



Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención Tecnologías de la Información)

**Interoperabilidad de dispositivos
en un tesbed de red de acceso
GPON mediante SDN y OpenFlow**

Autor:

D. Sergio Miguel López

Tutores:

Dr. D. Jesús M. Vegas Hernández

Dra. Dña. Noemí Merayo Álvarez

Resumen

El gran desarrollo y despliegue actual de las redes de fibra óptica en el segmento de acceso a través de tecnologías pasivas, redes PON (*Passive Optical Networks*), hace que exista una gran variedad de fabricantes de los dispositivos ópticos que componen este tipo de redes y que surjan problemas de incompatibilidad entre estos dispositivos a la hora de controlar su gestión y configuración de forma simultánea. Aunque estos dispositivos siguen los mismos estándares, cada fabricante añade sus propias implementaciones, haciendo que no haya dispositivos universales. Sin embargo, la tecnologías SDN (*Software Defined Networking*) resultan prometedoras en el campo de las redes ópticas de siguiente generación y en particular en las redes PON puesto que a través de protocolos comunes SDN se podrían gestionar de forma automática y simultánea redes de diferentes fabricantes, pues SDN propone la separación de la capa de control de la capa de datos. En este trabajo se tratará de analizar esta problemática en estas redes y de los protocolos y software utilizados en ellas para diseñar y elaborar un agente que permita salvar esas diferencias y permita la gestión y configuración SDN en este tipo de redes y que pueda ser compatible con cualquier red PON comercial.

Palabras clave

GPON (red óptica pasiva con capacidad de gigabit), SDN (redes definidas por software), OLT (terminación óptica de línea), ONU (unidad de red óptica), VOLTHA (abstracción hardware de OLTs virtuales), servicio de Internet, servicio de video, Python, ONOS, OpenFlow, Open vSwitch, Mininet, Docker.

Abstract

The great development and current deployment of fiber optic networks in the access segment through passive technologies, PON networks (Passive Optical Networks), means that there is a wide variety of manufacturers of the optical devices that form this type of network and also the incompatibility problems between these devices arise when controlling their management and configuration simultaneously. Although these devices follow the same standards, each manufacturer adds their own implementations, making there no universal devices. However, SDN (Software Defined Networking) technologies are promising in the field of next generation optical networks and in particular in PON networks since through common SDN protocols networks from different manufacturers could be managed automatically and simultaneously, since SDN proposes the separation of the control layer from the data layer. In this Final Project, we will try to analyze this problem in these networks and the protocols and software used in them to design and develop an agent, compatible with any PON network, which can solve these differences and allow SDN management and configuration in this type of network.

Keywords

GPON (Gigabit-capable Passive Optical Network), SDN (Software Defined Networking), OLT (Optical Line Termination), ONU (Optical Network Unit), VOLTHA (Virtual OLT Hardware Abstraction), internet service, video service, Python, ONOS, OpenFlow, Open vSwitch, Mininet, Docker.

Agradecimientos

Agradecer a mis dos tutores, Noemí y Jesús por su inestimable ayuda y dedicación, por estar siempre ahí y por brindarme la oportunidad de realizar este trabajo de investigación.

También quiero agradecer a Ignacio por echarme una mano con la instalación y puesta en marcha de VOLTHA.

La investigación desarrollada en este Trabajo Fin de Grado ha sido financiada por la Consejería de Educación de la Junta de Castilla y León en el marco del Proyecto ROBIN (VA085G19), 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) 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.

Tabla de contenidos

1	Introducción	11
1.1	Motivación.....	11
1.2	Objetivos.....	12
1.3	Estructura de la Memoria del TFG	13
2	Software, herramientas de trabajo y planificación del proyecto.....	14
2.1	Introducción.....	14
2.2	Introducción a las redes PON (Passive Optical Networks)	15
2.3	Introducción a SDN (Software Defined Networking)	16
2.4	Introducción a VOLTHA.....	17
2.4.1	VOLTHA BBSim	18
2.5	Introducción a OpenFlow	19
2.6	Controlador SDN ONOS	21
2.7	Introducción a Open vSwitch	22
2.8	Introducción a Mininet	24
2.9	Introducción al lenguaje de programación Python	24
2.10	Tecnología Docker.....	24
2.11	Entorno de trabajo.....	25
2.12	Planificación del proyecto	26
2.12.1	Presupuesto del proyecto	27
3	Despliegue y puesta en marcha de VOLTHA	29
3.1	Introducción.....	29
3.2	Componentes y funcionamiento de VOLTHA.....	30
3.3	Instalación y puesta en marcha de ONOS	33

3.4	Instalación y puesta en marcha de VOLTHA	34
3.5	Instalación y puesta en marcha de BBSim	38
3.6	Conexión de ONOS con Voltha y BBSIM.....	39
3.7	Versión actualizada de ONOS, VOLTHA y BBSim	43
4	Análisis de ONOS, Open vSwitch y Mininet	45
4.1	Introducción	45
4.2	Instalación de Mininet y Open vSwitch	47
4.3	Prueba 1: Mininet + ONOS con la aplicación fwd.....	47
4.4	Prueba 2: Mininet + ONOS sin aplicación fwd.....	49
4.5	Prueba 3: Creación de flows en ONOS	52
5	Desarrollo de un agente OpenFlow.....	55
5.1	Introducción	55
5.2	Análisis de la comunicación entre un switch OpenFlow y el controlador ONOS 55	
5.2.1	Mensaje OFTP_HELLO	58
5.2.2	Mensajes OFPT_FEATURES_REQUEST y OFPT_FEATURES_REPLY	59
5.2.3	Mensajes OFPT_MULTIPART_REQUEST y OFPT_MULTIPART_REPLY tipo OFPMP_PORT_DESC.....	60
5.2.4	Mensajes OFPT_GET_CONFIG_REQUEST y OFPT_GET_CONFIG_ REPLY	63
5.2.5	Mensajes OFPT_BARRIER_REQUEST y OFPT_BARRIER_REPLY.....	64
5.2.6	Mensajes OFPT_MULTIPART_REQUEST y OFPT_MULTIPART_REPLY tipo OFPMP_METER_FEATURES y OFPMP_DESC	64
5.2.7	Mensajes OFPT_ROLE_REQUEST y OFPT_ROLE_REPLY.....	66
5.2.8	Mensaje OFPT_FLOW_MOD	67
5.2.9	Mensaje OFPT_METER_MOD.....	71
5.2.10	Otros mensajes OFPT_MULTIPART.....	72
5.2.11	Mensajes OFPT_ECHO_REQUEST y OFPT_ECHO_REPLY	73

5.3	Diseño del Agente OpenFlow - CLI en la red GPON	73
5.3.1	Mejoras realizadas al agente	76
5.4	Implementación del Agente OpenFlow - CLI	77
5.4.1	Establecimiento de la comunicación entre controlador y agente.....	78
5.4.2	Funcionamiento principal del agente.....	79
5.4.3	Métodos de lectura de datos en el agente OpenFlow	85
5.4.4	Conexión del agente con la red GPON del laboratorio	85
5.5	Pruebas realizadas sobre el agente OpenFlow	90
5.5.1	Pruebas entre ONOS y el agente OpenFlow.....	91
5.5.2	Prueba extremo a extremo entre ONOS y la API GPON	93
6	Conclusiones y líneas futuras.....	99
6.1	Conclusiones.....	99
6.2	Líneas Futuras.....	99
7	Bibliografía	101
	Anexo I: Flows por defecto de ONOS.....	105
	Anexo II: Flows creados por fwd	107
	Anexo III: Script de creación de flows y meters de la prueba 3	110
	Anexo IV: Script de la primera prueba para el Agente OpenFlow .	113
	Anexo V: Script de la segunda prueba para el Agente OpenFlow ...	116

Lista de figuras

Figura 1: Diagrama con todas las herramientas que se desplegarán en este TFG	15
Figura 2: Arquitectura en árbol de una red PON	16
Figura 3: Arquitectura típica de una red SDN.....	17
Figura 4: Arquitectura de VOLTHA.....	18
Figura 5: Arquitectura de BBSim.....	18
Figura 6: Componentes principales de un switch OpenFlow.....	19
Figura 7: Recorrido del paquete a través del pipeline	20
Figura 8: Arquitectura por capas de ONOS	21
Figura 9: Subsistemas de ONOS.....	22
Figura 10: Arquitectura de Open vSwitch.....	23
Figura 11: Aspecto real de la red de acceso GPON desplegada en el laboratorio	26
Figura 12: Diagrama de Gantt de la planificación del proyecto	28
Figura 13: Conectividad entre los contenedores de VOLTHA vía Docker	30
Figura 14: Instalación de un flow en una OLT a través de VOLTHA.....	32
Figura 15: Conexión entre el adaptador y el agente OpenOLT	33
Figura 16: Cambio realizado en el archivo docker-compose-system-test.yml	37
Figura 17: Contenedores de VOLTHA levantados	38
Figura 18: Topología de la red ONOS, VOLTHA y BBSim.....	39
Figura 19: Menú de Aplicaciones de ONOS.....	40
Figura 20: CLI de VOLTHA con la OLT de BBSim habilitada.....	41
Figura 21: Vista en ONOS de la OLT de BBSim habilitada	42
Figura 22: Contenedores de la versión VOLTHA 2.2	44
Figura 23: Diagrama de Gantt de la planificación del proyecto modificado	46
Figura 24: Red creada en la prueba con Mininet y ONOS.....	49
Figura 25: Código de la red de la prueba 2	49

Figura 26: Red creada en la prueba 2	52
Figura 27: Visualización del meter a través de la API	54
Figura 28: Mensajes capturados de la comunicación entre el switch y el controlador.....	57
Figura 29: Inicio de la comunicación entre el switch y el controlador (mensajes OpenFlow)	58
Figura 30: Características y campos del mensaje OFPT_HELLO	58
Figura 31: Características y campos del mensaje del tipo OFPT_FEATURES_REQUEST	59
Figura 32: Características y campos del mensaje OFPT_FEATURES_REPLY	60
Figura 33: Tipos de mensajes OFPT_MULTIPART.....	61
Figura 34: Mensaje OFPT_MULTIPART_REQUEST tipo OFPMP_PORT_DESC.....	61
Figura 35: Características y campos principales del mensaje OFPT_MULTIPART_REPLY tipo OFPMP_PORT_DESC	62
Figura 36: Características y campos principales de un puerto	62
Figura 37: Características y campos principales del mensaje OFPT_GET_CONFIG_REPLY	64
Figura 38: Características y campos principales del mensaje OFPT_MULTIPART_REPLY del tipo OFPMP_METER_FEATURES	65
Figura 39: Características y campos principales del mensaje OFPT_MULTIPART_REPLY del tipo OFPMP_DESC	65
Figura 40: Campos y características del mensaje OFPT_ROLE_REQUEST	66
Figura 41: Campos y características del mensaje OFPT_ROLE_REPLY	66
Figura 42: Características y campos del mensaje OFPT_FLOW_MOD.....	67
Figura 43: Valores del campo OXM Field	70
Figura 44: Tipos de instrucciones.....	70
Figura 45: Tipos de acciones	71
Figura 46: Características y contenidos del mensaje OFPT_METER_MOD	72
Figura 47: Mensaje OFPT_MULTIPART_REPLY tipo OFPMP_METER	73
Figura 48: Diseño del agente OpenFlow para gestionar múltiples redes PON de diferentes fabricantes.....	74

Figura 49: Diagrama de clases utilizando el patrón de diseño Fachada.....	75
Figura 50: Estructura básica del agente OpenFlow.....	78
Figura 51: Diagrama con los campos de la estructura FlowStats	82
Figura 52: Diagrama con los campos de las estructura MeterConfig y MeterStats.....	83
Figura 53: Caso de un paquete con varios mensajes OpenFlow	84
Figura 54: Diagrama de flujo en la creación y borrado de servicios.....	90
Figura 55: Mensajes procesados por el agente OpenFlow	92
Figura 56: Meters que tiene el agente OpenFlow durante la realización de la prueba	92
Figura 57: Flows que tiene el agente OpenFlow durante la realización de la prueba.....	93
Figura 58: Diagrama con los componentes para la prueba de creación de servicios.....	94
Figura 59: Visualización en el agente OpenFlow de los servicios que se volcarán sobre la red GPON.....	95
Figura 60: Contenido del fichero de backup de servicios de Internet (a la izquierda) y de servicios de Video (a la derecha)	95
Figura 61: Contenido del fichero XML tras la creación de los servicios.....	96
Figura 62: Servicios borrados en la red GPON.....	98
Figura 63: Contenido del fichero XML tras el borrado de los servicios.....	98

1

Introducción

1.1 Motivación

Hoy en día existe un importante auge en el despliegue de la fibra óptica en el segmento de acceso a nivel mundial, más concretamente a través de tecnologías de acceso pasivas PON (*Passive Optical Networks*). De hecho, el nivel de penetración de las tecnologías de acceso por fibra óptica en España es cercano al 95% y en Europa esta cifra está creciendo de forma también significativa. Según el último estudio presentado por el FTTH en Europa [1], indican que en el año 2019 más de 172 millones de hogares estaban conectados a FTTH/B (*Fiber To The Home/Building*), que es aproximadamente el 38,2%. Pero las expectativas del FTTH para los siguientes años (2020-2025) [2], predicen que el número de hogares conectados en 2020 será en torno a los 189 millones y este valor se incrementará hasta los 263 millones de hogares en 2025. En este futuro escenario, las soluciones PON serán las predominantes a la hora de desplegar arquitecturas FTTHs, incrementándose este valor hasta el 73% en 2025. Sin embargo, existe una interoperabilidad limitada entre dispositivos PON de diferentes fabricantes a pesar de utilizar como base el mismo estándar y se hace necesario desarrollar técnicas para gestionar de forma transparente y automática estos dispositivos a partir de un lenguaje o estándar común. De este modo, analizar la interoperabilidad de los dispositivos ópticos usados en este tipo de redes, en concreto OLTs (*Optical Line Terminals*) y ONTs/ONUs (*Optical Network Terminals, Optical Network Units*), independientemente del fabricante cobra una gran importancia hoy en día para los operadores de red, y SDN (*Software Defined Networking*) se perfila como una solución factible para dar solución a esta problemática sobre todo debido al creciente auge de este tipo de tecnologías en las redes actuales.

En este sentido, SDN, que propugna la separación del plano de control del de datos [3], se está postulando como una tecnología prometedora para el diseño y la gestión de las redes de futura generación, puesto que el volumen de tráfico y la necesidad de redes seguras y manejables provocan que los protocolos y técnicas actuales resulten ineficientes y obsoletas. Por todas estas razones, este TFG se centra en analizar la interoperabilidad de dispositivos PON a partir de la integración de técnicas SDN usando

para su demostración un testbed GPON (*Gigabit PON*) de 25 kilómetros montado en uno de los laboratorios de investigación de la E.T.S.I. Telecomunicación.

En primer lugar se analizará el uso de VOLTHA [4] para dicha gestión SDN, un proyecto de código abierto a nivel europeo capaz de crear una abstracción hardware de estos equipos de red, y de este modo programar una API a partir de VOLTHA que pueda interactuar con redes GPON para su gestión y configuración. Pero también se usará el estándar SDN OpenFlow [5] para realizar dicho cometido y poder diseñar e implementar un agente opensource capaz de programar y configurar equipamiento PON (OLTs, ONTs/ONUs) y poder ser aplicado directamente en nuestro testbed GPON con la finalidad de poder llevar a cabo una prueba real.

1.2 Objetivos

Este Trabajo de Fin de Grado tiene una serie de objetivos, divididos en dos partes. Los primeros están relacionados con el trabajo de investigación realizado con VOLTHA:

- Análisis de las redes GPON y de las redes SDN.
- Instalación y puesta en marcha del controlador SDN ONOS [6].
- Instalación de VOLTHA en una máquina virtual.
- Puesta en marcha de todos los contenedores que forman VOLTHA.
- Conexión entre el controlador ONOS y una red PON virtual a través de VOLTHA.
- Análisis de la posible programación de una API a partir de VOLTHA que pueda interactuar con la red GPON del laboratorio.

En el Capítulo 3 de este trabajo se describirá cómo debido a un cierto nivel de complejidad introducido por VOLTHA y a una serie de requisitos que tienen que tener los equipos PON para ser gestionados con VOLTHA, el último objetivo no era viable cumplir, lo que conllevará un replanteamiento del trabajo hacia unos nuevos objetivos:

- Análisis del protocolo OpenFlow.
- Creación de redes virtuales mediante el programa Mininet [7] y el software Open vSwitch [8].
- Análisis de la estructura y los campos de los flows de OpenFlow.
- Análisis de los mensajes OpenFlow intercambiados entre un controlador SDN, ONOS en este caso, y un switch OpenFlow.
- Diseño y programación de un agente en Python [9] que mediante OpenFlow actúe como un switch para que pueda modificar la configuración de redes PON.

-
- Realización de pruebas para verificar el correcto funcionamiento del agente OpenFlow sobre el testbed GPON desplegado en un laboratorio.

1.3 Estructura de la Memoria del TFG

En el Capítulo 2 se describen todas las herramientas, software y programas utilizados durante todo el transcurso de realización del TFG.

En el Capítulo 3 nos centramos en VOLTHA, desde el análisis del software por dentro, hasta su instalación y puesta en marcha. También se describe el despliegue de una red virtual para conectar el controlador SDN ONOS con VOLTHA y una red GPON virtual.

En el Capítulo 4 utilizaremos de nuevo el controlador ONOS para controlar distintas redes de switches virtuales y analizar este tipo de conectividad y sus características principales. También se crearán diferentes tipos de flows mediante el protocolo OpenFlow para permitir la conectividad en redes mediante tecnología SDN.

En el Capítulo 5 se comenzará con un análisis de los mensajes OpenFlow intercambiados entre el controlador ONOS y un switch virtual. Después se diseñará un agente OpenFlow en Python que actúe como puente entre un controlador SDN y una red GPON, de modo que este tipo de redes puedan gestionarse de un modo automatizado mediante SDN.

La memoria finaliza en el Capítulo 6 con las conclusiones y las líneas de investigación futuras y el Capítulo 7 para las referencias utilizadas.

2

Software, herramientas de trabajo y planificación del proyecto

2.1 Introducción

En este capítulo se van a nombrar y a describir todas las herramientas y todo el software utilizado para la realización de este trabajo de investigación.

Estas herramientas serán necesarias para aplicar la tecnología SDN (*Software Defined Networking*) sobre redes, en este caso en el campo de las redes de acceso ópticas PON (*Passive Optical Networks*). Para aplicar SDN es necesario desplegar un elemento en común que gestione a toda la red de un modo centralizado bajo un mismo protocolo de comunicación, esto es, un controlador SDN que en nuestro caso será ONOS [6]. La primera red SDN que se analizará y desplegará estará formada por ONOS, VOLTHA y una red PON virtual llamada BBSim [10]. Después crearemos redes SDN más complejas con Mininet [7], un programa que nos permite crear redes de switches virtuales gestionadas por controladores SDN. Los switches que se crean con Mininet tienen como driver Open vSwitch [8] que es compatible con SDN, en concreto con OpenFlow [5]. Sin embargo, también se pueden crear redes controladas por SDN sin necesidad de Mininet utilizando directamente switches virtuales Open vSwitch.

Por otro lado, en este TFG también haremos uso de Python [9], y en concreto de la librería python-openflow de Kytos [11] para la programación de un agente desarrollado bajo el estándar SDN denominado OpenFlow, ampliamente utilizado para desarrollar arquitecturas SDN y en nuestro caso para gestionar una red GPON real desplegada. Por último se hará una introducción a Docker [12], una herramienta que permite crear contenedores para la virtualización de software. En nuestro caso concreto se usará Docker para desplegar VOLTHA y para crear hosts en los diferentes escenarios de red que se van a desplegar. En la Figura 1 se muestra un esquema con todas las herramientas y programas que se van a usar en este TFG.

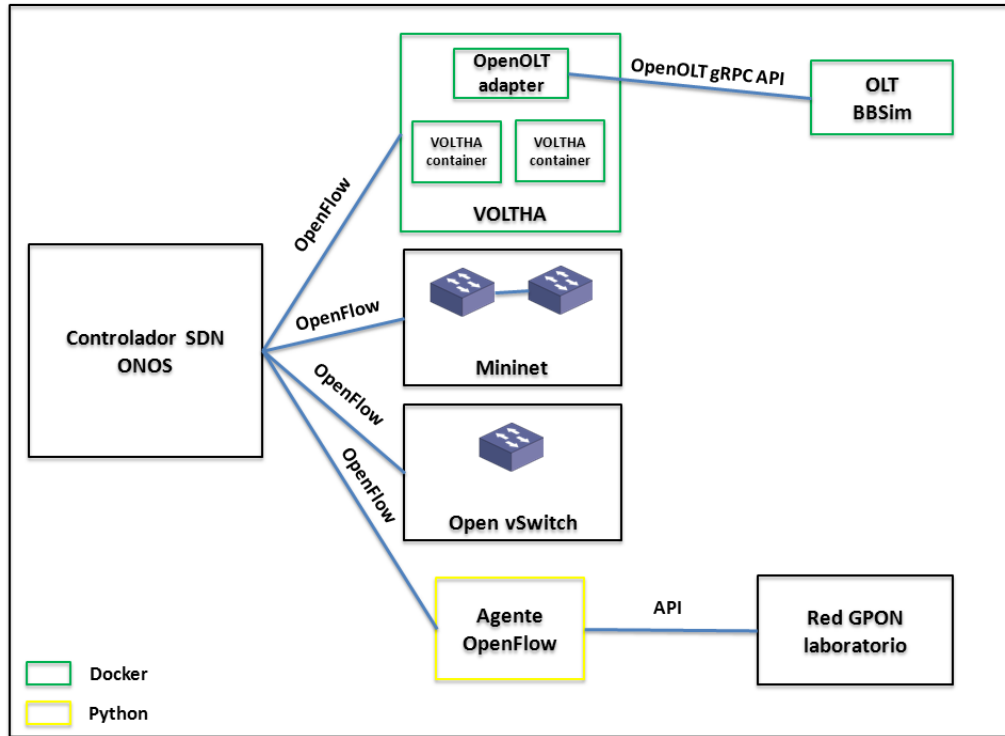


Figura 1: Diagrama con todas las herramientas que se desplegarán en este TFG

2.2 Introducción a las redes PON (Passive Optical Networks)

Una red PON es una red óptica pasiva (*Passive Optical Network*) y existen varios estándares PON tales como EPON (basado en Ethernet) y GPON (basado en Gigabit). El término red pasiva hace referencia a que no se necesitan dispositivos activos entre punto de origen de la red o el operador hasta el cliente o abonado. Este tipo de redes tienen una longitud física de entre 20 a 25 kilómetros de fibra óptica y son las redes de acceso más desplegadas a nivel mundial. Utilizan el protocolo G.984 publicado por la ITU-T (*International Telecommunication Union*) [13]. En este protocolo se define que el ancho de banda entre las OLTs y las ONTs/ONUs es de un máximo de 2,4 Gbps en el canal de bajada y de 1,2 Gbps en el canal de subida [14] [15].

Una red PON presenta una topología en árbol de muy bajo coste está formada principalmente por tres componentes, esto es, una OLT (*Optical Line Terminal*) o Unidad Óptica de Terminal de Línea que se encuentra en el nodo central y que da servicio a varias ONUs/ONTs (*Optical Network Unit/Terminal*) o Unidad de Red Óptica, que se encuentran localizadas en los domicilios de los clientes o abonados. El tercer componente es un divisor óptico pasivo, que permite la transmisión de los datos entre la OLT y las ONTs. En el canal descendente el tipo de conexión es punto-multipunto, donde la OLT envía los paquetes y el divisor los filtra y los transmite al usuario correcto utilizando TDM (*Time Division Multiplexing*). En el canal ascendente el tipo de conexión

es punto-punto, donde las ONTs envían los paquetes a la OLT usando TDMA (*Time Division Multiple Access*) [16]. En la Figura 2 podemos ver una representación de una red PON.

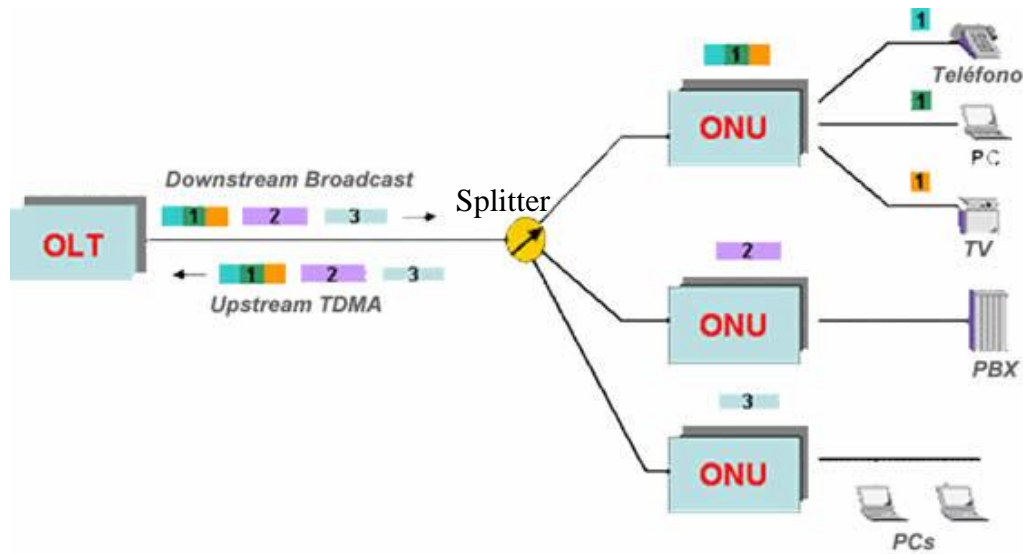


Figura 2: Arquitectura en árbol de una red PON

2.3 Introducción a SDN (Software Defined Networking)

Una vez que hemos definido que es una red GPON y cuáles son sus principales componentes, tenemos que hablar de SDN. La tecnología SDN (*Software Defined Networking*) es un enfoque de gestión de red que permite una configuración de red dinámica y programáticamente eficiente para mejorar su rendimiento y su administración.

SDN intenta centralizar la inteligencia de la red en un componente de red al disociar el proceso de reenvío de paquetes de red (plano de datos) del proceso de enrutamiento (plano de control). El plano de control está compuesto por uno o varios controladores, que son considerados el cerebro de la red. El controlador envía los datos, configura y administra los dispositivos de red a través de diferentes protocolos tales como OpenFlow. El plano de datos está formado por los dispositivos de red (routers, switches, access points, etc.), que deciden qué acción realizar con un paquete que llega a una de sus interfaces de entrada. Estos dispositivos son gestionados por uno o varios controladores SDN [3]. En la Figura 3 podemos ver un esquema de la arquitectura de SDN.

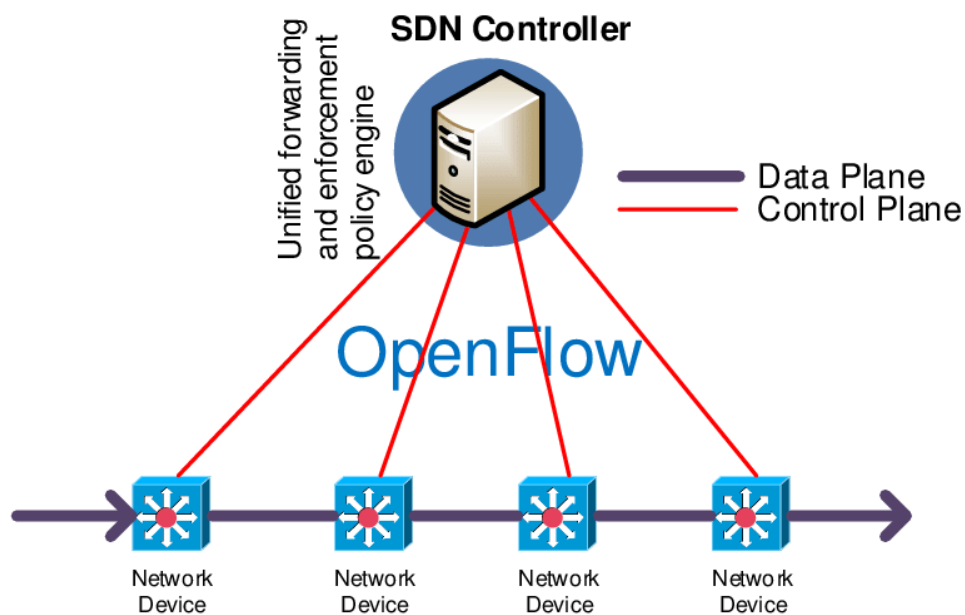


Figura 3: Arquitectura típica de una red SDN

2.4 Introducción a VOLTHA

VOLTHA (*Virtual OLT Hardware Abstraction*) es un proyecto de código abierto que permite crear una abstracción de hardware para equipos de acceso de banda ancha. Sigue el principio de múltiples proveedores. En la actualidad, VOLTHA proporciona un sistema de control y administración GPON común e independiente del proveedor para un conjunto de dispositivos hardware PON de caja blanca y dependientes del proveedor. Además, VOLTHA soportará en el futuro otras tecnologías de acceso como EPON, NG-PON2 y G.Fast [4].

VOLTHA se puede dividir en dos interfaces. En su interfaz con dirección norte, abstrae la red PON hasta hacerla parecer como un switch Ethernet programable a un controlador SDN. En su interfaz con dirección sur, VOLTHA se comunica con los dispositivos hardware PON usando los protocolos específicos del proveedor a través de adaptadores OLT y adaptadores ONU. La Figura 4 muestra un esquema con la arquitectura de VOLTHA, incluyendo el núcleo.

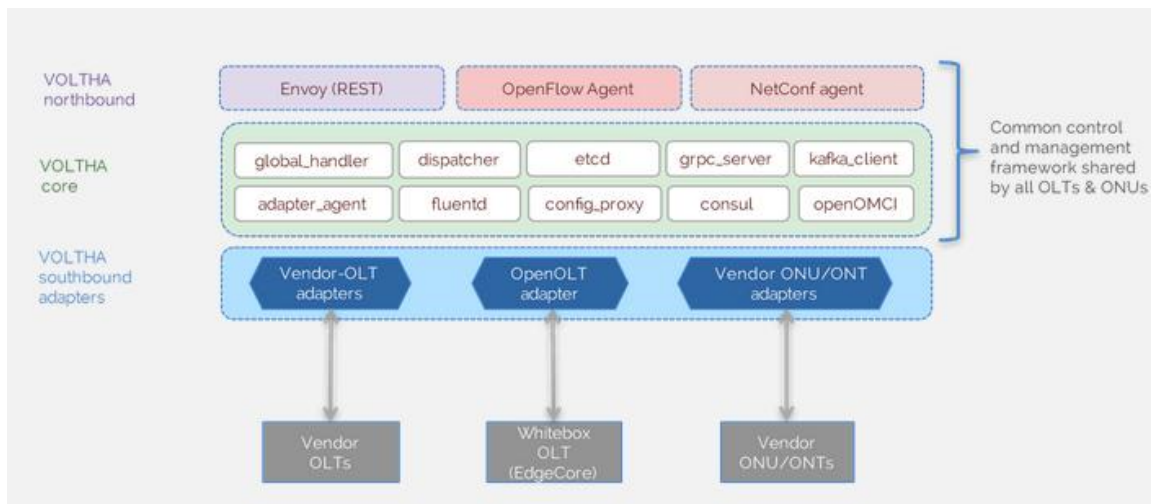


Figura 4: Arquitectura de VOLTHA

2.4.1 VOLTHA BBSim

BBSim (*Broadband Simulator*) es un emulador de software para el control de los mensajes de gestión (tales como OLTInd, DHCP, EAPOL, mensajes OpenOMCI, etc.) enviados desde la OLTs y las ONUs que están conectadas a VOLTHA y a ONOS a través del adaptador OpenOLT. BBSim puede ser usado para pruebas de escalabilidad de VOLTHA, DHCP L2 relay y aplicaciones AAA en ONOS [10]. Este simulador se implementa dentro de un contenedor Docker y actúa como si fuera un dispositivo OLT conectado a múltiples ONU/ONTS, emulando por lo tanto una red PON virtual completa. Por lo tanto, la arquitectura de BBSim es un contenedor Docker que tiene clientes OMCI/EAPOL/DHCP respondiendo dentro para emular los mensajes de control. Podemos ver en la Figura 5 un esquema de la arquitectura de BBSim.

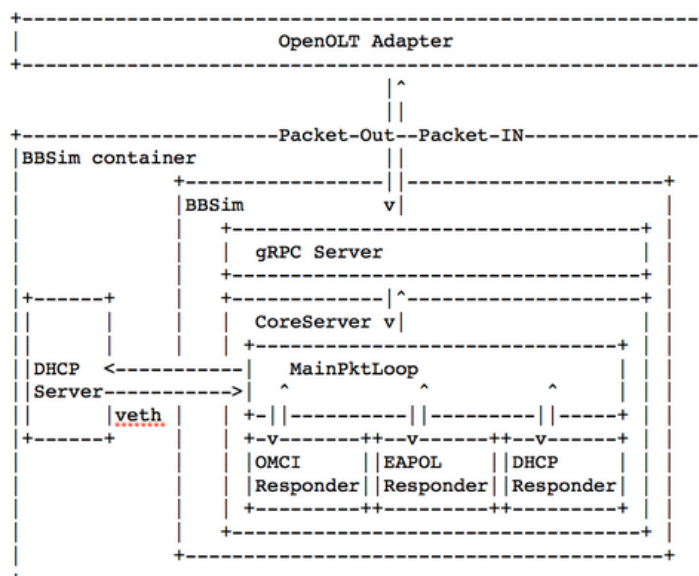


Figura 5: Arquitectura de BBSim

2.5 Introducción a OpenFlow

OpenFlow es un protocolo de comunicación que permite a los controladores de la red determinar el camino que seguirán los paquetes a través de la red de switches OpenFlow [5]. Un switch OpenFlow está compuesto por una o más tablas de flows y por una tabla de grupos, que se encargan de mirar los paquetes y de encaminarlos, y por un canal OpenFlow a un controlador externo. El switch OpenFlow se comunica con el controlador y el controlador gestiona el switch a través del protocolo OpenFlow. La Figura 6 muestra los componentes de un switch OpenFlow.

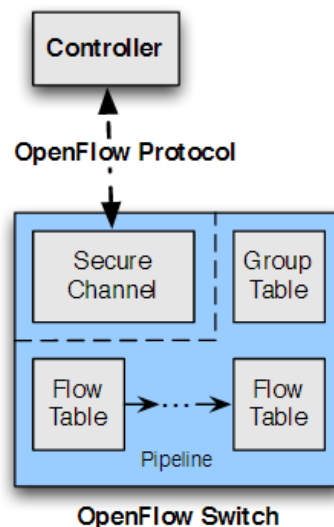


Figura 6: Componentes principales de un switch OpenFlow

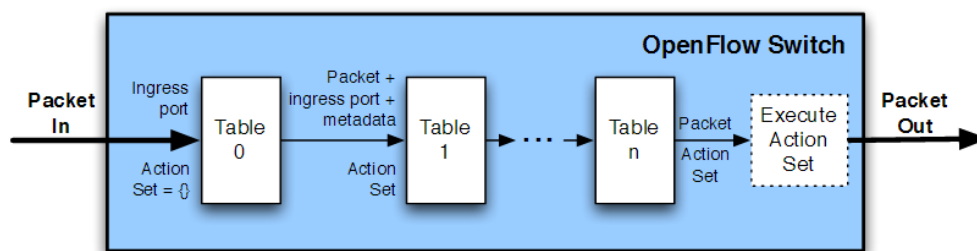
Un switch OpenFlow contiene una o varias tablas de flows y cada tabla contiene múltiples entradas de flows. La unidad de procesamiento pipeline de OpenFlow define como los paquetes interactúan con las tablas de flows. Este procesamiento lo podemos observar en la Figura 7. Las tablas de flows están numeradas empezando por el identificador 0. El procesamiento en el pipeline siempre empieza en la primera tabla de flows. Las demás tablas se usarán dependiendo de las condiciones que haya en la primera tabla.

Cuando llega un paquete, es procesado en la primera tabla de flows y se busca si concuerda (hace *Match*) con alguna de las entradas de flows. Si se encuentra un flow que se pueda aplicar a ese paquete, las instrucciones que lleve ese flow se ejecutan. alguna de esas instrucciones puede direccionar el paquete a otra siguiente tabla, donde se repetirá el proceso. Otras instrucciones pueden hacer que el paquete sea encaminado por un puerto específico. El procesamiento en el pipeline solo puede ir hacia delante, nunca hacia atrás, un paquete que haya llegado a ser procesado por la Tabla 2, no puede volver ni a la 1 ni a la 0. Si las instrucciones no direccionan el paquete a otra tabla, el procesamiento del pipeline se para en esa tabla. Si el procesamiento en el pipeline se

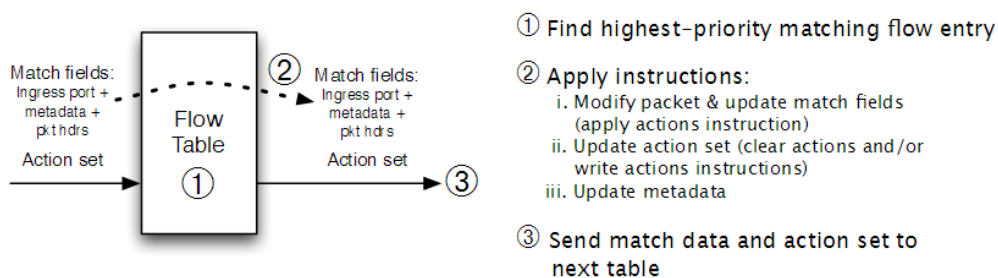
para, el paquete es procesado con sus acciones asociadas y probablemente será encaminado. En concreto, cada uno de los flows contiene los siguientes campos:

- *Match fields* (campos *Match*): condiciones que tendría que cumplir el paquete si quisiera que se ejecutasen las instrucciones de este flow.
- *Priority* (prioridad): prioridad del flow en la tabla. Se buscará hacer *Match* con los flows de mayor prioridad primero.
- *Counters* (contadores): contadores que se actualizan cuando los paquetes hacen *Match* con el flow.
- *Instructions* (instrucciones): modifican el conjunto de acciones o el procesamiento en el pipeline.
- *Timeouts*: tiempo máximo o tiempo de inactividad antes de que el flow sea eliminado de la tabla.
- *Cookie*: valor escogido por el controlador para filtrar estadísticas, modificar flows o identificar flows. No se usa a la hora de procesar paquetes.

Si un paquete no hace *Match* con ningún flow de la tabla, las tablas soportan un tipo especial de flow llamado *table-miss*, que se ejecuta en esos casos si está definido, y es la acción por defecto. Si no está definido, el paquete se descarta.



(a) Packets are matched against multiple tables in the pipeline



(b) Per-table packet processing

Figura 7: Recorrido del paquete a través del pipeline

Por último, el switch tiene el mecanismo independiente de control de expiración de los flows. Cada flow tiene dos tipos de *timeout*, *idle timeout* y *hard timeout*. Si cualquiera de los dos tipos tiene un valor distinto de 0, el switch debe anotar el momento en que llegó el flow a la tabla. Si el valor del *hard timeout* es distinto de 0, el flow será eliminado

después de los segundos que se indiquen en el valor, independientemente del número de paquetes con lo que haya hecho *Match* el flow. Si el valor del *idle timeout* es distinto de 0, el flow será eliminado de la tabla cuando no haya llegado ningún paquete que haga *Match* durante el número de segundos que tenga el *timeout*.

2.6 Controlador SDN ONOS

ONOS (*Open Network Operating System*) es el controlador SDN de código libre líder para construir soluciones de la siguiente generación de redes SDN [6]. ONOS es un controlador OpenFlow que dice al switch que tiene que hacer con los paquetes y los datos que llegan y que salen del switch mediante mensajes OpenFlow. Con un solo controlador se puede controlar desde un solo switch a toda una red completa de switches en tiempo real, por lo que por ejemplo se podrían cambiar las conexiones en la red para balancear el tráfico en cualquier momento.

ONOS está particionado en tres niveles principalmente: el primero para módulos orientados a la red con reconocimiento de protocolo que interactúan con la red. El segundo es el núcleo del sistema independiente del protocolo que rastrea y provee de información sobre el estado de la red. El tercero son las aplicaciones que consumen y actúan con la información que viene del núcleo. Entre ellos se forma una arquitectura por capas, donde los módulos interactúan con el núcleo a través de una API en dirección sur (proveedor), y el núcleo con las aplicaciones a través de una API en dirección norte (consumidor). La API de proveedor define unos medios sin protocolo para transmitir información de estado de la red al núcleo y para que el núcleo interactúe con la red a través de los módulos orientados a la red. La API de consumidor proporciona a las aplicaciones abstracciones que describen los componentes y propiedades de la red, de modo que puedan definir sus acciones deseadas en términos de política en lugar de mecanismo. En la Figura 8 podemos ver un esquema de estos tres niveles [17].

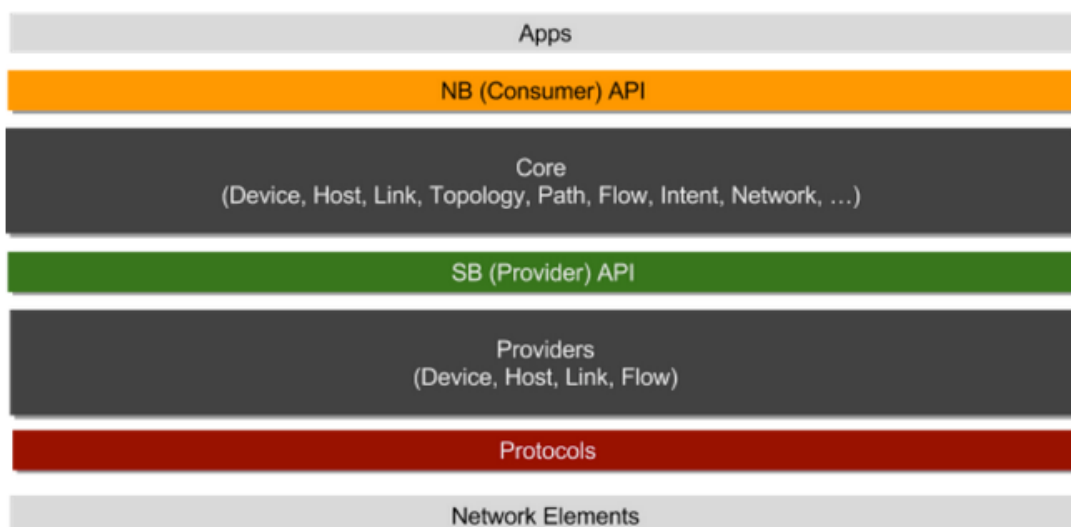


Figura 8: Arquitectura por capas de ONOS

Por último, ONOS define subsistema o servicio como una unidad de funcionalidad que se compone de múltiples componentes que crean un corte vertical a través de los niveles como una pila de software. ONOS define principalmente los siguientes subsistemas [18]:

- Subsistema de dispositivos: gestiona el inventario de los dispositivos de infraestructura.
- Subsistema de enlace: gestiona el inventario de los enlaces de infraestructura.
- Subsistema de host: gestiona el inventario de los hosts finales y de su localización en la red.
- Subsistema de topología: gestiona los *snapshots* de la topología de la red.
- Subsistema de rutas: encuentra y computa la ruta entre dispositivos o hosts usando el grafo de la topología más reciente.
- Subsistema de reglas de flows: gestiona el inventario de los flows que están en el dispositivo, así como sus campos *Match* y acciones. También proporciona métricas de los flows.
- Subsistema de paquetes: permite a las aplicaciones estar a la escucha de los paquetes que se reciben en la red y emitir paquetes por la red a través de uno o más dispositivos de red.

En la Figura 9 podemos ver algunos de los subsistemas de ONOS.

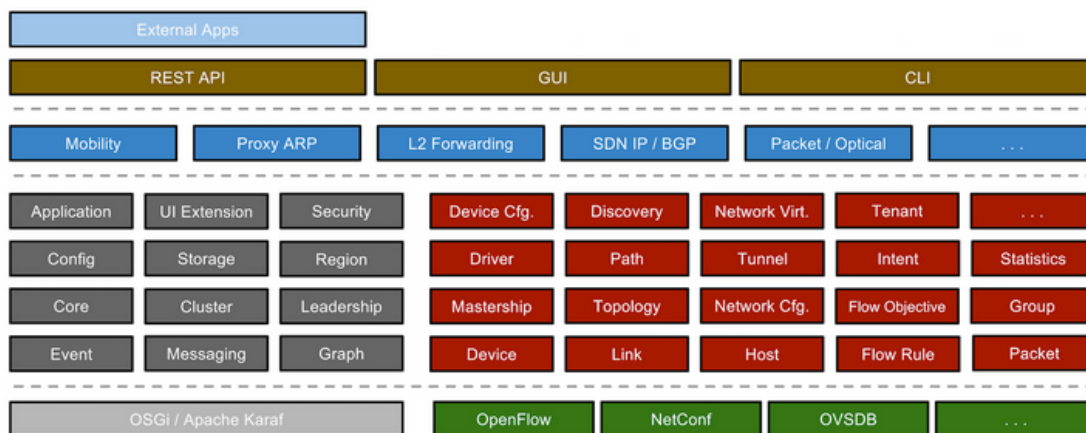


Figura 9: Subsistemas de ONOS

2.7 Introducción a Open vSwitch

Open vSwitch (OVS) es una implementación en código abierto de un switch de capa 2 o de un switch multicapa (switch de capa 3) virtual distribuido. Open vSwitch tiene como objetivo proporcionar una pila de conmutación para entornos de virtualización hardware mientras que soporta múltiples protocolos y estándares utilizados en las redes

informáticas. Está adaptado para funcionar como un switch virtual en entornos de máquinas virtuales y es compatible con el modelo estándar de VLAN 802.1Q, NetFlow, sFlow, OpenFlow y IPFIX, entre otros más [8] [19]. Los componentes principales de Open vSwitch son [20]:

- *OVS kernel module*: implementa los *datapaths* (switch) y sus *vports* (puertos del switch). Cada *datapath* tiene una tabla de flows, para procesar los paquetes. Si no existe ninguna regla en la tabla que se pueda aplicar al paquete, se debe pasar al *ovs-vswitchd*.
- *ovs-vswitchd*: componente central que se ejecuta en el espacio de usuario. Gestiona el *datapath* y procesa los mensajes OpenFlow.
- *ovsdb-server*: base de datos para mantener las tablas con los puertos, sFlows, interfaces, colas, etc.

Open vSwitch maneja dos tipos de flujos: los flujos OpenFlow en el espacio de usuario dentro de *ovs-vswitchd* y los flujos *datapath*, para los flujos instalados en el kernel. Para gestionar Open vSwitch se utilizan principalmente tres comandos:

- *ovs-dpctl* para gestionar el *datapath*.
- *ovs-ofctl* para gestionar los flujos OpenFlow.
- *ovs-vsctl* para gestionar la base de datos y los switches Open vSwitch.

En la Figura 10 se muestra la arquitectura de Open vSwitch.

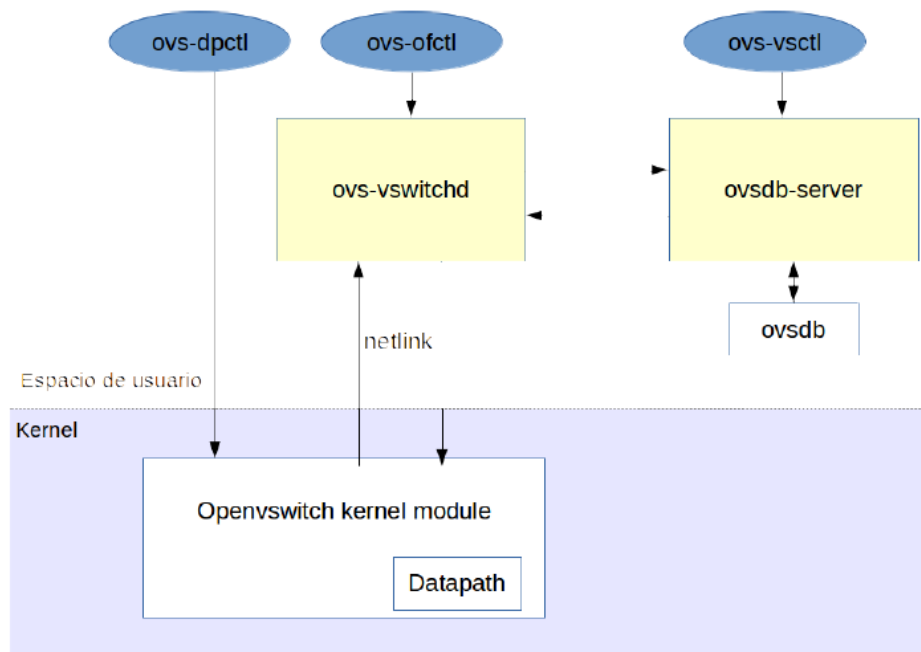


Figura 10: Arquitectura de Open vSwitch

2.8 Introducción a Mininet

Mininet es un programa que permite la creación de redes virtuales realistas, ejecutando un kernel real, switches y código de aplicaciones en una sola máquina, en tan solo unos segundos y con un solo comando. Ya que es muy fácil interactuar con la red usando la consola de comandos CLI (*Command Line Interface*) de Mininet, este programa es muy útil para experimentar con redes SDN o para experimentar con OpenFlow [7].

2.9 Introducción al lenguaje de programación Python

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general. Fue creado y lanzado al mercado en 1991 por Guido van Rossum, cuya filosofía de diseño enfatizaba la legibilidad del código. Es un lenguaje de programación multiparadigma, que además soporta orientación a objetos y programación imperativa, y combina un poder notable con una sintaxis muy clara [21].

En 2008 Python lanzó la versión 3 de Python, donde hubo una revisión importante del lenguaje e hizo que no fuese completamente compatible con las versiones anteriores de Python, por lo que no es posible ejecutar programas escritos en Python 3 sin que no haya que hacer algunas modificaciones al código.

2.10 Tecnología Docker

Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable de un entorno informático a otro. Una imagen de contenedor de Docker es un paquete de software de pequeño tamaño, independiente y ejecutable, que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones. Las imágenes se convierten en contenedores en tiempo de ejecución [22].

Las herramientas de contenedor, como Docker, ofrecen un modelo de implementación basado en imágenes. Esto permite compartir una aplicación, o un conjunto de servicios, con todas sus dependencias en varios entornos. Docker también automatiza la implementación de la aplicación (o conjuntos combinados de procesos que constituyen una aplicación) en este entorno de contenedores. Estas herramientas desarrolladas a partir de los contenedores de Linux, es lo que hace a Docker fácil de usar y único, y otorgan a los usuarios un acceso sin precedentes a las aplicaciones, así como la capacidad de implementación rápida y un control sobre las versiones y su distribución.

Los contenedores están independientes unos de los otros y cada uno de ellos tiene su propio software, librerías y ficheros de configuración. Los contenedores se pueden comunicar entre ellos a través de canales definidos. Todos los contenedores son

ejecutados por un solo sistema operativo y por tanto usan menos recursos que las máquinas virtuales.

2.11 Entorno de trabajo

Por último vamos a describir nuestro entorno de trabajo. En primer lugar, se decidió usar una máquina virtual proporcionada por la Escuela. La razón de esta elección es por varios motivos, entre ellos que no se disponía de un ordenador para soportar virtualización y que tuvieran los recursos necesarios para soportar la carga de trabajo que generan alguno de los programas que se han nombrado en este capítulo. Además, la escuela dispone de un entorno de virtualización de servidores llamado Proxmox VE, que permite la gestión de máquinas virtuales, las cuales pueden ser accedidas desde un navegador por Internet. Está característica es muy deseable, ya que podemos acceder en cualquier momento a la máquina desde cualquier localización. En cuanto a las características de la máquina, tiene un sistema operativo Linux Mint 19.2, 60 GB de memoria de disco y 12 GB de memoria RAM. Podemos acceder a ella desde Internet en la dirección *matrix.inf.uva.es*. De esta manera tenemos entorno gráfico. Si solo queremos la consola de comandos, nos podemos conectar por ssh a *virtual.lab.inf.uva.es* en el puerto 60021.

Por otro lado tenemos la red GPON del laboratorio y el ordenador donde se despliega el software necesario para gestionar esta red y sobre el que desplegaremos nuestro agente para que configure la red. La red de GPON desplegada en el laboratorio, tiene un OLT que da servicio a varias ONTs que tiene conectadas, por lo que su topología es en árbol típica de los estándares PON. La red GPON está conectada a un ordenador central de gestión, que es la puerta de enlace del laboratorio con la red externa de la facultad. Tiene conectado un cable Ethernet, que le conecta con el exterior y le proporciona conexión a Internet. Dicho ordenador, actúa en forma de router, para lo que tiene un archivo de configuración ejecutable, donde se encuentran las instrucciones necesarias para tal efecto, y además se definen varias VLANs. Avanzando hacia el interior de la red de acceso, nos encontramos con el OLT, que se conecta con el ordenador central, para dotar al OLT de la conexión a Internet pertinente. El otro cable que entra al OLT, se encuentra en una subred diferente que se utiliza para la gestión y configuración del OLT, dando acceso a TGMS (*TELNET GPON Management System*), que es software que cuenta una interfaz web suministrada por el fabricante y que permite la configuración de la red y de los servicios (Internet, vídeo y voz) de los clientes. El OLT tiene cuatro puertos de salida, con una capacidad máxima de 64 ONTs por puerto, de modo que se pueden conectar hasta 256 ONTs simultáneamente, dando servicio a otros tantos clientes. En el caso que nos ocupa, el puerto que está en servicio es el Puerto 1. De este puerto sale un cable de fibra óptica de 20 kilómetros de longitud total, divididos en una bobina de 10 kilómetros y otra de 5 kilómetros, que conecta con un divisor óptico 1:8, donde se divide la señal para dar servicio a las ONTs conectadas a través de una bobina de una longitud total de 5

kilómetros, compuesta por 48 fibras ópticas. En la Figura 11, se puede ver como es el aspecto real de la red GPON desplegada en el laboratorio.

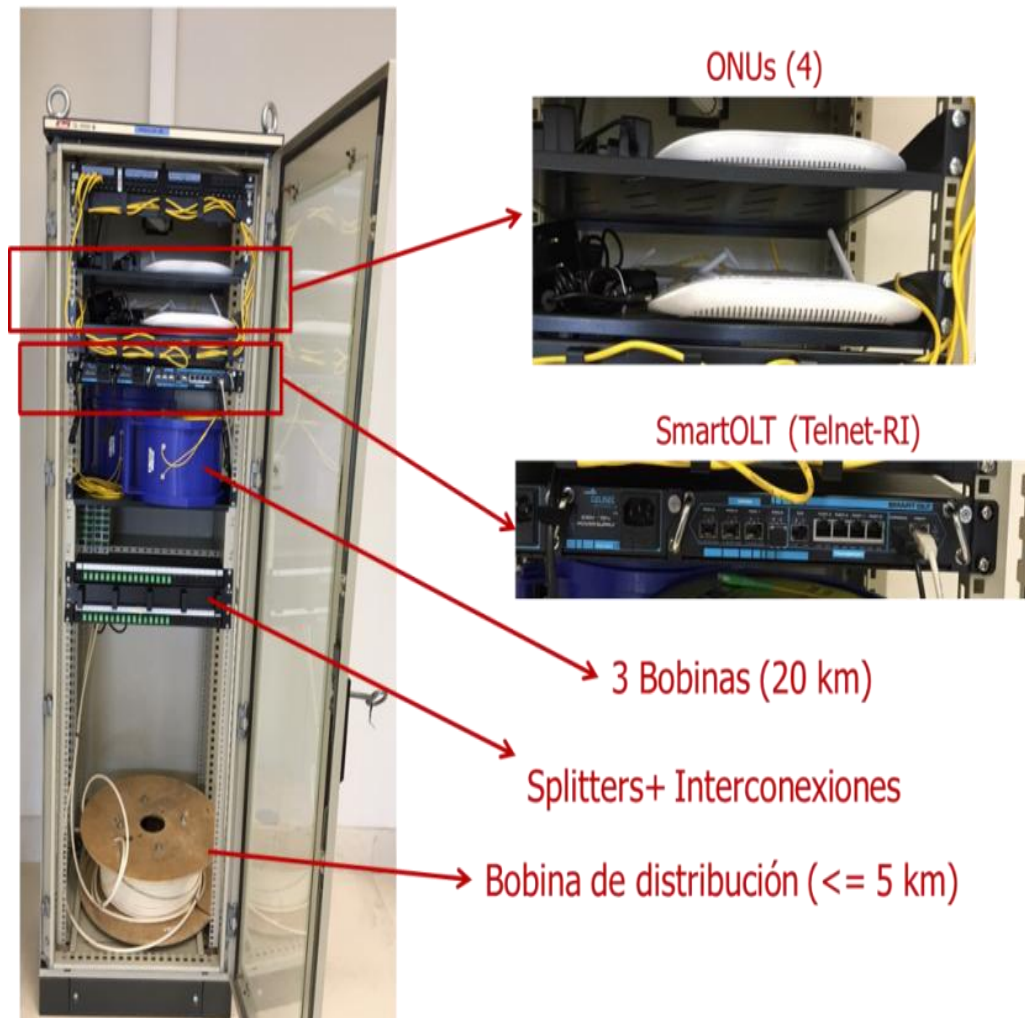


Figura 11: Aspecto real de la red de acceso GPON desplegada en el laboratorio

2.12 Planificación del proyecto

El proyecto se va a dividir en 5 fases: investigación, análisis, diseño, programación y pruebas.

El proyecto comenzará el día 17 de Febrero de 2020 y terminará el día 18 de Mayo del mismo año. La fase de investigación tendrá una duración de 20 días. Esta fase está dividida en dos tareas, investigación de las redes PON y las tecnologías SDN e investigación de los componentes de este tipo de redes y las tecnologías que se van a usar en el proyecto, cada una de las tareas de 10 días de duración. Esta fase es este capítulo.

La fase de análisis tendrá una duración de 12 días, siendo 5 de ellos para el análisis de la arquitectura de VOLTHA, 3 días para la instalación y puesta en marcha de VOLTHA, ONOS y 4 días para un análisis del agente y del adaptador OpenOLT.

En este punto tenemos un hito, ya que los resultados obtenidos de la fase de análisis nos permitirán saber si es viable el diseño, la programación y la realización de pruebas de una API a partir de VOLTHA que pueda interactuar con la red GPON del laboratorio. La fase de diseño tendrá una duración de 7 días y la de programación de 15 días. Por último, la fase de pruebas tendrá una duración de 12 días. En la Figura 12 podemos ver un diagrama de Gantt con la planificación del proyecto.

2.12.1 *Presupuesto del proyecto*

El proyecto puede ser desarrollado en su totalidad por un solo programador. Un programador contratado en una empresa gana de 13.000,00 a 20.000,00 euros al año si es principiante, y si tiene una gran experiencia puede llegar hasta 30.000,00 €. Como se ha estimado que la duración del proyecto serán 4 meses, si cogemos el sueldo intermedio, 20.000,00 €, el programador nos costaría 6.700 € aproximadamente [23]. En cuanto al entorno de trabajo, aunque nosotros utilizaremos una máquina virtual de la escuela, se considerará el coste que supondría alquilar una. Decidimos usar una máquina virtual Azure de Microsoft. El precio de una máquina virtual Linux con unas características similares a las que tenemos en la nuestra, es de 0,0448 €/hora por una máquina tipo E2a - E96a v4 de Azure con 4 núcleos, 100GiB de memoria de disco y 32GiB de memoria RAM [24]. Todas las herramientas y software nombradas en este capítulo son gratuitas, por lo que no aumentan el coste del proyecto. El coste total en 4 meses de duración del proyecto serían aproximadamente de 6.700 € de costes del programador, más aproximadamente 150 € de alquiler de la máquina virtual sería unos 6.850 € de coste total del proyecto.

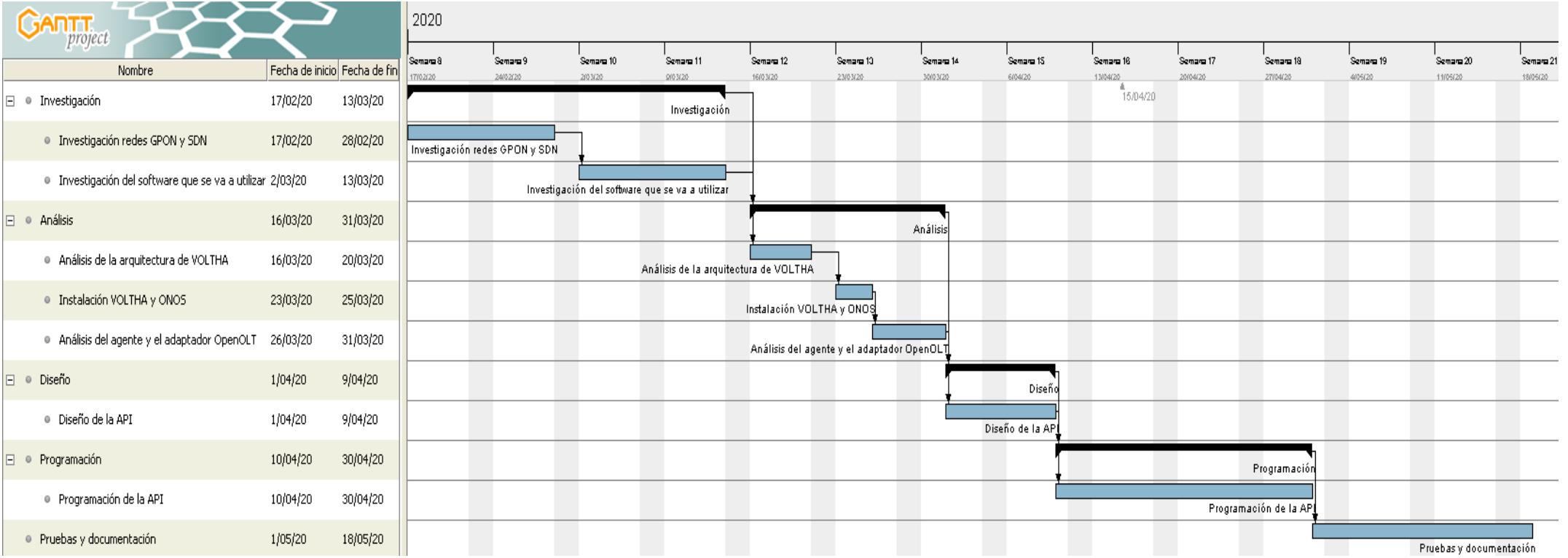


Figura 12: Diagrama de Gantt de la planificación del proyecto

3

Despliegue y puesta en marcha de VOLTHA

3.1 Introducción

El borde de la red (por ejemplo la oficina central para empresas de telecomunicaciones) es el lugar donde los operadores de red se conectan con sus clientes. CORD (*Central Office Re-architected as a Datacenter*) es un proyecto que intenta transformar esta parte de la red en una plataforma de prestación de servicios ágil permitiendo al operador ofrecer la mejor experiencia de usuario final junto con servicios innovadores de última generación. Esta plataforma utiliza SDN entre otras tecnologías para construir centros de datos ágiles en esta parte de la red. CORD ofrece una plataforma abierta, programable y ágil integrando múltiples proyectos de código abierto para que los operadores de red desarrollen servicios innovadores [25].

Uno de estos proyectos es VOLTHA, que permite crear una abstracción de hardware para equipamiento de redes PON. La razón por la que se ha elegido usar VOLTHA es porque cada tecnología utiliza sus propios protocolos y los proveedores utilizan sus propias interpretaciones de los mismos estándares. Además, estas diferencias se filtran en los sistemas OSS (*Operation Support Systems*) centralizados del proveedor de servicios, lo que provoca muchas ineficiencias [26]. En concreto, VOLTHA proporciona un sistema de control y gestión común de la tecnología GPON, independiente del proveedor, tanto para un conjunto de dispositivos hardware PON de caja blanca y como para otros específicos de diferentes fabricantes. Sin embargo, VOLTHA también admitirá el control de dispositivos de otras tecnologías de acceso como EPON, NG-PON2 y G.Fast [4].

En este capítulo comienza la fase de análisis e instalación de VOLTHA. Primero se va a explicar cómo funciona internamente VOLTHA cuando se instala un nuevo flow por parte del controlador mediante el protocolo OpenFlow en una OLT de una red GPON utilizando. Después se detallarán los pasos para instalar y poner en marcha el controlador ONOS, VOLTHA y red PON virtual.

3.2 Componentes y funcionamiento de VOLTHA

Al poner en marcha VOLTHA se despliegan los 15 contenedores Docker de los que está formado. Todos los contenedores tienen una sola interfaz de red excepto el contenedor principal, VOLTHA Core, que tiene dos interfaces de red. Docker crea 2 puentes Linux al desplegar los contenedores, uno para la conectividad entre contenedores y otro usado por los adaptadores de VOLTHA para hablar con las OLTs (solo VOLTHA Core) [27]. La Figura 13 muestra un esquema.

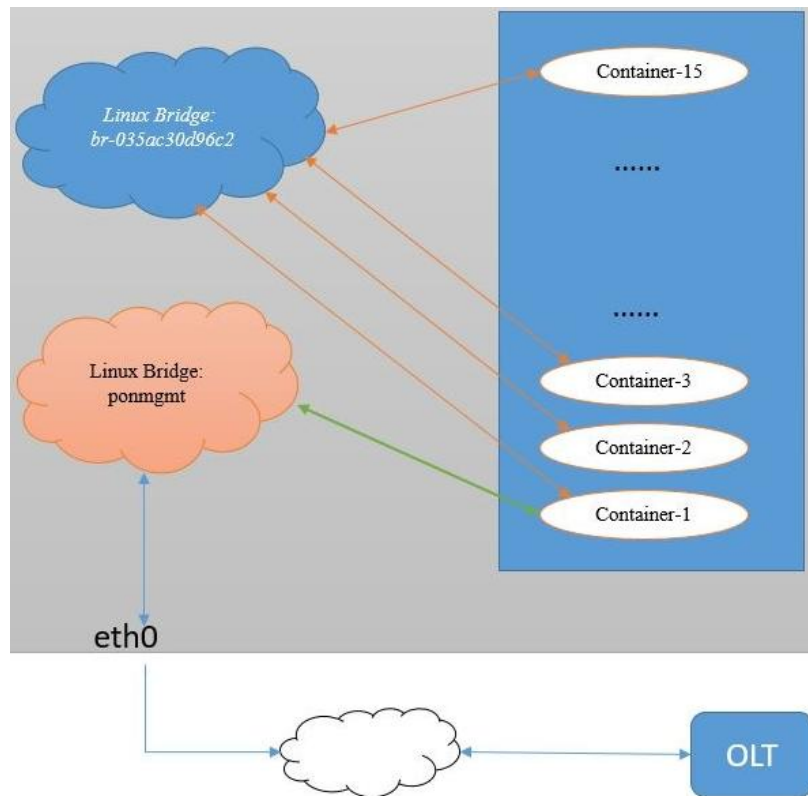


Figura 13: Conectividad entre los contenedores de VOLTHA vía Docker

Los contenedores que instala VOLTHA son los siguientes:

- *Compose_voltha_1*: VOLTHA Core, contenedor principal. Todos los adaptadores son parte de este contenedor.
- *Compose_nginx_1*: contenedor NGINX (servidor web/proxy inverso ligero de alto rendimiento).
- *Compose_chameleon_1/Compose_envoy_1*: servicio REST.
- *Compose_ofagent_1*: agente OpenFlow.
- *Compose_netconf_1*: agente Netconf.
- *Compose_consul_1*: almacenamiento clave-valor.
- *Compose_registrator_1*: servicio de registro.

-
- *Compose_kafka_1*: bus de mensajes.
 - *Compose_grafana_1*: visualización de métricas.
 - *Compose_fluent_1*: recolección de datos.
 - *Compose_zookeeper_1*: coordinación de procesos distribuidos.
 - *Compose_cli_1*: CLI (*Command Line Interface*).
 - *Compose_dashd_1*: demonio de la consola *dashboard*.
 - *Compose_shovel_1*: *gateway* entre *grafana* y *graphite*
 - *Compose_portainer_1*: Docker GUI.

El funcionamiento de VOLTHA cuando se instala un nuevo flow en una OLT u ONU/ONT empieza con el controlador SDN ONOS enviando el flow en un mensaje OpenFlow, tal y como se muestra la Figura 14. VOLTHA oculta los dispositivos físicos a ONOS y los disfraza como switches lógicos OpenFlow. El contenedor OF Agent (*Compose_ofagent_1*) coge el mensaje y se lo envía por gRPC al *LocalHandler*. gRPC es un moderno *framework* RPC (*Remote Procedure Calls*) de alto rendimiento y código abierto que puede ejecutarse en cualquier entorno. Conecta eficientemente servicios en centros de datos y entre ellos con soporte para balanceo de carga, rastreo, verificación de estado y autenticación. También se aplica para conectar dispositivos, aplicaciones móviles y navegadores a servicios de *back-end* [28].

El *LocalHandler* manda una instrucción al *LogicalDeviceAgent* para que actualice la tabla de flows. Descompone el flow que era enviado a un switch lógico y lo convierte para que sea compatible con un switch físico. Estos flows descompuestos se guardan en el almacén clave-valor (*Compose_consul_1*). A continuación, se realiza un *callback* para invocar al *DeviceAgent* de los dispositivos físicos que correspondan. Este agente llama al método preparado en el adaptador de cada dispositivo para instalar un nuevo flow a través del *AdapterAgent*. Por último, el adaptador OLT (*OLT Adapter*) envía el flow a la OLT o a la ONU/ONT [29].

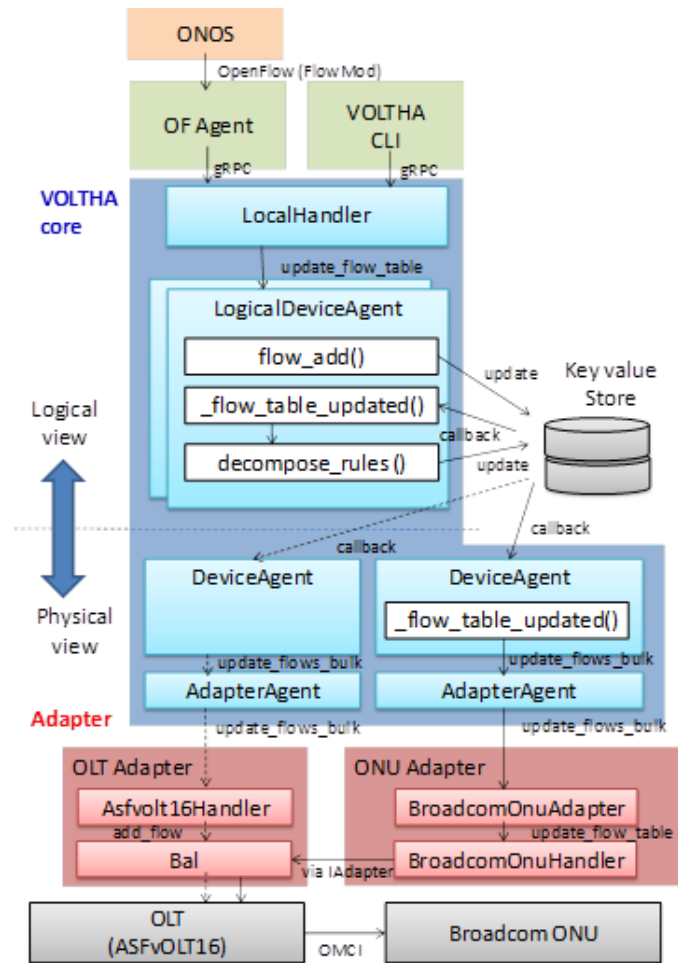


Figura 14: Instalación de un flow en una OLT a través de VOLTHA

Por lo tanto, en su lado sur, VOLTHA se comunica con dispositivos de hardware PON utilizando protocolos específicos del proveedor a través de adaptadores OLT y ONU, tal y como lo acabamos de ver. En este sentido, VOLTHA puede comunicarse con dichos dispositivos usando los adaptadores denominados OpenOLT. Para llevar a cabo esta comunicación, en la OLT debe existir un agente OpenOLT, que corre en las OLTs y da un servicio de administración y una interfaz de control a las OLTs basado en el protocolo gRPC. El agente OpenOLT es usado por VOLTHA a través del adaptador OpenOLT que se encuentra en el Core de VOLTHA. Por otro lado, el agente OpenOLT usa el software de Broadcom BAL (*Broadband Adaptation Layer*) para conectarse con la interfaz de los chips SoC FPGA (*System on Chip Field Programmable Gate Array*) en las OLTs para conectarse a sus APIs específicas y que dependen de cada fabricante de equipamiento PON [30]. En la Figura 15 podemos ver la conexión entre VOLTHA y la OLT. De este modo, un controlador SDN (ONOS) podrá gestionar y configurar equipamiento PON de diferentes fabricantes de forma centralizada a través de la plataforma VOLTHA usando el adaptador OpenOLT en el Core de VOLTHA y conectándose a los agentes OpenOLT desarrollados dentro de OLTs/ONTs y que interactúan con las APIs específicas de cada vendedor.

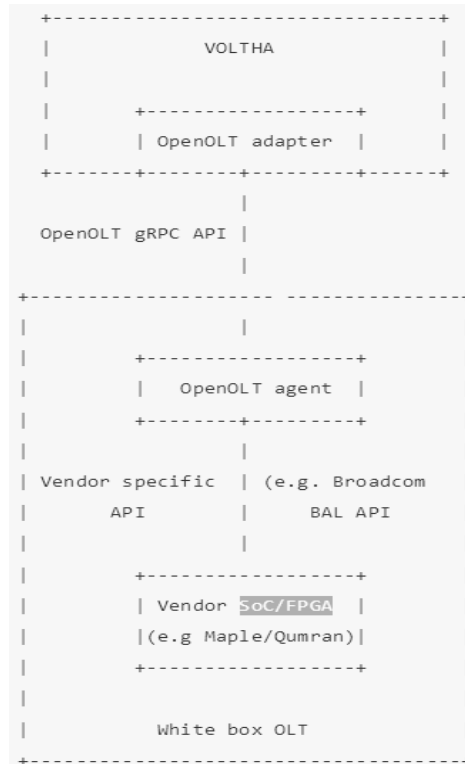


Figura 15: Conexión entre el adaptador y el agente OpenOLT

3.3 Instalación y puesta en marcha de ONOS

Tal y como se ha descrito anteriormente, el controlador ONOS es controlador SDN que usa la tecnología VOLTHA para interactuar con OLTs y ONTs en redes PON. Para la instalación de ONOS nos vamos a guiar con las instrucciones de la página oficial de ONOS [31]. Para ello, primero nos movemos a la carpeta `/opt`, ya que los paquetes predeterminados de ONOS asumen que ONOS va a ser instalado en esa ruta:

```
cd /opt
```

Descargamos la versión 2.2.1 de ONOS:

```
sudo wget -c http://downloads.onosproject.org/release/onos-2.2.1.tar.gz
```

El comando anterior nos descarga un archivo `.tar`, por lo que tenemos que descomprimirlo:

```
sudo tar xzf onos-2.2.1.tar.gz
```

Finalmente, renombramos la carpeta descomprimida a `onos`:

```
sudo mv onos-2.2.1 onos
```

Los siguientes pasos son para ejecutar ONOS como un servicio:

```
sudo cp /opt/onos/init/onos.initd /etc/init.d/onos
```

```
sudo cp /opt/onos/init/onos.conf /etc/init/onos.conf
```

```
sudo cp /opt/onos/init/onos.service /etc/systemd/system/
```

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable onos
```

Cada vez que queramos correr ONOS, debemos usar el comando:

```
sudo service onos start
```

3.4 Instalación y puesta en marcha de VOLTHA

Para la instalación de VOLTHA nos vamos a guiar con las instrucciones del repositorio oficial de VOLTHA [26]. Una vez que hemos accedido al repositorio, tenemos que elegir la versión de VOLTHA que vamos a instalar. Debido a que hemos instalado previamente la versión 2.2.1 de ONOS, la versión de VOLTHA compatible y más estable es la 1.7.0. Por tanto en el repositorio tenemos que indicar que vamos a usar la versión 1.7.0. Esta opción se encuentra en el menú para cambiar entre ramas y etiquetas y nosotros elegiremos la etiqueta 1.7.0.

Primero accedemos al archivo *GettingStartedLinux.md*, que nos indica qué paquetes, programas y librerías nos van a hacer falta. Aunque la guía está pensada para Ubuntu, también es válida para nuestro caso, Mint 19.3. A continuación, ejecutamos los siguientes comandos en nuestra máquina para instalar paquetes base:

```
sudo apt update
```

```
sudo apt upgrade --yes
```

```
sudo apt install python-pip
```

```
sudo pip install --upgrade pip
```

```
sudo apt install git --yes
```

```
sudo apt install make --yes
```

```
sudo apt install virtualenv --yes
```

```
sudo apt install curl --yes
```

```
sudo apt install jq --yes
```

```
sudo apt install libssl-dev --yes
```

```
sudo apt install libffi-dev --yes
```

```
sudo apt install libpcap-dev --yes
```

```
sudo apt install python --yes
```

```
sudo apt install python-dev --yes
```

```
sudo apt install netifaces --yes
```

También será necesario instalar Docker, ya que no vamos a instalar VOLTHA en otra máquina virtual, si no que se va a ejecutar en la máquina local. Para instalar Docker y docker-compose se siguen los siguientes pasos:

```
sudo apt-get update

sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
58118E89F3A912897C070ADBF76221572C52609D

sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial
main'

sudo apt-get update

apt-cache policy docker-engine

sudo apt-get install -y docker-engine

sudo systemctl status docker

sudo usermod -aG docker $(whoami)

sudo curl -L
"https://github.com/docker/compose/releases/download/1.10.0/docker-compose-
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

La guía continúa con la instalación de Virtualbox y otras herramientas. En nuestro caso ya estamos en una máquina virtual, por lo que no será necesario. Por tanto no instalamos ni Virtualbox, ni Repo, ni Vagrant.

El siguiente paso sería la instalación y despliegue de VOLTHA. El archivo *BUILD.md* del repositorio nos indica los pasos que deberíamos seguir para la instalación con Virtualbox y Vagrant, pero no es nuestro caso. Nosotros solo vamos a usar Docker, por lo que vamos al archivo *DOCKER_BUILD.md* y clonamos el repositorio a nuestra máquina mediante el comando:

```
git clone https://github.com/opencord/voltha.git
```

La versión de VOLTHA compatible con ONOS 2.2.1 es la 1.7.0, por lo que tenemos que movernos a esa rama. El comando anterior nos habrá creado una carpeta llamada *voltha*, y nos movemos a ella:

```
cd voltha
```

Para cambiar de rama ejecutamos el siguiente comando:

```
git checkout tags/1.7.0
```

Comprobamos que se ha cambiado correctamente con:

```
git branch
```

A continuación ejecutamos el comando:

```
VOLTHA_BUILD=docker make build
```

Este comando ejecuta una regla del fichero *Makefile*, que indica que se realice la operación *build*, es decir, la instalación. Esta operación tarda en torno a unos 30 minutos, y en nuestro caso no la realiza correctamente, sino que produce varios errores. Es necesario hacer cambios en varios ficheros para que funcione:

- En *voltha/docker/config/dependencies.xml* tenemos que hacer que todos los enlaces sean http seguro, es decir, cambiar en todos los enlaces que aparezcan http por https. Esto arregla un error que hace que al descargar uno de esos paquetes provoque un código de error 501 de internet y se pare la instalación.
- Otro problema surge debido a la configuración del DNS de Docker, posiblemente debido a que estamos en una máquina virtual de la escuela y tenga restricciones. En otro momento de la instalación debido a que Docker no puede descargar un determinado componente ya que no detecta que no está conectado a la red, el error es *“failed to fetch”*. Para solucionarlo primero ejecutamos el comando *nmcli dev show | grep 'IP4.DNS'* que nos devuelve la dirección de nuestro servidor DNS, esto es 10.0.0.250. Ahora tenemos que cambiar el servidor DNS en el demonio de Docker. Para ello vamos a */etc/docker/daemon.json* y si este archivo no existe, se crea. En el escribimos nuestro DNS de esta manera:

```
{  
  "dns": ["10.0.0.250", "8.8.8.8"]  
}
```

Podemos observar que añadimos el de Google como secundario. Guardamos los cambios y reiniciamos el servicio de Docker para aplicar los cambios mediante el comando *sudo service docker restart*.

- El último de los errores es provocado por el contenedor *portainer*, que no encuentra el paquete necesario para la instalación. Si vamos al archivo *voltha/docker/Dockerfile.portainer_d* encargado de la instalación en Docker de ese contenedor, nos encontramos en la línea 15 *FROM \${REGISTRY}portainer/portainer:1.15.2 as base*, lo que quiere decir que descargue la versión 1.15.2, pero no encuentra esa versión. Si vamos a la página oficial de ese contenedor en <https://hub.docker.com/r/portainer/portainer/tags> y buscamos la versión 1.15.2 vemos que no hay una que se llame exactamente 1.15.2, ese es el problema. Depende del tipo de procesador. Para ver cual tenemos nosotros ejecutamos el comando *dpkg --print-architecture* y nos indica que es amd64. Por tanto buscamos en el repositorio la versión para ese procesador, *linux-*

amd64-1.15.2. Finalmente cambiamos la línea anteriormente mencionada por *FROM \${REGISTRY}portainer/portainer:linux-amd64-1.15.2 as base*.

Una vez que se han resuelto todos los problemas, volvemos a ejecutar el comando:

```
VOLTHA_BUILD=docker make build
```

Para comprobar que la instalación se ha completado correctamente, vamos a iniciar VOLTHA. Primero vamos a la carpeta *voltha* y ejecutamos el siguiente comando, ya que VOLTHA corre como una pila *docker swarm*:

```
docker swarm init
```

Una vez acabe, ejecutamos el siguiente comando para levantar todos los servicios de VOLTHA:

```
VOLTHA_BUILD=docker make start
```

Tras la espera podemos comprobar que todos los servicios de VOLTHA se han levantado con:

```
docker service ls
```

Para entrar en la terminal de VOLTHA utilizamos el comando:

```
ssh -p 5022 voltha@localhost
```

El siguiente paso es iniciar todos los contenedores de VOLTHA que son 15 en total. Antes de nada tenemos que modificar el archivo *compose/docker-compose-system-test.yml*, de modo que en la parte del controlador de *ofagent* tenemos que poner la IP que tiene asignada ONOS (10.0.60.2) tal y como se ve en la Figura 16.

```
ofagent:
  image: "${REGISTRY}${REPOSITORY}voltha-ofagent${TAG}"
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"
  # Use the fluentd driver to push logs to fluentd instead
  # driver: "fluentd"
  # options:
  #   fluentd-address: ${DOCKER_HOST_IP}:24224
  command: [
    "/ofagent/ofagent/main.py",
    "-v",
    "--consul=${DOCKER_HOST_IP}:8500",
    "--controller=10.0.60.2:6653",
    "--grpc-endpoint=@voltha-grpc",
    "--instance-id-is-container-name",
    "-v"
  ]
```

Figura 16: Cambio realizado en el archivo *docker-compose-system-test.yml*

A continuación, ejecutamos el siguiente comando para comprobar que no hay contenedores ejecutándose en el servidor:

```
docker ps -a
```

Si hubiese, ejecutamos para pararlos el siguiente comando:

```
docker ps -a | grep -v CONT | awk '{print $1}' | xargs docker rm -f
```

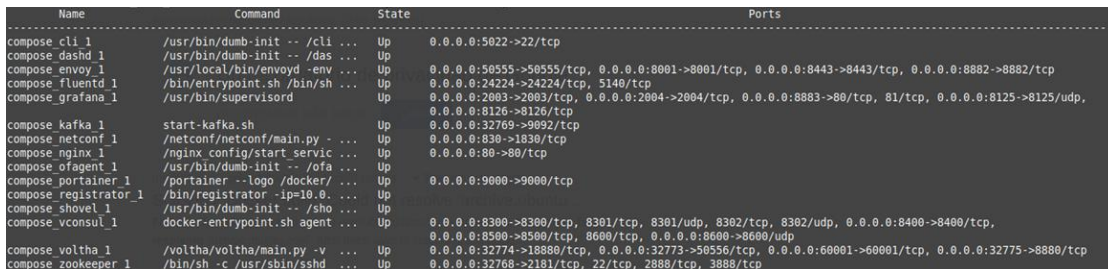
Por último, iniciamos los 15 contenedores de VOLTHA mediante el comando:

```
docker-compose -f compose/docker-compose-system-test.yml up -d
```

Para verificar que se han desplegado correctamente todos los contenedores podemos ejecutar:

```
docker-compose -f compose/docker-compose-system-test.yml ps
```

Y finalmente, todos deberían de tener el estado *Up*. En la Figura 17 vemos los 15 contenedores de VOLTHA levantados.



Name	Command	State	Ports
compose_cli_1	/usr/bin/dumb-init -- /cli ...	Up	0.0.0.0:5022->22/tcp
compose_dashd_1	/usr/bin/dumb-init -- /das ...	Up	
compose_envoy_1	/usr/local/bin/envoyd -env ...	Up	0.0.0.0:50555->50555/tcp, 0.0.0.0:8001->8001/tcp, 0.0.0.0:8443->8443/tcp, 0.0.0.0:8882->8882/tcp
compose_fluentd_1	/bin/entrypoint.sh /bin/sh ...	Up	0.0.0.0:24224->24224/tcp, 5140/tcp
compose_grafana_1	/usr/bin/supervisord	Up	0.0.0.0:2003->2003/tcp, 0.0.0.0:2004->2004/tcp, 0.0.0.0:8883->80/tcp, 81/tcp, 0.0.0.0:8125->8125/udp,
			0.0.0.0:8126->8126/tcp
compose_kafka_1	start-kafka.sh	Up	0.0.0.0:32769->9092/tcp
compose_netconf_1	/netconf/netconf/main.py - ...	Up	0.0.0.0:830->1830/tcp
compose_nginx_1	/nginx config/start_servic ...	Up	0.0.0.0:80->80/tcp
compose_ofagent_1	/usr/bin/dumb-init -- /ofa ...	Up	
compose_portainer_1	/portainer --logo /docker/ ...	Up	0.0.0.0:9000->9000/tcp
compose_registrator_1	/bin/registrator -ip=0.0. ...	Up	
compose_shovel_1	/usr/bin/dumb-init -- /sho ...	Up	
compose_vconsul_1	docker-entrypoint.sh agent ...	Up	0.0.0.0:8300->8300/tcp, 8301/tcp, 8301/udp, 8302/tcp, 8302/udp, 0.0.0.0:8400->8400/tcp,
			0.0.0.0:8500->8500/tcp, 8600/tcp, 0.0.0.0:8600->8600/udp
compose_voltha_1	/voltha/voltha/main.py -v ...	Up	0.0.0.0:32774->18880/tcp, 0.0.0.0:32773->50556/tcp, 0.0.0.0:60001->60001/tcp, 0.0.0.0:32775->8880/tcp
compose_zookeeper_1	/bin/sh -c /usr/sbin/sshd ...	Up	0.0.0.0:32768->2181/tcp, 22/tcp, 2888/tcp, 3888/tcp

Figura 17: Contenedores de VOLTHA levantados

3.5 Instalación y puesta en marcha de BBSim

Los comandos para la instalación nos los encontramos en el repositorio de una versión anterior de BBSim, llamada voltha-bbsim [32]. Pero aunque nosotros no vamos a instalar esa versión, sino una más actual [33], los comandos son los mismos. Así, en primer lugar clonamos el repositorio y nos movemos a la carpeta que se ha generado:

```
git clone https://github.com/opencord/bbsim.git
```

```
cd bbsim
```

Establecemos como versión la 0.0.19:

```
git checkout tags/v0.0.19
```

El siguiente paso es construir el contenedor:

```
make docker-build
```

Una vez creado, le ejecutamos con el comando:

```
docker run -it --rm --privileged=true --expose=50060 --  
network=compose_default bbsim:0.0.19 ./bbsim --nni 16
```

Tras esto ya tenemos el contenedor con BBSim corriendo.

3.6 Conexión de ONOS con VOLTHA y BBSIM

Una vez que tenemos instalados y listos todos las plataformas que necesitamos, vamos a proceder a construir la arquitectura de red cuya topología y conectividad se muestra en la Figura 18.

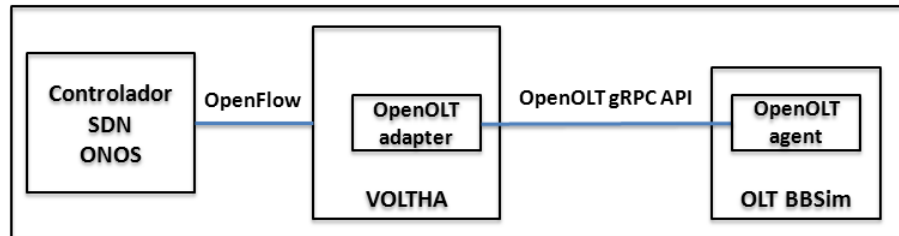


Figura 18: Topología de la red ONOS, VOLTHA y BBSim

En primer lugar tenemos que arrancar ONOS:

```
sudo service onos start
```

Esta operación tarda entre 3 y 4 minutos. Tras la espera, podemos acceder a la consola de comandos de ONOS, para asegurarnos que se está ejecutando correctamente:

```
ssh -p 8101 onos@localhost
```

Nos pedirá una contraseña, por defecto “*rocks*”. Aunque desde esta consola podemos realizar todas las operaciones que necesitemos de ONOS, tales como activar o desactivar aplicaciones, visualizar la topología o consultar las tablas de flows de los switches conectados a ONOS, nosotros preferimos usar la GUI de ONOS, que proporciona un modo gráfico que es más fácil e intuitivo de usar que el CLI (línea de comandos). Para entrar en el modo gráfico tenemos que poner en el navegador la siguiente dirección:

```
10.0.60.2:8181/onos/ui/#/topo2
```

La dirección IP 10.0.60.2 es la dirección donde está ejecutándose ONOS, como en nuestro caso es en la propia máquina podríamos haber puesto *localhost* también. Una vez dentro de ONOS, en el menú de la derecha, seleccionamos Aplicaciones. La Figura 19 muestra este menú. Tenemos que activar las siguientes aplicaciones:

- org.onosproject.drivers
- org.onosproject.drivers.ovsdb
- org.onosproject.hostprovider
- org.onosproject.lldpprovider
- org.onosproject.gui2
- org.onosproject.ovsdb-base
- org.onosproject.ovsdbhostprovider

- org.onosproject.ofagent
- org.onosproject.openflow-base
- org.onosproject.openflow
- org.onosproject.openflow-message
- org.onosproject.workflow.ofoverlay
- org.onosproject.optical-model
- org.onosproject.proxyarp
- org.onosproject.tunnel
- org.onosproject.virtual
- org.onosproject.workflow

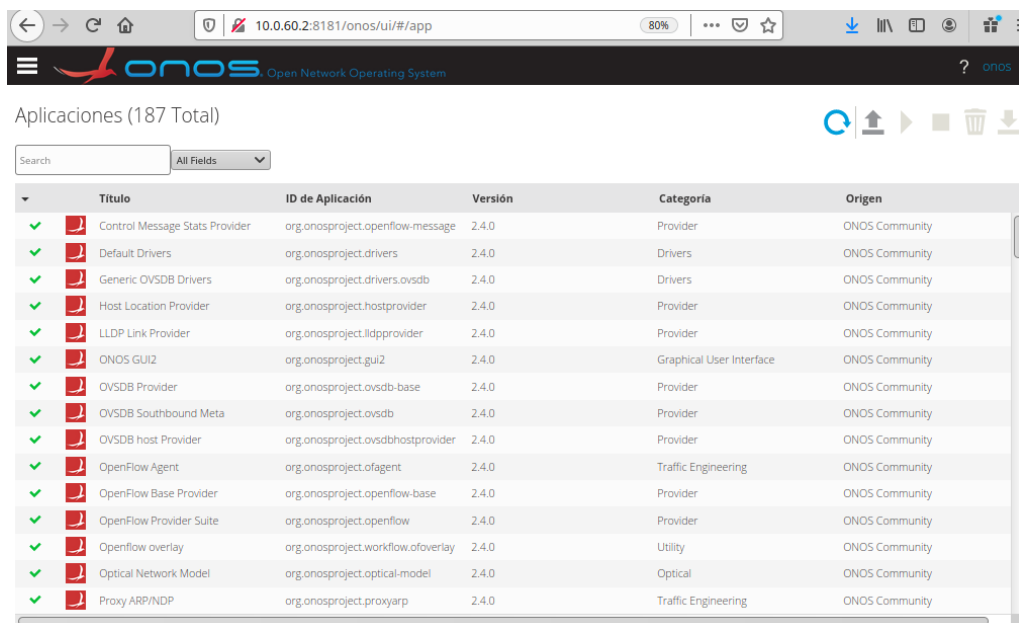


Figura 19: Menú de Aplicaciones de ONOS

Una vez activadas, ya tenemos el controlador listo. Puesto que en la sección anterior ya pusimos en marcha a VOLTHA, y especificamos que se conectará a un controlador en la dirección 10.0.60.2 y puerto 6653, que corresponden con la dirección IP y el puerto donde escucha ONOS, la consecuencia es que ya tenemos conectados ONOS y VOLTHA. El último paso es conectar BBSim con VOLTHA y que a su vez sea visible en ONOS. De este modo, nuestro controlador ONOS ya podrá gestionar y configurar la red PON virtual BBSim a través de la plataforma VOLTHA. Para conectar BBSim con VOLTHA tenemos que seguir una serie de pasos. En la sección anterior pusimos en marcha el contenedor de BBSim con el comando:

```
docker run -it --rm --privileged=true --expose=50060 --network=compose_default bbsim:0.0.19 ./bbsim --nni 16
```


Ahora ya abrimos otra terminal y en ella ejecutamos el comando:

```
docker ps -a
```

Buscamos el contenedor BBSim que acabamos de ejecutar y nos quedamos con su id de contenedor Docker. Ahora necesitamos saber la IP de ese contenedor y para ello ejecutamos:

```
docker inspect <id bbsim> | grep "IPAddress"
```

El cual nos devuelve que es la dirección IP 172.18.0.17. Seguidamente, nos conectamos al CLI de VOLTHA con:

```
ssh -p 5022 voltha@localhost
```

Y a continuación ejecutamos el comando:

```
health
```

Este comando ejecutado en el CLI de VOLTHA nos debería devolver *HEALTHY*, que indica que VOLTHA está funcionando correctamente. El siguiente paso es preprovisionar nuestra OLT de BBSim con el comando:

```
preprovision_olt -t openolt -H 172.18.0.17:50060
```

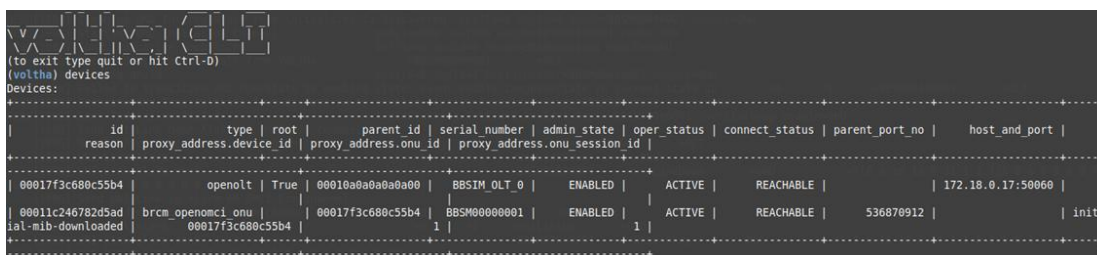
Como podemos observar, en este comando indicamos que el tipo de adaptador con el que nos queremos conectar a la red PON virtual es openOLT y que la dirección y puerto de la OLT a preprovisionar es la 172.18.0.17:50060, esto es, la dirección de BBSim. Preprovisionar es el acto de conectar la red con VOLTHA. Una vez que VOLTHA es capaz de alcanzar la red preprovisionada, la tenemos que activar. El siguiente comando habilita la OLT provisionada:

```
enable
```

Por último vemos que se ha creado la OLT con el comando:

```
devices
```

La salida del comando anterior la podemos observar en la Figura 20. La OLT tiene los estados ENABLED, ACTIVE y REACHABLE, que indican que está habilitada, activa y que es alcanzable en la red, respectivamente. En la GUI de ONOS también podemos observar que visualizamos la OLT de BBSim, como muestra la Figura 21.



```
(to exit type quit or hit ctrl-D)
(voltha) devices
Devices:
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id          | type | root | parent_id | serial number | admin state | oper_status | connect_status | parent_port_no | host_and_port |
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 00017f3c680c55b4 | openolt | True | 00010a8a0a0a0a00 | BBSIM_OLT_0 | ENABLED | ACTIVE | REACHABLE | | 172.18.0.17:50060 |
| 00011c246782d5ad | brcm_openmci_onu | | 00017f3c680c55b4 | BBSM00000001 | ENABLED | ACTIVE | REACHABLE | 536870912 | |
| ial=mib-downloaded | 00017f3c680c55b4 | | 1 | 1 | | | | | |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figura 20: CLI de VOLTHA con la OLT de BBSim habilitada

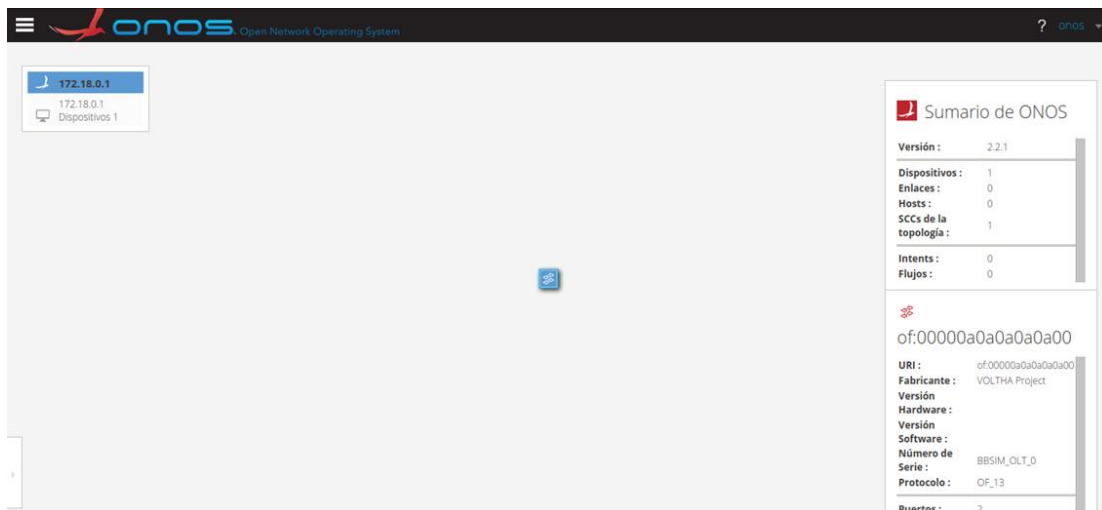


Figura 21: Vista en ONOS de la OLT de BBSim habilitada

El resultado final es que hemos logrado conectar nuestro controlador ONOS con VOLTHA, y este con la OLT de BBSim.

Cabe destacar que durante la realización de esta parte nos han surgido varios problemas que se van a resumir como:

- La versión 1.7.0 de VOLTHA está muy desactualizada. En estos momentos la última versión es la 2.4.0. Utilizar esta versión podría generar un problema en el futuro debido a la falta de soporte y que hubiese que migrar a una versión posterior.
- La versión 2.2.1 de ONOS tampoco es la última. Durante la realización del trabajo, ONOS ya lanzó la versión 2.4.0.
- El problema más grave surgió cuando por causas ajenas a nosotros, el servicio de virtualización de la escuela falló, lo que supuso que al restablecerse, las máquinas virtuales se reiniciasen, entre ellas, la nuestra. La consecuencia fue que de los 15 contenedores que se levantan de VOLTHA, el contenedor principal y el *ofagent* dejasen de funcionar. Alguno de los demás contenedores también se paraba y se reiniciaba a intervalos.

Tratamos de solucionar el último problema de diferentes maneras, tales como reiniciando VOLTHA, reiniciando la máquina virtual de nuevo, borrando VOLTHA y volviendo a realizar todo el proceso de instalación y puesta en marcha completo. Con ninguna de estas soluciones conseguimos que VOLTHA volviese a funcionar. La falta de soporte en el sitio web oficial tampoco ayudó. Finalmente decidimos volver a empezar con todo el proceso desde una nueva máquina.

3.7 Versión actualizada de ONOS, VOLTHA y BBSim

En este punto del trabajo, se decidió cambiar el sistema operativo a Ubuntu 18.04, ya que en las guías de instalación de VOLTHA recomienda como punto de partida Ubuntu 16.04. Sin embargo, nosotros instalamos una versión más nueva. Ya que tenemos una máquina limpia, tenemos que volver a instalar ONOS y BBSim. De BBSim mantenemos la misma versión, la 0.0.19, pero en ONOS instalamos la última versión, la 2.4.0. La guía de instalación de ONOS de la anterior sección sirve igual, pero cambiando el número de versión.

El repositorio de VOLTHA en Github ya no se llama *voltha*, sino *voltha-go* [34], ya que a partir de la versión 2 en adelante, VOLTHA deja de estar desarrollado en Python y pasa a un nuevo lenguaje llamado Go [35]. En este repositorio tenemos dos archivos interesantes, el primero llamado *quickstart.md*, que indica los prerequisites para instalar VOLTHA y entre ellos están la instalación de *docker-ce* (nueva versión de Docker) y de Go. Realizamos todos los pasos de instalación de los prerequisites de esta guía.

Una vez realizado este proceso vamos a clonar el nuevo repositorio y elegiremos la versión 2.2 de VOLTHA:

```
git clone https://github.com/opencord/voltha-go.git
cd voltha-go
git checkout tags/2.2.0
```

El segundo archivo, llamado *BUILD.md*, indica los pasos para instalar VOLTHA. En este caso no los vamos a escribir en este documento, ya que a diferencia de la versión 1.7.0, el proceso de instalación no genera ningún error. Esta versión de VOLTHA no tiene un contenedor de CLI, sino que hay que descargar e instalar un programa aparte llamado *volctl*, un CLI para gestionar y operar con los componentes de VOLTHA [36]. En el archivo *README.md* tenemos la guía de instalación de esta herramienta. Seguimos los pasos para instalar y poner en marcha esta herramienta.

Con la versión 2.2 de VOLTHA instalada y con *volctl* listo, solo nos queda poner en marcha VOLTHA, aunque primero debemos modificar el archivo *compose/system-test.yml* como hicimos con la anterior versión, es decir, poner la IP 10.0.60.2 en el campo del controlador en el *ofagent*. Además tenemos que comentar todas las referencias a ONOS del fichero, ya que si no lo hacemos nos instalaría ONOS en un contenedor, pero nosotros ya lo tenemos en la máquina local. Una vez hecho esto, ponemos en marcha VOLTHA mediante el comando:

```
DOCKER_HOST_IP=10.0.60.2 DOCKER_TAG=2.2.3-dirty docker-compose -f
system-test.yml up -d
```

El número de contenedores se reduce de 15 de la versión 1.7.0 a 7, como vemos en la Figura 22. De todos estos contenedores solo nos interesan 3 de ellos, *rw_core* como

contenedor principal, *ofagent* es agente OpenFlow y *adapter_openolt* es el adaptador de OpenOLT.

Name	Command	State	Ports
compose_adapter_openolt_1	/app/openolt -kafka_adapt ...	Up	0.0.0.0:50062->50062/tcp
compose_adapter_openonu_1	/voltha/adapters/brcm_open ...	Up	
compose_etcd_1	etcd --name=etcd0 --advert ...	Up	0.0.0.0:2379->2379/tcp, 0.0.0.0:32769->2380/tcp, 0.0.0.0:32768->4001/tcp
compose_kafka_1	start-kafka.sh	Up	0.0.0.0:9092->9092/tcp
compose_ofagent_1	/ofagent/ofagent/main.py - ...	Up	
compose_rw_core_1	/app/rw_core -kv_store_typ ...	Up	0.0.0.0:50057->50057/tcp
compose_zookeeper_1	/bin/sh -c /usr/sbin/sshd ...	Up	0.0.0.0:2181->2181/tcp, 22/tcp, 2888/tcp, 3888/tcp

Figura 22: Contenedores de la versión VOLTHA 2.2

Tal y como hicimos en la sección anterior, vamos a conectar VOLTHA con la OLT de BBSim. Es ahora donde usamos la herramienta de *voltctl*. Los comandos son los siguientes:

```
voltctl device create -t openolt -H 172.18.0.17:50060
```

Este comando devuelve un id, que se usará en los siguientes comandos:

```
voltctl device enable <id>
```

```
voltctl device list
```

El último comando nos sirve para listar todos los dispositivos conectados, para ver que hemos conectado correctamente la OLT de BBSim.

De nuevo, durante la realización de esta parte nos han surgido dos problemas principales que condujeron al replanteamiento del futuro trabajo para alcanzar el objetivo de este trabajo:

- Esta versión de VOLTHA está programada en el lenguaje Go, lenguaje de programación que desconocemos y que aprender para poder continuar con este proyecto nos llevaría demasiado tiempo para este TFG sin saber si después el objetivo de poder desarrollar la programación para gestionar nuestra red GPON podría ser abarcable en un trabajo de estas características y duración .
- El problema principal se encontró en el agente OpenOLT que se debe instalar en la OLT. En concreto, el agente OpenOLT utiliza el software BAL (*Broadband Adaptation Layer*) de Broadcom para interactuar con los chips Maple/Qumran en las OLT, de modo que Accton/Edgecore pone a disposición de sus clientes paquetes Debian preconstruidos del agente OpenOLT y código fuente que necesita ser ejecutado para hacer operativo al agente OpenOLT [30]. Puesto que el fabricante de nuestros dispositivos GPON no es cliente ni mantiene una colaboración con Accton/Edgecore, estos paquetes no han sido suministrados, con lo que en este punto nos replanteamos el trabajo para intentar conseguir el objetivo final, esto es, intentar desarrollar un sistema de gestión centralizada de nuestra red GPON con ONOS a través el protocolo OpenFlow.

4

Análisis de ONOS, Open vSwitch y Mininet

4.1 Introducción

Tal y como explicamos en el capítulo anterior, no podemos continuar con el desarrollo del trabajo en la misma dirección ya que no tenemos acceso a parte del código del agente OpenOLT para poder desarrollar un sistema para conectar con la red GPON del laboratorio. Por este motivo, rediseñamos el proyecto con una nueva planificación y rediseñando parte de los objetivos. Los nuevos objetivos del proyecto estaban ya definidos en el Capítulo 1.

El objetivo principal es crear un agente basado en OpenFlow para gestionar redes de acceso PON mediante OpenFlow y utilizando ONOS de modo parecido a VOLTHA usando el agente y adaptador OpenOLT. La idea es crear un agente de código abierto usando OpenFlow extensible para cualquier red PON, de modo que pueda ser instalado en cualquier OLT y pueda interactuar con su API realizando las adaptaciones pertinentes para su gestión y configuración.

Es necesario rediseñar el diagrama de Gantt para que refleje las nuevas tareas, que se muestra en la Figura 23, y donde las tareas en verde son las que ya se han realizado y las que están en azul las nuevas que se tienen que realizar para continuar con el proyecto. La fase de análisis sufre una modificación en su duración debido a la inserción de una nueva tarea para el análisis del controlador SDN ONOS y el protocolo OpenFlow. Esta tarea tiene una duración de 9 días. La fase de diseño, ahora del agente OpenFlow, mantiene su duración, pero la fase de programación de este agente se reduce a 12 días y la fase de pruebas y documentación a 6 días. La fecha de finalización del proyecto no se ve alterada.

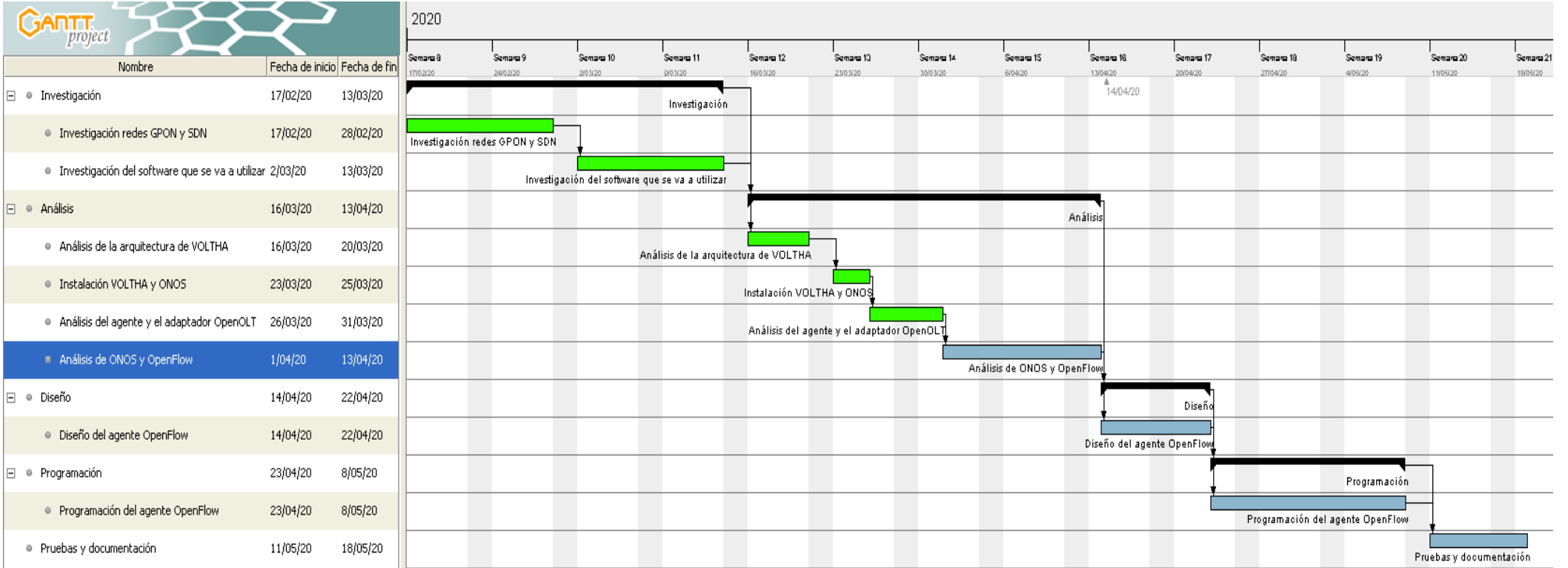


Figura 23: Diagrama de Gantt de la planificación del proyecto modificado

En este capítulo se explicará en primer lugar como se instala Mininet y Open vSwitch. Estas dos herramientas nos permitirán realizar dos pruebas, cuyo objetivo es ver como se crean redes de switches virtuales con Mininet para posteriormente crear los correspondientes flows que den conectividad a los host de la red. En ambas pruebas el controlador será ONOS, aunque la creación de los flows se realizará mediante una aplicación de ONOS en la primera prueba y directamente sobre Open vSwitch en la segunda prueba. Finalmente, en la tercera y última prueba, los flows se crearán desde la API de ONOS, que será desde donde se instalen los flows en el futuro para el agente OpenFlow que vamos a desarrollar. Además de los flows, también se crearán meters, una estructura OpenFlow que se explicará su funcionalidad en el futuro agente en esta prueba.

4.2 Instalación de Mininet y Open vSwitch

Para instalar Mininet solo tenemos que ejecutar el siguiente comando:

```
sudo apt-get install mininet
```

Al instalar Mininet, se nos instala también la versión 2.9.5 de Open vSwitch (OVS), ya que es una dependencia de Mininet. El problema es que esta versión de OVS no soporta la creación de meters en los switches y nosotros necesitamos que se puedan añadir meters. Por eso tenemos que desinstalar la versión 2.9.5 para instalar a continuación la versión 2.10.1 de OVS, que sí que soporta meters y es compatible con la versión 2.4.0 de ONOS. Para ello, primero desinstalamos la versión actual:

```
sudo apt-get remove openvswitch-switch
```

A continuación, se clona el repositorio y se cambia a la versión 2.10.1. También ejecutamos los comandos para instalar la nueva versión [37]:

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

```
cd ovs
```

```
./boot.sh
```

```
./configure
```

```
make
```

```
make install
```

El último paso es poner en marcha el OVS instalado mediante el comando:

```
sudo /usr/local/share/openvswitch/scripts/ovs-ctl start
```

4.3 Prueba 1: Mininet + ONOS con la aplicación fwd

La primera prueba que vamos a realizar es la más sencilla de todas. Con Mininet vamos a crear una red pequeña formada por un switch con un host conectado, otro switch

con otro host conectado y finalmente los dos switch conectados. Ambos switches serán controlados por ONOS. El objetivo será conseguir que se haga ping desde el host 1 al host 2. Para iniciar el servicio de ONOS ejecutamos el siguiente comando:

```
sudo service onos start
```

Y a continuación creamos una red simple con Mininet con el comando:

```
sudo mn --topo linear,2 --controller remote,ip=10.0.60.2,port=6633
```

Esto nos crea una topología de red formada por dos switches conectados entre sí y conectado a cada uno de ellos un host. La dirección IP 10.0.60.2 es nuestro *localhost*, que es donde está ONOS instalado y escuchando en el puerto 6633.

Abrimos una pestaña en Firefox con la dirección *10.0.60.2:8181/onos/ui/#/topo2* para ver una representación gráfica de nuestra red. En el menú de abajo tenemos que activar la opción de visibilizar los host. De momento aunque esté activada esa opción, los host no serán visibles hasta que no se hayan descubierto, hay que hacer ping entre ellos. En Mininet tenemos el comando *pingall*, que hace un ping desde cada host a todos los demás y ejecutamos ese comando para hacer ping entre los dos hosts. En este caso todos los pings fallan ya que no existe todavía ningún flow que indique a los switches por donde tienen que enviar los paquetes.

Así pues, ONOS tiene una aplicación llamada *fwd* (*Reactive Forwarding*) que al activarse instala por defecto un flow con una prioridad baja para interceptar todos los paquetes IPv4 con la acción de enviar al controlador. Después, cuando se envía un paquete, la aplicación lo procesa y calcula a partir de la información que recibe del subsistema de topología y del subsistema de localización de los hosts, la ruta más corta entre el host origen del paquete y el host destino e instala un flow de prioridad alta en cada salto para que los paquetes se puedan transmitir. El flow se mantiene durante un tiempo para que si otro paquete es enviado entre los dos mismos hosts, ya exista un flow con el que directamente haga *Match* el paquete. Para activar esta aplicación, en el menú aplicaciones activamos el paquete *Reactive Forwarding* (*org.project.fwd*). Esto también se puede activar en el CLI de ONOS con el comando:

```
app activate org.project.fwd
```

Una vez activada la aplicación volvemos a ejecutar en la terminal de Mininet el comando *pingall*. Ahora si tenemos conectividad entre los host y podemos visualizar la red completa en ONOS, tal y como se observa en la Figura 24.

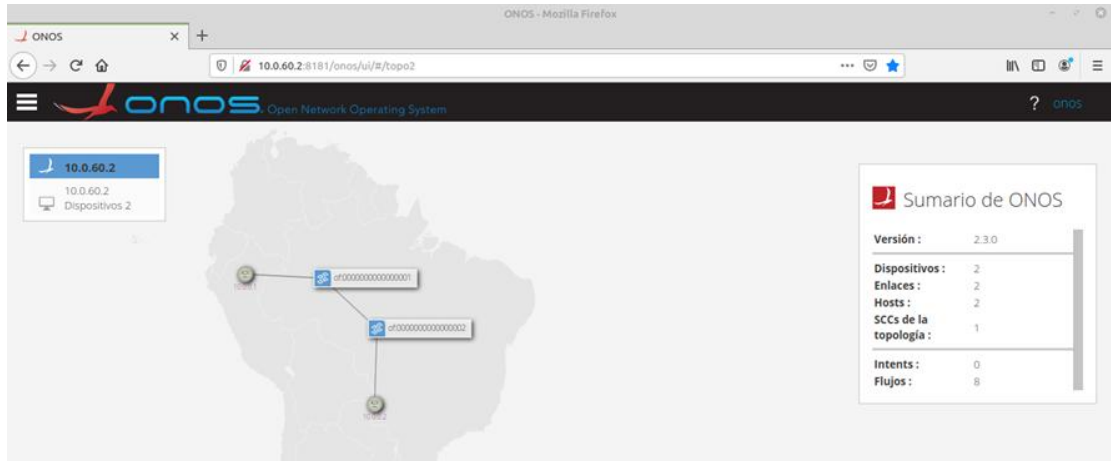


Figura 24: Red creada en la prueba con Mininet y ONOS

4.4 Prueba 2: Mininet + ONOS sin aplicación fwd

Para la segunda prueba se va a utilizar de nuevo ONOS como controlador SDN y Mininet para crear la red de switches. Tenemos dos objetivos, el primero es la creación de una red personalizada en Mininet y el segundo es la creación de flows manualmente (sin *fwd*) a nivel de capa 1, 2 y 3.

Lo primero que vamos a hacer es programar nuestra red en Python. Mininet tiene una serie de funciones para poder añadir hosts a la red (*addHost()*), añadir switches (*addSwitch()*) y añadir enlaces entre un host y un switch o entre dos switches (*addLink()*). Vamos a crear una topología circular, donde tenemos tres switches conectados entre ellos y dos hosts conectados a cada switch. El código de la red se puede ver en la Figura 25.

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        h5 = self.addHost( 'h5' )
        h6 = self.addHost( 'h6' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )

        # Add links
        self.addLink( h1, s1 )
        self.addLink( h2, s1 )
        self.addLink( s1, s2 )
        self.addLink( h3, s2 )
        self.addLink( h4, s2 )
        self.addLink( s2, s3 )
        self.addLink( h5, s3 )
        self.addLink( h6, s3 )
        self.addLink( s3, s1 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figura 25: Código de la red de la prueba 2

Para crear la red que hemos diseñado, ejecutamos el siguiente comando:

```
sudo mn --custom /home/usuario/mininet/custom/ejemplo1.py --topo mytopo --
controller=remote,ip=10.0.60.2,port=6633 --mac
```

En este comando decimos que nos cree una red a partir de una topología personalizada localizada en un determinado archivo (*ejemplo1.py*), que nuestro controlador está remoto en esa IP y puerto y que las direcciones MAC de los hosts se asignen de forma ordenada. Con esto hemos completado el primer objetivo.

Ahora tenemos que crear flows que permitan enviar paquetes entre los diferentes hosts. En este punto, ya que tenemos activada de la prueba anterior la aplicación *fwd*, si ejecutásemos en la terminal de Mininet el comando *pingall*, tendríamos conectividad entre todos los hosts. Pero nosotros queremos crear los flows manualmente, así que desactivamos en ONOS la aplicación. Los flows desaparecerán ya que tienen un *timeout*. En este momento si creásemos un flow en nuestra red tampoco funcionaría, ya que el flow estaría creado con OVS, pero ONOS no importa ni permite añadir flows que no se hayan creado por ONOS o por alguna aplicación de ONOS (como *fwd*). Para que nuestros flows se puedan añadir, tenemos que activar dos opciones y para ello primero nos conectamos al CLI de ONOS:

```
ssh -p 8101 onos@localhost
```

A continuación, tenemos que activar en las opciones de configuración de ONOS que se puedan añadir flows provenientes de otros programas distintos de ONOS y que ONOS los importe:

```
cfg set org.onosproject.net.flow.impl.FlowRuleManager allowExtraneousRules
true
```

```
cfg set org.onosproject.net.flow.impl.FlowRuleManager importExtraneousRules
true
```

A continuación pasaremos a configurar los flows a diferentes niveles, esto es, nivel físico, enlace y de red.

Flows en nivel físico

Los flows en nivel físico serán creados en el switch *s1*. Todos los comandos que vienen a continuación se ejecutan en la terminal de Mininet. Para crear los flows ejecutamos los comandos:

```
sh ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:2
```

```
sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1
```

Donde el comando *priority* es la prioridad del flow, *in_port* es el campo *Match* y *actions* es el campo que indica qué hacer con el paquete si se cumplen las condiciones del campo *Match*. Lo que estamos diciendo con estos flows es, ante un paquete que

provena del puerto 1 (2) del switch s1, se envíe por el puerto 2 (1). Para ver que los flows se han creado correctamente usamos el comando:

```
sh ovs-ofctl dump-flows s1
```

Vemos que los flujos se han creado correctamente. Probamos que tenemos conectividad:

```
h1 ping h2
```

El ping es se realiza correctamente.

Flows en nivel de enlace

Los flows en nivel de la capa de enlace se crean en el switch s2. Los comandos serán los siguientes:

```
sh ovs-ofctl add-flow s2
dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04,actions=output:3

sh ovs-ofctl add-flow s2
dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03,actions=output:2

sh ovs-ofctl add-flow s2 dl_type=0x806,nw_proto=1,actions=flood
```

Los dos primeros flows indican que ante un paquete que llegue al switch s2 con dirección Mac de origen 00:00:00:00:00:03 (00:00:00:00:00:04) y dirección Mac de destino 00:00:00:00:00:04 (00:00:00:00:00:03), se envíe el paquete por el puerto 3 (2). El tercer flow permite reenviar la solicitud ARP (0x806) por todos los puertos menos por el puerto entrante. A continuación vemos los flows creados y probamos la conectividad con:

```
sh ovs-ofctl dump-flows s2

h3 ping h4
```

El ping es se realiza correctamente.

Flows en nivel de red

Los flows en nivel de la capa de red se crean en s3. Los comandos son:

```
sh ovs-ofctl add-flow s3 nw_dst=10.0.0.5,actions=output:2

sh ovs-ofctl add-flow s3 nw_dst=10.0.0.6,actions=output:3

sh ovs-ofctl add-flow s3 dl_type=0x806,nw_proto=1,actions=flood
```

Los dos primeros flows indican que ante la llegada de un paquete al switch s3 con dirección IP de destino 10.0.0.5 (10.0.0.6), se envíe el paquete por el puerto 2 (3). El tercer flows es para las tramas ARP como en el caso anterior. A continuación, se observan los flows creados y probamos la conectividad con:

```
sh ovs-ofctl dump-flows s3
```

h5 ping h6

El ping es se realiza correctamente.

En la Figura 26 podemos finalmente ver la red creada. Hemos cumplido con el segundo objetivo de esta prueba que era crear los flows manualmente en las tres capas inferiores.

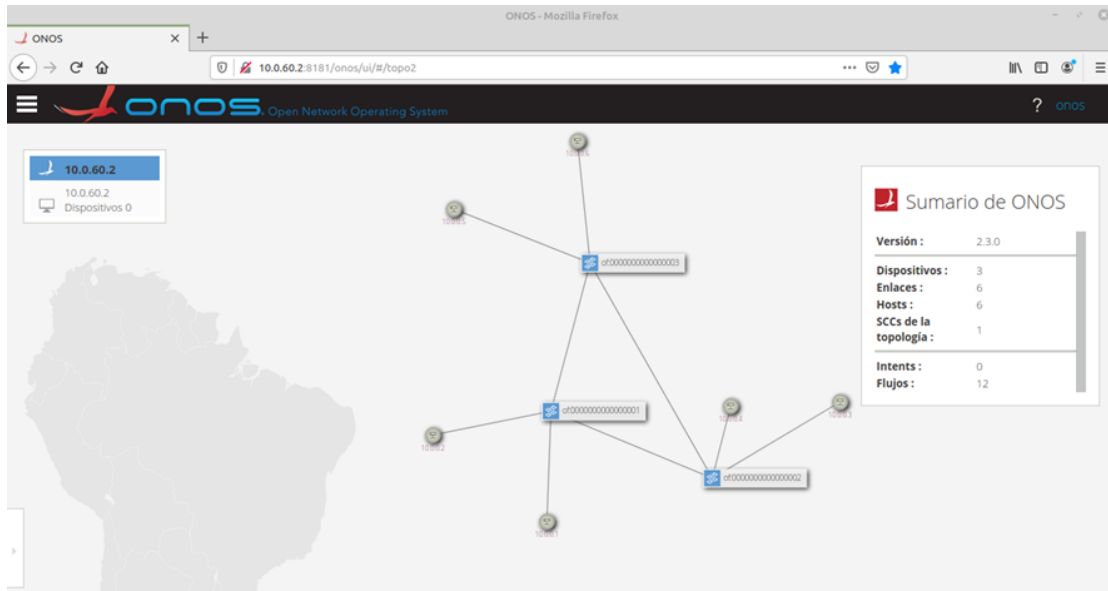


Figura 26: Red creada en la prueba 2

4.5 Prueba 3: Creación de flows en ONOS

Para la última prueba vamos a usar de nuevo Mininet y ONOS, pero en este caso los flows se van a crear con la REST API de ONOS, que es como se instalarán los flows cuando desarrollemos el agente OpenFlow. Para ello, se crea una red simple con Mininet, un switch con dos host. El comando utilizado es:

```
sudo mn --topo single,2 --controller remote,ip=10.0.60.2,port=6633 --  
switch=ovs,protocols=OpenFlow13 --mac --link=tc,bw=1000
```

Los nuevos parámetros nos dicen que el driver del switch es Open vSwitch pero con la versión de OpenFlow 1.3. El parámetro link permite hacer cambios en las interfaces mediante la utilidad *Traffic Control*, para poner que el ancho de banda en los enlaces es de 1000Mb.

Una vez creada la red, ejecutamos el comando *pingall* y puesto que teníamos desactivada la aplicación *fwd* de la prueba anterior, el ping fallará. Antes de volver a activarla, vamos a ver qué flows tiene el switch en estos momentos. Los extraemos de la dirección web *10.0.60.2:8181/onos/v1/flows*. Debido al tamaño que ocupan los flows los mostramos en el ANEXO I. Cuando ONOS detecta un nuevo switch que tiene que controlar, le envía los tres flows que vemos en el ANEXO I. Estos son flows que crea

ONOS por defecto, para enviar los mensajes a través del canal seguro hacia los switches. Deja pasar tramas ARP, LLDP y tramas Open Network OS.

Ahora activamos la aplicación *fwd* y ejecutamos un *pingall* en la terminal de Mininet. Se crearán en ONOS otros 3 flows. Podemos ver los 3 flows nuevos en el *ANEXO II*. El primer flow nuevo es como los anteriores 3, pero permitiendo IP. Los otros 2 nuevos ya los conocemos de las otras pruebas. El significado es que ante un paquete que llegue por el puerto 1 (2) con dirección Mac de origen 00:00:00:00:00:01 (00:00:00:00:00:02) y dirección Mac de destino 00:00:00:00:00:02 (00:00:00:00:00:01) se envíen por el puerto 2 (1). Cabe comentar que en este momento el switch tiene los 6 flows.

En este momento tenemos que explicar otra de las estructuras del protocolo OpenFlow que vamos a utilizar, los meters [5]. Igual que tenemos las tablas de flows, tenemos una sola tabla de meters en cada switch. Los meters permiten a OpenFlow implementar operaciones simples de calidad de servicio (QoS) como limitación de tasa o DiffServ. Un meter mide la tasa de los paquetes asignados a él y habilita que se pueda controlar la tasa de esos paquetes. Los meters están asociados directamente a los flows. Un flow solo puede tener como máximo un meter asociado. Cada meter se identifica en la tabla de meters por su identificador, que es único. De este modo, un meter tiene los campos:

- Identificador: número entero que identifica de manera unívoca al meter.
- *Bands* (Bandas): lista de bandas donde cada una especifica la tasa del meter y como procesar el paquete. Cada meter puede tener una o más bandas. Los campos de cada banda son:
 - Tipo: define como el paquete es procesado.
 - *Rate*: define la tasa mínima en que la banda es aplicada.
 - Contadores: contadores que se actualizan cuando la banda se aplica en paquetes y bytes.
 - Argumentos específicos del tipo de banda: algunos tipos de banda tienen argumentos específicos.
- Contadores: Contadores de paquetes y bytes afectados por el meter.

OpenFlow soporta dos tipos de meter distintos, DROP indica que se descarta el paquete si supera la tasa y DSCP REMARK que indica que se decrementa la precedencia de eliminación en el campo DSCP de la cabecera IP del paquete. Se usa para definir políticas simples de DiffServ. La tasa puede estar en kilobits por segundo o en paquetes por segundo. Además se puede añadir la opción *burst size*, una política que controla el número de bytes de tráfico que pueden pasar sin restricciones a través de una interfaz con una política de tasa límite [38].

Una vez explicados que son los meters, vamos a crear uno en ONOS y añadirlo a los dos flows principales que permiten la comunicación entre los dos hosts. El problema es

que no podemos modificar flows, tenemos que borrarlos y luego volverlos a crear. Tampoco podemos hacer que la aplicación *fwd* cree los flows con meters asociados. La solución es copiar los tres flows que crea *fwd*, pero editándolos a mano para añadir el campo del meter. De este modo, ONOS tiene un servicio REST API con el que podemos añadir flows y meters (entre otros) de forma fácil [39]. Podemos acceder a él en la dirección `10.0.60.2:8181/onos/v1/docs`. Nosotros primero necesitamos añadir un meter y después los flows. Si creáramos los flows antes que el meter, ONOS dejaría los flows en estado pendiente de añadir, ya que no encontraría el meter que llevan asociado. Para añadirlos de manera más automática, creamos un script que realice los mismos comandos que se hacen desde la API, pero en vez de hacerlo uno por uno, todos a la vez. En este script, que podemos ver en el *ANEXO III*, tenemos en primer lugar el meter, donde decimos que es de tipo *DROP*, sin *burst size* y con una tasa de 3000Kbps. A continuación tenemos los flows, donde incluimos en el campo de instrucciones que se aplique el meter con id 1. Antes de lanzar este script, tenemos que volver a desactivar *fwd*. Tras esto ejecutamos el script y comprobamos que tenemos de nuevo conectividad con el comando *pingall*. Para ver que el meter se ha añadido correctamente también, podemos hacer un GET de los meters desde la API, como vemos en la Figura 27.

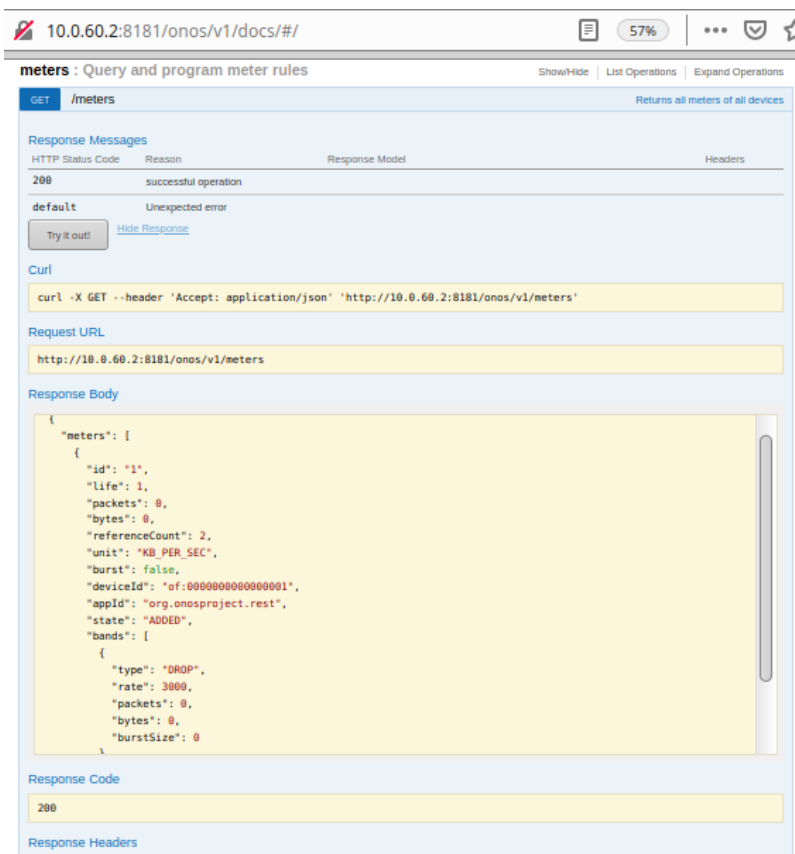


Figura 27: Visualización del meter a través de la API

Con esta prueba hemos aprendido a crear y borrar flows y meters desde ONOS. Esto nos va a ser útil cuando hagamos pruebas con el agente OpenFlow que vamos a diseñar en el siguiente capítulo.

5

Desarrollo de un agente OpenFlow

5.1 Introducción

Una vez analizados los problemas de VOLTHA para crear un agente para gestionar desde ONOS nuestra red GPON y analizado en profundidad ONOS para configurar flujos y tablas en switches virtuales, en este capítulo se presenta el diseño e implementación del agente desarrollado en OpenFlow para configurar nuestra red GPON usando para ello una librería en Python. En la primera parte del capítulo vamos a analizar la comunicación y los mensajes que se intercambian entre un switch y el controlador ONOS para posteriormente diseñar un agente OpenFlow que actúe como un software intermedio entre el controlador y la red GPON.

A continuación se describirá como está programado el agente y se explicarán sus funciones. También se describirán los métodos de la API que permite conectar el agente con la red GPON del laboratorio.

Por último se realizarán una serie de pruebas para verificar el correcto funcionamiento del agente.

5.2 Análisis de la comunicación entre un switch OpenFlow y el controlador ONOS

Cuando un switch se conecta con el controlador, se intercambian una serie de mensajes mediante el protocolo OpenFlow. Para poder ver que secuencia de mensajes se intercambian entre ellos, vamos a hacer uso de Wireshark. Wireshark es el analizador de protocolos de red más conocido y usado en su categoría. Permite capturar paquetes en tiempo real, desde cualquier interfaz de red e inspecciona una gran cantidad de protocolos [40].

Una vez que tenemos el analizador, necesitamos saber entre que dos elementos se dará la comunicación. El controlador será ONOS, que estará escuchando en la IP 10.0.60.2 y para implementar el switch virtual usaremos Mininet. Primero creamos un contenedor Docker para ejecutar desde ahí Mininet:

```
docker run -i -d --name=mininetvm --cap-add NET_ADMIN --privileged=true --net=bridge ubuntu:18.04
```

Entramos en el contenedor mediante el comando:

```
docker exec -it mininet /bin/bash
```

A continuación, ejecutamos los siguientes comandos en el contenedor para instalar Mininet y algunas herramientas de red necesarias:

```
apt-get update
```

```
apt-get upgrade
```

```
apt-get install net-tools iputils-ping iproute2 openvswitch-switch mininet
```

Una vez instalado todo el software, creamos una red de Mininet compuesta por un switch y dos host, el controlador será remoto en la IP 10.0.60.2 (ONOS), el driver del switch será OVS (versión de OpenFlow 1.3) y direcciones MAC serán automáticas. El comando es este:

```
mn --topo=single,2 --controller=remote,ip=10.0.60.2,port=6633 --switch=ovs,protocols=OpenFlow13 --mac
```

Antes de ejecutar ese comando, tenemos que arrancar Wireshark en modo administrador y elegir la interfaz donde va a analizar los paquetes. En nuestro caso la interfaz se llama *docker0*, ya que Mininet está ejecutándose en un contenedor Docker. Filtramos los mensajes que queremos ver a solo *openflow_v4* (la versión 4 es otro nombre para la 1.3). Una vez hecho esto, iniciamos la captura de paquetes y ejecutamos el comando para crear la red de Mininet. En la Figura 28 podemos ver los primeros mensajes capturados.

No.	Time	Source	Destination	Protocol	Length	Info
70	3.880515691	172.17.0.2	10.0.60.2	OpenFlow	82	Type: OFPT_HELLO
72	3.894552399	10.0.60.2	172.17.0.2	OpenFlow	90	Type: OFPT_FEATURES_REQUEST
74	3.895019249	172.17.0.2	10.0.60.2	OpenFlow	98	Type: OFPT_FEATURES_REPLY
75	3.896129014	10.0.60.2	172.17.0.2	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
76	3.896250607	172.17.0.2	10.0.60.2	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
77	3.896699209	10.0.60.2	172.17.0.2	OpenFlow	82	Type: OFPT_GET_CONFIG_REQUEST
78	3.896887351	172.17.0.2	10.0.60.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY
79	3.896928488	172.17.0.2	10.0.60.2	OpenFlow	78	Type: OFPT_GET_CONFIG_REPLY
81	3.898858249	10.0.60.2	172.17.0.2	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER_FEATURES
82	3.898977267	172.17.0.2	10.0.60.2	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_FEATURES
83	3.900961663	10.0.60.2	172.17.0.2	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
84	3.900162255	172.17.0.2	10.0.60.2	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
86	4.070481249	10.0.60.2	172.17.0.2	OpenFlow	90	Type: OFPT_ROLE_REQUEST
87	4.070710744	172.17.0.2	10.0.60.2	OpenFlow	90	Type: OFPT_ROLE_REPLY
89	4.074116374	10.0.60.2	172.17.0.2	OpenFlow	90	Type: OFPT_ROLE_REQUEST
90	4.075596391	172.17.0.2	10.0.60.2	OpenFlow	90	Type: OFPT_ROLE_REPLY
91	4.076936117	10.0.60.2	172.17.0.2	OpenFlow	1842	Type: OFPT_BARRIER_REQUEST
93	4.077069603	172.17.0.2	10.0.60.2	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
94	4.077227640	172.17.0.2	10.0.60.2	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
95	4.077410233	172.17.0.2	10.0.60.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY
96	4.077461084	172.17.0.2	10.0.60.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY
97	4.077580840	172.17.0.2	10.0.60.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY
99	4.091525661	10.0.60.2	172.17.0.2	OpenFlow	170	Type: OFPT_BARRIER_REQUEST
100	4.091768953	172.17.0.2	10.0.60.2	OpenFlow	74	Type: OFPT_BARRIER_REPLY
102	4.959277333	10.0.60.2	172.17.0.2	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
103	4.959648907	172.17.0.2	10.0.60.2	OpenFlow	418	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
105	4.977672565	10.0.60.2	172.17.0.2	OpenFlow	138	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE
106	4.978014781	172.17.0.2	10.0.60.2	OpenFlow	370	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
109	4.978434816	172.17.0.2	10.0.60.2	OpenFlow	386	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
112	5.101106466	10.0.60.2	172.17.0.2	OpenFlow	130	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER
113	5.101284141	172.17.0.2	10.0.60.2	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP
115	5.101375478	172.17.0.2	10.0.60.2	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP_DESC
117	5.101380022	172.17.0.2	10.0.60.2	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_METER

Figura 28: Mensajes capturados de la comunicación entre el switch y el controlador

Podemos ver en la Figura 28 que el controlador tiene la IP 10.0.60.2 y el switch tiene la IP 172.17.0.2. A continuación, en la Figura 29 podemos ver un diagrama de secuencia con los mensajes que se intercambian al iniciar la comunicación. En las siguientes secciones se procederá a explicar que es cada mensaje y que características y que campos tiene. Es importante comentar que hay veces que un mismo paquete analizado por Wireshark puede contener varios mensajes OpenFlow, por ejemplo en el caso del segundo paquete en la secuencia, que está formado por un OFPT_HELLO y un OFPT_FEATURES_REQUEST, pero son dos mensajes independientes.

Toda la información acerca de los mensajes OpenFlow, sus campos y sus características es extraída de la OpenFlow Switch Specification [5].

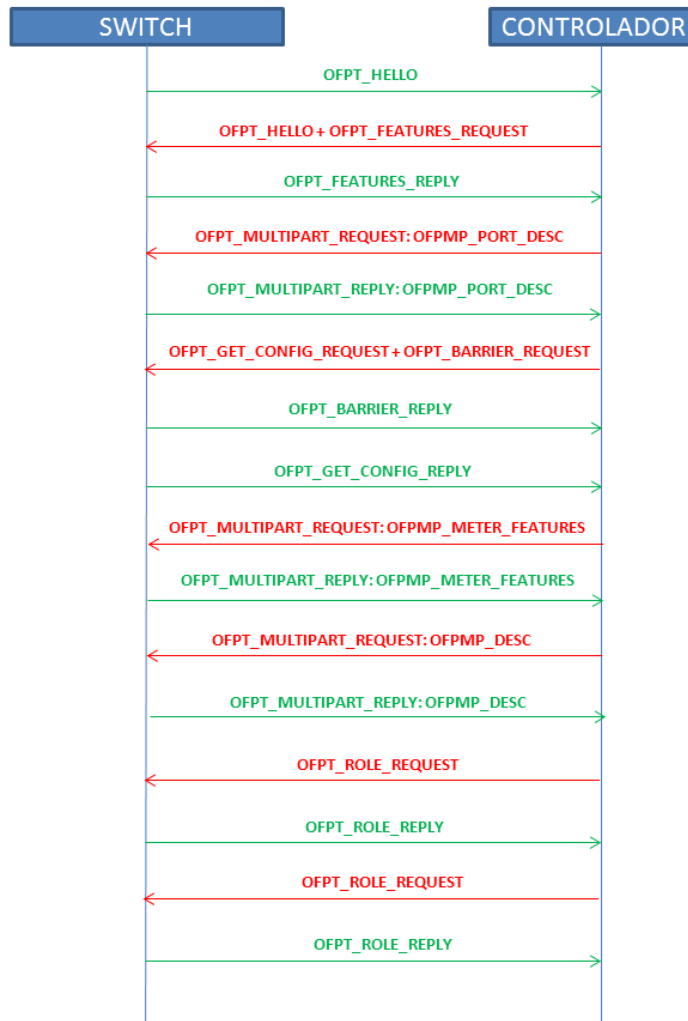


Figura 29: Inicio de la comunicación entre el switch y el controlador (mensajes OpenFlow)

5.2.1 Mensaje OFPT_HELLO

La comunicación empieza con un mensaje OFPT_HELLO enviado desde el switch. En la Figura 30 podemos ver el mensaje.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_HELLO (0)
  Length: 16
  Transaction ID: 10
  ▼ Element
    Type: OFPHET_VERSIONBITMAP (1)
    Length: 8
    Bitmap: 00000010
  
```

Figura 30: Características y campos del mensaje OFPT_HELLO

Los cuatro primeros campos que aparecen en el mensaje son comunes a todos los mensajes OpenFlow. En concreto, el campo *Version* indica la versión del protocolo

OpenFlow que se está utilizando, 1.3 en este caso. *Type* indica el tipo de mensaje enviado y *Length* es la longitud total del mensaje, en bytes. Por último, *Transaction ID* es el identificador de la transacción en el que la respuesta que dé el controlador a este mensaje enviado por el switch tendrá que tener el mismo *Transaction ID*.

El último campo, *Element*, es el más importante. *Type* es siempre OFPHET_VERSIONBITMAP. El campo *Bitmap* es el que indica las versiones de OpenFlow que soporta el switch. Si por ejemplo soportase la versión de OpenFlow 1.0 y 1.3, el campo *Bitmap* sería 0x00000012, ya que la versión 1.0 tiene el código 1 y la 1.3 el código 4, por lo que se pondrían a 1 los bits 1 y 4, dando 0x12. En este caso el campo *Version* se pondría a la versión más alta del protocolo OpenFlow que soporte, es decir, la 1.3. Ya que en la Figura 28 el campo *Bitmap* es 0x10, solo soporta OpenFlow 1.3. El controlador responde a este mensaje con otro OFPT_HELLO con *Bitmap* 0x00000072, es decir, que el controlador soporta las versiones OpenFlow 1.0, 1.3, 1.4 y 1.5. Como nuestro switch solo soporta 1.3, ésta será la versión del protocolo que se usará.

Cabe destacar que los mensajes OpenFlow se pueden agrupar en 3 categorías: mensajes asíncronos, mensajes simétricos y mensajes controlador/switch. OFPT_HELLO es del tipo simétrico.

5.2.2 Mensajes OFPT_FEATURES_REQUEST y OFPT_FEATURES_REPLY

Una vez que se ha acordado que versión del protocolo OpenFlow se va a usar, el controlador envía un mensaje OFPT_FEATURES_REQUEST para averiguar las capacidades básicas y características soportadas por el switch. En la Figura 31 podemos ver el contenido del mensaje OFPT_FEATURES_REQUEST, que solo tiene los campos comunes y en la Figura 32 el mensaje de respuesta que el switch da a este mensaje, esto es, el mensaje OFPT_FEATURES_REPLY.

```
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FEATURES_REQUEST (5)
  Length: 8
  Transaction ID: 4294967294
```

Figura 31: Características y campos del mensaje del tipo OFPT_FEATURES_REQUEST

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FEATURES_REPLY (6)
  Length: 32
  Transaction ID: 4294967294
  datapath_id: 0x0000000000000001
  n_buffers: 0
  n_tables: 254
  auxiliary_id: 0
  Pad: 0
  ▼ capabilities: 0x0000004f
    .....1 = OFPC_FLOW_STATS: True
    .....1. = OFPC_TABLE_STATS: True
    .....1.. = OFPC_PORT_STATS: True
    .....1... = OFPC_GROUP_STATS: True
    .....0. .... = OFPC_IP_REASM: False
    .....1.. .... = OFPC_QUEUE_STATS: True
    .....0 ..... = OFPC_PORT_BLOCKED: False
  Reserved: 0x00000000

```

Figura 32: Características y campos del mensaje OFPT_FEATURES_REPLY

Los campos específicos de este mensaje son:

- *datapath ID*: identifica unívocamente un *datapath*, que es el identificador del switch. Los 48 bits más bajos son la dirección MAC del switch, mientras que los 16 bits son establecidos por el vendedor.
- *n_buffers*: máximo número de paquetes que el switch puede almacenar en su buffer cuando éste manda paquetes al controlador usando el tipo de mensaje OpenFlow PACKET_IN.
- *n_tables*: número de tablas soportadas por el switch.
- *auxiliary_id*: identifica el tipo de conexión desde el switch al controlador. La conexión principal tendrá aquí un valor 0.
- *Pad*: es un campo de relleno. Los mensajes OpenFlow tienen que tener un tamaño que sea múltiplo de 8, si no lo tienen, se este campo se rellena con bytes hasta que el tamaño sea un múltiplo de 8.
- *capabilities*: capacidades que tiene el switch, las que están a 1 o en True, están activas. En nuestro caso soporta estadísticas de flows, tablas, puertos, grupos y colas, pero sin embargo no soporta reensamblaje de fragmentos IP ni el switch bloqueará puertos que formen un bucle.
- *Reserved*: campo reservado, sin utilidad en esta versión.

Estos mensajes son de tipo controlador/switch.

5.2.3 Mensajes **OFPT_MULTIPART_REQUEST** y **OFPT_MULTIPART_REPLY** tipo **OFPMMP_PORT_DESC**

El siguiente mensaje en la secuencia es un OFPT_MULTIPART_REQUEST. Este tipo de mensajes es enviado por controlador al switch para pedir información del estado del *datapath*. Los mensajes OFPT_MULTIPART tanto de REQUEST como de REPLY

tienen un campo que especifica qué tipo de información se pide o envía y cómo se interpreta el cuerpo del mensaje. La Figura 33 muestra los tipos de OFPT_MULTIPART.

TYPE	NUMBER	DESCRIPTION
OFPMMP_DESC	0	Description of this OpenFlow switch
OFPMMP_FLOW	1	Individual flow statistics
OFPMMP_AGGREGATE	2	Aggregate flow statistics
OFPMMP_TABLE	3	Flow table statistics
OFPMMP_PORT_STATS	4	Port statistics
OFPMMP_QUEUE	5	Queue statistics for a port
OFPMMP_GROUP	6	Group counter statistics
OFPMMP_GROUP_DESC	7	Group description
OFPMMP_GROUP_FEATURES	8	Group features
OFPMMP_METER	9	Meter statistics
OFPMMP_METER_CONFIG	10	Meter configuration
OFPMMP_METER_FEATURES	11	Meter features
OFPMMP_TABLE_FEATURES	12	Table features
OFPMMP_PORT_DESC	13	Port description
OFPMMP_EXPERIMENTER	0xFFFF	Experimenter extension

Figura 33: Tipos de mensajes OFPT_MULTIPART

El primer tipo de mensaje OFPT_MULTIPART_REQUEST es un OFPMMP_PORT_DESC, lo vemos en la Figura 34.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REQUEST (18)
  Length: 16
  Transaction ID: 4294967293
  Type: OFPMMP_PORT_DESC (13)
  ▼ Flags: 0x0000
    .....0 = OFPMMPF_REQ_MORE: 0x0
  Pad: 00000000

```

Figura 34: Mensaje OFPT_MULTIPART_REQUEST tipo OFPMMP_PORT_DESC

El campo *Flags* del mensaje tiene un parámetro que es OFPMMPF_REQ_MORE. Si vale 0 significa que no hay más peticiones o respuestas de este tipo. Al ser de tipo OFPMMP_PORT_DESC, el switch le envía como respuesta mediante un mensaje OFPT_MULTIPART_REPLY de tipo OFPMMP_PORT_DESC la descripción de los puertos que tiene. La Figura 35 muestra la apariencia de este mensaje.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 208
  Transaction ID: 4294967293
  Type: OFPMP_PORT_DESC (13)
  ▶ Flags: 0x0000
  Pad: 00000000
  ▶ Port
  ▶ Port
  ▶ Port

```

Figura 35: Características y campos principales del mensaje OFPT_MULTIPART_REPLY tipo OFPMP_PORT_DESC

El switch en nuestro caso tiene 3 puertos, 2 de ellos físicos y un puerto local reservado. En la Figura 36 se muestran los campos por los que está formado un puerto.

```

▼ Port
  Port no: 1
  Pad: 00000000
  Hw addr: 26:8f:82:81:86:65 (26:8f:82:81:86:65)
  Pad: 0000
  Name: s1-eth1
  ▼ Config: 0x00000000
    .....0 = OFPPC_PORT_DOWN: False
    .....0.. = OFPPC_NO_RECV: False
    .....0. .... = OFPPC_NO_FWD: False
    .....0.. .... = OFPPC_NO_PACKET_IN: False
  ▼ State: 0x00000004
    .....0 = OFPPS_LINK_DOWN: False
    .....0. .... = OFPPS_BLOCKED: False
    .....1.. .... = OFPPS_LIVE: True
  ▼ Current: 0x00000840
    .....0 = OFPPF_10MB_HD: False
    .....0.. = OFPPF_10MB_FD: False
    .....0.. = OFPPF_100MB_HD: False
    .....0... = OFPPF_100MB_FD: False
    .....0.... = OFPPF_1GB_HD: False
    .....0..... = OFPPF_1GB_FD: False
    .....1..... = OFPPF_10_GB_FD: True
    .....0..... = OFPPF_40GB_FD: False
    .....0..... = OFPPF_100_GB_FD: False
    .....0..... = OFPPF_1TB_FD: False
    .....0..... = OFPPF_OTHER: False
    .....1..... = OFPPF_COPPER: True
    .....0..... = OFPPF_FIBER: False
    .....0..... = OFPPF_AUTONEG: False
    .....0..... = OFPPF_PAUSE: False
    .....0..... = OFPPF_PAUSE_ASYM: False
  ▶ Advertised: 0x00000000
  ▶ Supported: 0x00000000
  ▶ Peer: 0x00000000
  Curr speed: 10000000
  Max speed: 0

```

Figura 36: Características y campos principales de un puerto

Los campos de este mensaje son los siguientes:

- *Port no*: número de puerto que identifica unívocamente el puerto en el switch. Si vale OFPP_LOCAL significa que es el puerto local de OpenFlow.
- *Hw addr*: dirección MAC del puerto.
- *Name*: nombre del puerto.

-
- *Config*: son ajustes administrativos del puerto y contiene los siguientes campos:
 - OFPPC_PORT_DOWN: si está a 1, el puerto está apagado administrativamente.
 - OFPPC_NO_RECV: si está a 1, desecha todos los paquetes que recibe.
 - OFPPC_NO_FWD: si está a 1, desecha todos los paquetes que son enviados a este puerto.
 - OFPPC_NO_PACKET_IN: si está a 1, no envía mensajes desde este puerto.
 - *State*: estado actual del puerto físico, que está indicado como:
 - OFPPS_LINK_DOWN: si está a 1, la capa física no está presente.
 - OFPPS_BLOCKED: si está a 1, el puerto está bloqueado.
 - OFPPS_LIVE: si está a 1, el puerto está vivo para el *Fast Failover Group*.
 - *Current, Advertised, Supported y Peer*: indican los modos de conexión (velocidad y modo full-duplex o semi-duplex), tipo de cable (cobre o fibra) y otras características de la conexión. En nuestro caso es de cobre a 10GB full-duplex.
 - *Curr speed y Max speed*: tasa actual y tasa máxima de bits del enlace en Kbps.

Los mensajes OFPT_MULTIPART son de tipo controlador/switch.

5.2.4 Mensajes OFPT_GET_CONFIG_REQUEST y OFPT_GET_CONFIG_REPLY

A continuación, en la secuencia de la comunicación, se envía un paquete formado por dos mensajes, un OFPT_GET_CONFIG_REQUEST y un OFPT_BARRIER_REQUEST. Del segundo paquete hablaremos en el siguiente apartado. El controlador envía un OFPT_GET_CONFIG_REQUEST para pedir parámetros de configuración del switch. Los campos de este mensaje son los comunes a todos los mensajes OpenFlow. El switch le responde con un OFPT_GET_CONFIG_REPLY. Se muestran las características y campos de este mensaje en la Figura 37.

```
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_GET_CONFIG_REPLY (8)
  Length: 12
  Transaction ID: 4294967291
  ▼ Flags: 0x0000
    .... ..00 = IP Fragments: OFPC_FRAG_NORMAL (0)
  Miss send length: 128
```

Figura 37: Características y campos principales del mensaje OFPT_GET_CONFIG_REPLY

Este mensaje tiene dos campos, el campo *Flags* que indica si los fragmentos IP tiene que ser tratados normalmente, desechados o reensamblados, y el campo *Miss send length*, que define el número de bytes de cada paquete enviado al controlador mediante el pipeline de OpenFlow cuando no se usa una acción de tipo Output hacia el puerto lógico OFPP_CONTROLLER. Los mensajes OFPT_GET_CONFIG son de tipo controlador/switch.

5.2.5 Mensajes OFPT_BARRIER_REQUEST y OFPT_BARRIER_REPLY

Cuando el controlador quiere asegurarse que los mensajes enviados anteriormente han sido procesados correctamente, envía un OFPT_BARRIER_REQUEST. Cuando el switch recibe este mensaje, tiene que acabar de procesar todos los mensajes previos antes de procesar mensajes posteriores al OFPT_BARRIER_REQUEST recibido. En ese momento envía al controlador un OFPT_BARRIER_REPLY. Los campos de estos mensajes son los comunes a todos los mensajes OpenFlow. Los mensajes OFPT_BARRIER son de tipo controlador/switch.

5.2.6 Mensajes OFPT_MULTIPART_REQUEST y OFPT_MULTIPART_REPLY tipo OFPMP_METER_FEATURES y OFPMP_DESC

Los dos siguientes mensajes que el controlador envía al switch son también del tipo OFPT_MULTIPART_REQUEST. La estructura del mensaje es igual para todos los OFPT_MULTIPART_REQUEST, pero cambiando el tipo de datos que se solicitan.

El primero de los dos mensajes es del tipo OFPMP_METER_FEATURES, donde el controlador espera recibir las características del subsistema de meters. En la Figura 38 podemos ver la estructura del mensaje. Los campos del mensaje son:

- *Max meters*: máximo número de meters.
- *Band types*: tipos de band que soporta. DROP indica que descarta los mensajes y DSCP_REMARK remarcar DSCP (*Differentiated Services Code Point*) en la cabecera IP.

- *Capabilities*: Banderas de configuración que soporta el switch para los meters. Dentro de este campo se definen una serie de campos internos, si están a True significa que el switch soporta esa bandera:
 - OFMF_KBPS: tasa en Kbps.
 - OFMF_PKTPTS: tasa en paquetes por segundo.
 - OFMF_BURST: tamaño de *burst*.
 - OFMF_STATS: recoger estadísticas.
- *Max bands*: máximo número de bandas (*band*) por meter.
- *Max colors*: valor máximo de color.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 32
  Transaction ID: 4294967290
  Type: OFPMP_METER_FEATURES (11)
  ▶ Flags: 0x0000
  Pad: 00000000
  Max meters: 65536
  ▼ Band types: 0x00000002
    .....1 = OFPMBT_DROP: True
    .....0.. = OFPMBT_DSCP_REMARK: False
  ▼ Capabilities: 0x0000000f
    .....1 = OFPMF_KBPS: True
    .....1.. = OFPMF_PKTPTS: True
    .....1.. = OFPMF_BURST: True
    .....1... = OFPMF_STATS: True
  Max bands: 8
  Max colors: 0
  Pad: 0000

```

Figura 38: Características y campos principales del mensaje OFPT_MULTIPART_REPLY del tipo OFPMP_METER_FEATURES

El segundo de los dos mensajes es del tipo OFPMP_DESC, utilizado por el controlador para saber la descripción del switch. En la Figura 39 se muestra el mensaje con el que el switch contesta a la petición del controlador.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 1072
  Transaction ID: 4294967289
  Type: OFPMP_DESC (0)
  ▶ Flags: 0x0000
  Pad: 00000000
  Manufacturer desc.: Nicira, Inc.
  Hardware desc.: Open vSwitch
  Software desc.: 2.9.5
  Serial no.: None
  Datapath desc.: None

```

Figura 39: Características y campos principales del mensaje OFPT_MULTIPART_REPLY del tipo OFPMP_DESC

Los campos de este mensaje describen el fabricante, el hardware, la versión del software, el número serial y la descripción del *datapath*.

5.2.7 Mensajes *OFPT_ROLE_REQUEST* y *OFPT_ROLE_REPLY*

El último tipo de mensaje que se envía al iniciar la comunicación es un *OFPT_ROLE_REQUEST*, que es enviado por el controlador cuando quiere cambiar su rol. El rol del controlador es importante ya que podría haber más de un controlador gestionando la red. En la Figura 40 se muestra un mensaje *OFPT_ROLE_REQUEST*. En el campo *Role* del mensaje, el controlador indica que rol quiere asumir, y existen cuatro tipos:

- *OFPCR_ROLE_NOCHANGE*: cuando el controlador no va a cambiar su rol actual. Este tipo de rol se usa cuando el controlador necesite saber su propio rol.
- *OFPCR_ROLE_EQUAL*: rol por defecto, que significa que tiene acceso de lectura y escritura en los todos los switches.
- *OFPCR_ROLE_MASTER*: cuando hay varios controladores en la red, solo uno puede tener este rol, todos los demás tendrán el rol *OFPCR_ROLE_SLAVE*. Con este rol el controlador tiene acceso de lectura y escritura en los switches.
- *OFPCR_ROLE_SLAVE*: campo que indica el rol que tienen los controladores que no son *OFPCR_ROLE_MASTER*. Con este rol el controlador solo tiene acceso de lectura en los switches.

```
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_ROLE_REQUEST (24)
  Length: 24
  Transaction ID: 0
  Role: OFPCR_ROLE_MASTER (0x00000002)
  Pad: 00000000
  Generation ID: 0x0000000000000000
```

Figura 40: Campos y características del mensaje *OFPT_ROLE_REQUEST*

El switch responde a este mensaje con un *OFPT_ROLE_REPLY* con los mismos campos que la petición. El campo *Role* de este mensaje indica el rol actual del controlador. En la Figura 41 se muestran las características y campos de este mensaje.

```
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_ROLE_REPLY (25)
  Length: 24
  Transaction ID: 0
  Role: OFPCR_ROLE_MASTER (0x00000002)
  Pad: 00000000
  Generation ID: 0x0000000000000000
```

Figura 41: Campos y características del mensaje *OFPT_ROLE_REPLY*

Tanto la petición como la respuesta llevan un campo llamado *Generation ID*. Este campo se utiliza cuando un controlador quiere cambiar de rol *Master* a *Slave* o de *Slave* a *Master*. El switch que recibe el mensaje *OFPT_ROLE_REQUEST* extrae este campo y si

es mayor que el *Generation ID* que se recibió en una petición anterior, se envía un OFPT_ROLE_REPLY con el mismo *Generation ID* y el controlador cambia de rol. Si el *Generation ID* por el contrario es menor que el *Generation ID* que se recibió en una petición anterior, está obsoleto y se descarta. En vez de enviar un OFPT_ROLE_REPLY, se envía un mensaje de error. Con esto se evita que haya dos controladores con el rol *Master* al mismo tiempo. Los mensajes OFPT_ROLE son de tipo controlador al switch.

A partir de este punto, todos los mensajes que se dan al inicio de la comunicación se han enviado y se han recibido sus repuestas correspondientes. A partir de ahora se explicarán los mensajes de tipo asíncrono que se enviarán cuando se quieren crear flujos de configuración y sus meters asociados.

5.2.8 Mensaje OFPT_FLOW_MOD

Cuando el controlador quiere realizar una modificación a la tabla de los flows, envía este tipo de mensaje. Con él se añaden, modifican o eliminan los flows asociados a un switch concreto. Aunque este mensaje puede ser enviado por el controlador en cualquier momento, nuestro controlador, ONOS, envía tres flows por defecto cada vez que un switch se conecta a él. Vamos a analizar uno de estos mensajes. En la Figura 42 se muestra el contenido de un mensaje OFPT_FLOW_MOD.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FLOW_MOD (14)
  Length: 96
  Transaction ID: 0
  Cookie: 0x00010000ea6f4b8e
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 0
  Priority: 40000
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
  ▼ Flags: 0x0001
    .... .1 = Send flow removed: True
    .... .0 = Check overlap: False
    .... .0 = Reset counts: False
    .... 0.. = Don't count packets: False
    .... ..0 = Don't count bytes: False
  Pad: 0000
  ▼ Match
    Type: OFPMT_OXM (1)
    Length: 10
    ▼ OXM field
      Class: OFPXM_OPENFLOW_BASIC (0x8000)
      0000 101. = Field: OFPXM_OFB_ETH_TYPE (5)
      .... .0 = Has mask: False
      Length: 2
      Value: ARP (0x0806)
      Pad: 000000000000
    ▼ Instruction
      Type: OFPIT_CLEAR_ACTIONS (5)
      Length: 8
      Pad: 00000000
    ▼ Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 24
      Pad: 00000000
      ▼ Action
        Type: OFPAT_OUTPUT (0)
        Length: 16
        Port: OFPP_CONTROLLER (4294967293)
        Max length: OFPML_NO_BUFFER (65535)
        Pad: 000000000000
  
```

Figura 42: Características y campos del mensaje OFPT_FLOW_MOD

Los campos específicos de este mensaje y sus valores se especifican a continuación:

- *Cookie*: identificador del flow.
- *Cookie mask*: si el valor es distinto de 0, se usa junto al campo *Cookie* para restringir coincidencias al modificar o eliminar un flow.
- *Table ID*: identificador de la tabla donde se guarda el flow.
- *Command*: acción que se hace con el flow que se está creando, en concreto, se pueden definir el siguiente conjunto de acciones:
 - OFPFC_ADD: se añade un nuevo flow.
 - OFPFC_MODIFY: se modifican todos los flows coincidentes.
 - OFPFC_MODIFY_STRICT: se modifican todos los flows coincidentes siempre que tengan el mismo campo de *Match* y la misma prioridad.
 - OFPFC_DELETE: se eliminan todos los flows coincidentes.
 - OFPFC_DELETE_STRICT: se eliminan todos los flows coincidentes siempre que tengan el mismo campo de *Match* y la misma prioridad.
- *Idle timeout*: Si el valor es distinto de 0, el flow será eliminado si no se recibe ningún paquete que haga *Match* con este flow en el número máximo de segundos especificados en este campo.
- *Hard timeout*: Si el valor es distinto de 0, el flow será eliminado al transcurrir el número de segundos especificado en este campo.
- *Priority*: prioridad del flow en su tabla. Cuanto más alta, más prioridad.
- *Buffer ID*: identificador referido a un paquete almacenado en el switch y enviado al controlador mediante un mensaje PACKET_IN.
- *Out port* y *Out group*: campos opcionales para filtrar el alcance de los mensajes OFPFC_DELETE y OFPFC_DELETE_STRICT por puerto de salida o grupo de salida.
- *Flags*: opciones de configuración del flow. En concreto se definen:
 - OFPFF_SEND_FLOW_REM: si el flow expira o está eliminado.
 - OFPFF_CHECK_OVERLAP: comprueba las entradas superpuestas.
 - OFPFF_RESET_COUNTS: reinicia los contadores de paquetes y bytes.
 - OFPFF_NO_PKT_COUNT: para no llevar la cuenta de paquetes.
 - OFPFF_NO_BYT_COUNT: para no llevar la cuenta de bytes.

Los dos campos restantes son los más importantes. El campo *Match* especifica qué características debe de tener un paquete para que se apliquen sobre él las instrucciones

definidas en el campo *Instructions*. El tipo del campo *Match* siempre es OFPMT_OXM. El campo *Length* indica la longitud de los campos *OXM field* más la longitud del campo *Type* y *Length*, no la longitud del campo *Match*; es decir, que si la longitud no es un múltiplo de ocho, se tendrá un campo *Pad* para rellenar con ceros hasta que la longitud del campo *Match* sea múltiplo de ocho.

Dentro del campo *Match* podemos encontrar uno o varios subcampos *OXM field*, cada uno define una regla que debe cumplir el paquete si quiere ser procesado por el flow. Cada subcampo *OXM field* tiene a su vez 5 campos:

- *Class*: Clase de *Match*. Para nosotros siempre será la clase OFPXMC_OPENFLOW_BASIC.
- *Field*: identifica el tipo del *Match* dentro de la clase. La Figura 43 nos muestra los tipos. En nuestro se van a utilizar los tipos: 0, 3, 4, 5, 6, 7, 11 y 12.
- *Has mask*: si está a 1, significa que hay un sub campo más que indica la máscara, utilizada en las direcciones IP.
- *Length*: longitud del sub campo *Value*.
- *Value*: valor del *OXM field*, que puede ser una IP, una dirección MAC, una VLAN, etc...y que depende del tipo del *Match*.

```

/* OXM Flow match field types for OpenFlow basic class. */
enum oxm_ofb_match_fields {
    OFPXMT_OFB_IN_PORT          = 0, /* Switch input port. */
    OFPXMT_OFB_IN_PHY_PORT     = 1, /* Switch physical input port. */
    OFPXMT_OFB_METADATA        = 2, /* Metadata passed between tables. */
    OFPXMT_OFB_ETH_DST         = 3, /* Ethernet destination address. */
    OFPXMT_OFB_ETH_SRC         = 4, /* Ethernet source address. */
    OFPXMT_OFB_ETH_TYPE        = 5, /* Ethernet frame type. */
    OFPXMT_OFB_VLAN_VID        = 6, /* VLAN id. */
    OFPXMT_OFB_VLAN_PCP        = 7, /* VLAN priority. */
    OFPXMT_OFB_IP_DSCP         = 8, /* IP DSCP (6 bits in ToS field). */
    OFPXMT_OFB_IP_ECN          = 9, /* IP ECN (2 bits in ToS field). */
    OFPXMT_OFB_IP_PROTO        = 10, /* IP protocol. */
    OFPXMT_OFB_IPV4_SRC        = 11, /* IPv4 source address. */
    OFPXMT_OFB_IPV4_DST        = 12, /* IPv4 destination address. */
    OFPXMT_OFB_TCP_SRC         = 13, /* TCP source port. */
    OFPXMT_OFB_TCP_DST         = 14, /* TCP destination port. */
    OFPXMT_OFB_UDP_SRC         = 15, /* UDP source port. */
    OFPXMT_OFB_UDP_DST         = 16, /* UDP destination port. */
    OFPXMT_OFB_SCTP_SRC        = 17, /* SCTP source port. */
    OFPXMT_OFB_SCTP_DST        = 18, /* SCTP destination port. */
    OFPXMT_OFB_ICMPV4_TYPE     = 19, /* ICMP type. */
    OFPXMT_OFB_ICMPV4_CODE     = 20, /* ICMP code. */
    OFPXMT_OFB_ARP_OP          = 21, /* ARP opcode. */
    OFPXMT_OFB_ARP_SPA         = 22, /* ARP source IPv4 address. */
    OFPXMT_OFB_ARP_TPA         = 23, /* ARP target IPv4 address. */
    OFPXMT_OFB_ARP_SHA         = 24, /* ARP source hardware address. */
    OFPXMT_OFB_ARP_THA         = 25, /* ARP target hardware address. */
    OFPXMT_OFB_IPV6_SRC        = 26, /* IPv6 source address. */
    OFPXMT_OFB_IPV6_DST        = 27, /* IPv6 destination address. */
    OFPXMT_OFB_IPV6_FLABEL     = 28, /* IPv6 Flow Label */
    OFPXMT_OFB_ICMPV6_TYPE     = 29, /* ICMPv6 type. */
    OFPXMT_OFB_ICMPV6_CODE     = 30, /* ICMPv6 code. */
    OFPXMT_OFB_IPV6_ND_TARGET  = 31, /* Target address for ND. */
    OFPXMT_OFB_IPV6_ND_SLL     = 32, /* Source link-layer for ND. */
    OFPXMT_OFB_IPV6_ND_TLL     = 33, /* Target link-layer for ND. */
    OFPXMT_OFB_MPLS_LABEL      = 34, /* MPLS label. */
    OFPXMT_OFB_MPLS_TC         = 35, /* MPLS TC. */
    OFPXMT_OFB_MPLS_BOS        = 36, /* MPLS BoS bit. */
    OFPXMT_OFB_PBB_ISID        = 37, /* PBB I-SID. */
    OFPXMT_OFB_TUNNEL_ID       = 38, /* Logical Port Metadata. */
    OFPXMT_OFB_IPV6_EXTHDR     = 39, /* IPv6 Extension Header pseudo-field */
};

```

Figura 43: Valores del campo OXM Field

El último campo del mensaje es el campo *Instructions*, que determina que se debe hacer cuando un paquete hace *Match* con el flow. Puede haber una o varias instrucciones por flow. Los distintos tipos de instrucciones se observan en la Figura 44.

```

enum ofp_instruction_type {
    OFPIT_GOTO_TABLE = 1, /* Setup the next table in the lookup
                           pipeline */
    OFPIT_WRITE_METADATA = 2, /* Setup the metadata field for use later in
                               pipeline */
    OFPIT_WRITE_ACTIONS = 3, /* Write the action(s) onto the datapath action
                              set */
    OFPIT_APPLY_ACTIONS = 4, /* Applies the action(s) immediately */
    OFPIT_CLEAR_ACTIONS = 5, /* Clears all actions from the datapath
                              action set */
    OFPIT_METER = 6, /* Apply meter (rate limiter) */
    OFPIT_EXPERIMENTER = 0xFFFF /* Experimenter instruction */
};

```

Figura 44: Tipos de instrucciones

En nuestro caso solo vamos a utilizar las instrucciones `OFPIT_CLEAR_ACTIONS`, `OFPIT_APPLY_ACTIONS` y `OFPIT_METER`. La instrucción `OFPIT_CLEAR_ACTIONS` borra todas las acciones del conjunto de acciones del

datapath. La instrucción del tipo OFPIT_APPLY_ACTIONS aplica las acciones especificadas por el sub campo *Action*. Este campo puede tener distintos valores, dependiendo del tipo de acción a realizar, tal y como se observa en la Figura 45.

```
enum ofp_action_type {
    OFPAT_OUTPUT      = 0, /* Output to switch port. */
    OFPAT_COPY_TTL_OUT = 11, /* Copy TTL "outwards" -- from next-to-outermost
                             to outermost */
    OFPAT_COPY_TTL_IN  = 12, /* Copy TTL "inwards" -- from outermost to
                             next-to-outermost */
    OFPAT_SET_MPLS_TTL = 15, /* MPLS TTL */
    OFPAT_DEC_MPLS_TTL = 16, /* Decrement MPLS TTL */

    OFPAT_PUSH_VLAN   = 17, /* Push a new VLAN tag */
    OFPAT_POP_VLAN    = 18, /* Pop the outer VLAN tag */
    OFPAT_PUSH_MPLS   = 19, /* Push a new MPLS tag */
    OFPAT_POP_MPLS    = 20, /* Pop the outer MPLS tag */
    OFPAT_SET_QUEUE   = 21, /* Set queue id when outputting to a port */
    OFPAT_GROUP       = 22, /* Apply group. */
    OFPAT_SET_NW_TTL  = 23, /* IP TTL. */
    OFPAT_DEC_NW_TTL  = 24, /* Decrement IP TTL. */
    OFPAT_SET_FIELD   = 25, /* Set a header field using OXM TLV format. */
    OFPAT_PUSH_PBB    = 26, /* Push a new PBB service tag (I-TAG) */
    OFPAT_POP_PBB     = 27, /* Pop the outer PBB service tag (I-TAG) */
    OFPAT_EXPERIMENTER = 0xffff
};
```

Figura 45: Tipos de acciones

En nuestro caso solo vamos a considerar el tipo de acción OFPAT_OUTPUT (valor 0 en la Figura 45). Dentro del campo *Action* aparece el campo *Port*, que indica por qué puerto saldrá el paquete. Si su valor es OFPP_CONTROLLER significa que se envía al controlador. El campo *Max length* indica la cantidad máxima de datos de un paquete que pueden ser enviados si el puerto es OFPP_CONTROLLER.

Por otro lado, la instrucción del tipo OFPIT_METER no tiene ninguna acción asociada, es decir, tiene un campo *Meter ID* que identifica el id del meter asociado a ese flow. Los mensajes OFPT_FLOW_MOD son de tipo controlador/switch.

5.2.9 Mensaje OFPT_METER_MOD

Cuando el controlador quiere realizar una modificación a la tabla de los meters, envía este tipo de mensaje. Con él se añaden, modifican o eliminan los meters. En la Figura 46 se muestra el contenido de un mensaje OFPT_METER_MOD. Los campos de este mensaje son:

- *Command*: acción que se realiza con ese meter, puede ser añadirlo, modificarlo o borrarlo.
- *Flags*: Banderas de configuración del meter. Se definieron ya en el mensaje OFPMP_METER_FEATURES.
- *Meter ID*: identificador del meter.
- *Meter band*: Un meter tiene desde una a varias bandas (*bands*) asociadas, tales como:

- *Type*: tipo de meter, puede ser DROP o REMARK.
- *Rate*: Tasa para desechar o remarcar paquetes.
- *Burst size*: Tamaño de burst.
- *Precendence level*: Campo opcional, usado solo si el tipo es REMARK. Número de nivel de precedencia para restar.

Los mensajes OFPT_METER_MOD son de tipo controlador al switch.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_METER_MOD (29)
  Length: 32
  Transaction ID: 1
  Command: OFPMC_ADD (0)
  ▼ Flags: 0x00000005
    .....1 = OFPMF_KBPS: True
    .....0 = OFPMF_PKTPS: False
    .....1 = OFPMF_BURST: True
    .....0 = OFPMF_STATS: False
  Meter ID: 1
  ▼ Meter band
    Type: OFPMBT_DROP (1)
    Length: 16
    Rate: 204800
    Burst size: 0
    Pad: 00000000

```

Figura 46: Características y contenidos del mensaje OFPT_METER_MOD

5.2.10 Otros mensajes OFPT_MULTIPART

Por último, hay otros seis tipos de OFPT_MULTIPART que son necesario comentar. Nuestro controlador, ONOS, cada cierto tiempo realiza peticiones para saber las estadísticas de los flows, meters, tablas, puertos, grupos y descripción de los grupos. Estos se corresponden con los tipos OFPMP_FLOW, OFPMP_METER, OFPMP_TABLE, OFPMP_PORT, OFPMP_GROUP y OFPMP_GROUP_DESC. Para el caso de los flows por ejemplo, en el mensaje OFPMP_MULTIPART_REPLY se envía cada flow junto con algunos campos más con la duración del flow en la tabla y el número de paquetes y bytes procesados por cada flow. En el caso de los meters, se envía solo la duración y el número de paquetes y bytes procesados por cada meter, sin el tipo ni las banderas del meter. Esto se muestra en la Figura 47.


```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 128
  Transaction ID: 0
  Type: OFPMP_METER (9)
  ▶ Flags: 0x0000
  Pad: 00000000
  ▶ Meter stats
  ▼ Meter stats
    Meter ID: 1
    Length: 56
    Pad: 000000000000
    Flow count: 0
    Packet in count: 0
    Byte in count: 10
    Duration sec: 10
    Duration nsec: 1000
  ▼ Meter band stats
    Packet count: 288
    Byte count: 288

```

Figura 47: Mensaje OFPT_MULTIPART_REPLY tipo OFPMP_METER

5.2.11 Mensajes OFPT_ECHO_REQUEST y OFPT_ECHO_REPLY

Este mensaje es enviado por el controlador al switch cuando quiere comprobar que el switch sigue activo. El mensaje solo tiene los campos comunes a todos los mensajes OpenFlow. El switch debe responder con un OFPT_ECHO_REPLY para indicar que sigue vivo. Los mensajes OFPT_ECHO son de tipo simétrico.

5.3 Diseño del Agente OpenFlow - CLI en la red GPON

Una vez realizado el análisis de los mensajes OpenFlow que se intercambia el controlador ONOS y el switch de Mininet, vamos a comenzar la fase de diseño del agente OpenFlow. Para el diseño del agente vamos a tener como base un agente ya desarrollado por un exalumno en cuyo Trabajo Final de Grado [41] creó en el 2017 un agente compatible con el controlador SDN OpenDayLigth. Nuestra tarea va a ser extender las funcionalidades de dicho agente mediante la creación de nuevos métodos y proporcionando respuesta a más tipos de mensajes. También se van a actualizar las librerías utilizadas hasta la última versión de las mismas y se va a permitir adaptándolo, la compatibilidad con nuestro controlador, ONOS.

El objetivo principal del agente es capturar los mensajes procedentes del controlador SDN ONOS y darles una respuesta como si de un switch virtual se tratase. Pero el agente además captura los flows enviados por ONOS que contienen órdenes en lenguaje OpenFlow para configurar la red GPON y los servicios de los usuarios conectados a las ONTs/ONUs a través del OLT, que es el elemento principal de gestión de una red PON, en nuestro caso de una red GPON. De esta manera, el agente extraerá los parámetros de configuración que le envíe ONOS a través de mensajes OpenFlow e interactuará con la API a la red GPON en el estándar propio de la dicha red. La API de la red GPON crea los servicios de Internet y/o vídeo contratados por los usuarios de la red (conectados a

ONTs/ONUs) y ha sido previamente desarrollada en trabajos de investigación anteriores. Por lo tanto, vamos a desarrollar un agente en OpenFlow capaz de transformar órdenes enviadas por un controlador SDN mediante el protocolo OpenFlow al lenguaje interno de cada red PON. Lo que cambiaría al gestionar redes GPON/PON de diferentes fabricantes sería la API interna de cada red, que ésta sí es dependiente del fabricante y sería necesario programar una pequeña capa de adaptación entre el agente SDN desarrollado en este trabajo y la API de gestión particular de cada red. La característica más importante de este agente es que puede ser desplegado a nivel software en cualquier OLT de cualquier fabricante ya que es una capa software implementada como un puente entre un controlador SDN y una red GPON/PON. De este modo, un mismo controlador SDN, ONOS en este caso, podría controlar diferentes redes PON de diferentes fabricantes usando el mismo agente OpenFlow. La Figura 48 muestra el escenario de red flexible con nuestro agente OpenFlow en un entorno de gestión SDN global donde se observa que un mismo controlador SDN puede interactuar con diferentes redes PON usando el mismo agente OpenFlow que interactúa a su vez con la capa CLI de cada OLT particular, es decir, con la API de gestión y configuración de cada red PON.

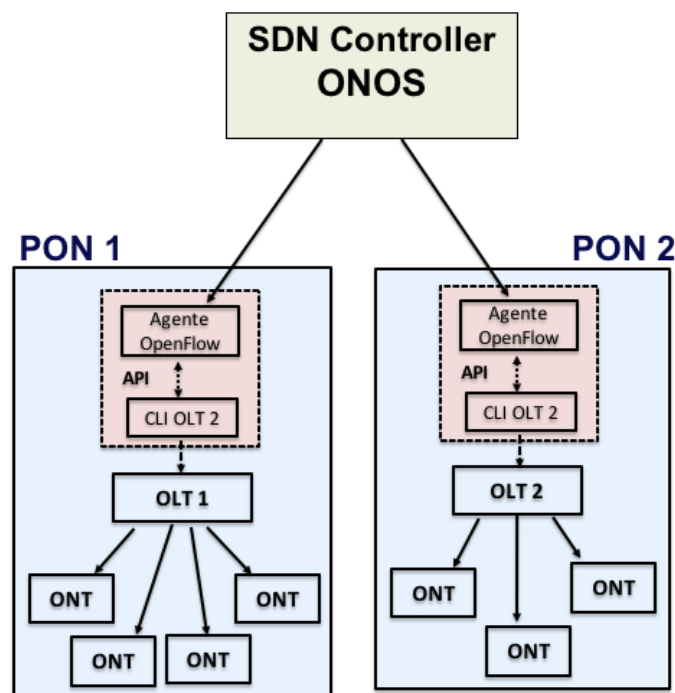


Figura 48: Diseño del agente OpenFlow para gestionar múltiples redes PON de diferentes fabricantes

Para el diseño del agente hemos utilizado el patrón de diseño Fachada (*Facade*). Una Fachada es un objeto *front-end* que es el único punto de entrada para los servicios de un subsistema, la implementación y otros componentes del subsistema son privados y no pueden verlos los componentes externos. La Fachada proporciona variaciones protegidas frente a los cambios en las implementaciones de un subsistema [42]. Este patrón proporciona una interfaz simple a un subsistema o conjunto de subsistemas complejos y

permite la independencia y portabilidad de los subsistemas [43]. El patrón está compuesto por tres elementos: el cliente, que interactúa con los subsistemas a través de la Fachada, la Fachada como interfaz para los subsistemas y los subsistemas. Para nosotros, ONOS será el cliente, el agente OpenFlow la Fachada y la API será uno de los subsistemas. Si en el futuro nuestro agente fuera a ser utilizado en una red distinta, será necesaria la programación de otra API que pueda interactuar con esa red. Esta segunda API sería un nuevo subsistema, de tal manera que desde ONOS podamos configurar las dos redes usando el mismo agente OpenFlow. En la Figura 49 se muestra el diagrama de clases utilizando este patrón de diseño.

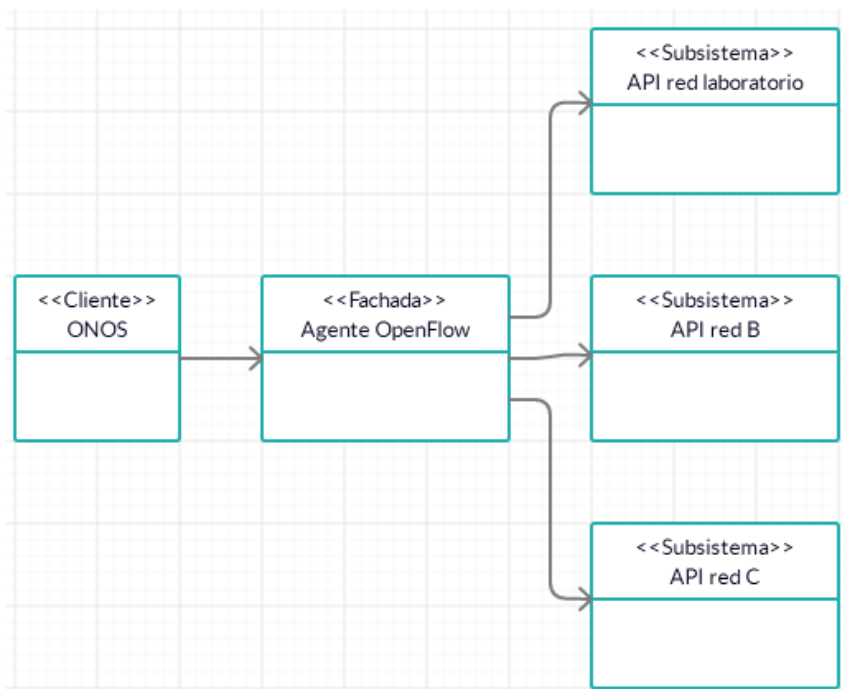


Figura 49: Diagrama de clases utilizando el patrón de diseño Fachada

Para la programación de los mensajes OpenFlow se va a utilizar la librería de Python `python-openflow`, desarrollada por Kytos [11], que permite la creación de todos los tipos de mensaje OpenFlow versión 1.3 o 1.0 de forma cómoda. La versión de la librería `python-openflow` utilizada en el código base de la primera versión fue la 2017.1b1, mientras que nosotros vamos a utilizar la última (2020.1b3), con el objetivo de tener el código lo más actualizado posible. Para instalar la librería `python-openflow` necesitamos tener instalado Python3 previamente. Nosotros tenemos Python 3.6.9. Los pasos a seguir son, primero clonar el repositorio, después instalar la librería en Python3 mediante el instalador de paquetes de Python3 `pip3` y por último ejecutar el archivo `setup.py`. Esta explicación se resume con los siguientes comandos:

```

git clone https://github.com/kytos/python-openflow.git
cd python-openflow
sudo pip3 install python-openflow
  
```

sudo python3 setup.py install

Cabe destacar que esta versión de la librería es compatible con la versión de nuestro controlador ONOS, esto es, la versión 2.4.0.

5.3.1 Mejoras realizadas al agente

El agente OpenFlow de partida tenía el siguiente comportamiento. En primer lugar, se conectaba al controlador SDN OpenDayLight y procesaba y daba respuesta a los mensajes OpenFlow de inicio de la comunicación. Esto se hacía mediante un bucle infinito, mientras llegasen mensajes. Después, ante la llegada de un flow y un meter, se guardaban solo los parámetros necesarios para configurar un servicio, y en ese momento el bucle se rompía y se pasaba a crear el servicio correspondiente. A continuación se pedía al usuario la dirección MAC de la ONT donde se iba a crear el servicio. Entonces el servicio se crea para esa ONT y con los parámetros extraídos previamente. Una vez que el servicio se creaba y se volcaba sobre la ONT/ONU, el agente terminaba. Nosotros hemos realizado las siguientes modificaciones y mejoras:

- Actualización de librerías: actualizamos la librería python-openflow, responsable de la creación de los mensajes OpenFlow en Python, a la última versión.
- Procesamiento y respuesta de más tipos de mensajes: el agente original era capaz de procesar y dar respuesta a los mensajes de tipo OFPT_FEATURES_REQUEST, OFPT_BARRIER_REQUEST, OFPT_ROLE_REQUEST y OFPT_MULTIPART_REQUEST de tipo OFPMP_DESC. Nosotros además de esos tipos de mensajes, añadimos los tipos OFPT_SET_CONFIG, OFPT_GET_CONFIG_REQUEST, OFPT_ECHO_REQUEST y OFPT_MULTIPART_REQUEST de tipo OFPMP_PORT_DESC, OFPMP_METER_FEATURES, OFPMP_FLOW, OFPMP_TABLE, OFPMP_PORT_STATS, OFPMP_GROUP, OFPMP_GROUP_DESC y OFPMP_METER.
- Mejoras en el procesamiento de los mensajes OFPT_FLOW_MOD y OFPT_METER_MOD: en el agente original, ante la llegada de estos dos tipos de mensajes, solo se extraían los campos necesarios para la creación de un servicio en la red GPON. Nosotros vamos a extraer todos los campos de dichos mensajes ya que vamos a tener en nuestro agente guardados en listas los flows y los meters que se hayan creado desde ONOS. En este sentido, cuando creamos un flow o un meter desde ONOS, se le asigna el estado de pendiente de añadir (PENDING_ADD). Para que ONOS de al flow o meter el estado de añadido (ADDED), es necesario que se le envíe esta información del flow o meter en el mensaje de tipo OFPT_MULTIPART_REPLY de tipo OFPMP_FLOW o OFPMP_METER. Por tanto, nosotros tenemos que

almacenar los flows y los meters en listas para posteriormente enviárselos a ONOS para indicarle que los flows se han instalado correctamente.

- **Funcionamiento ininterrumpido:** aunque nuestro agente es un programa en Python, nuestro controlador se comunica con él como si fuese un switch virtual. En este sentido, el agente inicial cuando recibía los parámetros necesarios para crear un servicio/os para una ONT/ONU concreta cortaba la comunicación con el controlador dejándole de contestar a sus mensajes. Esto hace que el controlador piense que el switch ha dejado de funcionar y le envía mensajes del tipo OFPT_ECHO_REQUEST. Si no recibe respuesta, deja de controlar el agente y se rompe la comunicación. En nuestro caso la creación de servicios no implica un corte en la comunicación de modo que nuestro agente siempre estará activo respondiendo a los mensajes de ONOS y creando o borrando servicios en la red GPON para cualquier ONT/ONU conectada en tiempo real. De este modo, hemos desarrollado un agente más global capaz de configurar y modificar en tiempo real servicios y perfiles de abonado de las ONTs/ONUs registradas y dadas de alta en dicha red de acceso.
- **Visibilidad de los flows y meters creados:** Los flows y los meters que se instalan en el agente, se procesan y se imprimen sus campos a un fichero, de manera que se pueda ver de forma fácil qué campos contienen en cualquier momento.
- **Mejoras en la estructura y legibilidad del código:** tanto del agente, como de las funciones que permiten la creación de los servicios en la red.

5.4 Implementación del Agente OpenFlow - CLI

Una vez diseñado el agente, pasamos a la fase de programación. Empezamos con el establecimiento de la comunicación entre ONOS y el agente OpenFlow. Después explicaremos en detalle la función principal del agente y finalmente los métodos del agente y de la API. La estructura del agente se muestra en la Figura 50.

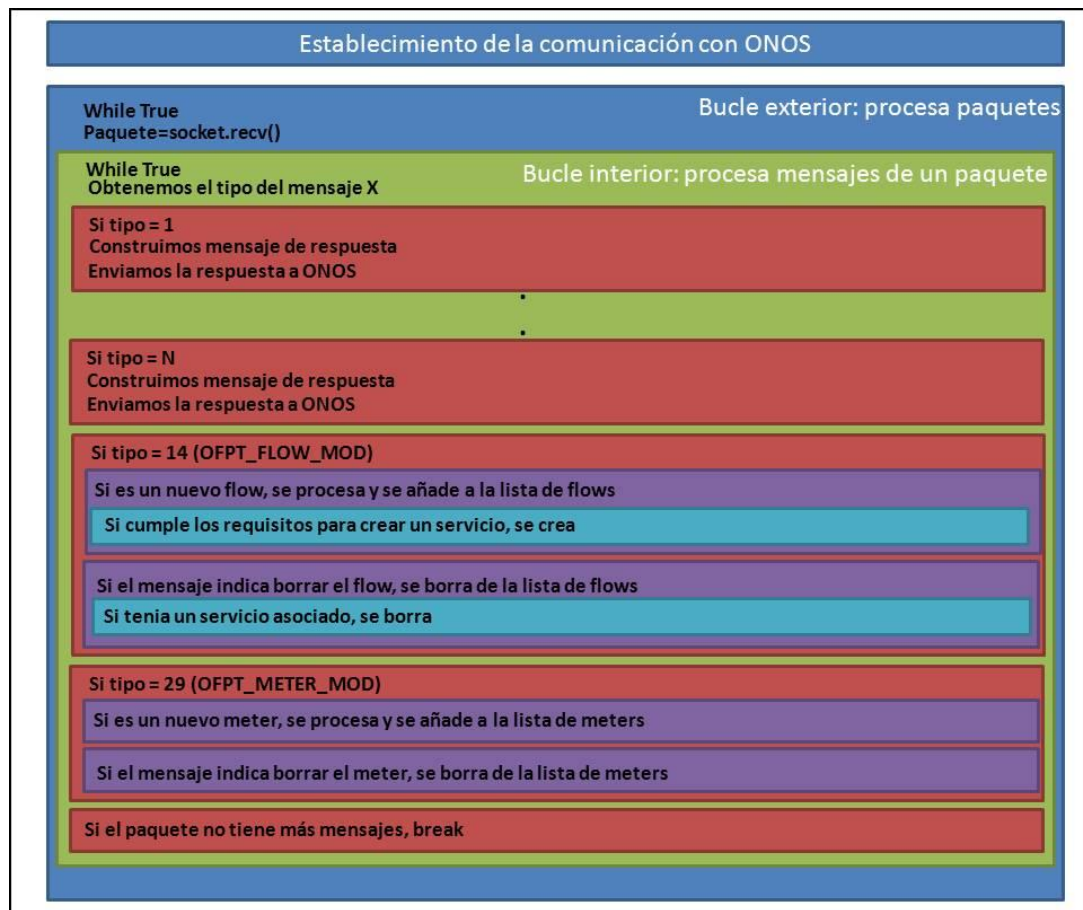


Figura 50: Estructura básica del agente OpenFlow

5.4.1 Establecimiento de la comunicación entre controlador y agente

Para establecer la comunicación entre ONOS y el agente OpenFlow desarrollado, tenemos que usar un socket TCP. Mediante el siguiente código creamos el socket:

```
mysocket=socket.socket()
```

Después tenemos que asociar el socket con un host y un puerto de destino. El host es la IP de nuestro controlador ONOS 10.0.60.2 y el puerto es el puerto OpenFlow por defecto, 6633. En concreto, este proceso se realiza mediante el código:

```
mysocket.connect((host,port))
```

Una vez que tenemos el socket conectado, lo primero que tenemos que hacer es enviar un mensaje OFPT_HELLO al controlador para establecer la comunicación. Para ello usamos la función *Hello()* de la librería python-openflow. Especificamos en los parámetros que vamos a utilizar la versión 4 de OpenFlow, que equivale a la 1.3. A continuación, empaquetamos el mensaje y lo enviamos con el código *mysocket.send(message)*. El controlador entonces contestará al agente con otro OFPT_HELLO para confirmar que se ha establecido correctamente la comunicación.

5.4.2 Funcionamiento principal del agente

A partir del establecimiento de la comunicación, comienza un bucle infinito denominado bucle exterior, que ante las peticiones del controlador mediante mensajes OpenFlow proporcionará las respuestas adecuadas. El agente no se parará hasta que no se haga una interrupción del programa (con ctrl+c), emulando así el comportamiento de un switch OpenFlow normal. Al principio este bucle recibe en el socket los datos enviados por el controlador con el código:

```
data=mySocket.recv(10240)
```

Siendo 10240 el número de bytes máximo que se espera recibir. Si se recibieran más de ese valor, solo se guardarían los bytes hasta ese valor, desechando los demás. Guardamos entonces la longitud en bytes de los datos recibidos.

En este momento comienza otro bucle, lo llamaremos bucle interior, donde dependiendo de qué tipo de mensaje recibamos, se enviará la respuesta adecuada a través de un mensaje OpenFlow concreto. Sabemos que en todos los mensajes OpenFlow el byte 1 nos dice el tipo de mensaje que hemos recibido, por lo que comparamos ese valor extraído del mensaje con los distintos casos posibles que nosotros contemplamos dentro del agente. Nuestro agente no da respuesta ni puede procesar todos los tipos de mensajes OpenFlow, ya que no es ese el objetivo. A continuación, se expondrán los mensajes a los que proporcionamos respuesta y con qué parámetros. Todos estos mensajes se crean utilizando las funciones y estructuras de la librería python-openflow. En concreto, el agente va a responder a los siguientes mensajes OpenFlow:

- OFPT_FEATURES_REQUEST: respondemos con un mensaje del tipo OFPT_FEATURES_REPLY indicando que tenemos 3 tablas (número arbitrario) y que en cuanto a capacidades podemos procesar estadísticas de flows, meters, tablas, puertos, grupos y colas.
- OFPT_GET_CONFIG_REQUEST: respondemos con un mensaje del tipo OFPT_GET_CONFIG_REPLY indicando que los fragmentos IP serán tratados normalmente.
- OFPT_BARRIER_REQUEST: respondemos con un mensaje del tipo OFPT_BARRIER_REPLY.
- OFPT_ROLE_REQUEST: respondemos con un OFPT_ROLE_REPLY indicando que ONOS tiene el rol master, ya que solo vamos a tener un controlador en la red.
- OFPT_MULTIPART_REQUEST: respondemos con un mensaje del tipo OFPT_MULTIPART_REPLY. Dependiendo del tipo que recibamos de este mensaje, el cuerpo del mensaje será:

-
- OFPMP_PORT_DESC: creamos previamente tres puertos, uno local y dos físicos. Respondemos enviándole una lista con estos tres puertos.
 - OFPMP_METER_FEATURES: respondemos indicando que soportamos ambos tipos de *meter* (DROP y REMARK), todos los tipos de capacidades, 65536 *meters* como máximo y 8 bandas (*bands*) máximo por meter.
 - OFPMP_DESC: respondemos con información inventada para nuestro agente en cuanto tipo de hardware, empresa desarrolladora, etc.
 - OFPMP_TABLE: respondemos con una lista con las tres tablas que tenemos en nuestro agente.
 - OFPMP_FLOW: respondemos con la lista de flows que tiene el agente en ese momento.
 - OFPMP_METER: respondemos con la lista de meters que tiene el agente en ese momento.
 - OFPMP_GROUP, OFPMP_GROUP_DESC, OFPMP_PORT_STAT: no proporcionamos respuesta para este tipo de mensajes.
 - OFPT_ECHO_REQUEST: respondemos con un OFPT_ECHO_REPLY.

Por último, nos faltan por comentar dos tipos más de mensajes que debido a su complejidad, serán explicados en detalle.

El primero de ellos es un OFPT_FLOW_MOD, que el controlador envía cuando quiere añadir o borrar un flow. Los flows en nuestro caso concreto van a usarse para configurar un servicio de Internet o de video en un usuario concreto (ONT) definido por una dirección MAC o IP concreta de la ONT que el usuario tiene conectada en su casa. Estos campos se encuentran en el campo *Match* del flow. Cada vez que llega este mensaje lo primero que hacemos es sacar el comando y la cookie del flow. Si el comando es añadir el flow, se realizan una serie de acciones para crear un flow y se guarda el mismo en una lista global de flows en el programa implementado. Si por el contrario el comando indica eliminar el flow, se recorre la lista en busca de un flow que tenga como cookie la cookie recibida en el mensaje y cuando se encuentre, se borra el flow de la lista.

En el caso de añadir el flow, tenemos que realizar varios bucles para sacar todos los posibles campos *Match*, todas las instrucciones y todas las acciones que contiene el flow. Primero se recorre la lista de flows para comprobar que el flow recibido no existe ya. Si existe no se hace nada, ya que el servicio para ese usuario y ONT ya ha sido creado, en nuestro caso en concreto solamente vamos a centrarnos en la creación de servicios de Internet y vídeo. Si no existe, se empieza a extraer la parte del campo *Match*. Cada *Match* tiene uno o varios campos OXM, y mediante la longitud total del *Match* y las longitudes de cada campo OXM se recorre en bucle todos los OXM, creando la estructura

OxmTLV, que representa cada uno de estos campos. Se van guardando en una lista de campos *OXM* y cuando se tienen todos, se crea la estructura llamada *Match* indicando que tiene como campos la lista anterior.

La siguiente parte es otro bucle que recorre las instrucciones. Dependiendo del tipo de instrucción se crea una estructura u otra. Nuestro agente es capaz de procesar tres tipos de instrucciones, que serán las necesarias para configurar los servicios en la red GPON mediante el agente: *OFPT_CLEAR_ACTIONS*, *OFPT_APPLY_ACTIONS* y *OFPT_METER*. Para la primera de ellas solo se crea la instrucción correspondiente con la función *InstructionClearAction()*. Para la segunda tenemos que mirar antes las acciones que lleva dentro dicho campo, y de nuevo se define otro nuevo bucle que recorre todas las acciones y las guarda en una lista de acciones. Después se crea la instrucción indicando que tiene como acción la lista de acciones guardadas anteriormente mediante el bucle. La función es *InstructionApplyAction(actions)*, donde *actions* es una lista de acciones. En nuestro caso esa lista solo tiene un tipo concreto, que es *Action_output*, *ActionOutput(port,max_length)*. Para el tercer caso (*OFPT_METER*), solo se indica en la instrucción el identificador del meter asociado a dicho flow, *InstructionMeter(meter_id)*. Todas las instrucciones se van guardando en una lista de instrucciones. En este momento ya tenemos guardados todos los campos del flow en el agente.

Además, se crea una estructura nueva denominada *FlowStats* para guardar todos los campos que se han ido recogiendo. Esta estructura de la librería guarda además de la información del flow sus estadísticas. Este tipo de estructura donde se guarda toda la información nos será muy útil ya que se tendrá que enviar dicha información cuando el controlador pida un *OFPT_MULTIPART_REQUEST* de tipo *OFPT_FLOW*. Tras rellenar la estructura con la lista de *Match* y la lista de instrucciones, además hay que añadir otros campos que se sacan del mensaje *OFPT_FLOW_MOD* como son la prioridad, el id de la tabla o los *timeout*. Una vez que tenemos el flow decodificado y guardado en el agente en forma de *FlowStats*, le añadimos a la lista global de flows. Por tanto, la lista global de flows es una lista de estructuras *FlowStats*. Un diagrama de los campos de esta estructura se muestra en la Figura 51. La estructura tiene los siguientes campos dentro:

FlowStats(length, table_id, duration_sec, duration_nsec, priority, idle_timeout, hard_timeout, flags, cookie, packet_count, byte_count, match, instructions)

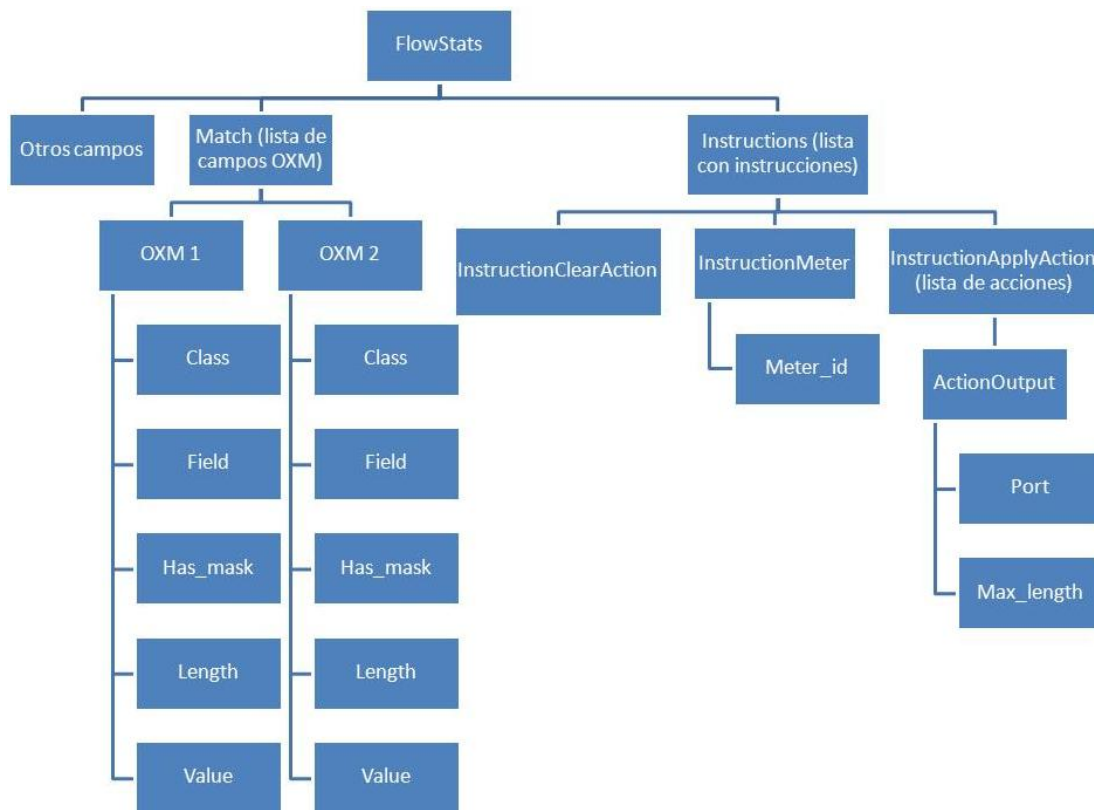


Figura 51: Diagrama con los campos de la estructura FlowStats

El otro mensaje interesante de comentar es el OFPT_METER_MOD. Puesto que consideramos que los flows permiten configurar los servicios para los usuarios en la red (en este caso servicio de Internet o vídeo), los meters podrían también usarse para asociar el ancho de banda máximo de cada servicio. En este caso tenemos dos listas globales, una con los meters, que guarda estructuras del tipo *MeterConfig* y otra con las estadísticas de los meter, *MeterStats*. La razón de esto es porque con los flows, la estructura *FlowStats*, que guarda las estadísticas del flow, también guarda todos los campos del flow, mientras que la estructura que guarda las estadísticas del meter, *MeterStats*, solo guarda el campo del identificador del meter y no guarda ninguna información acerca de qué tipo de meter es. Para eso necesitamos la estructura llamada *MeterConfig*. De este modo, cuando llega este tipo de mensaje, primero sacamos el identificador del meter y el comando asociado. Si el comando es eliminar el meter, recorremos las dos listas buscando un meter cuyo identificador coincida con el identificador del meter recibido en el mensaje y los borramos de las dos listas en el caso de que ya exista. Si el comando es añadir el meter, tenemos un bucle que recorre la parte de bandas (*bands*) del meter, ya que podría haber más de una banda (*band*) por meter. Estas se van guardando tanto en una lista de bandas (*bands*), formada por estructuras *MeterBandDrop* o *MeterBandDscpRemark* (dependiendo del tipo de banda), como en una lista de estadísticas de bandas, *BandStats*. Con esto ya podemos crear nuestras dos estructuras y añadirlas a sus respectivas listas. Un diagrama con estas estructuras se muestra en la Figura 52. Las estructuras anteriormente nombradas son las siguientes:

MeterConfig(flags, meter_id, bands)

MeterBandDrop(rate, burst_size)

MeterBandDscpRemark(rate, burst_size, prec_level)

MeterStats(meter_id, flow_count, packet_in_count, byte_in_count, duration_sec, duration_nsec, band_stats)

BandStats(packet_band_count, byte_band_count)

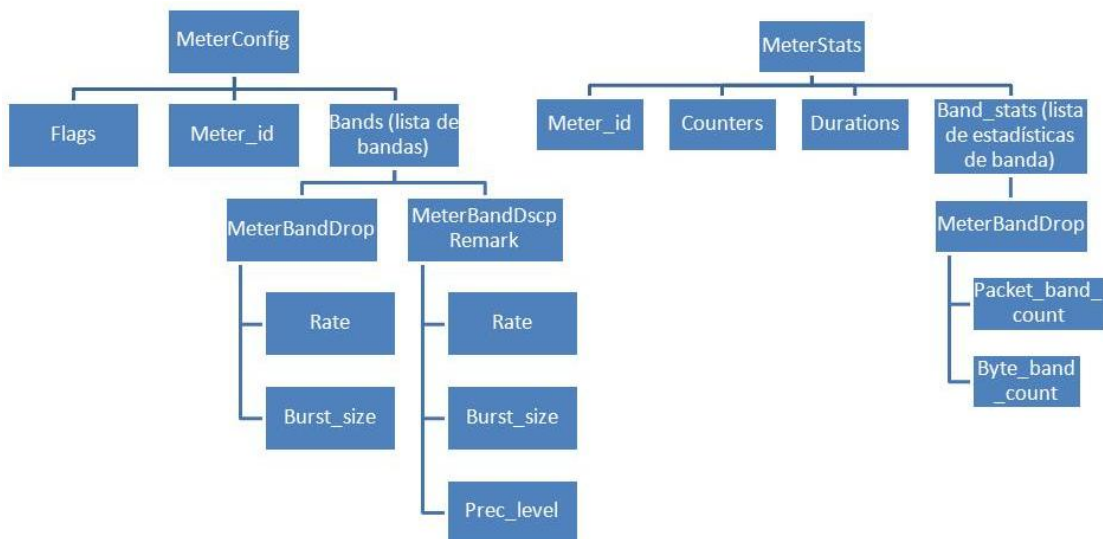


Figura 52: Diagrama con los campos de las estructura MeterConfig y MeterStats

Cabe destacar que los flows, tal como se ha explicado anteriormente, se guardan en una lista de estructuras *FlowStats*. Esto es porque la lista con los flows va a ser el cuerpo del mensaje *OFPT_MULTIPART_REPLY* de tipo *OFPT_FLOW* que enviará el agente al controlador para indicar que ha recibido ese flow correctamente. En este sentido, cuando desde ONOS se instala un flow, ONOS marca el estado del flow como pendiente de añadir (*PENDDING_ADD*). Cuando el agente recibe estos flows los va guardando en una lista, por lo que cada vez que se recibe un flow, ONOS lo marca en ese estado y no lo marca como añadido (*ADDED*) hasta que el agente envíe un mensaje *OFPT_MULTIPART_REPLY* de tipo *OFPT_FLOW* donde aparezca ese flow y sus características. Si nuestro agente no respondiese a este mensaje, ONOS no daría nunca el flow por añadido y seguiría enviando periódicamente el flow en mensajes de tipo *OFPT_FLOW_MOD*, hasta que el agente respondiera correctamente.

Una vez explicados todos los tipos de mensaje a los que se da respuesta, falta una característica más por comentar del programa. Algunas veces los mensajes que se reciben no vienen solos, sino que hay paquetes que tienen varios mensajes en ellos. Este comportamiento ya lo comentamos en al final del Apartado 5.2. En la Figura 53 podemos ver un caso de este tipo, donde el mismo paquete lleva un mensaje *OFPT_FLOW_MOD* y un mensaje *OFPT_BARRIER_REQUEST*.

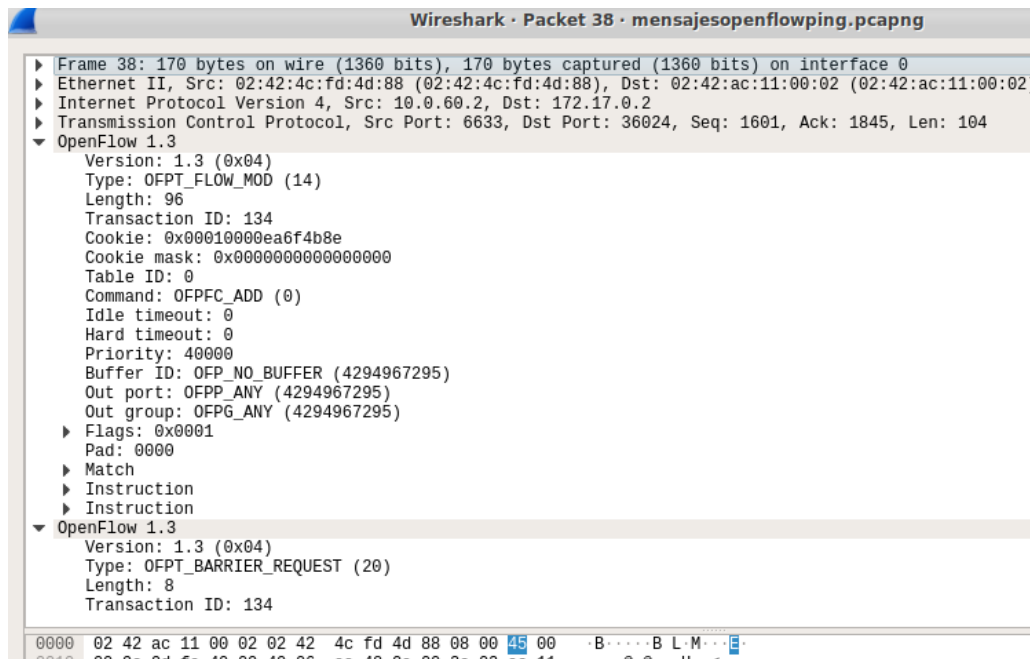


Figura 53: Caso de un paquete con varios mensajes OpenFlow

Para estos casos tenemos el doble bucle que se ha comentado antes. El bucle exterior procesa los paquetes, mientras que el bucle interior procesa los mensajes de cada paquete. Teniendo la longitud total del paquete y la longitud del primer mensaje podemos saber si hay más de un mensaje en el paquete. Si las longitudes son iguales, solo hay un mensaje. Si la longitud del mensaje es menor que la longitud total del paquete, entonces es que hay más de un mensaje. En estos casos, al acabar el bucle interior, se resta a la longitud total del paquete la longitud del primer mensaje y además sumamos la longitud del mensaje a una variable llamada *extraByte*. Esta variable comienza valiendo 0. Vamos a poner un ejemplo para explicar para qué sirve esta variable. Cuando recibimos el primer mensaje del paquete y sacamos el tipo para posteriormente enviar la respuesta correspondiente, sabemos que el tipo del mensaje siempre está en el byte 1. Si el mensaje recibido del socket se encuentra en una variable llamada *data*, para conseguir el tipo del mensaje hacemos *tipo=data[1+extraByte]*. Como *extraByte* al principio del bucle interior vale 0, estamos accediendo al byte 1 del paquete. Si después resulta que el paquete tiene más de un mensaje, el tipo del siguiente mensaje ya no estará en el byte 1 del paquete, si no en el byte 1+longitud del mensaje anterior. Si sumamos a la variable *extraByte* la longitud del mensaje anterior (por ejemplo 96 bytes, como vemos en el primer mensaje de la Figura 53), podemos acceder relativamente al tipo del mensaje actual de nuevo usando *data[1+extraByte]*, que ahora valdrá 97.

Usando estas técnicas podemos procesar todos los mensajes de un paquete. Sabremos que no hay más mensajes cuando la longitud del paquete a la que vamos restando las longitudes de los mensajes llegue a 0. En ese momento se saldrá del bucle interior y comenzará de nuevo el bucle exterior recibiendo un nuevo paquete y reiniciando estas variables que se han comentado.

5.4.3 Métodos de lectura de datos en el agente OpenFlow

Se tienen cuatro funciones para imprimir tanto por la pantalla como a un fichero los flows y los meters existentes para la configuración de los servicios y perfiles de usuario de la red GPON. La utilidad de estas funciones es poder visualizar fácilmente que flows y meters han sido procesados y guardados en el agente OpenFlow de la red GPON y poder ver todos sus campos de manera sencilla. Las dos primeras funciones, llamadas *writeFlows()* y *writeMeters()* simplemente imprimen los flows y los meters, respectivamente, a un fichero cada una. Estas funciones recorren las listas de flows o meters sacando los datos de los campos y haciendo la información legible para la API que posteriormente interactuará con la red GPON. Estos métodos se llaman cada vez que llega un OFPT_FLOW_MOD o un OFPT_METER_MOD.

Después tenemos otras funciones similares llamadas *printFlows()* y *printMeters()*, que imprimen la misma información que las dos funciones anteriores por la pantalla y de forma asíncrona. Esto es porque nuestro programa es un bucle infinito, por lo que no se podría por ejemplo pedir a un usuario si quiere imprimir alguna de estas listas, ya que no se sale del bucle y si se pidiese en medio de él, el programa se bloquearía. Para resolver esto, importamos una librería llamada *keyboard* [44], que permite añadir atajos de teclado al programa para realizar acciones de forma asíncrona. En nuestro caso se va a usar para llamar a estas funciones sin que se pare el bucle infinito. El código para implementar esta funcionalidad es:

```
keyboard.add_hotkey('shift+f', lambda: printFlows(listF)).
```

Lo que significa que cuando se pulse la combinación de teclas “mayúsculas + f”, se llamará a la función *printFlows()* con la lista de flows, de manera asíncrona. Análogamente, existe otra para los meters mediante el mismo comando pero con “mayúsculas + m”. La inclusión de esta librería implica que ahora el programa debe ser ejecutado como administrador.

5.4.4 Conexión del agente con la red GPON del laboratorio

En este punto del trabajo se ha conseguido conectar nuestro agente OpenFlow con el controlador ONOS, de manera que nuestro agente es capaz de recibir los flows y meters y almacenarlos en listas. El siguiente paso es conectar nuestro agente con la red GPON, de manera que sea capaz de configurar servicios para los usuarios. La comunicación con la red se hace a través de una API ya desarrollada y que nosotros hemos actualizado y mejorado para que sea compatible con nuestro agente. La API define dos tipos de servicio: servicio de Internet y servicio de Video (al que se le adscribe un servicio de Internet). Ante la llegada de nuevos flows al agente, este determinará si tienen los requisitos necesarios para crear un servicio. En el caso del servicio de Internet, el flow tiene que tener una VLAN concreta para los servicios de Internet (definida por el operador/proveedor de servicios de la red), un meter de tipo DROP con una tasa asociada

que indicará el ancho de banda contratado por el usuario y una dirección MAC de destino existente en la red que corresponderá con la ONT/ONU del usuario. Si el flow carece de alguno de estos campos, el servicio no será creado. Para el servicio de Vídeo, se requieren estos campos más una dirección IP de destino de tipo broadcast. La API define las siguientes funciones:

- *connect()*: establece una conexión Telnet con el CLI (*Command Line Interface*) de la OLT e introduce las contraseñas necesarias para habilitar el CLI y para entrar en modo configuración. La función devuelve el comunicador, de manera que todas las demás funciones llamen a esta para conectarse al CLI en primer lugar.
- *get_Puerto()*: obtiene el número de puerto de la OLT donde están conectadas las ONTs/ONUs. Primero llama a la función *connect()* para conectarse al CLI y después recorre los 4 puertos de la OLT en busca del que tiene las ONTs/ONUs conectadas. La red del laboratorio siempre tiene todas las ONTs/ONUs conectadas a un mismo puerto.
- *get_ID_ONU()*: devuelve una lista con las direcciones MAC de las ONTs/ONUs de la red. Primero se conecta al CLI y obtiene el puerto mediante las dos funciones anteriores. Después, guardan las direcciones MAC de las ONTs/ONUs que hay en ese puerto en una lista. Las ONTs/ONUs tienen un identificador interno (ID) dentro de la red GPON, y al guardarlas en una lista, el ID coincide con la posición de la dirección MAC de la ONT/ONU en la lista. Las direcciones MAC de las ONTs/ONUs de la red son de 64 bits, pero ONOS no soporta direcciones de 64 bits, solo de 48 bits, por lo que las direcciones MAC se convierten a direcciones MAC de 48 bits. Por último, en esta función también crea, si no existe ya, un fichero XML para donde a partir de un elemento raíz, se añade cada ONU con su dirección MAC como un subelemento del elemento raíz. Este fichero sirve para recuperar configuraciones de la red GPON y de los servicios y perfiles de abonado creados cuando se apaga la red o se cambia el modo de gestión de ella.
- *servicio_Internet(ID_ONU,MAC_ONU,VLAN_Internet,Bmax_Internet,num_servicios_Internet)*: esta función configura un servicio de Internet. Tiene como parámetros de entrada el ID de la ONU de la ONU donde se crea el servicio, (recordemos que el ID se obtiene de la posición de la MAC de la ONU en la lista de direcciones MAC), la dirección MAC de la ONT/ONU, la VLAN del servicio, la tasa del servicio y el número de servicios que se van a configurar (por ahora, siempre 1). La función empieza con una llamada a la función *connect()* y otra a la función *get_Puerto()*, para establecer una conexión con el CLI de la OLT y obtener el puerto donde se encuentra la ONT/ONU. Entonces en base a la tasa que entra como parámetro, se obtienen el ancho de banda en el canal de subida y bajada, tanto el ancho garantizado, como el de en exceso.

También se transforma la dirección MAC de 48 bits a 64 bits, es la transformación inversa a la comentada en la función anterior. En este punto comienzan a ejecutarse una larga serie de comandos en el CLI de la OLT, que no vamos a explicar ya que no es el objetivo de este trabajo, solo saber que el resultado es que se crea un servicio de Internet en la ONT/ONU. Todos estos comandos junto con la respuesta devuelta por el OLT se vuelcan a un fichero para que podamos comprobar que se han ejecutado correctamente. También se actualiza el fichero XML con los datos configurados.

- *servicio_Video(ID_ONU,MAC_ONU,VLAN_Video,Bmax_Video,num_servicios_Video)*: esta función realiza las mismas acciones que la función anterior, pero para un servicio de Video. Solo cambian los comandos que se ejecutan en la OLT para la definición de los canales multicast y otras características.
- *borrar_configuracion(ID_ONU,MAC_ONU,nombre_archivo)*: esta función borra un servicio. Tiene como parámetros el ID de la ONU, la MAC de la ONU que tiene el servicio y el nombre del archivo que contiene los comandos que se ejecutaron en la OLT mediante alguna de las dos funciones anteriores. Esta función se conecta al CLI y obtiene el puerto donde está la ONU con *connect()* y *get_Puerto()*. Después ejecuta una serie de comandos en el CLI de la OLT para borrar el servicio. Después borra el archivo cuyo nombre entró como parámetro y por último se actualiza el fichero XML.

Para poder implementar estas funcionalidades, hemos importado la API a nuestro programa. Nuestro agente también guarda los campos requeridos en un fichero para indicar que existe un servicio. Tenemos dos ficheros para ello, el primero para los servicios de Internet y el segundo para los servicios de Video. Si llegase un flow con los mismos campos o para crear un nuevo servicio para el mismo usuario (ONT/ONU), se comprobará este fichero para saber si ya existe un servicio asociado a ese usuario. Si existe, no se llama a la función de crear un servicio. Además, otro uso de estos ficheros es que si tenemos un servicio por ejemplo de Internet creado, el fichero de servicios de Internet contendrá los campos de este servicio. Si en algún momento se para la ejecución del agente, al volverlo a iniciar, leerá este fichero (y el de los servicios de Video) para recuperar los servicios que estaban creados. Funciona a modo de backup. Nos referiremos a estos dos ficheros en adelante como ficheros de backup de servicios.

Por último, vamos a explicar la secuencia de acciones que se realizan cuando se crea un servicio y más tarde se borra:

1. Ejecutamos el agente OpenFlow.
2. Se establece la comunicación con el controlador ONOS en su dirección IP.
3. Se ejecuta la función de la API *get_ID_ONU()*, que nos devuelve en una lista las direcciones MAC de las ONTs/ONUs conectadas en la red.

-
4. Se leen los ficheros de backup de servicios por si hubiese algún servicio ya creado.
 5. Se envía un mensaje OFPT_HELLO a ONOS para iniciar la comunicación.
 6. El agente y ONOS se siguen intercambiando otros tipos de mensajes.
 7. Se instala desde ONOS un meter de tipo DROP. Es importante destacar en este punto que se debe de instalar el meter antes que el flow, ya que para que ONOS de por instalado el flow, comprueba que exista el meter que tiene asociado (en el caso de tener uno asociado). Si no existe el meter, ONOS deja el flow en estado pendiente de añadir (PENDING_ADD).
 8. El agente recibe un OFPT_METER_MOD y guarda el meter.
 9. Se instala uno o varios flows desde ONOS con los parámetros necesarios para crear servicios bien de Internet o de Vídeo.
 10. El agente recibe un OFPT_FLOW_MOD (en este caso para añadir un nuevo flow) para crear un servicio sobre una ONT/ONU.
 - 10.1. Si el flow ya existe, no se realiza ninguna acción.
 - 10.2. Si el flow no existe, se sacan los campos del flow y se crea una estructura *FlowStats* y se añade la estructura a la lista de flows.
 - 10.3. Se comprueba si el flow cumple los requisitos para crear un servicio, estos son, para un servicio de Internet, que tenga una VLAN, un meter asociado y una dirección MAC de destino que coincida con alguna de las direcciones MAC que tenemos almacenadas de las ONTs/ONUs que están conectada a la red GPON. Si es de Video, son los anteriores añadiendo que tenga una dirección IP broadcast de destino.
 - 10.4. Si cumple los requisitos anteriores menos la IP de broadcast, es un servicio de Internet. Si además tiene IP de broadcast de destino, es un servicio de Video.
 - 10.5. Si no cumple los requisitos, el servicio no se crea.
 - 10.6. Se comprueba que no exista ya un servicio con los exactamente los mismos campos que otro servicio existente. Para ello se mira en el fichero de backup del tipo de servicio que se vaya a crear, si el identificador del flow ya existe en el fichero.
 - 10.7. Si existe, el servicio no se crea.
 - 10.8. Si no existe, se añaden a este fichero el id del flow y los parámetros del servicio (MAC de destino, VLAN y el identificador del meter y si es un servicio de Video, se añade además la IP de broadcast).
 - 10.9. Entonces se llama a la función *servicio_Internet()* o la función *servicio_Video()* si es un servicio de Video.

-
- 10.10. Una vez que la anterior función retorna, el servicio se ha creado. Vemos que el fichero XML se ha actualizado con el nuevo servicio y que tenemos un fichero nuevo donde vemos todos los comandos que se han ejecutado en el CLI de la OLT para configurar el servicio.
 11. El agente continúa con la ejecución normal.
 12. Ahora se borra el flow anteriormente creado desde ONOS.
 13. El agente recibe un OFPT_FLOW_MOD (en este caso para borrar un flow).
 - 13.1. Se borra el flow de la lista de flows
 - 13.2. Se recorren ambos ficheros de backup de servicios hasta encontrar el ID del flow en alguno de ellos. Cuando se encuentra, se borran todos los campos asociados a ese servicio del fichero, no sin antes coger la MAC de la ONT/ONU que tenía el servicio.
 - 13.3. Se llama a la función *borrar_servicio()*, sabemos a qué ONT/ONU pertenecía el servicio ya que tenemos su MAC. El nombre del archivo que también tiene esta función como parámetro de entrada, es el archivo con los comandos que se ejecutaron en el CLI de la OLT. Esta función ejecuta una serie de comandos en el CLI de la OLT para borrar el servicio en la ONT/ONU, después se actualiza el fichero XML y se borra el archivo con los comandos.
 14. El agente continúa con la ejecución normal.

La Figura 54 muestra un diagrama de flujo de la creación y borrado de los servicios para facilitar la comprensión de este punto. El diagrama comienza desde el punto 10 de la secuencia anterior.

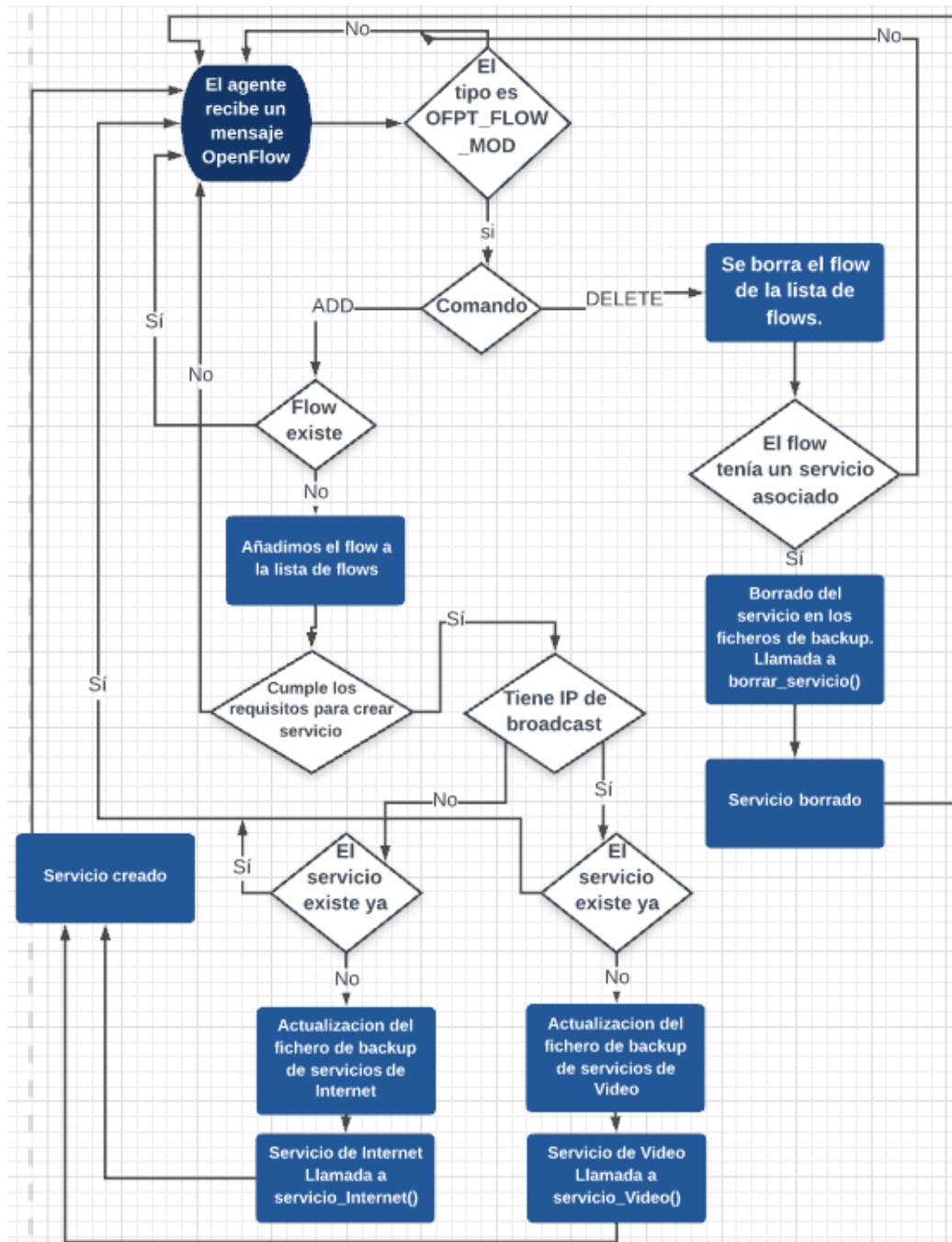


Figura 54: Diagrama de flujo en la creación y borrado de servicios

5.5 Pruebas realizadas sobre el agente OpenFlow

Vamos a realizar dos pruebas unitarias que nos permitan verificar el correcto funcionamiento del agente. La primera de ellas será para comprobar que la comunicación entre ONOS y el agente es satisfactoria. En la segunda ampliaremos esta comunicación hasta la red GPON, para asegurar que se crean los servicios correctamente.

5.5.1 Pruebas entre ONOS y el agente OpenFlow

Aunque a lo largo de la fase de programación se han ido realizando pruebas de control para verificar que se estaba programando correctamente, una vez que tenemos el agente completamente desarrollado debemos comprobar el comportamiento de nuestro agente OpenFlow y probar que los flows son creados bien y procesados correctamente por nuestro agente.

Esta prueba consiste en la creación de dos meters y dos flows para verificar que el inicio de la comunicación entre ONOS y el agente es correcto y que el agente es capaz de responder de manera adecuada a todos los mensajes que envía ONOS. Los flows no tendrán los parámetros necesarios para crear un servicio, ya que en esta prueba solo queremos comprobar la comunicación entre ONOS y el agente. Cada flow lleva asociado un meter. Cada meter es de un tipo diferente (DROP y DSCP REMARK). Uno de los dos flows lleva como campos *Match* a modo de prueba todos los que el agente es capaz de procesar, esto es, IN_PORT, ETH_TYPE, ETH_SRC, ETH_DST, VLAN_ID, VLAN_PCP, IPV4_SRC e IPV4_DST. Estos campos son el puerto de entrada, el tipo de paquete Ethernet, la dirección MAC de origen, la dirección MAC de destino, el identificador de VLAN, el campo PCP (*Priority Code Point*) de la VLAN, la IP versión 4 de origen y la IP versión 4 de destino, respectivamente. Como también lleva asociado un meter y una acción, probamos los tres tipos de instrucciones que el agente puede procesar, esto es, OFPIT_CLEAR_ACTIONS, OFPIT_APPLY_ACTIONS y OFPIT_METER. Estos flows y meters van a ser instalados en el agente desde ONOS a través de un script como ya presentamos en el Apartado 4.5.

Para ello ejecutamos el programa de nuestro agente para que esté activo y pueda recibir los mensajes OpenFlow enviados por el controlador ONOS. Después tenemos que instalar los flows y los meters desde dicho controlador, que los tenemos creados en un archivo que ejecutaremos como un script. En el *ANEXO IV* se recogen los flows y meters que se van a instalar en el agente a través de ONOS. El primer meter es de tipo DROP con una tasa de 204800 Kbps. El segundo meter es de tipo DSCP_REMARK con una tasa de 0 y un nivel de precedencia de 1. De los dos flows, el que más campos *Match* tiene definidos, tiene de puerto de entrada el 2, tipo de trama Ethernet 0x800, MACs de origen y destino 00:00:00:00:00:01 y 00:00:00:00:00:02, VLAN 1 con prioridad PCP de 1 y por último IPs de origen y destino 10.0.0.1/32 y 10.0.0.0/24. Como instrucciones se tiene la salida del paquete por el puerto 1 y que aplique el meter número 1 (que es el de tipo DROP). Ejecutamos a continuación el script que nos instala estos flows y meters en el agente desde ONOS. Tal y como se observa en la Figura 55, el programa imprime el tipo de los mensajes que el agente envía al controlador y también vemos los mensajes OFPT_METER_MOD y OFPT_FLOW_MOD, que en este caso indican la llegada de un nuevo meter o flow.

```
Terminal - u
File Edit View Terminal Tabs Help
usuario@virtual:~/Escritorio$ sudo python3 AgenteOpenFlow.py
OFFMP_FEATURES_REPLY
OFFMP_PORT_DESC
OFFMP_SET_CONFIG
OFFMP_BARRIER_REPLY
OFFMP_GET_CONFIG_REPLY
OFFMP_METER_FEATURES
OFFMP_DESC
OFFMP_ROLE_REPLY 3
OFFMP_ROLE_REPLY 3
OFFMP_PORT_DESC
OFFMP_PORT_DESC
OFFMP_FLOW_MOD
OFFMP_BARRIER_REPLY
OFFMP_FLOW_MOD
OFFMP_BARRIER_REPLY
OFFMP_FLOW_MOD
OFFMP_BARRIER_REPLY
OFFMP_PORT_STATS
OFFMP_FLOW
OFFMP_TABLE
OFFMP_METER
OFFMP_FLOW_MOD
OFFMP_METER_MOD
OFFMP_FLOW_MOD
OFFMP_FLOW_MOD
OFFMP_FLOW_MOD
OFFMP_METER_MOD
OFFMP_GROUP
OFFMP_GROUP_DESC
OFFMP_FLOW_MOD
OFFMP_BARRIER_REPLY
OFFMP_PORT_STATS
OFFMP_FLOW
OFFMP_TABLE
OFFMP_PORT_STATS
OFFMP_FLOW
OFFMP_TABLE
OFFMP_METER
OFFMP_GROUP
```

Figura 55: Mensajes procesados por el agente OpenFlow

Para probar las funciones de imprimir flows y meters utilizamos las dos maneras. Estas funciones nos van a servir para verificar que los flows que hemos enviado anteriormente han sido instalados correctamente y ONOS los da por añadidos (ADDED). En la Figura 56 vemos la primera forma de visualizar los flows o meters, en este caso los meters, que es utilizar el atajo de teclado “mayúsculas + m” para imprimir los meters por la pantalla mientras se está ejecutando el agente. Podemos comprobar que son los mismos que los que se definieron en el script, esto es, el primero con una tasa de 204800 Kbps (*Band rate*) de tipo DROP (*Band type*) y el segundo con una tasa de 0 Kbps, con nivel de precedencia 1 (*Band precedence level*) de tipo DSCP_REMARK (*Band type*).

```
OFFMP_PORT_STATS
OFFMP_FLOW
OFFMP_TABLE
Meter id: 2
Meter flags: 5
Bands:
  Band type: MeterBandType.OFPMBT_DSCP_REMARK
  Band rate: 0
  Band burst size: 0
  Band precedence level: 1
Meter id: 1
Meter flags: 5
Bands:
  Band type: MeterBandType.OFPMBT_DROP
  Band rate: 204800
  Band burst size: 0
MOFMP_PORT_STATS
OFFMP_FLOW
OFFMP_TABLE
OFFMP_METER
OFFMP_GROUP
OFFMP_GROUP_DESC
```

Figura 56: Meters que tiene el agente OpenFlow durante la realización de la prueba

En la Figura 57 vemos la segunda forma para visualizar estos campos, que es abrir el fichero (en este caso para los flows) y comprobar qué flows tiene guardados en estos momentos el agente. No se muestran todos los flows ya que ocupan mucho, pero

mostramos a modo de resumen uno de los flows que hemos creado, en concreto con el que hemos comentado arriba con puerto de entrada 2, trama Ethernet 0x800, MACs de origen y destino 00:00:00:00:00:01 y 00:00:00:00:00:02, VLAN 1 con prioridad PCP de 1 e IPs de origen y destino 10.0.0.1/32 y 10.0.0.0/24

```

/home/usuario/Escritorio/flows.txt - Mou
File Edit Search View Document Help
94 Flow number: 4
95 Flow id (cookie): 0x4f0000759530eb
96 Flow priority: 10
97 Flow table id: 0
98 Match:
99   OxmMatchField number: 0
100   OxmMatchField oxm class: 0x8000
101   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_IN_PORT
102   OxmMatchField oxm value: 2
103   OxmMatchField number: 1
104   OxmMatchField oxm class: 0x8000
105   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_ETH_DST
106   OxmMatchField oxm value: 00:00:00:00:00:01
107   OxmMatchField number: 2
108   OxmMatchField oxm class: 0x8000
109   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_ETH_SRC
110   OxmMatchField oxm value: 00:00:00:00:00:02
111   OxmMatchField number: 3
112   OxmMatchField oxm class: 0x8000
113   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_ETH_TYPE
114   OxmMatchField oxm value: 0x800
115   OxmMatchField number: 4
116   OxmMatchField oxm class: 0x8000
117   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_VLAN_VID
118   OxmMatchField oxm value: 1
119   OxmMatchField number: 5
120   OxmMatchField oxm class: 0x8000
121   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_VLAN_PCP
122   OxmMatchField oxm value: 1
123   OxmMatchField number: 6
124   OxmMatchField oxm class: 0x8000
125   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_IPV4_SRC
126   OxmMatchField oxm value: 10.0.0.1
127   OxmMatchField number: 7
128   OxmMatchField oxm class: 0x8000
129   OxmMatchField oxm field: OxmOfbMatchField.OFPXMT_OFB_IPV4_DST
130   OxmMatchField oxm value: 10.0.0.0
131   OxmMatchField oxm mask: 255.255.255.0
132 Instruction:
133   Instruction number: 0
134   Instruction type: InstructionType.OFPIT_APPLY_ACTIONS
135   Action:
136     Action type: ActionType.OFPAT_OUTPUT
137     Action port: 1
138     Action max length: 0
139 Instruction:
140   Instruction number: 1
141   Instruction type: InstructionType.OFPIT_METER
142   Instruction meter id: 1

```

Figura 57: Flows que tiene el agente OpenFlow durante la realización de la prueba

Tras la realización de estas pruebas podemos asegurar que el comportamiento del agente es correcto y pasamos a analizar el comportamiento de nuestro agente cuando interacciona con la API de la red GPON para analizar si se crean bien los servicios sobre la OLT de la red GPON.

5.5.2 Prueba extremo a extremo entre ONOS y la API GPON

A continuación, vamos a realizar una prueba para crear dos servicios, uno de Internet y otro de Video sobre dos de las ONUs/ONTs conectadas a la red GPON del laboratorio. Para la realización de esta prueba, trasladamos nuestro software de la máquina virtual donde hemos estado trabajando en este proyecto, al ordenador que administra la red GPON del laboratorio e instalamos ONOS en una máquina virtual. En nuestro agente tenemos que cambiar la dirección del controlador a la dirección de la máquina virtual que

está corriendo ONOS, ahora la 10.0.103.56. También es necesario instalar las librerías python-openflow y keyboard en el ordenador que gestiona la red GPON. Un diagrama con todos los componentes para esta prueba se muestra en la Figura 58.

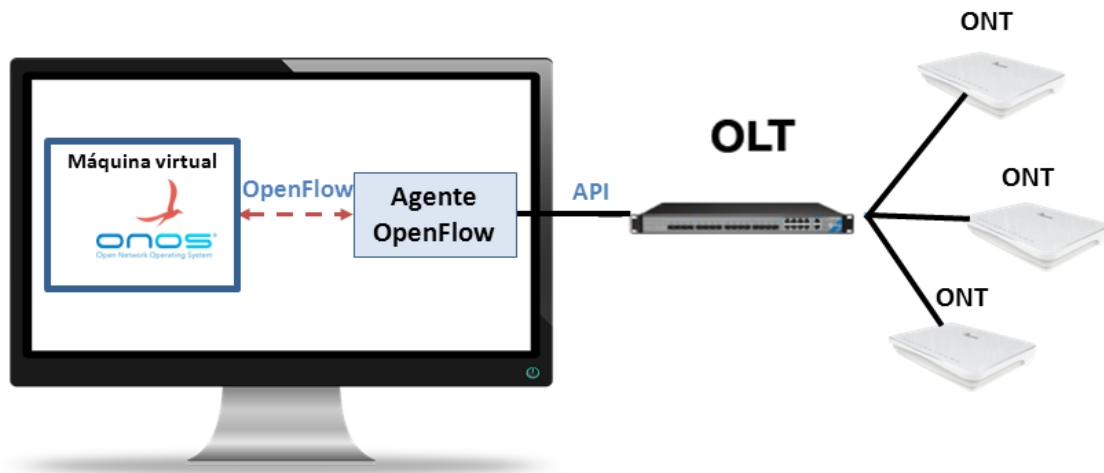


Figura 58: Diagrama con los componentes para la prueba de creación de servicios

Una vez configurado el escenario de red, se ejecuta el agente y para ello se crea otro script con el meter y los flows que se van a instalar esta vez para crear los servicios de Internet y vídeo y volcarlos sobre los usuarios de la red GPON. El meter es de tipo DROP con una tasa de 50000 Kbps (50 Mbps), es decir, vamos a crear un servicio en la ONT/ONU con un ancho de banda asociado de 50 Mbps. El primer flow asociado a un servicio de Internet, tiene la dirección MAC de destino 78:3d:5b:01:f7:30 (una ONUs/ONTs de la red del laboratorio) y la VLAN 806. El segundo flow, para el servicio de Video, tiene la dirección MAC de destino 78:3d:5b:01:f7:28 (otra de las ONUs/ONTs de la red del laboratorio), la VLAN 806 y la dirección IP de broadcast 10.0.0.255/32. Los dos flows tienen el meter de 50 Mbps asociado. En el ANEXO V se muestra el script con el meter y los flows configurados. A continuación, se lanza nuestro script para instalar los flows en el agente y crear los servicios desde ONOS. En la Figura 59 se muestran un conjunto de mensajes que aparecen en el agente OpenFlow desarrollado donde se puede ver que han creado los servicios junto con las tasas de ancho de banda garantizado y en exceso que serán de carácter simétrico en ambos canales (subida y bajada).

```

OFPMP_FLOW
OFPMP_TABLE
OFPT_METER_MOD
El identificador VLAN para el servicio 1 es: 806
El ancho de banda Downstream garantizado en Kbps es: 49984
El ancho de banda Downstream en exceso en Kbps es: 49984
El ancho de banda Upstream garantizado en Mbps es: 45
El ancho de banda Upstream BE en Mbps es: 50

Servicio de Internet configurado.

OFPT_FLOW_MOD
El identificador VLAN para el servicio 1 es: 806
El ancho de banda Downstream garantizado en Kbps es: 49984
El ancho de banda Downstream en exceso en Kbps es: 49984
El ancho de banda Upstream garantizado en Mbps es: 50
El ancho de banda Upstream BE en Mbps es: 50

Servicio de Internet+Video configurado.

OFPT_FLOW_MOD
OFPT_BARRIER_REPLY
OFPMP_PORT_STATS
OFPMP_FLOW

```

Figura 59: Visualización en el agente OpenFlow de los servicios que se volcarán sobre la red GPON

Comprobamos también los demás ficheros que se crean o se actualizan. La Figura 60 muestra el contenido de los ficheros de backup de los servicios. Podemos observar que tienen los parámetros de los servicios que nosotros hemos creado, esto es, el ID del flow asociado a ese servicio, la MAC de la ONU/ONT, la VLAN y finalmente el ancho de banda en Kbps. Además, el servicio de Video tiene asociada la dirección IP de broadcast. La Figura 61 muestra el contenido del archivo XML, donde se ven todas las ONUs de la red y los servicios que tiene configurados cada una.

internetServices.txt	videoServices.txt
1 48695173356753599	1 48695173965645017
2 78:3d:5b:01:f7:30	2 78:3d:5b:01:f7:28
3 806	3 806
4 50000	4 50000
5	5 10.0.0.255

Figura 60: Contenido del fichero de backup de servicios de Internet (a la izquierda) y de servicios de Video (a la derecha)

```

- <RedGPON>
- <ONU MAC="54-4c-52-49-5b-01-f7-30">
- <Servicio tipo="Internet">
  <VLAN_ID>806</VLAN_ID>
  <BW_Down_GR>49984</BW_Down_GR>
  <BW_Down_Excess>49984</BW_Down_Excess>
  <BW_Up_GR>45</BW_Up_GR>
  <BW_Up_BE>50</BW_Up_BE>
</Servicio>
</ONU>
<ONU MAC="54-4c-52-49-5b-01-f6-d8" />
<ONU MAC="54-4c-52-49-5b-02-e9-ca" />
<ONU MAC="54-4c-52-49-5b-03-fc-82" />
- <ONU MAC="54-4c-52-49-5b-01-f7-28">
- <Servicio tipo="Internet+Video">
  <VLAN_ID>806</VLAN_ID>
  <BW_Down_GR>49984</BW_Down_GR>
  <BW_Down_Excess>49984</BW_Down_Excess>
  <BW_Up_GR>50</BW_Up_GR>
  <BW_Up_BE>50</BW_Up_BE>
</Servicio>
</ONU>
</RedGPON>

```

Figura 61: Contenido del fichero XML tras la creación de los servicios

Por último, mostramos los comandos que se ejecutan en el CLI de la OLT en el caso del servicio de Internet cuando el agente interactúa con la API de la red GPON, junto con las respuestas devueltas por el OLT a cada uno de estos comandos de configuración. Tal y como se observa en dicho código, todos los comandos de configuración se ejecutan correctamente ya que devuelven “*Response result: Command success*”:

```

olt-device 0
OLT CLI(DEV0)# olt-channel 1
OLT CLI(DEV0 CH1)# onu-local 0
OLT CLI(DEV0 CH1 LOC-ONU0)# omci-port 0
OLT CLI(DEV0 CH1 LOC-ONU0)# exit
OLT CLI(DEV0 CH1)# onu-omci 0
OLT CLI(DEV0 CH1 ONU(OMCI)0)# ont-data mib-reset
Response result: Command success
OLT CLI(DEV0 CH1 ONU(OMCI)0)# exit
OLT CLI(DEV0 CH1)# fec direction uplink 0
OLT CLI(DEV0 CH1)# onu-local 0
OLT CLI(DEV0 CH1 LOC-ONU0)#
OLT CLI(DEV0 CH1 LOC-ONU0)# alloc-id 800
OLT CLI(DEV0 CH1 LOC-ONU0)#
OLT CLI(DEV0 CH1 LOC-ONU0)# exit
OLT CLI(DEV0 CH1)#
OLT CLI(DEV0 CH1)# port 800 alloc-id 800
OLT CLI(DEV0 CH1)#
OLT CLI(DEV0 CH1)# onu-omci 0
OLT CLI(DEV0 CH1 ONU(OMCI)0)#
OLT CLI(DEV0 CH1 ONU(OMCI)0)# t-cont set slot-id 128 t-cont-id 0 alloc-id 800
Response result: Command success
OLT CLI(DEV0 CH1 ONU(OMCI)0)#
OLT CLI(DEV0 CH1 ONU(OMCI)0)# mac-bridge-service-profile create slot-id 0 bridge-group-id 1
spanning-tree-ind true learning-ind true atm-port-bridging-ind true priority 32000 max-age 1536 hello-
time 256 forward-delay 1024 unknown-mac-address-discard false mac-learning-depth 255 dynamic-
filtering-ageing-time 1000
Response result: Command success
OLT CLI(DEV0 CH1 ONU(OMCI)0)#

```

OLT CLI(DEV0 CH1 ONU(OMCI)0)# mac-bridge-pcd create instance 1 bridge-id-ptr 1 port-num 1 tp-type lan tp-ptr 257 port-priority 2 port-path-cost 32 port-spanning-tree-ind true encap-method llc lanfcs-ind forward

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# mac-bridge-pcd create instance 2 bridge-id-ptr 1 port-num 2 tp-type gem tp-ptr 2 port-priority 0 port-path-cost 1 port-spanning-tree-ind true encap-method llc lanfcs-ind forward

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# gem-port-network-ctp create instance 2 port-id 800 t-cont-ptr 32768 direction bidirectional traffic-mgmt-ptr-ustream 0 traffic-descriptor-profile-ptr 0 priority-queue-ptr-downstream 0 traffic-descriptor-profile-ds-ptr 0 enc-key-ring 0

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# gem-interworking-termination-point create instance 2 gem-port-nwk-ctp-conn-ptr 2 interwork-option mac-bridge-lan service-profile-ptr 1 interwork-tp-ptr 0 gal-profile-ptr 0

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# vlan-tagging-filter-data create instance 2 forward-operation h-vid-a vlan-tag1 806 vlan-priority1 7 vlan-tag2 null vlan-priority2 null vlan-tag3 null vlan-priority3 null vlan-tag4 null vlan-priority4 null vlan-tag5 null vlan-priority5 null vlan-tag6 null vlan-priority6 null vlan-tag7 null vlan-priority7 null vlan-tag8 null vlan-priority8 null vlan-tag9 null vlan-priority9 null vlan-tag10 null vlan-priority10 null vlan-tag11 null vlan-priority11 null vlan-tag12 null vlan-priority12 null

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# extended-vlan-tagging-operation-config-data create instance 257 association-type pptp-eth-uni associated-me-ptr 257

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# extended-vlan-tagging-operation-config-data set instance 257 operations-entry filter-outer-prio filter-prio-no-tag filter-outer-vid none filter-outer-tpid none filter-inner-prio filter-prio-none filter-inner-vid 806 filter-inner-tpid none filter-ethertype none treatment-tag-to-remove 1 treatment-outer-prio none treatment-outer-vid copy-from-inner treatment-outer-tpid tpid-de-copy-from-outer treatment-inner-prio 0 treatment-inner-vid 806 treatment-inner-tpid tpid-de-copy-from-inner

Response result: Command success

OLT CLI(DEV0 CH1 ONU(OMCI)0)#

OLT CLI(DEV0 CH1 ONU(OMCI)0)# exit

OLT CLI(DEV0 CH1)#

OLT CLI(DEV0 CH1)# vlan uplink configuration port-id 800 min-cos 0 max-cos 7 de-bit disable primary-tag-handling false

OLT CLI(DEV0 CH1)# vlan uplink handling port-id 800 primary-vlan none destination datapath c-vlan-handling no-change s-vlan-handling no-change new-c-vlan 0 new-s-vlan 0

OLT CLI(DEV0 CH1)#

OLT CLI(DEV0 CH1)# policing downstream profile committed-max-bw 49984 committed-burst-size 1023 excess-max-bw 49984 excess-burst-size 1023

OLT_device_id: 0

OLT_channel_id: 1

downstream_profile_index: 68

OLT CLI(DEV0 CH1)#

OLT CLI(DEV0 CH1)# policing downstream port-configuration entity port-id 800 ds-profile-index 68

Downstream profile assigned

OLT CLI(DEV0 CH1)#

OLT CLI(DEV0 CH1)# exit

OLT CLI(DEV0)#

OLT CLI(DEV0)# pon

OLT CLI(OLT0 PON)# dba pythagoras 1

```

OLT CLI(OLT0 CH1 PON-DBA(Pythagoras))#
OLT CLI(OLT0 CH1 PON-DBA(Pythagoras))# sla 800 service data status-report nsr gr-bw 45 gr-fine
0 be-bw 50 be-fine 0
OLT CLI(OLT0 CH1 PON-DBA(Pythagoras))#
OLT CLI(OLT0 CH1 PON-DBA(Pythagoras))# end
OLT CLI#

```

A continuación, para finalizar la prueba se va a proceder al borrado de los servicios creados, para probar que se eliminan correctamente. Para ello desde el servicio API REST de ONOS, que hemos utilizado en otros capítulos borramos los flows, lo que hace que se borren los servicios asociados a estos flows. En la Figura 62 se muestra que el borrado de los servicios ha sido satisfactorio en el agente OpenFlow sobre ambas ONTs/ONUs.

```

OFPT_METER_MOD
OFPT_FLOW_MOD
OFPT_FLOW_MOD
OFPT_BARRIER_REPLY

```

La configuracion de la ONU con MAC 78:3d:5b:01:f7:30 ha sido borrada.

La configuracion de la ONU con MAC 78:3d:5b:01:f7:28 ha sido borrada.

Figura 62: Servicios borrados en la red GPON

Además, sabemos que el borrado es correcto ya que en los ficheros de backup de los servicios están vacíos, los archivos que contenían los comandos ejecutados en el CLI de la OLT ya no están y el fichero XML solo contiene las MAC de las ONUs de la red. El contenido de este último se muestra en la Figura 63.

```

- <RedGPON>
  <ONU MAC="54-4c-52-49-5b-01-f7-30" />
  <ONU MAC="54-4c-52-49-5b-01-f6-d8" />
  <ONU MAC="54-4c-52-49-5b-02-e9-ca" />
  <ONU MAC="54-4c-52-49-5b-03-fc-82" />
  <ONU MAC="54-4c-52-49-5b-01-f7-28" />
</RedGPON>

```

Figura 63: Contenido del fichero XML tras el borrado de los servicios

6

Conclusiones y líneas futuras

6.1 Conclusiones

En el presente Trabajo de Fin de Grado se ha diseñado y programado un agente OpenFlow capaz de configurar los servicios en redes GPON a través de controlador SDN ONOS.

En este proyecto hemos pasado todas las etapas para la creación de un software informático. Primero con la fase de investigación de la red y de las tecnologías SDN junto con sus protocolos y componentes. Después pasamos a la fase de análisis de VOLTHA, para tratar de crear una API para gestionar la red PON. Pero en este punto nos encontramos con varios problemas que hicieron que tuviésemos que replantear el proyecto con otro nuevo objetivo pero con el mismo resultado, esto es, crear un agente OpenFlow de código abierto para gestionar y configurar redes PON y en particular redes GPON. Entonces comenzamos con la fase de análisis del controlador ONOS mediante la realización de varias pruebas con redes de switches virtuales. A continuación, cuando ya teníamos claro como era el funcionamiento el controlador, nos pusimos a diseñar y programar el agente OpenFlow, no sin antes analizar previamente el protocolo OpenFlow 1.3. Tras esto adaptamos la API de la red GPON para que al recibir las órdenes enviadas por el controlador SDN mediante el protocolo OpenFlow en el agente, éstas se volcasen y se creasen los servicios correctamente en la red GPON del laboratorio. Cabe destacar que el agente OpenFlow desarrollado presenta una gran potencialidad, pues podrá ser instalado para gestionar otras redes PON de modo que pueda ser fácilmente adaptado a los parámetros de configuración que requieran las APIs de diferentes OLTs de distintos fabricantes.

Durante la realización de este proyecto nos hemos dado cuenta de las dificultades que suponen los trabajos de investigación para conseguir sacar adelante un proyecto de estas características.

6.2 Líneas Futuras

A partir del agente que hemos desarrollado, se podría seguir con en esta línea de investigación.

Se podría añadir más funcionalidades al agente, tales como que sea capaz de procesar más tipos de mensajes OpenFlow para incrementar las funcionalidades. O que se puedan añadir flows con más tipos de campos *Match*, instrucciones o acciones de las que nosotros hemos considerado.

Por otro lado, también se podría trabajar en la extensión del protocolo OpenFlow actual para integrar otros parámetros de configuración propios del estándar PON, puesto que este protocolo en sí no soporta campos, instrucciones o acciones para gestionar parámetros directos de la capa óptica PON, tales como puertos GEM o colas de prioridad TCONT para encolar el tráfico en diferentes niveles de prioridad.

Finalmente, se podría adaptar el agente para que fuese compatible con otros controladores SDN, aunque ONOS resulta ser la opción preferida hoy en día en el campo de las redes ópticas. Además, se podría probar el agente sobre otras redes PON comerciales de diferentes fabricantes.

7 Bibliografía

- [1] FTTH Council, "FTTH Council Europe-Panorama". [Online].
<https://www.ftthcouncil.eu/documents/FTTH%20Council%20Europe%20-%20Panorama%20at%20September%202019%20-%20Webinar%20Version4.pdf>
- [2] FTTH Council, "FTTH Forecast for EUROPE". [Online].
[https://www.ftthcouncil.eu/documents/Reports/2019/FTTH Council Europe - Forecast for EUROPE 2020-2025.pdf](https://www.ftthcouncil.eu/documents/Reports/2019/FTTH%20Council%20Europe%20-%20Forecast%20for%20EUROPE%202020-2025.pdf)
- [3] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey". Security and Communication Networks., Diciembre 2016. [Online].
<https://onlinelibrary.wiley.com/doi/full/10.1002/sec.1737>
- [4] Página web de VOLTHA. [Online]. <https://www.opennetworking.org/voltha/>
- [5] Protocolo OpenFlow 1.3. [Online]. <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>
- [6] Página web de ONOS. [Online]. <https://www.opennetworking.org/onos/>
- [7] Página web de Mininet. [Online]. <http://mininet.org/>
- [8] Página web de Open vSwitch. [Online]. <http://www.openvswitch.org/>
- [9] Página web de Python. [Online]. <https://www.python.org/>
- [10] Página web de VOLTHA BBSim. [Online].
<https://wiki.opencord.org/pages/viewpage.action?pageId=4982251>
- [11] Librería Python-OpenFlow, Kytos SDN. [Online]. <https://github.com/kytos/python->

openflow

- [12] Página web de Docker. [Online]. <https://www.docker.com/>
- [13] ITU-T. G.984.1 : Gigabit-capable passive optical networks (GPON): General characteristics. [Online]. <https://www.itu.int/rec/T-REC-G.984.1>
- [14] G. Kramer, B. Mukherjee, and G. Pesavento, "IPACT: a dynamic protocol for an Ethernet PON (EPON)", IEEE Commun. Mag., vol. 40, no. 2, pp. 74–80, 2002.
- [15] M. McGarry, M. Reisslein, and M. Maier, "Ethernet passive optical network architectures and dynamic bandwidth allocation algorithms", IEEE Commun. Surveys & Tutorials, vol. 10, no. 3, pp. 46-60, 2008.
- [16] B. Lung, "Fiber to the Home Using a PON Infrastructure", IEEE/OSA Journal of Lightwave Technology, vol. 2, pp. 4568-4583, Diciembre 2006.
- [17] ONOS Wiki - Overview. [Online].
<https://wiki.onosproject.org/display/ONOS/ONOS+%3A+An+Overview>
- [18] ONOS Wiki - Arquitectura. [Online].
<https://wiki.onosproject.org/display/ONOS/System+Components>
- [19] Why OVS? [Online].
<https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst>
- [20] Network namespaces, open vSwitch, mininet. [Online]. <http://mobiquo.gsync.es/lab-sdn/materialesPrevios/sdn.pdf>
- [21] The Python Wiki. [Online]. <https://wiki.python.org/moin/>
- [22] Docker - What is a container. [Online]. <https://www.docker.com/resources/what-container>
- [23] ¿Cuanto gana un programador web en España? [Online].
<https://www.tucursogratias.net/cuanto-gana-un-programador-web-en-espana/>

-
- [24] Precios maquinas virtuales Linux - Azure. [Online]. <https://azure.microsoft.com/es-es/pricing/details/virtual-machines/linux/>
- [25] Página web de CORD. [Online]. <https://www.opennetworking.org/cord/>
- [26] VOLTHA - Github. [Online]. <https://github.com/opencord/voltha/tree/1.7.0>
- [27] VOLTHA Docker Networking. [Online].
<https://wiki.opencord.org/display/CORD/VOLTHA+Docker+Networking>
- [28] Página web de gRPC. [Online]. <https://grpc.io/>
- [29] VOLTHA Flow Install and Packet Out. [Online].
<https://wiki.opencord.org/display/CORD/VOLTHA+Flow+Install+and+Packet+Out>
- [30] Edgecore OpenOLT agent. [Online]. <https://guide.opencord.org/openolt/>
- [31] Installing ONOS on a single machine. [Online].
<https://wiki.onosproject.org/display/ONOS/Installing+on+a+single+machine>
- [32] VOLTHA voltha-bbsim - Github. [Online]. <https://github.com/opencord/voltha-bbsim>
- [33] VOLTHA bbsim - Github. [Online].
<https://github.com/opencord/bbsim/tree/v0.0.19>
- [34] VOLTHA voltha-go - Github. [Online]. <https://github.com/opencord/voltha-go/tree/v2.3.3>
- [35] Página web de Go. [Online]. <https://golang.org/>
- [36] VOLTHA voltctl - Github. [Online]. <https://github.com/opencord/voltctl>
- [37] Open vSwitch Installation. [Online].
<http://docs.openvswitch.org/en/latest/intro/install/general/#installation-requirements>

-
- [38] Determining proper burst size for traffic policers. [Online].
https://www.juniper.net/documentation/en_US/junos/topics/concept/policer-mx-m120-m320-burstsize-determining.html
- [39] ONOS API. [Online].
<https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API>
- [40] Página web de Wireshark. [Online]. <https://www.wireshark.org/>
- [41] Ángel Gómez Aguado, "Diseño de un agente OpenFlow en una maqueta Red de Acceso Óptica real," 2017. [Online]. <http://uvadoc.uva.es/handle/10324/27578>
- [42] Craig Larman, UML y Patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado, Segunda ed., 2002.
- [43] Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm, design patterns elements of reusable object-oriented software.: Addison-Wesley, 1994,
<http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf>.
- [44] Librería keyboard Python. [Online]. <https://pypi.org/project/keyboard/>

Anexo I: Flows por defecto de ONOS

En este Anexo mostramos los flows por defecto que crea ONOS en cada switch:

```
{
  "flows": [
    {
      "groupId": 0,
      "state": "ADDED",
      "life": 29,
      "liveType": "UNKNOWN",
      "lastSeen": 1594133188281,
      "packets": 14,
      "bytes": 588,
      "id": "281478909873038",
      "appId": "org.onosproject.core",
      "priority": 40000,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:00000000000000001",
      "tableId": 0,
      "tableName": "0",
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "CONTROLLER"
          }
        ],
        "clearDeferred": true,
        "deferred": []
      },
      "selector": {
        "criteria": [
          {
            "type": "ETH_TYPE",
            "ethType": "0x806"
          }
        ]
      }
    },
    {
      "groupId": 0,
      "state": "ADDED",
      "life": 30,
      "liveType": "UNKNOWN",
      "lastSeen": 1594133188281,
      "packets": 0,
      "bytes": 0,
      "id": "281477466379610",
      "appId": "org.onosproject.core",
      "priority": 40000,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:00000000000000001",
      "tableId": 0,
      "tableName": "0",
      "treatment": {
        "instructions": [
```

```

        {
            "type": "OUTPUT",
            "port": "CONTROLLER"
        }
    ],
    "clearDeferred": true,
    "deferred": []
},
"selector": {
    "criteria": [
        {
            "type": "ETH_TYPE",
            "ethType": "0x88cc"
        }
    ]
}
},
{
    "groupId": 0,
    "state": "ADDED",
    "life": 30,
    "liveType": "UNKNOWN",
    "lastSeen": 1594133188281,
    "packets": 0,
    "bytes": 0,
    "id": "281477029321583",
    "appId": "org.onosproject.core",
    "priority": 40000,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:0000000000000001",
    "tableId": 0,
    "tableName": "0",
    "treatment": {
        "instructions": [
            {
                "type": "OUTPUT",
                "port": "CONTROLLER"
            }
        ]
    },
    "clearDeferred": true,
    "deferred": []
},
"selector": {
    "criteria": [
        {
            "type": "ETH_TYPE",
            "ethType": "0x8942"
        }
    ]
}
}
]
}

```

Anexo II: Flows creados por fwd

En este Anexo mostramos los flows creados por la aplicación *fwd* en la prueba 3 del Capítulo 4:

```
{
  "flows": [
    {
      "groupId": 0,
      "state": "ADDED",
      "life": 4,
      "liveType": "UNKNOWN",
      "lastSeen": 1594134553262,
      "packets": 3,
      "bytes": 294,
      "id": "22236524227526657",
      "appId": "org.onosproject.fwd",
      "priority": 10,
      "timeout": 10,
      "isPermanent": false,
      "deviceId": "of:000000000000000001",
      "tableId": 0,
      "tableName": "0",
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "2"
          }
        ],
        "deferred": []
      },
      "selector": {
        "criteria": [
          {
            "type": "IN_PORT",
            "port": 1
          },
          {
            "type": "ETH_DST",
            "mac": "00:00:00:00:00:02"
          },
          {
            "type": "ETH_SRC",
            "mac": "00:00:00:00:00:01"
          }
        ]
      }
    },
    {
      "groupId": 0,
      "state": "ADDED",
      "life": 4,
      "liveType": "UNKNOWN",
      "lastSeen": 1594134553262,
      "packets": 2,
      "bytes": 196,
      "id": "22236524294947356",
```

```

"appId": "org.onosproject.fwd",
"priority": 10,
"timeout": 10,
"isPermanent": false,
"deviceId": "of:0000000000000001",
"tableId": 0,
"tableName": "0",
"treatment": {
  "instructions": [
    {
      "type": "OUTPUT",
      "port": "1"
    }
  ],
  "deferred": []
},
"selector": {
  "criteria": [
    {
      "type": "IN_PORT",
      "port": 2
    },
    {
      "type": "ETH_DST",
      "mac": "00:00:00:00:00:01"
    },
    {
      "type": "ETH_SRC",
      "mac": "00:00:00:00:00:02"
    }
  ]
}
},
{
  "groupId": 0,
  "state": "ADDED",
  "life": 42,
  "liveType": "UNKNOWN",
  "lastSeen": 1594134553262,
  "packets": 7,
  "bytes": 686,
  "id": "281475012051420",
  "appId": "org.onosproject.core",
  "priority": 5,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:0000000000000001",
  "tableId": 0,
  "tableName": "0",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "CONTROLLER"
      }
    ]
  },
  "clearDeferred": true,
  "deferred": []
},
"selector": {
  "criteria": [

```

```
    {
      "type": "ETH_TYPE",
      "ethType": "0x800"
    }
  ]
}
]
```

Anexo III: Script de creación de flows y meters de la prueba 3

En este Anexo mostramos el script que se ha utilizado para instalar los flows y los meters en la prueba 3 del Capítulo 4:

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "deviceId": "of:000000000000000001",
  "unit": "KB_PER_SEC",
  "burst": false,
  "bands": [
    {
      "type": "DROP",
      "rate": "3000",
      "burstSize": "0",
      "prec": "0"
    }
  ]
}' 'http://localhost:8181/onos/v1/meters/of%3A000000000000000001'
```

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "flows": [
    {
      "priority": 5,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:000000000000000001",
      "tableId": 0,
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "CONTROLLER"
          }
        ],
        "clearDeferred": true,
        "deferred": []
      },
      "selector": {
        "criteria": [
          {
            "type": "ETH_TYPE",
            "ethType": "0x800"
          }
        ]
      }
    },
    {
      "priority": 10,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:000000000000000001",
      "treatment": {
        "instructions": [
          {
```

```

        "type": "OUTPUT",
        "port": "2"
    },
    {
        "type": "METER",
        "meterId": "1"
    }
],
"deferred": []
},
"selector": {
    "criteria": [
        {
            "type": "IN_PORT",
            "port": 1
        },
        {
            "type": "ETH_DST",
            "mac": "00:00:00:00:00:02"
        },
        {
            "type": "ETH_SRC",
            "mac": "00:00:00:00:00:01"
        }
    ]
}
},
{
    "priority": 10,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:0000000000000001",
    "treatment": {
        "instructions": [
            {
                "type": "OUTPUT",
                "port": "1"
            },
            {
                "type": "METER",
                "meterId": "1"
            }
        ]
    },
    "deferred": []
},
"selector": {
    "criteria": [
        {
            "type": "IN_PORT",
            "port": 2
        },
        {
            "type": "ETH_DST",
            "mac": "00:00:00:00:00:01"
        },
        {
            "type": "ETH_SRC",
            "mac": "00:00:00:00:00:02"
        }
    ]
}
}

```

```
    }  
  ]  
}  
' 'http://localhost:8181/onos/v1/flows?appId=org.onosproject.fwd'
```

Anexo IV: Script de la primera prueba para el Agente OpenFlow

En este Anexo mostraremos el script que se ha usado para la realización de la primera prueba comentada en el Apartado 5.5:

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "deviceId": "of:0000000000000005",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "DROP",
      "rate": "204800",
      "burstSize": "0",
      "prec": "0"
    }
  ]
}' 'http://localhost:8181/onos/v1/meters/of%3A0000000000000005'
```

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "deviceId": "of:0000000000000005",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "REMARK",
      "rate": "0",
      "burstSize": "0",
      "prec": "1"
    }
  ]
}' 'http://localhost:8181/onos/v1/meters/of%3A0000000000000005'
```

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "flows": [
    {
      "priority": 10,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:0000000000000005",
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "2"
          },
          {
            "type": "METER",
            "meterId": "2"
          }
        ]
      },
      "deferred": []
    }
  ]
}'
```

```

},
"selector": {
  "criteria": [
    {
      "type": "IN_PORT",
      "port": 1
    },
    {
      "type": "ETH_DST",
      "mac": "00:00:00:00:00:02"
    },
    {
      "type": "ETH_SRC",
      "mac": "00:00:00:00:00:01"
    },
    {
      "type": "VLAN_VID",
      "vlanId": "1"
    }
  ]
}
},
{
  "priority": 10,
  "timeout": 0,
  "isPermanent": true,
  "deviceId": "of:000000000000000005",
  "treatment": {
    "instructions": [
      {
        "type": "OUTPUT",
        "port": "1"
      },
      {
        "type": "METER",
        "meterId": "1"
      }
    ],
    "deferred": []
  },
  "selector": {
    "criteria": [
      {
        "type": "IN_PORT",
        "port": 2
      },
      {
        "type": "ETH_TYPE",
        "ethType": "0x800"
      },
      {
        "type": "ETH_DST",
        "mac": "00:00:00:00:00:01"
      },
      {
        "type": "ETH_SRC",
        "mac": "00:00:00:00:00:02"
      },
      {
        "type": "VLAN_VID",
        "vlanId": "1"
      }
    ]
  }
}

```

```
    },
    {
      "type": "VLAN_PCP",
      "priority": "1"
    },
    {
      "type": "IPV4_SRC",
      "ip": "10.0.0.1/32"
    },
    {
      "type": "IPV4_DST",
      "ip": "10.0.0.0/24"
    }
  ]
}
]
}
' 'http://localhost:8181/onos/v1/flows?appId=org.onosproject.fwd'
```

Anexo V: Script de la segunda prueba para el Agente OpenFlow

En este Anexo mostraremos el script que se ha usado para la realización de la segunda prueba comentada en el Capítulo 5.5:

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "deviceId": "of:000000000000000005",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "DROP",
      "rate": "50000",
      "burstSize": "0",
      "prec": "0"
    }
  ]
}' 'http://10.0.103.56:8181/onos/v1/meters/of%3A000000000000000005'
```

```
curl -u onos:rocks -X POST --header 'Content-Type: application/json' --
header 'Accept: application/json' -d '{
  "flows": [
    {
      "priority": 10,
      "timeout": 0,
      "isPermanent": true,
      "deviceId": "of:000000000000000005",
      "treatment": {
        "instructions": [
          {
            "type": "OUTPUT",
            "port": "2"
          },
          {
            "type": "METER",
            "meterId": "1"
          }
        ],
        "deferred": []
      },
      "selector": {
        "criteria": [
          {
            "type": "IN_PORT",
            "port": 1
          },
          {
            "type": "ETH_TYPE",
            "ethType": "0x800"
          },
          {
            "type": "ETH_DST",
            "mac": "78:3d:5b:01:f7:30"
          }
        ],
      }
    }
  ]
}'
```

```

        {
          "type": "ETH_SRC",
          "mac": "00:00:00:00:00:01"
        },
        {
          "type": "VLAN_VID",
          "vlanId": "806"
        },
        {
          "type": "IPV4_DST",
          "ip": "10.0.0.0/24"
        }
      ]
    },
  },
  {
    "priority": 10,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:000000000000000005",
    "treatment": {
      "instructions": [
        {
          "type": "OUTPUT",
          "port": "1"
        },
        {
          "type": "METER",
          "meterId": "1"
        }
      ],
      "deferred": []
    },
    "selector": {
      "criteria": [
        {
          "type": "IN_PORT",
          "port": 2
        },
        {
          "type": "ETH_TYPE",
          "ethType": "0x800"
        },
        {
          "type": "ETH_DST",
          "mac": "78:3d:5b:01:f7:28"
        },
        {
          "type": "ETH_SRC",
          "mac": "00:00:00:00:00:02"
        },
        {
          "type": "VLAN_VID",
          "vlanId": "806"
        },
        {
          "type": "VLAN_PCP",
          "priority": "1"
        },
        {
          "type": "IPV4_SRC",

```

```
        "ip": "11.0.0.1/32"
      },
      {
        "type": "IPV4_DST",
        "ip": "10.0.0.255/32"
      }
    ]
  }
]
}' 'http://10.0.103.56:8181/onos/v1/flows?appId=org.onosproject.fwd'
```