ofrece diferentes opciones en las cuales se pueden visualizar los distintos componentes de la red. Esto incluye una representación gráfica de la topología de la red y su estado (Figuras 46 y 49).

El siguiente paso es conectar el OVS del *chron2* con el controlador ONOS. Esto ya se explicó en el apartado anterior, correspondiente al despliegue del OVS. Para establecer la conexión, es necesario ejecutar el comando `ovs-vsctl set-controller br0 tcp:10.1.4.0:6633` en el contenedor del OVS.

Además de establecer la conexión entre el OVS y el controlador, es necesario activar dos aplicaciones: *org.onosproject.openflow* para poder comunicar los *switches* virtuales con el controlador a través del protocolo *OpenFlow*, y *org.onosproject.fwd* para habilitar el reenvío reactivo de paquetes. Hasta que no se activen ambas aplicaciones, el OVS no será visible por el controlador, ya que no se pueden comunicar entre sí. Para activarlas, vamos a la pestaña *Applications* y buscamos estas dos aplicaciones en la lista de aplicaciones instaladas, tal y como se puede observar en las Figuras 47 y 48.

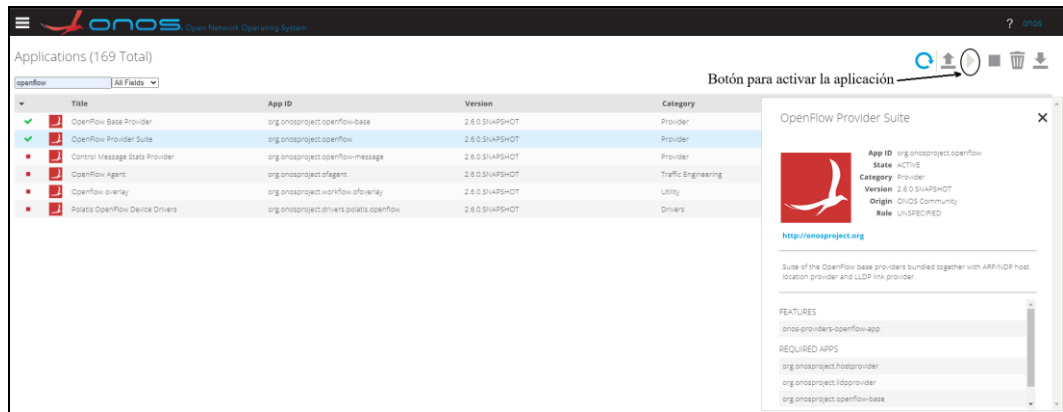


Figura 47: Activación de la aplicación *org.onosproject.openflow*

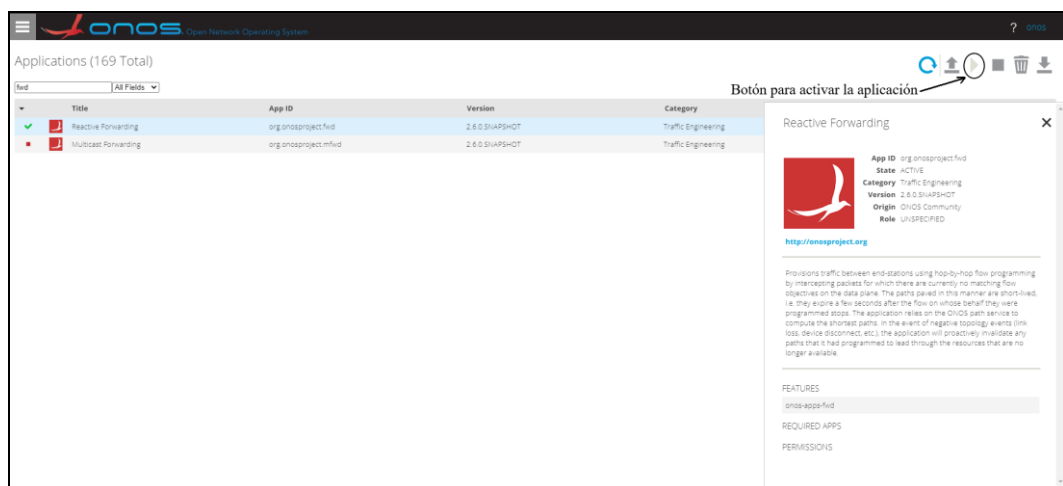


Figura 48: Activación de la aplicación *org.onosproject.fwd*

Una vez localizadas, las seleccionamos y posteriormente pinchamos en el botón de *activar aplicación*, situado arriba a la derecha. Tras la activación de estas dos aplicaciones, el controlador ONOS se conectará automáticamente al OVS que hemos desplegado en un contenedor docker en el *chron2*, lo cual se puede observar en la Figura 49.

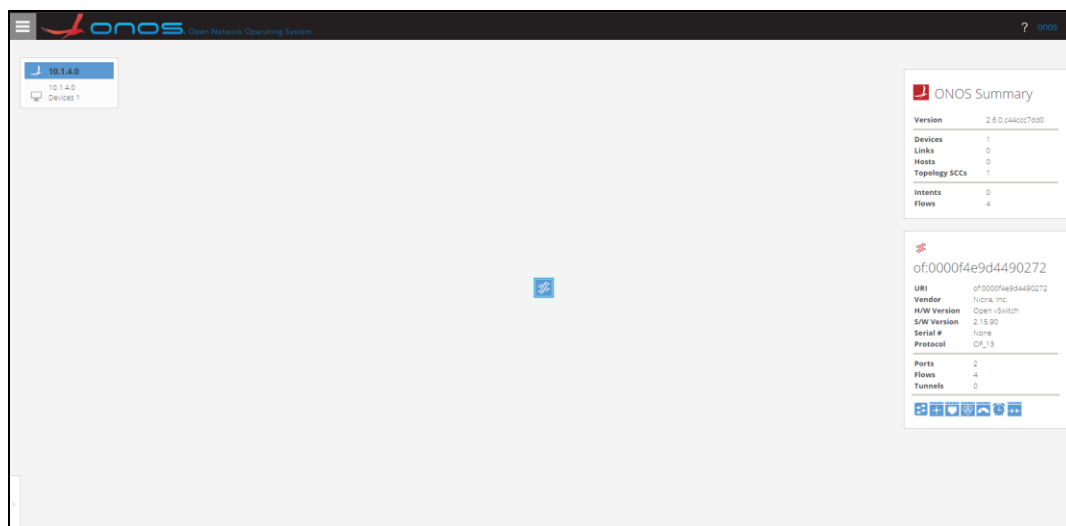


Figura 49: Topología de la red con un OVS activo

Una vez establecida la conexión entre el OVS del servidor y el controlador ONOS, el siguiente paso es modificar las entradas *iptables* del servidor.

Dado que el controlador se ejecuta dentro de un contenedor en modo *bridge*, este no es accesible desde fuera del *chron2*. Por lo tanto, el controlador no será capaz de comunicarse con los OVSs del laboratorio, los que están asociados a las ONTs de nivel 2. Para solventar este problema, modificamos las entradas *iptables* de la misma manera que hicimos para configurar el *router*. No obstante, la sintaxis en este caso es ligeramente distinta. La regla que vamos a aplicar es la siguiente:

```
iptables -t nat -A POSTROUTING -s 192.168.100.0/24 -o br-f73355c08a82 -j MASQUERADE
```

```
iptables -A FORWARD -i br-f73355c08a82 -j ACCEPT
```

```
iptables -A FORWARD -o br-f73355c08a82 -j ACCEPT
```

Las diferentes opciones de esta regla se describen a continuación:

- La opción *-t* especifica a qué tabla debe hacer referencia este comando, en este caso, a la tabla NAT. Esta tabla se consulta cuando un paquete entrante crea una nueva conexión, y tiene dos posibilidades: *PREROUTING* (para actuar sobre los paquetes según lleguen al *kernel*) y *OUTPUT* (para actuar sobre los paquetes que se generan localmente antes de ser enrutados).

- La opción `-A` especifica qué regla queremos agregar en la tabla previamente especificada. En este caso, a diferencia de la regla aplicada para el *router*, esta regla se agrega al final de la tabla. De este modo, la primera regla en aplicarse siempre será la del *router*, en el caso de que se cumplan las condiciones.
- La opción `-s` especifica el origen de los paquetes a los que queremos aplicar la regla. En este caso, agregaremos la subred 192.168.100.0/24, ya que esta es la subred que se ha configurado en la interfaz que conecta *chron2* con la red de acceso.
- La opción `-o` especifica la interfaz por la cual queremos sacar los paquetes que cumplan las reglas anteriores. En nuestro caso, la interfaz de red virtual *br-f73355c08a82*, en la cual se encuentra el contenedor del controlador.
- La opción `-j` indica qué hacer con los paquetes que cumplan con todo lo especificado anteriormente.

De esta manera, conseguimos establecer conexión entre los ordenadores del laboratorio y el contenedor de ONOS. Para comprobar que funciona correctamente, basta con hacer un *ping* desde uno de los ordenadores del laboratorio a la dirección IP 10.1.4.0, correspondiente al controlador ONOS. Nosotros emplearemos uno de los Lenovos disponibles en el laboratorio conectado a la ONT nivel 3 empleada en las pruebas realizadas para comprobar el correcto funcionamiento del *router* y del servidor DHCP, ambos configurados en contenedores docker en *chron2*. Esta prueba se puede observar en la Figura 50.

```
administrador@administrador-ThinkPad-L380:~$ ping 10.1.4.0
PING 10.1.4.0 (10.1.4.0) 56(84) bytes of data:
64 bytes from 10.1.4.0: icmp_seq=1 ttl=62 time=5.41 ms
64 bytes from 10.1.4.0: icmp_seq=2 ttl=62 time=2.84 ms
64 bytes from 10.1.4.0: icmp_seq=3 ttl=62 time=3.33 ms
64 bytes from 10.1.4.0: icmp_seq=4 ttl=62 time=3.07 ms
```

Figura 50: Prueba de conexión entre la red de acceso y el contenedor de ONOS

En la Figura 51 se puede visualizar la topología de la red con los OVSs del lado del cliente activos junto con el OVS desplegado en un contenedor en *chron2*.

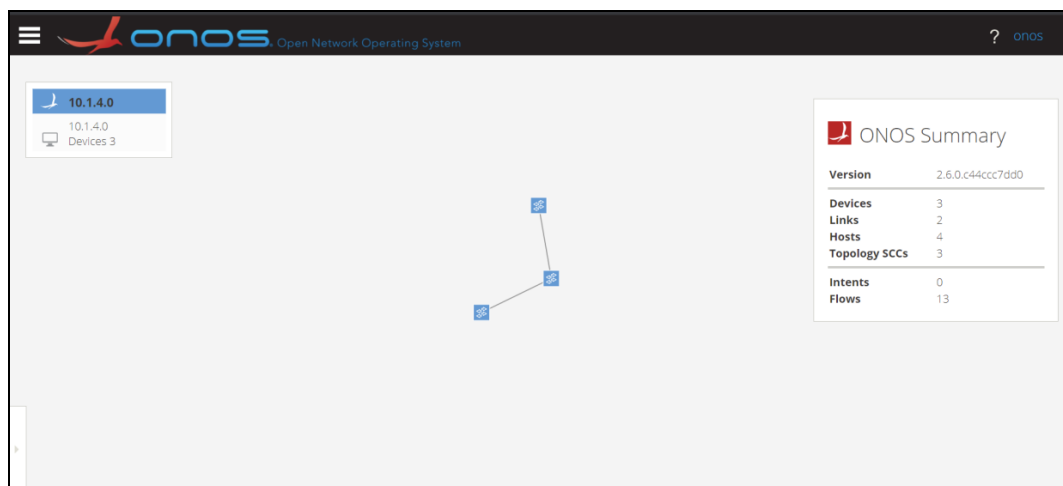


Figura 51: Topología de la red con varios OVS activos, incluidos los asociados a las ONTs de nivel 2 del laboratorio

Con el controlador ONOS correctamente configurado, el siguiente paso es la creación de distintos *flows* y *meters*.

4.6 Configuración de *flows* y *meters* en ONOS

Con ONOS es posible configurar distintos *flows* y *meters* a través de la interfaz gráfica de usuario. Para ello, hacemos uso de la API REST, accesible a través de la documentación de ONOS. Para acceder a la página, abrimos un navegador y escribimos la siguiente dirección URL [24]:

http://10.0.103.73:8181/onos/v1/docs/#/

Antes de ingresar en la interfaz web debemos introducir las credenciales de inicio (*onos/rocks*). Una vez realizada la autenticación, se visualizará una lista con las distintas funciones disponibles, entre las que se encuentran las correspondientes para la creación de los *flows* y los *meters*, tal y como se puede observar en la Figura 52.

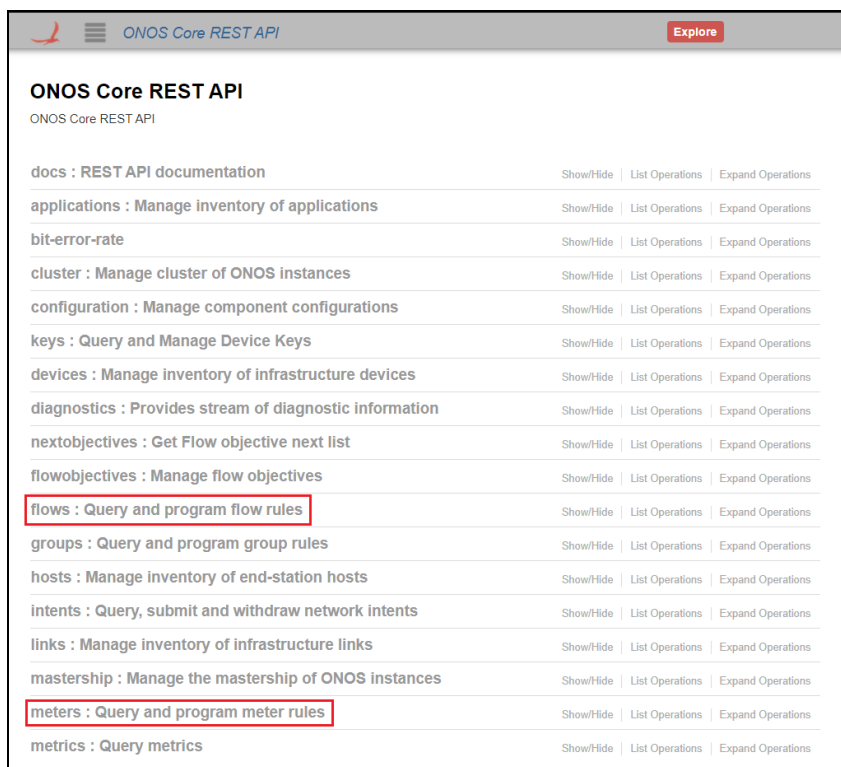


Figura 52: Lista con las funciones disponibles de la API de REST de ONOS

En los siguientes apartados utilizaremos estas dos funciones para configurar *flows* y *meters* con diferentes parámetros.

4.6.1 Configuración de flows en ONOS

De todas las funciones disponibles en la API REST de ONOS seleccionamos la correspondiente a los *flows*. Dentro de esta función tenemos diferentes opciones, tal y como se puede observar en la Figura 53.

flows : Query and program flow rules		Show/Hide	List Operations	Expand Operations
GET	/flows/table/{tableId}	Gets all flow entries for a table		
DELETE	/flows/application/{applId}	Removes flow rules by application ID		
GET	/flows/application/{applId}	Gets flow rules generated by an application		
DELETE	/flows	Removes a batch of flow rules		
GET	/flows	Gets all flow entries		
POST	/flows	Creates new flow rules		
DELETE	/flows/{deviceId}/{flowId}	Removes flow rule		
GET	/flows/{deviceId}/{flowId}	Gets flow rules		
GET	/flows/pending	Gets all pending flow entries		
GET	/flows/{deviceId}	Gets flow entries of a device		
POST	/flows/{deviceId}	Creates new flow rule		

Figura 53: Opciones disponibles dentro de la función *flows*

Las distintas opciones permiten:

- GET: visualizar los *flows* existentes
- DELETE: eliminar distintos *flows*
- POST: crear nuevos *flows*

Para crear un *flow* asociado a un OVS, seleccionamos la segunda de las opciones POST. Los campos que se deben rellenar para la creación de un *flow* son los que se observan en la Figura 54.

POST /flows/{deviceId} Creates new flow rule

Implementation Notes
Creates and installs a new flow rule for the specified device.
Flow rule criteria and instruction description: <https://wiki.onosproject.org/display/ONOS/Flow+Rules>

Parameters

Parameter	Value	Description	Parameter Type	Data Type
deviceId	(required)	device identifier	path	string
appId	(required)	application identifier	query	string
stream	(required)	flow rule JSON	body	Model

Parameter content type:

```

{
  "priority": 40000,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000001",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "CONTROLLER"
      }
    ]
  }
}

```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

Figura 54: Campos para la creación de un *flow*

Para crear un *flow* es necesario rellenar tres campos. El *deviceId* hace referencia al identificador del OVS, el cual se puede obtener dentro de la sección *Devices* de la interfaz web de ONOS. El *appId* hace referencia a la aplicación *org.onosproject.fwd* para el reenvío reactivo de los paquetes. Finalmente, el campo *stream* contendrá las reglas e instrucciones necesarias para la creación del *flow*. Como se puede observar en el lateral derecho de la Figura 52, la API REST proporciona un *flow* de ejemplo en formato JSON que se puede emplear para la creación de nuevos *flows*. Emplearemos este modelo como base para la creación de los distintos *flows*. Dado que posteriormente se realizarán pruebas *end-to-end* para analizar el rendimiento del nuevo escenario SDN-GPON, en este

apartado simplemente vamos a crear un *flow* a modo de ejemplo asociado al OVS instalado en el *chron2*. El formato del *flow* es el siguiente:

```
{
  "priority": 50000,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000f4e9d4490272",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "NORMAL"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      }
    ]
  }
}
```

En este ejemplo hemos creado un *flow* asociado al OVS del *chron2* con prioridad 50000, superior a la del resto de *flows* existentes en el OVS, indicando que el protocolo empleado es IPv4. Una vez configurado el *flow*, pinchamos en el botón *Try it out!* (Figura 54) y lo creamos. Para visualizarlo, podemos hacerlo desde el contenedor del OVS ejecutando el comando `ovs-ofctl -O OpenFlow13 dump-flows br0` o bien a través de la interfaz web seleccionando la opción GET que permite visualizar todos los *flows*. Si lo hacemos desde el contenedor del OVS el resultado es el que se puede observar en la Figura 55. Podemos ver que el *flow* que hemos configurado asociado al OVS del servidor, el que tiene una prioridad de 50000, se ha creado correctamente.

```

root@chron2:~# ovs-ofctl -O OpenFlow13 dump-flows br0
cookie=0x18000b270fce2, duration=18078.555s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
cookie=0x180004664f676, duration=18078.555s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
cookie=0x18000c89df467, duration=18078.555s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
cookie=0x18000cb2dc5ff, duration=18078.555s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
cookie=0x8a00035ed05d3, duration=30.873s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=50000,ip actions=NORMAL
    
```

Figura 55: Visualización del *flow* creado

4.6.2 Configuración de *meters* en ONOS

Para la creación de los *meters* empleando la API REST de ONOS seleccionamos la función correspondiente. Dentro de esta función tenemos diferentes opciones, tal y como se puede observar en la Figura 56.

meters : Query and program meter rules		Show/Hide	List Operations	Expand Operations
GET	/meters			Returns all meters of all devices
DELETE	/meters/{deviceId}/{meterId}			Removes the specified meter
GET	/meters/{deviceId}/{meterId}			Returns a meter by the meter id
GET	/meters/{deviceId}			Returns a collection of meters by the device id
POST	/meters/{deviceId}			Creates new meter rule

Figura 56: Opciones disponibles dentro de la función *meters*

Las distintas opciones permiten:

- GET: visualizar los *meters* existentes
- DELETE: eliminar distintos *meters*
- POST: crear nuevos *meters*

Para crear un *meter*, seleccionamos la opción POST. Los campos que se deben rellenar para la creación del *meter* son los que se observan en la Figura 57.

POST /meters/{deviceId} Creates new meter rule

Implementation Notes
Creates and installs a new meter rule for the specified device.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
deviceId	(required)	device identifier	path	string
stream	(required)	meter rule JSON	body	Model

Parameter content type:

```

{
  "deviceId": "of:0000000000000001",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "REMARK",
      "rate": "0",
      "burstSize": "0",
      "prec": "0"
    }
  ]
}

```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

[Try it out!](#)

Figura 57: Campos para la creación de un *meter*

Para crear un *meter* es necesario rellenar dos campos. El *deviceId* hace referencia al identificador del OVS. El campo *stream* contendrá las reglas e instrucciones necesarias para la creación del *meter*. Como se puede observar en el lateral derecho de la Figura 57, la API REST proporciona un *meter* de ejemplo en formato JSON que se puede emplear para la creación de nuevos *meters*. Emplearemos este modelo como base para la creación de los distintos *meters*. Del mismo modo, dado que posteriormente se realizarán pruebas *end-to-end* para analizar el rendimiento del nuevo escenario SDN-GPON, en este apartado simplemente vamos a crear un *meter* a modo de ejemplo. El formato del *meter* es el siguiente:

```

{
  "deviceId": "of:0000f4e9d4490272",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "DROP",
      "rate": "50000",
      "burstSize": "0",
      "prec": "0"
    }
  ]
}

```

```
}  
]  
}
```

En este ejemplo se ha creado un *meter*, asociado al OVS del *chron2*, que permitirá crear *flows* con una tasa de 50 Mbps. Una vez configurado el *meter*, pinchamos en el botón *Try it out!* (Figura 57) y lo creamos. Para visualizarlo, podemos hacerlo desde el contenedor del OVS ejecutando el comando `ovs-ofctl -O OpenFlow13 dump-meters br0` o bien a través de la interfaz web seleccionando la opción GET que permite visualizar todos los *meters*. Si lo hacemos desde el contenedor del OVS el resultado es el que se puede observar en la Figura 58.

```
root@chron2:~# ovs-ofctl -O OpenFlow13 dump-meters br0  
OFPST_METER_CONFIG reply (OF1.3) (xid=0x2):  
meter=1 kbps burst bands=  
type=drop rate=50000 burst_size=0
```

Figura 58: Visualización del *meter* creado

4.6.3 Configuración de flows con meters en ONOS

Ahora que ya sabemos como se crean los *flows* y los *meters*, vamos a crear el mismo *flow* pero asociándole el *meter* que hemos creado anteriormente. Para ello, el formato JSON del *flow* que hay que crear es el siguiente:

```
{  
  "priority": 50000,  
  "timeout": 0,  
  "isPermanent": true,  
  "deviceId": " of:0000f4e9d4490272",  
  "treatment": {  
    "instructions": [  
      {  
        "type": "OUTPUT",  
        "port": "NORMAL"  
      },  
      {  
        "type": "METER",  
        "meterId": 1
```

```
    }  
  ]  
},  
"selector": {  
  "criteria": [  
    {  
      "type": "ETH_TYPE",  
      "ethType": "0x0800"  
    }  
  ]  
}  
}
```

Por lo tanto, para asociar un *meter* a un *flow* simplemente hay que añadir las siguientes líneas de código al formato JSON del *flow*:

```
{  
  "type": "METER",  
  "meterId": 1  
}
```

4.7 Actualizaciones del escenario SDN en el lado del cliente

En este apartado se describirán los cambios realizados en el lado del cliente. Estos cambios incluyen la instalación de nuevos OVS en contenedores docker y la separación de la red del usuario final de la red de acceso en dos subredes distintas. Para poder instalar los nuevos OVS, es necesario instalar previamente Docker en los ordenadores del laboratorio.

4.7.1 Instalación de Docker en los ordenadores del laboratorio

Antes de nada, para poder tener conexión con la red de acceso, y por lo tanto a Internet, es necesario configurar la VLAN 833 asociada a la interfaz de red que conecta los dos ordenadores del laboratorio con la red de acceso. Esto es necesario ya que, como comentamos durante la configuración del servidor DHCP del servidor, en la red de acceso se emplea la VLAN 833 para la gestión de los servicios de Internet de las distintas

ONTs. Para ello, existe un *script* denominado *configuracionVLAN.sh* dentro de la carpeta *root* de los ordenadores del laboratorio, en el cual están todos los comandos necesarios para realizar la configuración. El contenido de este archivo es el siguiente:

```
ifconfig enp3s6 0
vconfig add enp3s6 833
ifconfig enp3s6.833 up
dhclient enp3s6.833
```

El comando necesario para ejecutar este *script* es *sh configuracionVLAN.sh*. Una vez configurada la VLAN 833, procedemos con la instalación de Docker.

Para llevar a cabo el proceso de instalación de Docker en los ordenadores del laboratorio, la página oficial de Docker [18] proporciona toda la documentación necesaria. No obstante, dado que la distribución instalada en los ordenadores del laboratorio es Linux Mint, hay ciertos pasos que son ligeramente distintos a los recogidos en la documentación oficial para sistemas basados en Ubuntu.

Los pasos que se deben seguir para realizar el proceso de instalación de Docker en Linux Mint son los siguientes [25]:

1. Actualizar el repositorio e instalar los paquetes necesarios, para lo cual ejecutamos los siguientes comandos:

```
sudo apt-get update
sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common
```

2. Importar la clave Docker GPG utilizada para firmar paquetes Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

3. Agregar el repositorio de Docker:

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(. /etc/os-release; echo
"$SUBUNTU_CODENAME") stable"
```

En el caso de que no se agregue el repositorio, lo hacemos de manera manual. Para ello, editamos el archivo `/etc/apt/sources.list.d/additional-repositories.list` con permisos de super-usuario e introducimos el repositorio:

```
deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable
```

4. Actualizar de nuevo el repositorio e instalar Docker Engine:

```
sudo apt-get update
```

```
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
```

En este punto, se crea el grupo de Docker pero no se agregan usuarios. Agregamos nuestro usuario al grupo para poder ejecutar comandos de Docker como usuario sin privilegios.

```
sudo usermod -aG docker $USER
```

Una vez hecho esto, reiniciamos el ordenador para que se apliquen los cambios.

5. Verificar que Docker este funcionando, para lo cual ejecutamos el siguiente comando:

```
docker --version
```

Si todo es correcto, obtendremos una respuesta similar a la siguiente:

```
Docker versión 20.10.6, build 370c289
```

Una vez realizados todos estos pasos, ya tenemos correctamente instalado Docker en los ordenadores del laboratorio y podemos proceder con el despliegue de los nuevos OVSs.

4.7.2 Instalación y despliegue de nuevos OVSs en contenedores docker en los ordenadores del laboratorio

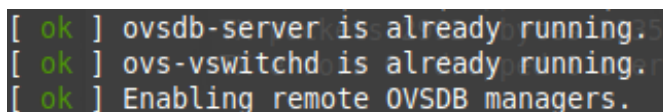
Para la instalación y despliegue de los nuevos OVSs dentro de contenedores docker en los ordenadores del laboratorio, los pasos a seguir son similares a los del OVS desplegado en un contenedor en el *chron2*, descritos en el Apartado 4 de este mismo capítulo. Por ello, en este apartado no se realizará una descripción detallada del proceso de instalación, pero sí del proceso de configuración.

Dado que hay que desplegar dos OVSs en dos ordenadores distintos, solo se describirá el proceso de configuración de uno de ellos, ya que para el otro OVS este proceso es exactamente el mismo.

Al igual que para el despliegue del OVS del servidor, se empleará la imagen base de Debian en su versión reciente. Para crear el contenedor ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name ovs --privileged ovs:carlos bash
```

Una vez instalado e inicializado el OVS en el contenedor, tal y como se puede observar en la Figura 59, el siguiente paso es configurarlo.

A terminal window showing three lines of output in green text on a dark background. The first line says "[ok] ovsdb-server is already running.", the second line says "[ok] ovs-vswitchd is already running.", and the third line says "[ok] Enabling remote OVSDB managers.".

```
[ ok ] ovsdb-server is already running.  
[ ok ] ovs-vswitchd is already running.  
[ ok ] Enabling remote OVSDB managers.
```

Figura 59: OVS inicializado

Para la configuración del OVS, al igual que en el caso del OVS del servidor, solo necesitamos un único puente, el cual además va a tener un único puerto, correspondiente a la interfaz de red que conecta el ordenador del laboratorio con la red de acceso y, por tanto, con el servidor *chron2*. Solo es necesario un puerto debido al proceso de separación de la red del usuario final de la red de acceso en dos subredes distintas, el cual se describirá en el siguiente apartado. Al tratarse de dos subredes independientes, es necesario configurar un *router* en el ordenador del mismo modo que se configuró el *router* del servidor *chron2*. De esta manera, los paquetes que lleguen al ordenador por la otra interfaz, procedentes de la red del usuario, serán encaminados por el *router* hacia el puente y, por lo tanto, hacia la red de acceso. Esta nueva configuración se puede observar en la Figura 60.

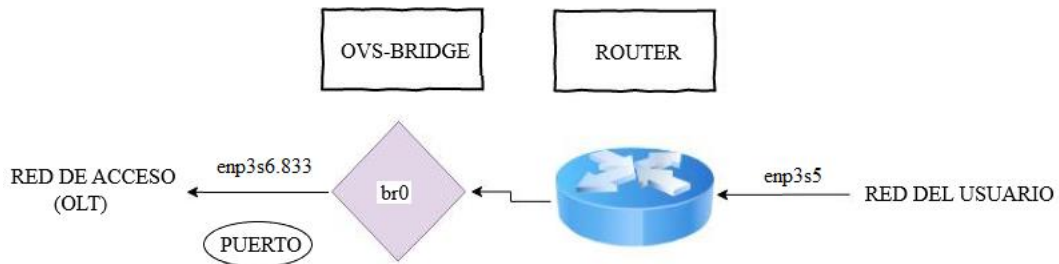


Figura 60: Implementación del puente OVS en el ordenador del laboratorio

Para la creación del puente y el puerto correspondientes, con el objetivo de alcanzar la configuración anterior, ejecutamos los siguientes comandos:

```
ovs-vsctl add-br br0
```

```
ovs-vsctl add-port br0 enp3s6.833
```

De esta manera, creamos un puente en el OVS, denominado *br0*, y agregamos la interfaz *enp3s6.833* al puente a través de un puerto. Para asegurarnos de que se ha realizado correctamente la configuración de este nuevo puente ejecutamos el comando *ovs-vsctl show*, cuyo resultado se muestra en la Figura 61.

```
root@tfglab7-System-Product-Name:~# ovs-vsctl show
1822b1cf-3472-4c62-a6c1-49c5995aab94
Bridge br0 net loopback
  Controller "tcp:10.1.4.0:6633"
  datapath_type: system
  Port br0 net static
    Interface br0 168.5.1
      net type: internal 15.0
  Port enp3s6.833 168.5.0
    Interface enp3s6.833 255
  ovs version: "2.15.90" product-Name:~# nano config
```

Figura 61: Configuración del puente *br0*

Una vez configurado el OVS, es necesario reiniciar la interfaz asociada a este (*enp3s6.833*) y dar una dirección IP a la interfaz *br0*, la correspondiente al puente creado. Para ello, ejecutamos los siguientes comandos:

```
ifconfig enp3s6.833 0
```

```
ifconfig br0 up
```

```
ifconfig br0 192.168.100.2 netmask 255.255.255.0
```

De este modo, asignamos al puente la dirección IP correspondiente a la interfaz `enp3s6.833`. Esto es necesario dado que, al agregar el puente a la salida de la interfaz, dicha interfaz se sustituye por el puerto al cual la hemos asociado.

Dado que cada vez que se inicializa el contenedor es necesario volver a ejecutar algunos de los comandos anteriores, se ha creado un *script* para automatizar este proceso. Este *script*, denominado *activarOpenVSwitch.sh*, se localiza en la carpeta *root* dentro del contenedor y su contenido es el siguiente:

```
#!/bin/bash
mkdir -p /usr/local/etc/openvswitch
ovsdb-tool create /usr/local/etc/openvswitch/conf.db \
    vswitchd/vswitch.ovsschema
mkdir -p /usr/local/var/run/openvswitch
ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
    --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
    --private-key=db:Open_vSwitch,SSL,private_key \
    --certificate=db:Open_vSwitch,SSL,certificate \
    --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
    --pidfile --detach --log-file
ovs-vsctl --no-wait init
ovs-vswitchd --pidfile --detach --log-file
export PATH=$PATH:/usr/local/share/openvswitch/scripts
ovs-ctl start
ifconfig enp3s6.833 0
ifconfig br0 up
ifconfig br0 192.168.100.2/24
route add -net default gw 192.168.100.1 dev br0
```

Mediante la ejecución de este *script* inicializamos el OVS, reinicamos la interfaz `enp3s6.833` y configuramos la interfaz `br0`, asignándole una dirección IP de manera estática. Además, agregamos en la tabla de encaminamiento la puerta de enlace correspondiente a la dirección IP del `chron2`. El comando necesario para ejecutar el *script* es `sh activarOpenVSwitch.sh`.

Por último, para establecer la conexión con el controlador SDN, de modo que sea este quien controle al OVS, es necesario ejecutar el siguiente comando:

```
ovs-vsctl set-controller br0 tcp:10.1.4.0:6633
```

No obstante, para que la conexión con el controlador funcione correctamente, es necesario indicarle al OVS la versión de *OpenFlow* que queremos emplear. Como se comentó en el Capítulo 2, el estándar empleado para el desarrollo de este trabajo es OpenFlow 1.3. Por tanto, para establecer esta versión ejecutamos el siguiente comando:

```
ovs-vsctl set bridge br0 protocols=OpenFlow13
```

Con este último paso, ya tendríamos completamente configurado el OVS. Para el otro OVS, el proceso de configuración es exactamente igual. De igual modo que hacíamos en el *chron2*, podemos guardar los cambios realizados en este contenedor en una nueva imagen mediante el comando `docker commit <container-ID> <new-image-name:tag>` o `docker commit <container-name> <new-image-name:tag>`.

4.7.3 Separación de la red del usuario final de la red de acceso en dos subredes distintas

Para llevar a cabo la separación de la red del cliente de la red de acceso, configuramos la interfaz de red de los ordenadores del laboratorio a la cual se conecta el usuario de manera que esta sea una subred distinta. Este proceso es idéntico para los dos ordenadores, por lo que solo se describirá el proceso de configuración de uno de ellos. En concreto, utilizaremos el mismo ordenador que hemos empleado para describir tanto el proceso de instalación de Docker como el despliegue y configuración del OVS en el contenedor. Para llevar a cabo la configuración, editamos el archivo `/etc/network/interfaces` del ordenador. Los pasos que se deben seguir son los que se describen a continuación:

- Ejecutamos el comando `sudo nano /etc/network/interfaces`
- Una vez dentro del archivo agregamos las siguientes líneas:

```
auto enp3s5
iface enp3s5 inet static
address 192.168.5.1
netmask 255.255.255.0
```

```
network 192.168.5.0
```

```
broadcast 192.168.5.255
```

- Guardamos los cambios efectuados en el archivo mediante la secuencia CTRL-o y salimos del archivo mediante la secuencia CTRL-x.

De este modo, configuramos la interfaz de red de manera estática, dándole una dirección IP fija (192.168.5.1). Al hacerlo de esta manera, es decir, editando el archivo */etc/network/interfaces*, cada vez que se encienda el ordenador la interfaz se configurará de manera automática.

Una vez configurada la interfaz de red que conecta el ordenador del laboratorio con la red del usuario, es necesario configurar un servidor DHCP que asigne direcciones IP a los distintos dispositivos conectados. Para ello, dado que hemos instalado Docker en los ordenadores del laboratorio, vamos a crear un contenedor de manera similar al creado en el servidor para instalar y configurar el servidor DHCP. Dado que los pasos a seguir son similares, no se va a realizar una descripción detallada del proceso de instalación, pero sí del proceso de configuración.

Al igual que para el despliegue del servidor DHCP en el *chron2*, se empleará la imagen base de Ubuntu en su versión más reciente. Para crear el contenedor ejecutamos el siguiente comando:

```
docker run -it --cap-add SYS_ADMIN --network host --name dhcp --privileged  
dhcp:carlos bash
```

Una vez instalados los paquetes necesarios, es necesario configurar una serie de archivos para que el DHCP funcione correctamente. El primer archivo se encuentra en el directorio */etc/dhcp/dhcpd.conf*, en el cual hay que agregar una serie de reglas para la subred 192.168.5.0/24 que hemos definido anteriormente. La configuración exacta es la siguiente:

```
subnet 192.168.5.0 netmask 255.255.255.0 {  
    range 192.168.5.2 192.168.5.254;  
    option domain-name-servers 8.8..8.8,8.8.4.4;  
    option routers 192.168.5.1;  
    option broadcast-address 192.168.5.255;
```

```

    default-lease-time 600;
    max-lease-time 7200;
}

```

El segundo archivo se encuentra en el directorio `/etc/default/isc-dhcp-server`. En este archivo hay que especificar la interfaz por la cual el servidor va a aceptar las peticiones DHCP. En nuestro caso, la interfaz correspondiente es la `enp3s5` configurada anteriormente. Por tanto, agregamos la siguiente regla al archivo:

```
INTERFACESv4="enp3s5"
```

Con esto ya tenemos configurado nuestro servidor DHCP. Para inicializarlo, basta con ejecutar el comando `service isc-dhcp-server start`. Para el otro ordenador, el proceso de configuración de la interfaz de red y del servidor DHCP es el mismo.

Una vez llevada a cabo la configuración del servidor DHCP, lo siguiente es configurar el *router*. Para ello, activamos la utilidad del reenvío de paquetes IP del *kernel* de Linux escribiendo un `1` en el archivo `ip_forward` que se encuentra en la ruta `/proc/sys/net/ipv4/ip_forward`. Esto hace que el ordenador se comporte como un enrutador, es decir, que pueda reenviar paquetes de una subred (la subred en la cual se encuentran los dispositivos de los clientes) a otra subred (la subred que conecta el ordenador del laboratorio con la red de acceso), tal y como se puede observar en la Figura 62.

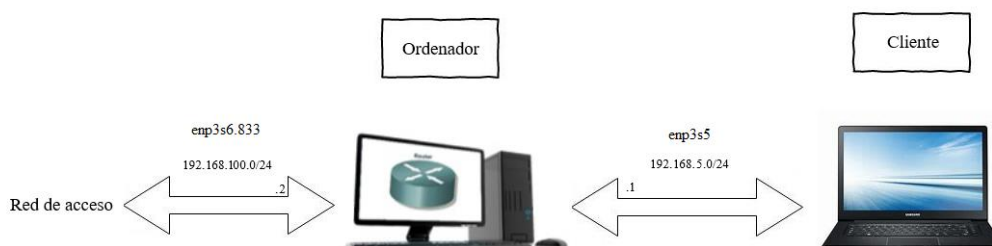


Figura 62: Topología de la configuración del *router* del ordenador del laboratorio

Para que el *router* actúe según lo previsto, vamos a modificar las entradas *iptables* de la misma manera que hicimos para configurar el *router* del *chron2*. No obstante, en este caso vamos a crear un *script*, dado que cada vez que se apague el ordenador se

perderá la configuración. En este *script*, denominado *pruebasArticulo.sh*, introducimos las siguientes líneas:

```
IP_OVS=$(ip address show dev br0 | grep -i "192\.168\.100\." | cut -d " " -f 6 |  
cut -d "/" -f 1)
```

```
iptables -t nat -I POSTROUTING -s 192.168.5.0/24 -j MASQUERADE
```

```
iptables -t nat -I POSTROUTING -s 192.168.5.2/32 -p tcp -j SNAT --to  
$IP_OVS:5000-5019
```

```
iptables -t nat -I POSTROUTING -s 192.168.5.2/32 -p udp -j SNAT --to  
$IP_OVS:5000-5019
```

```
iptables -A FORWARD -i br0 -j ACCEPT
```

```
iptables -A FORWARD -o br0 -j ACCEPT
```

Mediante la ejecución de este *script* obtenemos, en primer lugar, la dirección IP de la interfaz *br0*, ya que esta puede no ser siempre la misma, aunque en nuestro caso es fija. Por lo tanto, es necesario tener activo el OVS del contenedor. Posteriormente, empleamos esta dirección IP para aplicar las reglas *iptables* necesarias para la correcta configuración del *router*, de manera que el usuario que se conecte tenga conexión con el resto de la red de acceso. Se aplican concretamente dos reglas, cada una para un tipo de conexión diferente (TCP o UDP). Además, en estas dos reglas se especifican un conjunto de puertos. Estos puertos son los que nos van a permitir identificar al usuario más allá del *router*, puesto que todos los dispositivos que se conecten en la red del usuario van a emplear la misma dirección IP en la red de acceso, la de la interfaz *br0*. Por lo tanto, estas dos reglas son válidas únicamente para un usuario. Si se conectan más usuarios, hay que definir dos reglas nuevas para cada uno de ellos. Cada uno de estos pares de reglas tendrá asignado un rango distinto de puertos.

Con esto, ya tenemos configurado el *router* para dotar a la red del cliente de conexión con la red de acceso y, por lo tanto, a Internet, tal y como se puede observar en la Figura 63.

```
administrador@administrador-ThinkPad-L380:~$ ping www.google.es
PING www.google.es (142.250.200.131) 56(84) bytes of data.
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=1 ttl=113 time=16.6 ms
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=2 ttl=113 time=16.3 ms
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=3 ttl=113 time=16.9 ms
64 bytes from mad41s14-in-f3.1e100.net (142.250.200.131): icmp_seq=4 ttl=113 time=17.6 ms
```

Figura 63: Prueba de conexión a Internet desde el ordenador del cliente

4.8 Pruebas de rendimiento del nuevo escenario SDN-GPON mediante la creación de distintos *flows* y *meters* en ONOS

En este apartado de la memoria, se van a describir las pruebas *end-to-end* realizadas para analizar el rendimiento del nuevo escenario SDN-GPON desplegado a lo largo de este capítulo. Para ello, se crearán distintos *flows* en el controlador ONOS, cada uno de ellos con distintos *meters* asociados, de manera que podamos analizar el funcionamiento de la red para distintas tasas de transmisión empleando la herramienta *iperf/iperf3*.

Para poder llevar a cabo estas pruebas, tenemos que activar en el *chron2* el servidor DHCP, el OVS, y el controlador ONOS. Con respecto a la parte de la red del cliente, tenemos activar el OVS y el servidor DHCP. En este proyecto se han empleado, por un lado, dos portátiles Lenovo disponibles en el laboratorio, y por otro, dos ordenadores de sobremesa, cada uno de ellos asociado a una de las ONTs de nivel 2 registradas en el puerto PON 3 de la OLT. El ordenador más nuevo permite unas tasas máximas de transmisión superiores, ya que este cuenta con un procesador modelo INTEL I7 9700 3.0Ghz 12mb socket 1151 BOX [26], una placa base modelo GIGABYTE GA-Z390 DESIGNARE [27] y dos tarjetas de red externas modelo TPLINK TG-3468 LP [28]. Sin embargo, para realizar las pruebas vamos a emplear el ordenador más antiguo, ya que con el nuevo la conexión entre el OVS y el controlador ONOS no es del todo estable debido, en principio, a las tarjetas de red. No obstante, también se recogerán los valores obtenidos en pruebas realizadas con este ordenador al final de este apartado. El análisis profundo de este problema de inestabilidad del OVS en este ordenador se escapa al alcance de este TFG, por lo que su estudio se continuará en sucesivos trabajos.

Para realizar las pruebas mencionadas es importante que la ONT de nivel 2 tenga asociado un perfil de Internet en TGMS. Nosotros vamos a emplear el perfil

Internet_700Mbps, ya que este es el más alto de todos los configurados y nos permite analizar la estabilidad de los OVS con tasas de transmisión considerablemente altas. Una vez tengamos todos los contenedores activos y correctamente configurados, de acuerdo a lo que se ha ido describiendo en lo largo de capítulo, procedemos a realizar las distintas pruebas *end-to-end*. Para ello, hemos empleado el *chron2* como servidor y uno de los Lenovos como usuario final, conectado a uno de los ordenadores del laboratorio.

Dado que ahora, con la nueva configuración de la red del cliente, esta es independiente de la red de acceso, vamos a realizar las pruebas en el sentido *upstream*, es decir, desde el usuario final hacia el *chron2*. Por tanto, los *flows* que vamos a crear van a estar asociados al OVS desplegado en el ordenador del laboratorio, el cual controlará el tráfico de los usuarios conectados a la red residencial, en este caso el ordenador Lenovo.

No obstante, antes de comenzar con las pruebas, es necesario crear unos *flows* por defecto. Estos *flows* son necesarios dado que, por algún motivo, los *flows* que crea ONOS por defecto en los OVS limitan el ancho de banda de la conexión a unos 70 Mbps, tal y como se puede observar en la Figura 64.

```
administrador@administrador-ThinkPad-L380:~$ iperf3 -c 192.168.100.1
Connecting to host 192.168.100.1, port 5201
[ 4] local 192.168.5.2 port 36944 connected to 192.168.100.1 port 5201
[ ID] Interval      Transfer    Bandwidth   Retr  Cwnd
[ 4] 0.00-1.00    sec  5.73 MBytes  48.1 Mbits/sec    3   91.9 KBytes
[ 4] 1.00-2.00    sec  5.47 MBytes  45.9 Mbits/sec    1   126 KBytes
[ 4] 2.00-3.00    sec  8.76 MBytes  73.5 Mbits/sec    0   170 KBytes
[ 4] 3.00-4.00    sec  8.08 MBytes  67.8 Mbits/sec    0   202 KBytes
[ 4] 4.00-5.00    sec  9.51 MBytes  79.7 Mbits/sec    0   236 KBytes
[ 4] 5.00-6.00    sec  9.26 MBytes  77.7 Mbits/sec    1   262 KBytes
[ 4] 6.00-7.00    sec  8.82 MBytes  74.0 Mbits/sec    1   286 KBytes
[ 4] 7.00-8.00    sec  9.01 MBytes  75.6 Mbits/sec    0   308 KBytes
[ 4] 8.00-9.00    sec  9.07 MBytes  76.1 Mbits/sec    0   329 KBytes
[ 4] 9.00-10.00   sec  9.20 MBytes  77.1 Mbits/sec    0   351 KBytes
-----
[ ID] Interval      Transfer    Bandwidth   Retr
[ 4] 0.00-10.00   sec  82.9 MBytes  69.5 Mbits/sec    6
[ 4] 0.00-10.00   sec  81.2 MBytes  68.1 Mbits/sec    0
sender
receiver
```

Figura 64: Prueba de conectividad *end-to-end* inicial con el ordenador antiguo

En concreto tenemos que crear dos *flows* por defecto, uno para el OVS del *chron2* y otro para el OVS del lado del cliente. Estos *flows* deben tener una prioridad superior a los que crea ONOS por defecto en los OVSs, de manera que sean estos los que actúen por nivel de prioridad. El formato JSON de estos dos *flows* es el siguiente:


```
{
  "priority": 50000,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": " of:0000f4e9d4490272",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "NORMAL"
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      }
    ]
  }
}
```

En este caso, el *flow* mostrado está asociado al OVS del *chron2*. Para crear el otro *flow*, correspondiente al OVS del ordenador del laboratorio, simplemente tenemos que modificar el campo correspondiente al *deviceId* (*of:000000805a4a90cc*). Estos dos *flows* no tienen ningún efecto en la red, simplemente permiten solventar el problema de la limitación del ancho de banda de la conexión.

```

administrador@administrador-ThinkPad-L380:~$ iperf3 -c 192.168.100.1
Connecting to host 192.168.100.1, port 5201
[ 4] local 192.168.5.2 port 36964 connected to 192.168.100.1 port 5201
[ ID] Interval      Transfer    Bandwidth    Retr  Cwnd
[ 4]  0.00-1.00    sec  36.2 MBytes  304 Mbits/sec  2    301 KBytes
[ 4]  1.00-2.00    sec  33.7 MBytes  283 Mbits/sec  1    297 KBytes
[ 4]  2.00-3.00    sec  33.7 MBytes  283 Mbits/sec  1    293 KBytes
[ 4]  3.00-4.00    sec  33.7 MBytes  283 Mbits/sec  1    287 KBytes
[ 4]  4.00-5.00    sec  33.7 MBytes  283 Mbits/sec  1    283 KBytes
[ 4]  5.00-6.00    sec  33.7 MBytes  283 Mbits/sec  1    277 KBytes
[ 4]  6.00-7.00    sec  33.7 MBytes  283 Mbits/sec  1    272 KBytes
[ 4]  7.00-8.00    sec  33.8 MBytes  284 Mbits/sec  1    267 KBytes
[ 4]  8.00-9.00    sec  33.7 MBytes  283 Mbits/sec  1    263 KBytes
[ 4]  9.00-10.00   sec  33.8 MBytes  284 Mbits/sec  1    256 KBytes
-----
[ ID] Interval      Transfer    Bandwidth    Retr
[ 4]  0.00-10.00   sec  340 MBytes  285 Mbits/sec  11
[ 4]  0.00-10.00   sec  337 MBytes  283 Mbits/sec
sender
receiver

```

Figura 65: Prueba de conectividad *end-to-end* con los *flows* por defecto empleando el ordenador antiguo

En la Figura 65 se muestra que el ancho de banda máximo del enlace es de 285 Mbps, que se corresponde con la capacidad máxima de las tarjetas de red del ordenador antiguo en este escenario. Con el ordenador nuevo se obtiene un ancho de banda máximo de 644 Mbps, ya que el perfil de Internet configurado en TGMS para esta ONT es de 700 Mbps, tal y como se puede observar en la Figura 66.

```

csannun@chron2:~$ iperf3 -s
-----
Server listening on 5201
-----
Accepted connection from 192.168.100.66, port 5000
[ 5] local 192.168.100.1 port 5201 connected to 192.168.100.66 port 5001
[ ID] Interval      Transfer    Bitrate
[ 5]  0.00-1.00    sec  73.5 MBytes  616 Mbits/sec
[ 5]  1.00-2.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  2.00-3.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  3.00-4.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  4.00-5.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  5.00-6.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  6.00-7.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  7.00-8.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  8.00-9.00    sec  77.2 MBytes  647 Mbits/sec
[ 5]  9.00-10.00   sec  77.2 MBytes  647 Mbits/sec
[ 5] 10.00-10.05   sec  4.11 MBytes  647 Mbits/sec
-----
[ ID] Interval      Transfer    Bitrate
[ 5]  0.00-10.05   sec  772 MBytes  644 Mbits/sec
receiver

```

Figura 66: Prueba de conectividad *end-to-end* con los *flows* por defecto empleando el ordenador nuevo

Una vez configurados estos dos *flows* por defecto, ya podemos configurar los *flows* necesarios para llevar a cabo las pruebas. Como se ha mencionado anteriormente, las pruebas se van a realizar en el sentido *upstream*, de modo que los *flows* creados van a estar asociados al OVS de la red residencial del cliente (*deviceId:of:000000805a4a90cc*). Si empleamos una conexión TCP, el formato JSON del *flow* es el siguiente:

```
{
  "priority": 60000,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:000000805a4a90cc ",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "NORMAL"
      },
      {
        "type": "METER",
        "meterId": 1
      }
    ]
  },
  "selector": {
    "criteria": [
      {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
      {
        "type": "IP_PROTO",
        "protocol": 6
      },
      {
        "type": "IPV4_SRC",
        "ip": "192.168.100.2/32"
      }
    ]
  }
}
```

En concreto, creamos un *flow* con prioridad 60000, de modo que nos aseguramos de que sea este el que actúe, ya que es el valor más alto de todos los *flows* que tiene configurados el OVS. También indicamos el *meter* que queremos emplear, el protocolo de transporte, en nuestro caso TCP, y por último la dirección IP del OVS de origen. Esta dirección IP se corresponde con la de la interfaz *br0* del OVS.

Este es el formato de todos los *flows* que se van a crear. El único campo que varía en cada uno de ellos es el identificador del *meter* que queremos asociar (*meterId*). Por tanto, vamos a proceder ahora a la creación de los *meters*. Así pues, los *meters* que vamos a crear también van a estar asociados al OVS del cliente (*deviceId: of:000000805a4a90cc*). El formato JSON del *meter* es el siguiente:

```
{
  "deviceId": "of:000000805a4a90cc ",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "DROP",
      "rate": "50000",
      "burstSize": "0",
      "prec": "0"
    }
  ]
}
```

En este caso concreto, creamos un *meter* que permitirá crear *flows* con una tasa de transmisión de 50 Mbps. El formato de los *meters* es siempre el mismo. El único campo que varía en cada uno de ellos es el valor de la tasa de transmisión (*rate*) que queremos asociar al *flow* correspondiente.

Para realizar las pruebas con el ordenador antiguo vamos a crear distintos *meters*, empezando en 50 Mbps y hasta los 300 Mbps, en saltos de 50 Mbps. Para cada uno de estos *meters*, crearemos los *flows* correspondientes. Para analizar el ancho de banda con cada uno de los *flows* vamos a emplear *iperf3*. Los pasos que se deben seguir son:

1. Ejecutamos en el *chron2* el comando *iperf3 -s*.
2. Ejecutamos en el portátil del cliente el comando *iperf3 -c IP_servidor*. En nuestro caso, el comando sería *iperf3 -c 192.168.100.1* (Figura 67).

```

administrador@administrador-ThinkPad-L380:~$ iperf3 -c 192.168.100.1
Connecting to host 192.168.100.1, port 5201
[ 4] local 192.168.5.2 port 36984 connected to 192.168.100.1 port 5201
[ ID] Interval           Transfer     Bandwidth   Retr   Cwnd
[ 4]  0.00-1.00   sec   36.2 MBytes  303 Mb/s    2     298 K
[ 4]  1.00-2.00   sec   18.5 MBytes  155 Mb/s   1039    9.9 K
[ 4]  2.00-3.00   sec   17.2 MBytes  144 Mb/s    886   11.3 K
[ 4]  3.00-4.00   sec   17.2 MBytes  144 Mb/s   1050   45.2 K
[ 4]  4.00-5.00   sec   17.4 MBytes  146 Mb/s    950   12.7 K
[ 4]  5.00-6.00   sec   17.2 MBytes  144 Mb/s    958    9.9 K
[ 4]  6.00-7.00   sec   17.2 MBytes  144 Mb/s    826   24.0 K
[ 4]  7.00-8.00   sec   17.4 MBytes  146 Mb/s   1007   14.1 K
[ 4]  8.00-9.00   sec   17.2 MBytes  144 Mb/s    803   19.8 K
[ 4]  9.00-10.00  sec   17.3 MBytes  145 Mb/s    861   12.7 K
-----
[ ID] Interval           Transfer     Bandwidth   Retr
[ 4]  0.00-10.00  sec   193 MBytes  162 Mb/s   8382
[ 4]  0.00-10.00  sec   190 MBytes  159 Mb/s
sender
receiver

```

Figura 67: Prueba de conectividad *end-to-end* con un *flow* de 150 Mbps empleando el ordenador antiguo

Para las pruebas realizadas con el ordenador nuevo se crearon distintos *meters*. En ese caso se empezó en 50 Mbps y se llegó hasta los 700 Mbps, en saltos de 50 Mbps. Para cada *meter* se crearon los *flows* correspondientes. La Tabla 1 recoge los resultados obtenidos empleando el ordenador nuevo para los distintos *flows* configurados:

Meters (Mbps)	Resultados <i>iperf3</i> (Mbps)
50	49,3
100	98,7
150	148
200	197
250	247
300	296
350	345
400	395
450	444
500	493
550	542
600	591
650	641
700	647

Tabla 1: Resultados obtenidos de las distintas pruebas con el ordenador nuevo

Como se puede observar en la Tabla 1, los valores de ancho de banda obtenidos para cada uno de los distintos *flows* son similares a los configurados en los *meters*. Por lo tanto, el rendimiento del nuevo escenario SDN-GPON es muy bueno empleando este ordenador. Con el ordenador antiguo, el rendimiento de la red es igual de bueno, es decir, se obtienen valores de ancho de banda para cada uno de los *flows* similares a los configurados en los *meters*. No obstante, con esta torre la capacidad máxima del enlace es bastante inferior debido a las limitaciones de sus interfaces de red. El análisis de la capacidad máxima de las interfaces de red de todos los ordenadores del lado del cliente fue realizado en un TFG previo [29].

4.9 Conclusiones

En este capítulo de la memoria se ha realizado una descripción detallada del proceso de configuración del nuevo escenario SDN-GPON tras la activación del puerto 10G de la OLT. En primer lugar, se ha realizado una introducción a Docker, para lo cual se han presentado los principales comandos y un caso práctico concreto, en el cual se ha empleado una imagen base de Ubuntu para crear un contenedor e instalar en este algunos de los paquetes básicos para la configuración y gestión de redes. También se han descrito los distintos modos que proporciona Docker en cuanto al uso de las interfaces de red a la hora de crear contenedores, así como las características, ventajas e inconvenientes de cada uno de ellos. Posteriormente, se ha descrito de manera detallada el proceso de creación y despliegue de los distintos contenedores en los que se van a ejecutar cada una de las herramientas necesarias en este nuevo escenario SDN-GPON. Por último, se han comentado las actualizaciones SDN llevadas a cabo en el lado del cliente y se han realizado distintas pruebas para analizar el rendimiento de todo este nuevo escenario SDN-GPON desplegado en el servidor externo.

5

Despliegue de la aplicación de gestión de la red GPON en el servidor *chron2*

5.1 Introducción

En este capítulo de la memoria se va a describir el proceso de despliegue de la aplicación de gestión de la red GPON desarrollada en anteriores TFGs en el *chron2*, empleando para ello una nueva herramienta denominada Docker-Compose.

Para ello, se va a realizar, en primer lugar, una introducción a esta nueva herramienta. Posteriormente, se describirán todos los pasos necesarios para llevar a cabo el despliegue de la aplicación. Esto incluye la creación de dos nuevos contenedores docker: uno en el cual se va a ejecutar la aplicación desarrollada en Python y otro en el que se va a desplegar la base de datos empleada por la aplicación. Por último, se comentarán algunos cambios realizados en el código con el objetivo de mejorar el funcionamiento de la aplicación y se describirá el proceso de creación de distintos *flows* y *meters* a través del lenguaje de programación Python.

5.2 Docker-Compose

Docker-Compose es una herramienta que permite definir y ejecutar aplicaciones que emplean múltiples contenedores docker [30]. Docker-Compose emplea un archivo YAML para configurar los distintos servicios de los que consta una aplicación. De esta manera, todos los servicios se pueden crear e inicializar mediante la ejecución de un único comando. Para emplear Docker-Compose, los pasos a seguir son los siguientes:

1. Definir el entorno de la aplicación empleando un archivo Dockerfile, de modo que pueda reproducirse fácilmente en cualquier lugar.
2. Definir los servicios que componen la aplicación mediante un archivo denominado `docker-compose.yml`, de modo que puedan ejecutarse de manera conjunta en un entorno aislado.
3. Ejecutar el comando `docker-compose up` para inicializar y ejecutar toda la aplicación.

Para llevar a cabo el despliegue de la aplicación de gestión de la red GPON en el servidor *chron2* se seguirán estos tres pasos. Por comodidad, para definir y configurar los archivos necesarios vamos a emplear un ordenador en lugar de hacerlo directamente en el *chron2*, ya que el servidor carece de una interfaz de usuario. Posteriormente, se copiarán todos los archivos necesarios al servidor para poder lanzar la aplicación desde este.

5.3 Despliegue de la aplicación empleando un archivo Dockerfile y Docker-Compose

Para empezar, vamos a crear un directorio de trabajo en nuestro ordenador denominado *Pruebas_Aplicacion*, en el cual vamos a tener todas las carpetas y archivos necesarios, tal y como se puede observar en la Figura 68.

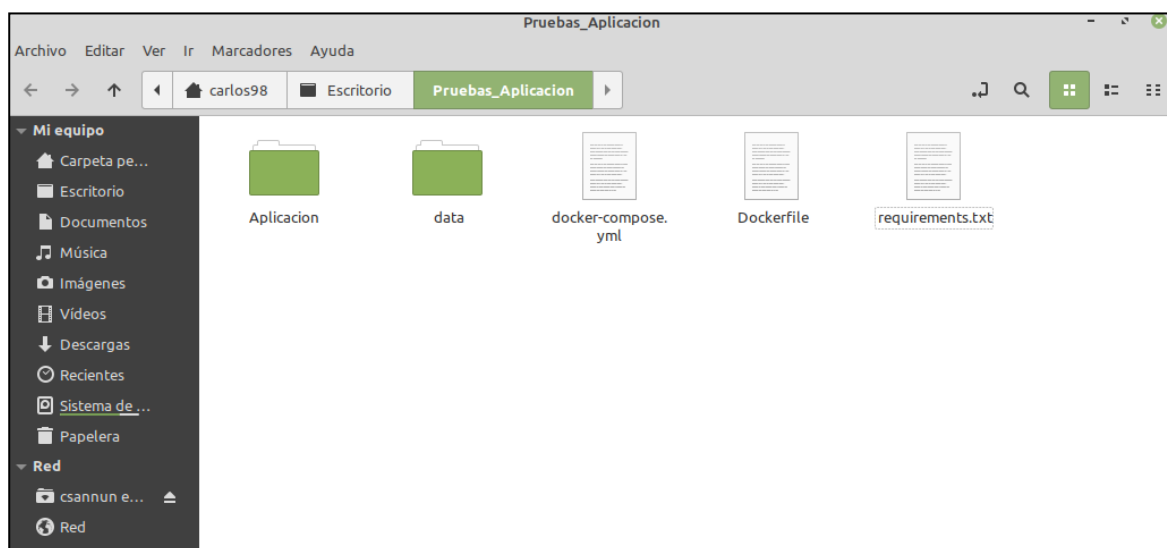
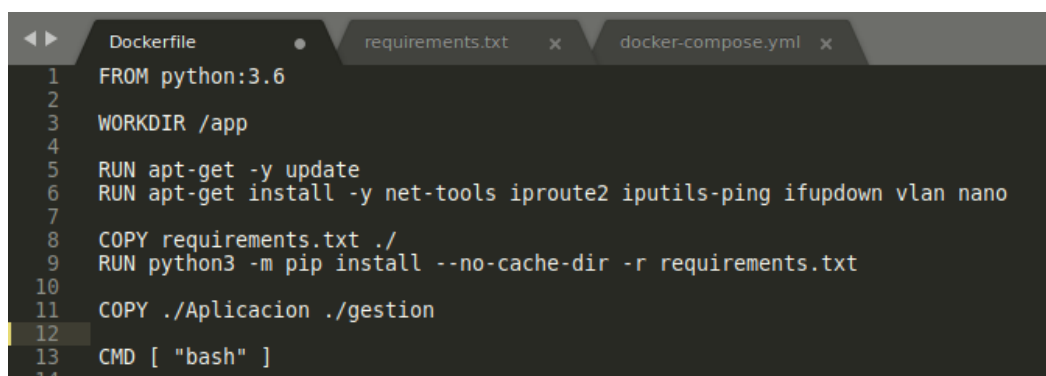


Figura 68: Contenido del directorio *Pruebas_Aplicacion*

En concreto tendremos dos carpetas (*Aplicacion* y *data*) y tres archivos (Dockerfile, requirements.txt y docker-compose.yml). En la carpeta *Aplicación* se encuentra el código correspondiente a la aplicación de gestión de la red GPON. La carpeta *data* se empleará posteriormente para almacenar la base de datos.

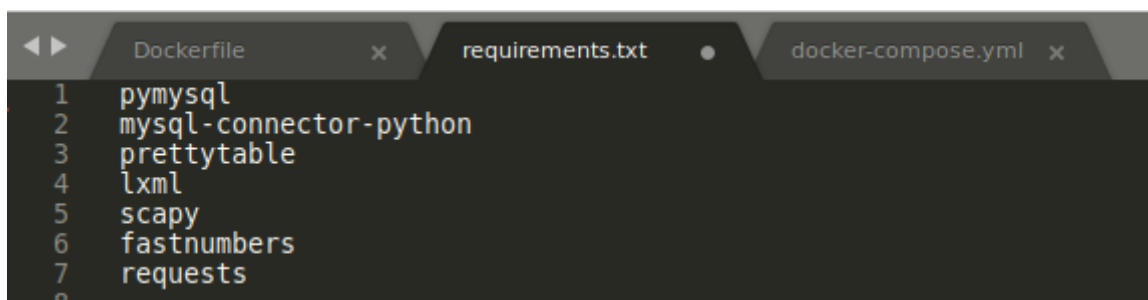
En primer lugar, vamos a crear el archivo Dockerfile con todas las instrucciones necesarias para definir el entorno de la aplicación. Como se comentó en el Apartado 4 del Capítulo 2, un Dockerfile es un archivo de texto simple con un conjunto de comandos o instrucciones necesarias para crear y ejecutar una imagen. Para la creación de una imagen es necesario definir, al principio del archivo Dockerfile, la imagen base que se quiere emplear. Dado que nuestra aplicación está desarrollada en Python, esta será nuestra imagen base. El archivo Dockerfile creado es el que se puede observar en la Figura 69.



```
1 FROM python:3.6
2
3 WORKDIR /app
4
5 RUN apt-get -y update
6 RUN apt-get install -y net-tools iproute2 iputils-ping ifupdown vlan nano
7
8 COPY requirements.txt ./
9 RUN python3 -m pip install --no-cache-dir -r requirements.txt
10
11 COPY ./Aplicacion ./gestion
12
13 CMD [ "bash" ]
14
```

Figura 69: Archivo Dockerfile con las instrucciones necesarias

Definimos, al comienzo del archivo, la imagen de Python que vamos a emplear. Dado que la versión de Python empleada en el laboratorio es la 3.6, nosotros vamos a emplear también esta misma versión para que no haya ningún tipo de problema. Posteriormente, creamos dentro del contenedor un directorio, denominado *app*, en el cual vamos a copiar el código de nuestra aplicación. Lo siguiente es instalar las librerías y dependencias necesarias para que nuestra aplicación se pueda ejecutar. Para ello, primero actualizamos el repositorio e instalamos los paquetes necesarios. A continuación, copiamos dentro del directorio *app* creado anteriormente el fichero *requirements.txt* con las distintas librerías que requiera la aplicación. El contenido de este fichero es el que se puede observar en la Figura 70.

A screenshot of a code editor window with three tabs: 'Dockerfile', 'requirements.txt', and 'docker-compose.yml'. The 'requirements.txt' tab is active and shows a list of Python packages. The text is as follows:

```
1  pymysql
2  mysql-connector-python
3  prettytable
4  lxml
5  scapy
6  fastnumbers
7  requests
8
```

Figura 70: Contenido del fichero requirements.txt

Una vez copiado este fichero dentro del contenedor, lo empleamos para instalar las distintas librerías. Para ello se ejecuta dentro del contenedor el comando `python3 -m pip install --no-cache-dir -r requirements.txt`. Mediante este comando instalamos las distintas librerías definidas en el fichero `requirements.txt`. La opción `-r` nos permite ejecutar este comando de manera recursiva, esto es, para cada uno de las filas que contiene el archivo, ya que en cada fila se define una librería. Lo siguiente es copiar la aplicación de gestión de la red GPON dentro del contenedor en un directorio denominado *gestión* definido dentro del directorio *app*, de modo que la ruta de la aplicación dentro del contenedor será `/app/gestión`. Por último, indicamos el programa que se debe ejecutar al crear el contenedor, en nuestro caso el programa *bash*. Esto nos va a permitir interactuar con el contenedor, es decir, disponer de una terminal para poder lanzar nuestra aplicación.

Una vez creado el archivo `Dockerfile`, en el cual hemos definido el entorno de nuestra aplicación, el siguiente paso es crear el archivo `docker-compose.yml`. El contenido de este archivo es el que se puede observar en la Figura 71.

```
Dockerfile x requirements.txt x docker-compose.yml
1  version: "3"
2
3  services:
4    app_gestion:
5      build: .
6      image: aplicacion python:v1
7      network_mode: host
8      container_name: aplicacion_python
9      volumes:
10     - ./Aplicacion:/app/gestion
11     stdin open: true
12     tty: true
13
14   mysql:
15     image: mysql:latest
16     network_mode: host
17     container_name: mysqldb
18     volumes:
19     - ./data/db:/var/lib/mysql
20     - ./data/schema:/schema
21     restart: always
22     environment:
23       MYSQL_DATABASE: gponServices
24       MYSQL_ROOT_PASSWORD: tfg_2017
25
```

Figura 71: Contenido del fichero docker-compose.yml

En este archivo vamos a definir los servicios de los que va a constar nuestra aplicación. Como se ha comentado anteriormente, la aplicación de gestión de la red GPON consta de dos servicios únicamente: uno es el código en Python de la propia aplicación y el otro una base de datos. Por tanto, estos dos servicios son los que vamos a definir.

En primer lugar, definimos el servicio relacionado con el código de la aplicación. Para ello, hacemos uso del archivo Dockerfile creado anteriormente, el cual se encuentra, dentro de nuestro ordenador, en el mismo directorio de trabajo que el archivo docker-compose.yml. Con la opción *build* vamos a generar la imagen correspondiente a partir de este archivo Dockerfile, acorde a las instrucciones definidas. Con la opción *image* damos un nombre a la imagen y con la opción *network_mode* indicamos que el contenedor que se va a crear a posteriormente a partir de la imagen va a estar en modo *host*, de modo que la aplicación se pueda comunicar, desde el servidor, con la red de acceso del laboratorio para poder llevar a cabo su gestión y control. Con la opción *container_name* damos un nombre al contenedor que se cree. Por último, con la opción *volumes* compartimos el contenido del directorio *Aplicacion* de nuestro ordenador con el contenido del directorio */app/gestion* del contenedor. Esto va a permitir poder realizar cambios en el código de la

aplicación desde nuestro ordenador, y que estos cambios tengan efecto dentro del contenedor. De esta manera, podemos emplear algún editor de código como *Sublime Text* para programar, en lugar de tener que hacerlo por terminal dentro del contenedor empleando, por ejemplo, *nano*. En resumen, con la opción *volumes* no es necesario crear una nueva imagen cada vez que se haga alguna modificación en el código de la aplicación, ya que estos cambios también tendrán efecto dentro del contenedor. Las dos últimas opciones nos van a permitir que una vez que se ejecute el programa *bash* dentro del contenedor su ejecución se siga mostrando por terminal, es decir, hacemos que la ejecución del programa *bash* sea interactiva.

En segundo lugar, definimos el servicio relacionado con la base de datos de la aplicación. Para ello, en este caso hacemos uso de la imagen oficial de MySQL disponible en Docker Hub [31]. Con la opción *image* indicamos la imagen que queremos emplear, en nuestro caso la imagen de MySQL en su versión más reciente. Con la opción *network_mode* indicamos que el contenedor que se va a crear a posteriormente a partir de la imagen va a estar en modo *host*. Hacemos esto para que no sea necesario modificar en el código de la aplicación las conexiones que se realizan con la base de datos, ya que de este modo, esta va a seguir siendo accesible a través de la dirección de *localhost*. Con la opción *container_name* damos un nombre al contenedor. Con la opción *volumes* vamos a compartir, en este caso, el contenido de dos directorios. Por un lado, copiamos el archivo *gponServices.sql* contenido dentro de la carpeta */data/schema* de nuestro ordenador en el contenedor, dentro de la carpeta *schema*. Este archivo contiene la configuración inicial para la creación de la base de datos. Por otro lado, copiamos la base de datos contenida dentro de la carpeta */var/lib/mysql* del contenedor en la carpeta */data/db* de nuestro ordenador. Esto nos permite guardar la base de datos y los cambios que se realicen en ella en nuestro ordenador de modo que, si se borra el contenedor y se crea uno nuevo, no sea necesario volver a configurar la base de datos. Inicialmente esta carpeta está vacía, ya que tras crear por primera vez el contenedor es necesario crear también la base de datos. Finalmente, establecemos las variables de entorno necesarias, en nuestro caso el nombre de la base de datos que vamos a crear y la contraseña.

Con esto, ya tenemos configurados los servicios de los cuales consta nuestra aplicación. El siguiente paso es copiar el directorio *Pruebas_Aplicacion*, el cual contiene

las distintas carpetas y archivos necesarios, en el servidor. Para ello, previamente comprimimos esta carpeta en un archivo `.zip` ejecutando el siguiente comando:

```
zip -r -q Aplicacion.zip Pruebas_Aplicacion/
```

Una vez creado el archivo `Aplicacion.zip` lo copiamos al servidor. Para ello empleamos el siguiente comando:

```
Scp /home/carlos98/Escritorio/Aplicacion.zip  
csannun@chron2.tel.uva.es:/home/csannun/
```

A continuación, descomprimos el archivo, para lo cual ejecutamos el siguiente comando:

```
unzip -q Aplicacion.zip
```

Una vez descomprimido el archivo, nos movemos dentro del directorio `Pruebas_Aplicacion` ejecutando el comando `cd Pruebas_Aplicacion/`. Por último, ejecutamos el siguiente comando:

```
docker-compose up -d
```

Mediante la ejecución de este comando obtenemos las dos imágenes definidas en el archivo `docker-compose.yml`, las configuramos y las ejecutamos para crear los correspondientes contenedores.

Dado que es la primera vez que se ejecutan y se crean los contenedores, es necesario crear la base de datos. Para ello, primero abrimos una terminal asociada al contenedor de `mysql` con el comando `docker exec -it mysqlpdb bash`. Una vez dentro ejecutamos el comando `mysql -u root -p < /schema/gponServices.sql` (contraseña: `tfg_2017`). De esta manera, creamos la base de datos empleando para ello el archivo de configuración `gponServices.sql` localizado en el directorio `schema` dentro del contenedor. Una vez creada la base de datos salimos del contenedor ejecutando el comando `exit`.

El último paso es lanzar la aplicación de gestión de la red GPON. Para ello, abrimos una terminal asociada al contenedor con el comando `docker exec -it aplicacion_python bash`, nos movemos al directorio en el cual se encuentra el fichero principal del programa (`GlobalProgram.py`) mediante el comando `cd gestion/TFM/` y

ejecutamos la aplicación mediante el comando `python3 GlobalProgram.py`. El resultado es el que se puede visualizar en la Figura 72.

```
OpenSSH SSH client
Welcome to the service management system.
1. Service configuration
2. CLI Configuration
3. OpenFlow configuration
4. ONT Router configuration
5. Exit
What do you want to do?:
```

Figura 72: Menú principal de la aplicación de gestión de la red GPON

Para salir del contenedor simplemente ejecutamos el comando `exit`. Además del comando `docker-compose up -d`, algunos otros comandos relevantes para interactuar con los contenedores son los siguientes:

- `docker-compose stop`: detenemos los dos contenedores
- `docker-compose start`: inicializamos los dos contenedores
- `docker-compose down`: detenemos los dos contenedores y los eliminamos

Si se ejecuta este último comando, para volver a crear los contenedores simplemente tenemos que ejecutar de nuevo el comando `sudo docker-compose up -d`. No obstante, en este caso simplemente creamos los contenedores, ya que las imágenes ya se obtuvieron tras la primera ejecución. Además, en este caso no sería necesario volver a crear la base de datos, ya que esta la tenemos guardada en la carpeta `/data/db` dentro del directorio `Pruebas_Aplicacion`.

Para comprobar que la conexión de la aplicación de gestión con la base de datos es correcta, seleccionamos la opción `Service configuration` del menú principal de la aplicación y dentro de esta la opción `List of configured services`. tal y como se muestra en la Figura 73.

```

OpenSSH SSH client
You have chosen: Service configuration

1. List of configured services
2. Create new service
3. Modify service
4. Delete service
5. Go back
Please, select what do you want to do:
    
```

Figura 73: Opciones del menú *Service configuration*

Como resultado, obtendremos una tabla con todos los servicios actualmente configurados, tal y como se puede observar en la Figura 74.

```

OpenSSH SSH client
+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Guaranteed Downstream | Excess Downstream | Guaranteed Upstream | Excess Upstream | VLAN | VLAN Priority | Type of service |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 26 | 10240.0 Kbps          | 0.0 Kbps          | 10.0 Mbps          | 0.0 Mbps          | 806  | 0              | Video            |
| 27 | 20480.0 Kbps          | 0.0 Kbps          | 20.0 Mbps          | 0.0 Mbps          | 806  | 0              | Video            |
| 28 | 30720.0 Kbps          | 0.0 Kbps          | 30.0 Mbps          | 0.0 Mbps          | 806  | 0              | Video            |
| 29 | 40960.0 Kbps          | 0.0 Kbps          | 40.0 Mbps          | 0.0 Mbps          | 833  | 0              | Video            |
| 30 | 61440.0 Kbps          | 0.0 Kbps          | 60.0 Mbps          | 0.0 Mbps          | 833  | 0              | Video            |
| 31 | 81920.0 Kbps          | 0.0 Kbps          | 80.0 Mbps          | 0.0 Mbps          | 833  | 0              | Video            |
| 32 | 102400.0 Kbps         | 0.0 Kbps          | 100.0 Mbps         | 0.0 Mbps          | 833  | 0              | Internet         |
| 33 | 153600.0 Kbps         | 0.0 Kbps          | 150.0 Mbps         | 0.0 Mbps          | 833  | 0              | Internet         |
| 34 | 204800.0 Kbps         | 0.0 Kbps          | 200.0 Mbps         | 0.0 Mbps          | 833  | 0              | Internet         |
| 35 | 149952.0 Kbps         | 0.0 Kbps          | 150.0 Mbps         | 0.0 Mbps          | 806  | 0              | Internet         |
| 36 | 307200.0 Kbps         | 0.0 Kbps          | 300.0 Mbps         | 0.0 Mbps          | 833  | 0              | Internet         |
| 37 | 204800.0 Kbps         | 0.0 Kbps          | 200.0 Mbps         | 0.0 Mbps          | 806  | 0              | Video            |
| 38 | 30720.0 Kbps          | 0.0 Kbps          | 30.0 Mbps          | 0.0 Mbps          | 833  | 0              | Internet         |
| 39 | 10240.0 Kbps          | 0.0 Kbps          | 10.0 Mbps          | 0.0 Mbps          | 833  | 0              | Internet         |
| 40 | 10240.0 Kbps          | 0.0 Kbps          | 10.0 Mbps          | 0.0 Mbps          | 806  | 0              | Internet         |
| 41 | 51200.0 Kbps          | 0.0 Kbps          | 50.0 Mbps          | 0.0 Mbps          | 833  | 0              | Internet         |
| 42 | 950000.0 Kbps         | 0.0 Kbps          | 950.0 Mbps         | 0.0 Mbps          | 833  | 0              | Internet         |
+-----+-----+-----+-----+-----+-----+-----+-----+
Press enter to continue...
    
```

Figura 74: Tabla con los distintos servicios configurados en la base de datos

5.4 Actualización de la aplicación de gestión de la red GPON

En este apartado se van a realizar algunos cambios en ciertas funciones de la aplicación con el objetivo de mejorar su funcionamiento. Para ello, previamente se describirá el proceso de instalación y configuración de una nueva herramienta, denominada Gigoló.

5.4.1 Instalación de Gigoló en un ordenador con Linux

Antes de nada, para poder realizar cambios en el código de la aplicación de una manera sencilla y cómoda desde nuestro propio ordenador, vamos a instalar la herramienta Gigoló [32].

Gigoló constituye una interfaz para administrar fácilmente conexiones a sistemas de archivos locales y remotos usando GIO/GVfs. Esto nos permite conectar y/o montar rápidamente un sistema de archivos remoto y administrar sus marcadores. Además, proporciona acceso a los recursos remotos como las conexiones FTP o SFTP (SSH), SMB (Windows Sharing) o recursos especiales, como la Papelera (trash: //). El propio Gigoló proporciona una interfaz para administrar las conexiones a esos recursos, para crear marcadores y conectarse automáticamente a los marcadores.

El proceso de instalación de Gigoló es muy sencillo, dado que está disponible en los repositorios oficiales de las versiones mas recientes de Ubuntu y se puede instalar usando el Centro de Software de Ubuntu. No obstante, nosotros vamos a realizar la instalación de Gigoló en nuestro ordenador usando la terminal, para lo cual ejecutamos los siguientes comandos:

```
sudo apt-get update
```

```
sudo apt-get install -y gigolo
```

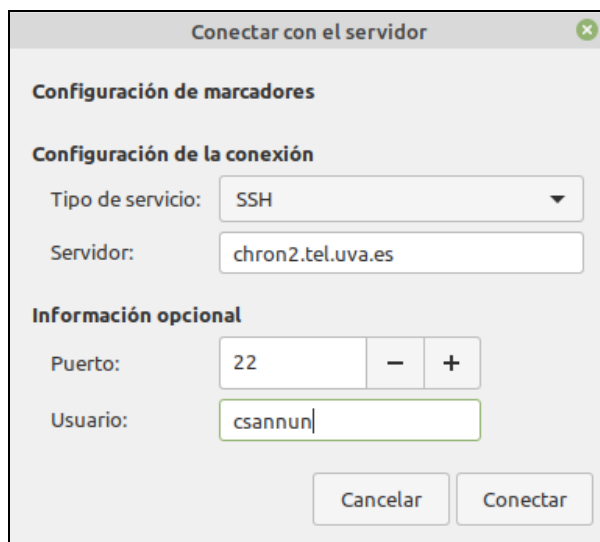
Primero actualizamos el repositorio y posteriormente instalamos el paquete correspondiente. Con esto ya tenemos instalado Gigoló en nuestro ordenador. Para inicializarlo ejecutamos por terminal el comando *gigolo*. Su ejecución se muestra en la Figura 75.



Figura 75: Interfaz del programa Gigoló

El siguiente paso es conectarnos al *chron2* para poder acceder al directorio *Pruebas_Aplicacion*. Para ello, pinchamos en el botón situado en la esquina superior

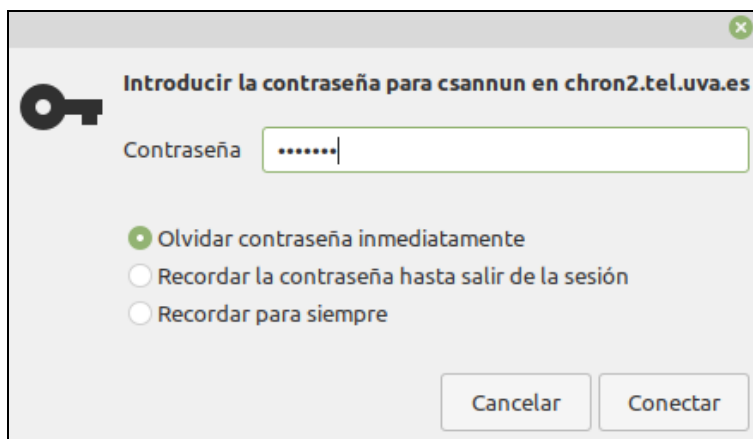
izquierda, el cual aparece rodeado en rojo en la Figura 75. Se abre entonces una ventana en la que tenemos que rellenar una serie de campos para establecer la conexión con el servidor (Figura 76).



The screenshot shows a dialog box titled "Conectar con el servidor". It is divided into three sections: "Configuración de marcadores", "Configuración de la conexión", and "Información opcional". In the "Configuración de la conexión" section, "Tipo de servicio" is set to "SSH" and "Servidor" is "chron2.tel.uva.es". In the "Información opcional" section, "Puerto" is "22" and "Usuario" is "csannun". There are minus and plus buttons next to the port field. At the bottom, there are "Cancelar" and "Conectar" buttons.

Figura 76: Configuración de la conexión con el *chron2*

Una vez rellenados estos campos, pinchamos en el botón *Conectar*. Se muestra entonces una nueva ventana en la cual tenemos que introducir nuestra contraseña para conectarnos con el servidor, tal y como se muestra en la Figura 77.



The screenshot shows a dialog box with a key icon and the title "Introducir la contraseña para csannun en chron2.tel.uva.es". It contains a "Contraseña" field with masked characters. Below the field are three radio button options: "Olvidar contraseña inmediatamente" (selected), "Recordar la contraseña hasta salir de la sesión", and "Recordar para siempre". At the bottom, there are "Cancelar" and "Conectar" buttons.

Figura 77: Ventana para introducir la contraseña

Finalmente, una vez introducida la contraseña, nos aparecerá una carpeta desde la cual podemos acceder a todos los archivos que tengamos en el *chron2*, tal y como se puede ver en la Figura 78.

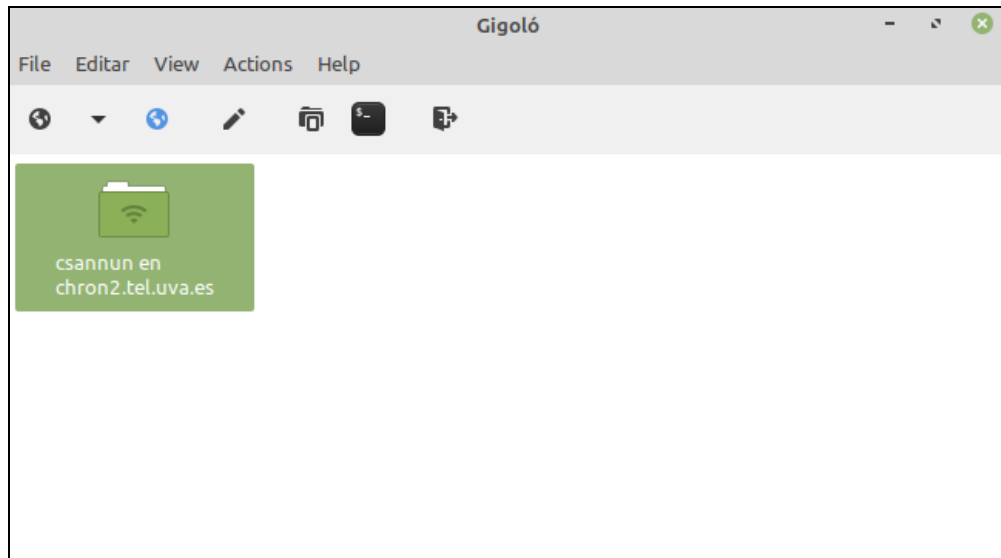


Figura 78: Interfaz principal del programa con la carpeta que nos permite acceder a nuestros archivos situados en *chron2*

En nuestro caso, únicamente tenemos el archivo *Aplicacion.zip* y el directorio *Pruebas_Aplicacion*, en el cual se encuentran todos los archivos y carpetas que hemos descrito en el apartado anterior. Esto lo podemos ver en la Figura 79.

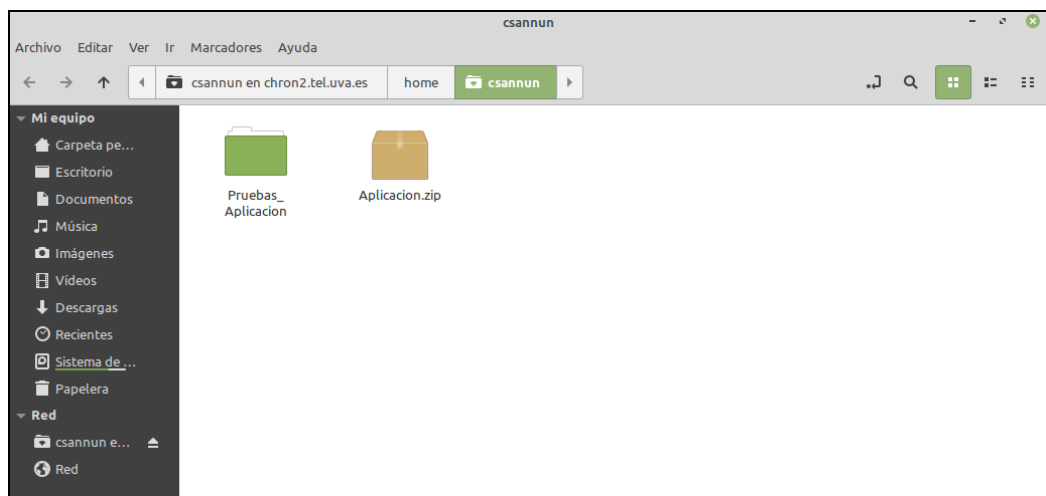
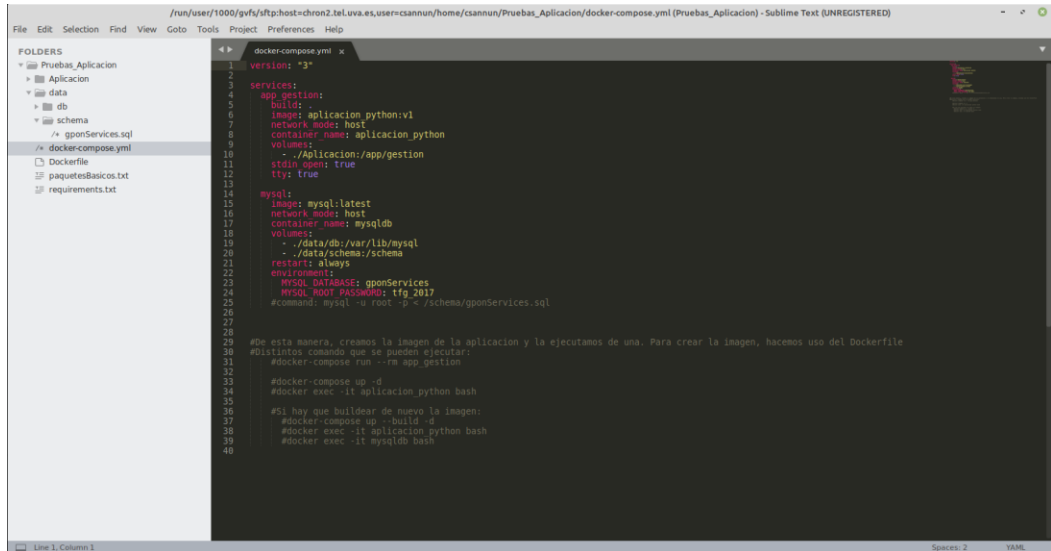


Figura 79: Contenido del directorio */home/csannun* del *chron2*

Una vez establecida la conexión con el servidor, abrimos el directorio *Pruebas_Aplicacion* con un editor de código. En nuestro caso, nosotros vamos a emplear *Sublime Text*, tal y como se puede observar en la Figura 80.

Figura 80: Editor de código *Sublime Text*

Una vez llegados a este punto, ya podemos llevar a cabo las modificaciones necesarias en el código para mejorar el funcionamiento de la aplicación. Los cambios que hagamos se aplicarán a los archivos que tenemos en *chron2*, dado que son estos los que estamos utilizando. Como anteriormente empleamos la opción *volumes* en el proceso de configuración de los servicios de la aplicación con Docker-Compose, los cambios efectuados en el código de la aplicación también tendrán efecto dentro del contenedor correspondiente. Esto hace que no sea necesario crear un nuevo contenedor cada vez que se lleva a cabo alguna modificación.

5.4.2 Descripción de las modificaciones realizadas en la aplicación de gestión de la red GPON

Los cambios realizados en el código de la aplicación de gestión de la red GPON tienen como objetivo mejorar el funcionamiento de las dos primeras opciones del menú principal de la aplicación, el cual se puede observar en la Figura 72.

La opción *Service configuration* del menú principal nos permite visualizar los servicios configurados, crear nuevos servicios, eliminarlos y modificar alguno de los servicios ya existentes, tal y como se puede ver en la Figura 73. La opción *Modify Service* permite modificar cualquiera de los servicios existentes, independientemente de si el servicio seleccionado está siendo empleado por alguna ONT. No obstante, esta configuración no es lo suficientemente robusta dado que si modificamos alguno de los servicios asignados a algunas de las ONTs de la red de acceso, el servicio se actualiza en

la base de datos, pero este no se actualiza en las ONTs. Para solventar este problema, vamos a modificar el código de la función `modifyService()` definida en el archivo `GlobalProgram.py`. En lugar de que la función primero actualice el servicio en la base de datos y posteriormente actualice dicho servicio en las ONTs, hacemos que la función previamente mire si el servicio está siendo empleado por alguna ONT. Si es así, no se podrá modificar el servicio, ya que primero debe eliminarse de las ONTs correspondientes. Para conseguir esto, añadimos las siguientes líneas de código que se muestran en la Figura 79 al comienzo de la función, justo después de solicitar al usuario cuál es el servicio que se desea modificar.

```
379 #Selecciono las onts de la base de datos que tienen configurado el servicio seleccionado
380 cursor.execute("select id_ont from ont_service where id_service='"+id_service+"'")
381 ont_ID=cursor.fetchall()
382 if ont_ID: #Si hay onts con el servicio que se ha seleccionado activo:
383     os.system('clear')
384     input("El servicio seleccionado no se puede modificar porque está siendo empleado \
385     por alguna ONT. Si desea modificarlo, haga un detach del servicio...")
386     return
```

Figura 81: Código para comprobar si el servicio seleccionado está siendo empleado por alguna ONT de la red de acceso

De este modo, primero vemos si el servicio seleccionado está siendo empleado por alguna ONT. Si es así, salimos de la función, ya que se le solicita al usuario que previamente elimine dicho servicio de las ONTs correspondientes. Si este servicio no está siendo usado por ninguna ONT, accedemos al menú de modificación del servicio y finalmente actualizamos el servicio en la base de datos.

La opción *CLI configuration* del menú principal de la aplicación (Figura 82) permite gestionar los servicios asignados a cada una de las ONTs de nivel 3 del laboratorio del mismo modo que TGMS, pero a través de CLI, empleando para ello comandos propios del estándar GPON. En la figura 82 podemos ver las opciones del menú *CLI configuration*.

```
C:\> OpenSSH SSH client
You have chosen: CLI Config

1. Attach service to ONT
2. Detach service from ONT
3. Go back
What do you want to do?:
```

Figura 82: Opciones del menú *CLI configuration*

No obstante, para poder configurar servicios en las ONTs de nivel 3 mediante CLI es necesario cambiar previamente el modo de gestión de la OLT. La OLT tiene dos modos de gestión: TGMS (por defecto) y CLI. Para cambiar el modo de gestión, hay que acceder a la OLT por CLI, para lo cual se emplea la interfaz que conecta el ordenador central del laboratorio con el puerto de gestión. No obstante, se ha activado un puerto en el *switch* del laboratorio que permite acceder a la OLT desde el *chron2*. Para ello, simplemente es necesario desconectar el cable Ethernet de color negro del puerto 5 del rack y conectarlo al puerto 7. Este cable es el que conecta con el puerto de gestión de la OLT. De este modo, podremos acceder a la OLT desde el *chron2*, para lo cual ejecutamos en una terminal el comando *telnet 172.26.128.38 4555*. Una vez se establezca la conexión, debemos introducir el usuario y la contraseña (*root/root*).

Una vez dentro, accedemos al menú “*system*”. Podemos visualizar el modo de gestión actual introduciendo el comando *showProvisionEngineMode*. El modo TGMS se corresponde con el valor *0*, mientras que el modo CLI se corresponde con el valor *1*. En nuestro caso, para activar el modo de gestión por CLI ejecutamos el comando *setProvisionEngineMode 1*. Por último, actualizamos el modo de gestión mediante el comando *applyProvisionEngineMode*. De esta manera, si ahora volvemos a ejecutar el comando *showProvisionEngineMode*, vemos que el modo de gestión es CLI. Todos estos pasos están recogidos en la Figura 83.

```
SmartOLT> system

SmartOLT/system> ShowProvisionEngineMode
Current provision mode is: TGMS

SmartOLT/system> SetProvisionEngineMode 1
Current provision mode is: CLI

SmartOLT/system> ApplyProvisionEngineMode
Current provision mode is save and applied

SmartOLT/system> ShowProvisionEngineMode
Current provision mode is: CLI

SmartOLT/system>
```

Figura 83: Ejecución de los comandos necesarios para conmutar al modo de gestión CLI

Una vez establecido el modo de gestión, ya se pueden configurar servicios en las ONTs de nivel 3 empleando para ello las opciones del menú *CLI configuración* (Figura 82). Dado que nosotros activamos al comienzo de este proyecto un nuevo puerto PON en

la OLT, el puerto PON 3, es necesario realizar una pequeña modificación en el código de la aplicación de modo que se puedan configurar correctamente los servicios en las ONTs de nivel 3. En concreto, es necesario modificar dentro de la función *servicio_Internet_Video()* definida en el archivo *clases.py* en qué puerto PON se debe ejecutar algoritmo DBA (en este caso pythagoras) para indicarle el nuevo puerto PON. El código actual se muestra en la Figura 84.

```
435 # Desde el menú de configuración del algoritmo DBA, se definirían los perfiles de
436 # ancho de banda en sentido Upstream
437 dba = "pon \n dba pythagoras 0 \n "
438 tn.write(dba.encode('ascii') + b"\n")
439 time.sleep(0.2)
```

Figura 84: Fragmento de código correspondiente al algoritmo DBA

Tal y como se observa, el puerto inicial establecido para la configuración del DBA es el puerto PON 0. Para establecer el puerto PON 3 simplemente es necesario cambiar el valor 0 por el valor 3. Con esto, ya se podrían configurar correctamente distintos servicios en las ONTs. No obstante, nosotros vamos a automatizar este proceso haciendo uso de la función *get_Puerto()* creada en anteriores TFGs, la cual devuelve el puerto activo de la OLT en una variable denominada *channel*. El código resultante tras la modificación se muestra en la Figura 85. De esta manera, se podrán configurar servicios en las ONTs de nivel 3 independientemente del puerto PON que esté activo en la OLT.

```
435 # Desde el menú de configuración del algoritmo DBA, se definirían los perfiles de
436 # ancho de banda en sentido Upstream
437 dba = "pon \n dba pythagoras "+str(channel)+" \n "
438 tn.write(dba.encode('ascii') + b"\n")
439 time.sleep(0.2)
```

Figura 85: Fragmento de código actualizado correspondiente al algoritmo DBA

Por último, relacionado con la gestión de servicios mediante CLI, cabe mencionar que la aplicación permite configurar de manera correcta únicamente el primer servicio que se asigne a una ONT. Si se añade un nuevo servicio existiendo ya uno, la aplicación deja de funcionar, no se llega a configurar este segundo servicio. Además, según el código, si ya hay un servicio configurado se elimina su configuración, por lo que cuando la aplicación deja de funcionar ni se ha agregado el nuevo servicio ni permanece el servicio anterior. También se ha comprobado que si se elimina el primer servicio asignado a una ONT, eliminamos la puerta de enlace del *router* de la ONT, de modo que esta pierde su dirección IP en la red de acceso. Esto hace que no sea posible configurar un nuevo servicio en la ONT.

Como conclusión, hemos detectado que hay problemas de conexión con el puerto de gestión de la OLT. En algunas ejecuciones de la aplicación no se consigue conectar con la OLT mediante los distintos *telnet* programados en las distintas funciones de la aplicación. Según parece, el *chron2* tarda un cierto tiempo en establecer esa conexión. El análisis profundo de este problema en la configuración de servicios mediante CLI se escapa al alcance de este TFG, por lo que su estudio se continuará en sucesivos trabajos.

5.5 Creación de *flows* y *meters* en ONOS empleando el lenguaje de programación Python

En este último apartado, se va a describir el proceso de creación de distintos *flows* y *meters* en ONOS empleando para ello el lenguaje de programación Python. El objetivo final es poder implementar esta configuración del controlador ONOS en la aplicación de gestión de la red GPON, de manera que se puedan gestionar los servicios asignados a las distintas ONTs de nivel 2 sin necesidad de hacer uso de la interfaz gráfica de ONOS. Actualmente, la aplicación permite la gestión de servicios mediante la creación de distintos *flows* y *meters* en ODL. No obstante, tras la actualización del escenario SDN-GPON, esta configuración se ha quedado obsoleta, puesto que el controlador desplegado en el servidor es ONOS.

Para la creación de los *flows* y los *meters* se van a crear distintos archivos *.py*, dado que es necesario configurar, por un lado, los *flows* por defecto asociados a los OVSs, por otro, los *meters* y, finalmente, los *flows* a los que se van a asociar estos *meters* para establecer el ancho de banda de la conexión.

Dado que la API REST de ONOS emplea la herramienta *cURL* para la creación de los *flows* y los *meters*, esto nos permite configurar el controlador SDN a través de peticiones HTTP empleando los métodos GET, POST y DELETE.

Para comprobar que los *flows* y los *meters* se crean correctamente, vamos a realizar una prueba *end-to-end* similar a la del Capítulo 4, por lo que emplearemos el ordenador antiguo del laboratorio. Dado que ahora, con la nueva configuración de la red del cliente, esta es independiente de la red de acceso, vamos a realizar las pruebas en el sentido *upstream*, es decir, desde el usuario final hacia el *chron2*. Por tanto, los *flows* que vamos a crear van a estar asociados al OVS desplegado en el ordenador del laboratorio.

No obstante, antes es necesario crear los *flows* por defecto para ambos OVSs. Para la creación de los *flows* por defecto asociados a los OVSs se ha desarrollado el código que se muestra en la Figura 86.

```
flows_defecto.py x
1 import requests
2 from requests.auth import HTTPBasicAuth
3 from requests.structures import CaseInsensitiveDict
4
5
6 headers = CaseInsensitiveDict()
7 headers["Content-type"] = "application/json"
8 headers["Accept"] = "application/json"
9
10 urlDownstream = "http://10.0.103.73:8181/onos/v1/flows/of%3A0000f4e9d4490272?appId=org.onosproject.fwd"
11
12 flowDown = '''{ \
13     "priority": 50000, \
14     "timeout": 0, \
15     "isPermanent": true, \
16     "deviceId": "of:0000f4e9d4490272", \
17     "treatment": { \
18         "instructions": [ \
19             { \
20                 "type": "OUTPUT", \
21                 "port": "NORMAL" \
22             } \
23         ] \
24     }, \
25     "selector": { \
26         "criteria": [ \
27             { \
28                 "type": "ETH_TYPE", \
29                 "ethType": "0x0800" \
30             } \
31         ] \
32     } \
33 }'''
34
35 resp=requests.post(urlDownstream,auth=HTTPBasicAuth('onos', 'rocks'),headers=headers,data=flowDown)
36 print(resp.status_code)
```

Figura 86: Código para la creación de los *flows* por defecto

Mediante este fragmento de código definimos las cabeceras, la dirección URL a la cual se va a realizar la petición HTTP y el *flow* que queremos crear. Finalmente, empleamos el método POST para llevar a cabo la creación del *flow* por defecto, en este caso correspondiente al OVS del *chron2*, dado que el valor del campo *deviceId* es el correspondiente a este. Para crear el *flow* por defecto correspondiente al OVS del lado del cliente simplemente es necesario modificar el valor del campo *deviceId* del *flow* y de la dirección URL (*of:000000805a4a90cc*).

El código que se ha programado para la creación de los *meters* y de los *flows* asociados a estos *meters* es similar al de la Figura 86, por lo que no se describirán los fragmentos de código correspondientes.

Para ejecutar este archivo *.py* abrimos una terminal, nos situamos en la carpeta donde se encuentra y ejecutamos el comando *python3 flows_defecto.py*. Para comprobar la correcta creación del *flow* ejecutamos el comando *ovs-ofctl -O OpenFlow13 dump-flows br0* dentro del contenedor del OVS del *chron2*. El resultado es el que se puede

observar en la Figura 87. Podemos ver que el *flow* que hemos configurado asociado al OVS del servidor, el que tiene una prioridad de 50000, se ha creado correctamente.

```
root@chron2:~# ovs-ofctl -O OpenFlow13 dump-flows br0
cookie=0x10000c88df467, duration=388.128s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
cookie=0x10000b270fce2, duration=388.119s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
cookie=0x100004664f076, duration=388.119s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
cookie=0x10000cb2dc5ff, duration=388.119s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
cookie=0x8a000035ed05d3, duration=18.223s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=50000,ip actions=NORMAL
```

Figura 87: Visualización del *flow* creado asociado al OVS del *chron2*

Para la creación de los *meters*, el código desarrollado es el que se muestra en la Figura 88.

```
flows_defecto.py x meters.py x
1 import requests
2 from requests.auth import HTTPBasicAuth
3 from requests.structures import CaseInsensitiveDict
4
5 headers = CaseInsensitiveDict()
6 headers["Content-type"] = "application/json"
7 headers["Accept"] = "application/json"
8
9 urlDownstream = "http://10.0.103.73:8181/onos/v1/meters/of%3A000000805a90cc"
10
11 meter = '{
12     "deviceId": "of:000000805a90cc", \
13     "unit": "KB PER SEC", \
14     "burst": true, \
15     "bands": [ \
16         { \
17             "type": "DROP", \
18             "rate": "50000", \
19             "burstSize": "0", \
20             "prec": "0" \
21         } \
22     ] \
23 }'
24
25 resp=requests.post(urlDownstream,auth=HTTPBasicAuth('onos', 'rocks'),headers=headers,data=meter)
26 print(resp.status_code)
```

Figura 88: Código para la creación de los *meters*

Igual que antes, para ejecutar este archivo *.py* abrimos una terminal, nos situamos en la carpeta donde se encuentra y ejecutamos el comando *python3 meters.py*. Para comprobar la correcta creación del *meter* ejecutamos el comando *ovs-ofctl -O OpenFlow13 dump-meters br0* dentro del contenedor del OVS asociado a la ONT de nivel 2. El resultado es el que se puede observar en la Figura 89.

```
root@tfglab7-System-Product-Name:~# ovs-ofctl -O OpenFlow13 dump-meters br0
OFPST_METER_CONFIG reply (OF1.3) (xid=0x2):
meter=1 kbps burst bands=
type=drop rate=50000 burst size=0
```

Figura 89: Visualización del *meter* creado asociado al OVS del cliente

Por último, para la creación de los *flows* con distintos *meters* asociados, el código desarrollado se muestra en la Figura 90.

```

1 import requests
2 from requests.auth import HTTPBasicAuth
3 from requests.structures import CaseInsensitiveDict
4
5
6 headers = CaseInsensitiveDict()
7 headers["Content-type"] = "application/json"
8 headers["Accept"] = "application/json"
9
10 urlDownstream = "http://10.0.103.73:8181/onos/v1/flows/of%3A000000805a4a90cc?appId=org.onosproject.fwd"
11
12 flowDown = '''{ \
13     "priority": 60000, \
14     "timeout": 0, \
15     "isPermanent": true, \
16     "deviceId": "of:000000805a4a90cc", \
17     "treatment": { \
18         "instructions": [ \
19             { \
20                 "type": "OUTPUT", \
21                 "port": "NORMAL" \
22             }, \
23             { \
24                 "type": "METER", \
25                 "meterId": 2 \
26             } \
27         ] \
28     }, \
29     "selector": { \
30         "criteria": [ \
31             { \
32                 "type": "ETH_TYPE", \
33                 "ethType": "0x0800" \
34             }, \
35             { \
36                 "type": "IP_PROTO", \
37                 "protocol": 6 \
38             }, \
39             { \
40                 "type": "IPV4_SRC", \
41                 "ip": "192.168.100.2/32" \
42             } \
43         ] \
44     } \
45 }'''
46
47 resp=requests.post(urlDownstream,auth=HTTPBasicAuth('onos', 'rocks'),headers=headers,data=flowDown)
48 print(resp.status_code)

```

Figura 90: Código para la creación de los *flows* con *meters*

Para ejecutar este archivo .py abrimos una terminal, nos situamos en la carpeta donde se encuentra y ejecutamos el comando `python3 flows.py`. Para comprobar la correcta creación del *flow* ejecutamos el comando `ovs-ofctl -O OpenFlow13 dump-flows br0` dentro del contenedor del OVS asociado a la ONT de nivel 2. El resultado es el que se puede observar en la Figura 91. Podemos ver que el *flow* que hemos configurado asociado al OVS del cliente, el que tiene una prioridad de 60000, se ha creado correctamente.

```

root@f9glab7-System-Product-Name:~# ovs-ofctl -O OpenFlow13 dump-flows br0
cookie=0x8a0000e316d35a, duration=42.032s, table=0, n_packets=111, n_bytes=19361, send_flow_rem priority=60000,tcp,nw_src=192.168.100.2 actions=meter:1,NORMA
L
cookie=0x1000068bfdbf9, duration=1013.440s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
cookie=0x1000041b30537, duration=1013.440s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,d1_type=0x8942 actions=CONTROLLER:65535,clear_actio
ns
cookie=0x100006d9f67b0, duration=1013.440s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,d1_type=0x88cc actions=CONTROLLER:65535,clear_actio
ns
cookie=0x1000060103c5a, duration=1013.439s, table=0, n_packets=73, n_bytes=9133, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
cookie=0x8a000059f23f21, duration=301.397s, table=0, n_packets=1299, n_bytes=856525, send_flow_rem priority=50000,ip actions=NORMAL

```

Figura 91: Visualización del *flow* creado asociado al OVS del cliente

Una vez visto el proceso de creación de los *flows* y los *meters* empleando Python, para realizar una prueba vamos a emplear el *meter* de 50 Mbps creado anteriormente (Figura 88). Para este *meter*, crearemos el *flow* correspondiente, similar al de la Figura 90. Para analizar el ancho de banda vamos a emplear la herramienta *iperf3*. Los pasos son los siguientes:

1. Ejecutamos en el *chron2* el comando *iperf3 -s*.
2. Ejecutamos en el portátil del cliente el comando *iperf3 -c IP_servidor*. En nuestro caso, el comando sería *iperf3 -c 192.168.100.1*.

```
administrador@administrador-ThinkPad-L380:~$ iperf3 -c 192.168.100.1
Connecting to host 192.168.100.1, port 5201
[ 4] local 192.168.5.2 port 34314 connected to 192.168.100.1 port 5201
[ ID] Interval          Transfer          Bandwidth         Retr  Cwnd
[ 4] 0.00-1.00 sec    12.3 MBytes      104 Mbits/sec    511  17.0 KBytes
[ 4] 1.00-2.00 sec    6.28 MBytes      52.6 Mbits/sec   377  11.3 KBytes
[ 4] 2.00-3.00 sec    5.84 MBytes      49.0 Mbits/sec   259  32.5 KBytes
[ 4] 3.00-4.00 sec    5.22 MBytes      43.8 Mbits/sec   470   2.83 KBytes
[ 4] 4.00-5.00 sec    5.90 MBytes      49.5 Mbits/sec   506   7.07 KBytes
[ 4] 5.00-6.00 sec    6.59 MBytes      55.3 Mbits/sec   488   4.24 KBytes
[ 4] 6.00-7.00 sec    5.10 MBytes      42.7 Mbits/sec   317  45.2 KBytes
[ 4] 7.00-8.00 sec    5.47 MBytes      45.9 Mbits/sec   233  25.5 KBytes
[ 4] 8.00-9.00 sec    5.90 MBytes      49.5 Mbits/sec   261  21.2 KBytes
[ 4] 9.00-10.00 sec   6.09 MBytes      51.1 Mbits/sec   495  29.7 KBytes
-----
[ ID] Interval          Transfer          Bandwidth         Retr
[ 4] 0.00-10.00 sec   64.7 MBytes      54.3 Mbits/sec   3917
[ 4] 0.00-10.00 sec   62.9 MBytes      52.7 Mbits/sec
```

Figura 92: Prueba de conectividad *end-to-end* con un flow de 50 Mbps empleando el ordenador antiguo

Tal y como se puede observar en la Figura 92, el ancho de banda de la conexión es de 52.7 Mbps, lo cual se corresponde con el *flow* creado de 50 Mbps. Por tanto, podemos concluir que la configuración de los *flows* y los *meters* en ONOS empleando los distintos archivos *.py* descritos anteriormente funciona correctamente.

5.6 Conclusiones

En este capítulo de la memoria se han descrito, de manera detallada, los pasos necesarios para desplegar la aplicación de gestión de la red GPON desarrollada en Python en varios contenedores docker en el servidor *chron2*. Para ello, se ha empleado una nueva herramienta denominada Docker-Compose. Esta herramienta permite definir y ejecutar aplicaciones que emplean múltiples contenedores docker. Para la configuración

de los distintos servicios de los que consta la aplicación se emplea un archivo YAML, de manera que todos estos servicios se pueden crear e inicializar ejecutando un único comando. Posteriormente, tras el despliegue de la aplicación en el servidor, se han descrito los cambios realizados en el código con el objetivo de mejorar el funcionamiento de la aplicación, empleando para ello la herramienta Gigoló. Gigoló permite conectar y/o montar rápidamente un sistema de archivos remoto en nuestro ordenador. De este modo, podemos modificar los archivos del servidor directamente desde nuestro ordenador. Por último, se ha descrito el proceso de creación de *flows* y *meters* en ONOS empleando el lenguaje de programación Python, cuyo objetivo es implementar esta configuración en la aplicación de gestión de la red GPON, de manera que se puedan controlar los servicios asignados a las distintas ONTs de nivel 2 sin necesidad de hacer uso de la interfaz gráfica de ONOS.

6

Conclusiones y líneas futuras

6.1 Conclusiones

En este Trabajo Fin de Grado se ha presentado, en primer lugar, el marco sobre el que se ha llevado a cabo la investigación, describiendo a grandes rasgos los principales fundamentos teóricos de las redes PON y de la tecnología SDN. A continuación, se han descrito las principales herramientas *software* empleadas para el despliegue SDN sobre la red GPON del laboratorio, tales como Docker, el protocolo OpenFlow, los *switches* virtuales OVS y el controlador ONOS. También se ha presentado el lenguaje de programación Python, usado para el desarrollo de la aplicación de gestión global de la red SDN-GPON. Por último, se ha descrito la metodología a seguir para la consecución de los objetivos previamente propuestos.

Por otro lado, se ha descrito el escenario SDN-GPON desplegado antes del comienzo de este trabajo, describiendo los principales elementos que lo componen e introduciendo su funcionamiento a grandes rasgos. A continuación, se ha llevado a cabo la activación del puerto PON 3 de la OLT, y se ha podido comprobar que su funcionamiento es correcto. Así mismo, se han descrito los pasos necesarios para la activación y configuración de una pasarela IoT, haciendo que esta se pueda integrar fácilmente en la red GPON conectándola directamente a una ONT de nivel 3. También se han realizado varias pruebas empleando distintos sensores conectados a la pasarela a través de una placa de Arduino usando la conexión USB. Finalmente, se ha llevado a cabo la activación del puerto 10G de la OLT, lo cual supone una importante actualización en el escenario SDN-GPON desplegado en el laboratorio hacia el uso de tecnologías de virtualización ampliamente empleadas en la actualidad. En concreto, en este TFG se ha trabajado con Docker, una plataforma de código abierto diseñada para desarrollar, implementar y ejecutar aplicaciones mediante el uso de contenedores.

Tras la activación del puerto 10G de la OLT, el ordenador central del laboratorio, el cual se encuentra situado justo antes de la OLT dentro de la topología de la GPON, se ha reemplazado por un servidor de la escuela situado fuera del laboratorio (*chron2*), de modo que dicho servidor se conecta a la fibra que va conectada directamente al puerto 10G de la OLT. Para configurar de manera correcta este nuevo escenario SDN-GPON se ha empleado Docker como herramienta de virtualización. Para ello, se ha llevado a cabo un proceso de creación y despliegue de un conjunto de contenedores en los que se van a ejecutar cada una de las herramientas necesarias en este nuevo escenario SDN-GPON. Estas herramientas son un *router*, un servidor DHCP, un OVS y un controlador SDN, en nuestro caso ONOS. Igualmente, se han comentado las actualizaciones SDN llevadas a cabo en el lado del cliente, lo cual incluye la instalación de nuevos OVS en contenedores docker (ordenadores conectados a las ONTs) y la separación de la red del usuario final de la red de acceso en dos subredes distintas, consiguiendo de esta manera un comportamiento similar al de una ONT de nivel 3. Por último, se han realizado distintas pruebas para analizar el rendimiento de todo este nuevo escenario SDN-GPON desplegado en el servidor *chron2* y se ha podido comprobar su correcto funcionamiento.

Por otro lado, se ha realizado el despliegue de la aplicación de gestión global de la red GPON desarrollada en Python en varios contenedores docker en el servidor *chron2*. Tras el despliegue de la aplicación en el servidor, se han descrito los cambios realizados en el código con el objetivo de mejorar su funcionamiento, empleando para ello la herramienta Gigoló, que permite conectar rápidamente un sistema de archivos remoto en nuestro ordenador.

Por último, se ha trabajado en la implementación de *flows* y *meters* en ONOS empleando el lenguaje de programación Python, con el objetivo de poder implementar esta configuración en la aplicación de gestión de la red GPON. En este sentido, cabe indicar que la aplicación origen tenía integrada esta implementación orientada al controlador ODL.

6.2 Líneas futuras

Tras la investigación desarrollada en este proyecto, se van a introducir algunas posibles vías de estudio que puedan mejorar o agregar funcionalidades a la red SDN-GPON del laboratorio.

Una de estas posibles vías de estudio es la activación simultánea de varios puertos PON y la implementación de nuevas funcionalidades, tales como protección mediante SDN. Con la activación del puerto 10G de la OLT, todo el tráfico de los 4 puertos PON es agregado en un único flujo 10G. Por tanto, sería interesante realizar la gestión de ONTs asociadas a distintos puertos PON de la OLT a través de este puerto 10G mediante tecnologías SDN, con el objetivo de analizar el comportamiento del nuevo escenario SDN-GPON que ha sido implementado en un servidor fuera del laboratorio.

Por otra parte, en busca de una mayor operatividad de la pasarela IoT integrada en la red, sería interesante llevar a cabo un estudio de las distintas tecnologías inalámbricas que esta permite, entre las que se incluye *ZigBee*. La pasarela IoT trae de fábrica un módulo interno *Xbee* que permite establecer comunicaciones inalámbricas empleando el estándar *GSM/GPRS* mediante el uso de una tarjeta SIM. No obstante, para el escenario presente en el laboratorio, sería más interesante emplear el protocolo *ZigBee*, de manera que sensores conectados en una o varias placas de Arduino (o similares) y situadas en diferentes ubicaciones se pudiesen comunicar con la pasarela de manera inalámbrica, lo que permite desplegar estos sensores en distintas zonas.

Por último, otra posible vía de estudio es la configuración del controlador ONOS en la aplicación de gestión global de la red GPON desplegada en el servidor. De esta manera se podrán gestionar los servicios asignados a las distintas ONTs de nivel 2 en el nuevo escenario SDN-GPON sin necesidad de hacer uso de la interfaz gráfica de ONOS. Para ello, se pueden emplear, los distintos fragmentos de código generados en este proyecto para la creación de *flows* y *meters* mediante Python.

7

Bibliografía

- [1] J.C. Ballesta y J. Boltimore «PASSIVE OPTICAL NETWORK (PON): FEATURES AND BENEFITS» [En línea]. Available: https://zenodo.org/record/2617267#.YJ-_WKgzaUk. [Último acceso: 15 Mayo 2021]
- [2] A. García Centeno, C. M. Rodríguez Vergel, C. Anías Calderón, F. C. Casmartiño Bondarenko «Controladores SDN, elementos para su selección y evaluación» [En línea]. Available: <http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/164/153>. [Último acceso: 15 Mayo 2021]
- [3] E. Haleplidis, Ed., K. Pentikousis, Ed., S. Denazis, J. Hadi Salim, D. Meyer, O. Koufopavlou «Software-Defined Networking (SDN): Layers and Architecture Terminology» [En línea]. Available: https://www.hjp.at/doc/rfc/rfc7426.html#sec_3 [Último acceso: 20 Mayo 2020]
- [4] Docker Documentation, «Get started with Docker» [En línea]. Available: <https://docs.docker.com/get-started/overview/> [Último acceso: 20 Mayo 2021]
- [5] Open Networking Foundation, «OpenFlow 1.3 Specification» 25 Junio 2012. [En línea]. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>. [Último acceso: 20 Mayo 2021]

- [6] «Open Network Operating System (ONOS)» [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/ONOS> [Último acceso: 20 Mayo 2021]
- [7] The Linux Foundation, «Open vSwitch» [En línea]. Available: <https://www.openvswitch.org/> [Último acceso: 20 Mayo 2021].
- [8] «Página web de Apache Karaf» [En línea]. Available: <https://karaf.apache.org/> [Último acceso: 20 Mayo 2021]
- [9] J. K. S. v. d. M. S. W. Andrei Bondkovskii, «Qualitative Comparison of Open-Source» Trinity College Dublin, 2016. [En línea]. Available: https://www.researchgate.net/publication/303564177_Qualitative_Comparison_of_Open-Source_SDN_Controllers [Último acceso: 20 Mayo 2021]
- [10] «Página web de Python» [En línea]. Available: <https://www.python.org/doc/essays/blurb/> [Último acceso: 20 Mayo 2021]
- [11] TELNET Redes Inteligentes, «TELNET-RI» [En línea]. Available: <http://www.telnet-ri.es> [Último acceso: 21 Mayo 2021]
- [12] «OLT SmartOLT 350» [En línea]. Available: <https://www.telnet-ri.es/olt-gpon-smartolt-350/> [Último acceso: 21 Mayo 2021]
- [13] «What is iPerf / iPerf3?» [En línea]. Available: <https://iperf.fr/> [Último acceso: 22 Mayo 2021]
- [14] «iperf - Linux man page» [En línea]. Available: <https://linux.die.net/man/1/iperf> [Último acceso: 22 Mayo 2021]
- [15] TELNET Redes Inteligentes, «Pasarela IoT BabelGate G5002» [En línea]. Available: <https://www.telnet-ri.es/productos/soluciones-iot/pasarela-iot-babelgate-g5002/> [Último acceso: 22 Mayo 2021]

- [16] Geek Factory, «DHT11 con Arduino, sensor de temperatura y humedad» [En línea]. Available: <https://www.geekfactory.mx/tutoriales/tutoriales-arduino/dht11-con-arduino/> [Último acceso: 22 Mayo 2021]
- [17] PromeTec, « BME280: presión, altitud, temperatura y humedad» [En línea]. Available: <https://www.prometec.net/bme280-presion-altitudtemperatura-y-humedad/>
- [18] Docker Documentation, «Install Docker Engine» [En línea]. Available: <https://docs.docker.com/engine/install/> [Último acceso: 22 Mayo 2021]
- [19] Docker Documentation, «Docker, Child commands» [En línea]. Available: <https://docs.docker.com/engine/reference/commandline/docker/> [Último acceso: 23 Mayo 2021]
- [20] The Linux Foundation, «Installing Open vSwitch» [En línea]. Available: <http://docs.openvswitch.org/en/latest/intro/install/> [Último acceso: 24 Mayo 2021]
- [21] Dokcer Hub, «Imagen oficial de ONOS» [En línea]. Available: <https://hub.docker.com/r/onosproject/onos> [Último acceso: 25 Mayo 2021]
- [22] ONOS – Wiki, «Single Instance Docker deployment» [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/Single+Instance+Docker+deployment> [Último acceso: 25 Mayo 2021]
- [23] ONOS – Wiki, «The ONOS Web GUI» [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/The+ONOS+Web+GUI> [Último acceso: 25 Mayo 2021]
- [24] ONOS – Wiki, «Appendix B: REST API» [En línea]. Available: <https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API> [Último acceso: 25 Mayo 2021]

- [25] «Instalar Docker y Docker Compose en Linux Mint 19» [En línea]. Available: <https://platzi.com/tutoriales/1432-docker/4574-instalar-docker-y-docker-compose-en-linux-mint-19/> [Último acceso: 26 Mayo 2021]
- [26] «Procesador Intel Core i7-9700» [En línea]. Available: <https://www.intel.es/content/www/es/es/products/processors/core/i7-processors/i7-9700.html> [Último acceso: 31 Mayo 2020]
- [27] «Z390 DESIGNARE Placas Base - GIGABYTE» [En línea]. Available: http://es.gigabyte.com/products/page/mb/z390_designare#kf [Último acceso: 31 Mayo 2020]
- [28] «TG-3468 Adaptador de red Gigabit PCI Express» [En línea]. Available: <https://www.tp-link.com/es/home-networking/adapter/tg-3468/> [Último acceso: 31 Mayo 2020]
- [29] Universidad de Valladolid, Repositorio Documental, «Configuración de la red residencial del usuario para la integración de SDN en redes de acceso PON» [En línea]. Available: <http://uvadoc.uva.es/handle/10324/42488> [Último acceso: 30 Mayo 2021]
- [30] Docker Documentation, «Overview of Docker Compose» [En línea]. Available: <https://docs.docker.com/compose/> [Último acceso: 27 Mayo 2021]
- [31] Docker Hub, «Imagen oficial de MySQL» [En línea]. Available: https://hub.docker.com/_/mysql [Último acceso: 27 Mayo 2021]
- [32] «Gigoló, un programa para configurar sistemas de archivos locales y remotos» [En línea]. Available: <https://ubunlog.com/gigolo-un-programa-para-configurar-sistemas-de-archivos-locales-y-remotos/> [Último acceso: 29 Mayo 2021]