

UNIVERSIDAD DE VALLADOLID



E.T.S.I. TELECOMUNICACIÓN

## TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE  
TELECOMUNICACIÓN

# **Evolución en la integración del estándar OpenFlow en una maqueta de red GPON**

Autor:

**Don Javier Néstor Azofra Ovejero**

Tutor:

**Doña Noemí Merayo Álvarez**



---

**TÍTULO: Evolución en la integración del estándar OpenFlow en una maqueta de red GPON**

**AUTOR: Don Javier Néstor Azofra Ovejero**

**TUTOR: Doña Noemí Merayo Álvarez**

**DEPARTAMENTO: Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

---

---

**TRIBUNAL**

**PRESIDENTE: Ignacio de Miguel Jiménez**

**SECRETARIO: Noemí Merayo Álvarez**

**VOCAL: Juan Carlos Aguado Manzano**

**SUPLENTE: Ramón J. Durán Barroso**

**SUPLENTE: Rubén M. Lorenzo**

---

---

**FECHA: 20 de Junio de 2018**

**CALIFICACIÓN:**

---

**Resumen de TFG**

La investigación realizada en este Trabajo de Fin de Grado se centra en la transformación de una red GPON a una red SDN utilizando el protocolo OpenFlow (SDN-GPON). Con esto conseguimos una disociación entre el plano de control encargado de enrutar los paquetes y el plano de datos en la red de acceso. Para ello, se ha implementado un router basado en Linux en el ordenador central y se han instalado varios switches virtuales OVS (*Open Virtual Switch*) que pueden utilizar el protocolo OpenFlow y se comunican con un controlador central OpenFlow, en nuestro caso OpenDayLight, situado en la red troncal. A través de este nuevo escenario de red SDN seremos capaces de configurar y gestionar servicios y perfiles de abonado en la red de acceso mediante OpenFlow.

Posteriormente, se continuó la investigación proponiendo un nuevo modelo de negocio para las operadoras, en los que los usuarios finales puedan realizar peticiones en tiempo real sobre ciertos parámetros de sus servicios contratados, previo pago por uso de la red y los recursos de la misma. Para ello, se programó una sencilla interfaz web para que el cliente pueda realizar peticiones dinámicas de su ancho de banda contratado en tiempo real gracias a la comunicación entre los switches virtuales y el controlador OpenDayLight por medio de sockets TCP programados en Python y PHP.

Para finalizar la investigación, se analizó la correcta implementación de la gestión y configuración de servicios mediante OpenFlow utilizando la utilidad Wireshark, con la que se pueden visualizar los mensajes OpenFlow de la comunicación entre el switch virtual y el controlador OpenDayLight y a la vez analizar parámetros tales como el ancho de banda garantizado de todos los servicios soportados en la red SDN-GPON.

### **Palabras clave**

SDN (*Redes Diseñadas por Software*), GPON (*Red Óptica Pasiva con Capacidad de Gigabit*), OLT (*Terminación Óptica de Línea*), ONU (*Unidad de Red Óptica*), OpenFlow, OpenDayLight, Python, PHP, Wireshark.

## **Abstract**

The research carried out in this end-of-degree project addresses the transformation of a GPON network into a SDN network using the OpenFlow protocol (SDN-GPON). We achieve a disassociation between the forwarding process of network packets (Data Plane) from the routing process (Control plane). In order to do this, we have implemented a router based on Linux in the central computer and we have installed several virtual switches OVS that are OpenFlow-enabled and they can communicate with the OpenDayLight controller located in the network backbone. This new scenario allows us to configure and manage services and subscriber profiles in the access network through OpenFlow.

Subsequently, we proposed a new business model for the Internet Service Providers, in which clients are able to make requests in real time to change some of the parameters of their Internet service, upon payment of the required fees. In order to do this, a simple web interface was coded so that the clients can dynamically manage their default hired bandwidth, thanks to this aforementioned communication between the virtual switches and the OpenDayLight controller through TCP sockets coded in Python and PHP.

To end with, the implementation of the dynamic bandwidth variation has been analyzed through the Wireshark utility, in which we can observe the OpenFlow messages between the virtual switches and the OpenDayLight controller and simultaneously, quantitatively measure the guaranteed bandwidth of all the services provided by the SDN-GPON network.

## **Keywords**

SDN (*Software-designed networks*), GPON (*Gigabit-capable Passive Optical Networks*), OLT (*Optical Line Termination*), ONU (*Optical Network Unit*), OpenFlow, OpenDayLight, Python, PHP, Wireshark.



# Agradecimientos

*A mis padres y mi hermana, por apoyarme y educarme en la cultura del esfuerzo y del pensamiento crítico.*

*A Alejandra, por confiar en mí siempre incluso en los momentos más duros de esta carrera. Te quiero mucho.*

*A mis amigas y amigos, por interesarse y preguntarme sobre este proyecto sin entender ni una palabra.*

*A Noemí y a Juan Carlos, por su inspiración, sus consejos y su flexibilidad y su guía para que este proyecto haya llegado a buen puerto.*

*Muchas gracias de corazón.*





# Índice

<b>1</b>	<b>Introducción.....</b>	<b>1</b>
1.1	Motivación.....	1
1.2	Objetivos.....	2
1.2.1	Objetivo General.....	2
1.2.2	Objetivos Específicos .....	2
1.3	Fases y Métodos .....	3
1.3.1	Fase de Análisis .....	3
1.3.2	Fase de Implementación .....	3
1.3.3	Fase de Pruebas.....	4
1.3.4	Fase de Realización de los Informes.....	4
1.4	Estructura de la Memoria del TFG .....	4
<b>2</b>	<b>Metodología y herramientas de trabajo .....</b>	<b>7</b>
2.1	Introducción.....	7
2.2	Montaje real y modos de gestión de la red GPON .....	8
2.2.1	Estructura y componentes de la red de acceso GPON.....	8
2.2.2	Modos de gestión de la red GPON .....	10
2.3	Principios de funcionamiento del estándar OpenFlow .....	11
2.4	Controladores OpenFlow: OpenDayLight.....	12
2.5	Open Virtual Switch (Open vSwitch – OVS).....	13
2.6	Metodología de trabajo .....	14
2.6.1	Programación de Linux Kernel Routing y un servidor DHCP en el ordenador central .....	14
2.6.2	Instalación del OVS en el ordenador central conectado al OLT .....	15
2.6.3	Montaje de las Raspberry Pi e instalación del OVS del lado de las ONTs .. .....	15

---

2.6.4	Conexión de los OVS's con el controlador OpenDayLight .....	16
2.6.5	Programación y despliegue de las tablas de flujo en el controlador OpenDayLight .....	17
2.6.6	Diseño de una interfaz web de usuario para gestionar de forma dinámica requisitos de servicio .....	18
2.7	Conclusiones .....	20
<b>3</b>	<b>Definición del escenario SDN en la red GPON .....</b>	<b>21</b>
3.1	Introducción .....	21
3.2	Estado del arte SDN en redes PON .....	21
3.3	Descripción del escenario SDN en la red GPON .....	22
3.4	Implementación del router y el servidor DHCP .....	25
3.5	Implementación del OLT basado en SDN con Open vSwitch (SDN – OLT)..	33
3.6	Implementación del ONT basado en SDN con Open vSwitch (SDN – ONT).	38
3.7	Conexión con el controlador OpenDayLight.....	40
3.8	Creación de flujos y <i>meters</i> en OpenFlow .....	40
3.9	Conclusiones .....	46
<b>4</b>	<b>Caso de uso: Gestión SDN de la QoS en la red GPON.....</b>	<b>47</b>
4.1	Introducción .....	47
4.2	Descripción del escenario de red experimental .....	47
4.3	Descripción del caso de uso.....	48
4.4	Desarrollo de una aplicación en el usuario para la gestión dinámica de los servicios contratados.....	53
4.5	Discusión de resultados experimentales .....	56
4.5.1	Análisis de resultados en la ONT 1 .....	58
4.5.2	Análisis de resultados en la ONT 2 .....	72
4.6	Conclusiones .....	82
<b>5</b>	<b>Conclusiones y Líneas Futuras.....</b>	<b>85</b>
5.1	Conclusiones .....	85

---

5.2	Líneas Futuras .....	86
<b>6</b>	<b>Bibliografía.....</b>	<b>89</b>
<b>Anexo I.....</b>	<b>.....</b>	<b>95</b>
	Script para habilitar el encaminamiento .....	95
<b>Anexo II .....</b>	<b>.....</b>	<b>96</b>
	Ficheros de configuración DHCP .....	96
<b>Anexo III.....</b>	<b>.....</b>	<b>97</b>
	Aplicación web en HTML5 .....	97
<b>Anexo IV .....</b>	<b>.....</b>	<b>99</b>
	Aplicación en PHP .....	99
<b>Anexo V .....</b>	<b>.....</b>	<b>100</b>
	Aplicación en Python.....	100

# Índice de figuras

Figura 1: Esquema general de la red de acceso GPON desplegada en el laboratorio .....	9
Figura 2: Activación del modo de gestión CLI para la configuración de la red GPON ...	10
Figura 3: Interfaz gráfica del controlador OpenDayLight .....	12
Figura 4: Tabla de versiones soportadas de OpenFlow para cada versión de OVS .....	14
Figura 5: Raspberry Pi 3 Model B .....	16
Figura 6: Estructura de las diferentes subredes presentes en nuestro escenario de red real .....	17
Figura 7: Captura de la interfaz gráfica de YangUI.....	18
Figura 8: Captura de la interfaz gráfica programada en HTML5/Javascript/PHP .....	20
Figura 9: Escenario LAN tradicional vs Escenario VLAN .....	23
Figura 10: Escenario GPON clásico .....	24
Figura 11: Escenario SDN–GPON implementado sobre la maqueta GPON de 25 kilómetros. ....	25
Figura 12: Trama 802.1Q con campos Prioridad, CFI y VLAN ID .....	27
Figura 13: Pasos en la manipulación del campo prioridad de la trama VLAN 802.1Q ...	29
Figura 14: Sockets en la capa de protocolos TCP/IP .....	30
Figura 15: Topología de red centrada en el Linux Kernel router implementado en la máquina central.....	31
Figura 16: Implementación típica de un switch virtual .....	34
Figura 17: Implementación errónea del switch virtual en la oficina central .....	35
Figura 18: Implementación del switch virtual correcta .....	36
Figura 19: Captura del comando ifconfig -a.....	36
Figura 20: Topología en el switch virtual de la ONT .....	39
Figura 21: Estructura de árbol en la interfaz YangUI.....	41
Figura 22: Campos relacionados con un flujo .....	42

Figura 23: Campos del desplegable <i>match</i> en YangUI.....	43
Figura 24: Distintas instrucciones en YangUI.....	43
Figura 25: <i>output-action-case</i> en YangUI .....	44
Figura 26: Acciones previstas en el protocolo OpenFlow 1.3 para <i>output-action-case</i> ...	45
Figura 27: Campos de un <i>meter</i> en YangUI .....	45
Figura 28: Escenario de red experimental SDN-GPON .....	47
Figura 29: Esquema de los diferentes campos de los flujos que vamos a utilizar en el caso de uso .....	49
Figura 30: Interfaces con varias VLAN en el lado WAN en nuestra topología de red ....	50
Figura 31: Campo <i>match</i> del flujo .....	51
Figura 32: <i>Meter</i> configurado a 40 Mbps .....	53
Figura 33: Página web para la gestión dinámica del ancho de banda.....	55
Figura 34: Mensaje de confirmación en PHP .....	56
Figura 35: Información sobre el paquete OFPMP_FLOW del flujo por defecto de 10 Mbps .....	59
Figura 36: Información sobre el paquete OFPMP_METER_CONFIG con ID 10 .....	60
Figura 37: Captura de la interfaz web solicitando el ancho de banda a 20 Mbps .....	61
Figura 38: Información sobre el paquete OFPT_FLOW_MOD.....	61
Figura 39: Información sobre el <i>meter</i> de la Figura 38 .....	62
Figura 40: Información sobre el OFPT_FLOW_MOD del segundo 600 .....	63
Figura 41: Gráfica del throughput de la conexión TCP en la ONT 1 sentido <i>upstream</i> ..	64
Figura 42: Media de la transmisión configurada a 10 Mbps .....	64
Figura 43: Media de la transmisión configurada a 20 Mbps .....	65
Figura 44: Pico en el segundo 300 de la comunicación en la ONT 1 sentido <i>upstream</i> ..	66
Figura 45: Pico en el segundo 600 de la comunicación en la ONT 1 sentido <i>upstream</i> ..	66
Figura 46: Mensaje OFPMP_FLOW en el switch virtual de la Oficina Central.....	68
Figura 47: Mensaje OFPT_FLOW_MOD en el switch de la oficina central .....	69

Figura 48: Gráfica del throughput de la conexión TCP en la ONT 1 sentido <i>downstream</i> .....	70
Figura 49: Gráfica del rizado detallado de 10 Mbps en la ONT 1 en sentido <i>downstream</i> .....	70
Figura 50: Gráfica del rizado detallado de 20 Mbps en la ONT 1 en sentido <i>downstream</i> .....	71
Figura 51: Pico en el segundo 300 de la comunicación en la ONT 1 sentido <i>downstream</i> .....	71
Figura 52: Pico en el segundo 300 de la comunicación en la ONT 1 sentido <i>downstream</i> .....	72
Figura 53: Información sobre el paquete OFPMP_FLOW del flujo por defecto de 20 Mbps .....	73
Figura 54: Información sobre el <i>meter</i> de la Figura 53 .....	73
Figura 55: Mensaje OFPT_FLOW_MOD en el segundo 301.23 .....	74
Figura 56: <i>Meter</i> relacionado con la Figura 55 .....	74
Figura 57: Gráfica del <i>throughput</i> en la conexión TCP de la ONT 2 en sentido <i>upstream</i> .....	75
Figura 58: Media de la transmisión configurada a 20 Mbps .....	76
Figura 59: Media de la transmisión configurada a 35 Mbps .....	76
Figura 60: Pico en el segundo 300 de la comunicación de la ONT 2 .....	77
Figura 61: Pico en el segundo 300 de la comunicación de la ONT 2 .....	77
Figura 62: Flujo por defecto en la ONT 2 en sentido <i>downstream</i> .....	78
Figura 63: Flujo <i>premium</i> en la ONT 2 en sentido <i>downstream</i> .....	79
Figura 64: Gráfica del <i>throughput</i> en la conexión TCP de la ONT 2 en sentido <i>downstream</i> .....	80
Figura 65: Pico en el segundo 300 de la comunicación con la ONT 2 en sentido <i>downstream</i> .....	81
Figura 66: Pico en el segundo 600 de la comunicación con la ONT 2 en sentido <i>downstream</i> .....	81
Figura 67: Gráfica del rizado detallado de 20 Mbps en la ONT 2 en sentido <i>downstream</i> .....	82
Figura 68: Gráfica del rizado detallado de 35 Mbps en la ONT 2 en sentido <i>downstream</i> .....	82

---

Figura 69: Esquema MPLS clásico frente a MPLS con OpenFlow .....	87
Figura 70: Topología virtualizada en un escenario de red SDN en una red GPON .....	88

# 1

## Introducción

### 1.1 Motivación

En este Trabajo de Fin de Grado se ha llevado a cabo la implementación de un escenario SDN (*Software Designed Network*) en una red de acceso GPON (*Gigabit-capable Passive Optical Network*) a través del protocolo OpenFlow. Esta red de acceso está situada en el laboratorio de Comunicaciones Ópticas (2L007) de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

Un escenario SDN implica que el plano de control y el plano de datos están separado, estando el plano de control separado por software en un controlador (en nuestro caso, un controlador OpenFlow). OpenFlow es un estándar abierto que permite la creación de diferentes protocolos experimentales. En routers y switches clásicos, el *datapath* (envío de datos de un lugar a otro) y el *control path* (toma de decisiones de encaminamiento) ocurrían en el mismo equipo [1]. Utilizando OpenFlow, el *control path* se puede mover a un controlador que se comunicará con el switch o router donde aún reside el *datapath* utilizando mensajes OpenFlow. Una de las grandes ventajas de este diseño es que un solo controlador podría operar sobre diferentes switches o routers simultáneamente, permitiendo que las conexiones e interacciones entre ellos puedan variar dependiendo del estado de la red.

Gracias a esta flexibilidad y control en tiempo real de nuestra red SDN-GPON, vamos a poder ofrecer al cliente final la posibilidad de gestionar dinámicamente su ancho de banda, pudiéndolo cambiar instantáneamente a través de una interfaz web programada por nosotros.



## 1.2 Objetivos

### 1.2.1 *Objetivo General*

El principal objetivo de este Trabajo de Fin de Grado es profundizar en la implementación de OpenFlow en una red de acceso GPON para continuar convirtiendo la red a un paradigma SDN en el que los planos de datos y los planos de control estén separados. Para ello, se van a configurar y gestionar servicios y requisitos de parámetros de calidad usando OpenFlow por lo que será necesaria la instalación de varios switches virtuales implementados en diferentes partes de la red GPON. En concreto, se colocarán antes de la OLT (*Optical Line Terminal*) y después de la ONT (*Optical Network Terminal*) para poder controlar y monitorizar el ingreso de cualquier paquete a nuestra red de acceso GPON.

Cuando nuestra red sea completamente SDN y podamos controlar los servicios y requisitos en tiempo real tanto en sentido *upstream* como en sentido *downstream* vamos a implementar una interfaz web para que el cliente controle dinámicamente su ancho de banda en tiempo real. De este modo, se introducirá un nuevo modelo de negocio para las operadoras en las que los usuarios puedan ser capaces de modificar sus requisitos en tiempo real previo pago por el uso de dichos recursos de la red.

Este objetivo general puede ser desglosado en otros más específicos, como se verá a continuación.

### 1.2.2 *Objetivos Específicos*

Con la realización de este estudio se han cubierto los siguientes objetivos específicos:

1. Análisis de la topología de red GPON previa a la realización de la investigación para su modificación.
2. Implementación de un router y del servidor DHCP.
3. Implementación de la capa SDN en la red GPON mediante la instalación y configuración de switches virtuales en el OLT y en cada ONT.

4. Comunicación entre los switches virtuales y el controlador OpenDayLight para su gestión OpenFlow.
5. Creación de servicios y gestión de requisitos mediante el estándar OpenFlow en la implementación SDN-GPON.

## 1.3 Fases y Métodos

La metodología a seguir para el desarrollo de los objetivos del Trabajo Fin de Grado ha constado fundamentalmente de las fases que se explicarán a continuación.

### 1.3.1 Fase de Análisis

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

- *Análisis de la topología de red GPON del laboratorio 2L007*: estudio de los distintos componentes de red y conexión entre ellos para implementar un router y un servidor DHCP dentro de dicha red.
- *Análisis del protocolo OpenFlow*: estudio del funcionamiento básico de este protocolo, sus diferentes versiones y tipos de controladores existentes.
- *Análisis de Open vSwitch*: estudio de su funcionamiento, implementación y configuración, así como el análisis entre sus distintas versiones y funcionalidades.
- *Análisis de la implementación SDN en la red GPON*: estudio del funcionamiento interno del controlador OpenDayLight y los switches virtuales, y la comunicación entre ellos para la gestión dinámica del ancho de banda.

### 1.3.2 Fase de Implementación

Esta fase tiene como objetivo la implementación de los distintos programas o máquinas necesarias para cumplir los objetivos específicos. Antes de cada implementación se habrá realizado el análisis específico necesario.

Para ello, vamos a necesitar manejar de manera óptima el sistema operativo Linux, ya que muchas de las implementaciones se basan en comandos y utilidades específicos a este sistema operativo, como *iptables* o *vconfig*.

### **1.3.3 Fase de Pruebas**

En esta fase final del proyecto se llevará a cabo una tarea de pruebas para observar el rendimiento de la red real gracias a dos herramientas, *iperf* y Wireshark. Con la primera vamos a poder medir la velocidad de la red y el tráfico real de subida y de bajada en nuestra red. Con la segunda vamos a poder observar los mensajes OpenFlow y vamos a poder obtener gráficas sobre el rendimiento de las comunicaciones TCP gracias a *tcptrace*, una utilidad incluida dentro de Wireshark.

### **1.3.4 Fase de Realización de los Informes**

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

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

## **1.4 Estructura de la Memoria del TFG**

El Capítulo 2 presenta el análisis realizado a nivel físico de la red GPON. Se describe el montaje general de la red con su topología y principales elementos. A continuación, en dicho capítulo también se describen brevemente las herramientas que se utilizarán en este trabajo y la metodología seguida.

El Capítulo 3 comienza definiendo el escenario SDN en nuestra red GPON y la implementación de los componentes necesarios para transformar la red convencional GPON separando el plano de control y el plano de datos.

En el Capítulo 4 se describe y se analiza un caso de uso concreto, partiendo de un escenario de red en que los clientes puedan realizar modificaciones en sus servicios contratados gracias a una interfaz web diseñada en HTML5/CSS/Javascript. También se mostrarán una serie de pruebas para poder valorar el rendimiento de la red para el ancho de banda contratado por el cliente.

Por último, en el Capítulo 5 se tratan las conclusiones y diferentes líneas futuras que pueden seguirse a partir de este trabajo.

Al final de la Memoria se pueden consultar la bibliografía y varios Anexos relevantes sobre los temas explicados en esta Memoria.



# 2

## Metodología y herramientas de trabajo

### 2.1 Introducción

En este capítulo de la memoria se realizará un análisis descriptivo de los componentes principales utilizados en este Trabajo de Fin de Grado, es decir, la red de acceso GPON, el protocolo OpenFlow, el controlador OpenDayLight y el switch virtual Open vSwitch. Además, también se describirá la metodología utilizada para desarrollar este trabajo.

La red de acceso GPON (del fabricante Telnet Redes Inteligentes [1]) está situada en el laboratorio de Comunicaciones Ópticas (2L007) de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid. Los principales elementos de esta red son el OLT (*Optical Line Termination*) y las ONUs/ONTs (*Optical Network Unit/Terminal*) a las que el OLT da servicios. Estos servicios pueden ser configurados de dos formas diferentes: mediante el uso del interfaz visual TGMS (*TELNET GPON Management System*) o, directamente, accediendo por línea de comandos mediante CLI (*Command Line Interface*) al OLT mediante su puerto de configuración. Sin embargo, en este proyecto, vamos a configurar y gestionar los servicios y perfiles de abonado de la red GPON ayudándonos del protocolo OpenFlow [1], convirtiendo nuestra red en una red SDN (*Software Defined Networks*), esto es, con una separación del plano de datos y el plano de control.

Por su parte, OpenFlow es un estándar que permite la creación de protocolos experimentales a partir de la separación que hace entre *datapath* (envío de un lugar a otro) y *controlpath* (toma de decisiones), lo que permite introducir SDN en redes. Estos términos serían similares a plano de control y plano de datos. OpenFlow cuenta con varias versiones

y controladores, siendo la versión 1.3 y el controlador OpenFlow OpenDayLight [2] los que se van a utilizar en este trabajo. Finalmente, el switch virtual Open vSwitch [3] es un switch virtual de código abierto, con varias funciones actualmente en desarrollo y/o experimentales que nos permite implementar un switch OpenFlow de manera transparente al usuario y a la red en general. Este switch virtual permite conectarse a los controladores OpenFlow para ser gestionado de forma fácil y transparente por ellos a través del protocolo OpenFlow. Este nuevo escenario de red es el que se desea implementar en nuestra red de acceso óptica real GPON.

## **2.2 Montaje real y modos de gestión de la red GPON**

En esta sección del capítulo se van a describir tanto la estructura y los componentes de la red de acceso como los diferentes modos de gestión con los que cuenta la red GPON desplegada en el laboratorio.

### **2.2.1 Estructura y componentes de la red de acceso GPON**

Una red de acceso es un conjunto de elementos que permiten a usuarios finales conectarse con los proveedores de servicio, de forma que éstos puedan darles a dichos usuarios los servicios que han contratado.

Nuestra red de acceso óptica GPON tiene una topología en árbol, donde un OLT da servicio a varias ONUs/ONTs. En el caso de la red de nuestro laboratorio, el OLT da servicio a un total de 4 ONUs en el puerto 0, siendo ampliable hasta un máximo de 64 ONUs por puerto. Como el OLT tiene un total de 4 puertos, el OLT puede soportar un total de hasta 256 ONUs. Otros elementos de la red será la fibra óptica monomodo que conecta entre sí los diferentes componentes ópticos y los *splitters* (divisores ópticos), responsables de dividir la señal para que cada uno de los puertos del OLT de servicio a varias ONUs. La fibra óptica desplegada en este testbed puede llegar a 25 km (acorde con el estándar GPON), y los divisores ópticos son de razón 1:8 (se dispone de dos aunque realmente está conectado uno actualmente). El aspecto general que presenta la red de acceso GPON es el que se muestra en la imagen de la Figura 1.

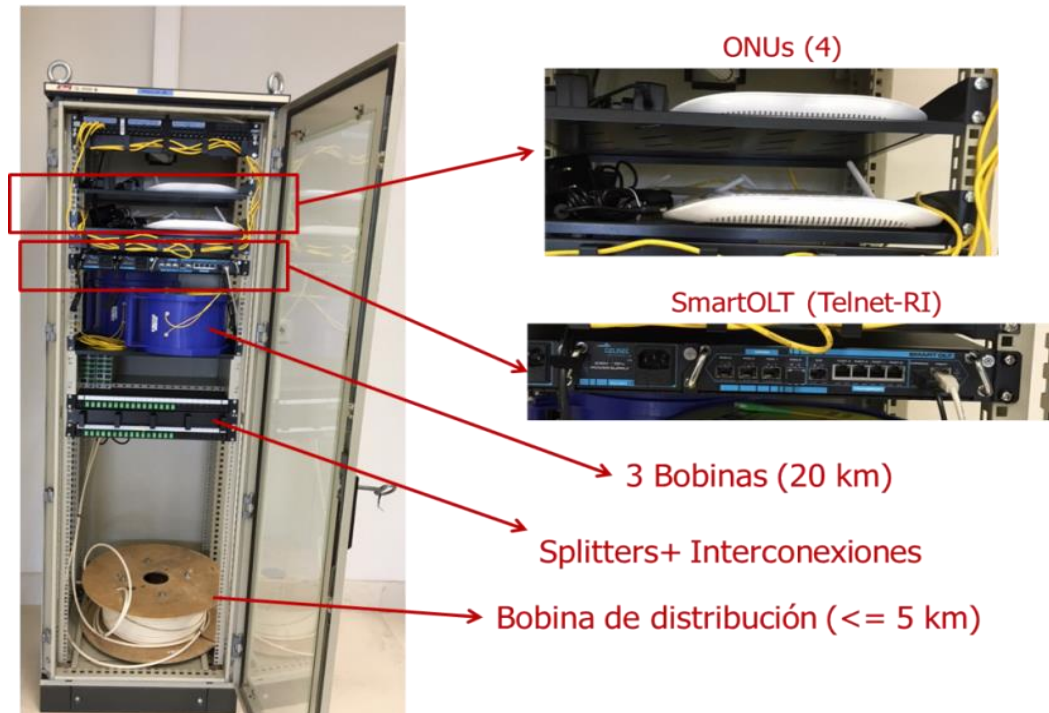


Figura 1: Esquema general de la red de acceso GPON desplegada en el laboratorio

Los componentes de la red de acceso han sido fabricados por la empresa TELNET Redes Inteligentes [1], de forma que la compatibilidad entre ellos está garantizada. Dependiendo de la dirección que lleven los datos, se puede hablar de dos flujos o canales diferentes:

- **Flujo descendente**: conocido como *downstream*, es el flujo que lleva los datos desde la oficina central (OLT) a los usuarios finales (las ONUs). Según el estándar GPON, la tasa máxima permitida en el *downstream* es de 2.5 Gbps y la longitud de onda de 1490 nm.
- **Flujo ascendente**: conocido como *upstream*, es el flujo que lleva los datos desde los usuarios a la oficina central (las ONUs son el origen). Según el estándar GPON, la tasa máxima permitida en el *upstream* es de 1,25 Gbps y la longitud de onda está situada en 1310 nm.

Es importante tener en cuenta estos dos flujos, ya que la suma total de las tasas de *upstream* y *downstream* dadas a cada uno de los servicios proporcionados a las diferentes ONTs/ONUs no pueden exceder los máximos impuestos por el

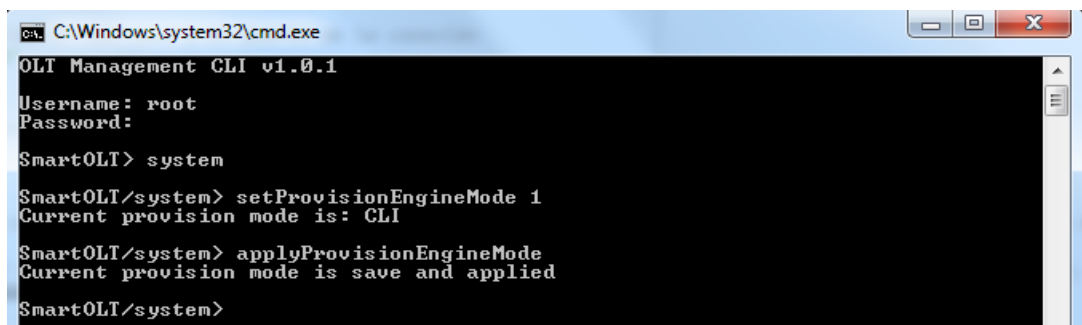


estándar GPON. En caso contrario, la red hará saltar un error en la configuración y puesta en marcha de la misma.

### 2.2.2 Modos de gestión de la red GPON

A la hora de configurar servicios y perfiles de abonado, existen dos métodos diferentes: el TGMS (*TELNET GPON Management System*) y el CLI (*Command Line Interface*). El TGMS es una máquina virtual que, ejecutándola en Virtualbox con la configuración adecuada, permite al ordenador que la contiene acceder a una página web desde la que se pueden configurar los servicios y los perfiles de abonado y asociarlos a cada una de las ONTs/ONUs conectadas al OLT. Además, también muestra información que puede ser de utilidad a nivel físico, como la potencia media recibida en el OLT y las ONUs. Este método está diseñado para que el usuario pueda configurar la red a través de un interfaz visual de un modo más amigable, rápido e intuitivo.

El otro método existente, denominado CLI (Figura 2), consiste en conectarse mediante telnet a la dirección IP (*Internet Protocol*) y puerto de configuración del OLT para poder configurar manualmente la red a base de comandos propios del estándar GPON. Este método es mucho más complicado que utilizar el TGMS, pero no requiere de la carga computacional que implica ejecutar la máquina virtual del TGMS y permite la creación de programas externos que configuren el OLT de forma automatizada introduciendo los comandos adecuados. La red GPON actualmente se modifica utilizando ambos métodos, pero nuestro objetivo es sustituir esta configuración de servicios clásica realizada con las herramientas dadas por el fabricante, por un control lo más análogo posible utilizando el protocolo OpenFlow.



```
C:\Windows\system32\cmd.exe
OLT Management CLI v1.0.1
Username: root
Password:
SmartOLT> system
SmartOLT/system> setProvisionEngineMode 1
Current provision mode is: CLI
SmartOLT/system> applyProvisionEngineMode
Current provision mode is save and applied
SmartOLT/system>
```

Figura 2: Activación del modo de gestión CLI para la configuración de la red GPON

## 2.3 Principios de funcionamiento del estándar OpenFlow

OpenFlow es un protocolo emergente y abierto de comunicaciones que permite la creación de nuevos protocolos gracias a la división que hace entre la transmisión de los datos de un punto a otro y la toma de decisiones de encaminamiento dentro de un *switch* o *router* [1]. Con OpenFlow, una parte del *datapath* reside en el mismo switch, pero es un controlador el que realiza las decisiones de encaminamiento de alto nivel. Ambos elementos se comunican por medio del protocolo OpenFlow. Esta metodología, conocida como SDN permite una mayor efectividad en el uso de los recursos de la red que en una red convencional.

El funcionamiento básico es el siguiente: un controlador OpenFlow, responsable de la toma de decisiones de encaminamiento, se conecta a un switch o router OpenFlow con varios terminales conectados en sus diferentes puertos. Este controlador se comunica constantemente con el switch o router a través del envío de mensajes OpenFlow, de forma que puede configurar las conexiones entre los hosts conectados al switch o router en función de datos que le lleguen de allí. Por ejemplo, imaginemos dos hosts diferentes A y B conectados a los puertos 1 y 2 de un switch respectivamente, y que el switch no tiene ningún tipo de configuración establecida. Si el host A quiere enviar algo al host B, en principio no podría debido a esa falta de configuración. Sin embargo, gracias a OpenFlow, el switch puede detectar que tiene un paquete en su puerto 1 que debe ir a su puerto 2, enviar una notificación al controlador en forma de mensaje OpenFlow y recibir como respuesta una o varias tablas OpenFlow con la configuración necesaria para que los hosts A y B se puedan comunicar sin ningún problema.

Por otro lado, se ha hablado de la configuración de tablas OpenFlow. Una tabla OpenFlow es una entidad que contiene diferentes *flows* o flujos, que sirven para que el switch haga diferentes operaciones (*instructions*) si se cumplen unas determinadas condiciones (*match*). En el ejemplo anterior una de las tablas OpenFlow que envía el controlador al switch podría ser una con un flujo que diga que, si el puerto de entrada al switch del paquete es el 1, se envíe dicho paquete por el puerto 2. Además, pueden anidarse tablas y flujos con diferentes órdenes de prioridad, de forma que el switch pueda tener en cuenta muchos parámetros diferentes al procesar los mensajes que le llegan [4].

## 2.4 Controladores OpenFlow: OpenDayLight

El controlador OpenFlow es el “cerebro” que dicta al switch o router qué hacer con los datos que entran y salen, utilizando diferentes tipos de mensajes OpenFlow. Un solo controlador puede gestionar varios routers o switches en tiempo real, lo que permite poder cambiar radicalmente las conexiones de una red en función, por ejemplo, de parámetros como el tráfico que pasa por ciertos puntos [4].

Actualmente hay varios tipos de controladores que se diferencian en las versiones del estándar OpenFlow que soportan y en el lenguaje de programación en el que están escritas las diferentes aplicaciones disponibles en cada uno de ellos. Para este trabajo se utilizó el controlador OpenDayLight [2], por ser este controlador el implementado en trabajos anteriores del que es continuación la investigación desarrollada en éste.

El controlador OpenDayLight es un controlador versátil programado en Java y que soporta las versiones de OpenFlow 1.0 y 1.3. Este controlador no está pensado exclusivamente para el uso de OpenFlow, ya que OpenFlow es solo uno de los varios protocolos y estándares que forman parte de las redes definidas por software (SDN, *Software Defined Networks*), redes que pueden variar su funcionalidad mediante el uso de diferentes programas [2]. Sin embargo, en este trabajo nos limitaremos a utilizar su vertiente OpenFlow. Una captura del interfaz de dicho controlador es la mostrada en la Figura 3.

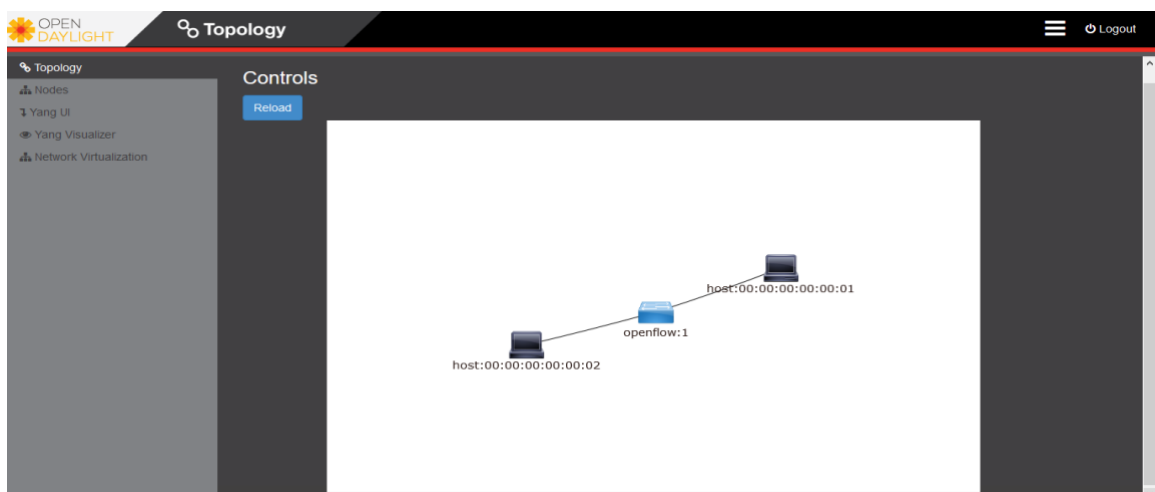


Figura 3: Interfaz gráfica del controlador OpenDayLight

## 2.5 Open Virtual Switch (Open vSwitch – OVS)

Open vSwitch, abreviado OVS, es un software de código abierto, diseñado para ser utilizado como un switch virtual en entornos virtualizados. Está diseñado para habilitar una automatización masiva de red, mientras que soporta interfaces de gestión y protocolos estándares (NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). Además, está diseñado para soportar una distribución para múltiples servidores físicos similar al switch de VMWare's vNetwork o el Cisco's Nexus 1000V [5].

En relación a nuestro proyecto, Open vSwitch es una de las implementaciones más populares de OpenFlow y soporta este protocolo por defecto. Esto nos permite implementar una capa OpenFlow en diferentes dispositivos de manera transparente al usuario y al resto de la red. Conceptualmente, las funciones de un switch pueden ser divididas en dos planos (el plano de control y el plano de datos). El plano de control es la inteligencia central del switch, la cual se encarga del descubrimiento, enrutamiento, computación de caminos y de comunicación con otros switches. En este sentido, OpenFlow nos permite descargar el plano de control de todos los switches a un controlador central que define el comportamiento de la red (lo que se conoce hoy en día como SDN) [6].

De esta manera, Open vSwitch se convierte en una opción perfecta para nuestro proyecto, ya que conjuga un soporte por defecto a las distintas versiones de OpenFlow, como se puede ver en la Figura 4. Además, está preparado para realizar SDN y es ligero y transparente para el usuario y otras redes.

Como punto negativo, debemos destacar que todavía se encuentra en fase de desarrollo, habiéndonos encontrado con hasta tres versiones distintas en el periodo comprendido entre diciembre de 2017 y abril de 2018, durante el cual que se desarrolló parte de esta investigación. Algunas funciones del switch, sobre todo las relacionadas con la implementación de funciones más avanzadas de OpenFlow, se encuentran todavía en fase experimental, y no se comportan de una manera predecible ni tienen un rendimiento adecuado al cien por cien.

Open vSwitch	OF1.0	OF1.1	OF1.2	OF1.3	OF1.4	OF1.5	OF1.6
1.9 and earlier	si	—	—	—	—	—	—
1.10, 1.11	si	—	Por defecto	Por defecto	—	—	—
2.0, 2.1	si	Por defecto	Por defecto	Por defecto	—	—	—
2.2	si	Por defecto	Por defecto	Por defecto	Experimental	Por defecto	—
2.3, 2.4	si	si	si	si	Por defecto	Por defecto	—
2.5, 2.6, 2.7	si	si	si	si	Por defecto	Por defecto	Por defecto
2.8	si	si	si	si	si	Por defecto	Por defecto

Figura 4: Tabla de versiones soportadas de OpenFlow para cada versión de OVS

## 2.6 Metodología de trabajo

En esta sección del capítulo se hará una descripción de cuál ha sido la metodología y pasos seguidos para la consecución final de los objetivos propuestos. Recordemos que el objetivo del trabajo es la integración del estándar OpenFlow en nuestra maqueta GPON para implementar un escenario SDN extremo a extremo y así separar el plano de datos del plano de control.

### 2.6.1 Programación de Linux Kernel Routing y un servidor DHCP en el ordenador central

El primer paso para conseguir el escenario deseado es programar un router en un ordenador con Linux Mint que situaremos en nuestra Oficina Central (CO, *Central Office*) virtual. Para ello, vamos a utilizar la utilidad del kernel de Linux de reenvío de paquetes IP [7] entre varias redes (es decir, que nuestro ordenador actúe como un router). Necesitamos realizar esta tarea para controlar la red que llega a la maqueta GPON, y no depender del router de la Escuela para las distintas configuraciones necesarias. Posteriormente, se procederá a instalar un servidor DHCP (*Dynamic Host Configuration Protocol*) para dar los parámetros de configuración de red a las máquinas que se conecten al ordenador central.

### **2.6.2 Instalación del OVS en el ordenador central conectado al OLT**

El segundo paso consistió en la instalación del OVS en el ordenador central (en el que se implementó el router y el servidor DHCP), para simular la implementación de la capa SDN del lado del OLT. Para ello, se estudió la instalación del OVS en una máquina virtual separada apoyándonos de las utilidades de Virtualbox, pero debido a la necesidad de trabajar en un modo concreto del OVS (que posteriormente explicaremos en el Capítulo 3) conseguíamos unas tasas de transmisión muy bajas. Por esta razón se decidió instalar el switch virtual en el sistema operativo principal de la máquina del ordenador central. Para ello, seguimos la guía proporcionada por su página oficial [8] y después completamos la instalación.

### **2.6.3 Montaje de las Raspberry Pi e instalación del OVS del lado de las ONTs**

El tercer paso consistió en implementar otro OVS en una Raspberry Pi (versión 3B) [9] para simular la capa SDN en el lado de la ONT. Hemos elegido este dispositivo para implementar nuestro switch virtual debido a su versatilidad y la portabilidad (Figura 5). En caso contrario, instalar el switch virtual en un ordenador implicaría tener un dispositivo de mayor tamaño con un consumo energético bastante más elevado que un microprocesador basado en ARM (*Advanced RISC Machine*, una de las arquitecturas RISC más conocidas) además de que en las pruebas realizadas (de las que hablaremos con más profundidad en el Capítulo 4) se vió que el rendimiento era más elevado en un OVS instalado en un sistema operativo basado en ARM que en cualquier sistema operativo Linux clásico basado en arquitecturas convencionales (Intel-AMD). Además, como el sistema operativo de la Raspberry Pi es Raspbian (una versión modificada de Debian), nos permite tener Linux como sistema operativo e instalar Open vSwitch (sólo disponible para distribuciones Linux) [10]. Para realizar dicha instalación, seguiremos la guía proporcionada por la página web oficial de Open vSwitch e implementaremos la aplicación a nuestro gusto [8].



Figura 5: Raspberry Pi 3 Model B

#### **2.6.4 Conexión de los OVS's con el controlador OpenDayLight**

El cuarto paso fue configurar dentro de los diferentes switches OVS en qué dirección se encuentra el controlador OpenDayLight. Después de especificar ésto en todos los switches, tenemos que configurar las tablas de encaminamiento en las Raspberry Pi y en el ordenador central, para lograr que haya comunicación entre los switches y el controlador, aunque estén en subredes diferentes. A nivel telemático, una subred en capa 3 (capa de red) se define como todos aquellos dispositivos que se encuentran entre un router y otro router. Es decir, los routers dividen el espacio de red en diferentes redes o subredes. En la Figura 6 se puede observar como queda nuestra red experimental real dividida en diferentes subredes. En esta topología tenemos 7 subredes diferentes, aunque se pueden categorizar como 3, ya que las subredes entre la ONT y el cliente final son iguales y las subredes 2 y 3 entre el router de la oficina central y las ONTs engloban los mismos dispositivos, pero cada uno en una VLAN. Por lo tanto, tenemos la subred 10.0.103.0/22 como subred troncal de la Escuela; la subred 2 y 3 (192.168.0.0/24 y 10.19.59.0/24) como subred entre el router de la oficina central y cada una de las ONTs (aunque si añadimos más VLANs tendríamos una subred por cada VLAN diferente); y las subredes de cada ONT que se define entre la propia ONT y el cliente final (192.168.x.0/24). De este modo, si dividimos la red correctamente manteniendo la conectividad entre los dispositivos, mejoraremos la seguridad y el rendimiento de nuestra

red. En la red GPON, el OLT es transparente en capa de red, esto es, no es un router como tal, aunque sí tiene ciertas funcionalidades de capa de red. Por tanto, este dispositivo no divide subredes entre sus diferentes interfaces.

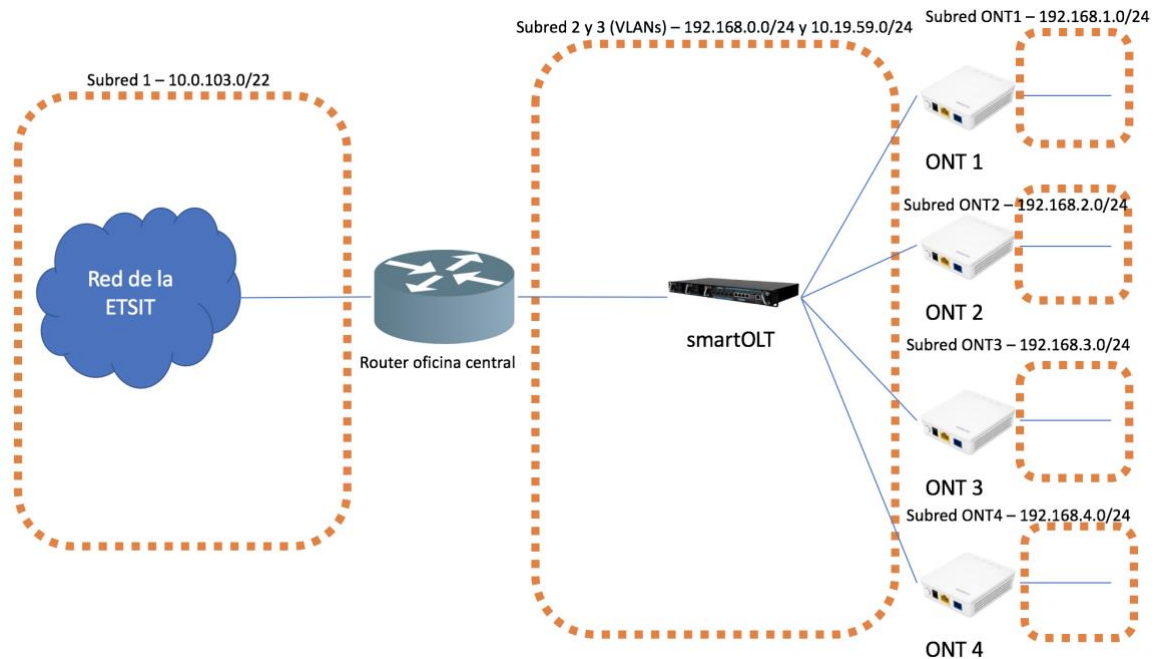


Figura 6: Estructura de las diferentes subredes presentes en nuestro escenario de red real

### 2.6.5 Programación y despliegue de las tablas de flujo en el controlador OpenDayLight

El siguiente paso es programar las tablas de flujo, así como los flujos en el controlador OpenDayLight en vez de hacerlo manualmente en cada OVS. Para ello, nos ayudaremos de la utilidad YangUI disponible para la instalación de OpenDayLight (Figura 7). Esta utilidad permite configurar los flujos con una interfaz gráfica más amigable para el administrador, en vez de realizar peticiones HTTP GET y PUT a través de RESTCONF. Si tuviéramos que usar esta segunda utilidad a través de RESTCONF, se tendría que rellenar una petición HTTP entera, por ejemplo: `https://token:$token@<controller-IP-address>/controller/restconf/config/opendaylight-inventory:nodes/node/openflow:<Openflow-device-ID>/table/<table-ID>/flow/<Flow-ID>`.

OpenFlow 1.3 tiene varias tablas en las que introducir nuestros flujos. Por defecto, se usa la tabla 0 cada vez que añadimos nuevos flujos, aunque OpenFlow soporta hasta 255 tablas. Aunque nosotros utilizamos sólo una tabla en la que introducimos todos los flujos, debido a que nuestros flujos sólo tienen dos instrucciones diferentes, algunos



administradores de red prefieren añadir más tablas y dedicar cada tabla a realizar una acción determinada y separar cada flujo en diferentes tablas de manera lógica y según su comportamiento [11]. En la Figura 7 se puede ver que hay una *table list* para el nodo de OpenFlow que representa nuestro switch virtual en el que hay configuradas varias tablas.

Cada flujo tendrá configurados varios campos (*match, action, instruction...*) que posteriormente definiremos en el Capítulo 4. Puesto que el objetivo de esta investigación es configurar los servicios de la red GPON mediante OpenFlow, intentaremos definir dichos servicios a través de estos flujos OpenFlow. Por lo tanto, una vez configurados los diferentes flujos y las tablas, se probará que los servicios y el tráfico de la red GPON se gestionan de forma eficiente y automática mediante estos flujos configurados. Para ello se lanzarán y se observarán las pruebas pertinentes en la red real GPON, cuyos resultados se mostrarán en el Capítulo 4.

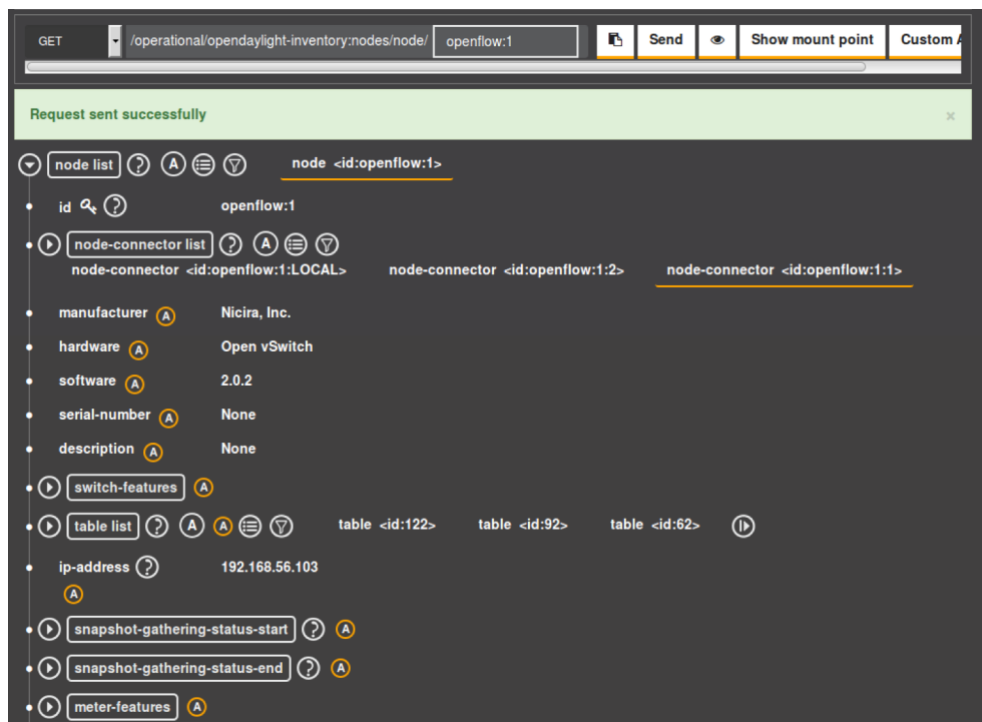


Figura 7: Captura de la interfaz gráfica de YangUI

### **2.6.6 Diseño de una interfaz web de usuario para gestionar de forma dinámica requisitos de servicio**

El último paso de este trabajo de investigación es la programación de una interfaz web (en HTML5/Javascript/PHP) para el usuario, tal y como se muestra en la Figura 8.

Esta interfaz se ha desarrollado para proponer un nuevo modelo de negocio para las operadoras en el que los usuarios pagan por un uso dinámico de la red de manera que puedan pedir un incremento del ancho de banda contratado durante periodos concretos para utilizar aplicaciones que requieran altos requisitos, por ejemplo, televisión en 4K o juegos online.

Así pues, a través de esta interfaz web (alojada en la Raspberry Pi de cada ONT) el usuario puede seleccionar el ancho de banda deseado (en Mbps) y el periodo de tiempo durante el cual va a solicitar este ancho de banda. Posteriormente, transmitirá de forma transparente estos datos por un socket TCP a un programa situado en la Oficina Central del operador, que interpreta los datos y procesa dicha petición.

En nuestro caso en concreto, esta petición realizada por el usuario es enviada a un programa de dicha Oficina Central que gestiona de forma inteligente y dinámica los servicios de la red GPON. Posteriormente, desde dicho programa se envían las órdenes pertinentes (aceptación o no de la petición del usuario) al controlador OpenDayLight, que creará y enviará los flujos OpenFlow, para realizar una gestión más optimizada de los recursos y de los servicios ofrecidos en tiempo real a los usuarios adscritos.



Figura 8: Captura de la interfaz gráfica programada en HTML5/Javascript/PHP

## 2.7 Conclusiones

En este capítulo se han descrito las herramientas que se van a utilizar en este Trabajo Fin de Grado, así como la metodología de trabajo para lograr el objetivo final, es decir, implementar una capa SDN haciendo uso del protocolo OpenFlow en una maqueta real GPON. En concreto, se han explicado los pasos para llevar a cabo dicha implementación. Para ello, hemos implementado un router con DHCP, hemos instalado un switch virtual en la Oficina Central, hemos montado una Raspberry Pi en cada ONT en la que se instaló un switch virtual y, finalmente, se conectó cada switch con el controlador central OpenDayLight. Con esta implementación se podrán programar flujos en el controlador OpenDayLight conectado a la subred de la Escuela para así gestionar el tráfico de los diferentes servicios asignados en cada ONT (usuario). Finalmente, se propone un nuevo modelo de negocio para las operadoras en las que los usuarios de forma temporal puedan demandar nuevos requisitos en los servicios contratados, por lo que se programará una interfaz web que permite realizar dichas peticiones en tiempo real.

# 3

## Definición del escenario SDN en la red GPON

### 3.1 Introducción

En este capítulo se va a realizar un análisis sobre el estado del arte relacionado con tecnologías SDN en redes PON, para posteriormente describir la implementación de nuestro escenario SDN concreto. Después, se describirán secuencialmente los pasos a seguir para implantar nuestra solución particular SDN-GPON, desde la implementación de un router y servidor DHCP en nuestra Oficina Central, a la configuración de los switches virtuales (OVS) del lado del OLT y del ONT. Finalizaremos con la conexión y puesta en marcha de dichos OVS con el controlador OpenDayLight, necesaria para la configuración y gestión de los flujos OpenFlow. De este modo, se implementará una gestión y configuración de los servicios y del tráfico de datos en la red GPON mediante el protocolo OpenFlow.

### 3.2 Estado del arte SDN en redes PON

Las redes definidas por software (SDN, *Software Defined Networking*) permiten la gestión eficiente de una red centralizada gracias a la abstracción de los planos de control y de datos, permitiendo un balance de carga eficiente y una mejor distribución del tráfico. De esta manera, la tecnología SDN consigue una importante reducción de costes de operación y *hardware* gracias a esta centralización y a una mayor vida útil de los componentes. Además, SDN mejora la seguridad gracias a un controlador centralizado que distribuye la información de manera consistente y segura por toda la red SDN [12]. Finalmente, permite una infraestructura lista para trabajar en la nube, con un nivel de automatización muy alto y un paradigma de red altamente flexible y escalable. Esta

infraestructura novedosa permite desarrollar y desplegar nuevas aplicaciones y servicios en las redes. Por todo ello, la integración de SDN en redes ópticas pasivas PON (*Passive Optical Networks*) está ganando mucha importancia recientemente y muestra una gran proyección en el sector. En este sentido, varias investigaciones integran los paradigmas de encaminamiento de OpenFlow en el plano de control para lidiar con diferentes problemas. Así, Lee et al [13] proponen sustituir el OLT de una red GPON por un virtual switch SDN basado en OpenFlow. Pasos previos a este diseño de switches virtuales fueron propuestos por Gu et al [14] en centros de tratamiento de datos. En esta misma línea, los autores en el artículo [15] desarrollaron módulos para convertir mensajes OpenFlow a comandos nativos de configuración de redes PON. Los autores de [16] [17] proponen un controlador centralizado basado en OpenFlow para gestionar varios SDN-OLTs. Por otro lado, en el trabajo presentado en [18] se define una extensión del protocolo OpenFlow para redes GPON, denominada OpenFlowPLUs, que interactuará sobre la interfaz de control y la capa óptica (OMCI, *Optical Management and Control Interface*). Sin embargo, paradigmas relacionados con la asignación de ancho de banda dinámica (DBA, *Dynamic Bandwidth Allocation*) o el registro de las ONTs en la red no están incluidos en dicha especificación OpenFlow, debido a la latencia existente entre el controlador SDN y el OLT. Aún así, existen trabajos de investigación en los que se propone mover la elección de ciertas políticas DBA más estáticas y globales al controlador centralizado SDN.

Consecuentemente, en este trabajo se propone el diseño de un escenario SDN implementado en un maqueta real GPON en la que un controlador centralizado (OpenDayLight) se encargará de gestionar la calidad de servicio (QoS, *Quality of Service*) de la red GPON mediante el protocolo OpenFlow. De este modo, el controlador central OpenDayLight gestionará de forma dinámica el tráfico de los diferentes servicios asociados a cada usuario final (conectados a las ONTs) en los dos canales de la red GPON, esto es, *downstream* y *upstream*.

### **3.3 Descripción del escenario SDN en la red GPON**

Es necesario para el resto de este Trabajo Fin de Grado comprender bien lo que nosotros definimos como escenario SDN en la red GPON. Nuestro escenario inicial es un escenario clásico GPON, en el que hay un OLT que da servicio a varias ONTs (en nuestro caso cuatro, pero escalable a N nodos terminales) a través de la configuración de servicios

y perfiles de abonado usando el interfaz TGMS o la línea de comandos CLI, tal y como se explicó en el capítulo anterior.

De manera global, el OLT de nuestra red (smartOLT), tiene varias interfaces, cada una correspondiente a una VLAN diferente, que van conectadas a un router en la oficina central. Una VLAN (Virtual LAN) es un grupo lógico de dispositivos de red que no están limitados a estar físicamente conectados a un único segmento físico. Los dispositivos o usuarios de las VLANs se pueden agrupar por función, departamento o aplicación independientemente de a qué segmento físico estén conectados. Por tanto, una VLAN define un dominio de *broadcast* y es una subred lógica en sí misma. En la Figura 9 se puede ver la diferencia entre una LAN convencional y un conjunto de VLANs.

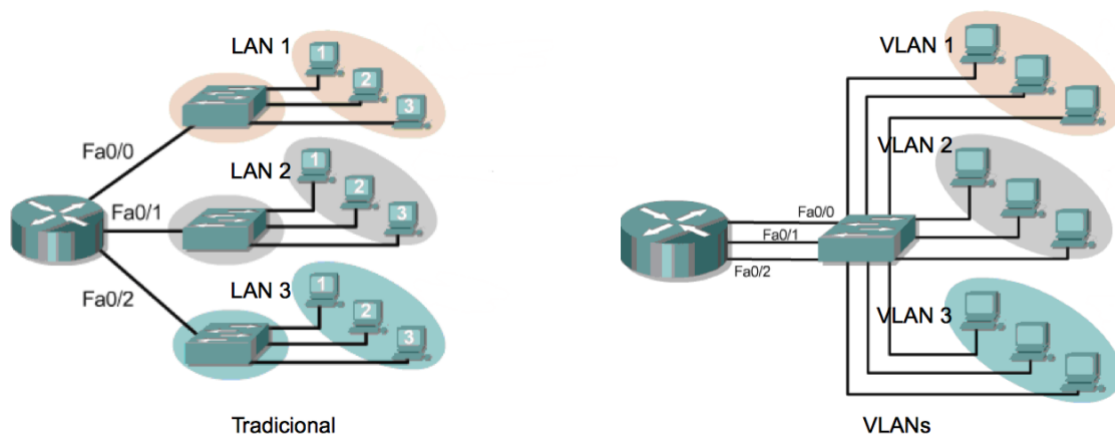


Figura 9: Escenario LAN tradicional vs Escenario VLAN

El router de la oficina central anteriormente citado va conectado a la red troncal del ISP (*Internet Service Provider*) correspondiente, proveyendo de los diferentes servicios (actualmente Internet, vídeo y voz) a nuestra red de acceso GPON. Normalmente, un ISP separa cada uno de los servicios en una VLAN diferente, teniendo una VLAN para el servicio de Internet, una VLAN para el servicio de vídeo y una VLAN para el servicio de voz si está implementado por VoIP (la mayoría de ISPs ya han dejado de implementar servicio de voz analógica). Para emular este caso en nuestro escenario real, el router lo hemos instalado en el ordenador central que conecta el smartOLT a la red de la Escuela, con diferentes VLANs que simulan los distintos servicios que tendría un operador tal y como se observa en la Figura 10.

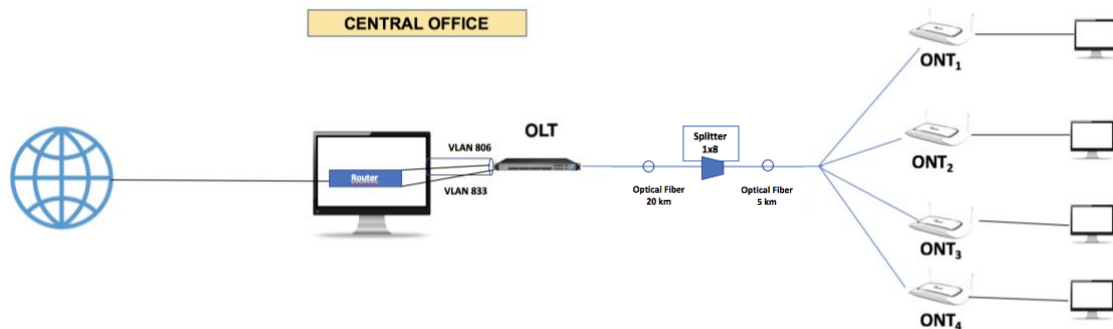


Figura 10: Escenario GPON clásico

A este escenario clásico GPON queremos añadirle una funcionalidad SDN. Para ello, se nos presentan una serie de retos a nivel telemático, que pasaremos a describir brevemente.

En primer lugar, necesitamos ser capaces de controlar y poder configurar como queramos nuestro router de la Oficina Central. En nuestro laboratorio de la ETSIT (Escuela Técnica Superior de Ingenieros de Telecomunicación), el router que da servicio a nuestra red GPON da también servicio a varios laboratorios y no tenemos acceso a él para configurar las redes virtuales, el DHCP y todas las interfaces que necesitamos. Además, el smartOLT (OLT) de nuestra red GPON admite el añadido de prioridades a las tramas 802.1Q (VLAN) [19], pero si se añade una diferente de la 0, algún router o switch de dentro de la Escuela descarta los paquetes (ya que marca todos los paquetes con prioridad 0). En este sentido, fue imposible diagnosticar qué router o switch y por qué, debido a que el administrador de la red de la Escuela no pudo realizar estas labores de diagnóstico por la complejidad de la red y la imposibilidad de parar la red de la Escuela el suficiente tiempo. Posteriormente, se detallará en el Apartado 3.4 qué pasos se deben realizar para intentar evitar este problema y por qué finalmente no se implementó.

En segundo lugar, necesitamos implementar varios switches virtuales que tengan activado OpenFlow para poder gestionar nuestra red GPON mediante SDN, en concreto uno en el OLT (*Central OVS*, *COVS*) y uno por cada ONT (*ROVS*, *Remote OVS*). En estos Open vSwitch se configurarán diferentes flujos que abstraerán los planos de control y de datos de la parte óptica de la red y se gestionará la red como un todo desde un controlador centralizado, en nuestro caso OpenDayLight. Para ello, necesitaremos que estos switches

que acabamos de configurar se conecten con el controlador OpenDayLight y reconozcan a este controlador como maestro, de manera que dicho controlador pueda editar su configuración y sus flujos en tiempo real. Esta configuración del controlador como maestro se detallará posteriormente en el Apartado 3.5.

Una vez configurado todo, nuestra red, que tendrá funcionalidades SDN y denominaremos SDN-GPON, ya será plenamente configurable por nosotros en tiempo real desde nuestra Oficina Central a través del controlador OpenDayLight. Por tanto, nuestro escenario final quedará como el que refleja la Figura 11.

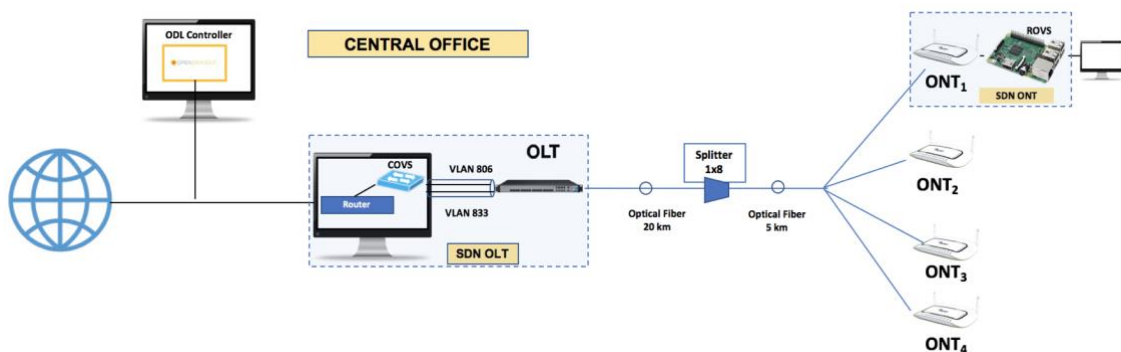


Figura 11: Escenario SDN-GPON implementado sobre la maqueta GPON de 25 kilómetros.

### 3.4 Implementación del router y el servidor DHCP

Puesto que el ordenador central del laboratorio tiene instalado un Linux Mint, nuestro router va a estar basado en el reenvío de paquetes IP implementado ya en los sistemas Linux y UNIX (*Linux Kernel Routing*). Esto se puede hacer de una manera muy sencilla escribiendo un '1' en el fichero `ip_forward` que se puede encontrar en la ruta `/proc/sys/net/ipv4/ip_forward`. Esto hace que el ordenador central se comporte como un router, es decir, estrictamente, será capaz de reenviar paquetes de una subred (la subred a la que pertenece la red troncal de la Escuela) y la subred que va a dar a la OLT (así como a las subredes implementadas detrás de cada ONT), como ya se ha comentado en el Apartado 2.6.4 y se observa en la Figura 6. Para que nuestro router actúe como nosotros queremos, tenemos que configurar las tablas y las interfaces. Nosotros vamos a usar la utilidad *iptables* [20] que, aunque bien es cierto que se suele usar más como una utilidad para establecer reglas de cortafuegos, también nos sirve para establecer reglas estáticas de cómo queremos que nuestro kernel reenvíe los paquetes. Estas reglas se



especifican a través de la línea de comandos, utilizando los parámetros necesarios que veremos posteriormente en este apartado, y normalmente se limitan a especificar de qué subred a qué subred queremos llevar los paquetes, y si los queremos tratar de manera o de otra. Específicamente, nosotros vamos a aplicar dos reglas<sup>1</sup>.

La primera regla será `iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o VLAN833 -j MASQUERADE`. La primera opción de este comando es `-t`, que especifica a que tabla debe referirse este comando. En este caso, se refiere a la tabla de NAT (*Network Address Translation*). Esta tabla se consulta cuando un paquete que llega crea una nueva conexión, y tiene dos posibilidades, `PREROUTING` (para actuar sobre los paquetes en cuanto llegan a nuestro kernel) y `OUTPUT` (para actuar sobre los paquetes que se generan localmente antes de que se encaminen). Desde la versión del kernel de Linux 2.4.18, se soporta también la opción `POSTROUTING`, que actúa sobre los paquetes cuando ya van a abandonar nuestro kernel. Con la opción `-A`, estamos especificando qué regla queremos añadir en la tabla previamente especificada. A continuación, la regla `-s` especifica el origen de los paquetes a los que queremos aplicar la regla. En nuestro caso, añadiremos la subred 192.168.0.0/24, ya que esa va a ser la subred en la que nuestra máquina será la encargada de enrutar. La opción `-o` es la interfaz por la cual queremos sacar los paquetes que cumplan las reglas anteriores. La interfaz que está conectada al router de la Escuela es una interfaz VLAN con dirección IP 10.0.103.48/24 previamente configurada en la máquina Linux Mint con nombre VLAN833, por eso hemos puesto directamente ese nombre en el parámetro. La regla `-j` indica qué hacer con los paquetes que coinciden con todo lo especificado anteriormente. Nuestra opción `MASQUERADE` es una opción sólo posible en la tabla NAT para mapear la dirección de una interfaz de dentro y una interfaz exterior que es la que da realmente acceso a Internet. Los paquetes se enmascaran y se les mapea con la dirección IP por la interfaz de red por la que salen. Esta dirección puede ser variable (caso típico en el que la interfaz de salida pertenece a otro proveedor de servicios de Internet y nos tiene asignados una IP

---

<sup>1</sup> Para la explicación de las reglas de *iptables* vamos a citar la información recogida en el manual de Linux referente a la utilidad *iptables*. Para la comodidad del lector, proponemos [38] para la consulta completa de todas sus posibilidades.

dinámica que cambia diaria o semanalmente) y es una de las razones por la que usamos el parámetro MASQUERADE. Si nuestra dirección IP de la interfaz que nos da acceso a Internet fuera fija, podríamos usar el parámetro SNAT, siendo el comando de *iptables* bastante análogo al que hemos mencionado anteriormente. Este comando sería el siguiente:

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 10.0.103.48/24.
```

Como las VLANs dividen una interfaz en diferentes subredes, tenemos que especificar varias reglas similares para cada VLAN diferente, ya que cada VLAN será a todos los efectos una red diferente. La segunda regla que se usará, `iptables -t nat -A POSTROUTING -s 10.19.59.0/24 -o VLAN833 -j MASQUERADE`, es una regla exactamente igual que la anterior, pero con otra subred diferente a la anterior y que posteriormente usaremos con otra VLAN.

Llegados a este punto, debemos explicar una serie de conceptos que, aunque finalmente no se han implementado en nuestra maqueta GPON, se estudiaron. El OLT de Telnet (smartOLT) permite especificar qué prioridad se quiere incluir en la trama VLAN. Sin embargo, los routers/switches más allá de la subred del laboratorio cambiaban automáticamente la prioridad a “0”, probablemente debido a un asunto de compatibilidad con otros routers/switches. Cuando esta trama llegaba a la red GPON con una prioridad diferente a la especificada en las reglas de la smartOLT (definidas mediante TGMS o CLI) dicha trama se descartaba automáticamente, en el caso de que la trama perteneciera a un servicio con una prioridad distinta a “0”. Para poder seguir utilizando esta funcionalidad y dar un tratamiento distinto a diferentes servicios, aunque vinieran con la misma etiqueta VLAN, se planteó una posible solución para poder seguir especificando una prioridad distinta para la misma VLAN. Nuestra solución fue crear una regla con la utilidad *iptables* para volver a especificar una regla para los paquetes que fueran a una subred concreta.

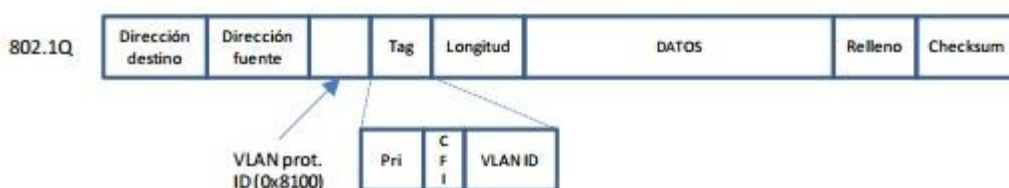


Figura 12: Trama 802.1Q con campos Prioridad, CFI y VLAN ID

Así pues, esta nueva regla que se especificó fue `iptables -t mangle -A POSTROUTING -d 192.168.0.0/24 -j CLASSIFY --set-class 0:5`. Vemos respecto a las anteriores reglas que ésta utiliza una tabla diferente, la tabla `mangle`. Esta tabla se utiliza para una manipulación especializada de los paquetes, y tiene las mismas cadenas para añadir reglas que la tabla NAT desde el kernel versión 2.4.18. En esta regla vamos a utilizar la opción `-d`, que especifica un destino de los paquetes (lo contrario que el campo `-s` que usamos en la anterior regla). Este campo se puede ir modificando dependiendo de nuestra topología de red. En la opción `-j`, que especificaba qué queríamos hacer con los paquetes coincidentes con la regla establecida, observamos que se ha elegido el parámetro `CLASSIFY`. Este parámetro cambia la prioridad `skb->priority` de la prioridad que ya tenía a la que queremos que tenga. Esta prioridad es diferente que la prioridad de los paquetes VLAN (podemos ver donde se sitúa este campo de prioridad en la Figura 12) y de la utilidad `vconfig` proporcionada en Linux, y directamente cambia la prioridad en el búfer de sockets en Linux. Es decir, esta prioridad es un parámetro interno del sistema operativo para clasificar a los diferentes sockets que tiene abiertos el sistema operativo con una prioridad diferente en caso de que se tenga que priorizar alguna comunicación por alguna razón. Esta reconfiguración se debería poder hacer desde la utilidad `vconfig` (con las opciones `set_ingress_map` y `set_egress_map`), y cada vez que un paquete ingrese, el kernel debe copiar esa prioridad a `skb_priority`. Desgraciadamente, el mismo problema que tienen los switches y routers de la Escuela lo tiene el kernel de Linux de nuestra máquina, es decir, que no conservan la prioridad que tiene la trama VLAN y las modifica a prioridad “0”. Por lo tanto, nuestra solución pasa por calificar los paquetes según las características ya sabidas por nosotros mediante la utilidad `iptables` y poner una prioridad diferente en el campo *prioridad* (Figura 12) de la trama 802.1Q (del 1 al 7) que vaya con destino a la subred deseada. Posteriormente, utilizar el comando `vconfig set_egress_map enp4s1.833 1 1, vconfig set_egress_map enp4s1.833 2 2... [21]`, repitiendo este comando de la prioridad 1 a la 7. Con esto, estamos mapeando diferentes prioridades del kernel con prioridades VLAN, de modo que las tramas con prioridad de kernel “1” que salgan por la interfaz `enp4s1.833` se les asigna la misma prioridad “1”. En nuestro caso, hemos optado por asignar prioridades con el mismo número para que el resultado fuera menos engorroso, pero podrían asignarse desigualmente, por ejemplo, `vconfig set_egress_map enp4s1.833 1 6`. En este segundo caso, el

significado del comando indica que las comunicaciones con prioridad de kernel “1” que fueran a salir por la interfaz *enp4s1.833* se les asignaría una nueva prioridad “6”. Todo este proceso está ilustrado por orden en la Figura 13.

Así pues, el comando nos pide 3 parámetros, que son la interfaz a la que nos referimos, la prioridad del kernel asociada a la comunicación concreta (*skb->priority*) y la prioridad de VLAN deseada. Cuando los paquetes vayan a salir por esa interfaz, el sistema operativo comprobará la prioridad de *skb->priority* para después cambiar la prioridad de la trama VLAN. Análogamente, habría que ejecutar este comando para la otra interfaz (*enp4s1.806*) y hacer lo mismo para todas las prioridades, esto es, de la “1” a la “7”.

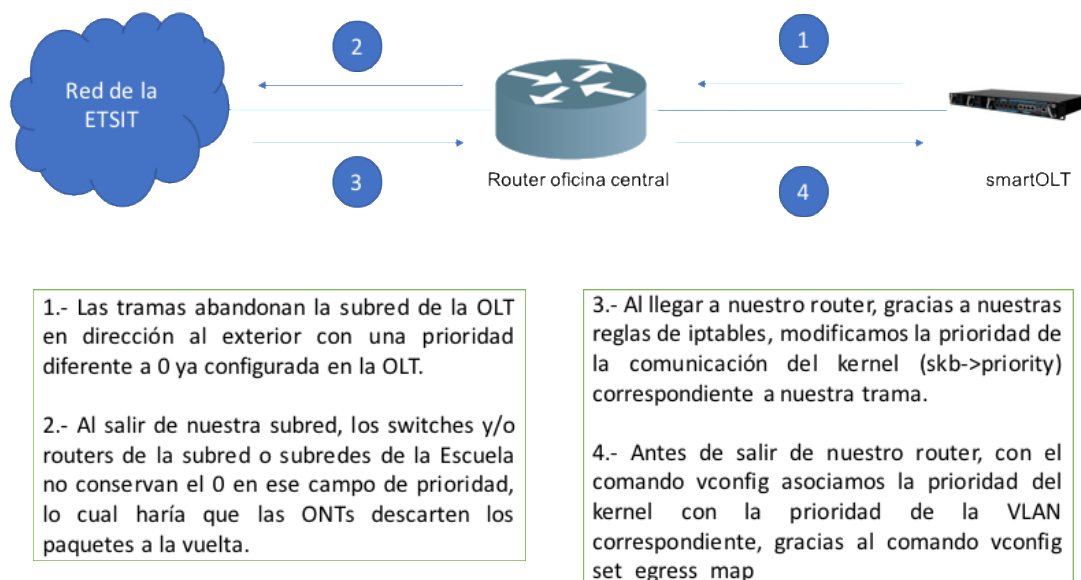


Figura 13: Pasos en la manipulación del campo prioridad de la trama VLAN 802.1Q

Como ya hemos advertido antes, esta solución no se terminó implementando, aunque el resultado esperado sí que fue satisfactorio. Esto fue debido a que el servidor DHCP incluido en Linux (*isc-dhcp-server*) usa *raw sockets*, aunque la mayoría de servidores DHCP de código abierto utilizan también este tipo de sockets. Estos sockets son diferentes a los sockets usados normalmente y no son interceptados por el kernel del sistema operativo al llegar a la máquina destino.

Esto implica que las utilidades de firewall (por ejemplo, *iptables*) no llegan a interceptarlos (o en nuestro caso, modificarlos), ya que no son procesados en la capa de

transporte del kernel del sistema operativo. Nuestra solución se basaba principalmente en interceptar los paquetes en el paso 3 de la Figura 13 y modificar la prioridad del kernel para posteriormente modificar la prioridad de la trama VLAN en el paso 4 de la Figura 13. Sin embargo, este proceso nos resulta imposible llevarlo a cabo al no poder interceptar los paquetes, puesto que el kernel no los detecta en su entrada a la máquina, ya que se salta la capa de transporte, como se observa en la Figura 14. Como no podemos modificar la prioridad a tiempo, el smartOLT recibe las contestaciones a las peticiones DHCP con una prioridad diferente a la esperada (concretamente, prioridad “0”).

Como las dos prioridades no coinciden, el smartOLT descarta estas tramas y no llega nunca a aceptar la dirección DHCP que le propone el router y no se le asigna una dirección IP a ninguna de las ONTs a través del smartOLT. Por tanto, nuestra solución no funciona a no ser que el smartOLT tenga direcciones estáticas (siendo este modelo bastante estático y poco escalable). Por eso se decidió abandonar esta solución y optar por trabajar siempre con prioridad “0” y distinguir servicios únicamente por VLAN.

Esta solución aparece también en el Anexo I, ya que está dentro del *script* que proporciona encaminamiento a nuestra subred. Sin embargo, los comandos relativos a lo anteriormente explicado aparecen comentados ya que ya no se requieren para el funcionamiento del *script*.

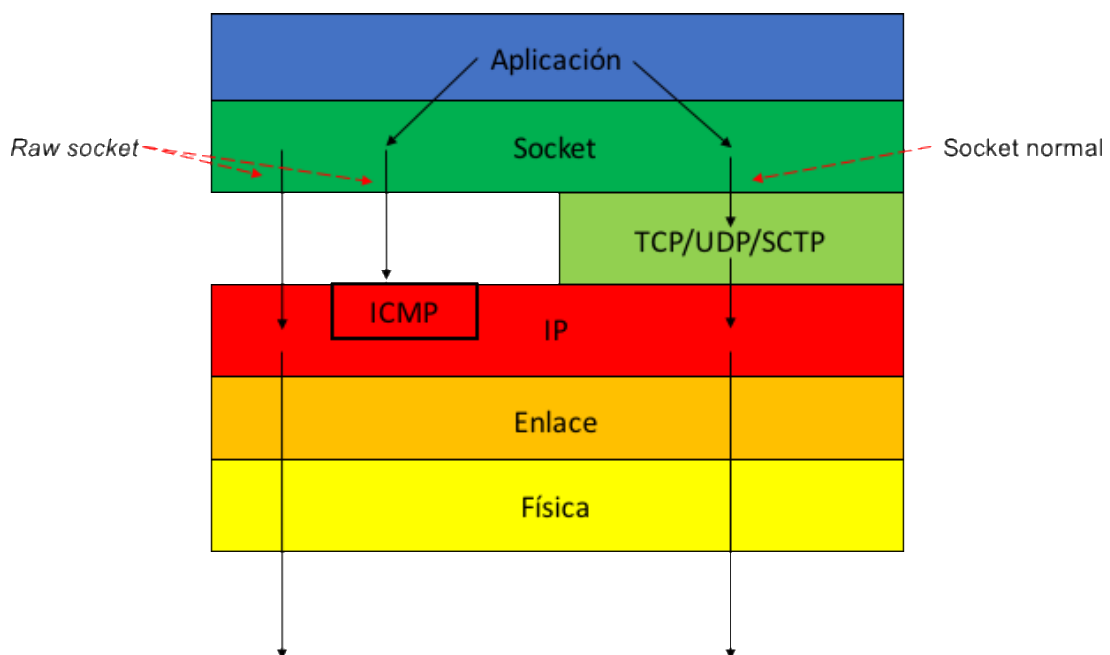


Figura 14: Sockets en la capa de protocolos TCP/IP

Una vez desechada esta solución, debemos seguir configurando nuestro router, una vez acabada la configuración de las *iptables*. En concreto, seguiremos con la configuración de las VLANs. Primero añadiremos las dos redes VLAN que hemos especificado anteriormente, una tendrá una etiqueta con número 833 y pertenecerá a la red 192.168.0.0/24 y otra tendrá una etiqueta con número 806 y pertenecerá a la red 10.19.59.0/24. Para ello, usamos los comandos `vconfig add enp4s1 833` y `vconfig add enp4s1 806`. La utilidad `vconfig` pertenece al kernel de Linux y permite añadir interfaces VLAN en interfaces Ethernet convencionales. Las LAN virtuales son las VLAN de la LAN física que sería la interfaz Ethernet [22]. En esta utilidad debemos escribir la interfaz real física de la que queremos crear las VLANs y el número de etiqueta. Posteriormente activamos la interfaz real física a través del comando `ifup enp4s1` y borramos cualquier información sobre direccionamiento que pueda tener la máquina de esta interfaz usando el comando `ip addr flush dev enp4s1`. Finalmente, se configurará la dirección de cada interfaz virtual (192.168.0.1/24 y 10.19.59.1/24) con los comandos `ifconfig enp4s1.833 192.168.0.1 netmask 255.255.255.0` y `ifconfig enp4s1.06 10.19.59.1 netmask 255.255.255.0`. La configuración de todo lo anteriormente explicado, esto es, *iptables* y las interfaces, se encuentra recogido en un script basado en Bash que se tiene que ejecutar cada vez que se encienda la red GPON y cuyo código completo se muestra en el Anexo I. Por lo tanto, nuestra topología final de red será la mostrada en la Figura 15.

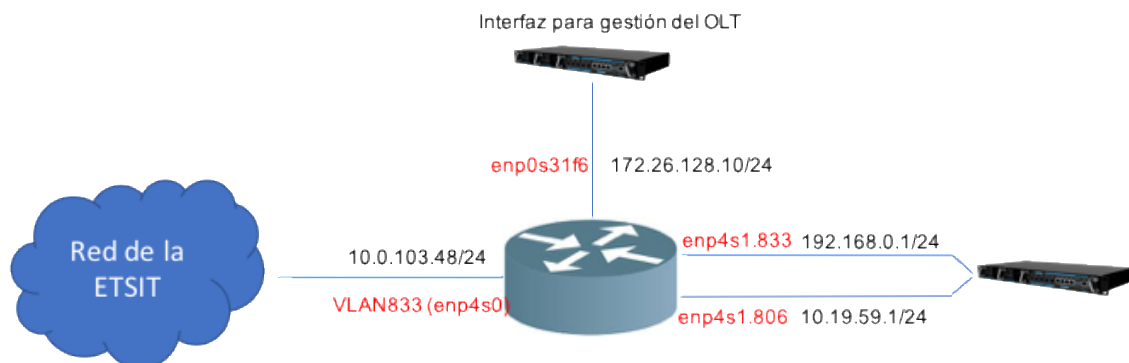


Figura 15: Topología de red centrada en el Linux Kernel router implementado en la máquina central

Después de programar todo lo relacionado con el encaminamiento y las diferentes subredes, tenemos que proporcionar el direccionamiento a las diferentes máquinas que se

conecten a nuestro router. Para ello, instalaremos un servidor DHCP en el ordenador central y lo configuraremos para repartir direcciones en el rango que nosotros queramos. Primero, elegiremos un servidor DHCP que sea de código libre y abierto. Nuestra elección ha sido el servidor del consorcio ISC (*Internet Systems Consortium*) [23], *isc-dhcp-server*. Para instalarlo en cualquier máquina Linux, ejecutamos el siguiente comando `apt-get install isc-dhcp-server`.

Teniendo ya instalado el servidor DHCP en nuestra máquina, ahora tendremos que configurarlo. Para configurarlo, debemos modificar dos ficheros de configuración en nuestra máquina. El primero se encuentra en `/etc/dhcp/dhcpd.conf`, aunque esto puede variar entre diferentes distribuciones de Linux. En este fichero encontramos diferentes subredes a configurar, pudiendo crear nosotros más. Se utiliza la palabra clave *subnet* seguida de la máscara de red *netmask*. En esta subred, nuestro servidor DHCP responderá a peticiones DHCP (*DHCP requests*) con las características concretas que nosotros configuremos.

Las características a configurar son las siguientes: rango (*range*), servidores de nombre de dominio (*DNS*), nombre del dominio (*domain-name*), encaminador de la subred (*router*), dirección de difusión (*broadcast*), tiempo de concesión por defecto (*default-lease-time*) y tiempo de concesión máximo (*max-lease-time*). Este fichero sería el único a configurar para cada subred si quisiéramos responder a peticiones DHCP en condiciones normales. En nuestra red, recordemos que las dos interfaces en las que queremos enrutar y dar direcciones IP a través de DHCP son interfaces virtuales, por tanto, tenemos que especificar que queremos dar direccionamiento a través de estas interfaces, ya que el servidor *isc-dhcp-server* sólo proporciona direccionamiento a través de las interfaces reales por defecto. Por tanto, tenemos que modificar un fichero que se encuentra en la ruta `/etc/default/isc-dhcp-server`, y modificar la línea *INTERFACES*, añadiendo las interfaces `enp4s1.833` y la interfaz `enp4s1.806`. Esta configuración del servidor DHCP se encuentra en el Anexo II.

## 3.5 Implementación del OLT basado en SDN con Open vSwitch (SDN – OLT)

Para implementar el Open vSwitch que controle el tráfico que va hacia la OLT (COVS, *Central OVS*), debemos instalarlo en el ordenador de la máquina central. El COVS se situará entre el ordenador y la OLT para así controlar el flujo *downstream* antes de que los datos lleguen a la OLT y simular así una capa SDN sobre el OLT. Para instalar el Open vSwitch, debemos hacernos con una copia del repositorio oficial usando la utilidad *git*. El comando `git clone https://github.com/openvswitch/ovs.git` [10]. Después de obtener la copia, podemos usar el comando `git checkout` para elegir la versión que queremos, ya que el repositorio contiene todo el histórico de versiones. En nuestro caso, por detalles que comentaremos posteriormente, no instalaremos la más reciente e instalaremos la versión 2.8.1. Para ello, ejecutamos el comando `git checkout v2.8.1`.

Una vez tenemos la utilidad copiada en nuestro ordenador, necesitamos instalar ciertas dependencias que exige la utilidad Open vSwitch para que pueda funcionar correctamente. Para instalarlas, ejecutamos el comando `apt-get install python-simplejson python-qt4 python-twisted-conch automake autconf gcc uml-utilities libtool build-essential pkg-config`. Después de instalar las dependencias necesarias, nos situamos en la raíz del directorio `ovs` (directorio donde se encuentran todos los ficheros copiados después de ejecutar el comando `git clone`) y ejecutamos el comando `./boot.sh`. Este comando construye el script de configuración que ejecutamos de manera inmediatamente posterior con el comando `./configure`. Este comando admite otras opciones para modificar la instalación por defecto que nosotros no usaremos, ya que la instalación por defecto cumple con todos nuestros requisitos. Después, ejecutamos el comando `make`, y posteriormente el comando `make install`. Estos dos comandos compilan e instalan todo lo necesario para que el Open vSwitch funcione correctamente.

Una vez instalado, debemos configurarlo para que funcione como nosotros queremos. Para ello, es importante entender cómo funciona un switch virtual. Un switch virtual actúa como un puente al que se le pueden conectar diferentes interfaces, ya sean reales o virtuales. La implementación más típica de un switch, en nuestro caso, un switch



virtual, es unir dos interfaces que previamente no estaban conectadas. Para ello, la herramienta OVS en cuestión crea de forma automática una interfaz virtual de tipo puente (por defecto, denominada *br0*) que va conectada al kernel del sistema operativo, y el usuario posteriormente se encarga de añadir conexiones a este puente para conectarlas. Esta implementación se puede observar en la Figura 16.

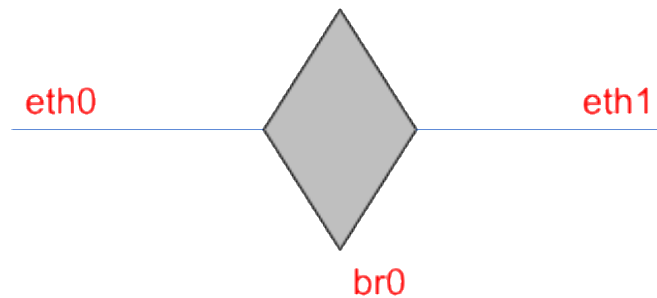


Figura 16: Implementación típica de un switch virtual

Si unimos dos interfaces por un switch, estas dos interfaces pasan a pertenecer a la misma subred y no hace falta ningún encaminador entre las dos interfaces para que los paquetes (en general, *PDU*s o *Protocol Data Unit*) lleguen a su destino correctamente. Esto podría parecer una ventaja, pero en el caso de nuestro ordenador central, es un inconveniente, ya que si unimos la interfaz VLAN833 (que da acceso a la red de la ETSIT) y las interfaces virtuales conectadas a la OLT a través del puente *br0* (creado por el OVS automáticamente al iniciar la utilidad), esta conexión puenteará nuestro router implementado en el ordenador central y la red dejará de funcionar. Esto ocurre porque los paquetes de la subred 10.0.103.0/24 pasarán por el puente *br0* directamente hacia las siguientes subredes pertenecientes a las VLAN (192.168.0.0/24 y 10.19.59.0/24), siguiendo el sentido de las flechas rojas de la Figura 17 y no pasan por el router para ser procesados. Al no pasar por el router, la cabecera de los paquetes no se tratará ni se modificará para encaminarlos de forma adecuada, por lo que el OLT recibirá paquetes de otra subred externa con destino al router de la Oficina Central (por el que no pasará), produciendo bucles y una conectividad nula en nuestra red.

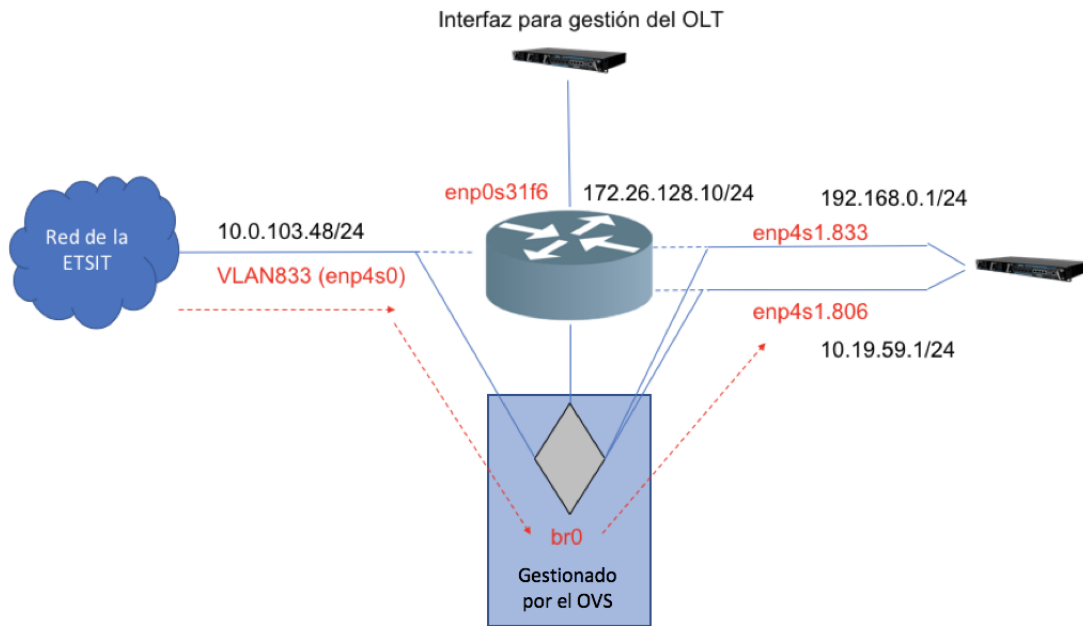


Figura 17: Implementación errónea del switch virtual en la oficina central

Para poder implementar de forma correcta nuestro escenario de red, necesitamos que los paquetes pasen por nuestro COVS (para poder manipularlos y así convertir la red en SDN), pero no podemos dejar la red sin conectividad puenteando las interfaces, es decir, tienen que pasar de forma obligatoria antes por el router del ordenador central. Por lo tanto, se optó por conectar las dos interfaces virtuales (`enp4s1.833` y `enp4s1.806`) al puente y este a su vez al router, tal y como se muestra en la Figura 17.

De esta manera el encaminador instalado en el ordenador central sigue actuando como tal y las dos subredes siguen estando separadas sin un puente que las haga pertenecer al mismo dominio de colisión. De tal manera que la topología de red queda como en la Figura 18.

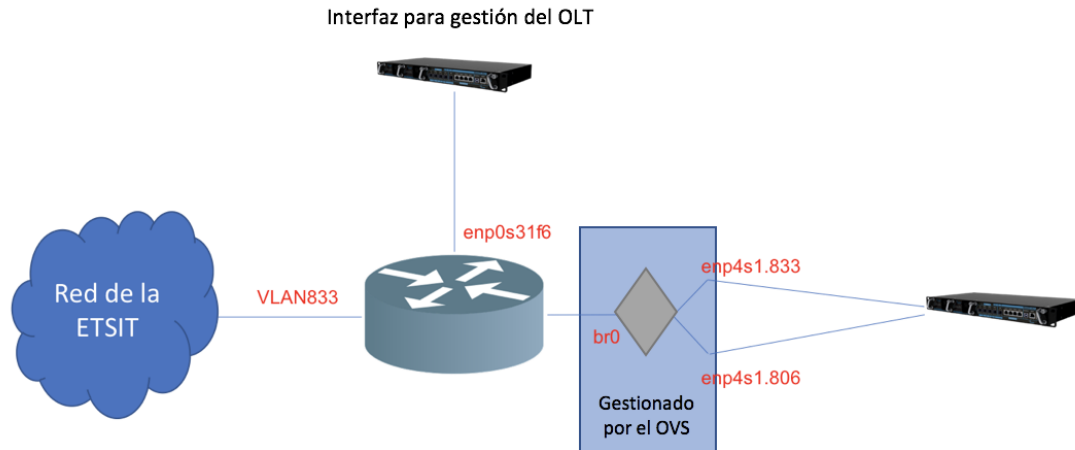


Figura 18: Implementación del switch virtual correcta

En este punto, debemos explicar un punto importante para entender otra parte del funcionamiento de los puentes que generan los switches virtuales. A nivel de red, el puente debe suplantar el direccionamiento de la interfaz a la que sustituye. Como nuestra topología tiene dos interfaces virtuales conectadas al puente, el puente no puede tener a la vez dos direcciones diferentes pertenecientes a las dos subredes (recordemos que cada interfaz virtual es una VLAN que, a todos los efectos, es una subred diferenciada de la otra). Por tanto, debemos conseguir que la interfaz puente tenga a la vez las dos direcciones de cada VLAN. Para ello, vamos a aprovecharnos de los alias. Un alias en Linux es una manera de asignar a una misma interfaz varias direcciones de red diferentes. Para lograr esto simplemente ejecutamos el comando `ifconfig br0:0 10.19.59.1 up`. Como podemos ver en la captura del comando `ifconfig -a` de la Figura 19 para una misma dirección MAC de capa de enlace tenemos dos direcciones diferentes.

```
br0      Link encap:Ethernet  HWaddr 90:e2:ba:e2:42:b2
         inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
         inet6 addr: fe80::92e2:baff:fee2:42b2/64 Scope:Link
         UP BROADCAST RUNNING PROMISC MTU:1500 Metric:1
         RX packets:17 errors:0 dropped:0 overruns:0 frame:0
         TX packets:45 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:1444 (1.4 KB)  TX bytes:2562 (2.5 KB)

br0:0    Link encap:Ethernet  HWaddr 90:e2:ba:e2:42:b2
         inet addr:10.19.59.1  Bcast:10.19.59.255  Mask:255.255.255.0
         UP BROADCAST RUNNING PROMISC MTU:1500 Metric:1
```

Figura 19: Captura del comando `ifconfig -a`

Una vez que hemos explicado todo este proceso, debemos configurar nuestro COVS teniendo en cuenta todo lo que hemos explicado anteriormente. Para encender el switch virtual, usamos uno de los scripts integrados en la instalación del propio switch `sh /usr/local/share/openvswitch/scripts/ovs-ctl start`. Una vez encendido, debemos crear los ejemplares de puente que queramos y las conexiones deseadas a nuestro ordenador. Estas se guardarán, y sólo habrá que configurarlas la primera vez que vayamos a usar la utilidad OVS. OVS tiene dos comandos principales `ovs-vsctl` y `ovs-ofctl`. El primero manipula las opciones de configuración de la base de datos del switch y sus conexiones y el segundo controla todas las opciones relacionadas con OpenFlow. Creamos un puente con el comando `ovs-vsctl add-br br0`. Ahora debemos añadirle las interfaces que queramos conectar a nuestro puente. Para ello, usamos el comando `ovs-vsctl add-port br0 enp4s1.833` y `ovs-vsctl add-port br0 enp4s1.806`. A continuación, simplemente tenemos que suplantar las direcciones de cada interfaz conectada. Para ello, borramos la información de direccionamiento de las interfaces suplantadas (`ifconfig enp4s1.833 0`, `ifconfig enp4s1.806 0`) y rellenamos la del puente y su alias (`ifconfig br0 192.168.0.1 netmask 255.255.255.0`, `ifconfig br0:0 10.19.59.1 netmask 255.255.255.0`).

Después de realizar estos pasos, las subredes de las interfaces virtuales deben tener conectividad a Internet, aunque dependiendo de cada topología de red, a veces puede ser necesario modificar la tabla de encaminamiento para que haya conectividad plena con la siguiente subred, es decir, la subred que genera cada ONT en el lado del cliente final. En nuestro caso, la subred final de cada ONT dependerá del número de la ONT, siendo la subred final de la ONT 1 la 192.168.1.0/24, la de la ONT 2 la 192.168.2.0/24 y así sucesivamente. Como en nuestro escenario tenemos operativas la ONT 2 y la ONT 3 debemos especificar esto en la tabla de encaminamiento del kernel. Para ello, ejecutamos el siguiente comando: `route add -net 192.168.2.0 gw 192.168.0.104 netmask 255.255.255.0 dev br0`. Es decir, que para alcanzar una máquina de la subred 192.168.2.0 se deben encaminar esos paquetes por la dirección 102.168.0.104 (dirección IP de la ONT 2) por la interfaz de red `br0` (interfaz del puente creado por Open vSwitch). De manera análoga ejecutamos el siguiente comando: `route add -net 192.168.3.0 gw 10.19.59.102 netmask 255.255.255.0 dev br0:0`.

Dependiendo de en que VLAN se encuentre cada ONT y que dirección IP le de el servidor DHCP, debemos modificar las direcciones de estos comandos adecuadamente. Para evitar tener que configurar estos parámetros cada vez que se pone en funcionamiento la red, es interesante hacer un DHCP estático (para que a cada dirección MAC de capa 2 le asigne siempre la misma dirección IP).

Una vez está todo configurado a nivel de interfaces y de direccionamiento, tenemos que configurar dos cosas muy importantes para nuestro escenario SDN. Primero, que nuestros puentes usen siempre la versión 1.3 del protocolo OpenFlow [4]. Y segundo, que estemos en el espacio de usuario y no en el espacio de kernel. Estar en el espacio de usuario (espacio *netdev*, también denominado así en la documentación de Open vSwitch) [5] implica poder utilizar herramientas y opciones más avanzadas, como el uso de *meters* de OpenFlow, que son fundamentales para la consecución de nuestro escenario SDN. Por tanto, ejecutamos los siguientes comandos: `ovs-vsctl set bridge br0 protocols=OpenFlow13` y `ovs-vsctl set bridge br0 datapath_type=netdev`. Para finalizar, debemos indicar que queremos establecer un controlador OpenFlow y donde se encuentra éste para que el switch intercambie mensajes OpenFlow con él. Para ello, debemos ejecutar el comando `ovs-vsctl set-controller br0 tcp:10.0.103.45`.

### **3.6 Implementación del ONT basado en SDN con Open vSwitch (SDN – ONT)**

El Open vSwitch del lado de la ONT debe situarse entre el cliente final y la ONT y lo denominaremos ROVS (*Remote OVS*). Por cuestión de simplicidad y eficiencia hemos decidido implementarlo en una Raspberry Pi conectada a la ONT y al usuario a la vez (Figura 19). En este caso, se observa que la implementación topológica de la red es la típica de un switch virtual, y no una especial como discutimos en el Apartado 3.5. La interfaz puente que crea el switch virtual de forma automática (*br0*) unirá dos interfaces de red, una dirigida a la ONT (la que tiene acceso al resto de la red, y por tanto, conectividad) y otra dirigida al cliente final (ordenador en casa del usuario). Dicha topología global se muestra en la Figura 20.

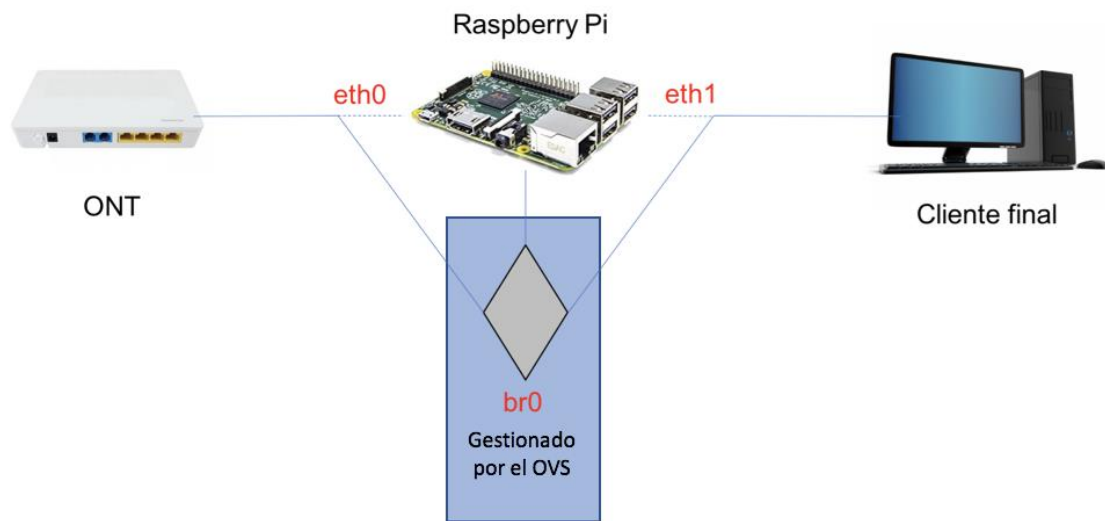


Figura 20: Topología en el switch virtual de la ONT

El resto de configuración es exactamente igual que la realizada en el Apartado 3.5, excepto la configuración particular de cada interfaz. Debemos poner a “0” cada interfaz conectada al puente (`ifconfig eth0 0`, `ifconfig eth1 0`) y posteriormente asignar dirección a la interfaz `br0`. En este punto, para poder adquirir información de la tabla de encaminamiento y hacer el direccionamiento más sencillo, en vez de hacerlo nosotros a mano, vamos a forzar que el puente pida direccionamiento DHCP a la ONT con el siguiente comando (`dhclient br0`).

La conexión con el controlador y la configuración del protocolo y del espacio del usuario es exactamente igual que en el Apartado 3.5. Como el controlador está a dos subredes de distancia, ya que está situado en la subred troncal de la Escuela (10.0.103.0/22), es de vital importancia que las tablas de encaminamiento y la conectividad estén muy bien configuradas, para que los mensajes OpenFlow sigan esta ruta. En el caso concreto de la ONT 2, deberá seguir la ruta 10.0.103.45/24 (subred de la Escuela) -> 10.0.103.48/24 (subred de la Escuela) -> 192.168.0.1/24 (subred entre el ordenador central y las ONT) -> 192.168.0.104/24 (subred entre el ordenador central y las ONT) -> 192.168.2.1/24 (subred entre la ONT 2 y el cliente final) -> 192.168.2.7/24 (subred entre la ONT 2 y el cliente final).

### 3.7 Conexión con el controlador OpenDayLight

Cabe destacar que la distribución de Open vSwitch es muy inestable, sobre todo en el espacio del usuario, e incluso más inestable cuando tres switches se conectan a la vez al mismo controlador. Para poder conectar los tres switches a la vez (los 2 de la Raspberry Pi y el de la Oficina Central), después de varias pruebas sobre esta cuestión, se ha averiguado una manera de mantener la conexión estable a lo largo del tiempo pudiendo modificar flujos y *meters* en tiempo real. Para ello, debemos encender los tres switches e incluir un flujo que permita pasar todos los paquetes. Para incluir este flujo, usamos el comando `ovs-ofctl -O OpenFlow13 add-flow br0 priority=20,actions=normal`. El flujo tiene una prioridad de 20 para que los posteriores flujos que creamos en un futuro tengan más prioridad que el del principio (que solamente sirve para que la conectividad sea la adecuada). Una vez el flujo este funcionando en los tres switches virtuales (podemos visualizar la lista de flujos usando el comando `ovs-ofctl -O OpenFlow13 dump-flows br0`), y comprobemos que hay conectividad con la dirección del controlador (ya sea con la utilidad *ping* u otras utilidades de red), ya podemos encender el controlador OpenDayLight y los tres switches asumirán a este controlador con un rol maestro y podrá añadir, modificar y borrar flujos en cada switch. Para comprobar el estado del controlador, podemos utilizar el comando `ovs-vsctl list controller br0`.

### 3.8 Creación de flujos y *meters* en OpenFlow

Para crear los distintos flujos y *meters* necesarios para el funcionamiento de la red SDN, nos vamos a ayudar de un *plugin* incluido en OpenDayLight llamado YangUI [24]. Un *meter*, aunque la traducción exacta es contador o medidor, se utiliza para restringir la tasa de salida de paquetes de un determinado flujo. Esto nos servirá en el futuro para limitar la tasa de una conexión de nuestra red. YangUI no es más que una interfaz gráfica amigable para el usuario en el que se pueden configurar nuestros flujos usando el lenguaje YANG (que es un lenguaje de modelado de datos basado en NETCONF [25]) accediendo a la interfaz web del controlador OpenDayLight. Gracias a esto, vamos a poder añadir flujos mediante métodos HTTP PUT o consultarlos mediante métodos HTTP GET. Por medio de plantillas ya incluidas en la interfaz gráfica, sólo tenemos que navegar a través de una estructura de árbol para seleccionar la tabla de OpenFlow (*table {id}*) en la que queremos

añadir un flujo (*flow{id}*), y se nos mostrarán una serie de campos para rellenar relacionados con las características del flujo. Para elegir el switch en el que queremos añadir dicho flujo en la tabla deseada, debemos especificar el nodo de OpenFlow que corresponde a cada switch, por ejemplo, *openflow:2*. Si quisiéramos añadir un *meter* simplemente tendríamos que navegar por la estructura de árbol hasta la opción *meter{meter-id}*. Estas opciones en la estructura de árbol se pueden ver resaltados en rojo en la Figura 21.

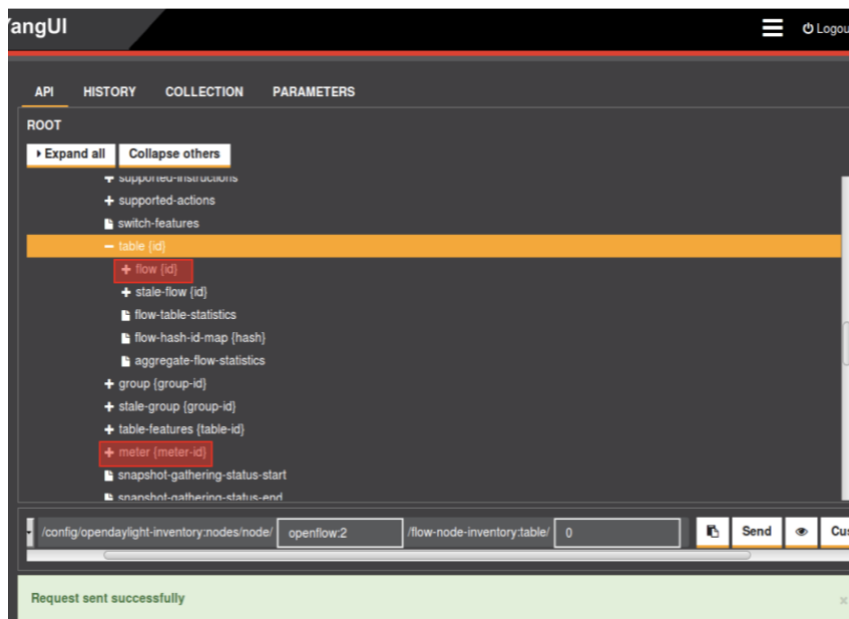


Figura 21: Estructura de árbol en la interfaz YangUI

En la Figura 22 podemos observar una de estas plantillas de YangUI con las opciones referentes a un flujo. Aunque la mayoría de campos son identificativos (nombre del flujo, id del flujo...), hay dos desplegable que permiten configurar dos conceptos muy importantes sobre los flujos de OpenFlow. Estos conceptos son *match* e *instructions*, y los explicaremos en los siguientes párrafos. Por otra parte, los campos resaltados *Priority* y *Hard Timeout* no suelen ser muy importantes en los flujos normalmente, pero serán muy importantes en nuestra implementación particular como veremos en el Capítulo 4. El campo *Priority* permite fijar un número de prioridad para decidir qué flujo se aplica en caso de que algún paquete que pasa por el switch virtual encaje en varios flujos. Este campo lo vamos a utilizar debido a la implementación todavía en desarrollo del OVS comentada en este Capítulo, ya que para que la conexión con el controlador funcione perfectamente necesitamos un flujo inicial que nos permite comunicarnos sin problemas (este flujo tendrá



una prioridad baja), para luego sustituirlo ya por flujos con características más concretas (que tendrán asociados prioridades más altas para que sean los que aplique nuestro switch virtual). *Hard Timeout*, por otra parte, permite especificar un tiempo en segundos a partir del cual el switch virtual borrará un flujo de su tabla de flujos. Esto nos ayudará a automatizar el borrado y la sobrescritura de flujos como veremos en el Capítulo 4 para nuestro caso de uso real planteado.

Figura 22: Campos relacionados con un flujo

El concepto de *match* hace referencia a cómo queremos distinguir que ciertos paquetes que pasan por un switch virtual pasen a formar parte de un flujo concreto. Es decir, podríamos decidir que todos los paquetes que pasan por el switch formen parte de un flujo genérico, pero a veces nos interesa que sólo una serie de paquetes con ciertas características concretas formen parte de un flujo más específico. El campo *match* nos permite distinguir estos paquetes en base a muchas características, tanto en capa de enlace, capa de red o capa de transporte. Recordemos que en principio un switch virtual que no tuviera OpenFlow sólo realiza encaminamiento a nivel de capa de enlace, ya que no distingue tramas de niveles superiores de la capa TCP/IP. Algunos de los campos más utilizados son *ethernet-match*, *vlan-match*, *ip-match* o *tcp-flags-match*, aunque el *match*

predeterminado propuesto por OpenFlow es usar sus propios puertos (*in-port*). Estos campos son visibles en la interfaz de la Figura 23.

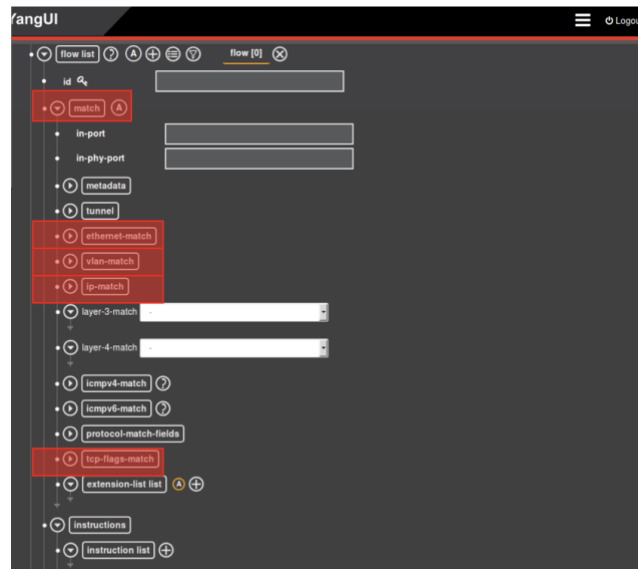


Figura 23: Campos del desplegable *match* en YangUI

El concepto de *instructions*, por otra parte, hace referencia a qué vamos a realizar una vez hemos identificados qué paquetes forman parte de un flujo en concreto. Estas instrucciones son variadas, por ejemplo, aplicar un *meter* (*meter-case*), dirigirnos a otra tabla de OpenFlow (*table-case*) o pasar por realizar una acción genérica (*apply-actions-case*), entre otras. Todas estas instrucciones se pueden consultar en la Figura 24.

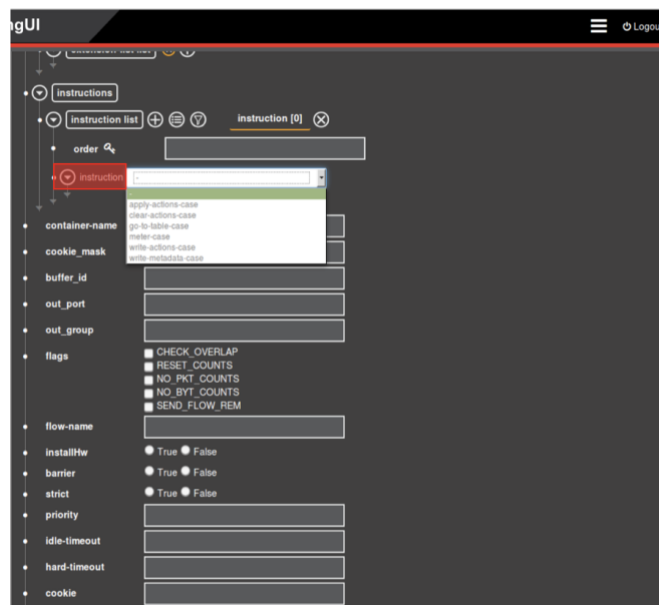


Figura 24: Distintas instrucciones en YangUI

En concreto, la instrucción *apply-actions-case* tiene muchas opciones también, siendo la más importante la acción *output-action-case*, tal y como se observa en el Figura 25.

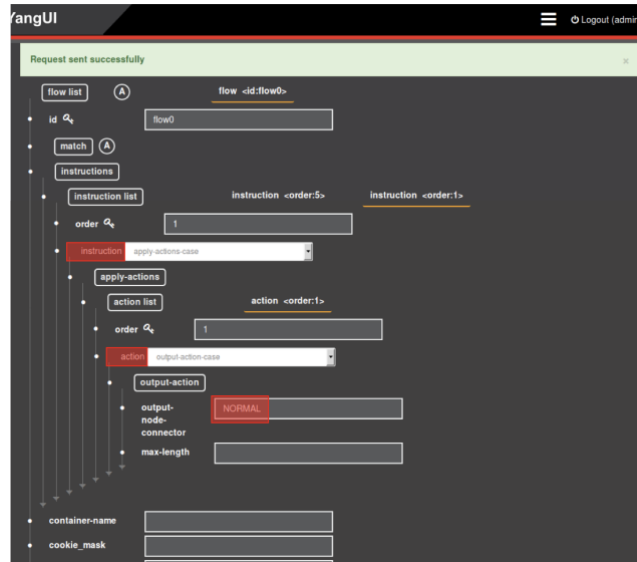


Figura 25: *output-action-case* en YangUI

Esta opción permite especificar una acción de que hacer con los paquetes de salida. La opción más típica es *NORMAL*, como la que tenemos en la Figura 25, que indica que el switch debe comportarse como un switch de capa de enlace normal, pero podemos optar por otras opciones más complejas, desde encaminarlos por una interfaz de OpenFlow concreta hasta acciones relacionadas con la capa de transporte. En la Figura 26 podemos ver una figura propuesta en la documentación de OpenFlow con todas las acciones de *output-actions-case*.

Por otra parte, configurar un *meter* es parecido a configurar un flujo, pero tendremos que rellenar otros campos relacionados con los parámetros consustanciales a un *meter*. Igual que en un flujo, muchos de ellos son simplemente identificativos (como *meter-id* o *meter-name*) pero hay ciertos parámetros que van a influir directamente en como se comporta el *meter*. Los elementos principales que modelan el comportamiento del *meter* son: los *flags*, que nos permiten indicar en que medimos todas las cifras relacionadas con el *meter* (paquetes por segundo, kbps...), los *meter-band*, que nos permiten especificar a partir de que tasa se empieza a aplicar la *band-type*, y la *band-type*, que nos permite especificar si queremos descartar los paquetes a partir de esa tasa (*drop*) o marcarlos a

través del protocolo DSCP (*dscp-remark*). En la Figura 27 podemos ver todos estos campos en la interfaz de YangUI.

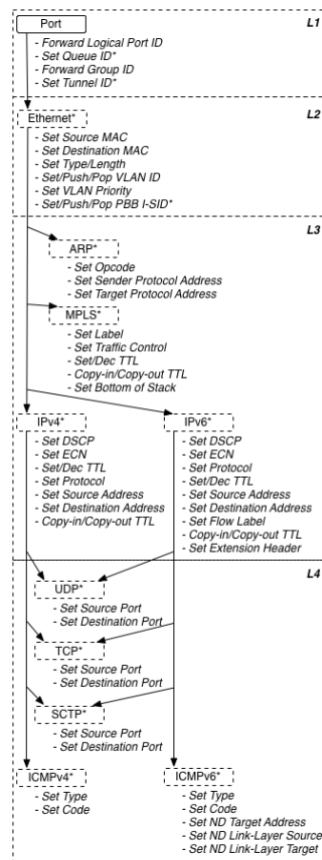


Figura 26: Acciones previstas en el protocolo OpenFlow 1.3 para *output-action-case*

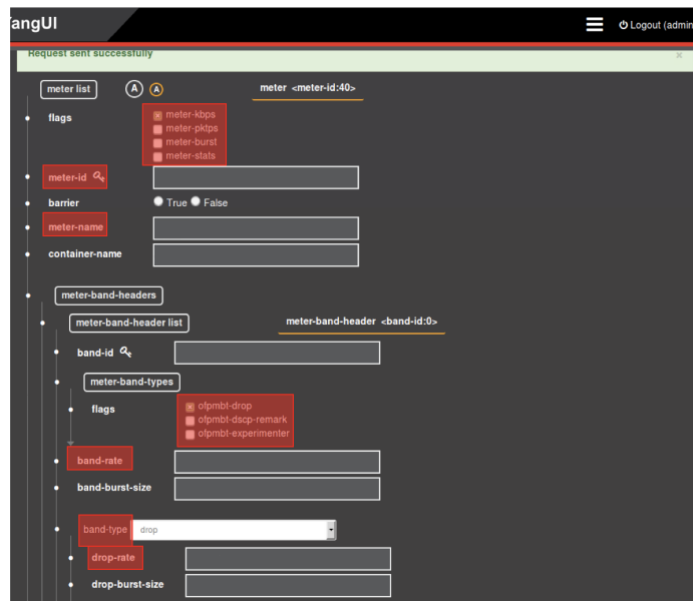


Figura 27: Campos de un *meter* en YangUI

Podemos ver que, tal y como está implementada la interfaz en YangUI y OpenDayLight, aparte de especificar que tipo de banda queremos, tenemos que tener activa una *flag* que indica también lo mismo que habíamos indicado en el desplegable *band-type*. Estas *flags* nos dan tres opciones: *ofpmbt-drop* (que es lo mismo que *drop*), *ofpmbt-dscp-remark* (que es lo mismo que *dscp-remark*) y *ofpmbt-experimenter* (que es una función experimental de OpenFlow 1.3 que no aparece en la documentación de The Open Networking Foundation [4] aunque investigando en los comentarios del código en GitHub sobre OpenDayLight parece funcionar de manera parecida a marcar los paquetes en DSCP [26]).

### 3.9 Conclusiones

En este capítulo de la memoria, hemos descrito la implementación y configuración de un router y un servidor DHCP ayudándonos de utilidades de Linux de red, solventando las diferentes dificultades de esta instalación particular para nuestro caso de uso. Posteriormente, se procedió a instalar los switches virtuales en el ordenador central (COVS) y en las Raspberry Pi (ROVS), cada instalación con sus particularidades a nivel topológico. A continuación, hemos descrito la conexión de los tres switches con el controlador ODL siguiendo una rutina especial para evitar la inestabilidad propia de la utilidad Open vSwitch, pudiendo añadir, modificar o eliminar flujos o meters en cada switch. Finalmente, se ha descrito la creación y configuración de los flujos en el controlador OpenDayLight con la finalidad de configurar y gestionar los servicios de la red GPON a través del protocolo OpenFlow.

# 4

## Caso de uso: Gestión SDN de la QoS en la red GPON

### 4.1 Introducción

En este capítulo vamos a profundizar en el caso de uso que proponemos para testear nuestro escenario SDN-GPON. Para ello, se va a describir el escenario de red experimental, nuestro caso de uso concreto y los pasos necesarios para desarrollar una aplicación del lado del usuario que le permita realizar peticiones en tiempo real de los servicios contratados. Concluiremos con una discusión sobre los resultados experimentales de nuestro caso de uso.

### 4.2 Descripción del escenario de red experimental

Nuestro escenario de red final consiste en una pequeña modificación sobre nuestro escenario principal. En nuestro escenario experimental, tenemos sólo activas dos ONTs basadas en SDN (es decir, tendremos sólo dos usuarios finales), tal y como se muestra en la Figura 28.

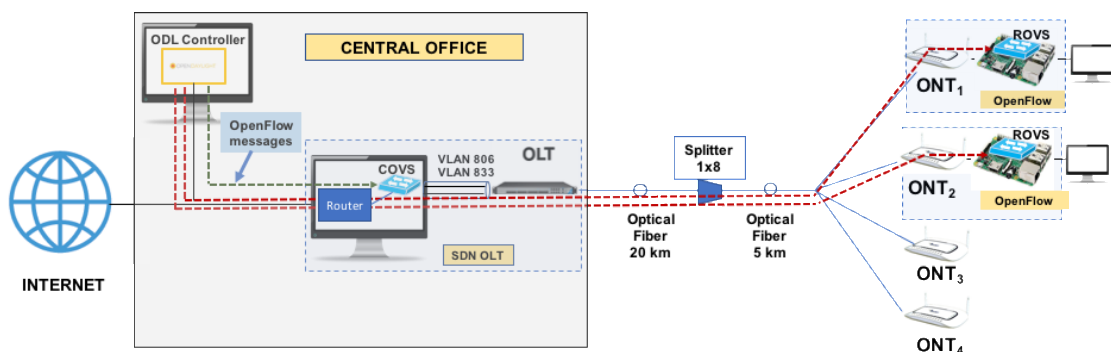


Figura 28: Escenario de red experimental SDN-GPON

Como se muestra en la Figura 28, el controlador OpenDayLight se comunica a la vez con el Open vSwitch de la oficina central (COVS) y con los Open vSwitch de la ONT 1 y 2 (ROVS). Como necesitamos dos switches virtuales detrás de cada ONT (entre el usuario final y la ONT), usaremos dos Raspberry Pi, cada una para simular el ROVS necesario para cada ONT activada. Cada usuario tendrá asociado dos flujos de OpenFlow para cada servicio, uno relacionado con los requisitos en el sentido de subida y otro para los requisitos en el sentido de bajada. Los flujos de bajada deberán configurarse y enviarse al COVS del OLT, mientras que los de subida deberán configurarse y enviarse al ROVS de cada ONT. Aunque hablamos de configuración de flujos en cada uno de los OVS, ya que son los que realmente van a controlar el tráfico en ambos canales de la red, éstos se configuran automáticamente en un controlador centralizado ODL que se comunicará de forma automática y transparente con ellos a través de la red GPON. Estos flujos tienen que reconocer a los paquetes que coinciden con las condiciones aplicadas y después aplicaremos las medidas que queramos a los paquetes pertenecientes a estos flujos (dejarlos pasar, encaminarlos a otra subred, limitar la tasa de paquetes enviados, etc...).

### 4.3 Descripción del caso de uso

Nuestro caso de uso lo vamos a centrar en un nuevo modelo de negocio de las operadoras en las que los usuarios de la red pagan por aumentar su ancho de banda en ciertos momentos en los que se usen servicios que requieren de un alto ancho de banda, tales como juegos en línea o televisión en Ultra Alta Definición (*UHD* o 4K). Siguiendo con el escenario propuesto en la Figura 28, vamos a definir para el Usuario 1 (conectado a la ONT 1) un servicio de Internet con un ancho de banda simétrico garantizado fijado a 20 Mbps y para el Usuario 2 (conectado a la ONT 2) otro servicio de Internet con un ancho de banda simétrico garantizado de 10 Mbps. Estos dos servicios de Internet van a estar en la VLAN 833, sin perjuicio de que en un futuro pudieran incluirse otros servicios en la VLAN 806 ya configurada u otras VLAN que se implementasen. Los usuarios están conectados con la ONT a través de una Raspberry Pi en la que tenemos instalado un switch virtual (ROVS) que controla el tráfico del servicio en el sentido *upstream* de la comunicación, y antes de llegar a la oficina central tenemos instalado otro switch virtual que controla el tráfico del servicio en el sentido *downstream* (COVS). Como todos estos switches virtuales están habilitados para trabajar con el protocolo OpenFlow, vamos a controlar los servicios y a monitorizar su rendimiento a través de este protocolo. En la

Figura 29 se muestra una tabla con los campos típicos asociados a un flujo y los que vamos a utilizar nosotros (marcados con una “x” en color rojo).

Para poder distinguir los paquetes que van dirigidos en sentido de bajada (desde el OLT a las ONTs) utilizaremos la opción “*Dirección destino MAC*” del campo *match* de nuestro flujo, en el que tenemos que introducir una dirección destino de nivel de capa de enlace. Como los paquetes en sentido *downstream* van dirigidos hacia la ONT, aquí pondremos la dirección MAC del lado WAN de la ONT. En cambio, cuando queremos distinguir los paquetes que van en sentido *upstream*, también vamos a usar el campo “*Dirección destino MAC*”, pero en este caso pondremos la dirección MAC del lado LAN de la ONT.

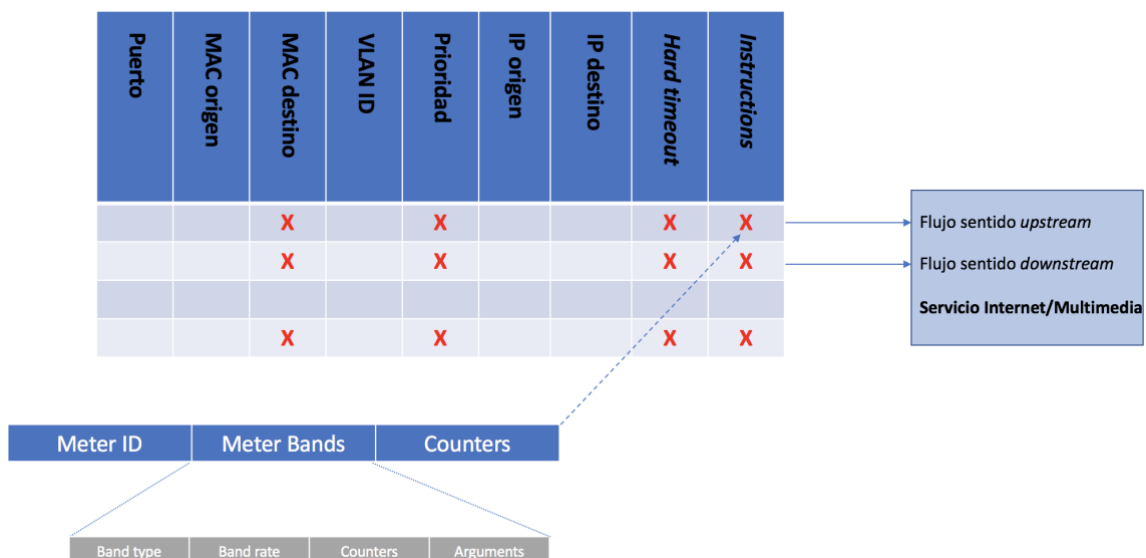


Figura 29: Esquema de los diferentes campos de los flujos que vamos a utilizar en el caso de uso

En este sentido, cabe recordar que la smartOLT crea una interfaz diferente de red para cada servicio configurado en la red en cada ONT. Como cada servicio se crea en una VLAN (y, por tanto, una subred diferente), necesita crear en la ONT interfaces de red diferentes. Esto implica que al tener diferentes interfaces de red habrá paquetes que vayan dirigidos a direcciones MAC de la capa de enlace diferentes, cuando en realidad están yendo a la misma ONT. Este fenómeno es debido a la creación de estas interfaces de red dedicadas a cada VLAN. Podemos observar las distintas interfaces de red en la Figura 30.



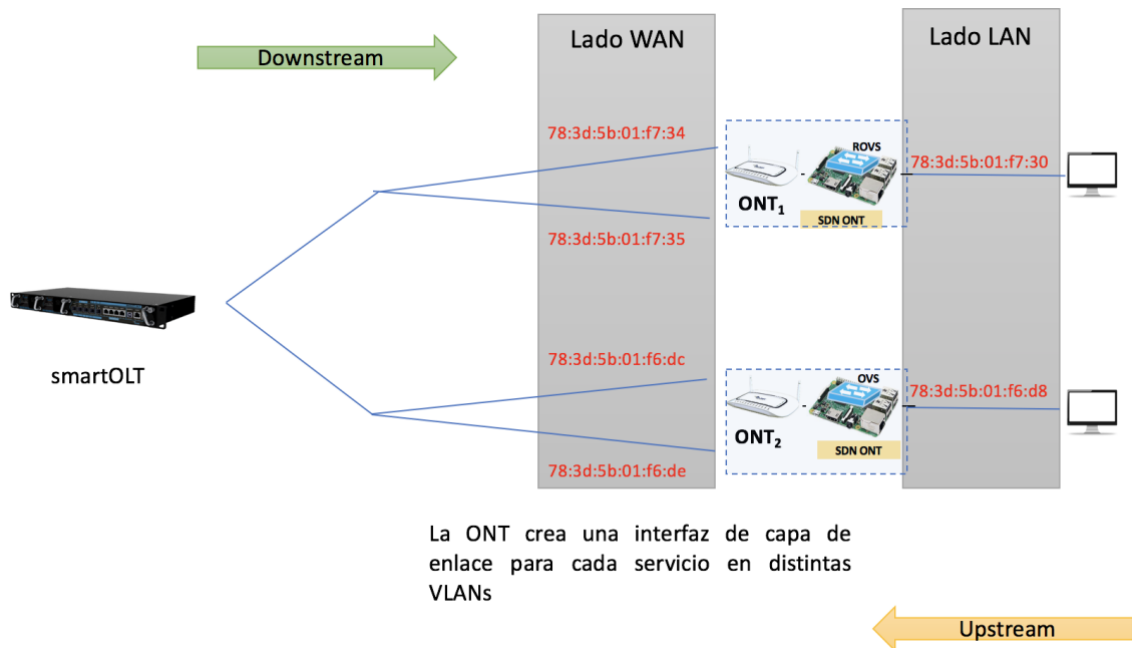


Figura 30: Interfaces con varias VLAN en el lado WAN en nuestra topología de red

El lector podrá observar que entre la oficina central y la ONT está colocado el OLT, pero es transparente para las comunicaciones, por tanto, las tramas que salen con destino a los usuarios no tienen como dirección destino la OLT. Con todo lo anteriormente mencionado, simplemente vamos a distinguir los paquetes que queremos interceptar, pero posteriormente tendremos que manipularlos para alterarlos, borrarlos, etc.

Para conseguir interceptar y tratar los paquetes de subida y bajada con los flujos de OpenFlow, vamos a ayudarnos del campo *match* de estos flujos, como se observa en la Figura 31. Como explicamos en el Capítulo 3, este campo se usa para interceptar paquetes. Por simplicidad de nuestros flujos, y para una más fácil implementación de los módulos requeridos en Python y los programas en PHP que posteriormente se desarrollarán en el Apartado 4.4, nosotros hemos decidido utilizar el campo “*dirección MAC destino*” (campo *ethernet-destination address* en Figura 31) como *match*. Es decir, vamos a interceptar los paquetes según su destino a nivel de enlace (capa 2).

Figura 31: Campo *match* del flujo

Después de interceptar los paquetes gracias a nuestros flujos de OpenFlow, vamos a utilizar dos *instructions* diferentes en cada flujo para lograr que el flujo actúe como nosotros queremos. Una instrucción será un *apply-actions-case* con una acción *output-action-case* configurada como NORMAL para que trabaje como un switch normal y la otra será una instrucción *meter-case* asociada a un parámetro *meter* en la que vamos a configurar la velocidad máxima de cada cliente. Es necesario configurar las dos en el mismo flujo ya que necesitamos tener conectividad normal y fijar una velocidad a la vez.

El campo *Hard Timeout* va a contener los minutos de navegación *premium* del cliente final, para que como hemos explicado en el Capítulo 3 cuando se acabe su tiempo de ancho de banda más elevado, el flujo se borre automáticamente y el usuario vuelva a su configuración normal. Si el flujo se borrara y no hubiera otro flujo por defecto configurado previamente, el cliente dejaría de tener conectividad, ya que sus paquetes dejarían de coincidir con un flujo concreto y el switch descartaría dichos paquetes. Es por eso que el campo *Priority* tiene una importancia añadida aparte de la comentada en el Capítulo 3, ya que nos va a permitir distinguir el flujo con ancho de banda base y el flujo con ancho de banda *premium*. Primero, vamos a configurar siempre un flujo con el ancho de banda base del cliente (10 Mbps para el Usuario 1 y 20 Mbps para el Usuario 2) que tendrá una prioridad baja. Posteriormente, los flujos que tengan un ancho de banda elevado tendrán asociada una prioridad alta, pero se borrarán por el campo *Hard Timeout* cuando se acabe el tiempo de navegación *premium*. De esta manera, cuando se borran, se vuelve a aplicar

el flujo con ancho de banda base con prioridad de baja para que el cliente final disfrute otra vez de la velocidad contratada inicialmente.

Es importante entender que en algunas circunstancias por cada cliente tendremos 2 flujos en sentido *upstream*, uno de ellos con la tarifa base y otro de ellos que se activará a petición del cliente con una tarificación diferente, y otros dos flujos análogos en sentido *downstream*. Estos flujos se separan por cada sentido de la red y se comprueba si la red GPON tiene capacidad suficiente para soportar que ciertos usuarios gocen de más ancho de banda justo antes de programar un flujo que permita tasas más elevadas durante ciertos intervalos de tiempo. Así, los flujos en sentido *downstream* se programan y envían al el switch virtual situado en el lado de la OLT y los flujos en sentido *upstream* se programan en los switches virtuales situados del lado de la ONT. Posteriormente, en este capítulo veremos que una aplicación en Python (en el lado de la Oficina Central) que se comunica con la interfaz web del cliente será la encargada de calcular si hay capacidad en la red de acceso antes de confirmar al usuario de que se va a cumplir su solicitud y de comunicarse con el controlador OpenDayLight para programar los flujos necesarios.

Respecto a los *meters*, como ya hemos explicado en el Capítulo 3, para configurar la velocidad máxima de los servicios del usuario debemos modificar los campos *band-rate*, *band-burst-size*, *drop-rate*, *drop-burst-size*. Para ello, realizamos varias pruebas para lograr un comportamiento adecuado, y concluimos que la manera de fijar la velocidad del cliente a la deseada será ajustar la *band-rate* y la *drop-rate* a la velocidad que queramos. Según la documentación de OpenFlow 1.3 y la de OVS, si el campo de *band-burst-size* y *drop-burst-size* está vacío, el switch virtual optimiza las ráfagas para descartar paquetes a un ritmo adecuado, pero YangUI y OpenDayLight obligan a completar este campo y no se puede dejar vacío. Por todo lo anteriormente citado, para configurar una tasa de velocidad a nuestro cliente de 40 Mbps, debemos rellenar el *band-rate* y el *drop-rate* a 40000 (ya que las unidades son Kbps) y rellenar el *band-burst-size* y *drop-burst-size* a 0. En la Figura 32 se muestra un ejemplo de un *meter* configurado a 40 Mbps (*meter-id:40*).

Request sent successfully

meter list A A meter <meter-id:40>

flags  meter-kbps  
 meter-pktps  
 meter-burst  
 meter-stats

meter-id

barrier  True  False

meter-name

container-name

meter-band-headers

meter-band-header list meter-band-header <band-id:0>

band-id

meter-band-types

flags  ofpmbt-drop  
 ofpmbt-dscp-remark  
 ofpmbt-experimenter

band-rate

band-burst-size

band-type

drop-rate

drop-burst-size

Figura 32: Meter configurado a 40 Mbps

#### 4.4 Desarrollo de una aplicación en el usuario para la gestión dinámica de los servicios contratados

Para facilitar al usuario la gestión dinámica y autónoma de su ancho de banda contratado se ha programado una aplicación web para que dicho usuario pueda realizar peticiones temporales bajo demanda de nuevos requisitos en los servicios contratados. Esta aplicación ha sido desarrollada en HTML5/CSS/Javascript y consiste en una interfaz sencilla con varios campos donde el usuario puede introducir los datos requeridos (nuevo ancho de banda, franja temporal) y enviar dicha petición a la Oficina Central del operador de forma transparente. En nuestro caso de uso concreto, la aplicación web está almacenada

en la Raspberry Pi que usamos de switch virtual en el lado de la ONT y se accederá a ella a través de una dirección IP.

Para instalar la aplicación en la Raspberry Pi nos hemos ayudado de un servidor web típico como Apache [27], en concreto, una versión de Apache que se distribuye como *apache2*. Para instalar dicho servidor web en la Raspberry Pi, tenemos que introducir una serie de comandos que nos van a permitir descargar la versión más reciente de dicho servidor web, además de una serie de dependencias necesarias para ejecutar PHP y MySQL (en caso de que el desarrollo posterior de esta aplicación web incluya el uso de bases de datos). Para ello, se deben introducir los comandos `apt-get install apache2` (que instala el servidor web propiamente dicho), `sudo apache2ctl configtest` (que se cerciora de que el servidor web está configurado correctamente) y `apt-get install php libapache2-mod-php php-mcrypt php-mysql` (que instala las dependencias relacionadas con PHP y MySQL previamente mencionadas). Después de configurar todo lo anterior, simplemente tenemos que colocar la raíz de la página web en la ruta `/var/www/html` y nuestro servidor web será accesible desde cualquier ordenador. Por ejemplo, si nuestro servidor web está en una Raspberry Pi con una dirección IP 192.168.3.4 sólo tendremos que introducir en el navegador 192.168.3.4 y automáticamente el navegador nos dirigirá a la página `http://192.168.3.4/index.html`, siendo este fichero html el que está almacenado en el directorio `/var/www/html`. Esto nos permite que la aplicación web sea accesible desde la subred privada del cliente, pero también desde el ordenador central en caso de que haya que realizar cualquier mantenimiento o modificación en tiempo real. Esta interfaz se puede observar en la Figura 33. El código HTML5 completo se puede consultar en el Anexo III. En esta interfaz, el cliente puede indicar el ancho de banda que queremos en una barra deslizadora e indicar el tiempo en minutos en otro campo del formulario para posteriormente pulsar el botón “*Enviar*” para que se tramite su solicitud. También se ha programado un formulario de contacto para que el cliente se comunique con el administrador de red y una opción para visitar la página traducida a inglés.



Figura 33: Página web para la gestión dinámica del ancho de banda

El código PHP que se ejecuta de forma paralela al enviar la petición a través del formulario, se encarga de enviar los datos con los nuevos requisitos del servicio a un programa situado en la Oficina Central. Este envío se hará mediante un socket TCP al programa anteriormente mencionado (desarrollado en Python) que está escuchando en la Oficina Central. Uno de los métodos programados dentro de este programa recibe los datos y cierra el socket anteriormente creado. Posteriormente, el programa situado en la Oficina Central se encarga de comprobar si el ancho de banda deseado por el cliente se puede reservar durante el tiempo que éste requiere, monitorizando la red de acceso en tiempo real para ver si tiene capacidad suficiente para proporcionar el servicio durante todo ese tiempo. Una vez ha comprobado positiva o negativamente dicha cuestión, abre otro socket TCP con la respuesta a la interfaz web del cliente que se queda escuchando hasta recibir esta confirmación para informar al usuario si la nueva petición de ancho de banda ha sido aprobada o no, tal y como se observa en la Figura 34.

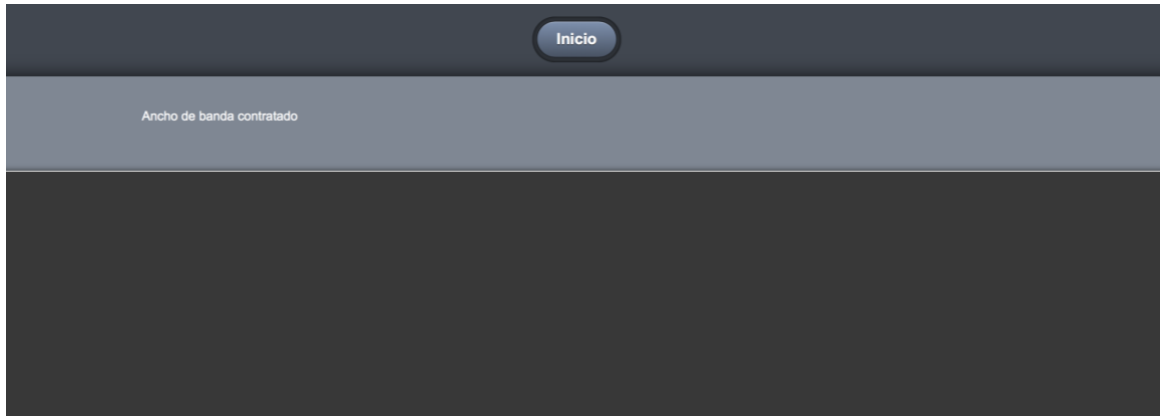


Figura 34: Mensaje de confirmación en PHP

Finalmente, el socket TCP se cierra, y el programa de la Oficina Central se encarga de programar y enviar los distintos flujos necesarios a todo los switches virtuales a través del controlador OpenDayLight. El código PHP de la interfaz web y el método en Python encargado de la comunicación entre el usuario y la Oficina Central se pueden consultar en los Anexos IV y el Anexo V, respectivamente.

## 4.5 Discusión de resultados experimentales

Para analizar el comportamiento de la red y obtener los resultados experimentales, hemos configurado la ONT 1 con un servicio de Internet de 10 Mbps de ancho de banda simétrico por defecto y un aumento a 20 Mbps requerido durante 5 minutos. Para la ONT 2, hemos configurado un servicio de Internet de 20 Mbps de ancho de banda simétrico por defecto y un aumento a 35 Mbps requerido durante 5 minutos. Cabe preguntarse por qué hemos configurado unos anchos de banda tan bajos si la maqueta GPON de nuestro laboratorio admite anchos de banda mucho más altos. Esto es debido principalmente a dos fenómenos que son ajenos a nosotros.

El primero de los efectos, es que necesitamos estar en el espacio de usuario del switch virtual para poder configurar *meters* en dicho switch. Como ya se comentó en el Capítulo 3, este modo (también llamado *netdev* en la documentación de Open vSwitch [5]) está todavía en desarrollo y no se comporta de una manera correcta. En nuestras pruebas, debido a que el switch virtual se encuentra en el espacio de usuario llamado *netdev* anteriormente citado, nuestra velocidad se reducía a un valor de ~40 Mbps como máximo si se instalaba en una Raspberry Pi. Por otra parte, si lo instalábamos en otro sistema

operativo convencional (PC/Mac/Linux) basado en una arquitectura x86/64 la velocidad de bajada se reducía considerablemente, pero la velocidad de subida se reducía a un valor de ~1 Mbps haciendo imposible cualquier tipo de conexión estable y satisfactoria para el cliente. El segundo fenómeno es que el conector Ethernet de la Raspberry Pi 3 Model B [28] (la más reciente en el mercado al realizar este trabajo) está limitado a 100 Mbps y además no es un conector Ethernet real, si no que está soldado al hub USB de la placa base de la Raspberry Pi, compartiendo el ancho del bus la conexión Ethernet y los 4 puertos USB2.0 de los que se dispone. Esto se traduce en una limitación en la velocidad de la conexión que se recibe desde la ONT a 100 Mbps en las mejores condiciones (teniendo unos resultados reales de ~80 Mbps sin encender el switch virtual, ya que si lo encendíamos nos afectaba el problema de *netdev* en las medidas realizadas).

Una vez analizada la problemática asociada a nuestras conexiones, se procedió a realizar las pruebas reales. Para ello, vamos a utilizar la utilidad *iperf* [29], que se utiliza para medir la velocidad de conexiones TCP, UDP y SCTP. Para instalarla en un sistema Linux, simplemente ejecutamos el comando `apt-get install iperf`. Una vez está instalada, se ejecuta en los dos extremos de la conexión, en el que una máquina actuará de emisor y la otra de receptor. Para utilizar la utilidad debemos familiarizarnos con sus diferentes opciones y parámetros [30]. En el lado del sumidero, debemos ejecutar el comando `iperf -s`, mientras que en el lado del cliente debemos indicar que queremos correr la aplicación en modo cliente (`iperf -c`). También tenemos que indicar en qué dirección IP se encuentra el sumidero y durante cuánto tiempo queremos realizar la prueba en segundos (`-t`). Por defecto la utilidad transmite durante 10 segundos, lo cual es demasiado corto para simular una conexión de un cliente. Por todo lo anteriormente citado, en el lado del cliente debemos ejecutar el comando `iperf -c 10.0.103.45 -t 900` para una conexión con un sumidero situado en la dirección IP 10.0.103.45 durante 15 minutos (900 segundos).

Mientras la utilidad *iperf* se encarga de medir la velocidad de la conexión, nosotros vamos a ir cambiando los flujos desde el controlador para modificar la velocidad del cliente. Al mismo tiempo, se va a monitorizar la red con la utilidad *Wireshark* [31], un conocido analizador de protocolos de red, que se va a utilizar para agrupar los paquetes correspondientes a nuestra prueba de velocidad para posteriormente realizar una gráfica del rendimiento de la conexión una vez finalicen los 15 minutos programados en la utilidad



*iperf*. En nuestro caso de uso, se va a dejar 5 minutos al ancho de banda por defecto contratado por el cliente, 5 minutos al nuevo ancho de banda contratado por el cliente, y el retorno al ancho de banda por defecto cuando se acaban los 5 minutos contratados por el cliente con la nueva velocidad.

En las dos pruebas, los tres switches virtuales (el situado entre el ordenador central y la OLT, el situado detrás de la ONT 1 y el situado detrás de la ONT 2) se encuentran encendidos y conectados con el controlador OpenDayLight a la vez para simular el escenario y el caso de uso propuesto de una manera fiel.

#### **4.5.1 Análisis de resultados en la ONT 1**

En este primer caso, vamos a analizar el flujo *upstream* de la ONT 1, cuya velocidad por defecto es de 10 Mbps y en la que se realiza una petición de 20 Mbps durante 5 minutos. Para poder observar los distintos mensajes OpenFlow que se envían en la red vamos a monitorizar la red fijándonos en los intercambios de mensajes entre las direcciones 10.0.103.45/24 (controlador ODL situado en una máquina virtual de la máquina de la oficina central) y 192.168.0.114/24 (dirección IP de la ONT 1, que es de donde vendrán los mensajes del switch virtual situado detrás de la ONT 1 en una dirección del tipo 192.168.1.x/24). Si nos fijáramos simplemente en los mensajes que mandamos desde la interfaz gráfica asociada al controlador ODL no sabríamos si realmente se está mandando los requisitos deseados y si los parámetros coinciden con los introducidos por nosotros. Por tanto, vamos a aprovecharnos de que el protocolo OpenFlow 1.3 [4] exige un intercambio constante de mensajes OpenFlow entre el controlador ODL y los switches OVS con información relativa a flujos, *meters*, tablas... etc., de modo que se podrán observar en todos los instantes qué flujos hay almacenados en el cada uno de los OVS's. Al comienzo de la prueba, deberíamos tener un flujo activo con un *match* con dirección Ethernet destino la ONT 1 del lado LAN (78:3d:5b:01:f6:d8) y dos instrucciones asociadas a este flujo relativas a como se debe encaminar (*output-action-case*) y que *meter* se debe asociar a este flujo (*meter-case*). Se observa en la Figura 35 que en el segundo 2.53 (parámetro *Time* en *Wireshark*) aparece un flujo que cumple todo lo anteriormente dicho en una trama de tipo OFPT\_MULTIPART\_REPLY sobre una trama OFMP\_FLOW. Esto quiere decir que el switch virtual está contestando al controlador que previamente le ha mandado una trama de tipo OFPT\_MULTIPART\_REQUEST en la que le pide toda la información relacionada con los flujos almacenados en dicho switch (OFMP\_FLOW). Si

quisiera la información de las tablas, por ejemplo, mandaría una trama OFPT\_MULTIPART\_REQUEST sobre OFPMP\_TABLE, y el switch virtual contestaría con una trama OFPMP\_TABLE sobre OFPT\_MULTIPART\_REPLY. Además, tiene un campo *Priority* con valor 4000 (una prioridad arbitraria diferente que 0 para distinguirla del flujo inicial) y un campo *Hard Timeout* con valor 0 ya que no queremos que el flujo se borre automáticamente.

No.	Time	Source	Destination	Protocol	Length	Info
4378	2.521081243	192.168.0.114	10.0.103.45	OpenF...	388	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
4405	2.532864690	10.0.103.45	192.168.0.114	OpenF...	122	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW
4406	2.534216363	192.168.0.114	10.0.103.45	OpenF...	620	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
4436	2.551766832	10.0.103.45	192.168.0.114	OpenF...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC

```

Transaction ID: 459
Type: OFPMP_FLOW (1)
Flags: 0x0000
Pad: 00000000
Flow stats
  Length: 104
  Table ID: 0
  Pad: 00
  Duration sec: 120
  Duration nsec: 372000000
  Priority: 4000
  Idle timeout: 0
  Hard timeout: 0
  Flags: 0x0000
  Pad: 00000000
  Cookie: 0x06a3e59f50932018
  Packet count: 9182
  Byte count: 11273066
Match
  Type: OFPMT_OXM (1)
  Length: 20
  OXM field
    Class: OFPXMC_OPENFLOW_BASIC (0x8000)
    0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
    ... ..0 = Has mask: False
    Length: 6
    Value: TelnetRe_01:f6:d8 (78:3d:5b:01:f6:d8)
  OXM field
    Pad: 00000000
Instruction
  Type: OFPIT_METER (6)
  Length: 8
  Meter ID: 10
Instruction
  Type: OFPAT_APPLY_ACTIONS (4)
  Length: 24
  Pad: 00000000
  Action
    Type: OFPAT_OUTPUT (0)
    Length: 16
    Port: OFPP_NORMAL (0xfffffafa)
  
```

Figura 35: Información sobre el paquete OFPMP\_FLOW del flujo por defecto de 10 Mbps

Por otra parte, en la Figura 36 se muestra la trama de tipo OFPMP\_METER\_CONFIG en una trama OFPT\_MULTIPART\_REPLY. De manera análoga a lo explicado con los mensajes OFPMP\_FLOW esta trama contiene información sobre *meters*, concretamente, en este caso, la información asociada al *meter* con identificador 10 (*Meter ID:10* en la Figura 35). Este *meter* tiene la instrucción de tirar paquetes para lograr una tasa máxima de 10 Mbps (10000 en la Figura 35, ya que las cantidades se expresan en Kbps), justamente el ancho de banda que tiene contratado

nuestro cliente por defecto. También podemos observar en dicha figura que Wireshark erróneamente detecta el *Drop-rate* del que hablábamos en el Apartado 4.3 como *Burst Size* y el *Band-rate* lo especifica simplemente como *Rate*. Esto no es un error del protocolo o que los mensajes se estén enviando incorrectamente, sino que la información relativa al protocolo OpenFlow 1.3 está mapeada incorrectamente en la utilidad Wireshark, así que no debemos preocuparnos por eso.

No.	Time	Source	Destination	Protocol	Length	Info
4439	2.558008833	192.168.0.114	10.0.103.45	OpenF...	84	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP
4440	2.558244607	10.0.103.45	192.168.0.114	OpenF...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER_CONFIG
4441	2.559106368	192.168.0.114	10.0.103.45	OpenF...	156	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_CONFIG
4458	2.567198813	10.0.103.45	192.168.0.114	OpenF...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER

```

▶ Frame 4441: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits) on interface 0
▶ Ethernet II, Src: TelnetRe_01:f6:dc (78:3d:5b:01:f6:dc), Dst: IntelCor_e2:42:b2 (90:e2:ba:e2:42:b2)
▶ Internet Protocol Version 4, Src: 192.168.0.114, Dst: 10.0.103.45
▶ Transmission Control Protocol, Src Port: 44620, Dst Port: 6633, Seq: 6705, Ack: 597, Len: 88
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 88
  Transaction ID: 462
  Type: OFPMP_METER_CONFIG (10)
  ▶ Flags: 0x0000
  ▶ Pad: 00000000
  ▶ Meter config
  ▶ Meter config
  ▼ Meter config
    Length: 24
    ▶ Flags: 0x00000001
    Meter ID: 10
    ▼ Meter band
      Type: OFPMBT_DROP (1)
      Length: 16
      Rate: 10000
      Burst size: 10000
      Pad: 00000000
  
```

Figura 36: Información sobre el paquete OFPMP\_METER\_CONFIG con ID 10

Estos flujos iniciales van a conservarse durante 5 minutos hasta que en el segundo 300 deberíamos ver un nuevo mensaje OFPT\_FLOW\_MOD desde el controlador ODL hacia el switch del lado de la ONT indicándole que añada un flujo concreto a través del comando OFPFC\_ADD. Este mensaje aparece porque a través de la interfaz web de la Figura 37 se configura un nuevo servicio de 20 Mbps durante 5 minutos. Para ello, se crea un nuevo flujo que tendrá las mismas características de *match* que el flujo anterior ya presente excepto tres características. La primera, debe tener una prioridad mayor (*Priority: 6000*) que el anterior flujo (*Priority: 4000*) para que sea el flujo a aplicar por el switch virtual durante los siguientes 5 minutos. La segunda, es que debe tener un campo *hard timeout* para que el flujo se borre automáticamente cuando acabe el tiempo deseado por el cliente (en este caso, 300 segundos, que son 5 minutos). Por último, debe tener un *meter* asociado diferente al del flujo ya presente, en nuestro caso, de 20 Mbps (*Meter ID: 20*). Efectivamente, en el segundo 300.9 se observa un mensaje OpenFlow

OFPT\_FLOW\_MOD con esas características que podemos observar con detenimiento en la Figura 38.

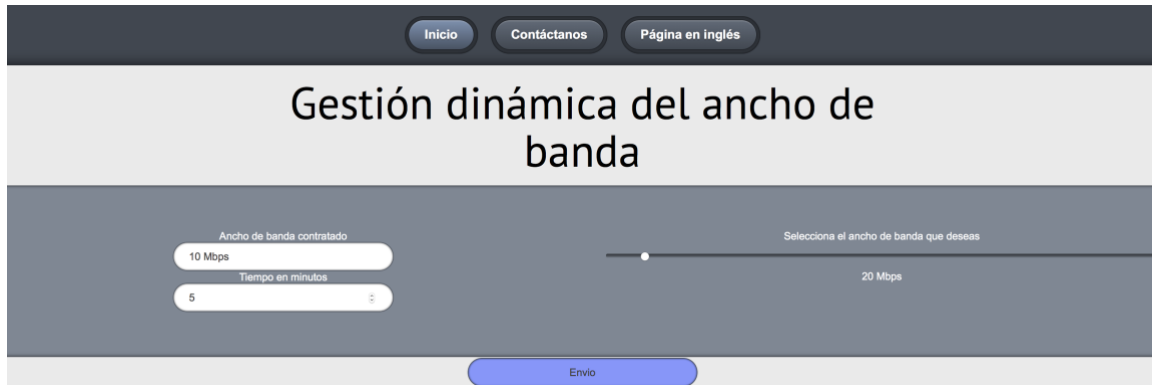


Figura 37: Captura de la interfaz web solicitando el ancho de banda a 20 Mbps

515...	300.7079516...	10.0.103.45	192.168.0.104	OpenF...	518	Type: OFPT_PACKET_OUT
515...	300.7098276...	10.0.103.45	192.168.0.114	OpenF...	518	Type: OFPT_PACKET_OUT
515...	300.9067265...	10.0.103.45	192.168.0.114	OpenF...	170	Type: OFPT_FLOW_MOD
515...	300.9634030...	10.0.103.45	192.168.0.104	OpenF...	74	Type: OFPT_BARRIER_REQUEST

```

Cookie mask: 0x0000000000000000
Table ID: 0
Command: OFPFC_ADD (0)
Idle timeout: 0
Hard timeout: 300
Priority: 6000
Buffer ID: OFP_NO_BUFFER (0xffffffff)
Out port: OFPP_ANY (0xffffffff)
Out group: OFPG_ANY (0xffffffff)
Flags: 0x0000
Pad: 0000
Match
Type: OFPMT_OXM (1)
Length: 20
OXM field
Class: OFPXM_OPENFLOW_BASIC (0x8000)
0000 011. = Field: OFPXM_OF_ETH_DST (3)
... ..0 = Has mask: False
Length: 6
Value: TelnetRe_01:f6:d8 (78:3d:5b:01:f6:d8)
OXM field
Class: OFPXM_OPENFLOW_BASIC (0x8000)
0000 110. = Field: OFPXM_OF_VLAN_VID (6)
... ..0 = Has mask: False
Length: 2
...0 ... .. = OFPVID_PRESENT: False
... 0000 0000 0000 = Value: 0
Pad: 00000000
Instruction
Type: OFPIT_APPLY_ACTIONS (4)
Length: 24
Pad: 00000000
Action
Type: OFPAT_OUTPUT (0)
Length: 16
Port: OFPP_NORMAL (0xffffffffa)
Max length: 0
Pad: 000000000000
Instruction
Type: OFPIT_METER (6)
Length: 8
Meter ID: 20
    
```

Figura 38: Información sobre el paquete OFPT\_FLOW\_MOD

Como en el anterior caso, aunque el identificador del *meter* sea 20, debemos cerciorarnos a través de algún paquete OFPMP\_METER\_CONFIG de que realmente está asociado a un *meter* que descarte los paquetes para lograr una tasa de 20 Mbps. Si este

*meter* no existiera en la lista de *meters* configurados en el switch virtual, la configuración no sería adecuada y no se aplicaría ninguna restricción al ancho de banda del cliente. En la Figura 39 se observar que efectivamente este *meter* está configurado correctamente.

No.	Time	Source	Destination	Protocol	Length	Info
520...	303.0641554...	192.168.0.114	10.0.103.45	OpenF...	84	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP
520...	303.0643302...	10.0.103.45	192.168.0.114	OpenF...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER_CONFIG
520...	303.0652926...	192.168.0.114	10.0.103.45	OpenF...	156	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_CONFIG
520...	303.0683220...	10.0.103.45	192.168.0.114	OpenF...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER

▶ Frame 520632: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits) on interface 0						
▶ Ethernet II, Src: TelnetRe_01:f6:dc (78:3d:5b:01:f6:dc), Dst: IntelCor_e2:42:b2 (90:e2:ba:e2:42:b2)						
▶ Internet Protocol Version 4, Src: 192.168.0.114, Dst: 10.0.103.45						
▶ Transmission Control Protocol, Src Port: 44620, Dst Port: 6633, Seq: 734073, Ack: 49517, Len: 88						
▼ OpenFlow 1.3						
Version: 1.3 (0x04)						
Type: OFPT_MULTIPART_REPLY (19)						
Length: 88						
Transaction ID: 1573						
Type: OFPMP_METER_CONFIG (10)						
▶ Flags: 0x0000						
Pad: 00000000						
▼ Meter config						
Length: 24						
▶ Flags: 0x00000001						
Meter ID: 20						
▼ Meter band						
Type: OFPMBT_DROP (1)						
Length: 16						
Rate: 20000						
Burst size: 20000						
Pad: 00000000						
▶ Meter config						
▶ Meter config						

Figura 39: Información sobre el *meter* de la Figura 38

Este nuevo flujo se mantendrá en la tabla durante 5 minutos hasta que en el segundo 600 desaparezca de la tabla de flujos. Esto es debido al parámetro *hard timeout* que hemos comentado anteriormente. Para comprobar que esto es cierto, vamos a consultar un mensaje de OpenFlow a partir de este segundo en el que el switch virtual mande al controlador su lista de flujos. En la Figura 40 se muestra el flujo original otra vez en la tabla de flujos con el *meter* adecuado (*Meter ID: 10*) y vemos que ha estado activo durante 725 segundos (campo *Duration sec*) ya que no se sobrescribe por otro flujo, simplemente se superpone otro flujo con mayor prioridad.

Posteriormente, para comprobar que la comunicación entre el controlador y el switch virtual es correcta y los parámetros se ajustan a los que nosotros hemos querido enviar, tenemos que comprobar que la tasa de velocidad se ajusta a lo configurado en los *meters* anteriormente. Para ello, vamos a hacer uso de una utilidad instalada dentro de Wireshark llamada *TCP Stream Graphs*. Esta utilidad se encarga de realizar gráficas relacionadas con los flujos de las comunicaciones TCP. Permiten hacer gráficas sobre el número de secuencia de los segmentos TCP, el tiempo de ida y vuelta (*Round Trip Time*), la evolución de la ventana TCP y, el que más nos interesa, el *throughput* de la

comunicación TCP (es decir, la velocidad de la comunicación). Para ello, tenemos que seleccionar un paquete, ir al menú *Statistics->TCP Stream Graphs->Throughput* y automáticamente Wireshark nos generará una gráfica de la comunicación TCP a la que pertenezca dicho paquete. En nuestro caso, se deberá seleccionar un paquete perteneciente a la comunicación TCP que ha generado *iperf* al crear un flujo constante de datos para medir la velocidad del enlace entre la máquina de la oficina central y el cliente. Esta gráfica se puede observar en la Figura 41.

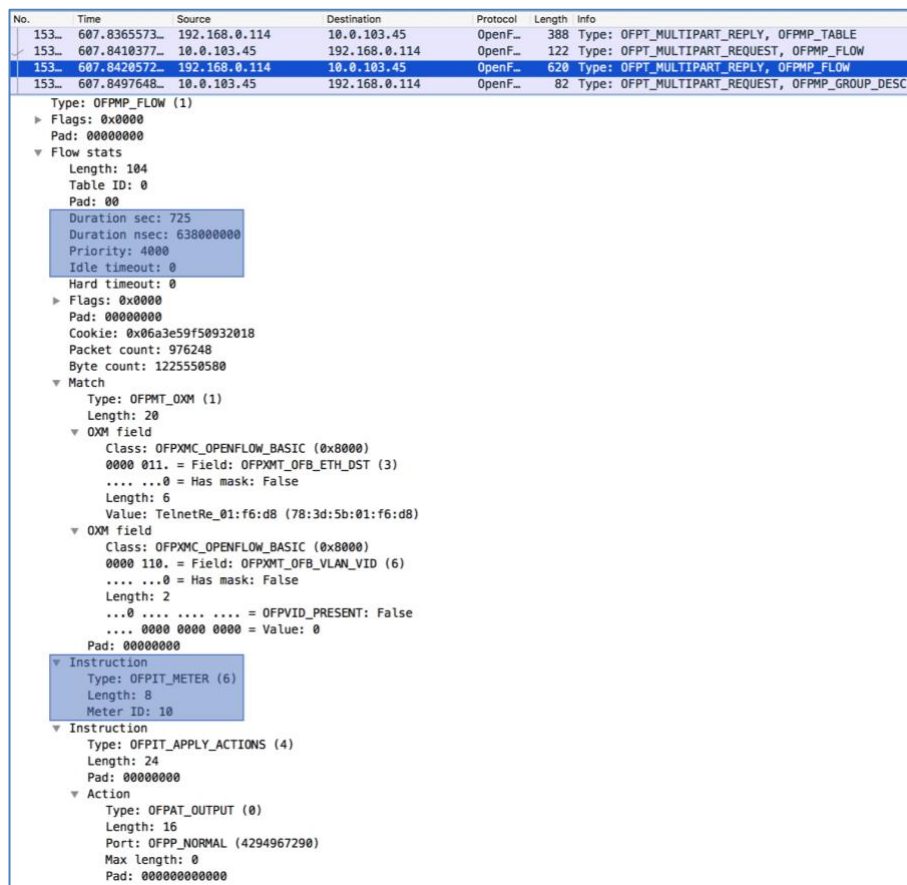


Figura 40: Información sobre el OFPT\_FLOW\_MOD del segundo 600

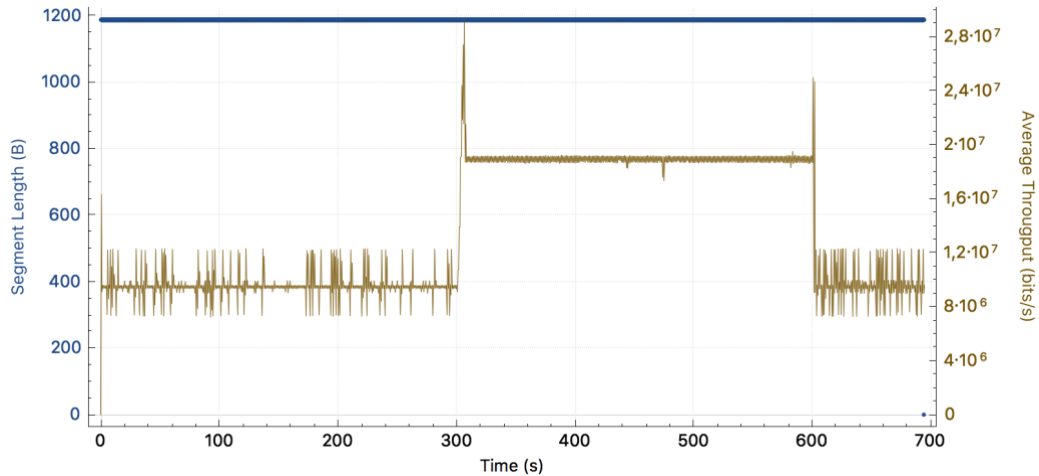


Figura 41: Gráfica del throughput de la conexión TCP en la ONT 1 sentido *upstream*

Se puede observar que la gráfica tiene bastante rizado, sobre todo en el valor de 10 Mbps, aunque en el valor de 20 Mbps también existe, pero de un modo mucho más reducido. Esto es debido a que el switch virtual tiene que ir adaptando la tasa de descarte de paquetes para intentar ajustar la tasa de velocidad a la configurada en el *meter* y la red no tiene una tasa homogénea de paquetes. Aparte de este fenómeno típico en cualquier red de comunicaciones, los *meters* están implementados de manera experimental en OVS, y quizás el comportamiento del switch mejore en futuras versiones. Podemos ver gráficas más detalladas del rizado en las Figuras 42 y 43.

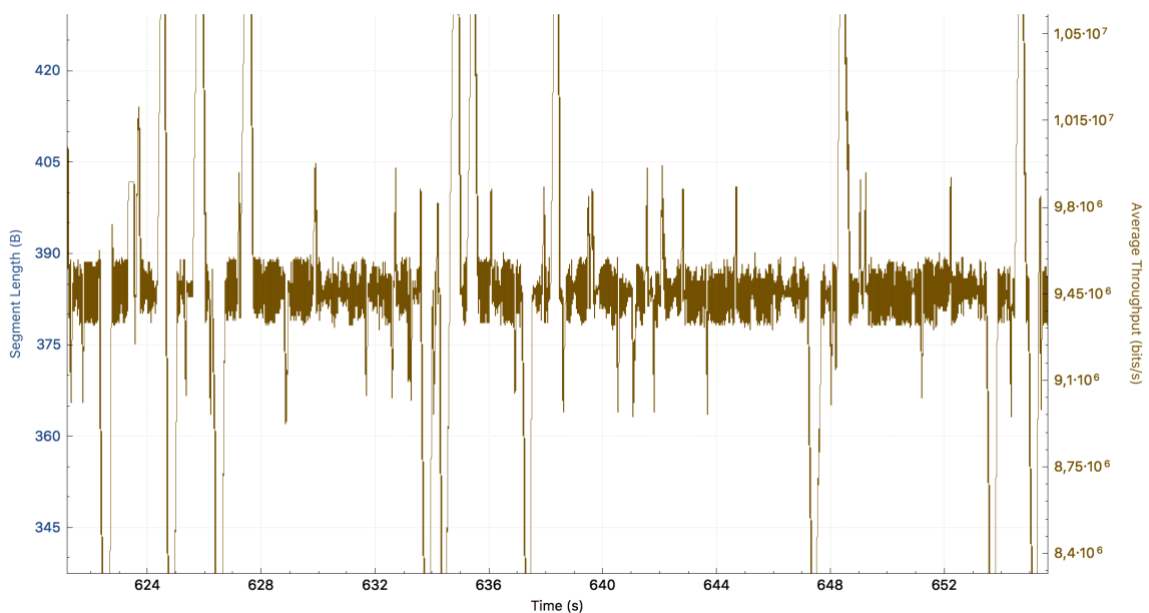


Figura 42: Media de la transmisión configurada a 10 Mbps

Por otra parte, en las transiciones entre los 10 Mbps y los 20 Mbps en los segundos 300 y 600 se puede observar un pico bastante elevado durante unos 4 segundos en el primer salto hasta que el siguiente flujo aplica la restricción necesaria a la velocidad de la conexión, mientras que en el segundo pico la transición es de aproximadamente 2 segundos, como se muestra en las Figuras 42 y 43. Estos picos en las transiciones entre las distintas tasas de velocidad son muy breves con lo cual el cliente no va a notar casi dilación entre la contratación del servicio *premium* y la aplicación del mismo. Estos saltos se pueden ver más detalladamente en las Figuras 44 y 45.

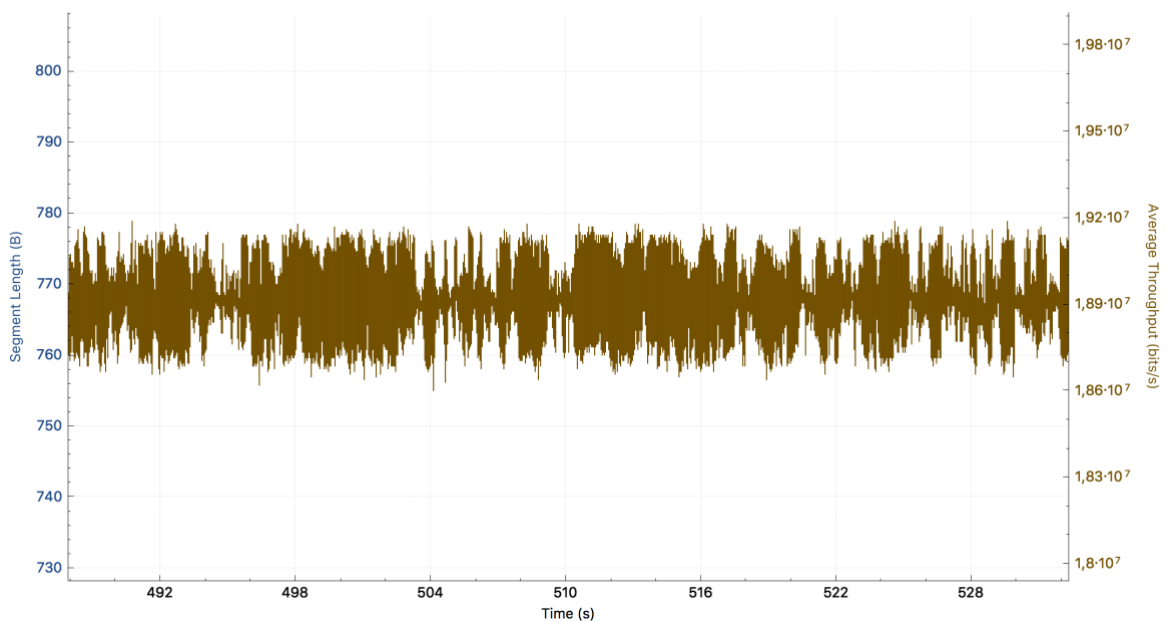


Figura 43: Media de la transmisión configurada a 20 Mbps

También podemos observar que, de media, la transmisión es un poco más baja de lo deseado, del orden de entre un 5-10% más baja. En el caso de 10 Mbps la conexión es de 9,5 Mbps y en el caso de 20 Mbps es de 19 Mbps ( $1,9 \cdot 10^7$  bits/s en la Figura 43), como se puede ver en las Figuras 42 y 43. Aunque no es exactamente la configuración deseada, podemos solucionarlo fácilmente configurando los *meters* siempre con un 10% de margen positivo para que el cliente tenga un poco más de la velocidad contratada y así garantizar siempre la QoS.

Este fenómeno es debido a que el *meter* no es capaz de mantener una media adecuada de tasa de descarte de paquetes para limitar bien la velocidad de transmisión. Es más, los *meter* se ajustan con diferentes tasas de error dependiendo de la versión de OVS.



En la versión más nueva de OVS (2.9.2) los *meters* se aplican con un 20-25% de error mientras que en la versión que utilizamos en nuestro proyecto (2.8.4) se aplican con un 5-10% de error, una tasa muy inferior a la ofrecida por la nueva versión. En este sentido, suponemos que al ser una característica experimental de OpenFlow implementada todavía de manera provisional en OVS, en futuras versiones del switch virtual irá teniendo niveles de error menores.

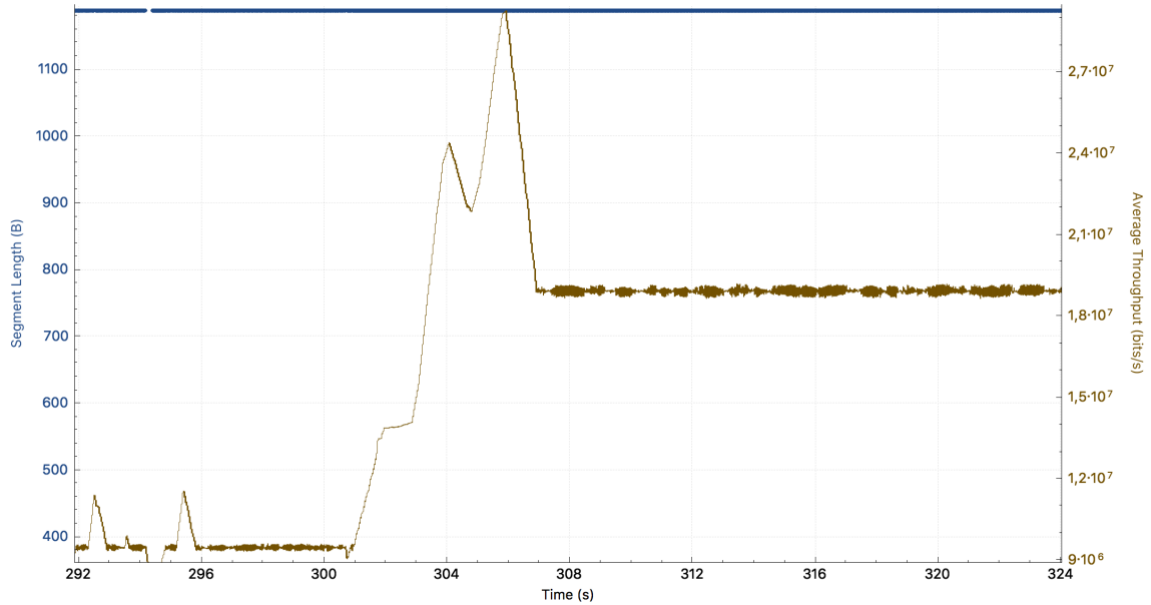


Figura 44: Pico en el segundo 300 de la comunicación en la ONT 1 sentido *upstream*

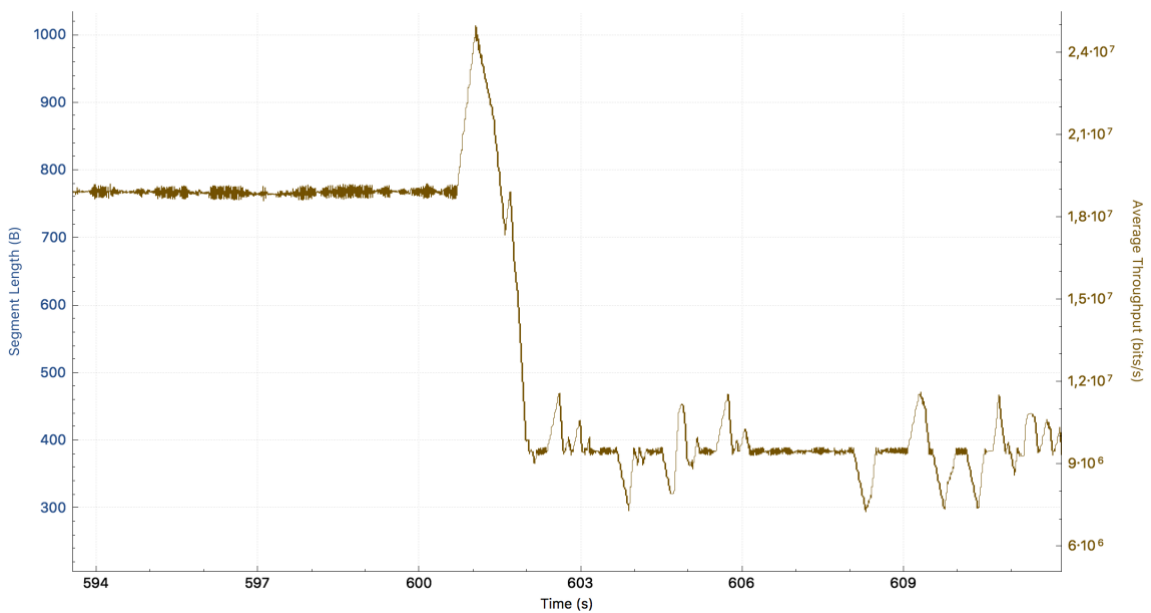


Figura 45: Pico en el segundo 600 de la comunicación en la ONT 1 sentido *upstream*

A continuación, se va a analizar el flujo *downstream* de la ONT 1, para poder satisfacer también el cambio de ancho de banda de 10 a 20 Mbps también en sentido de descarga para el cliente. Los mensajes OpenFlow irán dirigidos desde el controlador OpenDayLight (10.0.103.45/24) hacia el switch virtual situado en el ordenador de la oficina central (10.0.103.48/24) que recordemos es el switch que se encarga de controlar los flujos en sentido *downstream*. Para analizar el flujo descendente vamos a utilizar los mismos mecanismos que en el sentido *upstream*, pero tendremos que observar los mensajes del switch virtual de la oficina central.

En primer lugar, se va a analizar un mensaje OFPMP\_FLOW para ver las características del flujo por defecto, que corresponde con el ancho de banda por defecto contratado por el usuario. Podemos ver que tiene una prioridad de 3000 y no tiene asignado un *hard timeout* como nuestro flujo por defecto de subida. Como En el campo *match* se asigna la dirección MAC del lado WAN de la ONT 1 (78:3d:5b:01:f6:dc) y en el campo *instructions* tenemos dos instrucciones, una con una acción normal (OFPIT\_APPLY\_ACTIONS, OFPAT\_OUTPUT) y otra con un *meter* asociado a 10 Mbps (OFPIT\_METER). Este mensaje se puede observar en la Figura 46.

El flujo anterior tenía una prioridad de 4000 para facilitar la identificación a nivel interno de los flujos en el laboratorio, pero no tiene una razón de ser técnica para la realización del proyecto como tal. Simplemente tiene que ser razonablemente mayor que 0 y luego ser sustituida por un valor más grande en el flujo que representa al ancho de banda *premium* solicitado por el usuario.

No.	Time	Source	Destination	Protocol	Length	Info
780	0.298749943	10.0.103.48	10.0.103.45	OpenF...	386	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
782	0.302931395	10.0.103.45	10.0.103.48	OpenF...	122	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW
783	0.310288520	10.0.103.48	10.0.103.45	OpenF...	610	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
784	0.316294149	10.0.103.45	10.0.103.48	OpenF...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC

```

▶ Internet Protocol Version 4, Src: 10.0.103.48, Dst: 10.0.103.45
▶ Transmission Control Protocol, Src Port: 44418, Dst Port: 6633, Seq: 6113, Ack: 73, Len: 544
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 544
  Transaction ID: 17691
  Type: OFPMP_FLOW (1)
  ▶ Flags: 0x0000
  Pad: 00000000
  ▼ Flow stats
    Length: 96
    Table ID: 0
    Pad: 00
    Duration sec: 1415
    Duration nsec: 954000000
    Priority: 3000
    Idle timeout: 0
    Hard timeout: 0
    ▶ Flags: 0x0000
    Pad: 00000000
    Cookie: 0x0c8200d4f86b6154
    Packet count: 638046
    Byte count: 47995789
  ▼ Match
    Type: OFPMT_OXM (1)
    Length: 14
    ▼ OXM field
      Class: OFPXM_OPENFLOW_BASIC (0x0000)
      0000 011. = Field: OFPXM_OFB_ETH_DST (3)
      ... ..0 = Has mask: False
      Length: 6
      Value: TelnetRe_01:f6:dc (78:3d:5b:01:f6:dc)
      Pad: 0000
  ▼ Instruction
    Type: OFPIT_METER (6)
    Length: 8
    Meter ID: 10
  ▼ Instruction
    Type: OFPIT_APPLY_ACTIONS (4)
    Length: 24
    Pad: 00000000
    ▼ Action
      Type: OFPAT_OUTPUT (0)
      Length: 16
      Port: OFPP_NORMAL (4294967290)
      Max length: 0
      Pad: 000000000000
  
```

Figura 46: Mensaje OFPMP\_FLOW en el switch virtual de la Oficina Central

En el segundo 300 el cliente modifica la tasa de velocidad como hemos visto en la Figura 37 y por eso podemos observar también un mensaje de OFPT\_FLOW\_MOD en el que se añade un flujo con una prioridad superior (*Priority: 6000*) al flujo por defecto, un *hard timeout* de 300 segundos y un *meter* con ID 20 asociado a un *meter* de 20 Mbps.

No.	Time	Source	Destination	Protocol	Length	Info
502...	300.1589294...	10.0.103.48	10.0.103.45	OpenF...	82	Type: OFPT_MULTIPART_REPLY, OFPMP_QUEUE
504...	300.3098429...	10.0.103.45	10.0.103.48	OpenF...	162	Type: OFPT_FLOW_MOD
504...	300.6451643...	10.0.103.45	10.0.103.48	OpenF...	74	Type: OFPT_BARRIER_REQUEST
504...	300.6452238...	10.0.103.48	10.0.103.45	OpenF...	74	Type: OFPT_BARRIER_REPLY

```

Frame 504033: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface 1
Ethernet II, Src: PcsCompu_57:8e:c2 (08:00:27:57:8e:c2), Dst: IntelCor_e2:42:5e (90:e2:ba:e2:42:5e)
Internet Protocol Version 4, Src: 10.0.103.45, Dst: 10.0.103.48
Transmission Control Protocol, Src Port: 6633, Dst Port: 49266, Seq: 50105, Ack: 783505, Len: 96
OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FLOW_MOD (14)
  Length: 96
  Transaction ID: 18906
  Cookie: 0x0c8200d4f86b6154
  Cookie mask: 0x0000000000000000
  Table ID: 0
  Command: OFPFC_ADD (0)
  Idle timeout: 0
  Hard timeout: 300
  Priority: 6000
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Out port: OFPP_ANY (4294967295)
  Out group: OFPG_ANY (4294967295)
  Flags: 0x0000
  Pad: 0000
  Match
    Type: OFPMT_OXM (1)
    Length: 14
    OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
      .... ..0 = Has mask: False
      Length: 6
      Value: TelnetRe_01:f6:dc (78:3d:5b:01:f6:dc)
      Pad: 0000
    Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 24
      Pad: 00000000
      Action
        Type: OFPAT_OUTPUT (0)
        Length: 16
        Port: OFPP_NORMAL (4294967290)
        Max length: 0
        Pad: 000000000000
      Instruction
        Type: OFPIT_METER (6)
        Length: 8
        Meter ID: 20
  
```

Figura 47: Mensaje OFPT\_FLOW\_MOD en el switch de la oficina central

Posteriormente en el segundo 600 volveremos a tener solo el flujo por defecto y la velocidad del cliente volverá a ser la que tenía contratada por defecto, es decir, 10 Mbps. Después de analizar que los flujos se envían y se reciben correctamente, vamos a utilizar la herramienta *Statistics->TCP Stream Graphs->Throughput* para poder observar en una gráfica la tasa de velocidad de nuestra conexión en sentido de bajada. La Figura 48 muestra la gráfica de la velocidad de la conexión. En este caso, la gráfica tiene muchísimo más rizado que en el sentido *upstream*, tanto en el tramo de 10 Mbps como en el tramo de 20 Mbps. Como hemos contado anteriormente, el switch virtual tiene que ajustar la tasa de descarte de paquetes. El switch virtual situado en la Oficina Central tiene dos flujos a los que ir aplicando las acciones (el flujo *downstream* de la ONT 1 y el flujo *downstream* de la ONT 2) y esto hace que la tasa tenga más rizado y sea menos constante. Suponemos,

como hemos dicho en el sentido *upstream*, que en futuras versiones de OVS los *meters* se comportarán de una manera más estable. Podemos ver gráficas más detalladas del rizado en las Figuras 49 y 50.

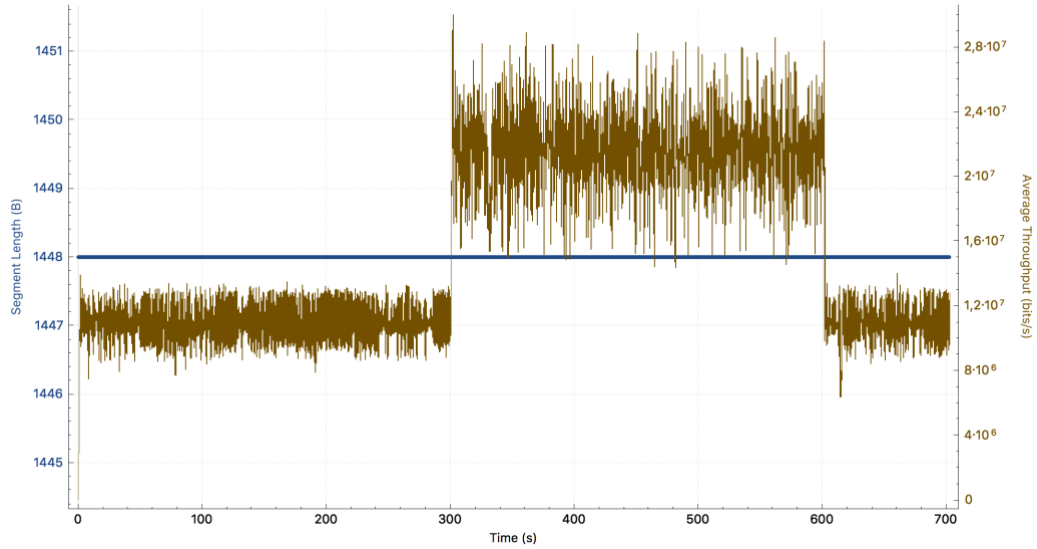


Figura 48: Gráfica del throughput de la conexión TCP en la ONT 1 sentido *downstream*

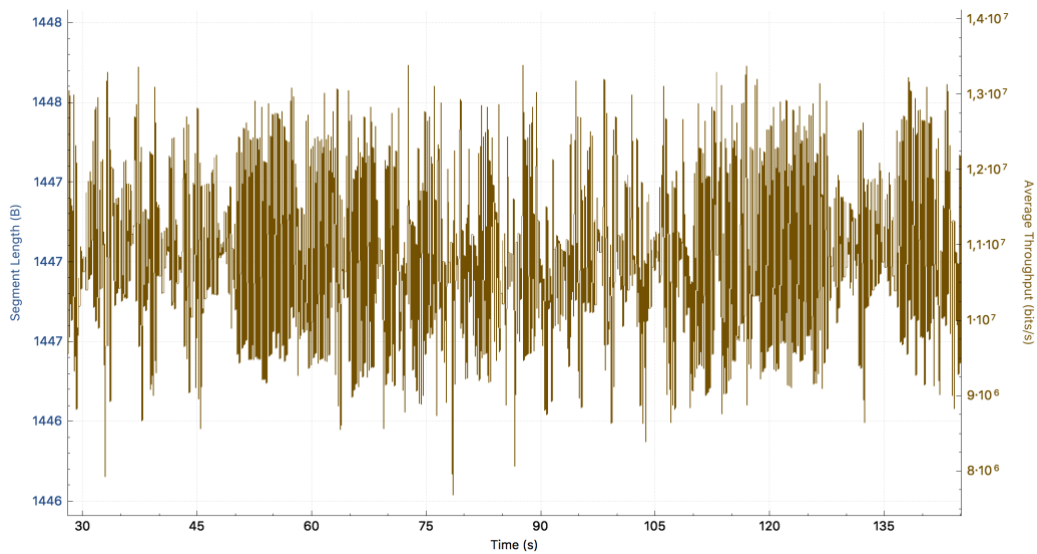


Figura 49: Gráfica del rizado detallado de 10 Mbps en la ONT 1 en sentido *downstream*

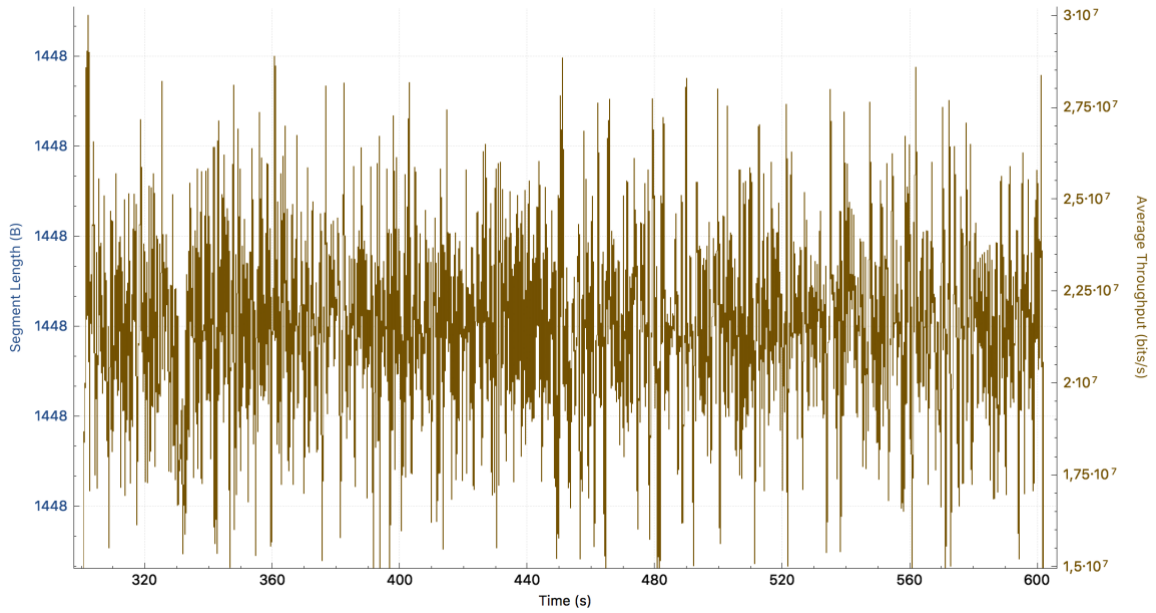


Figura 50: Gráfica del rizado detallado de 20 Mbps en la ONT 1 en sentido *downstream*

Por otra parte, en las transiciones entre los 10 Mbps y los 20 Mbps en los segundos 300 y 600 se observan picos (como en el *upstream*) pero duran menos de dos segundos y son menos bruscos. Creemos que esto es debido a que el switch virtual de la Oficina Central está situado más cerca del controlador OpenDayLight y por tanto los mensajes pueden tener un poco menos de latencia y el switch virtual podrá empezar a aplicar la tasa más rápido. Estos picos se muestran en la Figura 51 y 52.

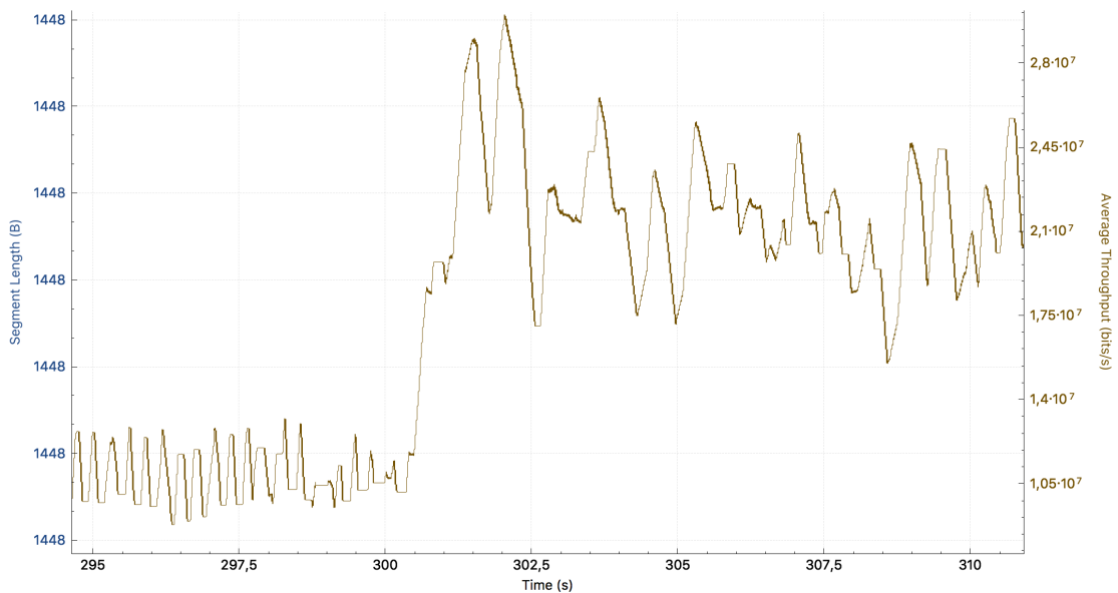


Figura 51: Pico en el segundo 300 de la comunicación en la ONT 1 sentido *downstream*

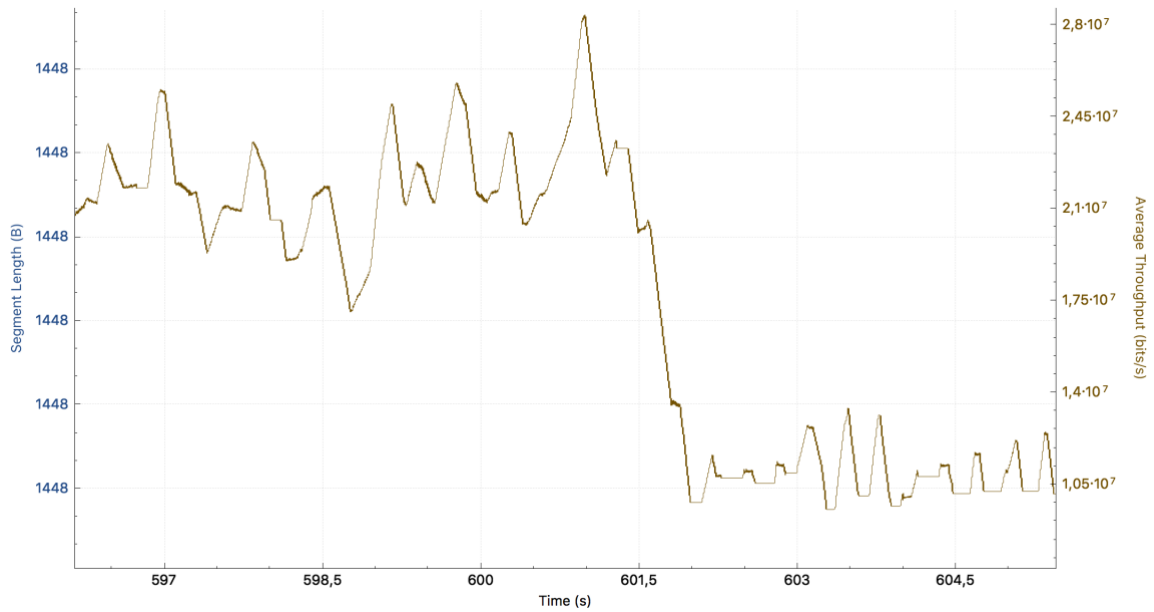


Figura 52: Pico en el segundo 300 de la comunicación en la ONT 1 sentido *downstream*

#### 4.5.2 Análisis de resultados en la ONT 2

De manera análoga a lo propuesto en el Apartado 4.5.1, vamos a monitorizar los distintos mensajes OpenFlow que se envían en la red, primero entre el switch virtual situado detrás de la ONT 2 (configurada con 20 Mbps por defecto y un aumento a 35 Mbps) y en el controlador para analizar el sentido *upstream*. El controlador va a seguir estando situado en la dirección IP 10.0.103.45/24 y los mensajes del switch virtual serán enviados desde 192.168.0.104/24 (dirección IP de la ONT 2, ya que el switch virtual tendrá una dirección tipo 192.168.2.x/24).

Primero, vamos a observar los flujos que tiene el switch virtual por defecto, que deberían tener los campos *match* e *instructions* deseados para la configuración por defecto de nuestra red. El campo *match* tendrá una dirección de capa de enlace como en el Apartado 4.5.1, pero con la dirección adecuada de la interfaz del lado WAN de la ONT 2 (78:3d:5b:01:f7:30). En la Figura 53 se muestra que en el mensaje OFPMP\_FLOW que el flujo es el adecuado, mientras que en la Figura 54 se observa que efectivamente existe un *meter* con un ID 20 (*Meter ID: 20*) en el switch virtual, con las características de *Band-rate* y *Drop-rate* necesarias, esto es 20000 Kbps.

No.	Time	Source	Destination	Protocol	Length	Info
7948	2.852103299	10.0.103.45	192.168.0.104	OpenF...	122	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW
7954	2.853174277	192.168.0.104	10.0.103.45	OpenF...	612	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
8013	2.863457909	10.0.103.45	192.168.0.104	OpenF...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC
8019	2.864304339	192.168.0.104	10.0.103.45	OpenF...	84	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP_DESC

```

Version: 1.3 (0x04)
Type: OFPT_MULTIPART_REPLY (19)
Length: 544
Transaction ID: 4653
Type: OFPMP_FLOW (1)
Flags: 0x0000
Pad: 00000000
Flow stats
Length: 96
Table ID: 0
Pad: 00
Duration sec: 1248
Duration nsec: 476000000
Priority: 2000
Idle timeout: 0
Hard timeout: 0
Flags: 0x0000
Pad: 00000000
Cookie: 0x06a3e59f50932018
Packet count: 95569
Byte count: 80345954
Match
Type: OFPMT_OXM (1)
Length: 14
OXM field
Class: OFPXM_OPENFLOW_BASIC (0x8000)
0000 011. = Field: OFPXM_OFB_ETH_DST (3)
... ..0 = Has mask: False
Length: 6
Value: TelnetRe_01:f7:30 (78:3d:5b:01:f7:30)
Pad: 0000
Instruction
Type: OFPIT_METER (6)
Length: 8
Meter ID: 20
Instruction
Type: OFPIT_APPLY_ACTIONS (4)
Length: 24
Pad: 00000000
Action
Type: OFPAT_OUTPUT (0)
Length: 16
Port: OFPP_NORMAL (4294967290)
Max Length: 0
Pad: 000000000000
    
```

Figura 53: Información sobre el paquete OFPMP\_FLOW del flujo por defecto de 20 Mbps

No.	Time	Source	Destination	Protocol	Length	Info
8027	2.865412082	192.168.0.104	10.0.103.45	OpenF...	84	Type: OFPT_MULTIPART_REPLY, OFPMP_GROUP
8028	2.865523726	10.0.103.45	192.168.0.104	OpenF...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER_CONFIG
8033	2.866400497	192.168.0.104	10.0.103.45	OpenF...	396	Type: OFPT_MULTIPART_REPLY, OFPMP_METER_CONFIG
8111	2.880033060	10.0.103.45	192.168.0.104	OpenF...	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER

```

Frame 8033: 396 bytes on wire (3168 bits), 396 bytes captured (3168 bits) on interface 0
Ethernet II, Src: TelnetRe_01:f7:34 (78:3d:5b:01:f7:34), Dst: IntelCor_e2:42:b2 (90:e2:ba:e2:42:b2)
Internet Protocol Version 4, Src: 192.168.0.104, Dst: 10.0.103.45
Transmission Control Protocol, Src Port: 59574, Dst Port: 6633, Seq: 6705, Ack: 605, Len: 328
OpenFlow 1.3
Version: 1.3 (0x04)
Type: OFPT_MULTIPART_REPLY (19)
Length: 328
Transaction ID: 4656
Type: OFPMP_METER_CONFIG (10)
Flags: 0x0000
Pad: 00000000
Meter config
Meter config
Meter config
Meter config
Meter config
Length: 24
Flags: 0x00000001
Meter ID: 20
Meter band
Type: OFPMBT_DROP (1)
Length: 16
Rate: 20000
Burst size: 20000
Pad: 00000000
    
```

Figura 54: Información sobre el meter de la Figura 53

Este flujo se mantendrá durante 5 minutos hasta que en el segundo 300 el cliente solicita un servicio *premium* para que su ancho de banda aumente a 35 Mbps. Esto hace que aparezca un mensaje OFPT\_FLOW\_MOD que envía el controlador ODL al switch virtual para añadir un flujo con una prioridad mayor. Este flujo es el que está asociado al





A los 600 segundos de comunicación el flujo que representa la petición *premium* del cliente se borra automáticamente de la tabla de flujos del switch virtual y pasa a aplicarse el flujo del servicio por defecto del cliente (20 Mbps).

A continuación, se va a analizar la velocidad durante esta prueba experimental con la misma utilidad que en el Apartado anterior (*TCP Stream Graph*), viendo cómo se comporta la tasa de transmisión (*throughput*) durante los 12 minutos de prueba. En la Figura 57 podemos ver la gráfica durante los casi 700 segundos de comunicación.

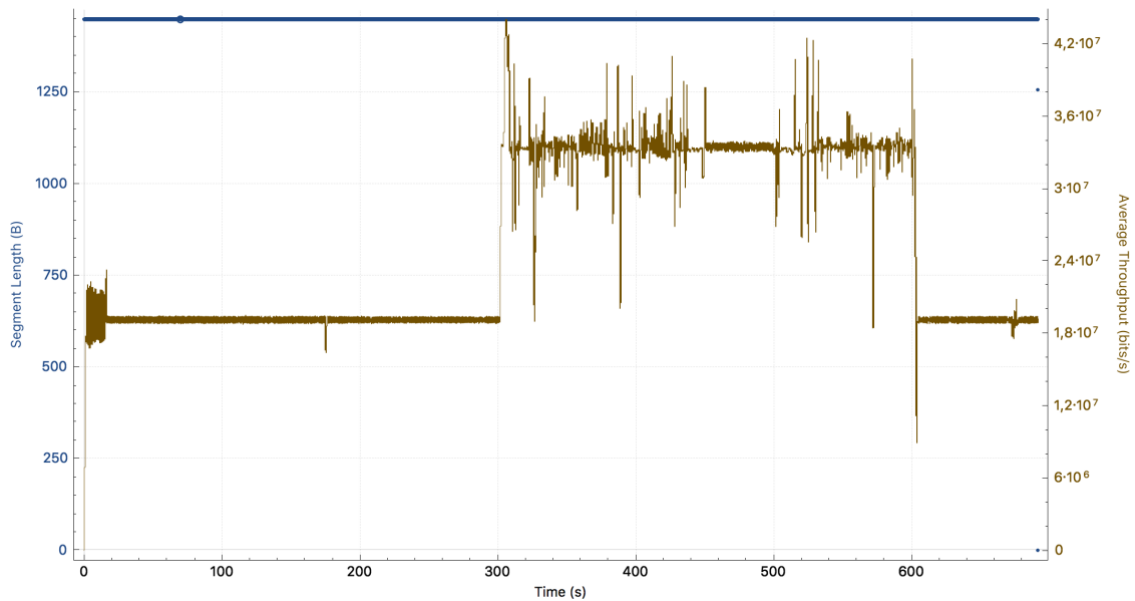


Figura 57: Gráfica del *throughput* en la conexión TCP de la ONT 2 en sentido *upstream*

Podemos observar que el rizado es muy alto cuando la conexión está configurada a 35 Mbps. Esto es debido a que el switch virtual intenta corregir la tasa cuando la conexión baja por razones ajenas al propio switch (recordemos que el límite en el que la conexión es aproximadamente estable es  $\sim 35$  Mbps). Para corregir estos vaivenes, el switch reduce su tasa de descarte de paquetes en periodos muy cortos de tiempo, y cuando esta tasa varía en instantes cortos se producen picos muy bruscos (como los producidos en el mensaje `OFPT_FLOW_MOD`). El rizado de la conexión cuando está configurada a 20 Mbps podemos ver que es muy reducido, aunque también la red tiene una ligera caída en el segundo 170 aproximadamente. En la Figura 58 y en la Figura 59 podemos ver una parte reducida de la gráfica con la media efectiva de la velocidad de la conexión.

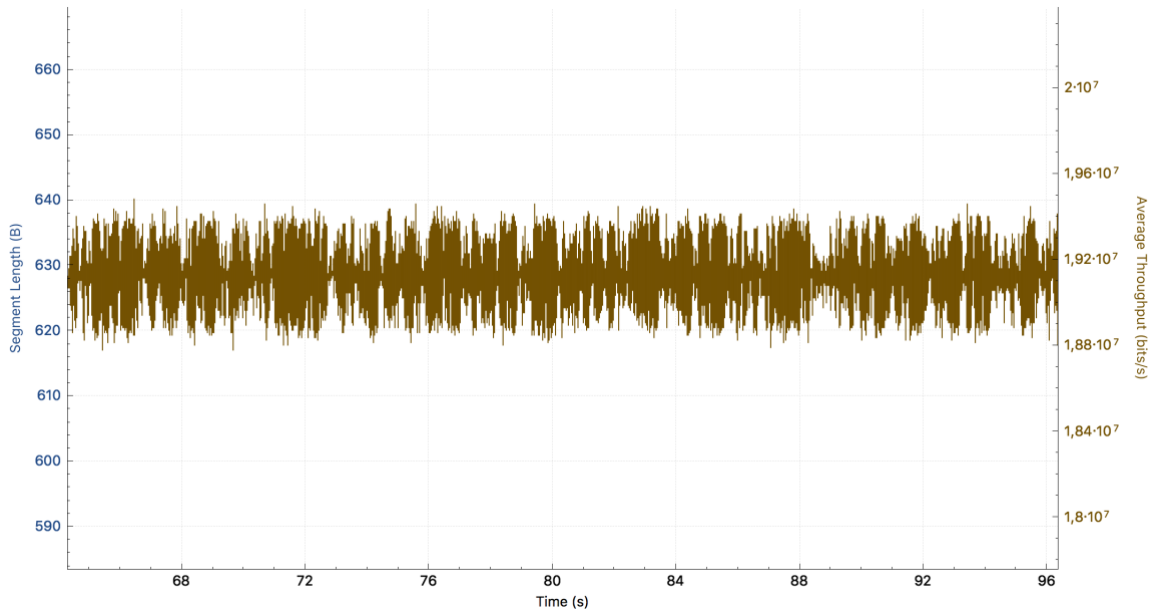


Figura 58: Media de la transmisión configurada a 20 Mbps

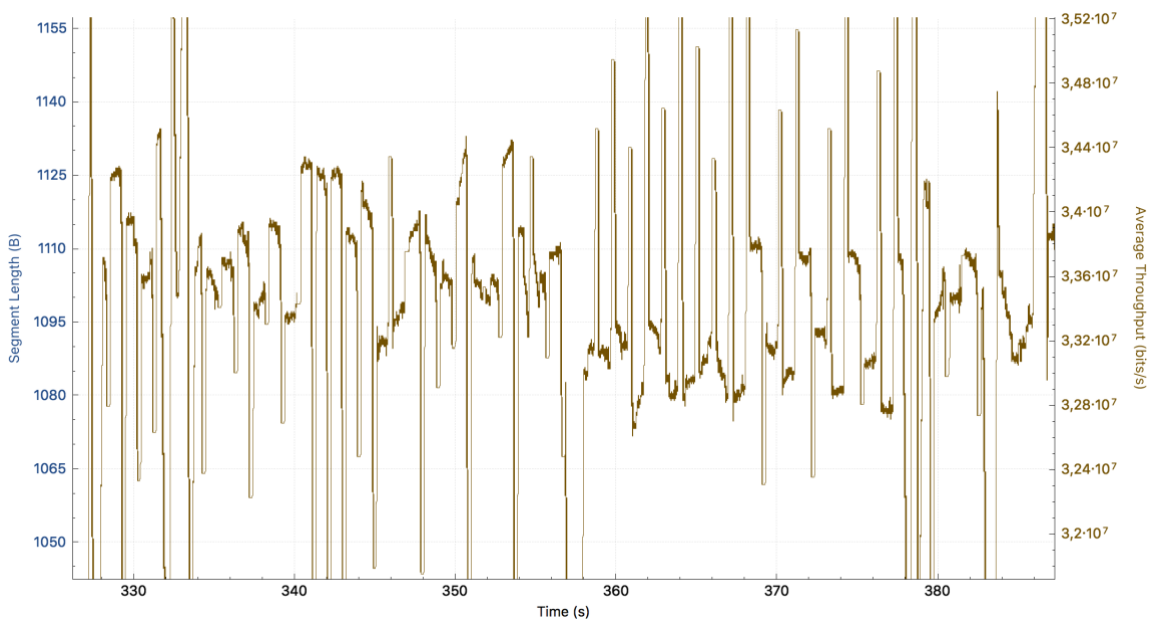


Figura 59: Media de la transmisión configurada a 35 Mbps

Vemos que la media de la transmisión sigue siendo en torno a un 5-10% más baja de lo configurado en el *meter* como ocurría en la otra ONT y además podemos ver de cerca en la Figura 59 que la velocidad de la transmisión es cambiante y un poco inestable, con lo cual algunas aplicaciones que requieren poca latencia y poco *jitter* quizás podrían verse afectadas en algunos momentos. Respecto a los picos que se producen cuando se mandan los mensajes OFPT\_FLOW\_MOD en los segundos 300 y 600, tienen aproximadamente la

misma duración que en los anteriores apartados como podemos ver en las Figuras 60 y 61. En concreto, los dos picos del flujo de la ONT 2 en el flujo de *upstream* tienen una magnitud menor que en la prueba anterior y dura aproximadamente 2 segundos hasta que el siguiente flujo empieza a funcionar. Al ser la conexión de 35 Mbps bastante inestable es más difícil saber cuándo cambia realmente sin observar los mensajes y las tablas del protocolo OpenFlow 1.3.

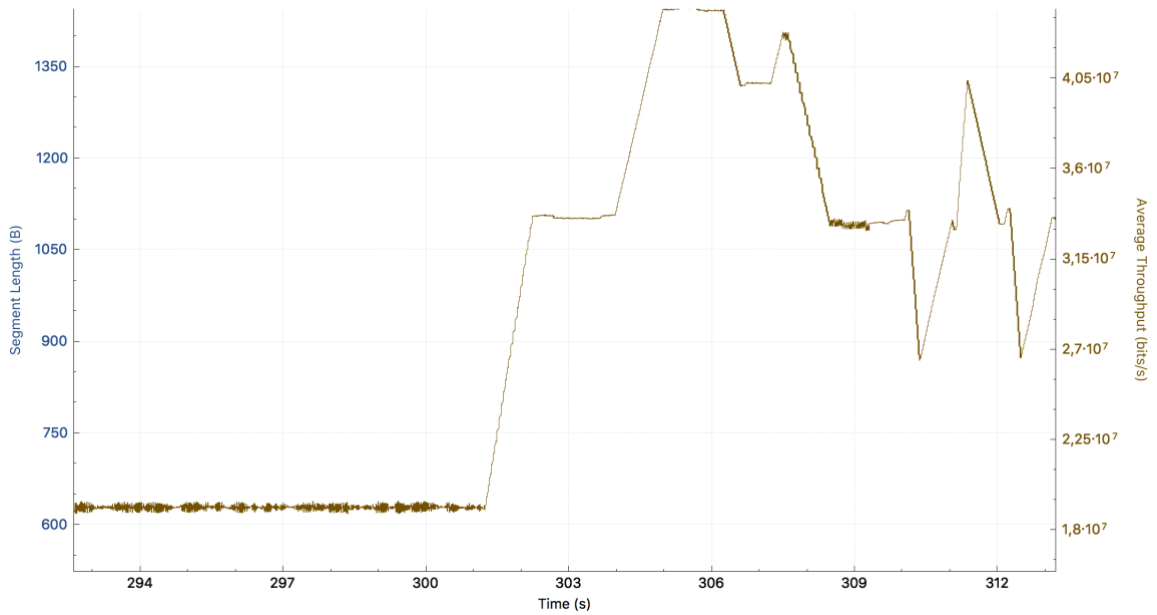


Figura 60: Pico en el segundo 300 de la comunicación de la ONT 2

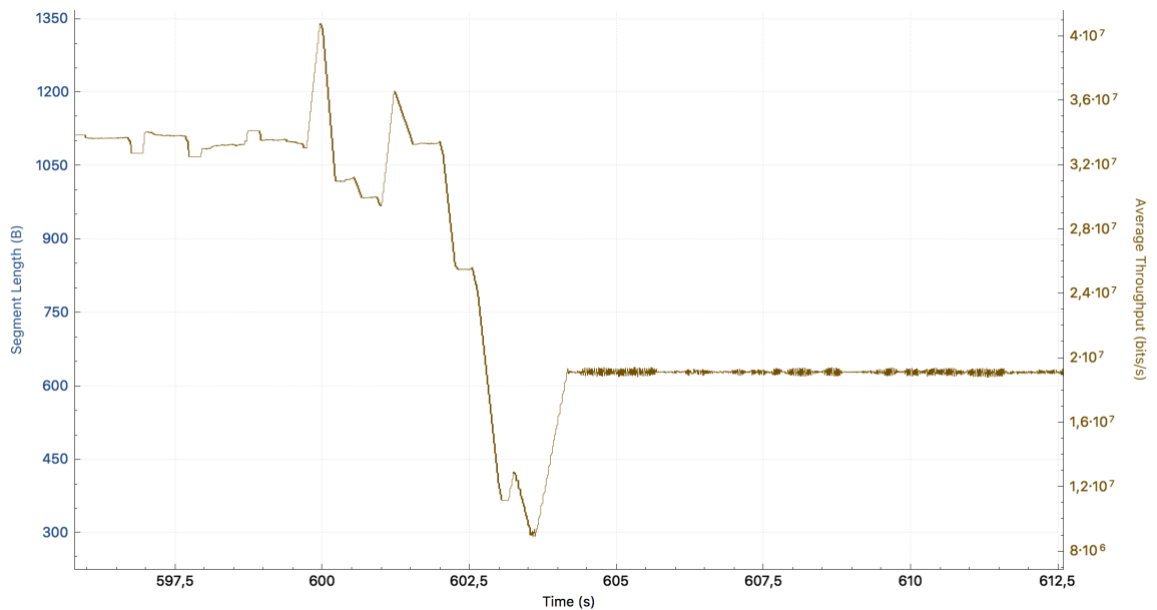


Figura 61: Pico en el segundo 300 de la comunicación de la ONT 2

Ahora vamos a pasar a analizar la comunicación *downstream* de esta ONT. Es necesario también garantizar el ancho de banda *premium* al cliente en los dos sentidos, por eso necesitamos cerciorarnos de que todos los mensajes se envían correctamente y están bien configurados. En este caso, los mensajes OpenFlow irán dirigidos desde el controlador OpenDayLight (10.0.103.45/24) hacia el switch virtual situado en el ordenador de la Oficina Central (10.0.103.48/24) que recordemos es el switch que se encarga de controlar los flujos en sentido *downstream*. Para analizar el flujo descendente vamos a utilizar los mismos mecanismos que en el sentido *upstream*, pero tendremos que observar los mensajes del switch virtual de la Oficina Central.

Primero vamos a analizar la tabla de flujos del switch virtual de la Oficina Central para ver si el flujo por defecto está configurado correctamente. Deberá tener un campo *match* con la dirección MAC del lado WAN de la ONT 2 (78:3d:5b:01:f7:34), además de unos campos *Instructions* con un comportamiento normal y otra con un *meter* con ID 20 asociado a una limitación a 20 Mbps. Como podemos ver en la Figura 62, este flujo está correctamente configurado.

No.	Time	Source	Destination	Protocol	Length	Info
522...	16.753645990	10.0.103.48	10.0.103.45	OpenF...	386	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
522...	16.759943098	10.0.103.45	10.0.103.48	OpenF...	122	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW
522...	16.760539846	10.0.103.48	10.0.103.45	OpenF...	706	Type: OFPT_MULTIPART_REPLY, OFPMP_FLOW
522...	16.766901262	10.0.103.45	10.0.103.48	OpenF...	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC

Internet Protocol Version 4, Src: 10.0.103.48, Dst: 10.0.103.45
Transmission Control Protocol, Src Port: 49266, Dst Port: 6633, Seq: 45289, Ack: 2579, Len: 640
OpenFlow 1.3
Version: 1.3 (0x04)
Type: OFPT_MULTIPART_REPLY (19)
Length: 640
Transaction ID: 46293
Type: OFPMP_FLOW (1)
Flags: 0x0000
Pad: 00000000
Flow stats
Length: 96
Table ID: 0
Pad: 00
Duration sec: 12441
Duration nsec: 583000000
Priority: 3000
Idle timeout: 0
Hard timeout: 0
Flags: 0x0000
Pad: 00000000
Cookie: 0x0c8200d4f86b6154
Packet count: 262875
Byte count: 295981585
Match
Type: OFPMT_OXM (1)
Length: 14
OXM field
Class: OFPXMC_OPENFLOW_BASIC (0x8000)
0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
... ..0 = Has mask: False
Length: 6
Value: TelnetRe_01:f7:34 (78:3d:5b:01:f7:34)
Pad: 0000
Instruction
Type: OFPIT_METER (6)
Length: 8
Meter ID: 20
Instruction
Type: OFPIT_APPLY_ACTIONS (4)
Length: 24
Pad: 00000000
Action
Type: OFPAT_OUTPUT (0)
Length: 16
Port: OFPP_NORMAL (4294967290)
Max length: 0
Pad: 000000000000

Figura 62: Flujo por defecto en la ONT 2 en sentido *downstream*

Posteriormente, vamos a observar si en el segundo 300, cuando el cliente hace una petición de servicio *premium* de 35 Mbps durante 5 minutos, se crea un flujo con una prioridad mayor que la del flujo por defecto, con una caducidad (*hard timeout*) de 5 minutos y con un *meter* adecuado a lo pedido por el cliente (35 Mbps). Como podemos observar en la Figura 63 se crea un flujo con una prioridad de 5000 (el flujo por defecto tenía 3000), un *hard timeout* de 300 segundos, unas características de *match* idénticas a la del flujo por defecto y un *meter* asociado a una velocidad de 35 Mbps como el de la Figura 56.

No.	Time	Source	Destination	Protocol	Length	Info
951...	300.5543863...	10.0.103.45	10.0.103.48	OpenF...	74	Type: OFPT_BARRIER_REQUEST
951...	300.5560850...	10.0.103.48	10.0.103.45	OpenF...	74	Type: OFPT_BARRIER_REPLY
952...	300.7973029...	10.0.103.45	10.0.103.48	OpenF...	258	Type: OFPT_FLOW_MOD
953...	300.9516058...	10.0.103.45	10.0.103.48	OpenF...	74	Type: OFPT_BARRIER_REQUEST

▶ Frame 952730: 258 bytes on wire (2064 bits), 258 bytes captured (2064 bits) on interface 1  
 ▶ Ethernet II, Src: PcsCompu\_57:8e:c2 (08:00:27:57:8e:c2), Dst: IntelCor\_e2:42:5e (90:e2:ba:e2:42:5e)  
 ▶ Internet Protocol Version 4, Src: 10.0.103.45, Dst: 10.0.103.48  
 ▶ Transmission Control Protocol, Src Port: 6633, Dst Port: 49266, Seq: 50105, Ack: 783505, Len: 192  
 ▶ OpenFlow 1.3  
 ▼ OpenFlow 1.3  
   Version: 1.3 (0x04)  
   Type: OFPT\_FLOW\_MOD (14)  
   Length: 96  
   Transaction ID: 47355  
   Cookie: 0x0c8200d4f86b6154  
   Cookie mask: 0x0000000000000000  
   Table ID: 0  
   Command: OFPFC\_ADD (0)  
   Idle timeout: 0  
   Hard timeout: 300  
   Priority: 5000  
   Buffer ID: OFP\_NO\_BUFFER (4294967295)  
   Out port: OFPP\_ANY (4294967295)  
   Out group: OFPG\_ANY (4294967295)  
 ▶ Flags: 0x0000  
   Pad: 0000  
 ▼ Match  
   Type: OFPMT\_OXM (1)  
   Length: 14  
   ▼ OXM field  
     Class: OFPXM\_OPENFLOW\_BASIC (0x8000)  
     0000 011. = Field: OFPXM\_OFB\_ETH\_DST (3)  
     ... ..0 = Has mask: False  
     Length: 6  
     Value: TelnetRe\_01:f7:34 (78:3d:5b:01:f7:34)  
     Pad: 0000  
 ▼ Instruction  
   Type: OFPIT\_APPLY\_ACTIONS (4)  
   Length: 24  
   Pad: 00000000  
   ▼ Action  
     Type: OFPAT\_OUTPUT (0)  
     Length: 16  
     Port: OFPP\_NORMAL (4294967290)  
     Max length: 0  
     Pad: 000000000000  
   ▼ Instruction  
     Type: OFPIT\_METER (6)  
     Length: 8  
     Meter ID: 35

Figura 63: Flujo *premium* en la ONT 2 en sentido *downstream*

Después de analizar los mensajes OpenFlow, ahora vamos a analizar las gráficas de tasa de velocidad de la comunicación, usando las herramientas de Wireshark anteriormente citadas. En la Figura 64 vemos una gráfica de la velocidad durante los 700 segundos de comunicación.

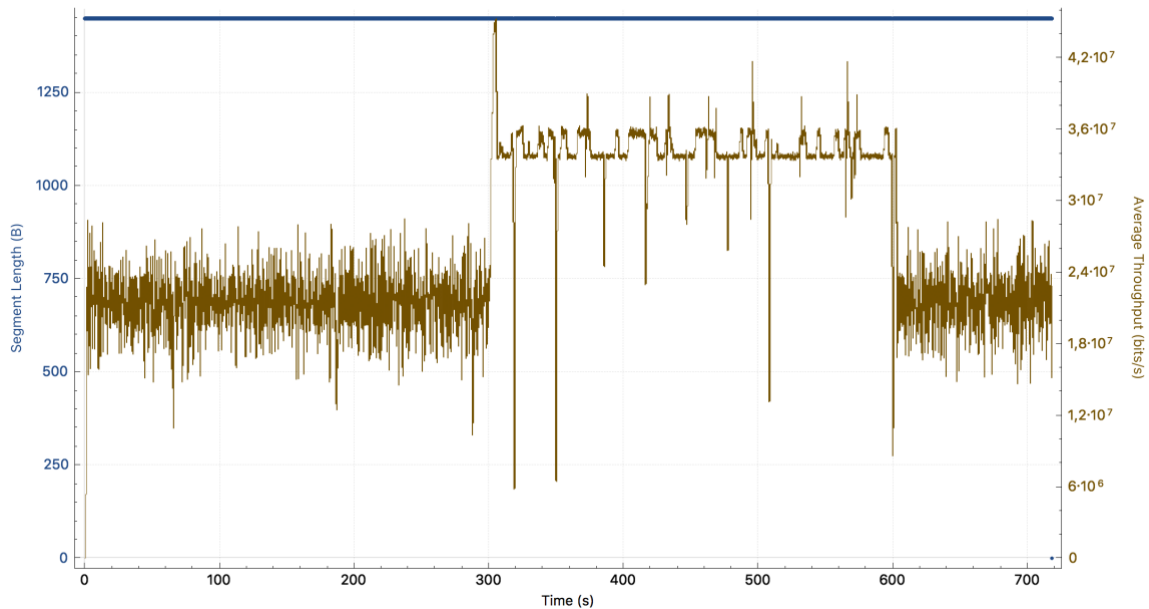


Figura 64: Gráfica del *throughput* en la conexión TCP de la ONT 2 en sentido *downstream*

Podemos observar que hay bastante rizado en los tramos de comunicación en los que la velocidad es de 20 Mbps y que el rizado es menor en las fases de 35 Mbps, pero tiene varias caídas en la velocidad de 1 a 2 segundos durante la comunicación. Este último problema no es debido al funcionamiento de OpenFlow si no al problema que explicamos en otros capítulos de la necesidad de ejecutar OVS en el espacio de usuario (*netdev*) y la implementación experimental de este. Como podemos ver en las Figuras 65 y 66, los picos son de aproximadamente 1 segundo como ya ocurría en sentido *downstream* en la ONT 1. Esto es debido a la cercanía del switch virtual y el controlador. Por otra parte, en las Figuras 67 y 68 podemos observar que la media de la tasa de velocidad en cada tramo es bastante cercana a lo pedido por el cliente, aunque tiene una variación muy grande sobre todo en la parte del servicio *premium* por factores ajenos a nuestro control como es la programación experimental del OVS.

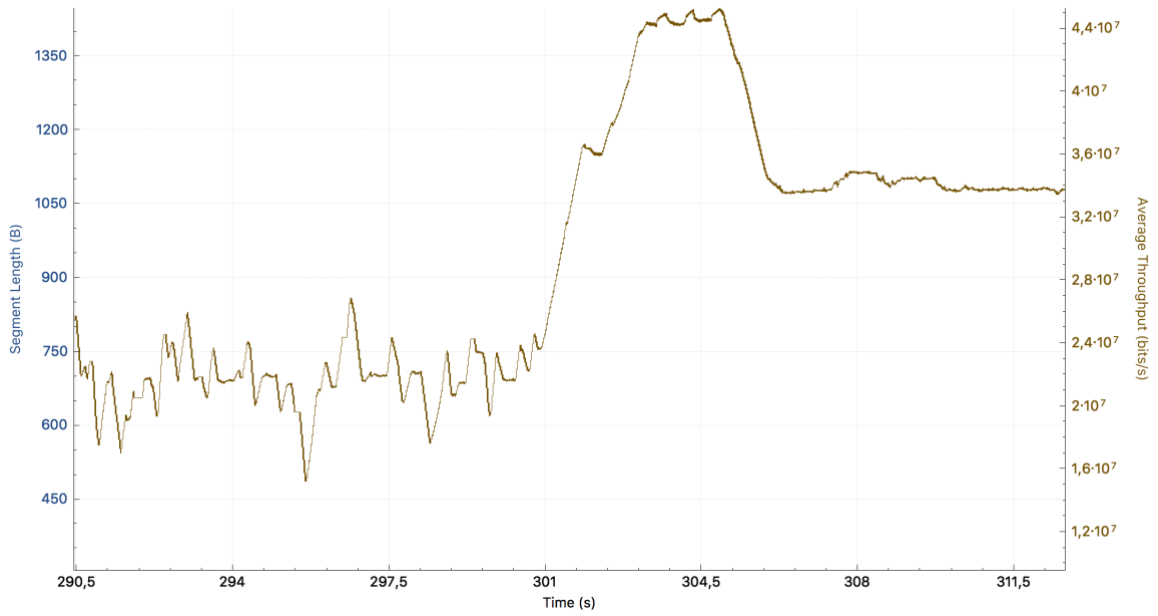


Figura 65: Pico en el segundo 300 de la comunicación con la ONT 2 en sentido *downstream*

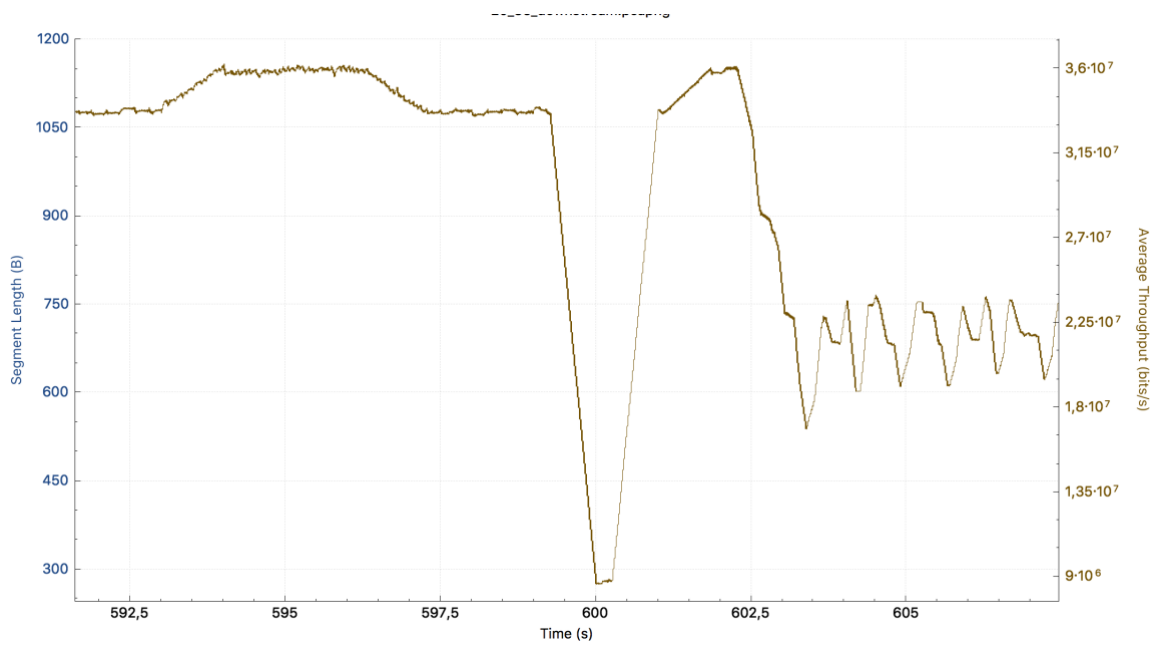


Figura 66: Pico en el segundo 600 de la comunicación con la ONT 2 en sentido *downstream*



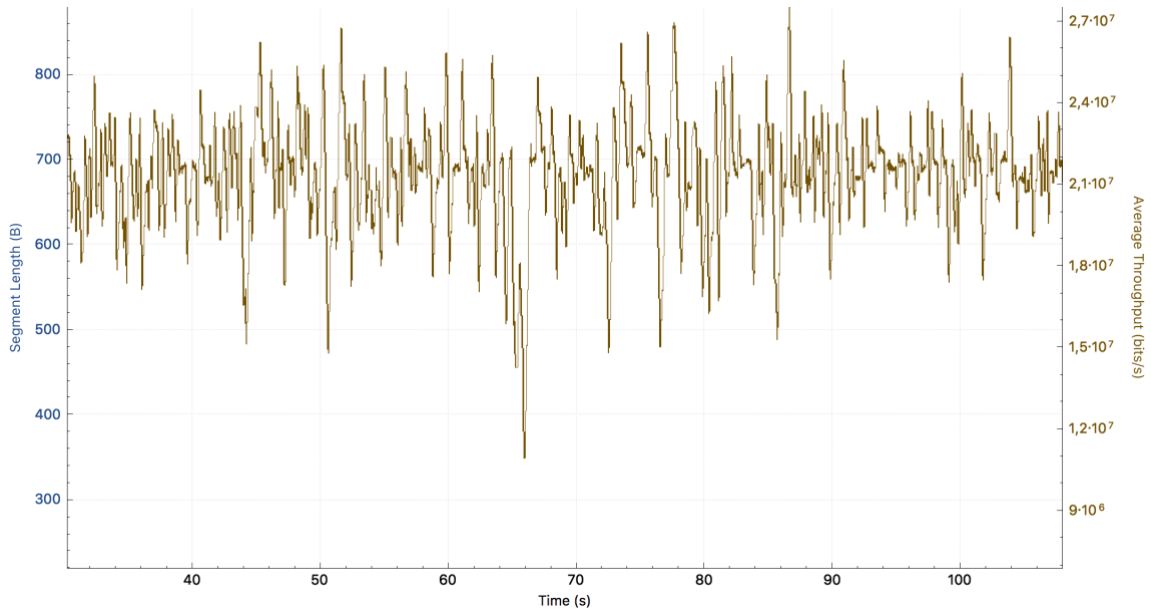


Figura 67: Gráfica del rizado detallado de 20 Mbps en la ONT 2 en sentido *downstream*

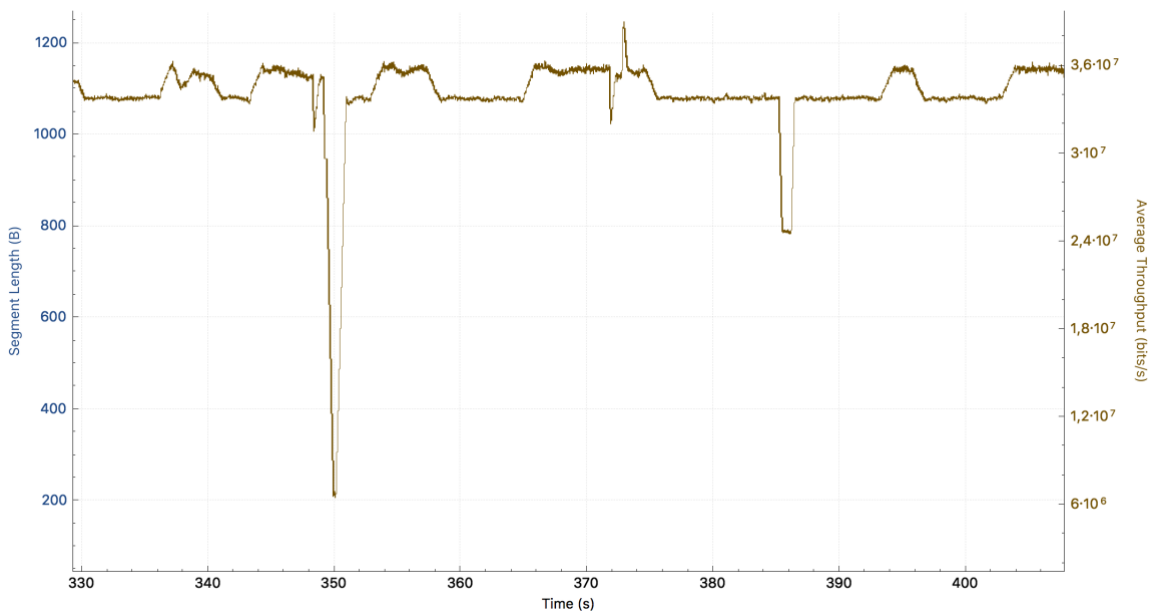


Figura 68: Gráfica del rizado detallado de 35 Mbps en la ONT 2 en sentido *downstream*

## 4.6 Conclusiones

En este capítulo, se ha descrito el escenario de red experimental necesario para poder implementar un caso de uso real, que también hemos explicado posteriormente. Para ello, además se ha programado e instalado una aplicación web programada en lenguajes de programación típicos de tecnologías web (HTML5, CSS, JavaScript, PHP) en cada Raspberry Pi que tiene la capacidad de comunicarse con un programa central diseñado en

Python en la Oficina Central que gestiona de forma automática y dinámica los recursos reales de la red GPON.

Finalmente, se han analizado las distintas problemáticas asociadas a este caso de uso debidas a razones ajenas a nuestro control del diseño e implementación del proyecto, y se han analizado los resultados experimentales obtenidos en el laboratorio, observando las distintas gráficas del ancho de banda obtenidas a través de la utilidad Wireshark, en la que observamos la fluctuación en el ancho de banda a lo largo de la comunicación TCP y el retardo entre la aplicación del flujo que sustituye al servicio por defecto y al flujo *premium*. Además, se ha analizado la comunicación OpenFlow entre los switches virtuales y el controlador OpenDayLight, llegando a la conclusión de que se ha logrado una gestión eficaz del tráfico de los servicios en tiempo real para las dos ONT del escenario SDN, utilizando el protocolo OpenFlow.



# 5

## Conclusiones y Líneas Futuras

### 5.1 Conclusiones

En este Trabajo de Fin de Grado se ha llevado a cabo la integración de un escenario de red SDN sobre una maqueta de red GPON (SDN-GPON) a través de un controlador y varios switches virtuales OpenFlow. Además, se ha propuesto un nuevo modelo de negocio para las operadoras en las que los usuarios pagan por el uso de la red en un escenario bajo demanda en tiempo real, demostrando su comportamiento en un caso de uso real.

Para lograr el objetivo, primero se ha analizado la estructura de la red de acceso GPON del laboratorio L2007 de la Escuela Técnica Superior de Ingenieros de Telecomunicación de Valladolid. Posteriormente, se han implementado los diferentes componentes cruciales para convertir una red de acceso GPON convencional en una red SDN, implementando un router en el ordenador central (que simula nuestra Oficina Central) para poder controlar la red a través de las habilidades de enrutamiento propias de un sistema Linux y usando otras utilidades como *iptables* (famoso cortafuegos y en general, interceptor de paquetes) y *vconfig* (para controlar VLANs). Además, se instalaron switches virtuales en el ordenador central (COVS) y en las Raspberry Pi colocadas detrás de cada ONT (ROVS) con la finalidad de emular funcionalidades SDN en el OLT y las ONTs. Finalmente, se implementó una comunicación OpenFlow entre todos estos switches virtuales y un controlador central (usando OpenDayLight) situado en la red troncal de la Escuela.

Posteriormente a la implementación técnica de todo el escenario de red SDN-GPON, se programó una interfaz web que se comunica con programa en Python situado en la Oficina Central para que el cliente, a través del servidor web instalado en cada

Raspberry Pi y accesible desde la red local final del cliente, pueda gestionar dinámicamente su ancho de banda con peticiones en tiempo real al proveedor de servicios de Internet.

Durante el desarrollo de este trabajo, se ha tomado consciencia de la dificultad del trabajo de investigación y de los retos a los que hay que enfrentarse para poder llevar un proyecto novedoso a buen puerto, además de la dificultad de plantear un proyecto basado en código abierto todavía en desarrollo, ya que muchas de las funciones no funcionan exactamente como están detalladas en la documentación o todavía no están implementadas.

## 5.2 Líneas Futuras

A partir de este proyecto, se puede llevar a cabo futuras implementaciones relacionadas con OpenFlow, siendo una de ellas mover ciertas políticas globales relacionadas con la asignación dinámica de ancho de banda (DBA, *Dynamic Bandwidth Allocation*) y recursos de la red GPON al controlador central OpenFlow. En este sentido, los algoritmos DBA distribuyen dinámicamente ciclo tras ciclo el ancho de banda disponible en una red PON basándose en las necesidades en tiempo real de cada usuario (conectado a una ONU/ONT) y a la prioridad de sus servicios contratados. Por tanto, este tipo de algoritmos proporcionan un reparto de ancho de banda más realista, flexible y eficiente en redes PON. El problema de dicha movilidad es que el protocolo con el que se implementa este algoritmo es muy cerrado entre la OLT y las ONUs en una red PON típica. En concreto, las ONTs/ONUs informan del tamaño de sus colas ciclo tras ciclo al OLT y este elemento central reparte el ancho de banda disponible en un ciclo, de manera acorde a la información recibida y siguiendo un esquema TDMA (*Time Division Multiplexing Access*) entre todos los usuarios. Puesto que esta asignación dinámica se realiza ciclo tras ciclo, el comportamiento de los algoritmos DBA depende crucialmente de que no exista un retardo excesivo entre el OLT y las ONTs/ONUs, por lo que su implementación directa en un controlador OpenFlow externo no es trivial y no resultaría eficiente. Sin embargo, se podría plantear mover ciertas políticas DBA genéricas, tales como la reconfiguración del ancho de banda máximo asignado a cada ONT/ONU en función de los recursos de la red, de la prioridad del usuario y del tráfico en tiempo real, lo cual podría resultar en una gestión más fina y optimizada de la QoS global en la red GPON. En uno de los artículos que mejor resume la problemática de implementar SDN en redes xPON [32] y como

sustituir el algoritmo DBA, se puede observar que el autor distingue una clave para realizar esta conversión. La creación de servicios suele estar controlada por GEM/XGEM Port-ID en la familia GPON, y estos parámetros suelen estar muy cerrados en los protocolos entre OLT y ONTs. Sin embargo, en nuestro proyecto la creación de servicios ya se puede realizar exitosamente con OpenFlow, así que el único problema que persiste es la latencia con el controlador a la hora de implementar este algoritmo a través de SDN.

Otra línea futura que también sería de gran interés para llevar a cabo una migración de la red GPON a una red SDN, es la creación de flujos y tablas OpenFlow a través de un programa central en Python para poder automatizar la creación de servicios y perfiles de abonado y la suscripción de estos a través de OpenFlow.

En este trabajo también hemos podido observar que el hecho de tener una ONT con capacidades de capa de red (capa 3) nos ha obligado a realizar la configuración de una determinada manera. Esto se puede solventar utilizando ONT que sólo tengan capacidades de capa de enlace (capa 2). Además de hacer más sencilla la configuración, de esta manera podemos programar el encaminamiento a través del protocolo OpenFlow y controlar completamente el encaminamiento en todos los puntos de nuestra red de acceso, además de poder soportar MPLS (*Multiprotocol Label Switching*), un protocolo que opera entre la capa de red y la capa de enlace y puede ser utilizado para transportar tráfico de voz o paquetes IP a lo largo de una red compleja (una buena alternativa para incluir los servicios de un operador clásico: voz, Internet y video) como se propone en [33].

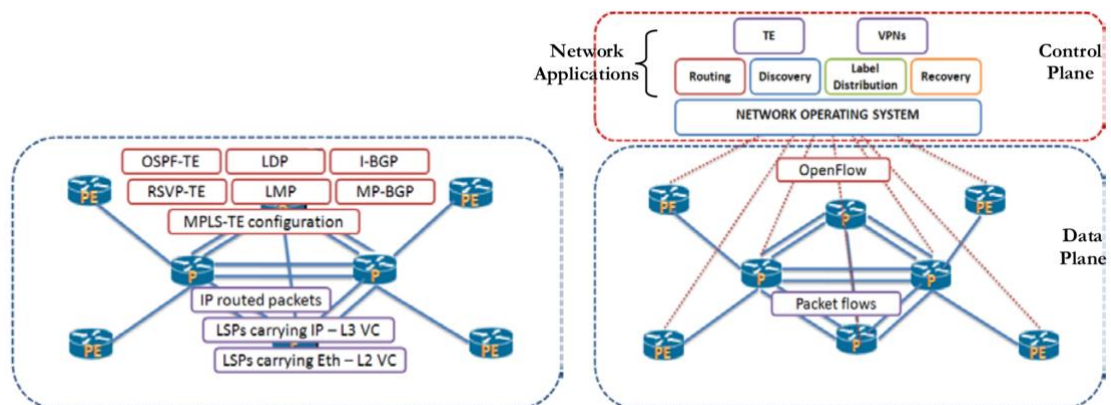


Figura 69: Esquema MPLS clásico frente a MPLS con OpenFlow

Para finalizar, se podría pensar en implementar la red SDN en nuestra red de acceso GPON con OpenStack [34], un sistema de computación en la nube que tiene capacidad para comunicarse con OpenDayLight y puede sustituir al protocolo OpenFlow migrando la estructura troncal de nuestra topología y virtualizando en la nube nuestra OLT y todo lo que hay detrás, como se observa en la Figura 70.

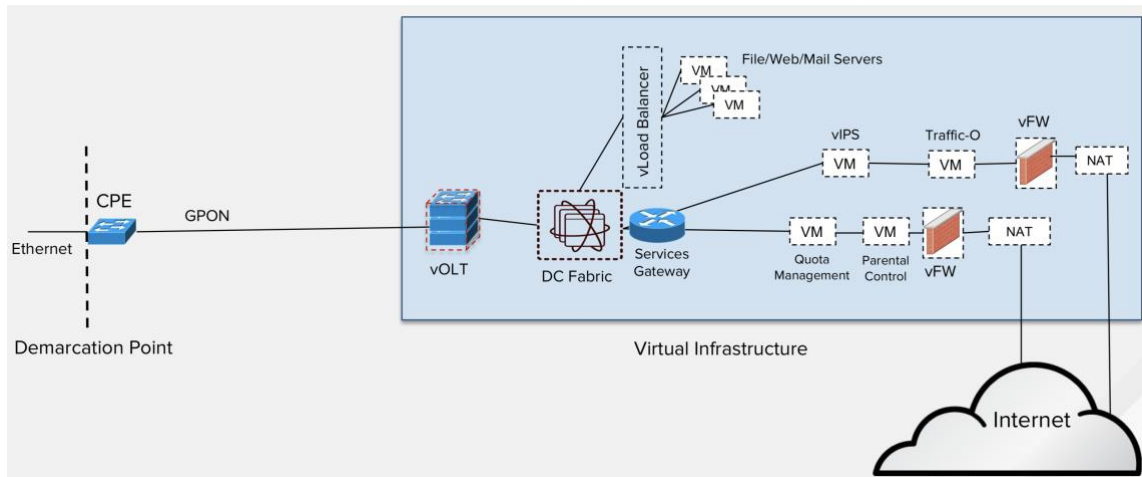


Figura 70: Topología virtualizada en un escenario de red SDN en una red GPON

# 6

## Bibliografía

- [1] «Antigua página web oficial de OpenFlow,» [En línea]. Available: <http://archive.openflow.org/wp/learnmore/>.
- [2] «Página web del controlador OpenDayLight,» [En línea]. Available: <https://www.opendaylight.org/>.
- [3] Linux Foundation, «Open vSwitch Documentation,» [En línea]. Available: <http://docs.openvswitch.org/en/latest/>. [Último acceso: 11 Mayo 2018].
- [4] «Estándar OpenFlow 1.3,» [En línea]. Available: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>.
- [5] The Linux Foundation, «Open vSwitch Documentation,» 2018. [En línea]. Available: <http://docs.openvswitch.org/en/latest/>. [Último acceso: 2018].
- [6] P. Göransson, Software defined networks: a comprehensive approach, 2014.
- [7] E. Nemeth, G. Snyder, T. R. Hein, B. Whaley y D. Mackin, Unix and Linux system administration handbook, 5ª edición ed., Boston: Addison-Wesley, 2018.



- [8] The Linux Foundation, «Open vSwitch on Linux, FreeBSD and NetBSD,» 2018. [En línea]. Available: <http://docs.openvswitch.org/en/latest/intro/install/general/>. [Último acceso: 26 Abril 2018].
- [9] D. Molloy, *Raspberry Pi® a fondo para desarrolladores*, Madrid: Marcombo, D.L., 2017.
- [10] The Linux Foundation, «Installing Open vSwitch,» 2018. [En línea]. Available: <http://docs.openvswitch.org/en/latest/intro/install/>. [Último acceso: 26 Abril 2018].
- [11] T. B. O. M. F. T. a. TTPs, «Open Networking Foundation,» 2 Febrero 2015. [En línea]. Available: [https://www.opennetworking.org/wp-content/uploads/2014/10/TR\\_Multiple\\_Flow\\_Tables\\_and\\_TTPs.pdf](https://www.opennetworking.org/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_and_TTPs.pdf). [Último acceso: 29 Mayo 2018].
- [12] Z. Hu, «A Comprehensive Security Architecture for SDN,» *18th International Conference on Intelligence in Next Generation Networks*.
- [13] S. W. e. a. Lee, Design and implementation of a GPON-based virtual OpenFlow-enabled SDN switch, vol. 34.
- [14] R. e. a. Gu, «Software defined flexible and efficient passive optical networks for intra-datacenter communications,» *Optical Switching Networking*, vol. 14, p. 289, 2014.
- [15] A. Amokrane, «Software defined enterprise passive optical network,» de *10th International Conference on Network and Service Management (CSNM)*, Rio de Janeiro, 2014.
- [16] H. Kahlili, D. Rincón y S. Sallent, «Towards and integrated SDN NFV architecture for EPON networks,» de *Advances in Communication Networking (LNCS 8846)*, Cham, 2014.

- [17] L. Youngsuk, «A design of 10 Gigabit Capable Passive Optical Network(XG-PON1) architecture based on Software Defined Network (SDN),» de *International Conference on Information Networking (ICOIN)*, 2015.
- [18] P. Parol y M. Pawlowski, «Towards networks of the future: SDN paradigm introduction to PON networking for business applications,» de *Federated Conference on Computer Science and Information Systems*, Krakow, 2013.
- [19] IEEE, «802.1Q-2011 - IEEE Standard for Local and metropolitan area networks--Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks». 31 Agosto 2011.
- [20] S. Seth y M. A. Venkatesulu, *TCP/IP Architecture, Design and Implementation in Linux*, Wiley-IEEE Press eBook Chapters, 2008.
- [21] «vconfig(8) - Linux man page,» [En línea]. Available: <https://linux.die.net/man/8/vconfig>. [Último acceso: 7 Mayo 2018].
- [22] «vconfig(8) - Linux man page,» [En línea]. Available: <https://linux.die.net/man/8/vconfig>. [Último acceso: 9 Mayo 2018].
- [23] Internet Systems Consortium, «ISC's open source DHCP software system,» 2003. [En línea]. Available: <https://www.isc.org/downloads/dhcp/>. [Último acceso: 21 Mayo 2018].
- [24] Y. U. I. i. OpenDayLight, «OpenDayLight Summit,» 29 Julio 2015. [En línea]. Available: <https://events.static.linuxfound.org/sites/events/files/slides/YANGUI-metz-malachovsky-sebin-ODL-Summit-final-July29.pdf>. [Último acceso: 30 Mayo 2018].

- [25] R. 6. YANG, «IETF,» Octubre 2010. [En línea]. Available: <https://tools.ietf.org/html/rfc6020>. [Último acceso: 30 Mayo 2018].
- [26] C. R. o. openflowplugin.git, «OpenDayLight,» 18 Septiembre 2013. [En línea]. Available: <https://git.opendaylight.org/gerrit/gitweb?p=openflowplugin.git;f=model/model-flow-base/src/main/yang/opendaylight-meter-types.yang;a=blob;hb=refs/heads/stable/boron>. [Último acceso: 2018].
- [27] The Apache Software Foundation, «Versión 2.4 de la documentación del Servidor de HTTP Apache,» [En línea]. Available: <http://httpd.apache.org/docs/2.4/>. [Último acceso: 23 Mayo 2018].
- [28] Raspberry Pi Foundation, «Raspberry Pi 3 Model B Specifications,» [En línea]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. [Último acceso: 23 Mayo 2018].
- [29] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer y K. Prabhu, «iPerf - The ultimate speed test tool for TCP, UDP and SCTP,» [En línea]. Available: <https://iperf.fr>. [Último acceso: 23 Mayo 2018].
- [30] «iperf(1) - Linux man page,» [En línea]. Available: <https://linux.die.net/man/1/iperf>. [Último acceso: 23 Mayo 2018].
- [31] G. Combs, «Wireshark,» 1998. [En línea]. Available: <https://www.wireshark.org>. [Último acceso: 23 Mayo 2018].
- [32] E. Dai y W. Dai, «Towards SDN For Optical Access Networks,» *Spring Technical Forum Proceedings*, 2016.
- [33] I. S. C. i. M. Networks, «Department of Computer Science,» [En línea]. Available: [http://yuba.stanford.edu/~sd2/Ch\\_5.pdf](http://yuba.stanford.edu/~sd2/Ch_5.pdf). [Último acceso: 12 Junio 2018].

- [34] O. Project, «OpenStack,» [En línea]. Available:  
<https://www.openstack.org>. [Último acceso: 8 Junio 2018].



## Anexo I

### Script para habilitar el encaminamiento

```
echo 1 > /proc/sys/net/ipv4/ip_forward

iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o
VLAN833 -j MASQUERADE

iptables -t nat -A POSTROUTING -s 10.19.59.0/24 -o
VLAN833 -j MASQUERADE

#iptables -t mangle -A POSTROUTING -d 192.168.0.0/24 -j
CLASSIFY --set-class 0:5

vconfig add enp4s1 833

vconfig add enp4s1 806

ifup enp4s1

ip addr flush dev enp4s1

ifconfig enp4s1.833 192.168.0.1 netmask 255.255.255.0

ifconfig enp4s1.806 10.19.59.1 netmask 255.255.255.0

ifconfig enp4s0 0

sudo route add -net 192.168.1.0/24 gw 192.168.0.103 dev
enp4s1.833

service isc-dhcp-server restart
```

## Anexo II

### Ficheros de configuración DHCP

```
subnet 192.168.0.0 netmask 255.255.255.0 {  
  
    range 192.168.0.100 192.168.0.250;  
  
    option domain-name-servers 157.88.129.90;  
  
    option domain-name "RouterTFGLab7-833";  
  
    option routers 192.168.0.1;  
  
    option broadcast-address 192.168.0.255;  
  
    default-lease-time 600;  
  
    max-lease-time 7200;  
  
}  
  
subnet 10.19.59.0 netmask 255.255.255.0 {  
  
    range 10.19.59.100 10.19.59.200;  
  
    option domain-name-servers 157.88.129.90;  
  
    option domain-name "RouterTFGLab7-866";  
  
    option routers 10.19.59.1;  
  
    option broadcast-address 10.19.59.255;  
  
    default-lease-time 600;  
  
    max-lease-time 7200;  
  
}
```

# Anexo III

## Aplicación web en HTML5

```

<!DOCTYPE html>
<html>
<!--
index.html está realizada en HTML5
Autor: Javier Azofra
Última modificación: 17 de Abril de 2018
-->
<head>
  <title>Gestión dinámica del ancho de banda</title>

  <!--Metadatos
  -->
  <meta charset="utf-8"/>
  <!-- Información para que la página web sea completamente responsiva para cualquier dispositivo
  -->
  <meta name = "viewport" content = "width=device-width, maximum-scale = 1, minimum-scale=1" />
  <meta name ="description" content = "Gestión dinámica del ancho de banda"/>
  <meta name = "author" content = "Javier Azofra"/>

  <!--Información de estilo. La información propia de estilo para está página se ha introducido en
  default.css
  -->
  <link rel="stylesheet" type="text/css" href="css/default_form.css" media="all" />
  <link href='http://fonts.googleapis.com/css?family=PT+Sans' rel='stylesheet' type='text/css' />
  <link rel="stylesheet" href="css/fixed-navigation.css" type="text/css" />
  <link href='http://fonts.googleapis.com/css?family=PT+Sans' rel='stylesheet' type='text/css' />

  <!--Scripts utilizados
  -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
  <script src="js/jquery.flexslider.js"></script>
  <script src="js/default.js"></script>
  <script src="js/validarencuesta.js"></script>
  <script src="js/avisorestablecer.js"></script>
</head>

<!--La función initialize() debe ser inicialmente llamada para el desplazamiento dentro de la página
-->
<body onload="initialize()">
  <div id="pagewidth">
    <header id="header">
      <div class="center">
        <nav id="mainNav">
          <ul>
            <li class="active"><a href="#start"><span>Inicio</span></a></li>
            <li><a href="#sugerencias"><span>Contáctanos</span></a></li>
            <li><a href="index_en.html"><span>Página en inglés</span></a></li>
          </ul>
        </nav>
      </div>
    </header>
  </div>

  <!--La primera sección irá destinada a enfatizar sobre la idea. Debería tener un
  estilo propio que ayudara a tal fin.
  -->
  <section class="row">
    <div id="start" class="center">
      <br><br><br><br>
      <h1>Gestión dinámica del ancho de banda</h1>
    </div>

    <!--A continuación, establecemos el método de realimentación para el formulario en sí, siendo est
    un documento PHP
    -->
    <form action="php/abrirSocket.php" name="formulario" method="post">

    <!--A continuación se informa al usuario de la forma de proceder en el formulario
    -->
    <section id="horas" class="row grey">
    <!--Preguntamos por los parámetros necesarios
    -->
      <div class="columns">
        <div class="half">
          <label><center>Ancho de banda contratado<input class="number" type="text" value="50 Mbps"></
          center></label>
          <label><center>Tiempo en minutos<input class="number" type="number" value="0" name="minutos"
          id="minutos"></center></label>
        </div>

```



```

        <div class="half">
        <label><center>Selecciona el ancho de banda que deseas
        </center></label>
        <input type="range" id="r" name="r" min="10" max="300" list="tickmarks">
        <!--Aquí se incluye un script en javascript que permite que el slider de selección del ancho de
        banda funcione correctamente
        -->
        <script type="text/javascript">
            function myTimer() {
                var r = document.getElementById('r').value;
                var x = document.getElementById('prueba');
                x.innerHTML = r + " Mbps";
            }
            var myVar = setInterval(function(){ myTimer() }, 50);
        </script>

        <center><label><div id="prueba"></label><center>
        </section>

        <center><input class="lezcano2" type="submit" name="envio" value="Envio"></center>
    </form>
    <section id="sugerencias" class="row grey">
        <div class="center">
            <h1>Contáctanos</h1>
            <div class="columns">
                <div class="half">
                    <!-- Formulario con el manejador de eventos onSubmit, que retorna la función
                    validarEmail, función recogida en validaremail.js.-->
                    <form action="mailto:javiazofra@gmail.com" name="contactoFormulario"
                    method="post" enctype="text/plain" onSubmit="return validarEmail(window.document.contactoFormulario);">
                        <fieldset>
                            <h2>Formulario de contacto</h2>
                            <div id="message"></div>
                            <div class="formRow">
                                <div class="textField"><input type="text" name="nombre-contacto"
                                id="name" placeholder="Nombre..." /></div>
                                </div>
                            <div class="formRow">
                                <div class="textField"><input type="text" name="email"
                                id="email" placeholder="Email..." /></div>
                                </div>
                            <div class="formRow">
                                <div class="textField"><textarea cols="20" name="comentario-
                                contacto" id="comment" rows="4" placeholder="Comentario..." required></textarea>
                                </div>
                                </div>
                            <div class="formRow">
                                <button class="btnSmall btn submit right" type="submit"
                                value="Mandar mensaje">
                                    <span>Mandar mensaje</span>
                                </button>
                            </div>
                        </fieldset>
                    </form>
                </div>
                <div class="half">
                    <div>
                        <a class="imgHolder fullWidth">
                </div>
            </div>
        </div>
    </section>
    </section>
    <footer id="footer">
        <div class="center">
            <span class="copy">Creado por Javier Azofra</span>
        </div>
    </footer>
</body>
</html>

```

## Anexo IV

### Aplicación en PHP

```

<?php
/*-----*/
/- Documento: abrirSocket.php -/
/- Autores: Javier Azofra -/
/- Última modificación: 03/05/2018 -/
/- Explicación: Este documento PHP recibirá del documento -/
/- index.html los datos necesarios del ancho de banda -/
/- y los minutos de servicio seleccionados por el usuario -/
/*-----*/
include("cabecera.html");

//Obtenemos del formulario los parámetros necesarios
@$minutos=$_POST['minutos'];
@$ancho_banda=$_POST['r'];

$minutos=trim($minutos);
$r=trim($r);

//No dejamos continuar si no se ha rellenado el campo de minutos
if($minutos=='0')
{
    echo "Tiene que rellenar el campo de minutos";
    exit;
}
else
{
    $cadena = $minutos . "/" . $ancho_banda;
    $cliente = stream_socket_client("tcp://192.168.3.4:5005", $errno, $errorMessage);
    if ($cliente === false)
    {
        throw new UnexpectedValueException("Failed to connect: $errorMessage");
    }
    fwrite($cliente, $cadena);
    echo $cadena;

    $host = "192.168.3.4";
    $port = 65500;
    // don't timeout!
    set_time_limit(0);
    // creamos el socket
    $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP) or die("Could not create socket\n");
    // hacemos un bind al puerto que queremos
    $result = socket_bind($socket, $host, $port) or die("Could not bind to socket\n");
    // empezamos a escuchar conexiones
    $result = socket_listen($socket, 3) or die("Could not set up socket listener\n");

    // Aceptamos la conexiones y leemos del socket
    $spawn = socket_accept($socket) or die("Could not accept incoming connection\n");
    $input = socket_read($spawn, 1024) or die("Could not read input\n");
    // Limpiamos la cadena recibida del socket
    $input = trim($input);
    // Si el mensaje que recibimos por el socket es si, imprimimos que se ha contratado
    echo "server Message : ".$input;
    if (strcmp("Si", $input))
    {
        echo "El ancho de banda ha sido contratado";
    }
    else
    {
        echo "El ancho de banda no se ha podido contratar";
    }
}
include("pie.html");
?>

```

# Anexo V

## Aplicación en Python

```

def Main():

    MAX_DOWN=2.5*1024*1024
    MAX_UP=1.25*1024*1024

    TCP_IP = '10.0.103.48'
    TCP_PORT = 5005
    BUFFER_SIZE = 1024

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((TCP_IP, TCP_PORT))
    s.listen(1)

    conn, addr = s.accept()
    print ("Connection address:" + str(addr))
    while 1:
        data = conn.recv(BUFFER_SIZE)
        if not data: break
        print ("received data:" + data.decode())
        cadena = data.decode()
        if len(cadena)!=0: break
    s.close()

    posicion=cadena.find("/")
    primera_cadena=cadena[0:(posicion)]
    segunda_cadena=cadena[posicion+1:(len(cadena))]
    tercera_cadena=segunda_cadena + ' Mbps'

    print ("Ancho de banda:" + tercera_cadena)
    segundos=int(primera_cadena)*60
    print ("Segundos:" + str(segundos))

    ONTs=('54-4c-52-49-5b-01-f6-90','54-4c-52-49-5b-01-
        f7-30','54-4c-52-49-5b-01-f6-d8','54-4c-52-49-5b-01-f7-28')
    cnx = mysql.connector.connect(user='root',
        password='tfg_2017',host='127.0.0.1',database='gponServices')
    cursor = cnx.cursor()
    downAcumulado=0
    upAcumulado=0

    for i in ONTs:
        cursor.execute("select sum(gDownstream),sum(gUpstream) from services
            where id_service in (select id_service from ont_service where
                id_ont='"+i+"')")
        anchos=cursor.fetchall()
        anchos=anchos[0]
        if((anchos[0] is not None) & (anchos[1] is not None)):
            downAcumulado=anchos[0]+downAcumulado
            upAcumulado=anchos[1]*1024+upAcumulado
    print(downAcumulado)
    print(upAcumulado)
    cursor.close()
    cnx.close()

```

```

if(((downAcumulado+float(segunda_cadena)*1024)>MAX_DOWN) |
((upAcumulado+float(segunda_cadena)*1024)>MAX_UP)):
    mensaje='No'
else:
    mensaje='Si'
    print(mensaje)
    #Aquí se hace arp -a para conocer la MAC de downstream a utilizar
    proc = subprocess.Popen('arp -a', stdout=subprocess.PIPE, shell=True)
    (out, err) = proc.communicate()
    out=out.decode()
    ipQuery=str(addr[0])
    posicion=out.find(ipQuery)
    print(posicion)
    mac_downstream=out[(posicion+18):(posicion+35)]
    print(mac_downstream)
    #Aquí se elige la MAC de upstream en función de la IP desde la que
    se ha hecho la petición
    if ipQuery == "192.168.0.114":
        ovs_upstream="346653124572" #Id del OVS para el upstream en los
        servicios del ONT f6-d8
        mac_upstream="78:3d:5b:01:f6:d8"
    elif ipQuery == "192.168.0.113":
        ovs_upstream="" #Id del OVS para el upstream en los servicios
        del ONT f7-28
        mac_upstream=""
    elif ipQuery == "192.168.0.104":
        ovs_upstream="346653127080" #Id del OVS para el upstream en los
        servicios del ONT f7-30
        mac_upstream="78:3d:5b:01:f7:30"
    # elif ipQuery == "":
    #     ovs_upstream="" #Id del OVS para el upstream en los servicios
    #     del ONT f6-90
    #     mac_upstream=""

    servicio_Internet_OpenFlow(int(segunda_cadena)*1024,int(segunda_cad
    ena)*1024,segundos,mac_downstream,mac_upstream,ovs_upstream)

s2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s2.connect(("192.168.3.4",9005))

time.sleep(2)
while 1:
    s2.send(mensaje.encode())
    break;

```