UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Diseño de un agente OpenFlow en una maqueta Red de Acceso Óptica real

Autor:
Don Ángel Gómez Aguado
Tutor:
Dña Noemí Merayo Álvarez

TÍTULO: **Diseño de un agente OpenFlow en una maqueta Red de Acceso Óptica real**
AUTOR: **Don Ángel Gómez Aguado**
TUTOR: **Dña. Noemí Merayo Álvarez**
DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE: Ignacio de Miguel Jiménez
VOCAL: Ramón J. Durán Barroso
SECRETARIO: Juan Carlos Aguado Manzano
SUPLENTE: Noemí Merayo Álvarez
SUPLENTE: Rubén M. Lorenzo

FECHA: 20 de Julio de 2017
CALIFICACIÓN:

Resumen de TFG

La investigación realizada en este Trabajo de Fin de Grado se basa en la creación de un agente OpenFlow capaz de configurar la red de acceso GPON situada en el laboratorio de Comunicaciones Ópticas de la Escuela Técnica Superior de Ingenieros de Telecomunicación.

En primer lugar, se analizó la topología de la red y sus elementos principales, es decir, el OLT (*Optical Line Termination*) y las ONUs/ONTs (*Optical Network Unit/Terminal*), así como la configuración de un servicio de red mediante TGMS (*TELNET GPON Management System*) y CLI (*Command Line Interface*) y eligiendo finalmente CLI por su versatilidad y posibilidad de automatización en la configuración de la red de acceso.

A continuación, se buscó información sobre el protocolo OpenFlow y su funcionamiento, llevándose a cabo una comparación de diferentes controladores con el objetivo de utilizar el más adecuado para el propósito de este trabajo. Una vez elegido el tipo controlador, en concreto OpenDayLight, se hicieron pruebas con él y una red virtual Mininet probando diferentes funcinalidades básicas, para finalmente idear un método que nos llevase a alcanzar el objetivo propuesto. Se analizaron diferentes mensajes del protocolo OpenFlow y se creó el programa en Python.

Palabras clave

GPON (red óptica pasiva con capacidad de gigabit), OLT (terminación óptica de línea), ONU (unidad de red óptica), TGMS (sistema de gestión GPON de TELNET), servicio de Internet, servicio de vídeo, CLI (interfaz de línea de comandos), Python, OpenDayLigth, OpenFlow.

Abstract

The research carried out in this End-of-Grade Work is based on the development of one OpenFlow agent able to configure the GPON access network located at the Optical Communications Laboratory of the School of Telecommunication Engineers.

Firstly, the network topology and its principal devices were analyze, that is, the OLT (Optical Line Termination) and the ONUs/ONTs (Optical Network Unit/Terminal), as well as the configuration of network services using the TGMS (TELNET GPON Management System) and the CLI (Command Line Interface) management systems and finally choosing the CLI due to its versatility and automation power when configuring the access network.

Then, information about the OpenFlow protocol and its operation was searched, comparing different controllers with the objective of using the most suitable our objectives. Once the controller was chosen, namely OpenDayLight, it was tested with a virtual network using Mininet testing different basic functions, to finally devise a method that would lead us to reach the proposed objective. Different messages of the OpenFlow protocol were analyzed and the final agente was programmed in Python.

Keywords

GPON (Gigabit-capable Passive Optical Network), OLT (Optical Line Termination), ONU (Optical Network Unit), TGMS (TELNET GPON Management System), internet service, video service, CLI (Command Line Interface), Python, OpenDayLigth, OpenFlow.

ÍNDICE

ÍNDICE DE FIGURAS

1

Introducción

1.1 Motivación

En este Trabajo de Fin de Grado se ha llevado a cabo la creación de un agente OpenFlow capaz de configurar servicios de red en una red de acceso GPON (*Gigabit-capable Passive Optical Network*). 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.

La red GPON consiste, como su propio nombre indica, en una red óptica pasiva con capacidad para el estándar Gigabit. Gracias a su simplicidad, debida a la no necesidad de elementos activos entre el operador y las dependencias del abonado (20-25 km), las redes GPON son las redes de acceso más desplegadas a nivel mundial en la actualidad. La red se basa en una topología en árbol, donde hay un elemento central que da servicio a varios usuarios utilizando el protoloco GPON, descrito en G.984.x del ITU-T (*International Telecommunication Union*). Dicho protocolo permite una tasa máxima de 2,5 Gbps en el downstream y 1,25 Gbps en el upstream.

Como se ha mencionado, la red utiliza una topología en arbol. El componente central es el OLT (*Optical Line Termination*), que da servicio a varias ONUs/ONTs (*Optical Network Unit/Terminal*) que tendrían los usuarios en sus casas. En el caso de la red de acceso en la que se ha trabajado, dichos componentes han sido diseñados por la empresa TELNET Redes Inteligentes [1]. Otros componentes importantes son los divisores ópticos, que permiten dividir la señal para poder llegar a más usuarios y, cómo no, la fibra óptica monomodo que conecta el resto de componentes.

Por otro lado, 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 [2]. 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. Con esta filosofía, el objetivo de este trabajo ha sido utilizar OpenFlow para que un controlador configure los servicios de una red GPON a través de la gestión de un OLT en vez de un switch o router, con vista por ejemplo a que en el futuro se puedan configurar varias OLTs a la vez en función de las necesidades del momento o que el controlador gestione la red de acceso, la red metropolitana o la red troncal de forma simultánea.

Para la creación del agente OpenFlow que hará de intermediario entre el controlador y el OLT se utilizará el lenguaje de programación Python, ya que existen unas librerías en este lenguaje que permiten la creación de mensajes OpenFlow con cierta facilidad para interacturar directamente sobre el OLT a tráves del controlador elegido.

1.2 Objetivos

1.2.1 Objetivo General

El principal objetivo de este Trabajo de Fin de Grado es el desarrollo de un programa (agente OpenFlow-Python) que permita a un controlador OpenFlow configurar diferentes servicios de red y perfiles de abonado en una red de acceso GPON utilizando mensajes OpenFlow. 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 maqueta de la red de aceso GPON: topología, arquitectura y componentes principales que la forman.

2. Análisis del protocolo OpenFlow y comparativa entre diferentes controladores.

3. Instalación y análisis de un entorno de pruebas virtual en Virtualbox utilizando Mininet y OpenDayLight.

4. Creación de un agente en Python que, utilizando el protocolo OpenFlow, sea capaz de a partir de las tablas diseñadas dentro controlador pueda modificar la configuración de la red GPON (servicios, perfiles) sin necesidad de un switch openflow.

1.3 Fases y Métodos

La metodología a seguir para el desarrollo de los objetivos del Proyecto Fin de Carrera 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 del montaje real de la red GPON*: estudio de la topología y estructura de la red GPON, así como sus diferentes modos de configuración.
- *Análisis del protocolo OpenFlow*: estudio del funcionamiento básico de este protocolo, sus diferentes versiones y tipos de controladores existentes.

1.3.2 Fase de Configuración

Esta fase tiene como objetivo la configuración de servicios de Internet y vídeo, así como perfiles de abonado en una red GPON. Una vez se ha adquirido el suficiente manejo de la red a través del análisis físico anterior, se ha de poner en marcha (es decir, configurar) de forma que los usuarios de la red puedan disponer de estos servicios. Esta fase está divida en dos partes, atendiendo al modo de gestión de la red utilizado:

- ✓ Configuración de servicios utilizando el modo de gestión TGMS: se analizará el sistema de gestión web desarrollado por TELNET Redes

Inteligentes [1] para configurar los diferentes parámetros que forman los servicios y posteriormente, agrupar los mismos en perfiles de usuario que serán asignados a los clientes de la red (ONTs/ONUs).

- ✓ Configuración de servicios utilizando el modo de gestión CLI: se analizará en profundidad dicho modo de gestión basado en comandos y más cercano al estándar GPON [3] para la configuración de servicios. Esta configuración de la red se llevará a cabo a través de comandos acordes a las entidades definidas en dicho protocolo.

Por otro lado, en esta fase también se elegirá el controlador OpenDaylight que tomaremos para gestionar la red GPON una vez hecha la comparativa en la fase anterior. El siguiente paso de este punto será configurar el controlador de forma adecuada para que se pueda comunicar con nuestra red de acceso. Para ello, también se configurarán las tablas adecuadas a dicha gestión y a la configuración de servicios y perfiles de abonado asociados a las ONTs/ONUs conectadas a la red.

1.3.3 Fase de Programación

En esta fase final del proyecto se llevará a cabo una tarea de programación. Se utilizará el lenguaje de programación Python [4] como base, partiendo de una serie de programas implementados en trabajos anteriores que automatizan la gestión y configuración de la red de acceso mediante el modo CLI proporcionado por el fabricante de los dispositivos. Partiendo de lo desarrollado en dicho trabajo, se diseñará un agente en Python que se comunicará por un lado con el controlador OpenDaylight mediante el intercambio de mensajes OpenFlow, y por otro lado, dicho agente interacturá con el OLT transformando la información de dichos mensajes OpenFlow en el lenguaje del CLI, que será el que entienda el OLT para configurar sus servicios y perfiles de abonado dentro de la red GPON.

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 TFG 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 analizando diferentes controladores OpenFlow. Tras decantarnos por uno de ellos, se describe una prueba del funcionamiento básico de OpenFlow y después se describe cómo y qué tablas OpenFlow hay que configurar para el funcionamiento adecuado del programa objetivo.

En el Capítulo 4 se describen con detalle los mensajes OpenFlow intercambiados entre un controlador OpenDayLight y un swich virtual en Mininet. Posteriormente se describe el diseño del agente OpenFlow-Python describiendo en primer lugar la emulación de la comunicación OpenFlow entre dicho agente y el controlador OpenDayLight. Finalmente, se describirá cómo el controlador OpenDayLight envía las tablas OpenFlow pertinentes y el agente desencapsula dicha información y la transformar en el lenguaje que entiende el OLT (comandos CLI) para la configuración y gestión de servicios y perfiles de abonado en la red.

Por último, en el Capítulo 5 se tratan las conclusiones y diferentes líneas futuras que pueden seguirse a partir de este trabajo, y en el Capítulo 6 todas las referencias utilizadas. Finalmente, en el Anexo I se encuentra el código del agente OpenFlow-Python desarrollado en este Trabajo Fin de Grado.

2

Metodología y herramientas de trabajo

2.1 Introducción

En este capítulo se realizará un análisis descriptivo de los componentes principales de este Trabajo de Fin de Grado, es decir, la red de acceso GPON y el protocolo OpenFlow. Además, también se describirá la metodología utilizada para desarrollar este trabajo.

La red de acceso GPON 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 TGMS (*TELNET GPON Management System*) o, directamente, accediendo al CLI (*Command Line Interface*) del OLT mediante su puerto de configuración. Para este trabajo, va a ser utilizado el CLI, por su versatilidad a la hora de automatizar la gestión de servicios en la red de acceso.

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 *control path* (toma de decisiones). OpenFlow cuenta con varias versiones y controladores, siendo la versión 1.3 y el controlador OpenDayLigth los que se van a utilizar en este trabajo.

En cuanto a la metodología, se explicará paso a paso los pasos que se han ido siguiendo; desde la elección del controlador hasta la programación y ejecución del programa, pasando por las pruebas hechas con el controlador y la elección del lenguaje de programación.

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 estos puedan darles a dichos usuarios los servicios que han contratado.

Nuestra red de acceso óptica GPON tiene una topología en árbol, donde un OLT (*Optical Line Terminal*) da servicio a varias ONUs/ONTs (*Optical Network Unit/Terminal*). 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 son, por supuesto, la fibra óptica monomodo que conecta los diferentes componentes 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.

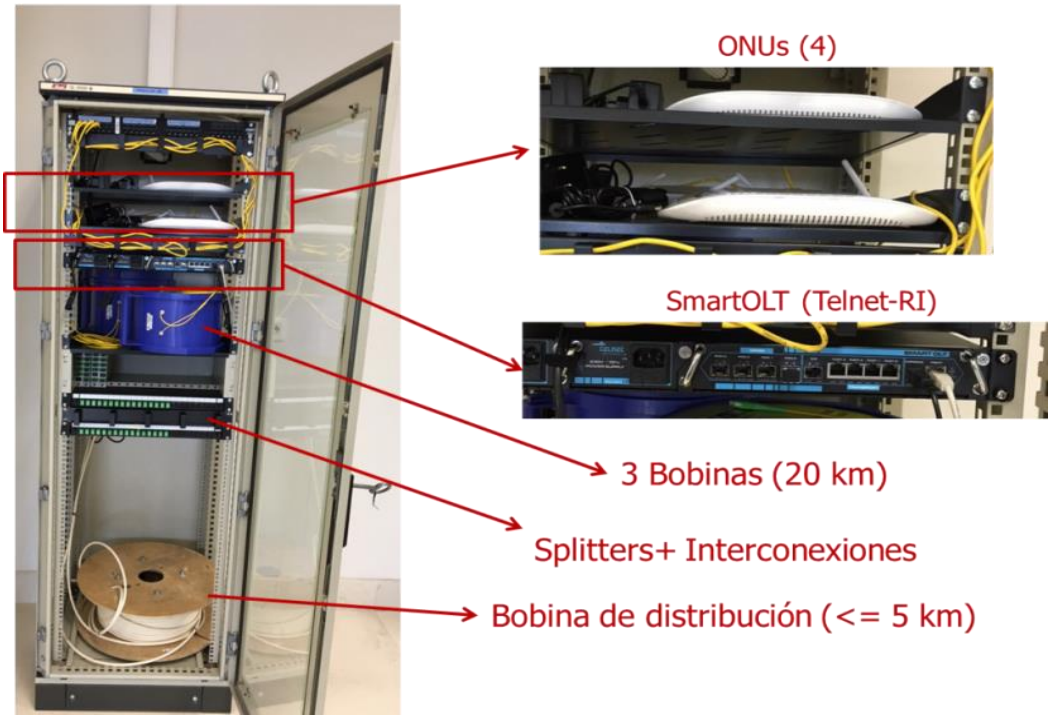

Figura 1: Esquema general de la red de acceso GPON desplegada en el laboratorio

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

- **Flujo descendente**: conocido más comúnmente como *downstream*, es el flujo que lleva los datos con las ONUs como destino. 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 más comúnmente como *upstream*, es el flujo que lleva los datos con las ONUs como 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 2º ventana 1310 nm.

Es importante tener en cuenta estos dos flujos, ya que la suma de las tasas de *upstream* y *downstream* dadas a cada uno de los servicios proporcionados a las diferentes 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, existen dos métodos diferentes: el TGMS (*TELNET GPON Management Sstem*) 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 a cada una de las ONUs conectadas al OLT. Además, también muestra información que puede ser de utilidad, como la potencia recibida en el OLT y las ONUs. Este método está diseñado para que el usuario pueda configurar todo de forma más amigable.

El otro método, denominado CLI (Figura 2), consiste en conectarse mediante telnet a la IP y puerto de configuración del OLT para poder acceder a su CLI y configurarlo manualmente a base de comandos. 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 d forma automatizada introduciendo los comandos adecuados. Como nuestro objetivo es modificar los servicios que el OLT ofrece a las ONUs de forma automática y sencilla, este va a ser el método que se va a utilizar.

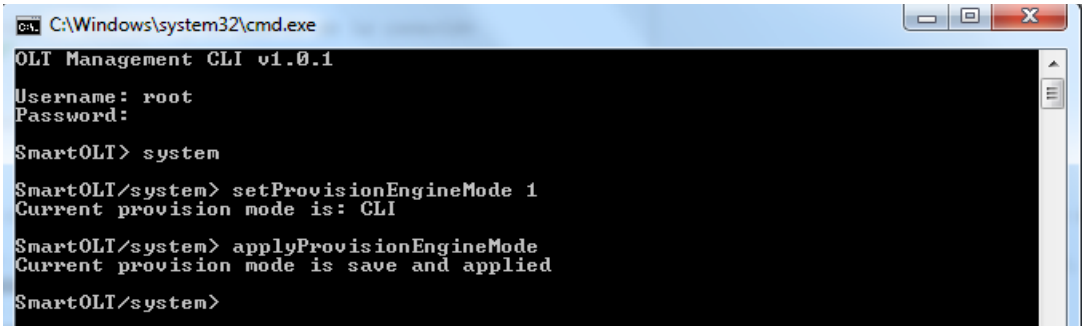

Figura 2: Activación del modo de gestión CLI para la configuración de la red GPON

2.3 Principios de funcionamiento del estándar OpenFlow

Como se dijo en la introducción, OpenFlow es un estándar 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* [2].

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, 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 [5].

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 controlar uno o varios routers o switches en tiempo real, lo que permite poder cambiar radicalmente las conexiones de una red en función de parámetros como el tráfico que pasa por ciertos puntos, por ejemplo [5].

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 [6]. En el siguiente capítulo se describirá la comparativa entre diversos controladores y la justificación de dicha elección final.

El controlador OpenDayLight es un versátil controlador 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 [6]. 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.

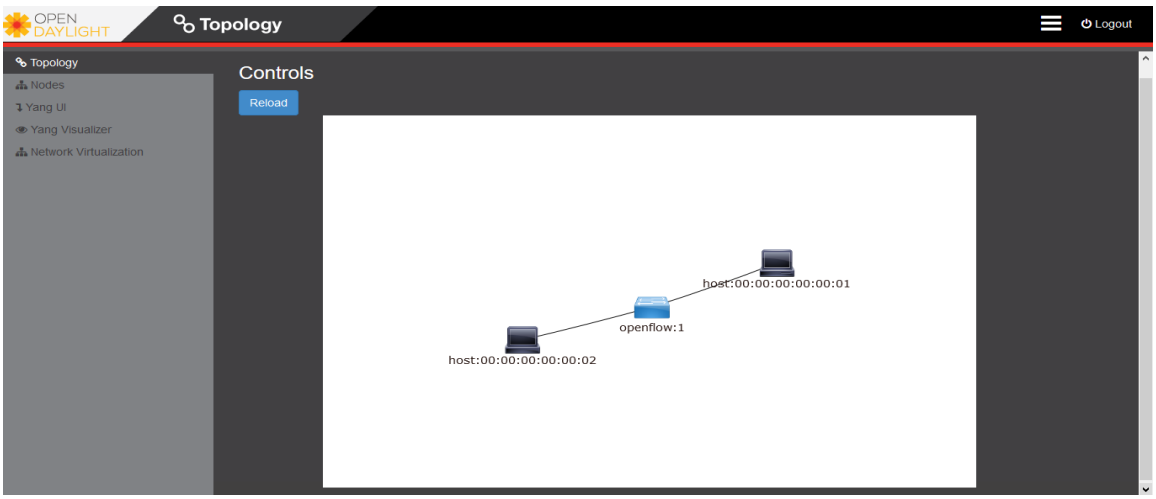

Figura 3: Interfaz gráfica del controlador OpenDayLight

2.5 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 el desarrollo de un agenten OpenFlow-Python que sea capaz de configurar la red de acceso GPON (servicios, perfiles de abonado) en función de parámetros que enviará el controlador en tablas OpenFlow.

2.5.1 Elección y puesta en marcha del controlador

El primer paso fue la elección del controlador a utilizar. Como se ha mencionado anteriormente, existen muchos controladores OpenFlow diferenciados en las versiones de OpenFlow que soportan y los lenguajes de programación en los que están escritos tanto ellos como sus aplicaciones. Entre los controladores en Java, dos de los más utilizados son ONOS [7] y OpenDayLight [6]. Ambos son totalmente abiertos, pero de una gran complejidad debido a su gran potencialidad, flexiblidad y versatilidad. No obstante, nos decantamos por OpenDayLight pese a su complejidad debido a nuestro interés por utilizar tecnologías de las que más se usan actualmente. Tras elegir OpenDayLigth, se instaló y se puso en marcha en una máquina virtual dentro del ordenador desde el que se va a controlar la red. Los detalles de su instalación y configuración inicial se encuentran en otro capítulo posterior de este trabajo.

2.5.2 Tablas OpenFlow y sus parámetros

Con el controlador puesto en marcha, lo siguiente que se haría sería estudiar qué parámetros admite una tabla OpenFlow y como podemos utilizarlos para manipular los servicios, parámetros y pefiles de abonado que se pueden configurar a través del OLT gestinar las diferentes ONUs conectadas a la red GPON.

La clave de OpenFlow está en las tablas OpenFlow que el controlador envía al switch o router. Estas tablas son utilizadas por el switch o router para operar con los diferentes datos que recibe desde cada uno de sus puertos o interfaces.

Cada tabla OpenFlow tiene diferentes entradas *flow*, o flujos. Si algún parámetro de un paquete a procesar coincide con con la sección *match fields* de uno de los *flows* dentro de la tabla, se realiza la acción designada en la sección *instruction* de ese *flow*. Si hay una coincidencia del paquete con varios *flows*, se ejecuta el más prioritario de ellos

gracias a otro campo para designar la prioridad. Hay otros campos, pero los mostrados en la Figura 4 son los más importantes [7].

Figura 4: Campos de un *flow* en el interior de una tabla OpenFlow

El campo que se va a utilizar es *match fields*, ya que admite varios parámetros necesarios para la configuración de servicios en la red de acceso: IP origen/destino, dirección MAC origen/destino y VLAN (*Virtual LAN*). La idea es configurar en el controlador las tablas con los parámetros que nos interesen y recogerlos y gestionarlos en nuestro programa, para proceder a configurar con ellos los diferentes servicios de la red GPON. Utilizando el campo de MAC, podemos identificar la ONU a en la que se configura el servicio. Con el campo IP, se puede saber si es un servicio broadcast o no y configurar un servicio de internet o de multimedia, mientras que con el campo VLAN se puede definir la VLAN que utilizará ese servicio.

Sin embargo, queda un parámetro muy importante por configurar, esto es, el ancho de banda del servicio. Como este parámetro no está disponible entre los *match fields*, la única forma de enviarlo por un mensaje OpenFlow desde el controlador es dentro de una tabla de mediciones, o *Meter Table*. Las tablas de mediciones permiten a OpenFlow contar con operaciones QoS (*Quality of Service*), tales como limitar la tasa de paquetes que están pasando por un determinado *flow* de una tabla OpenFlow. En el campo perteneciente a esta tasa será donde se introduzca el ancho de banda del servicio que se quiera configurar [5].

2.5.3 Programación en Python del agente

Una vez definidos los parámetros y tablas OpenFlow y puesto en marcha el controaldor el siguiente paso será crear un programa que sea capaz de leer los mensajes OpenFlow, sacar los parámetros deseados y pasarlos al OLT mediante el CLI para hacer las configuraciones. El programa debe ejecutarse en una máquina entre el OLT y el controlador OpenFlow, ya que es una especie de “agente traductor”. Este agente corresponde al módulo de control de red (*network control module*) que se muestra en la Figura 5.

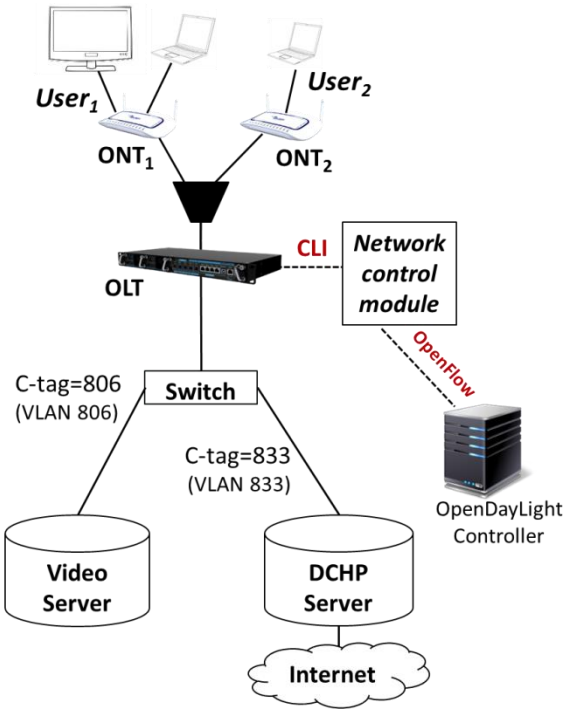

Figura 5: Diseño de un módulo de control de red que interacciona con el controlador y al mismo tiempo con el OLT a través del interfaz CLI

En este sentido, se utilizó Python para escribir el programa agente. La razón es que ya se contaba con un programa en Python capaz de configurar servicios en el OLT con los cuatro parámetros necesarios para ello utilizando comandos en el CLI de configuración de la red, por lo que desarrollar este nuevo programa a partir de ahí ahorraría mucho tiempo.

Como OpenFlow está pensado para utilizarse con switches y routers que soportan dicho protocolo, el primer paso fue emular una comunicación OpenFlow con el controlador haciendo pasar al programa por un switch. Para ello, se analizó la comunicación real entre un switch y el controlador utilizando el programa Wireshark. En Wireshark se observaron los diferentes tipos de mensajes OpenFlow intercambiados y se fueron programando uno a uno con la ayuda de unas librerías que permitían hacerlo de una forma más cómoda, llamadas Python-Openflow, desarrolladas por Kytos SDN [8]. Lo primero que hace el programa es establecer la conexión con el controlador. Para ello, abre una conexión TCP mediante un *socket* en Python a la IP del controlador en su puerto OpenFlow. Una vez la conexión está establecida, el programa entra en un bucle infinito donde va respondiendo al controlador con mensajes acordes a la información que este le pide, creyendo el controlador que se trata de un switch. En algún punto de esta

comunicación, el controlador manda la tabla OpenFlow y la tabla de mediciones a nuestro programa. Tras recoger los datos deseados, se sale del bucle infinito y se configura la red GPON a partir de la información recibida.

2.6 Conclusiones

En este capítulo de la memoria 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, diseñar y programar un agente traductor OpenFlow-Python para configurar nuestra maqueta de red GPON. En concreto, en este capítulo hemos descrito la topología y arquitectura de la red GPON junto con los dispositivos que la conforman. Del mismo modo, se ha descrito el modo de gestión CLI que se utilizará para configurar la red GPON así como sus servicios y perfiles de abonado que se asociarán a las diferentes ONTs/ONUs. A continuación, se describió el estándar OpenFlow y su modo de funcionamiento. En la última parte de la memoria, se describió paso a paso la metodología que se utilizará en este trabajo.

3

Instalación y configuración del controlador OpenDayLight

3.1 Introducción

En este capítulo de la memoria se describirá paso a paso la instalación y configuración del controlador OpenDayLight, uno de los controladores de redes SDN más utilizados en la actualidad y el utilizado en el desarrollo de las tareas de este Trabajo Fin de Grado.

En primer luegar, se analizarán y compararán varios controladores disponibles en la actualidad y más utilizados en redes ópticas. Después se detallará cómo se instaló el controlador OpenDayLigth, cómo se puso en funcionamiento y cómo se configuraron las tablas OpenFlow para gestionar y configurar los servicios en la red de acceso.

3.2 Análisis y comparativa de controladores OpenFlow

NOX [9] fue el primer controlador OpenFlow existente. Está escrito en C++, pero tiene una API para Python también. Actualmente se divide en dos, NOX Classic y New NOX. Dado que NOX Classic está obsoleto y ya no recibe mantenimiento, conviene más trabajar con New NOX. Sin embargo, New NOX solo soporta C++ y tiene menos aplicaciones de red que su versión clásica. Por otro lado, está POX [10], que realmente es una implementación de NOX utilizando solo Python y cuyo principal objetivo es la investigación.

Por otro lado, se encuentran los controladores Beacon [11] y Floodlight [12], implementados en Java [13]. Beacon soporta switches tanto físicos como virtuales, e

implementa operaciones en diferentes hilos o basadas en eventos. Floodlight es un controlador basado en Beacon que tiene el objetivo de soportar una gran variedad de operaciones en la red que sirvan para solucionar problemas que tienen las redes actualmente [14].

A parte de los mencionados anteriormente, existen más controladores, como Ryu [15], OpenDayLight [6] y ONOS (*Open Networking Operating System*) [7].

Ryu es un controlador en Python, al igual que POX. Es un controlador libre y gratuito que soporta todas las versiones de OpenFlow desde 1.0 a 1.5. Sus APIs están bien definidas, y es fácil para desarrolladores crear nuevas aplicaciones de gestión o control de red gracias a ello [15].

OpenDayLight está programado en Java. Debido a que es de código abierto, muchas compañías se han basado en este controlador para realizar sus propios controladores, o aplicaciones compatibles con este controlador. ONOS también está programado en Java, y es competencia directa de OpenDayLight. Actualmente, OpenDayLight está más centrado para su uso en centros de datos, mientras que ONOS parece ser el mejor controlador para WAN (*Wide Area Network*) [16].

Como se mencionó con anterioriorioridad, nos decantamos por OpenDayLight para este trabajo. Nos interesa usar tecnologías de las más utilizadas a nivel global, pese a su complejidad. Sin embargo, la idea de introducir Ryu de forma paralela en un futuro inmediato resultaría de gran interés, pues es relativamente fácil de usar y permite introducir extensiones del protocolo OpenFlow.

3.3 Instalación y configuración del controlador OpenDayLight

El controlador ha sido instalado en una máquina virtual en Virtualbox [17] dentro del ordenador que configurará la red GPON. El ordenadador y el controlador se comunican entre sí mediante una red interna "*Host Only Network*" configurada en Virtualbox. Tanto el ordenador como la máquina virtual utilizan el sistema operativo Linux Mint [18], una distribución de Linux basada en Ubuntu.

Desde la máquina virtual se descargó una de las versiones disponibles del controlador, Boron-SR3 [19]. Para instalarlo, bastó con extraerlo y ejecutar el fichero */bin/karaf* en el interior del directorio extraido del fichero descargado, tal y como se muestra en la Figura 6.

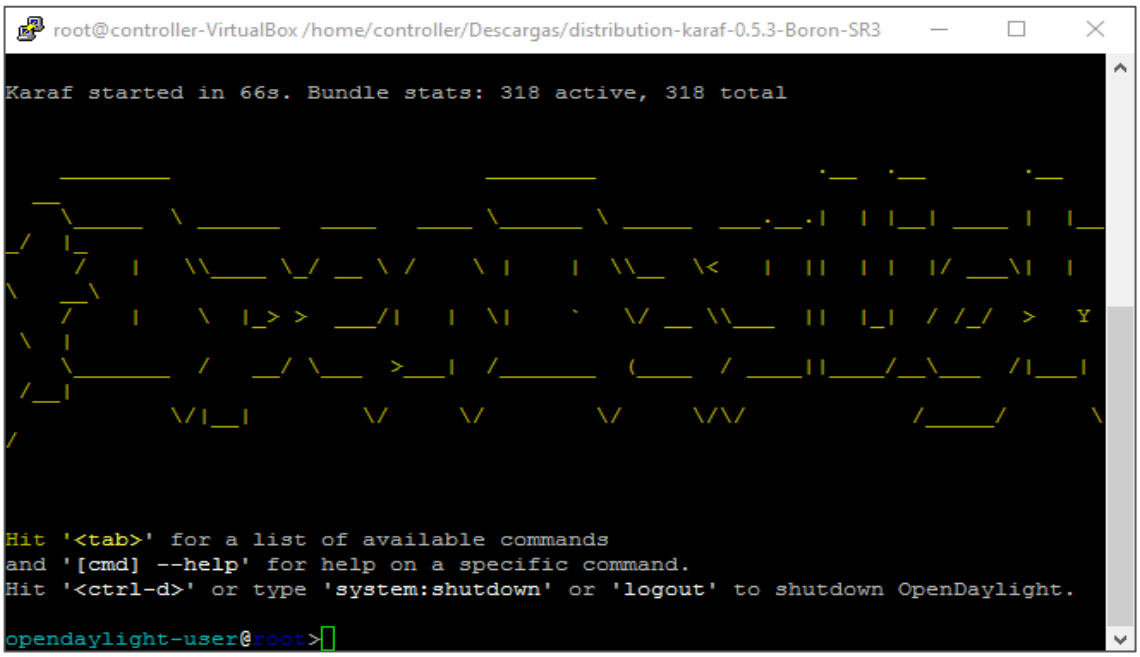

Figura 6: Pantalla de inicio del controlador OpenDayLight

Una vez ejecutado el controlador, lo primero que hay que hacer es instalar diferentes aplicaciones o *features* para probar que funciona correctamente. Las aplicaciones que se van a instalar son las siguientes:

- **odl-restconf**: permite acceder a la API de RESTCONF. El protocolo RESTCONF permite utilizar aplicaciones web para tener acceso a datos en el interior de un elemento de la red [20].
- **odl-mdsal-apidocs**: permite acceder a la API de YANG. Los datos a los que se accede desde RESTCONF están definidos en YANG, un lenguaje de modelo de datos para el protocolo de configuración de red NETCONF (*Network Configuration Protocol*) [21].
- **odl-l2switch-switch**: sirve para que el controlador convierta al switch en un switch Ethernet, de forma que sea capaz de reconocer los hosts conectados en sus puertos y configurar conexiones entre ellos de ser

necesario. Esta aplicación ha sido activada para realizar unas pequeñas pruebas con el controlador y Mininet.

- **old-dlux-all**: interfaz gráfica web para utilizar el controlador de forma cómoda.

Para instalar las aplicaciones, se utiliza el comando "*feature:install*" en el CLI del controlador, seguido de todas las aplicaciones que se quieran instalar. En este caso, se introduciría el comando "*feature:install odl-restconf odl-l2switch-switch odl-mdsal-apidocs odl-dlux-all***"**

Con el controlador instalado y configurado, se utilizó otra máquina virtual para crear una red virtual en Mininet con un switch OpenFlow al que están conectados dos hosts. Dentro de Mininet, se configuró este switch para que usara el controlador OpenDayLight. Para configurar la red virtual Mininet, se han seguido los siguientes pasos:

1) Se ejecuta una máquina virtual Mininet en Virtualbox, configurada para que también esté dentro de la "*Host Only Netowork*" y así se pueda comunicar con el controlador.
2) Dentro en el CLI de Mininet, mostrado en la Figura 7, se ejecuta el comando "*sudo mn --mac --controller=remote,ip=192.168.56.101,port=6633 --switch ovs,protocols=OpenFlow13*", siendo la IP 192.168.56.101 la IP del controlador OpenDayLight, y el puerto 6633 el puerto al que debe conectarse el switch para iniciar la comunicación OpenFlow. El parámetro "*protocols=OpenFlow13*" es para especificar que se usará la versión OpenFlow 1.3.

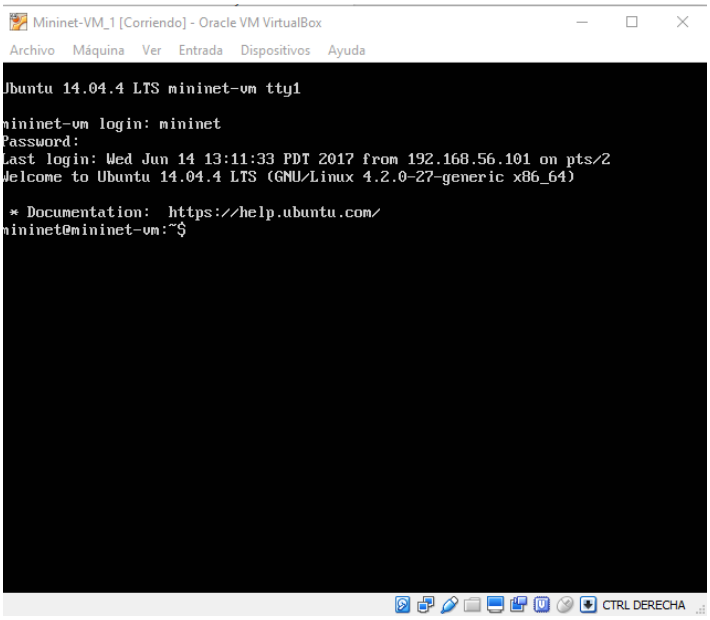

Figura 7: CLI de la máquina virtual Mininet

Tras esto, se accedió a la interfaz web del controlador. Para acceder a ella, basta con poner la URL http://192.168.56.101:8181/index.html en el navegador de un ordenador conectado a la misma subred que el controlador, y escribir el usuario "*admin*" y la contraseña "admin". Tras esto, se llega a lo que se muestra en la Figura 8.

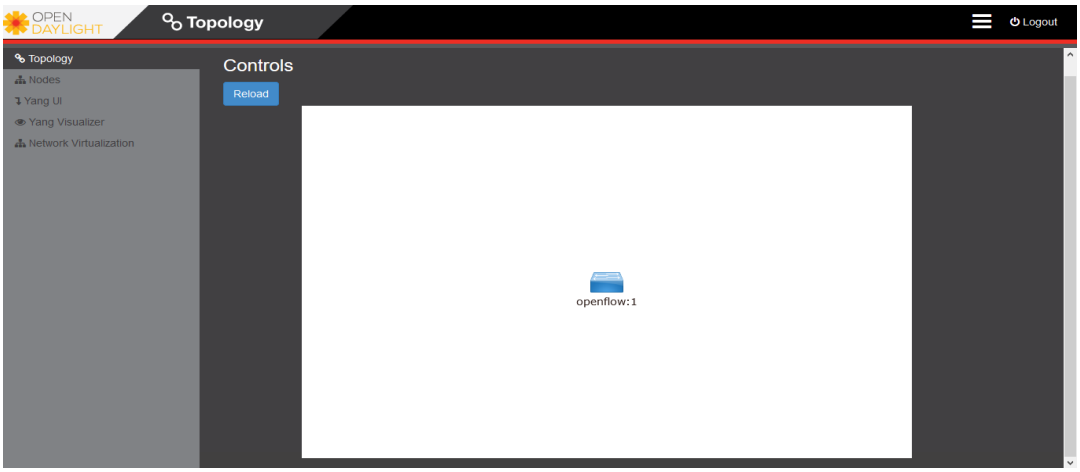

Figura 8: switch de la red virtual reconocido por el controlador

Como se aprecia en la Figura 8, el controlador fue capaz de reconocer al switch OpenFlow de la red virtual en Mininet. Sin embargo, como no ha habido tráfico entre los hosts conectados al switch, el controlador no es capaz de detectarlos. Si se hacen varios pings cruzados entre los hosts, el switch detectará tráfico que quiere ir de un host a otro, pero que no puede debido a la falta de configuración de tablas OpenFlow en el interior

del switch. Esto provocará la activación de la funcionalidad "*l2switch*", que hará que se manden tablas OpenFlow para comunicar los hosts entre sí al switch y además hará que el controlador sepa de la existencia de esos hosts, tal y como se observa en la Figura 9.

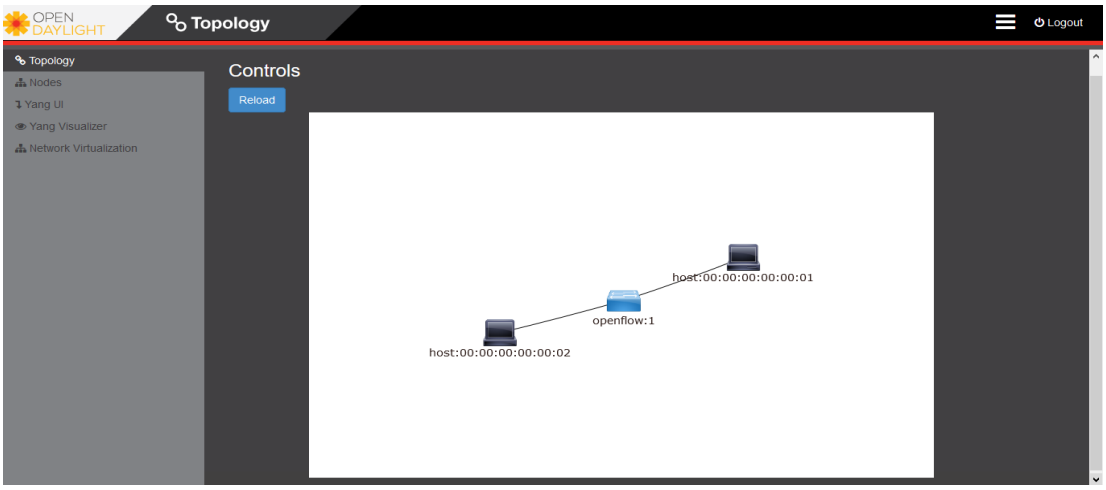

Figura 9: red virtual reconocida por el controlador

3.4 Tablas OpenFlow. Visualización y configuración

3.4.1 Visualización de las tablas creadas por l2switch

Una vez instalado y configurado nuestro controlador, el siguiente paso será visualizar las tablas y *flows* creados por *l2switch*. Para ello, primero hay que acceder a la sección "*Yang UI*" de la Figura 10, en la sección izquierda de dicha pantalla.

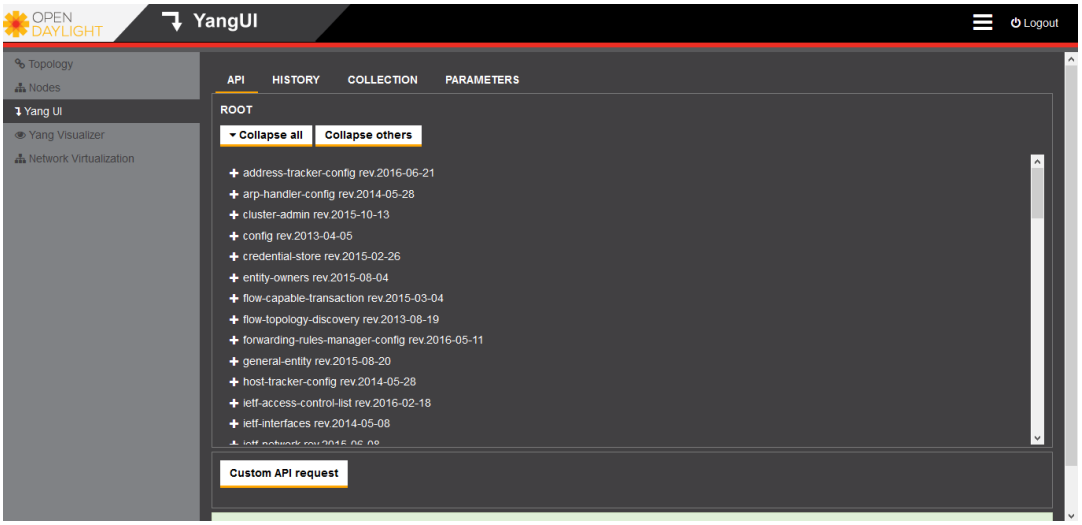

Figura 10: Sección *Yang UI* de la interfaz web del controlador

Yang UI es una interfaz gráfica que permite acceder a datos definidos en YANG utilizando métodos HTTP GET, o introducirlos usando HTTP PUT. Para acceder a las tablas y flows OpenFlow, primero hay que irse a la sección "*opendaylight-inventory*" de *Yang UI*. Esta sección es parte de la lista de secciones mostradas en la Figura 10. Una vez se haya encontrado, se hace click en ella y se desplegará el menú mostrado en la Figura 11.

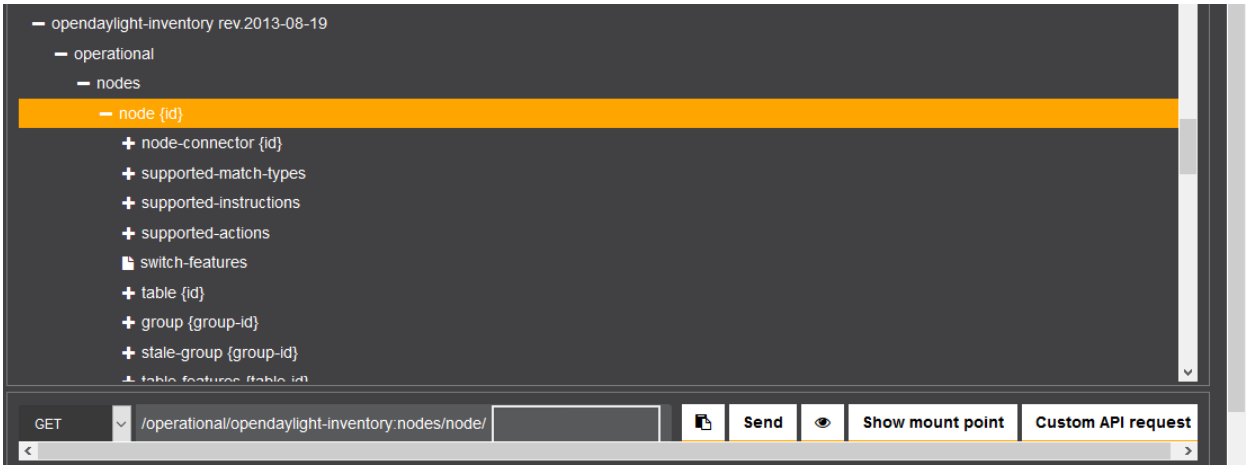

Figura 11: Sección *inventory* dentro de *Yang UI*

Como se puede observar en la Figura 11, los apartados dentro de "*opendaylight-inventory*" se van desplegando en forma de árbol, y en la parte inferior de la pantalla aparece una pestaña para elegir métodos http (GET activo en la Figura 11) seguida de una URL, que puede completarse con parámetros que dependen del nivel en el que nos encontremos. Si se selecciona el método GET en el selector de métodos de la esquina inferior izquierda, en el recuadro que completa la URL se escribe el ID del nodo donde se quieren ver las tablas "*openflow:1*", y se hace click en el botón "*Send*" (a la derecha de la URL), será como hacer un GET a "*/operational/opendaylight-inventory:nodes/node/openflow:1*", y debajo se mostrarán los datos tal y como aparecen en la Figura 12.

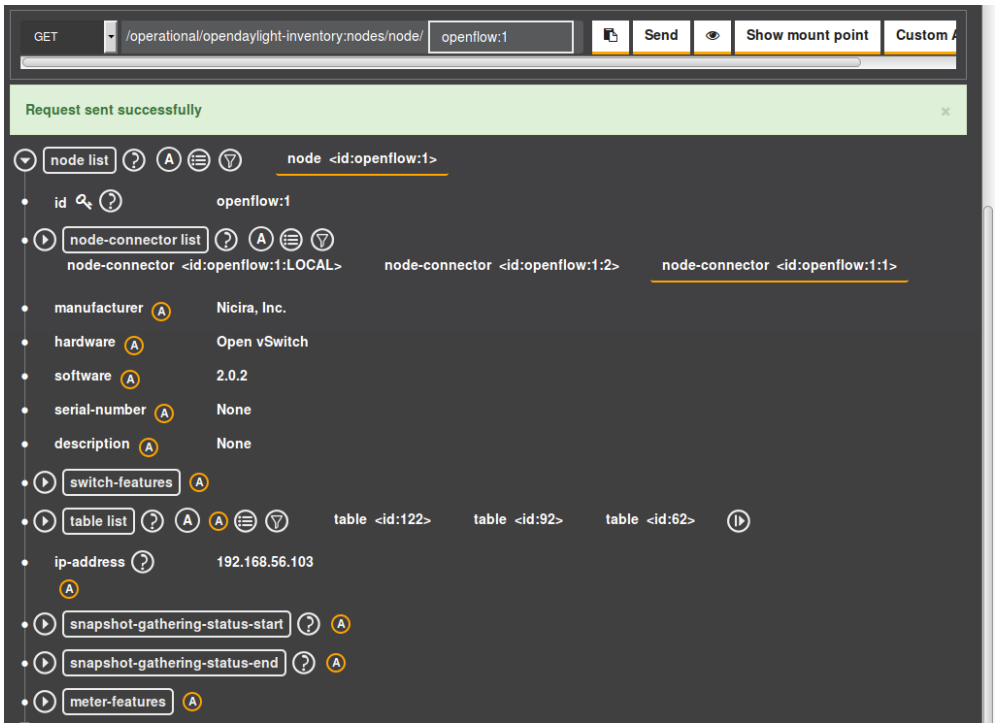

Figura 12: Datos mostrados sobre el switch *openflow:1*

Uno de los submenús de *inventory* mostrados la Figura 11 es *table {id}*. Si se pincha en él, se selecciona el método GET, y se rellena la URL que aparecerá de tal forma que sea "*/operational/opendaylight-inventory:nodes/node/openflow:1/flow-node-inventory:table/0"* (ID del nodo *openflow:1* e ID de la tabla 0, que es la que edita *l2switch* por defecto), se obtendrá lo que se muestra en la Figura 13.

Como se puede puede observar en la Figura 13, se ha creado una tabla con ID 0 que contiene varios *flows*. Uno de ellos dice que, si entra algo que tiene como MAC origen 00:00:00:00:00:01 y como MAC destino 00:00:00:00:00:02, debe salir por el puerto 2.

La MAC origen se muestra en el apartado "*ethernet-source*", mientras que la MAC destino se muestra en el apartado "*ethernet-destination*". El puerto de salida aparece en el apartado *instructions*, que es donde se designa lo que le ocurrirá al paquete cuya MAC de origen y destino coincidan con los citados anteriormente. Ahí, pasando por "*instruction-list*", "*apply-actions*" y "*action-list*", se puede leer "*Action output-action-case"*. Esto significa que el tipo de acción sobre el paquete será sacarlo por algún puerto del switch. El puerto aparece debajo, en la sección "*output-action*", concretamente en "*output-node-connector*", en este caso será el puerto 2.

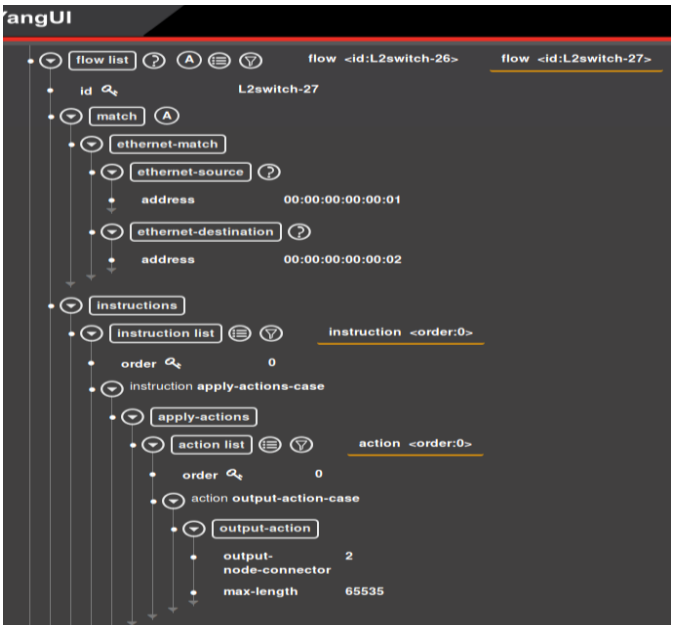

Figura 13: Primer *flow* creado por l2switch

Por el otro lado, el otro flujo funciona del revés, es decir, si llega algo que tiene MAC origen 00:00:00:00:00:02 (etiqueta "*ethernet-source*") y MAC destino 00:00:00:00:00:01 (etiqueta "*ethernet-destination*"), debe salir por el puerto 1, tal y como se muestra en la Figura 14 (resto de parámetros iguales a los descritos anteriormente).

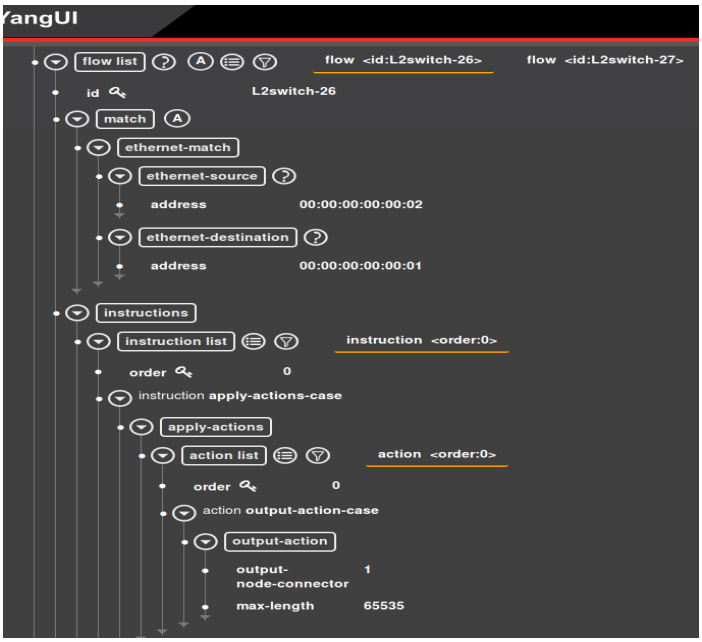

Figura 14: Segundo flujo creado por *l2switch*

De esta forma, se ha implementado la comunicación entre los dos hosts conectados al switch en Mininet.

3.4.2 Creación manual de tablas OpenFlow

Por otro lado, a nosotros nos interesa poder hacer las tablas de forma manual, ya que tenemos que diseñar los parámetros insertados en la tabla de acuerdo a los parámetros que van a configurar en la red GPON para hacerla operativa y funcional. Para ello, basta con acceder a la sección de *Yang UI* perteneciente a "*/config/opendaylight-inventory:nodes/node/nodeID/flow-node-inventory:table/tableID*", donde *nodeID* sería la ID del nodo en el que se quieren meter tablas nuevas, del tipo *openflow:1* y donde *tableID* sería un entero que indica la ID de la tabla que se va a editar. Para llegar a esa sección, habría que abrir submenús desde *inventory* de una forma similar a la forma de llegar las secciones descritas en el Apartado 3.4.1. Si vamos, por ejemplo, a "*/config/opendaylight-inventory:nodes/node/openflow:2/flow-node-inventory:table/0*", aparece la captura que se muestra en la Figura 15.

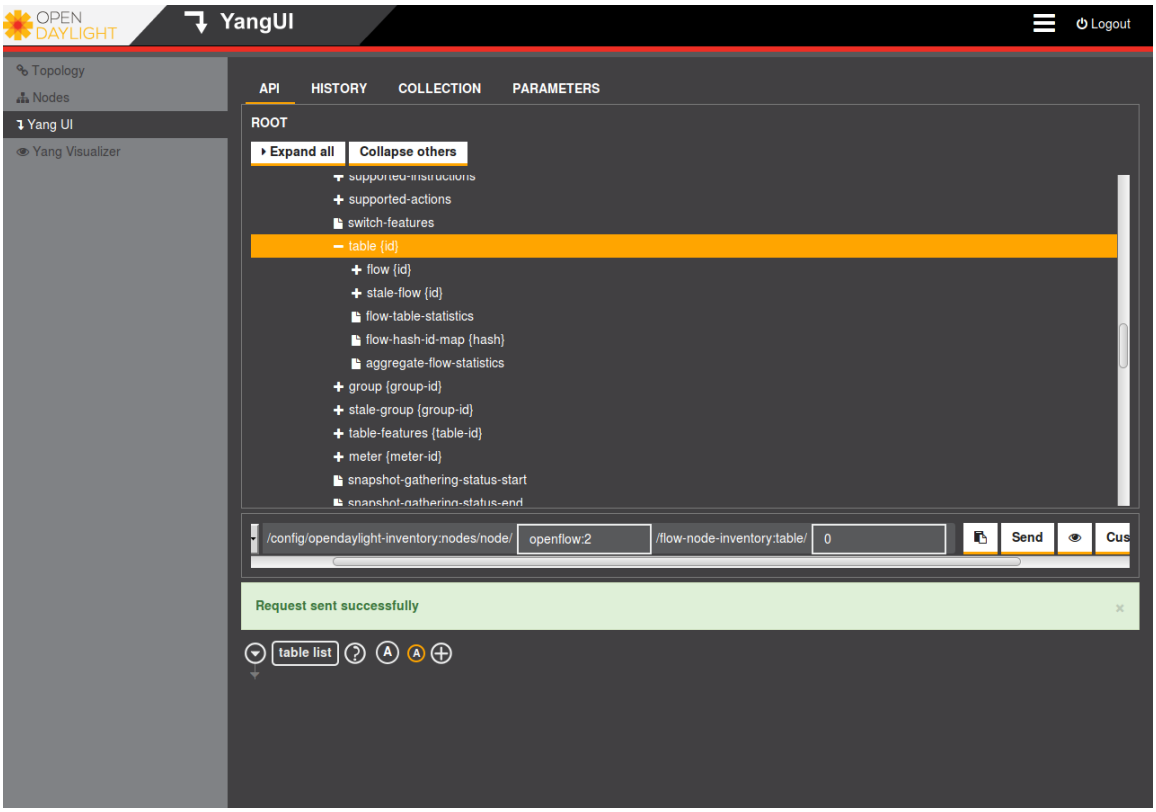

Figura 15: Tabla OpenFlow inicialmente vacía

Como se puede observar en la Figura 15, la tabla aparece vacía. Para empezar a rellenarla, hay que hacer click en la pestaña "*add list ítem*", cuyo símbolo es un "+", como se muestra en la Figura 16 resaltado en naranja en la parte superior de la captura.

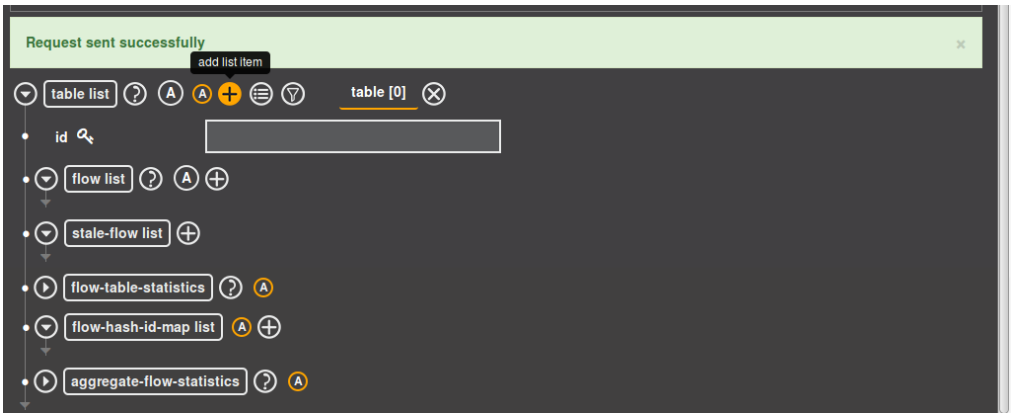

Figura 16: Ejemplo de cómo añadir una tabla nueva con diferentes parámetros

Tras rellenar el campo "*id*" con un número de identificación, habría que desplegar otro "*add ítem list*" en "*flow list*" (pulsando el botón + a la derecha de la etiqueta "*flow lists*") para comenzar a añadir *flows* a esta tabla, como se muestra en la Figura 17.

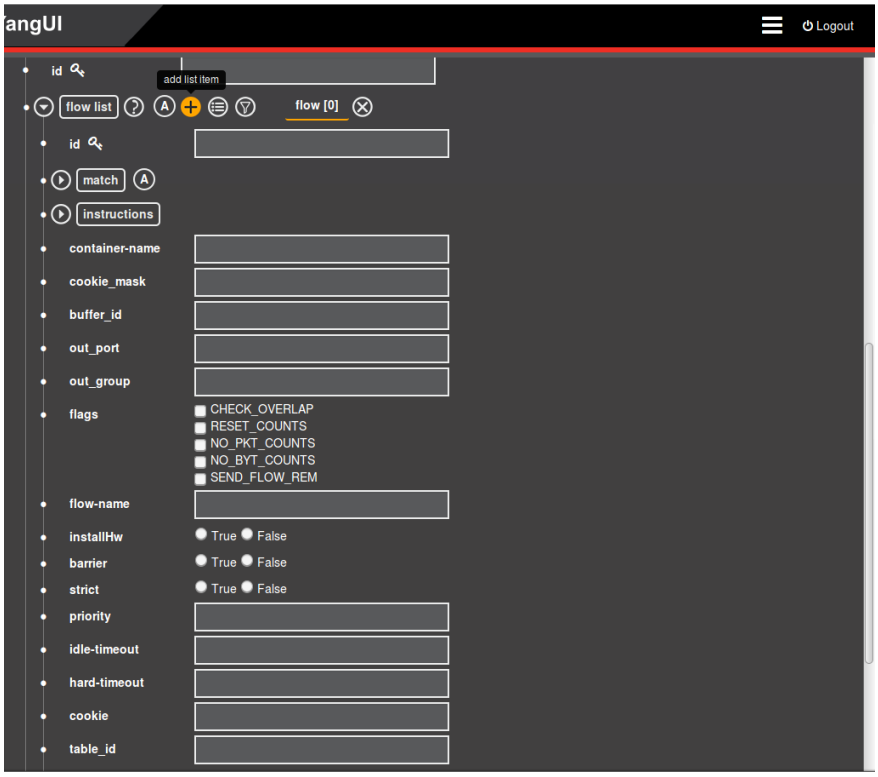

Figura 17: Campos vacíos de un *flow*

De todos los campos que hay, solo son interesantes para este trabajo *match* e *instructions*. En la Figura 18 se pueden observar todos los campos disponibles en *match*, donde los más importantes son:

- Campo *in-port*: se refiere al puerto por el que ha entrado el paquete al switch.
- Campo *ethernet-match*: se refiere a la dirección MAC de origen o la dirección MAC de destino (esto se vería al desplegar el campo).
- Campo *vlan-match*: se refiere a la etiqueta VLAN del paquete.
- Campo *layer-3-match*: si en la pestaña que sigue a este campo se selecciona "*ip-v4-match*", permite hacer una coincidencia con la IP de origen o de destino de un paquete.

Figura 18: Campos *match* de la tabla OpenFlow creada

En cuanto a los campos en el interior de *instructions*, pulsando el botón "+" que hay al lado de *instruction list* (Figura 17), se pueden añadir diferentes instrucciones que se aplicarían al paquete cuyos campos coincidieran con los designados en *match*. El tipo de instrucciones disponibles se muestra en la Figura 19.

Figura 19: Tipos de instrucciones que se pueden añadir a la tabla OpenFlow creada

Por ejemplo, si se selecciona la instrucción "*apply-actions-case*", se aplican inmediatamente las instrucciones descritas en ese *flow*, que aparecerían en un menú en la parte inmediatamente inferior. Sin embargo, mediante la instrucción "*go-to-table-case*" el paquete sería enviado a otra tabla OpenFlow (tubería de tablas OpenFlow). Otra opción importante es "*meter-case*", para asociar una ID de una tabla de mediciones con el *flow*.

En una "*instruction-list*" solo puede haber uns instrucción de cada tipo. Además, por cada instrucción dentro de "*instruction-list",* se puede añadir un orden de ejecución en el campo "*order*", que también sale en la Figura 19. Este orden, sin embargo, debe de tener en cuenta ciertas restricciones, como que la instrución "*meter-case*" debe aplicarse antes que la "*apply-action-case*", la instrución "*clear-actions-case*" se debe aplicar antes que la "*write-action-case*", y que la "*go-to-table-clase*" debe de ser la última en ejecutarse [5].

En el caso de las tablas que se utilizarán para configurar servicios y perfiles en el OLT, hay que recordar que, en realidad, los *flows* que se van a utilizar para este trabajo van a servir simplemente como contenedores de datos de los que se sacarán los parámetros para configurar la red de acceso GPON. Estos parámetros de configuración de la red se han metido en los campos más similares a ellos existentes en *flows* de tablas OpenFlow. Después dichos *flows* serán enviados mediante mensajes OpeFlow a un programa que sacará de ellos la información necesaria y configurará servicios en el OLT. Esto quiere decir que en ningún momento se utilizarán para su cometido real, que es el ejecutar unas *instructions* con los paquetes en los que haya campos que coincidan con los designados en *match*.

Con esto en mente, se comenzará con el campo *match*. En la Figura 20 se muestra un ejemplo del uso que se le dará a este campo. Lo que se muestra en dicha captura es el caso de configuración de un servicio de internet en una ONU concreta. En "*in-port*" se ha escrito "*openflow:5:1*", que hace referencia al puerto 1 de un nodo llamado "*openflow:5*". El campo de *in-port* se ha utilizado porque si se dejaba en blanco, la tabla no se creaba correctamente. El nodo se denomina *openflow:5* porque el agente diseñado se programó para tener ese *datapath_id*, pero en realidad podría haber sido cualquier número (se explicará en el Capítulo 4 qué es *datapath_id*). Para la dirección MAC de la ONU en la que se configurará el servicio, se ha utilizado el campo "*ethernet-destination"*, que está en el interior de "*ethernet-source"*. Para distinguir entre servicio de internet y servicio de internet y vídeo, se ha utilizado el campo "*ipv4-destination"* que se encuentra en "*layer-3-match-ipv4-match***"**. Si se deja vacío, es un servicio de internet. Si se pone una con IP acabada de 255 (broadcast) sería de internet y vídeo, tal y como se muestra en la Figura 21. Para la VLAN asociada al servicio, se utiliza el campo "*vlan-id"*.

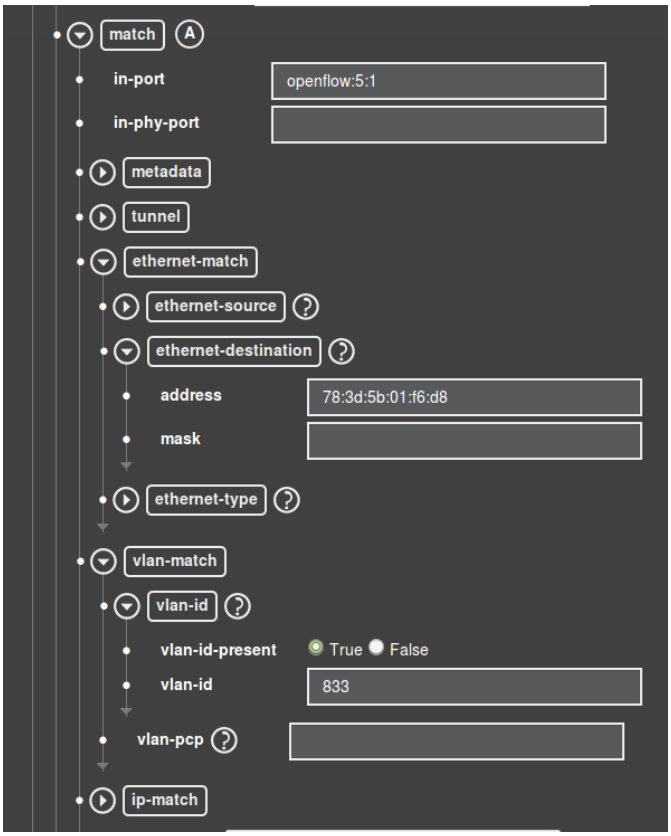

Figura 20: Ejemplo de tabla para configurar un servicio de Internet asociado a una ONU

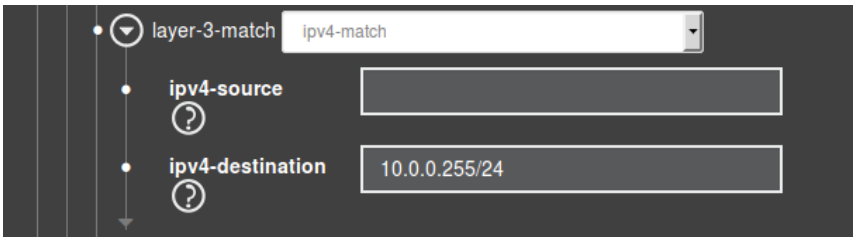

Figura 21: Ejemplo de tabla para la creación de un servicio de internet+vídeo asociado a una ONU

Por último, queda configurar las tablas de mediciones, o *meter tables.* Estas tablas en realidad sirven para limitar la tasa de paquetes que están pasando por un *flow*, pero como tienen un campo donde colocar la tasa a la que se quiere limitar, lo vamos a utilizar para mandar el ancho de banda máximo al que se configurará el servicio (en kbps). Para utilizar una tabla de mediciones asociada a un flujo (*flow*), hay que seleccionar en instruction "*meter-case*", y escribir en "*meter-id"* el ID de la tabla de mediciones donde se va a colocar el ancho de banda.

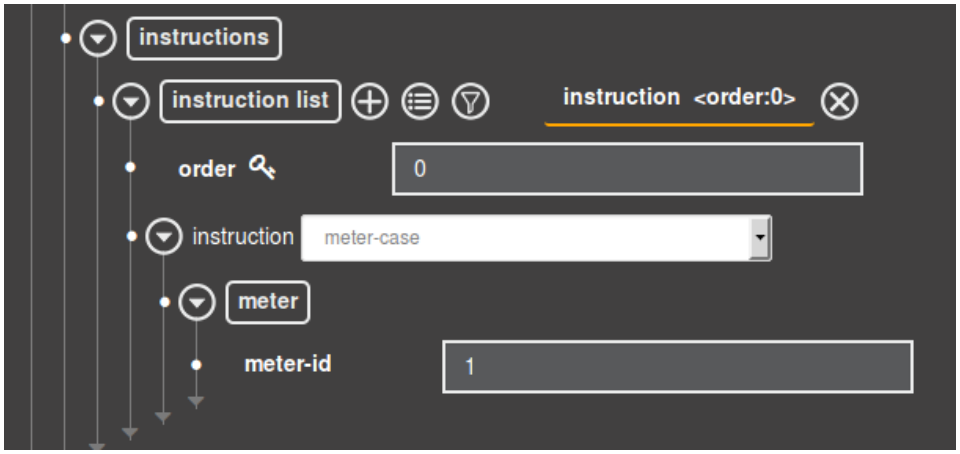

Figura 22: Asignación de una tabla de medición a un *flow* asociado a un servicio

Para hacer la tabla de mediciones, lo primero que hay que hacer es navegar por *inventory* hasta el menú perteneciente a *"/config/opendaylight-inventory:nodes/node/nodeID/flow-node-inventory:meter/meterID*", siendo meterID el ID designado en la Figura 22. Es importante mencionar que, en principio, las tablas de mediciones son únicas para cada servicio. Sin embargo, diferentes servicios pueden utilizar la misma tabla si se configuran con los mismos anchos de banda. Además, se van a utilizar dos tablas de mediciones por servicio. En una irá el ancho de banda del *downstream*, y en la otra el ancho de banda del *upstream*, con IDs 1 y 2 respectivamente.

En la Figura 23 se muestra un *meter* configurado satisfactoriamente, en este caso el del *downstream*. El único campo realmente útil para lo que se ha hecho es "*drop-rate*", donde se ha introducido el ancho de banda *downstream* que se quiere configurar para el servicio. Este campo, en realidad, sirve para designar a qué tasa se va a activar el funcionamiento de la tabla de mediciones. En este caso, lo que haría sería descartar paquetes a partir de una tasa de 40000 kbps (40 Mbps), es decir, cuando se alcanza una tasa pasando por el *flow* de 40 Mbps. Las unidades de "*band-rate*" y "*drop-rate*" se designan en el campo "*flags*", y la ID se coloca en "*meter-id*". El resto de parámetros que se muestran carecen de importancia y solo están configurados con el objetivo de que la tabla de mediciones se envíe desde el controlador al agente programado de forma correcta.

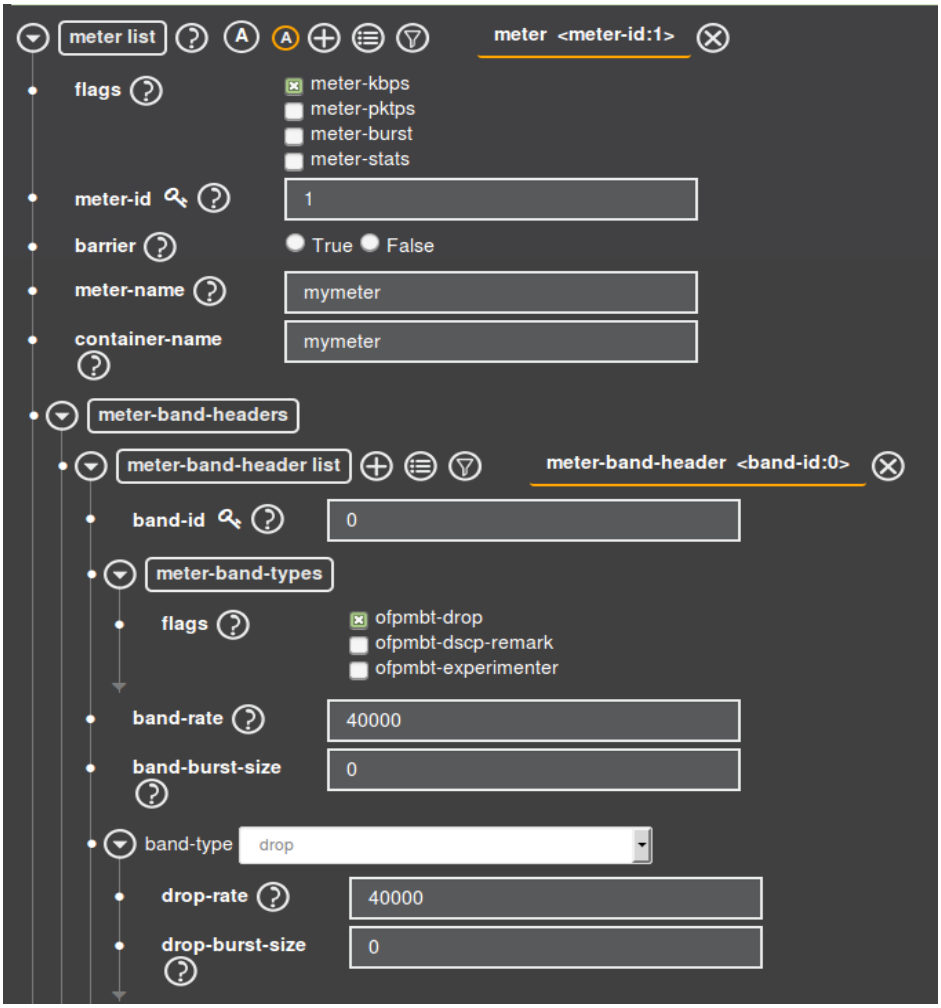

Figura 23: Configuración de una tabla de medición

El *meter* que correspondería al tráfico de subida (*upstream*) para ese mismo *flow* es exactamente igual, pero con ID igual a 2. Para designar el ancho de banda *upstream*, se utilizarían los mismos campos, aunque con distintos o iguales valores. Una vez se tienen los anchos de banda *downstream* y *upstream*, el 90% de los valores introducidos serán configurado como garantizados en ambos canales y el 10% como exceso en la red de acceso GPON para ese servicio configurado. Cabe destacar que nosotros hemos hecho esa consideración en la distribución del ancho de banda, pero podría plantearse alguna otra más optimizada.

3.5 Conclusiones

En este capítulo se ha descrito cómo se ha instalado y configurado el controlador OpenDayLight una vez que se ha hecho una comparativa de los controladores más populares existentes en el mercado. También se ha mostrado como funciona la aplicación del controlador *l2switch* para ejemplificar un funcionamiento normal de OpenFlow, y se ha descrito como se utilizarían las tablas y f*lows* OpenFlow para configurar nuestra red GPON de forma más concreta. El siguiente y último paso será mandar todos estos datos mediante mensajes OpenFlow, y programar un agente que sea capaz de entender dicho estándar de comunicación, desencapsular dicha información y pasar estos datos al lenguaje CLI que entiende el OLT para configurar los servicios y perfiles asociados a los usuarios finales (ONUs) de la red GPON.

4

Diseño de un agente OpenFlow en Python

4.1 Introducción

El capítulo anterior de esta memoria trataba sobre el controlador OpenFlow, tanto su instalación como su configuración e inserción de los parámetros de los servicios y perfiles de abonado a configurar en la red GPON a través del OLT. El último paso de este Trabajo Fin de Grado y descrito en este capítulo, es conseguir que el controlador envié estos parámetros a un agente OpenFlow-Python que hará de puente entre el controlador y el OLT mediante el protocolo Openflow, de forma que desde el controlador se puedan configurar la red de acceso de forma dinámica e inteligente usando dicho protocolo de comunicación.

4.2 Análisis de una comunicación OpenFlow entre el agente programado y el controlador

Como se ha mencionado anteriormente, OpenFlow está pensado para funcionar con switches o routers, por lo que, ¿cómo se comunicará el controlador con el agente programado? La solución que se ha llevado a cabo es la siguiente, el programa iniciará y mantendrá una comunicación OpenFlow emulada similar a la que tendrían un switch OpenFlow y un controlador. Dicho de otra forma, se va a "engañar" al controlador para que crea que el agente es uno más de los numerosos swithches que se van a conectar a él.

Para lograr este objetivo, lo primero que se ha hecho ha sido analizar la comunicación OpenFlow y los mensajes intercambiados entre el controlador y un switch

virtual en Mininet. Para capturar los paquetes que se envían entre sí, se ha utilizado la herramienta Wireshark. La Figura 24 muestra el inicio de la comunicación.

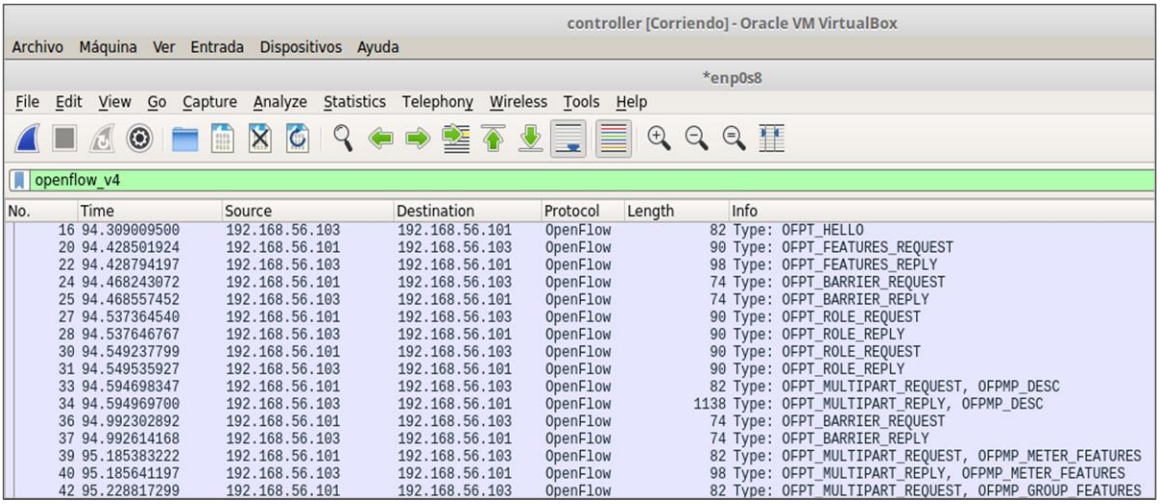

Figura 24: Inicio de la comunicación OpenFlow entre el switch de Mininet y el controlador OpenDayLight

La IP 192.168.56.101 corresponde al controlador, mientras que la IP 192.168.56.103 corresponde a la máquina virtual donde se está alojando Mininet y desde donde los componentes de la red virtual se comunican al exterior. En las siguientes secciones se explicarán con detalle todos los mensajes OpenFlow intercambiados entre ambos.

4.2.1 OFTP_HELLO

La comunicación comienza con un mensaje OFPT_HELLO, enviado desde el switch. En este mensaje se envía una lista de las versiones de OpenFlow que soporta el switch. Como el switch está configurado para usar únicamente la versión 1.3, listada como la versión número 4 del protocolo [5], el *bitmap* enviado es “00000000 00000000 00000000 00010000”, o “0x00000010” en hexadecimal. Es decir, ha puesto a “1” el bit menos significativo número 4, numerando los 32 bits de 0 a 31.

El controlador responde con otro mensaje OFPT_HELLO. Este mensaje no se ve en la Figura 24 porque a veces lo manda junto al siguiente mensaje, OFPT_FEATURES_REQUEST. Sin embargo, dicho mensaje puede verse pinchando en el segundo mensaje que se muestra en la Figura 24, como se observa en la Figura 25.

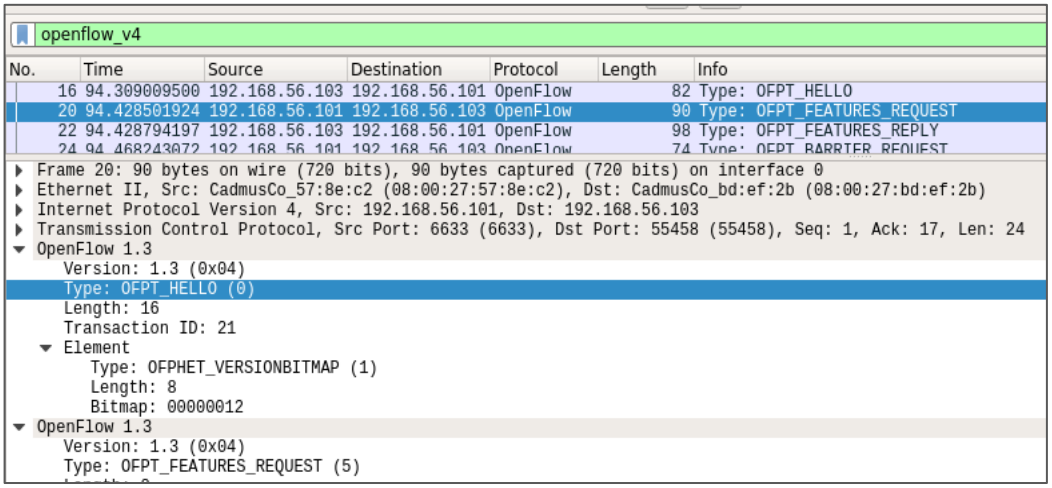

Figura 25: Mensaje OFPT_HELLO mandado por el controlador

Además, como se aprecia en la Figura 25, el bitmap enviado por el controlador es "0x00000012", es decir, "00000000 00000000 00000000 00010010" en binario. Esto se debe a que el controlador soporta tanto la versión 1.0, listada como la número 1, como la versión 1.3, listada como la número 4 [5]. Por lo tanto, se ponen a "1" los bits menos significativos 1 y 4, siendo la numeración de 0 a 31.

Como tanto el controlador como el switch OpenFlow soportan la versión de OpenFlow 1.3, el resto de la comunicación se hará utilizando esta versión.

4.2.2 OFPT_FEATURES_REQUEST y OFPT_FEATURES_REPLY

Tras acordar la versión entre ambos usando los OFPT_HELLO, el controlador envía un mensaje del tipo OFPT_FEATURES_REQUEST. En este mensaje, el controlador le pregunta al switch qué capacidades tiene. A este mensaje, el switch responde con un mensaje OFPT_FEATURES_REPLY.

El contenido de este mensaje se puede ver en la Figura 26. Los primeros campos son la versión, el tipo de mensaje OpenFlow, la longitud del mensaje (en bytes) y la *Transaction ID* del mensaje, que debe coincidir tanto en la petición como en la respuesta. En este caso vale 2 porque el mensaje OFPT_FEATURES_REQUEST tenía una *Transaction ID* de valor 2. Estos campos existen en todos los mensajes OpenFlow, por ejemplo, en la Figura 25 se puede ver cómo existen también en el OFPT_HELLO que se muestra.

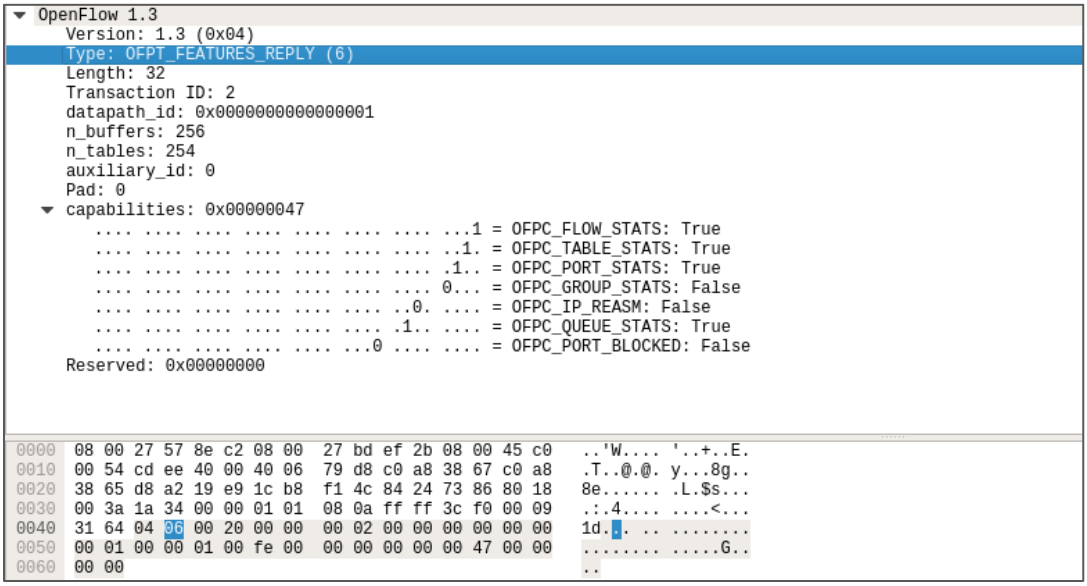

Figura 26: Contenido del mensaje OFPT_FEATURES_REPLY

Los siguientes campos ya son específicos del tipo de mensaje OpenFlow. El primero es *datapath_id*, o ID del camino de datos. Esta es la ID del switch, y la razón de que en la Figura 8 el switch virtual se llame OpenFlow:1, ya que su valor es "0x0000000000000001" [5].

A continuación, se encuentran los campos *n_buffers* y *n_tables*. El primero sirve para indicar el máximo número de paquetes a la vez admitidos en el *buffer* del switch, que son 256 para este ejemplo. El segundo es el máximo número de tablas admitidias por el switch, que en este caso son 254 [5].

El campo *auxiliary_id* indica el tipo de conexión entre switch y controlador. Una conexión principal fija este valor a 0, como es este caso, pero si se trata de una conexión auxiliar el valor debe de ser diferente a 0. Las conexiones auxiliares pueden ser creadas por el switch OpenFlow y son útiles para mejorar su rendimiento de procesado aprovechando los procesamientos en paralelo implementados en la mayoría de los switches [5].

Por último, el campo *capabilities* indica las capacidades del switch en función de los bits que se pongan a "1", de una forma similar al bitmap con las versiones que se manda en los OFPT_HELLO. Las capacidades disponibles pueden ser diferentes estadísticas de *flows*, tablas, puertos, grupos y colas, si se pueden reensamblar mensajes

fragmentados y si el switch puede o no bloquear puertos para prevenir paquetes siendo enviados en *loop* [5].

4.2.3 OFPT_BARRIER_REQUEST y OFPT_BARRIER_REPLY

El controlador manda mensajes de tipo OFPT_BARRIER_REQUEST de forma habitual para asegurarse de que el switch ha procesado correctamente los mensajes enviados con anterioridad. Cuando el switch recibe un OFPT_BARRIER_REQUEST, termina el procesamiento de los mensajes enviados por el controlador con anterioridad y responde con un mensaje OFPT_BARRIER_REPLY [5].

Estos mensajes son muy sencillos, ya que como se puede ver en las Figura 27 y 28. solo constan de los campos mínimos que debe tener un mensaje OpenFlow.

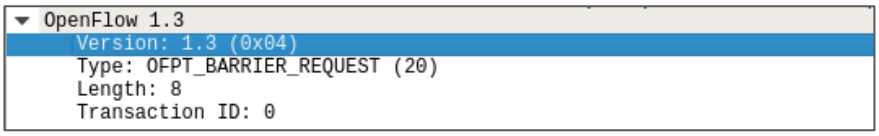

Figura 27: Campos del mensaje OFPT_BARRIER_REQUEST

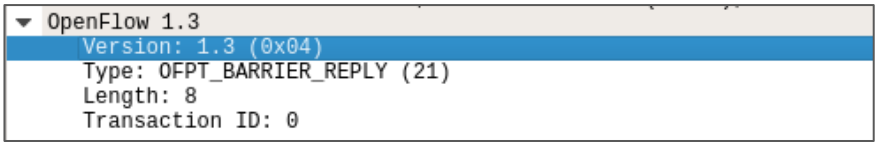

Figura 28: Campos del mensaje OFPT_BARRIER_REPLY

4.2.4 OFPT_ROLE_REQUEST y OFPT_ROLE_REPLY

Cuando el controlador quiere cambiar su rol, manda un mensaje OFPT_ROLE_REQUEST. Estos mensajes tienen un campo que, dependiendo de su valor, indican cual es el nuevo rol que quiere tomar el controlador. Los valores pueden ser los siguientes:

- OFPCR_ROLE_NOCHANGE: este caso se da cuando el campo perteneciente al rol vale 0. Indica que el controlador no va a cambiar su rol actual. Sirve para que el controlador pueda hacer peticiones de su propio rol [5].

- OFPCR_ROLE_EQUAL: este caso se da cuando el campo perteneciente al rol vale 1. Es el modo por defecto y el controlador tiene permiso de lectura y escritura en el switch o switches [5].
- OFPCR_ROLE_MASTER: este caso se da cuando el campo perteneciente al rol vale 2. Si hay varios controladores en la red, solo uno de ellos puede tener este rol. Al igual que OFPCR_ROLE_EQUAL, con este rol se tiene acceso de lectura y escritura en el switch o switches [5].
- OFPCR_ROLE_SLAVE: este caso se da cuando el campo perteneciente al rol vale 3. Si hay varios controladores en la red, los que no tengan el rol OFPCR_ROLE_MASTER deben tener este rol. Si un controlador pasa de esclavo a maestro, el maestro anterior debe pasar a ser esclavo. Un controlador con el rol esclavo solo tiene acceso de lectura al switch o switches [5].

Tras recibir un OFPT_ROLE_REQUEST, el switch debe responder con un OFPT_ROLE_REPLY. Dentro del campo del rol del OFPT_ROLE_REPLY se confirma el rol actual del controlador tras haber enviado el OFPT_ROLE_REQUEST [5].

Además, tanto en OFPT_ROLE_REQUEST como en OFPT_ROLE_REPLY existe un campo más llamado *generation_id.* Si algún controlador pretende cambiar de maestro a esclavo o viceversa, se genera este identificador. Cuando el switch recibe el OFPT_ROLE_REQUEST, valida el identificador para ver si es mayor o menor que el identificador recibido en una petición anterior. Si es mayor no está obsoleto, manda el OFPT_ROLE_REPLY con el mismo *generation_id* y el controlador cambia de rol. Sin embargo, si es menor está obsoleto, y en vez de responder con un OFPT_ROLE_REPLY el switch debe responder con un mensaje de error que indique que el cambio de rol no se puede llevar a cabo [5]. Gracias a *generation_id* se evita que dos controladores sean maestros simultáneamente.

En la Figura 24 se puede observar que se hacen dos peticiones de cambio de rol del controlador. En la primera, el controlador manda su OFPT_ROLE_REQUEST con el valor correspondiente a OFPCR_ROLE_NOCHANGE, con el objetivo de saber qué rol tiene actualmente. El switch responde con su OFPT_ROLE_REPLY con el valor de rol correspondiente a OFPCR_ROLE_EQUAL, respondiéndolele al controlador que

actualmente tiene el rol por defecto. El contenido de los mensajes se puede ver en las Figuras 29 y 30.

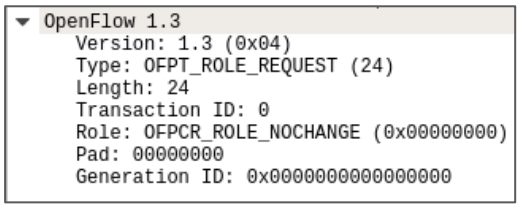

Figura 29: Primer OFPT_ROLE_REQUEST

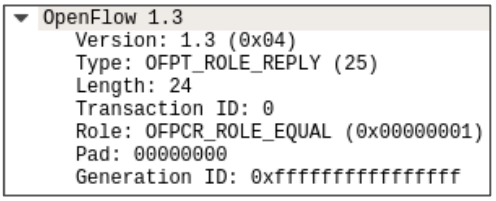

Figura 30: Primer OFPT_ROLE_REPLY

Como se puede observar en la Figura 30, el contenido de *generation_id* del OFPT_ROLE_REPLY no es el mismo que fue mandado por el controlador en su OFPT_ROLE_REQUEST. Esto se debe a que es el primer contacto entre el switch y el controlador [5].

En la segunda petición, el controlador pide cambiar a esclavo. Esto se debe a que, cuando un controlador se conecta a un switch, no sabe si ese switch ya está conectado a un maestro, por lo que inicialmente comienza como esclavo para pasar a ser maestro más adelante. El switch le responde confirmando al controlador que ahora es un controlador esclavo, y con la misma *generation_id* enviada por controlador, que es 0 al ser el primer cambio de rol a esclavo o maestro que efectúa ese controlador. Los campos de la segunda petición y respuesta se pueden ver en las Figuras 31 y 32.

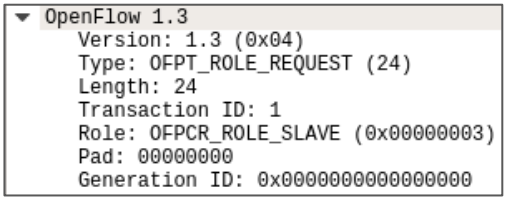

Figura 31: Segundo OFPT_ROLE_REQUEST

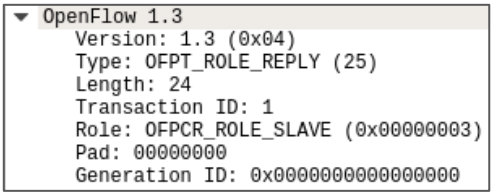

Figura 32: Segundo OFPT_ROLE_REPLY

Hay una tercera y cuarta peticiones que no se muestran en la Figura 24, al no ser parte del inicio de la comunicación. En la tercera, el controlador vuelve a pedir una confirmación de su rol al switch, con el mismo *generation_id* de la segunda petición. El switch responde confirmándole que es esclavo, y le vuelve a mandar la misma *generation_id*, como se muestra en las Figuras 33 y 34.

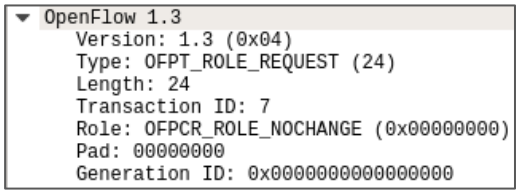

Figura 33: Tercer OFPT_ROLE_REQUEST

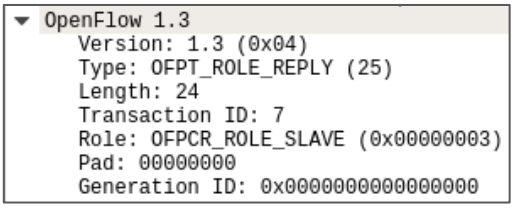

Figura 34: Tercer OFPT_ROLE_REPLY

Esto significa que, pasado un tiempo, ningún otro controlador ha pedido ser maestro, por lo que en la cuarta petición el controlador pide ser el maestro, y para ellos envía una *generation_id* de valor 1. El switch la acepta y manda su OFPT_ROLE_REPLY confirmando al controlador que ahora es el maestro, con la misma *generation_id* que envió el controlador en su OFPT_ROLE_REQUEST, como se muestra en las Figuras 35 y 36. Una vez que el controlador es maestro, ya podría operar sobre las tablas OpenFlow en el interior del switch.

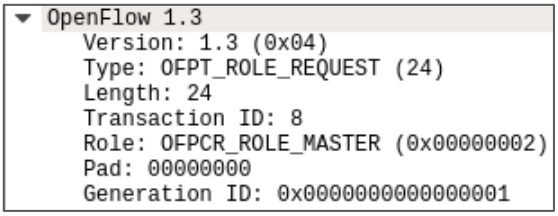

Figura 35: Cuarto OFPT_ROLE_REQUEST

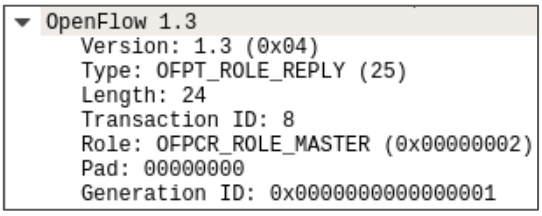

Figura 36: Cuarto OFPT_ROLE_REPLY

4.2.5 OFPT_MULTIPART_REQUES y OFPT_MULTIPART_REPLY

Los últimos mensajes que se ven en la captura son peticiones y respuestas OFPT_MULTIPART. Hay varios tipos de peticiones *multipart*, pero solo es necesaria una respuesta a OFPT_MULTIPART_REQUEST OFPMP_DESC para una correcta comunicación OpenFlow, por lo que el resto de peticiones se van a ignorar.

En general, los mensajes OFPT_MULTIPART_REQUEST sirven para que el controlador pida al switch el estado de sus diferentes componentes, tales como estadísticas sobre las tablas OpenFlow que hay en funcionamiento, estadísticas de los diferentes puertos del switch, o estadísticas de un *flow* en particular. La petición que nos interesa, OFPMP_DESC, es la más sencilla, ya que simplemente es una pequeña descripción del switch al que se ha conectado el controlador. Los campos de la petición OFPT_MULTIPART_REQUEST OFPMP_DESC se puede ver en la Figura 37.

A parte de los campos comunes en todo mensaje OpenFlow, este mensaje tiene uno para determinar el tipo de mensaje OFPT_MULTIPART_REQUEST que es. También tiene un campo llamado OFMPF_REQ_MORE que, dado que está completamente a 0, significa que no quedan más peticiones de este tipo por hacer. A continuación, es el turno de la respuesta, OFPT_MULTIPART_REPLY OFPMP_DESC y sus campos se pueden ver en la Figura 38.

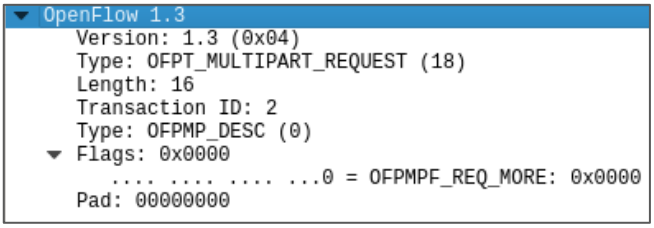

Figura 37: Mensaje OFPT_MULTIPART_REQUEST OFPMP_DESC

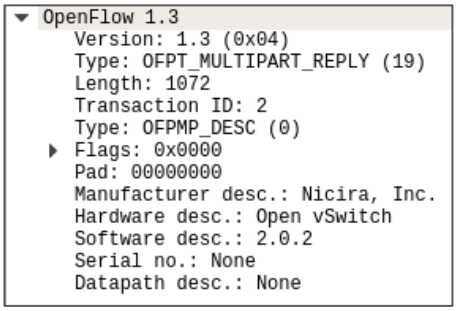

Figura 38: Mensaje OFPT_MULTIPART_REPLY OFPMP_DESC

La respuesta tiene los mismos campos que la petición, añadiendo además datos sobre el switch. Esos datos son la marca del switch, una descripción de qué hardware es, una descripción de que software está utilizando el switch, un número de serie y una descripción del camino de datos. Los dos últimos campos en este caso los envía vacíos.

4.2.6 OFPT_FLOW_MOD

Los últimos mensajes que se van a analizar son OFPT_FLOW_MOD y OFPT_METER_MOD, dos mensajes OpenFlow que el controlador envía al switch para modificar las tablas OpenFlow y las tablas de mediciones respectivamente. Estos mensajes son muy importantes, ya que dentro son enviados los parámetros para configurar la red de acceso GPON.

Se comenzará con OFPT_FLOW_MOD. Las causas por las que el controlador puede mandar un mensaje de este tipo al switch son variadas. Una de ellas es que el funcionamiento de una aplicación en el controlador provoque el envío, debido a datos que le han llegado desde el switch. En este caso entrarían los OFPT_FLOW_MOD que son enviados debido al funcionamiento de la aplicación *l2switch* en el controlador. Otra razón es que, en el controlador, estén las tablas y sus *flows* configurados de antemano para un switch con una *datapath_id* conocida, en cuyo caso el OFPT_FLOW_MOD es enviado tras establecer la comunicación entre switch y controlador. Además, si hay

establecida una conexión entre switch y controlador y se introduce o edita un *flow* de una tabla en el controlador, el nuevo *flow* es enviado al switch inmediatamente.

Para analizar los campos de un OFPT_FLOW_MOD se utilizará uno de los OFPT_FLOW_MOD enviados por el controlador debido al funcionamiento de *l2switch* sobre la red virtual creada en Mininet. El mensaje se puede ver en la Figura 39.

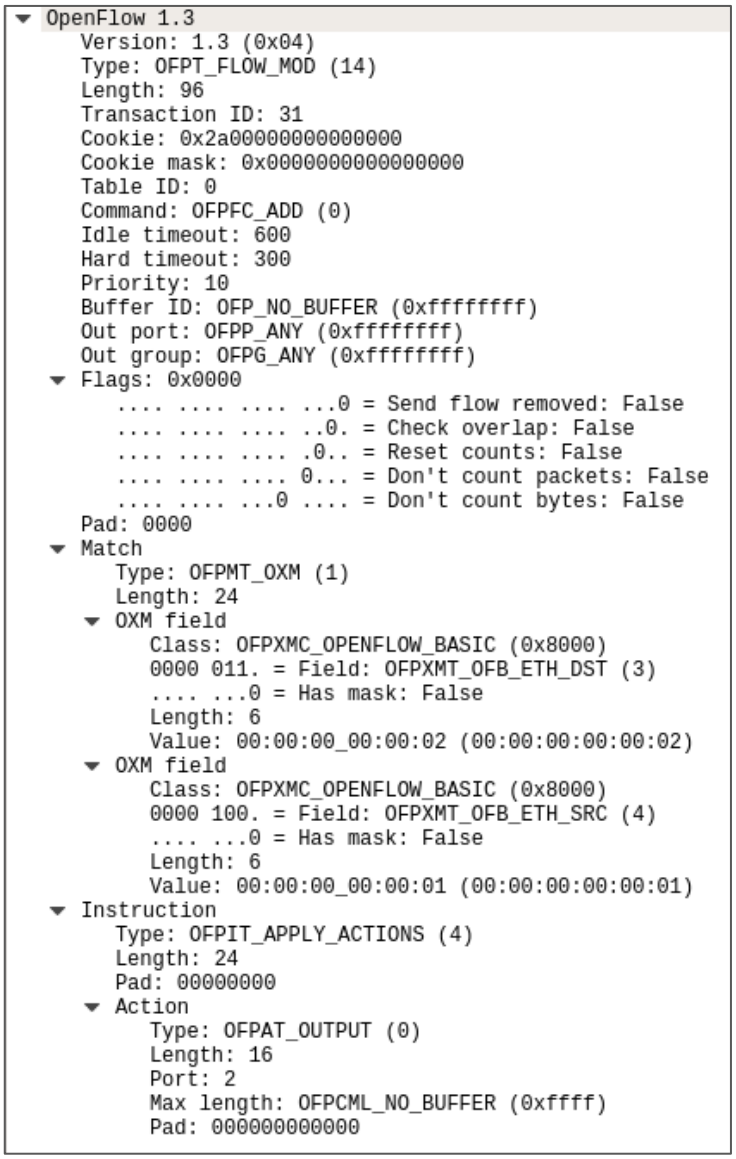

Figura 39: Mensaje OFPT_FLOW_MOD

El primer campo que aparece tras los campos comunes en todos los mensajes OpenFlow es *cookie.* El valor de este campo es elegido por el controlador para filtrar estadíticas del *flow*, modificaciones del *flow* o eliminaciones del *flow* [5]. En cuanto a *cookie_mask,* si no es 0, es utilizado junto a *cookie* para restringir las coincidencias entre

flows al borrar o modificar entradas *flow* [5]. Se puede pensar en *cookie_mask* como en una especie de máscara de subred que sirve para agrupar varios uno o varios *flows* en "subredes" con el objetivo de aplicar modificaciones sobre esas "subredes" en vez de sobre la totalidad de los *flows*.

El siguiente campo es "*table_id*". Este sencillo campo simplemente indica a qué tabla pertenece el *flow* que se está modificando. Tras "*table_id*" se encuentra "*command*", que identifica el tipo de mensaje OFPT_FLOW_MOD. Los tipos de mensajes posibles son los siguientes [5]:

- OFPFC_ADD: se da cuando el campo vale 0. El mensaje se limita a añadir un nuevo *flow* a la tabla designada.
- OFPFC_MODIFY: se da cuando el campo vale 1. Este mensaje sirve para editar el campo *instructions* de un *flow* ya existente.
- OFPFC_MODIFY_STRICT: se da cuando el campo vale 2. Versión estricta de OFPFC_MODIFY en la que es preciso que todos los campos *match*, sus máscaras y la prioridad coincidan con los de la entrada para poder modificar.
- OFPFC_DELETE: se da cuando el campo vale 3. Este mensaje sirve para borrar un *flow existente.*
- OFPFC_DELETE_STRICT: se da cuando el campo vale 4. Versión estricta de OFPFC_DELETE en la que es preciso que todos los campos *match*, sus máscaras y la prioridad coincidan con los de la entrada para poder eliminar.

Los siguientes campos sirven para designar una duración al *flow*. El primero, "*idle_timeout*", indica un número de segundos que ese *flow* puede estar sin recibir tráfico. Si pasado ese tiempo no ha pasado tráfico por el *flow*, éste expira. El siguiente es "*hard_timeout*", que indica que, pasado ese tiempo, la entrada será borrada sin importar si está siendo o no utilizada. Si cualquiera de los campos vale 0 su efecto es ignorado, de forma que se pueden configurar flows donde se aplique solo uno de los dos *timeouts*, o incluso una entrada permanente si ambos valen 0 [5].

El campo *priority* indica la prioridad con la que se aplica ese flow cuando un paquete entrante hace *match* con varias entradas [5].

El campo "*buffer_id*" se refiere a un paquete almacenado en el switch y siendo después mandado al controlador mediante un mensaje PACKET_IN. Un mensaje PACKET_IN puede ser enviado al controlador por el switch al recibir un paquete que el switch no sabe cómo procesar, normalmente por la falta de un *flow* que haga *match* con ese paquete. Entonces, el controlador respondería con un OFPT_FLOW_MOD, dándole al switch el o los *flows* que necesita.

Por su parte, "*out_port*" y "*out_group*" sirven para filtrar el alcance de OFPT_FLOW_DELETE y OFPT_FLOW_DELETE_STRICT por puerto de salida o grupo.

El siguiente campo, *flags*, activa ciertas opciones en el *flow* en función de los bits que tiene a 1. Esas opciones son las siguientes [5]:

- OFPFF_SEND_FLOW_REM: si el bit menos significativo está a 1, el switch debe mandar al controlador una notificación cuando es eliminado, ya sea por un OFPT_FLOW_DELETE, un OFPT_FLOW_DELETE_STRICT o que simplemente hayan expirado alguno de sus *timeouts*.
- OFPFF_CHECK_OVERLAP: si el segundo bit menos significativo está a 1, el switch debe asegurarse de que no se provoque una entrada conflictiva con otra con la misma prioridad y al final causen problemas. Si hay alguna, el OFPT_FLOW_MOD falla y se envía una notificación.
- OFPFF_RESET_COUNTS: al modificar un flow, indica si los contadores de paquetes y bytes que han pasado por allí se tienen que resetear o no. Se activa con el tercer bit menos significativo.
- OFPFF_NO_PKT_COUNTS: se activa con el cuarto bit menos significativo e indica que ese *flow* no debe contar los paquetes que pasan por él.
- OFPFF_NO_BYT_COUNTS: se activa con el quinto bit menos significativo e indica que ese *flow* no debe contar los bytes que pasan por él.

A continuación, nos encontramos con el campo "*match*". Este es uno de los campos más importantes, pues define las características que debe tener un paquete para que se ejecuten sobre él las *instructions* que se verán más adelante. El campo "*match*" está formado por diferentes subcampos. El primero de ellos define la estructura que utilizarán los subcampos en el interior de *match* que deben coincidir con campos de los paquetes que lleguen al switch [5]. Este subcampo solo admite dos opciones, OFPMT_STANDARD si vale 0, y OFPMT_OXM si vale 1. Dado que OFPMT_STANDARD está obsoleto, actualmente se utiliza OFPMT_OXM [5].

El siguiente subcampo, *length*, simplemente indica la longitud total del campo *match* (sin tener en cuenta rellenos con ceros) [5].

Los siguientes son los subcampos OXM (*OpenFlow Extensible Match*). Cada uno de estos subcampos define una regla con la que deberá coincidir un paquete si va a ser procesado por este *flow*. Por ejemplo, en un subcampo OXM se puede definir una dirección MAC de origen o de destino determinada, como se verá a continuación. Como se puede ver en la Figura 39, los subcampos OXM tienen a su vez cuatro parámetros diferentes. Con *oxm_class* se identifica el tipo de campos con los que va a haber coincidencias que se están utilizando. Este parámetro puede tener a su vez cuatro valores diferentes, siendo OFPXMC_OPENFLOW_BASIC (0x8000) el valor típico, ya que contiene el set básico de campos con los que coincidirán los campos de los paquetes de entrada (VLAN, IP origen, etc…). De los tres valores que quedan, dos son para tener compatibilidad con un formato utilizado anteriormente, el NXM (*Nicira Extensible Match*) y el último para desarrollar *matches* experimentales [5].

Después de *oxm_class*, se encuentra *oxm_field.* Dependiendo del valor de este parámetro, se sabrá ante que tipo de campo con el que coincidirá otro campo del paquete a procesar por el flow estamos. En la Figura 40 se describen sus diferentes valores y significados.

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

Figura 40: Valores y siginificados de *oxm_field*

El siguiente campo es *oxm_hasmask*. Si este valor está a 1, dentro del subcampo OXM se añare un parámetro más, *oxm_mask*. Esta máscara es de la misma longitud que el valor *oxm_value* y se aplicaría sobre él, de tal forma que se puedan lograr coincidencias parciales [5]. Por ejemplo, esta máscara podría conseguir que se procesen dos direcciones MAC diferentes por el mismo *flow* "escondiendo" sus diferencias.

Por último, el mencionado *oxm_value* es simplemente el valor que debe de tener el campo del paquete entrante para que lo procese el *flow*.

Para que quede más claro cómo funciona el subcampo OXM, en el primero de los que aparecen en la Figura 39 se aprecia que *oxm_field* vale “0000 011” (3), su campo *has_mask* vale “0” y su campo *oxm_value* vale “00 00 00 00 00 02”. Esto significa que, si en el switch entra un paquete con la dirección MAC de destino 00:00:00:00:00:02, será procesado por las instructions de ese *flow* (siempre y cuando también haya coincidencia con el resto de subcampos OXM del campo *match*).

Tras analizar el campo *match* y sus subcampos, solo queda el campo *instruction.* Hay diferentes tipos de instrucciones que se pueden aplicar, y cada *flow* puede aplicar diferentes instrucciones simultáneamente teniendo varios campos de *instruction*. El tipo de instrucción que ejecutará cada *instruction* se indica en el primer subcampo. La lista de diferentes tipos de instrucción y sus valores se muestra en la Figura 41.

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

    OFPIT_EXPERIMENTER = 0xFFFF  /* Experimenter instruction */
};
```

Figura 41: Tipos de *instruction* y sus valores

Para este trabajo, solo se van a utilizar OFPIT_APPLY_ACTIONS y OFPIT_METER. El primero simplemente aplica las acciones definidas en *instruction* sobre el paquete inmediatamente; el segundo sirve para asignar una tabla de mediciones al *flow* que se encargue de limitar el número de bytes o paquetes que pasen por él.

El siguiente subcampo ya define la acción que tomará *instruction*, y su estructura depende del tipo de *instruction* definido con anterioridad. Si se trata de un OFPIT_APPLY_ACTIONS, el primer parámetro define el tipo de acción que se va a realizar. Los diferentes tipos de acciones y sus valores se definen en la Figura 42.

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

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

Figura 42: Tipos de acciones aplicables en *instruction* y sus valores

Después, el siguiente parámetro es la longitud en bytes de la estructura que define la acción, y a partir de aquí los parámetros dependen del tipo de acción especificada. En el *instruction* de la Figura 39, como es una OFPAT_OUTPUT, los parámetros son el puerto por el que saldrá el paquete y la máxima cantidad de datos de ese paquete que se enviará al controlador si el campo del puerto fuera OFPP_CONTROLLER, que significaría que se enviaría al controlador [5].

Si el tipo de acción fuese OFPIT_METER, la estructura del subcampo descrito anteriormente variaría a partir del parámetro que define su longitud. Tras la longitud, simplemente habría un parámetro *meter_id* que identifica la tabla de mediciones que va a utilizar ese *flow* [5].

4.2.7 OFPT_METER_MOD

Con OFPT_METER_MOD, el controlador envía al switch las tablas de mediciones que va a necesitar. En la Figura se puede ver un ejemplo de mensaje OFTP_METER_MOD capturado en Wireshark.

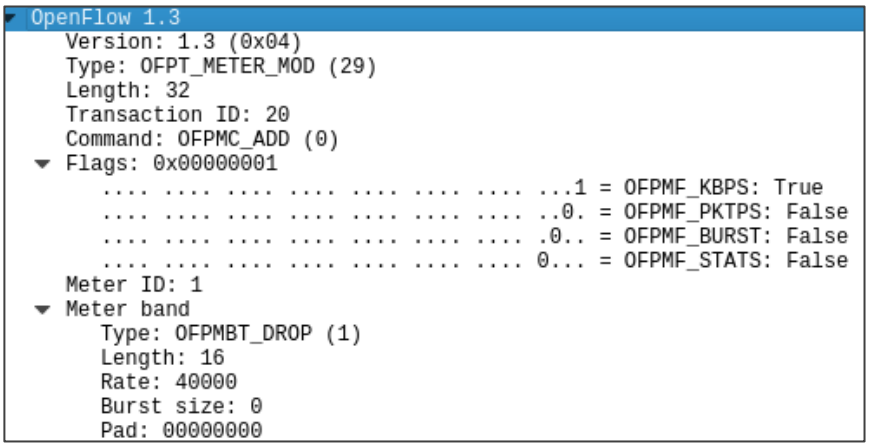

Figura 43: Mensaje OFPT_METER_MOD

Tras los campos comunes en todos los mensajes OpenFlow descritos con anterioridad, el primer campo de este mensaje es *command*, que indica si se va a añadir un flow (OFPMC_ADD), se va a modificar (OFPMC_MODIFY) o se va a eliminar (OFPMC_DELETE) [5].

El siguiente campo es *flags*. Dependiendo de los bits que estén a 1 en este campo, las mediciones serán en Kbps si es el menos significativo o en paquetes por segundo si es el segundo menos significativo [5].

Tras *flags*, se encuentra *meter_id*. Este identificador identifica al meter en el switch, y sirve para asignar el *meter* a uno de los *flows*.

Los últimos campos importantes son el tipo de tabla de mediciones y la tasa asignada. Para este trabajo, el tipo más importante es OFPMBT_DROP. Este tipo de tabla de mediciones descarta paquetes cuando alcanzan una tasa máxima definida en el siguiente campo. Las unidades de esta tasa son las asignadas en el campo *flags*.

4.3 Programación

Una vez se han analizado los diferentes mensajes OpenFlow que se van a encontrar, se va a pasar a la programación del propio agente. Para crear los mensajes OpenFlow, se han utilizado unas librerías denominadas python-openflow creadas por Kytos SDN [8], que permiten la creación de mensajes OpenFlow 1.3 en Python de forma cómoda. Además, como ya existía un programa en Python que permite configurar servicios en el OLT a través de su CLI, se decidió adaptar dicho programa, de forma que

los parámetros se sacasen de los mensajes OpenFlow enviados por el controlador en vez de ser indtroducidos por teclado.

Dichos parámetros se guardarán en un vector de tamaño cinco, llamado "*parametros1*[]", definido de forma global en el programa. Se crean e inicializan las variables *flagRole*, *storeRole*, *extraByte*, *flagBW* y *storeID*. Las cuatro primeras se inicializan a 0, mientras que la última se inicializa al entero correspondiente a 0xffffffffffffffff en hexadecimal. Las variables *storeRole* y *storeID* guardarán el rol del controlador y su *generation_id* respectivamente, con el objetivo de dar respuestas a los cambios de rol del controlador de forma satisfactoria. La variable *flagRole* simplemente sirve para saber si una OFPT_ROLE_REQUEST que pide el rol actual del controlador es o no la primera que ha sido enviada al programa y enviar en función de ésto el OFPT_ROLE_REPLY correspondiente. La variable *flagBW* sirve para saber si ha llegado el OFTP_METER_MOD correspondiente al *downstream*, con el objetivo de utilizar el siguiente mensaje de este tipo para configurar el *upstream*. Por último, la variable extraByte sirve para operar con cada uno de los mensajes OpenFlow que llegan en cada paquete, ya que a veces el controlador envía varios mensajes OpenFlow utilizando el mismo paquete. En la Figura 44 se muestra una captura del código desarrollado en Python correspondiente a esta parte explicada. Cabe destacar que el resto del código correspondiente al agente OpenFlow-Pyghon se encuentra recogido al final de la memoria en el Anexo I.

```
1360  def Main():
1361  #DATOS INICIALES
1362      flagBW =0
1363      flagRole=0
1364      storeRole=0
1365      extraByte=0
1366      storeID=int("0xffffffffffffffff",16)
1367  #ESTABLECIMIENTO DE COMUNICACION----------------------
1368      host = '192.168.56.101'
1369      port = 6633
1370      mySocket = socket.socket()
1371      mySocket.connect((host,port))
1372
1373      mapa = bytes.fromhex('00000010')
1374      element = HelloElemVersionbitmap(bitmaps=mapa)
1375      element.length = 8
1376      sendHello = Hello(xid = 10, elements = [element])
1377      message = sendHello.pack()
1378      mySocket.send(message)
```

Figura 44: Inicio del código del programa del agente OpenFlow-Python

4.3.1 Establecimiento de la comunicación

El establecimiento de la comunicación consiste en crear un socket TCP (*Transmission Control Protocol*) que permita la comunicación entre el programa y el controlador, mandar el primer OFPT_HELLO y recibir la respuesta OFPT_HELLO del controlador.

Para crear el socket, se han creado unas variables *host* y *port*, que almacenan la IP del controlador (192.168.56.101) y su puerto para utilizar OpenFlow (663), respectivamente. Con el código "*mySocket = socket.socket()*" se crea un socket, y con "*mySocket.connect((host,port))*" se hace la conexión con el controlador.

Después, gracias a la librería python-openflow de Kytos [8], se crea un mapa de bits para indicar en el mensaje OFPT_HELLO que se va a enviar la versión de OpenFlow que utiliza el programa (versión 1.3), se empaqueta el mensaje y se envía con el código "*mySocket.send(message)*". Después, mediante la acción "*data = mySocket.recv(1024)*" se guarda en la variable data los bytes recibidos desde el controlador.

4.3.2 Comunicación OpenFlow

El resto de la comunicación consiste en un bucle infinito que va respondiendo las peticiones del controlador hasta que se obtienen los parámetros deseados, en cuyo caso se sale del bucle.

El bucle infinito comienza poniendo a 0 la variable "*extraByte*". Después, con la orden "*data = mySocket.recv(1024)",* se espera recibir algo del controlador. Cuando se recibe algo, se guarda la longitud de lo recibido en la variable "*longReal*" y se entra en un nuevo bucle. En este bucle, primero se miran los bytes correspondientes al tipo de mensaje OpenFlow y se actúa en consecuencia. Si se reciben mensajes del tipo OFPT_FEATURES_REQUEST, OFPT_ROLE_REQUEST y OFPT_BARRIER_REQUEST, se responde con un mensaje tipo OFTP_FEATURES_REPLY, un OFPT_ROLE_REPLY o un OFTP_BARRIER_REPLY respectivamente. Todas las respuestas utilizan la misma *Transaction ID* que las peticiones que las preceden. El resto de campos de cada mensaje se rellena de la misma forma que lo haría un switch y que dicta el estándar OpenFlow.

Cuando llega un mensaje del tipo OFPT_MULTIPART_REQUEST, solo se contesta cuando se trata de un OFPMP_DESC, y se rellena con la marca "*GPON, UVA*", descripción del hardware "*Agente OpenFlow para configurar OLT*" y descripción del software "*1.0*". Cuando llegan otros tipos de mensajes OFPT_MULTIPART_REQUEST, se ignoran, ya que no aportan nada a la comunicación.

Por último, cuando llega un OFPT_FLOW_MOD, se mira el campo correspondiente a su longitud. Si mide 120 bytes, significa que es apto para configurar un servicio en el OLT ya que los campos mínimos para una configuración de un servicio hacen que el mensaje mida 120 bytes, y se sacan de los bytes recibidos los datos de configuración, salvo el ancho de banda. Aquí sería cuando el campo "*parametros1[3]"* del vector se pondría a 0 o 1 dependiendo del tipo de servicio a configurar, mientras que "*parametros1[0]*" y "*parametros1[1]*" se rellenan con la dirección MAC de la ONU donde se configurará el servicio y la VLAN, respectivamente. Después, cuando el controlador envía el OFPT_METER_MOD asociado al OFPT_FLOW_MOD, el programa obtiene el ancho de banda *downstream* o *upstream*, dependiendo de si ha configurado ya el *downstream* o no, y lo introduce en el campo "*parametros1[2]*" del vector si se trata del *downstream*, y *parametros1[4]* si es el *upstream*.

En la Figura 45 se puede apreciar como se sacan los parámetros de configuración desde los mensajes OpenFlow OFPT_FLOW_MOD u OFTP_METER_MOD.

```
elif (data[1+extraByte] == 14):
    if (int.from_bytes(data[(2+extraByte):(4+extraByte)],byteorder='big') == 120):
        parametros1[1] = int.from_bytes(data[(74+extraByte):(76+extraByte)],byteorder='big')-2**12
        parametros1[0] = data[(64+extraByte):(70+extraByte)].hex()
        parametros1[3] = data[83+extraByte]
        if(parametros1[3] == 255):
            parametros1[3] = 1
        else:
            parametros1[3] = 0
elif (data[1+extraByte] == 29):
    if (flagBW == 0):
        parametros1[2] = int.from_bytes(data[(20+extraByte):(24+extraByte)],byteorder='big')
        flagBW += 1
    elif (flagBW == 1):
        parametros1[4] = int.from_bytes(data[(20+extraByte):(24+extraByte)],byteorder='big')
```

Figura 45: Código responsable de identificar si un mensaje es OFPT_FLOW_MOD (data[1+*extraByte*] = 14) o si es OFPT_METER_MOD (data[1+e*xtraByte*] == 29), y sacar los parámetros de configuración en consecuencia

Tras dar la respuesta a un mensaje, se mira si la longitud de los datos que han llegado, guardada en *longReal*, es igual a la longitud que dice tener el mensaje OpenFlow

que se acaba de procesar. Si es igual, significa que el paquete solo llevaba ese mensaje OpenFlow, se sale del bucle y se espera a recibir otro mensaje OpenFlow. Sin embargo, si son diferentes, significa que el paquete llevaba varios mensajes OpenFlow. Para que el bucle procese el siguiente mensaje del paquete, *longReal* actualiza su valor disminuyendo en la misma cantidad que el número de bytes del primer mensaje OpenFlow, y *extraByte* aumenta su valor en esa misma cantidad. De esta forma, se podrán sacar los datos del siguiente mensaje en los mismos campos mirados si se tratara de un único mensaje, pero sumando la cantidad *extraByte*, y cuando se llega al último mensaje del paquete *longReal* va a coincidir con el tamaño de este y se saldrá del bucle. Por ejemplo, si en un mismo paquete llegasen un mensaje 1 de 8 bytes y un mensaje 2 de 16 bytes, *longReal* inicialmente serían 24 bytes. El tipo de mensaje OpenFlow de mensaje 1 estaría en *data*[1]. Tras procesarlo, *longReal* y la longitud del mensaje 1 son diferentes, por lo que *longReal* se pone a 24-8=16, y *extraByte* se pone a 0+8=8. En la iteración correspondiente al mensaje 2, se mirará *data[1+extraByte]*, es decir, data[9], para saber qué tipo de mensaje OpenFlow es. Tras procesarlo, como el valor actual de *longReal* es igual a la longitud real del mensaje 2, ya se saldría del bucle.

Una vez se han obtenido los cinco parámetros, se sale del bucle y se introducen estos valores dentro de las variables ya existentes en el programa previo en Python que configura el OLT a través de comandos CLI. De este modo, se terminaría de configurar un servicio en la red de acceso GPON con éxito, ya que el agente es capaz de pasar información encapsulada en mensajes OpenFlow al estándar CLI que entiende la red GPON. En la Figura 46 se muestra un ejemplo entre la interacción entre el programa desarrollado en este proyecto y el el que configura los servicios en el OLT.

```
204     while i < num_servicios_Internet:
205         num_servicio = str(i+1)
206         true_VLAN = 0
207         cont=0
208         # Las etiquetas VLAN van de 1 a 4094. Si el usuario introduce un valor fuera de ese
209         # rango, se le volverá a pedir que introduzca el valor. Hay que recordar que la red
210         # solo ofrece servicio en las VLAN 833 y 806
211         while true_VLAN == 0:
212             if cont == 0:
213                 print("Identficador VLAN [1-4094] para el servicio " + num_servicio + " (833 - Servido
214             else:
215                 print("El identificador VLAN no es válido. Vuelva a probar:")
216             cont = cont+1
217             entrada = parametros1[1]
218             entrada = int(entrada)
219             if entrada > 0 and entrada < 4095:
220                 VLAN_ID.append(entrada)
221                 print("El identificador VLAN para el servicio " + num_servicio + " es:", VLAN_ID[i])
222                 i=i+1
223                 true_VLAN = 1
224
```

Figura 46: Identificador VLAN pasado por el nuevo programa al programa que configura la red de acceso GPON

Actualmente, el programa contempla que cada servicio configurado tiene un *flow* diferente, donde se incluye la edición del *flow* y *meter* descrita en el Capítulo 3. Sin embargo, en el futuro todos los *flows* de configuración de servicios podrían ir en la misma tabla OpenFlow, pero también se podría crear una tabla diferente por cada ONU y en el interior de cada una de ellas introducir solo los *flows* con servicios referentes a esa ONU, de forma que quede todo más organizado.

4.3.3 Conclusiones

En este capítulo se ha analizado la comunicación real entre nuestro controlador OpenDayLight y un swich virtual en Mininet, estudiando paso a paso el protoloco OpenFlow y los mensajes intercambiados. El siguiente paso fue emular esta misma comunicación entre dicho controlador y un agente OpenFlow-Python cuyo objetivo final será configurar una red GPON. En este sentido, el agente programado es capaz de comunicarse vía OpenFlow con el controlador OpenDayLigth y a partir de la información enviada en mensajes OpenFlow, es capaz de desecapsularla y volcarla sobre el OLT y así configurar servicios y perfiles de abonado y asociarlos a los usuarios conectados a la red GPON (ONTs/ONUs).

Además, a partir de lo descrito aquí, se podrían crear diversos programas capaces de hacerse pasar por un switch OpenFlow para otro tipo de tareas de configuración. No obstante, también se ha llegado a la conclusión de lo duro que puede llegar a ser aprender un protocolo con el objetivo de programarlo, algo totalmente novedoso para mi.

5

Conclusiones y Líneas Futuras

5.1 Conclusiones

En este Trabajo de Fin de Grado se ha llevado a cabo la programación de un agente OpenFlow capaz de hacer posible la configuración de servicios de red en una red de acceso GPON a través de un controlador OpenFlow, en concreto OpenDayLight. La principal motivación ha sido la integración de OpenFlow para la gestión de servicios y perfiles de abonado en una red PON, hasta ahora no existente, pues los OLT no soportan este tipo de funcionalidades.

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. Después, se ha buscado información sobre diferentes controladores OpenFlow, siendo OpenDayLight el finalmente escogido. Con el controlador elegido, se probó y analizó el funcionamiento básico de OpenFlow en una red virtual creada en Mininet, así como el diseño de tablas OpenFlow de forma manual a través de la interfaz web del controlador. Por último, se analizó el intercambio de mensajes OpenFlow entre el controlador OpenDayLihgt y un swich virtual creado en Mininet. Finalmente, se llevo a cabo el diseño y programación del agente OpenFlow-Python emulando en primer lugar la comunicación entre el controlador OpenDayLight y nuestro programen en Python, gracias a la librería Kytos. En segundo lugar, se programó el envío de las tablas OpenFlow necesarias para la configuración de la red GPON de modo que el agente es capaz de desencapular dicha información OpenFlow y pasarla al lenguaje CLI que entiende el OLT.

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.

5.2 Líneas Futuras

A partir de este proyecto, pueden llevarse a cabo varios desarrollos e investigaciones. En primer lugar, se podría desarrollar algún tipo de aplicación en el interior del controlador que automatice la creación de tablas en tiempo real en función de parámetros de la red, de forma que sea capaz de responder a picos de tráfico inesperados que congestionen la red, por ejemplo.

Otra línea sería el uso de controladores diferentes a OpenDayLight, como ONOS o Ryu, con el objetivo de comparar sus rendimientos para este tipo de trabajo. Además, cabe destacar que Ryu es más sencillo que OpenDayLight pues requiere una curva de aprendizaje menor y además está preparado para desarrollar extensiones del propio protocolo OpenFlow.

Por último, también sería interesante diseñar un agente similar al de este trabajo, pero con la finalidad de controlar y configurar parámetros de las ONTs/ONUs. De este modo, el controlador OpenDayLight (o cualquier otro elegido) será capaz de gestionar de manera inteligente y automatizada ambos agentes activos de la red de acceso.

6 Referencias

1] «Página web oficial de TELNET Redes Inteligentes,» [En línea]. Available: http://www.telnet-ri.es/.

2] «Antigua página web oficial de OpenFlow,» [En línea]. Available: http://archive.openflow.org/wp/learnmore/.

3] «G.984.1: Gigabit-capable passive optical networks (GPON): General characteristics,» International Telecommunication Union, [En línea]. Available: http://www.itu.int/rec/T-REC-G.984.1-200803-I.

4] «Página web oficial del lenguaje de programación Python,» [En línea]. Available: https://www.python.org/.

5] «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.

6] «Página web del controlador OpenDayLight,» [En línea]. Available: https://www.opendaylight.org/.

7] «Página web del controlador ONOS,» [En línea]. Available: http://onosproject.org/.

8] K. S. Platform, «Librerías Python-OpenFlow,» [En línea]. Available: https://github.com/kytos/python-openflow.

9] «Página oficial del controlador NOX,» [En línea]. Available: https://github.com/noxrepo/nox.

«Página web del controlador POX,» [En línea]. Available:
10] https://github.com/noxrepo/pox.

«Página web del controlador Beacon,» [En línea]. Available:
11] https://openflow.stanford.edu/display/Beacon/Home.

«Página web del controlador Floodlight,» [En línea]. Available:
12] http://www.projectfloodlight.org/floodlight/.

«Página web del lenguaje de programación Java,» Oracle, [En línea].
13] Available: https://www.oracle.com/es/java/index.html.

«Información sobre diferentes controladores OpenFlow,» [En línea].
14] Available: https://www.packtpub.com/books/content/openflow-controllers.

«Página oficial del controlador Ryu,» [En línea]. Available:
15] https://osrg.github.io/ryu/index.html.

R. Chua, «2016 SDN Controller Landscape,» Open Networking Foundation,
16] [En línea]. Available: https://events.linuxfoundation.org/sites/events/files/slides/3 2016 ONS ONF Mkt Opp Controller Landscape RChua Mar 14 2016.pdf.

«Página web de Virtualbox,» [En línea]. Available:
17] https://www.virtualbox.org/.

«Página web del sistema operativo Linux Mint,» [En línea]. Available:
18] https://www.linuxmint.com/.

«Enlace para descargar OpenDayLight Boron-SR3,» [En línea]. Available:
19] https://www.opendaylight.org/software/downloads/boron-sr3.

M. B. K. W. A. Bierman, «RESTCONF Protocol,» Internet Engineering
20] Task Force, January 2017. [En línea]. Available: https://tools.ietf.org/html/rfc8040.

E. M. Bjorklund, «YANG - A Data Modeling Language for,» Internet Engineering Task Force, October 2010. [En línea]. Available:

21] https://tools.ietf.org/html/rfc6020.

«Página web del desarrollador XML,» [En línea]. Available:
22] https://www.w3.org/XML/.

Anexo I: Código del programa

```
import socket
from pyof.v0x04.symmetric.hello import *
from pyof.v0x04.asynchronous.packet_in import *
from pyof.v0x04.asynchronous.error_msg import ErrorMsg
from pyof.v0x04.asynchronous.error_msg import *

from pyof.v0x04.controller2switch import features_reply

from pyof.foundation.base import GenericMessage
from pyof.foundation.basic_types import BinaryData
from pyof.v0x04.common.header import Header, Type
from pyof.v0x04.common.flow_match import *
from pyof.v0x04.controller2switch.features_reply import *
from pyof.v0x04.controller2switch.barrier_reply import *
from pyof.v0x04.controller2switch.role_reply import *
from pyof.v0x04.controller2switch.common import *
from pyof.v0x04.controller2switch.multipart_reply import *
from pyof.v0x04.controller2switch.multipart_request import *
import requests


import http.client
import telnetlib
import time
# Módulo math: permite realizar operaciones matemáticas con los anchos de banda
import math
# Módulos os y path: permite comprobar la existencia de ficheros en el directorio de trabajo
```

```
import os.path as path
import os
# Módulo necesario para la creación y modificación del fichero XML. Este fichero permitirá
# recuperar las configuraciones de las ONUs al apagar la red o al cambiar de modo de
# gestión (hay que recordar que el CLI NO TIENE PERSISTENCIA en los datos configurados).
import xml.etree.cElementTree as etree
import binascii


__all__ = ('EchoRequest',)


# Classes
parametros1 = [None] * 5


def get_ID_ONU():

        # Host y puerto al que se hace la conexión Telnet para acceder al CLI
        host = "172.26.128.38"
        port = "4551"

        # Claves de acceso al CLI
        password1 = "TLNT25"
        password2 = "TLNT145"
        enable = "enable"

        # Acceso al CLI: conexión Telnet al host y puerto indicados anteriormente
        tn = telnetlib.Telnet(host,port,1)
        # Mediante la función write de telnetlib, escritura de los comandos que permiten
        # acceder al menú de privilegios del CLI
        tn.write(password1.encode('ascii') + b"\n")
```

```
time.sleep(0.1)
tn.write(enable.encode('ascii') + b"\n")
time.sleep(0.1)
tn.write(password2.encode('ascii') + b"\n")
time.sleep(0.1)

# Declaración y escritura del comando que permite ver las ONUs conectadas a
# la red así como sus direcciones MAC
comandos = "configure \n olt-device 0 \n olt-channel 0 \n show serial-number allocated
\n"
tn.write(comandos.encode('ascii') + b"\n")
time.sleep(0.1)

# Lectura de los datos (tanto enviados como recibidos) del CLI y volcado en
# un fichero de texto para su posterior análisis
data = tn.read_very_eager().decode()
# Se abre el fichero con modo de escritura, de esta forma cada vez que cambie
# el estado de la red el fichero se sobreescribirá con la información nueva
outfile = open('IDs_ONUs.txt', 'w')
# Se escriben los datos procedentes de la escritura de los comandos de arriba
outfile.write(data)
# Ciere del fichero
outfile.close()

# Creación del vector que almacenará las direcciones MAC de las ONUs conectadas
MAC = []
# Se abre el archivo anterior en modo lectura
outfile = open('IDs_ONUs.txt', 'r')
# Se almacenan todas las lineas del fichero con la función readlines()
lines = outfile.readlines()
```

```
        # Bucle que recorre cada línea del fichero
        for line in lines:
            # Se almacenan todas las palabras de cada línea
            palabras = line.split()
            # Bucle que recorre cada palabra de la línea
            for p in palabras:
                # Si los 18 primeros caracteres de una palabra coinciden con los indicados,
                # se trata de una direccion MAC -> Se ha detectado una ONU
                if p[:18]=='54-4c-52-49-5b-01-':
                    # Se añade la MAC al vector. La posición en la que se añade indica
                    # el identificador de la ONU.
                    MAC.append(p)

    #   FORMA DE BUSCAR ONUs ONU A ONU -> Menos eficiente y hay que añadir código
    #   si se conecta alguna ONU más a la red
    #   if p=='54-4c-52-49-5b-01-f6-90':
    #      MAC[id_ONU] = '54-4c-52-49-5b-01-f6-90'
    #      id_ONU = id_ONU + 1
    #   if p=='54-4c-52-49-5b-01-f7-30':
    #      MAC[id_ONU] = '54-4c-52-49-5b-01-f7-30'
    #      id_ONU = id_ONU + 1
    #   if p=='54-4c-52-49-5b-01-f6-d8':
    #      MAC[id_ONU] = '54-4c-52-49-5b-01-f6-d8'
    #      id_ONU = id_ONU + 1
    #   if p=='54-4c-52-49-5b-01-f7-28':
    #      MAC[id_ONU] = '54-4c-52-49-5b-01-f7-28'
    #      id_ONU = id_ONU + 1
```

```
# Cierre del archivo
outfile.close()

# Si existe un fichero con este nombre en el directorio de trabajo, sale de la función
# devolviendo el vector de direcciones MAC
if path.exists("configuracionGPON.xml"):
    return MAC

# Si no existe, se ha de crear el fichero XML
else:
    # Se define el elemento raíz
    root = etree.Element("RedGPON")

    # Se añaden las ONUs (el nº de ONUs es la longitud del vector de direcciones MAC)
    i=0
    while i<len(MAC):
        # Cada ONU se añade con su dirección MAC como un subelemento del elemento raíz
        ONU = etree.SubElement(root,"ONU", MAC=MAC[i])
        i=i+1

# FORMA DE CREAR EL ÁRBOL ONU A ONU -> Menos eficiente y hay que añadir código
# si se conecta alguna ONU más a la red
# ONU1 = etree.SubElement(root,"ONU", MAC='54-4c-52-49-5b-01-f6-90')
# ONU2 = etree.SubElement(root,"ONU", MAC='54-4c-52-49-5b-01-f7-30')
# ONU3 = etree.SubElement(root,"ONU", MAC='54-4c-52-49-5b-01-f6-d8')
# ONU4 = etree.SubElement(root,"ONU", MAC='54-4c-52-49-5b-01-f7-28')

    # Se crea el árbol XML y se escribe en el fichero con el nombre indicado
```

```
                tree = etree.ElementTree(root)
                tree.write("configuracionGPON.xml")
                # Se devuelve el vector de direcciones MAC
                return MAC

        # Función servicio_Internet(ID_ONU,MAC_ONU,num_servicios_Internet): permite configurar
        # servicios de Internet y toma como parámetros el identificador de la ONU
        # (posición en el vector de direcciones MAC), la MAC y el número de servicios a configurar.
        # Guarda la configuración actual en un fichero txt (esta configuración es temporal y se
        # mantiene mientras no se apague la red ni se cambie al modo de gestión TGMS).
        # La función también actualiza el fichero XML para recuperar configuraciones
        # cuando se apaga la red o se cambia al TGMS.
        def servicio_Internet(ID_ONU,MAC_ONU,num_servicios_Internet):

                # Creación de los vectores en los que se almacenarán los parámetros
                # internos de configuración
                port_ID = []
                alloc_ID = []
                tcont_ID = []
                num_instancia = []
                puntero = []
                ds_profile_index = []

                # Dependiendo del número de servicios, se asginan tantos valores como sean necesarios
                # a los vectores anteriormente creados
                i=0
                while i < num_servicios_Internet:
                        # Para que no se solapen puertos y allocs-ID, se asignan en función del
identficiador
                        # de la ONU y del número de servicio en cuestión
```

```
        port_ID.append(600+100*ID_ONU+i)
        alloc_ID.append(600+100*ID_ONU+i)
        # Estos valores se asignan de este modo también para evitar solapamientos
        num_instancia.append(i+2)
        tcont_ID.append(i)
        puntero.append(32768+i)
        ds_profile_index.append(i)
        i=i+1

# Flags que marcarán la salida de los diferentes bucles cuando el parámetro
# requerido haya sido introducido de forma válida
true_VLAN = 0
true_BW_Downstream_GR = 0
true_BW_Downstream_Excess = 0
true_BW_Upstream_GR = 0
true_BW_Upstream_BE = 0

# Creación de los vectores en los que se almacenarán los parámetros
# de configuracion que el USUARIO deberá introducir: identificador de VLAN,
# ancho de banda Downstream garantizado y en exceso y ancho de banda Upstream
# garantizado y Best Effort
VLAN_ID = []
BW_Downstream_GR = []
BW_Downstream_Excess = []
BW_Upstream_GR = []
BW_Upstream_BE = []

# En los sucesivos bucles, se irán pidiendo los parámetros de configuración al usuario.
```

```
		# Primer bucle: se piden el identificador VLAN que corresponda a cada servicio
		# La VLAN 833 conecta con un servidor DHCP que asigna la dirección mientras que
		# la VLAN 806 recibe una IP de forma estática (no la tiene que introducir el usuario)
		i=0
		while i < num_servicios_Internet:
			num_servicio = str(i+1)
			true_VLAN = 0
			cont=0
			# Las etiquetas VLAN van de 1 a 4094. Si el usuario introduce un valor fuera de
ese
			# rango, se le volverá a pedir que introduzca el valor. Hay que recordar que la
red
			# solo ofrece servicio en las VLAN 833 y 806
			while true_VLAN == 0:
				if cont == 0:
					print("Identficador VLAN [1-4094] para el servicio " +
num_servicio + " (833 - Servidor DHCP, 806 - Dirección IP estática): ")
				else:
					print("El identificador VLAN no es válido. Vuelva a probar:")
				cont = cont+1
				entrada = parametros1[1]
				entrada = int(entrada)
				if entrada > 0 and entrada < 4095:
					VLAN_ID.append(entrada)
					print("El identificador VLAN para el servicio " +
num_servicio + " es:", VLAN_ID[i])
					i=i+1
					true_VLAN = 1

		print("\n")

		# Segundo bucle: se pide el ancho de banda garantizado en sentido Downstream.
```

```
# Este ancho de banda se introduce en Kbps y debe estar entre 0 y 2488000 (aunque
# a efectos prácticos no tiene sentido meter más de 600000 Kbps, aproximadamente).
# El CLI solo admite múltiplos de 64 Kbps de manera que el programa trunca el
# valor del usuario para que coincida con un múltiplo de 64 Kbps.
i=0
while i < num_servicios_Internet:
        num_servicio = str(i+1)
        true_BW_Downstream_GR = 0
        cont=0
        while true_BW_Downstream_GR == 0:
                if cont == 0:
                        print("Ancho de banda Downstream garantizado en Kbps [0-2488000] para el servicio " + num_servicio + ": ")
                else:
                        print("Ancho de banda no válido. Vuelva a probar:")
                cont=cont+1
                entrada = parametros1[2]
                entrada = int(entrada)
                entrada = entrada / 64
                entrada = math.floor(entrada)
                entrada = 0.9*entrada * 64
                if entrada > -1 and entrada < 2488000:
                        BW_Downstream_GR.append(entrada)
                        print("El ancho de banda Downstream garantizado en Kbps es:", BW_Downstream_GR[i])
                        i=i+1
                        true_BW_Downstream_GR = 1

# Tercer bucle: se pide el ancho de banda en exceso en sentido Downstream.
# Este ancho de banda se introduce en Kbps y debe estar entre 0 y 2488000 (aunque
```

```
                # a efectos prácticos no tiene sentido que la suma del garantizado y exceso sea
                # mayor que 600000 Kbps, aproximadamente). El CLI solo admite múltiplos de 64 Kbps
                # de manera que el programa trunca el valor del usuario para que coincida con
                # un múltiplo de 64 Kbps.
                i=0
                while i < num_servicios_Internet:
                        num_servicio = str(i+1)
                        true_BW_Downstream_Excess = 0
                        cont=0
                        while true_BW_Downstream_Excess == 0:
                                if cont == 0:
                                        print("Ancho de banda Downstream en exceso en Kbps [0-
2488000] para el servicio " + num_servicio + ": ")
                                else:
                                        print("Ancho de banda no válido. Vuelva a probar:")
                                cont=cont+1
                                entrada = parametros1[2]
                                entrada = int(entrada)
                                entrada = entrada / 64
                                entrada = math.floor(entrada)
                                entrada = 0.1*entrada * 640
                                if entrada > -1 and entrada < 2488000:
                                        BW_Downstream_Excess.append(entrada)
                                        print("El ancho de banda Downstream en exceso en Kbps es:",
BW_Downstream_Excess[i])
                                        i=i+1
                                        true_BW_Downstream_Excess = 1


                print("\n")
```

```
# Cuarto bucle: se pide el ancho de banda garantizado en sentido Upstream.
# Este ancho de banda se introduce en Mbps y debe estar entre 0 y 1244 (aunque
# a efectos prácticos no tiene sentido meter más de 600 Mbps, aproximadamente).
i=0
while i < num_servicios_Internet:
        num_servicio = str(i+1)
        true_BW_Upstream_GR = 0
        cont=0
        while true_BW_Upstream_GR == 0:
                if cont == 0:
                        print("Ancho de banda Upstream garantizado en Mbps [0-
1244] para el servicio " + num_servicio + ": ")
                else:
                        print("Ancho de banda no válido. Vuelva a probar:")
                cont=cont+1
                entrada = parametros1[4]
                entrada = 0.9*int(entrada)/1000
                if entrada > -1 and entrada < 1245:
                        BW_Upstream_GR.append(entrada)
                        print("El ancho de banda Upstream garantizado en Mbps es:",
BW_Upstream_GR[i])
                        i=i+1
                        true_BW_Upstream_GR = 1

print("\n")

# Quinto bucle: se pide el ancho de banda garantizado Best Effort Upstream.
# Este ancho de banda se introduce en Mbps, debe estar entre 0 y 1244 (aunque
# a efectos prácticos no tiene sentido meter más de 600 Mbps, aproximadamente) y
# tiene que ser igualo mayor que el ancho de banda Upstream garantizado (el valor Best
```

```
                # Effort es el mayor ancho de banda que podrá recibir la ONU)
                i=0
                while i < num_servicios_Internet:
                        num_servicio = str(i+1)
                        true_BW_Upstream_BE = 0
                        cont=0
                        while true_BW_Upstream_BE == 0:
                                if cont == 0:
                                        print("Ancho de banda Upstream BE en Mbps [0-1244] para el
servicio " + num_servicio + ": ")
                                else:
                                        print("Ancho de banda no válido. Vuelva a probar:")
                                cont=cont+1
                                entrada = parametros1[4]
                                entrada = int(entrada)/1000
                                if entrada > -1 and entrada < 1245 and entrada >=
BW_Upstream_GR[i]:
                                        BW_Upstream_BE.append(entrada)
                                        print("El ancho de banda Upstream BE en Mbps es:",
BW_Upstream_BE[i])
                                        i=i+1
                                        true_BW_Upstream_BE = 1


                print("\n")


                # Host y puerto al que se hace la conexión Telnet para acceder al CLI
                host = "172.26.128.38"
                port = "4551"


                # Claves de acceso al CLI
                password1 = "TLNT25"
```

```
password2 = "TLNT145"

enable = "enable"


# Acceso al CLI: conexión Telnet al host y puerto indicados anteriormente

tn = telnetlib.Telnet(host,port,1)

# Mediante la función write de telnetlib, escritura de los comandos que permiten

# acceder al menú de privilegios del CLI

tn.write(password1.encode('ascii') + b"\n")

time.sleep(0.1)

tn.write(enable.encode('ascii') + b"\n")

time.sleep(0.1)

tn.write(password2.encode('ascii') + b"\n")

time.sleep(0.1)


# A la hora de introducir variables en la declaración de cadenas, aunque sean números,

# se deben introducir en forma de string. Por ello, aparece a lo largo del programa muchas

# ocasiones la función str(), que convierte una variable en string

ID_ONU = str(ID_ONU)


# A continuación, se definen todos los comandos necesarios para dar el servicio de Datos.

# Posteriormente serán ejecutados en el CLI con la función write.

# Primero se crea el canal OMCI de comunicación (con el mismo identificador que el de la ONU por convención)

# Se resetean las entidades MIB que pudiera haber y se activa fec en uplink (pasos opcionales)

inicio = "configure \n olt-device 0 \n olt-channel 0 \n onu-local " + ID_ONU + " \n omci-port  " + ID_ONU + "  \n exit \n onu-omci  " + ID_ONU + "  \n ont-data mib-reset \n exit \n fec direction uplink  " + ID_ONU + "  \n onu-local  " + ID_ONU + "  \n"

tn.write(inicio.encode('ascii') + b"\n")

time.sleep(0.2)
```

```
# Se crean, dentro del menú de la ONU a configurar, tantos Alloc-ID como servicios
while i < num_servicios_Internet:
    allocID = "alloc-id " + str(alloc_ID[i]).strip('[]') + " \n"
    tn.write(allocID.encode('ascii') + b"\n")
    time.sleep(0.2)
    i=i+1

# El comando exit hace salir hacia el menú anterior del CLI en la estructura de menús
salir = "exit \n"
tn.write(salir.encode('ascii') + b"\n")
time.sleep(0.2)

# Cada Alloc-ID irá asociado a un puerto (por convención, se utiliza el mismo identificador)
# Estos Alloc-IDs y puertos no se pueden utilizar para otras ONUs ni para otros servicios
i=0
while i < num_servicios_Internet:
    portalloc = "port " + str(port_ID[i]).strip('[]') + " alloc-id " + str(alloc_ID[i]).strip('[]') + " \n"
    tn.write(portalloc.encode('ascii') + b"\n")
    time.sleep(0.2)
    i=i+1

# Desde el menú de configuración del canal OMCI de comunicación, se crean las entidades
# MIB que forman el servicio de Internet. Estas entidades vienen en el estándar GPON.
omci = "onu-omci " + ID_ONU + " \n"
tn.write(omci.encode('ascii') + b"\n")
time.sleep(0.2)

# Creación de las entidades T-Cont (colas). Cada servicio está asociado con un T-Cont
```

```
                # y está vinculado a un Alloc-ID

                i=0

                while i < num_servicios_Internet:

                        tcont = "t-cont set slot-id 128 t-cont-id " + str(tcont_ID[i]).strip('[]') + " alloc-id
" + str(alloc_ID[i]).strip('[]') + "  \n"

                        tn.write(tcont.encode('ascii') + b"\n")

                        time.sleep(0.2)

                        i=i+1



                # Creación de MAC Brigde Service Profile. Se asociará esta entidad con los MAC Bridge

                # Port Configuration Data a través del bridge-group-id en los siguientes comandos.

                macservice = "mac-bridge-service-profile create slot-id 0 bridge-group-id 1 spanning-
tree-ind true learning-ind true atm-port-bridging-ind true priority 32000 max-age 1536 hello-time 256
forward-delay 1024 unknown-mac-address-discard false mac-learning-depth 255 dynamic-filtering-ageing-
time 1000 \n"

                # Creación del primer MAC Bridge Port Configuration Data. Irá asociado a la entidad

                # extended-vlan-tagging-operation-config-data. Para ello, el tp-type (tipo del punto

                # de terminación del MAC Bridge) ha de ser lan y el puntero tp-ptr debe tener el mismo

                # valor que el nº de instancia de la entidad extended-vlan-tagging-operation-config-data.

                macbridge1 = "mac-bridge-pcd create instance 1 bridge-id-ptr 1 port-num 1 tp-type lan
tp-ptr 257 port-priority 2 port-path-cost 32 port-spanning-tree-ind true encap-method llc lanfcs-ind forward
\n"

                tn.write(macservice.encode('ascii') + b"\n")

                time.sleep(0.2)

                tn.write(macbridge1.encode('ascii') + b"\n")

                time.sleep(0.2)



                # Creación de los restantes Mac Brigde Port Configuration Data. Irán asociado a la
entidades

                # VLAN-tagging-filter-data mediante los números de instancia. También irán asociados a
los

                # GEM Interworking Termination Point mediante el tp-type (tipo gem) y el tp-ptr, que

                # tiene que coincidir con el número de instancia del GEM Interworking Termination
Point
```

```
        i=0

        while i < num_servicios_Internet:

            macbridge2 = "mac-bridge-pcd create instance " + str(num_instancia[i]).strip('[]') + " bridge-id-ptr 1 port-num " + str(num_instancia[i]).strip('[]') + " tp-type gem tp-ptr " + str(num_instancia[i]).strip('[]') + " port-priority 0 port-path-cost 1 port-spanning-tree-ind true encap-method llc lanfcs-ind forward  \n"

            tn.write(macbridge2.encode('ascii') + b"\n")

            time.sleep(0.2)

            i=i+1



        # Creación de los GEM Port Network CTP, que irán asociado a los puertos especificados

        # anteriormente. Los identificadores tienen que coincidir con el gem-port-nwk-ct-conn-ptr

        # de los GEM Interworking Termination Point.

        i=0

        while i < num_servicios_Internet:

            gemport = "gem-port-network-ctp create instance " + str(num_instancia[i]).strip('[]') + " port-id " + str(port_ID[i]).strip('[]') + " t-cont-ptr " + str(puntero[i]).strip('[]') + " direction bidirectional traffic-mgnt-ptr-ustream 0 traffic-descriptor-profile-ptr 0 priority-queue-ptr-downstream 0 traffic-descriptor-profile-ds-ptr 0 enc-key-ring 0 \n"

            tn.write(gemport.encode('ascii') + b"\n")

            time.sleep(0.2)

            i=i+1



        # Creación de los GEM Interworking Termination Point (en este punto, se produce la

        # transformación de flujo de bytes a tramas GEM y viceversa). Estas entidades se vinculan

        # a los GEM Port Network CTP a través del campo gem-port-nwk-ctp-conn-ptr, que debe coincidir

        # con el número de instancia que  utilizado en el GEM Port Network CTP. Estas entidades también

        # se asocian con los MAC Bridge Port Configuration. Para ello, hay que seleccionar como

        # interwork-option mac-bridge-lan y el campo service-profile-ptr debe ser un 1.

        # El número de instancia debe ser el mismo que el tp-ptr del MAC Bridge Point Configuration Data.
```

```
i=0

while i < num_servicios_Internet:

    geminterworking = "gem-interworking-termination-point create instance " + str(num_instancia[i]).strip('[]') + " gem-port-nwk-ctp-conn-ptr " + str(num_instancia[i]).strip('[]') + " interwork-option mac-bridge-lan service-profile-ptr 1 interwork-tp-ptr 0 gal-profile-ptr 0 \n"

    tn.write(geminterworking.encode('ascii') + b"\n")

    time.sleep(0.2)

    i=i+1


# Creación de los VLAN Tagging Filter Data con los identficadores VLAN definidos arriba.

# Los números de instancia deben coincidir con los de los MAC Bridge Port Configuration Data asociados.

i=0

while i < num_servicios_Internet:

    vlantagging = "vlan-tagging-filter-data create instance " + str(num_instancia[i]).strip('[]') + "  forward-operation h-vid-a vlan-tag1 " + str(VLAN_ID[i]).strip('[]') + " vlan-priority1 7  vlan-tag2 null vlan-priority2 null vlan-tag3 null vlan-priority3 null vlan-tag4 null vlan-priority4 null vlan-tag5 null vlan-priority5 null vlan-tag6 null vlan-priority6 null vlan-tag7 null vlan-priority7 null vlan-tag8 null vlan-priority8 null vlan-tag9 null vlan-priority9 null vlan-tag10 null vlan-priority10 null vlan-tag11 null vlan-priority11 null vlan-tag12 null vlan-priority12 null \n"

    tn.write(vlantagging.encode('ascii') + b"\n")

    time.sleep(0.2)

    i=i+1


# Creación de la entidad Extended VLAN Tagging Operation Config Data, que será configurada en el paso posterior.

# Sirve para gestionar los identificadores VLAN. Esta entidad está asociada al primer MAC Bridge Port Configuration Data

# a través del número de instancia, que coincide con el tp-ptr del MAC Bridge Port Configuration Data.

extendedvlan = "extended-vlan-tagging-operation-config-data create instance 257 association-type pptp-eth-uni associated-me-ptr 257 \n"

tn.write(extendedvlan.encode('ascii') + b"\n")

time.sleep(0.2)
```

```
# Configuración de la entidadExtended VLAN Tagging Operation Config Data. Se debe configurar

# para cada identificador VLAN.

i=0

while i < num_servicios_Internet:

    extendedvlanconf = "extended-vlan-tagging-operation-config-data set instance 257 operations-entry filter-outer-prio filter-prio-no-tag filter-outer-vid none filter-outer-tpid none filter-inner-prio filter-prio-none filter-inner-vid " + str(VLAN_ID[i]).strip('[]') + " filter-inner-tpid none filter-ethertype none treatment-tag-to-remove 1 treatment-outer-prio none treatment-outer-vid copy-from-inner treatment-outer-tpid tpid-de-copy-from-outer treatment-inner-prio 0 treatment-inner-vid " + str(VLAN_ID[i]).strip('[]') + " treatment-inner-tpid tpid-de-copy-from-inner\n"

    tn.write(extendedvlanconf.encode('ascii') + b"\n")

    time.sleep(0.2)

    i=i+1


# El comando exit hace salir hacia el menú anterior del CLI en la estructura de menús

salir = "exit \n"

tn.write(salir.encode('ascii') + b"\n")

time.sleep(0.2)


# Configuración de las reglas VLAN en la OLT asociadas a los puertos definidos al principio de la función

i=0

while i < num_servicios_Internet:

    reglasvlan = "vlan uplink configuration port-id " + str(port_ID[i]).strip('[]') + " min-cos 0 max-cos 7 de-bit disable primary-tag-handling false \n vlan uplink handling port-id  " + str(port_ID[i]).strip('[]') + "  primary-vlan none destination datapath c-vlan-handling no-change s-vlan-handling no-change new-c-vlan 0 new-s-vlan 0 \n"

    tn.write(reglasvlan.encode('ascii') + b"\n")

    time.sleep(0.2)

    i=i+1


# Creación de los perfiles de ancho de banda en sentido Downstream con los parámetros

# definidos por el usuario anteriormente.

i=0
```

```
while i < num_servicios_Internet:

perfildownstreamconf = "policing downstream profile committed-max-bw " + str(BW_Downstream_GR[i]).strip('[]') + " committed-burst-size 1023 excess-max-bw " + str(BW_Downstream_Excess[i]).strip('[]') + " excess-burst-size 1023 \n"

tn.write(perfildownstreamconf.encode('ascii') + b"\n")

time.sleep(0.2)

i=i+1


# Nombre del fichero en el que se guardará la configuración del servicio de Internet

nombre_archivo = 'Servicio_Internet_ONU_MAC_' + MAC_ONU + '.txt'


# Al crear los perfiles de  ancho de banda en sentido de bajada, el CLI devuelve un

# índice de perfil. Para asociar los perfiles creados, se deben utilizar esos índices

# de perfil y asociarlos a los puertos definidos anteriormente. Para ello, se vuelcan los datos

# enviados y recibidos del CLI en un fichero, del que se extraerán esos índices.

datos_perfil = tn.read_very_eager().decode()

outfile = open(nombre_archivo, 'a')

outfile.write(datos_perfil)

outfile.close()

outfile = open(nombre_archivo, 'r')

lines = outfile.readlines()

true_ds_profile = 0

# Se busca el índice de perfil que primero aparezca en el fichero.

i=0

for i in range (0,500):

if true_ds_profile == 1:

break

j = str(i)

cadena = 'downstream_profile_index: ' + j + ' '

for line in lines:
```

```
                    if cadena in line:
                        j = int(j)
                        ds_profile_index[0] = j
                        true_ds_profile = 1

        outfile.close()

        # El índice encontrado corresponde al primer servicio. En caso de haber más, el índice
        # de perfil será ese aumentado en una unidad.
        i=0
        for i in range (0,num_servicios_Internet):
            ds_profile_index[i] = ds_profile_index[0] + i

        # Asignación de los perfiles de ancho de banda Downstream a los puertos correspondientes
        # mediante los índices de perfil buscados anteriormente.
        i=0
        while i < num_servicios_Internet:
            perfildownstreamassign = "policing downstream port-configuration entity port-id " + str(port_ID[i]).strip('[]') + " ds-profile-index " + str(ds_profile_index[i]).strip('[]') + " \n"
            tn.write(perfildownstreamassign.encode('ascii') + b"\n")
            time.sleep(0.2)
            i=i+1

        # El comando exit hace salir hacia el menú anterior del CLI en la estructura de menús
        salir = "exit \n"
        tn.write(salir.encode('ascii') + b"\n")
        time.sleep(0.2)

        # Desde el menú de configuración del algoritmo DBA, se definirián los perfiles de
```

```
# ancho de banda en sentido Upstream

dba = "pon \n dba pythagoras 0 \n "

tn.write(dba.encode('ascii') + b"\n")

time.sleep(0.2)


# Cada perfil upstream irá asociado a un Alloc-ID y estará configurado con los

# parámetros que haya definido el usuario.

i=0

while i < num_servicios_Internet:

        perfilupstream = "sla " + str(alloc_ID[i]).strip('[]') + " service data status-report nsr gr-bw " + str(BW_Upstream_GR[i]).strip('[]') + " gr-fine 0 be-bw " + str(BW_Upstream_BE[i]).strip('[]') + " be-fine 0 \n"

        tn.write(perfilupstream.encode('ascii') + b"\n")

        time.sleep(0.2)

        i=i+1


# El comando end hace salir directamente al modo privilegiado en la estructura

# de menús del CLI

final = "end \n"

tn.write(final.encode('ascii') + b"\n")

time.sleep(0.2)


# Se vuelcan todos los datos en el fichero definido anteriormente (la opción 'a'

# hace que los datos se añadan al final del fichero) de forma que el fichero recogerá

# toda la configuración del servicio de Internet

data = tn.read_very_eager().decode()

outfile = open(nombre_archivo, 'a')

outfile.write(data)

outfile.write("\n\n\n")

outfile.close()
```

```
# Una vez configurado el servicio, se muestra un mensaje al usuario
print("Servicio de Internet configurado. \n")

# Actualización del fichero XML con los datos configurados
doc = etree.parse("configuracionGPON.xml")
# Extracción del elemento raíz
redGPON = doc.getroot()

# Obtención del índice correspondiente a la ONU que se está configurando.
# Para ello, se hace una comparación de las diferentes direcciones MAC con
# la de la ONU a configurar.
index=0
true_index=0
while true_index==0:
        for attr,value in redGPON[index].items():
                if value == MAC_ONU:
                        true_index=1
                        break
                index=index+1

# Se borra cualquier configuración anterior que en este caso, al configurar un
# nuevo servicio, se va a desechar
num_servicios = len(redGPON[index])
i=0
while i<num_servicios:
        redGPON[index].remove(redGPON[index][0])
        i=i+1
```

```
                # Se actualiza la parte del árbol correspondiente a la ONU en cuestión con los
                # parámetros que ha definido el usuario. Se utiliza la función SubElement para
                # crear el servicio así como los parámetros del mismo.
                i=0
                while i < num_servicios_Internet:
                        Servicio = etree.SubElement(redGPON[index], "Servicio", tipo='Internet')
                        VLAN = etree.SubElement(redGPON[index][i], "VLAN_ID").text = str(VLAN_ID[i]).strip('[]')
                        BW_Down_GR = etree.SubElement(redGPON[index][i], "BW_Down_GR").text = str(BW_Downstream_GR[i]).strip('[]')
                        BW_Down_Excess = etree.SubElement(redGPON[index][i], "BW_Down_Excess").text = str(BW_Downstream_Excess[i]).strip('[]')
                        BW_Up_GR = etree.SubElement(redGPON[index][i], "BW_Up_GR").text = str(BW_Upstream_GR[i]).strip('[]')
                        BW_Up_BE = etree.SubElement(redGPON[index][i], "BW_Up_BE").text = str(BW_Upstream_BE[i]).strip('[]')
                        i=i+1

                # Finalmente se crea el nuevo árbol actualizado y se escribe en el fichero
                doc = etree.ElementTree(redGPON)
                doc.write("configuracionGPON.xml")

                return

        # Función servicio_Video(ID_ONU,MAC_ONU,num_servicios_Video): permite configurar
        # servicios de Internet + Vídeo (el servicio multicast del vídeo debe ir sobre uno
        # Ethernet para gestionar el tráfico IMGP asociado) y toma como parámetros el identificador de la ONU
        # (posición en el vector de direcciones MAC), la MAC y el número de servicios a configurar.
        # Guarda la configuración actual en un fichero txt (esta configuración es temporal y se
        # mantiene mientras no se apague la red ni se cambie al modo de gestión TGMS).
        # La función también actualiza el fichero XML para recuperar configuraciones
        # cuando se apaga la red o se cambia al TGMS.
```

```
def servicio_Video(ID_ONU,MAC_ONU,num_servicios_Video):


        # Creación de los vectores en los que se almacenarán los parámetros
        # internos de configuración
        port_ID = []
        alloc_ID = []
        tcont_ID = []
        num_instancia = []
        puntero = []
        ds_profile_index = []


        # Dependiendo del número de servicios, se asginan tantos valores como sean necesarios
        # a los vectores anteriormente creados
        i=0
        while i < num_servicios_Video:
                # Para que no se solapen puertos y allocs-ID, se asignan en función del identficiador
                # de la ONU y del número de servicio en cuestión
                port_ID.append(600+100*ID_ONU+i)
                alloc_ID.append(600+100*ID_ONU+i)
                # Estos valores se asignan de este modo también para evitar solapamientos
                num_instancia.append(i+3)
                tcont_ID.append(i)
                puntero.append(32768+i)
                ds_profile_index.append(i)
                i=i+1


        # Flags que marcarán la salida de los diferentes bucles cuando el parámetro
        # requerido haya sido introducido de forma válida
```

```
true_VLAN = 0

true_BW_Downstream_GR = 0

true_BW_Downstream_Excess = 0

true_BW_Upstream_GR = 0

true_BW_Upstream_BE = 0


# Creación de los vectores en los que se almacenarán los parámetros

# de configuracion que el USUARIO deberá introducir: identificador de VLAN,

# ancho de banda Downstream garantizado y en exceso y ancho de banda Upstream

# garantizado y Best Effort

VLAN_ID = []

BW_Downstream_GR = []

BW_Downstream_Excess = []

BW_Upstream_GR = []

BW_Upstream_BE = []


# En los sucesivos bucles, se irán pidiendo los parámetros de configuración al usuario.

# IMPORTANTE: EN EL CLI, NO HAY QUE DEFINIR ANCHOS DE BANDA PARA EL SERVICIO MULTICAST.

# EL ANCHO DE BANDA UTILIZADO POR EL VÍDEO ES EL DEFINIDO EN EL SERVICIO ETHERNET.


# Primer bucle: se piden el identificador VLAN que corresponda a cada servicio

# La VLAN 833 conecta con un servidor DHCP que asigna la dirección mientras que

# la VLAN 806 recibe una IP de forma estática (no la tiene que introducir el usuario)

i=0

while i < num_servicios_Video:

        num_servicio = str(i+1)

        true_VLAN = 0

        cont=0
```

```
                # Las etiquetas VLAN van de 1 a 4094. Si el usuario introduce un valor fuera de
ese
                # rango, se le volverá a pedir que introduzca el valor. Hay que recordar que la
red
                # solo ofrece servicio en las VLAN 833 y 806
                while true_VLAN == 0:
                        if cont == 0:
                                print("Identficador VLAN [1-4094] para el servicio " +
num_servicio + " (833 - Servidor DHCP, 806 - Dirección IP estática): ")
                        else:
                                print("El identificador VLAN no es válido. Vuelva a probar:")
                        cont = cont+1
                        entrada = parametros1[1]
                        entrada = int(entrada)
                        if entrada > 0 and entrada < 4095:
                                VLAN_ID.append(entrada)
                                print("El identificador VLAN para el servicio " +
num_servicio + " es:", VLAN_ID[i])
                                i=i+1
                                true_VLAN = 1

        print("\n")

        # Segundo bucle: se pide el ancho de banda garantizado en sentido Downstream.
        # Este ancho de banda se introduce en Kbps y debe estar entre 0 y 2488000 (aunque
        # a efectos prácticos no tiene sentido meter más de 600000 Kbps, aproximadamente).
        # El CLI solo admite múltiplos de 64 Kbps de manera que el programa trunca el
        # valor del usuario para que coincida con un múltiplo de 64 Kbps.
        i=0
        while i < num_servicios_Video:
                num_servicio = str(i+1)
                true_BW_Downstream_GR = 0
```

```
                cont=0
                while true_BW_Downstream_GR == 0:
                        if cont == 0:
                                print("Ancho de banda Downstream garantizado en Kbps [0-2488000] para el servicio " + num_servicio + ": ")
                        else:
                                print("Ancho de banda no válido. Vuelva a probar:")
                        cont=cont+1
                        entrada = parametros1[2]
                        entrada = int(entrada)
                        entrada = entrada / 64
                        entrada = math.floor(entrada)
                        entrada = entrada * 64
                        if entrada > -1 and entrada < 2488000:
                                BW_Downstream_GR.append(entrada)
                                print("El ancho de banda Downstream garantizado en Kbps es:", BW_Downstream_GR[i])
                                i=i+1
                                true_BW_Downstream_GR = 1

        # Tercer bucle: se pide el ancho de banda en exceso en sentido Downstream.
        # Este ancho de banda se introduce en Kbps y debe estar entre 0 y 2488000 (aunque
        # a efectos prácticos no tiene sentido que la suma del garantizado y exceso sea
        # mayor que 600000 Kbps, aproximadamente). El CLI solo admite múltiplos de 64 Kbps
        # de manera que el programa trunca el valor del usuario para que coincida con
        # un múltiplo de 64 Kbps.
        i=0
        while i < num_servicios_Video:
                num_servicio = str(i+1)
                true_BW_Downstream_Excess = 0
```

```
                cont=0
                while true_BW_Downstream_Excess == 0:
                        if cont == 0:
                                print("Ancho de banda Downstream en exceso en Kbps [0-2488000] para el servicio " + num_servicio + ": ")
                        else:
                                print("Ancho de banda no válido. Vuelva a probar:")
                        cont=cont+1
                        entrada = 0
                        if entrada > -1 and entrada < 2488000:
                                BW_Downstream_Excess.append(entrada)
                                print("El ancho de banda Downstream en exceso en Kbps es:", BW_Downstream_Excess[i])
                                i=i+1
                                true_BW_Downstream_Excess = 1


        print("\n")


        # Cuarto bucle: se pide el ancho de banda garantizado en sentido Upstream.
        # Este ancho de banda se introduce en Mbps y debe estar entre 0 y 1244 (aunque
        # a efectos prácticos no tiene sentido meter más de 600 Mbps, aproximadamente).
        i=0
        while i < num_servicios_Video:
                num_servicio = str(i+1)
                true_BW_Upstream_GR = 0
                cont=0
                while true_BW_Upstream_GR == 0:
                        if cont == 0:
                                print("Ancho de banda Upstream garantizado en Mbps [0-1244] para el servicio " + num_servicio + ": ")
```

```
                else:
                    print("Ancho de banda no válido. Vuelva a probar:")
                cont=cont+1
                entrada = parametros1[4]
                entrada = int(entrada)/1000
                if entrada > -1 and entrada < 1245:
                    BW_Upstream_GR.append(entrada)
                    print("El ancho de banda Upstream garantizado en Mbps es:", BW_Upstream_GR[i])
                    i=i+1
                    true_BW_Upstream_GR = 1

        print("\n")

        # Quinto bucle: se pide el ancho de banda garantizado Best Effort Upstream.
        # Este ancho de banda se introduce en Mbps, debe estar entre 0 y 1244 (aunque
        # a efectos prácticos no tiene sentido meter más de 600 Mbps, aproximadamente) y
        # tiene que ser igualo mayor que el ancho de banda Upstream garantizado (el valor Best
        # Effort es el mayor ancho de banda que podrá recibir la ONU)
        i=0
        while i < num_servicios_Video:
            num_servicio = str(i+1)
            true_BW_Upstream_BE = 0
            cont=0
            while true_BW_Upstream_BE == 0:
                if cont == 0:
                    print("Ancho de banda Upstream BE en Mbps [0-1244] para el servicio " + num_servicio + ": ")
                else:
                    print("Ancho de banda no válido. Vuelva a probar:")
```

```
                cont=cont+1
                entrada = parametros1[4]
                entrada = int(entrada)/1000
                if entrada > -1 and entrada < 1245 and entrada >= BW_Upstream_GR[i]:
                    BW_Upstream_BE.append(entrada)
                    print("El ancho de banda Upstream BE en Mbps es:", BW_Upstream_BE[i])
                    i=i+1
                    true_BW_Upstream_BE = 1

        print("\n")

        # Host y puerto al que se hace la conexión Telnet para acceder al CLI
        host = "172.26.128.38"
        port = "4551"

        # Claves de acceso al CLI
        password1 = "TLNT25"
        password2 = "TLNT145"
        enable = "enable"

        # Acceso al CLI: conexión Telnet al host y puerto indicados anteriormente
        tn = telnetlib.Telnet(host,port,1)
        # Mediante la función write de telnetlib, escritura de los comandos que permiten
        # acceder al menú de privilegios del CLI
        tn.write(password1.encode('ascii') + b"\n")
        time.sleep(0.1)
        tn.write(enable.encode('ascii') + b"\n")
        time.sleep(0.1)
```

```
                tn.write(password2.encode('ascii') + b"\n")

                time.sleep(0.1)


                # A la hora de introducir variables en la declaración de cadenas, aunque sean números,

                # se deben introducir en forma de string. Por ello, aparece a lo largo del programa
muchas

                # ocasiones la función str(), que convierte una variable en string

                ID_ONU = str(ID_ONU)


                # A continuación, se definen todos los comandos necesarios para dar el servicio de Datos.

                # Posteriormente serán ejecutados en el CLI con la función write.

                # Primero se crea el canal OMCI de comunicación (con el mismo identificador que el de
la ONU por convención)

                # Se resetean las entidades MIB que pudiera haber y se activa fec en uplink (pasos
opcionales)

                inicio = "configure \n olt-device 0 \n olt-channel 0 \n onu-local " + ID_ONU + " \n omci-
port  " + ID_ONU + "  \n exit \n onu-omci  " + ID_ONU + "  \n ont-data mib-reset \n exit \n fec direction
uplink  " + ID_ONU + "  \n onu-local  " + ID_ONU + "  \n"

                tn.write(inicio.encode('ascii') + b"\n")

                time.sleep(0.2)


                # Se crean, dentro del menú de la ONU a configurar, tantos Alloc-ID como servicios
Ethernet

                # ¡¡¡IMPORTANTE!!! El servicio multicast ocupa solo el canal descendente y  no
necesita la

                # asignación de un Alloc-ID para el acceso al canal ascendente de transmisión ni
parámetros de SLA.

                i=0

                while i < num_servicios_Video:

                        allocID = "alloc-id " + str(alloc_ID[i]).strip('[]') + " \n"

                        tn.write(allocID.encode('ascii') + b"\n")

                        time.sleep(0.2)

                        i=i+1
```

```
                # El comando exit hace salir hacia el menú anterior del CLI en la estructura de menús.
                salir = "exit \n"
                tn.write(salir.encode('ascii') + b"\n")
                time.sleep(0.2)

                # Cada Alloc-ID irá asociado a un puerto (por convención, se utiliza el mismo identificador).
                # Estos Alloc-IDs y puertos no se pueden utilizar para otras ONUs ni para otros servicios.
                # El servicio multicast irá sobre el puerto 4094 (podría ser otro) pero como se explicó arriba,
                # no necesita la asignación de un Alloc-ID
                i=0
                while i < num_servicios_Video:
                        portalloc = "port " + str(port_ID[i]).strip('[]') + " alloc-id " + str(alloc_ID[i]).strip('[]') + " \n"
                        tn.write(portalloc.encode('ascii') + b"\n")
                        time.sleep(0.2)
                        i=i+1

                # Desde el menú de configuración del canal OMCI de comunicación, se crean las entidades
                # MIB que forman el servicio de Internet y Vídeo. Estas entidades vienen en el estándar GPON.
                omci = "onu-omci " + ID_ONU + " \n"
                tn.write(omci.encode('ascii') + b"\n")
                time.sleep(0.2)

                # Creación de las entidades T-Cont (colas). Cada servicio Ethernet está asociado
                # con un T-Cont y está vinculado a un Alloc-ID
                i=0
                while i < num_servicios_Video:
                        tcont = "t-cont set slot-id 128 t-cont-id " + str(tcont_ID[i]).strip('[]') + " alloc-id " + str(alloc_ID[i]).strip('[]') + "  \n"
```

```
                tn.write(tcont.encode('ascii') + b"\n")

                time.sleep(0.2)

                i=i+1


        # Creación de MAC Brigde Service Profile. Se asociará esta entidad con los MAC Bridge

        # Port Configuration Data a través del bridge-group-id en los siguientes comandos.

        macservice = "mac-bridge-service-profile create slot-id 0 bridge-group-id 1 spanning-tree-ind true learning-ind true atm-port-bridging-ind true priority 32000 max-age 1536 hello-time 256 forward-delay 1024 unknown-mac-address-discard false mac-learning-depth 255 dynamic-filtering-ageing-time 1000 \n"

        # Creación del primer MAC Bridge Port Configuration Data. Irá asociado a la entidad

        # extended-vlan-tagging-operation-config-data. Para ello, el tp-type (tipo del punto

        # de terminación del MAC Bridge) ha de ser lan y el puntero tp-ptr debe tener el mismo

        # valor que el nº de instancia de la entidad extended-vlan-tagging-operation-config-data.

        # Esta entidad está vinculada a los servicios Ethernet.

        macbridge1 = "mac-bridge-pcd create instance 1 bridge-id-ptr 1 port-num 1 tp-type lan tp-ptr 257 port-priority 2 port-path-cost 32 port-spanning-tree-ind true encap-method llc lanfcs-ind forward \n"

                tn.write(macservice.encode('ascii') + b"\n")

                time.sleep(0.2)

                tn.write(macbridge1.encode('ascii') + b"\n")

                time.sleep(0.2)


        # Creación del segundo MAC Bridge Port Configuration Data. Esta entidad está

        # vinculada al servicio multicast. Se asocia a la entidad Multicast GEM Interworking

        # Termination Poing. Para ello, el tipo de puntero (tp-type) ha de ser de tipo multicast

        # (mc-gem) y el tp-ptr ha de coincidir con el nº de instancia la entidad Multicast GEM

        # Interworking Termination Poing.

        macbridge2 = "mac-bridge-pcd create instance 2 bridge-id-ptr 1 port-num 2 tp-type mc-gem tp-ptr 2 port-priority 0 port-path-cost 1 port-spanning-tree-ind true encap-method llc lanfcs-ind forward \n"

                tn.write(macbridge2.encode('ascii') + b"\n")

                time.sleep(0.2)
```

```
                # Creación de los restantes Mac Brigde Port Configuration Data. Irán asociado a la entidades

                # VLAN-tagging-filter-data mediante los números de instancia. También irán asociados a los

                # GEM Interworking Termination Point mediante el tp-type (tipo gem) y el tp-ptr, que

                # tiene que coincidir con el número de instancia del GEM Interworking Termination Point.

                # Estas entidades están vinculadas a los servicios de tipo Ethernet.

                i=0

                while i < num_servicios_Video:

                        macbridge3 = "mac-bridge-pcd create instance " + str(num_instancia[i]).strip('[]') + " bridge-id-ptr 1 port-num " + str(num_instancia[i]).strip('[]') + " tp-type gem tp-ptr " + str(num_instancia[i]).strip('[]') + " port-priority 0 port-path-cost 1 port-spanning-tree-ind true encap-method llc lanfcs-ind forward  \n"

                        tn.write(macbridge3.encode('ascii') + b"\n")

                        time.sleep(0.2)

                        i=i+1



                # Creación del GEM Port Network CTP vinculado al servicio multicast. Está asociado al puerto

                # 4094 (en el que va el servicio multicast). Se diferencia de la entidad que forma el servicio Ethernet

                # en que direction ya no es de tipo bidirectional sino de tipo ani-to-uni. Asimismo, esta entidad está

                # asociada a la entidad Multicast GEM Interworking Termination Point: el nº de instancia debe coincidir con

                # el campo gem-port-nwk-ctp-conn-ptr de la otra entidad.

                gemport_multicast = "gem-port-network-ctp create instance 2 port-id 4094 t-cont-ptr 0 direction ani-to-uni traffic-mgnt-ptr-ustream 0 traffic-descriptor-profile-ptr 0 priority-queue-ptr-downstream 0 traffic-descriptor-profile-ds-ptr 0 enc-key-ring 0  \n"

                tn.write(gemport_multicast.encode('ascii') + b"\n")

                time.sleep(0.2)



                # Creación de los GEM Port Network CTP, que irán asociado a los puertos especificados

                # anteriormente para los servicios Ethernet. Los identificadores tienen que coincidir
```

```
                # con el gem-port-nwk-ct-conn-ptr de los GEM Interworking Termination Point.

                i=0

                while i < num_servicios_Video:

                        gemport = "gem-port-network-ctp create instance " + str(num_instancia[i]).strip('[]') + " port-id " + str(port_ID[i]).strip('[]') + " t-cont-ptr " + str(puntero[i]).strip('[]') + " direction bidirectional traffic-mgnt-ptr-ustream 0 traffic-descriptor-profile-ptr 0 priority-queue-ptr-downstream 0 traffic-descriptor-profile-ds-ptr 0 enc-key-ring 0 \n"

                        tn.write(gemport.encode('ascii') + b"\n")

                        time.sleep(0.2)

                        i=i+1



                # Creación del Multicast GEM Interworking Termination Point (en este punto, se produce la

                # transformación de flujo de bytes a tramas GEM y viceversa). Esta entidad está vinculada al

                # servicio multicast. Se vincula al GEM Port Netwok CTP de tipo multicast mediante el campo

                # gem-port-nwk-ctp-conn-ptr. También se asocia con elMAC Bridge Port Configuration Data de

                # tipo multicast. Para ello, hay que seleccionar como interwork-option mac-bridge y el campo

                # service-profile-ptr debe tener el valor que se indica en la declaración del comando.

                # El número de instancia debe ser el mismo que el tp-ptr del MAC Bridge Point Configuration Data asociado.

                multicast_geminterworking = "multicast-gem-interworking-termination-point create instance 2 gem-port-nwk-ctp-conn-ptr 2 interwork-option mac-bridge service-prof-ptr 65535 interwork-tp-ptr 0 gal-prof-ptr 65535 gal-lpbk-config 0 \n"

                tn.write(multicast_geminterworking.encode('ascii') + b"\n")

                time.sleep(0.2)



                # Creación de los GEM Interworking Termination Point (en este punto, se produce la

                # transformación de flujo de bytes a tramas GEM y viceversa). Estas entidades se vinculan

                # a los GEM Port Network CTP a través del campo gem-port-nwk-ctp-conn-ptr, que debe coincidir

                # con el número de instancia que  utilizado en el GEM Port Network CTP. Estas entidades también
```

```
# se asocian con los MAC Bridge Port Configuration. Para ello, hay que seleccionar como

# interwork-option mac-bridge-lan y el campo service-profile-ptr debe ser un 1.

# El número de instancia debe ser el mismo que el tp-ptr del MAC Bridge Point Configuration Data.

i=0

while i < num_servicios_Video:

    geminterworking = "gem-interworking-termination-point create instance " + str(num_instancia[i]).strip('[]') + " gem-port-nwk-ctp-conn-ptr " + str(num_instancia[i]).strip('[]') + " interwork-option mac-bridge-lan service-profile-ptr 1 interwork-tp-ptr 0 gal-profile-ptr 0 \n"

    tn.write(geminterworking.encode('ascii') + b"\n")

    time.sleep(0.2)

    i=i+1


# Creación de los VLAN Tagging Filter Data con los identficadores VLAN definidos arriba.

# Los números de instancia deben coincidir con los de los MAC Bridge Port Configuration Data asociados.

i=0

while i < num_servicios_Video:

    vlantagging = "vlan-tagging-filter-data create instance " + str(num_instancia[i]).strip('[]') + "  forward-operation h-vid-a vlan-tag1 " + str(VLAN_ID[i]).strip('[]') + " vlan-priority1 7  vlan-tag2 null vlan-priority2 null vlan-tag3 null vlan-priority3 null vlan-tag4 null vlan-priority4 null vlan-tag5 null vlan-priority5 null vlan-tag6 null vlan-priority6 null vlan-tag7 null vlan-priority7 null vlan-tag8 null vlan-priority8 null vlan-tag9 null vlan-priority9 null vlan-tag10 null vlan-priority10 null vlan-tag11 null vlan-priority11 null vlan-tag12 null vlan-priority12 null \n"

    tn.write(vlantagging.encode('ascii') + b"\n")

    time.sleep(0.2)

    i=i+1


# Creación de la entidad Extended VLAN Tagging Operation Config Data, que será configurada en el paso posterior.

# Sirve para gestionar los identificadores VLAN. Esta entidad está asociada al primer MAC Bridge Port Configuration Data

# a través del número de instancia, que coincide con el tp-ptr del MAC Bridge Port Configuration Data.
```

```
                extendedvlan = "extended-vlan-tagging-operation-config-data create instance 257 association-type pptp-eth-uni associated-me-ptr 257 \n"

                tn.write(extendedvlan.encode('ascii') + b"\n")

                time.sleep(0.2)


                # Configuración de la entidadExtended VLAN Tagging Operation Config Data. Se debe configurar

                # para cada identificador VLAN.

                i=0

                while i < num_servicios_Video:

                        extendedvlanconf = "extended-vlan-tagging-operation-config-data set instance 257 operations-entry filter-outer-prio filter-prio-no-tag filter-outer-vid none filter-outer-tpid none filter-inner-prio filter-prio-none filter-inner-vid " + str(VLAN_ID[i]).strip('[]') + " filter-inner-tpid none filter-ethertype none treatment-tag-to-remove 1 treatment-outer-prio none treatment-outer-vid copy-from-inner treatment-outer-tpid tpid-de-copy-from-outer treatment-inner-prio 0 treatment-inner-vid " + str(VLAN_ID[i]).strip('[]') + " treatment-inner-tpid tpid-de-copy-from-inner\n"

                        tn.write(extendedvlanconf.encode('ascii') + b"\n")

                        time.sleep(0.2)

                        i=i+1


                # El comando exit hace salir hacia el menú anterior del CLI en la estructura de menús

                salir = "exit \n"

                tn.write(salir.encode('ascii') + b"\n")

                time.sleep(0.2)


                # Configuración de las reglas VLAN en la OLT asociadas a los puertos definidos al principio de la función.

                # Estos puertos son los vinculados a los servicios de tipo Ethernet.

                i=0

                while i < num_servicios_Video:

                        reglasvlan = "vlan uplink configuration port-id " + str(port_ID[i]).strip('[]') + " min-cos 0 max-cos 7 de-bit disable primary-tag-handling false \n vlan uplink handling port-id  " + str(port_ID[i]).strip('[]') + "  primary-vlan none destination datapath c-vlan-handling no-change s-vlan-handling no-change new-c-vlan 0 new-s-vlan 0 \n"

                        tn.write(reglasvlan.encode('ascii') + b"\n")
```

```
                time.sleep(0.2)

                i=i+1


        # Creación de los perfiles de ancho de banda en sentido Downstream con los parámetros

        # definidos por el usuario anteriormente.

        i=0

        while i < num_servicios_Video:

                perfildownstreamconf = "policing downstream profile committed-max-bw " + str(BW_Downstream_GR[i]).strip('[]') + " committed-burst-size 1023 excess-max-bw " + str(BW_Downstream_Excess[i]).strip('[]') + " excess-burst-size 1023 \n"

                tn.write(perfildownstreamconf.encode('ascii') + b"'\n")

                time.sleep(0.2)

                i=i+1


        # Nombre del fichero en el que se guardará la configuración del servicio de Internet + Video

        nombre_archivo = 'Servicio_Internet+Video_ONU_MAC_' + MAC_ONU + '.txt'

        # Al crear los perfiles de  ancho de banda en sentido de bajada, el CLI devuelve un

        # índice de perfil. Para asociar los perfiles creados, se deben utilizar esos índices

        # de perfil y asociarlos a los puertos definidos anteriormente. Para ello, se vuelcan los datos

        # enviados y recibidos del CLI en un fichero, del que se extraerán esos índices.

        datos_perfil = tn.read_very_eager().decode()

        outfile = open(nombre_archivo, 'a')

        outfile.write(datos_perfil)

        outfile.close()

        outfile = open(nombre_archivo, 'r')

        lines = outfile.readlines()

        true_ds_profile = 0

        # Se busca el índice de perfil que primero aparezca en el fichero.

        i=0

        for i in range (0,500):
```

```
                if true_ds_profile == 1:
                        break
                j = str(i)
                cadena = 'downstream_profile_index: ' + j + ' '
                for line in lines:
                        if cadena in line:
                                j = int(j)
                                ds_profile_index[0] = j
                                true_ds_profile = 1

        outfile.close()

        # El índice encontrado corresponde al primer servicio. En caso de haber más, el índice
        # de perfil será ese aumentado en una unidad.
        i=0
        for i in range (0,num_servicios_Video):
                ds_profile_index[i] = ds_profile_index[0] + i

        # Asignación de los perfiles de ancho de banda Downstream a los puertos correspondientes
        # mediante los índices de perfil buscados anteriormente. Estos anchos de banda se asignan
        # a los servicios de tipo Ethernet. Para el servicio multicast, no hay que definir anchos
        # de banda.
        i=0
        while i < num_servicios_Video:
                perfildownstreamassign = "policing downstream port-configuration entity port-id " + str(port_ID[i]).strip('[]') + " ds-profile-index " + str(ds_profile_index[i]).strip('[]') + " \n"
                tn.write(perfildownstreamassign.encode('ascii') + b"\n")
                time.sleep(0.2)
                i=i+1
```

```
# El comando exit hace salir hacia el menú anterior del CLI en la estructura de menús
salir = "exit \n"
tn.write(salir.encode('ascii') + b"\n")
time.sleep(0.2)

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

# Cada perfil upstream irá asociado a un Alloc-ID y estará configurado con los
# parámetros que haya definido el usuario. Como ya se se explicó anteriormente,
# solo los servicios de tipo Ethernet tienen asociados Alloc-IDs; el de tipo
# multicast, no.
i=0
while i < num_servicios_Video:
        perfilupstream = "sla " + str(alloc_ID[i]).strip('[]') + " service data status-report nsr gr-bw " + str(BW_Upstream_GR[i]).strip('[]') + " gr-fine 0 be-bw " + str(BW_Upstream_BE[i]).strip('[]') + " be-fine 0 \n"
        tn.write(perfilupstream.encode('ascii') + b"\n")
        time.sleep(0.2)
        i=i+1

# El comando end hace salir directamente al modo privilegiado en la estructura
# de menús del CLI
final = "end \n"
tn.write(final.encode('ascii') + b"\n")
time.sleep(0.2)
```

```
# Se vuelcan todos los datos en el fichero definido anteriormente (la opción 'a'
# hace que los datos se añadan al final del fichero) de forma que el fichero recogerá
# toda la configuración del servicio de Internet + Vídeo
data = tn.read_very_eager().decode()
outfile = open(nombre_archivo, 'a')
outfile.write(data)
outfile.write("\n\n\n")
outfile.close()

# Una vez configurado el servicio, se muestra un mensaje al usuario
print("Servicio de Internet+Vídeo configurado. \n")

# Actualización del fichero XML con los datos configurados
doc = etree.parse("configuracionGPON.xml")
# Extracción del elemento raíz
redGPON = doc.getroot()

# Obtención del índice correspondiente a la ONU que se está configurando.
# Para ello, se hace una comparación de las diferentes direcciones MAC con
# la de la ONU a configurar.
index=0
true_index=0
while true_index==0:
        for attr,value in redGPON[index].items():
                if value == MAC_ONU:
                        true_index=1
                        break
                index=index+1
```

```
                # Se borra cualquier configuración anterior que en este caso, al configurar un
                # nuevo servicio, se va a desechar.
                num_servicios = len(redGPON[index])
                i=0
                while i<num_servicios:
                        redGPON[index].remove(redGPON[index][0])
                        i=i+1

                # Se actualiza la parte del árbol correspondiente a la ONU en cuestión con los
                # parámetros que ha definido el usuario. Se utiliza la función SubElement para
                # crear el servicio así como los parámetros del mismo.
                i=0
                while i < num_servicios_Video:
                        Servicio = etree.SubElement(redGPON[index], "Servicio",
tipo='Internet+Video')
                        VLAN = etree.SubElement(redGPON[index][i], "VLAN_ID").text =
str(VLAN_ID[i]).strip('[]')
                        BW_Down_GR = etree.SubElement(redGPON[index][i],
"BW_Down_GR").text = str(BW_Downstream_GR[i]).strip('[]')
                        BW_Down_Excess = etree.SubElement(redGPON[index][i],
"BW_Down_Excess").text = str(BW_Downstream_Excess[i]).strip('[]')
                        BW_Up_GR = etree.SubElement(redGPON[index][i], "BW_Up_GR").text =
str(BW_Upstream_GR[i]).strip('[]')
                        BW_Up_BE = etree.SubElement(redGPON[index][i], "BW_Up_BE").text =
str(BW_Upstream_BE[i]).strip('[]')
                        i=i+1

                # Finalmente se crea el nuevo árbol modificado y se escribe en el fichero XML
                doc = etree.ElementTree(redGPON)
                doc.write("configuracionGPON.xml")

                return
```

```
# Función modificar_configuracion(ID_ONU,MAC_ONU,nombre_archivo): esta función permite
# cambiar la configuración existente en una ONU. Toma como argumento el identificador
# de ONU, la dirección MAC y el nombre del archivo donde está la configuración que será
# reemplazada. La función en primer lugar borra la configuración existente tomando datos de este
# fichero y posteriormente, llama a las funciones de servicio_Internet o servicio_Video
# dependiendo de lo que el usuario haya seleccionado para la nueva configuración.
def modificar_configuracion(ID_ONU,MAC_ONU,nombre_archivo):

    # Host y puerto al que se hace la conexión Telnet para acceder al CLI
    host = "172.26.128.38"
    port = "4551"

    # Claves de acceso al CLI
    password1 = "TLNT25"
    password2 = "TLNT145"
    enable = "enable"

    # Acceso al CLI: conexión Telnet al host y puerto indicados anteriormente
    tn = telnetlib.Telnet(host,port,1)
    # Mediante la función write de telnetlib, escritura de los comandos que permiten
    # acceder al menú de privilegios del CLI
    tn.write(password1.encode('ascii') + b"\n")
    time.sleep(0.1)
    tn.write(enable.encode('ascii') + b"\n")
    time.sleep(0.1)
    tn.write(password2.encode('ascii') + b"\n")
    time.sleep(0.1)
```

```
# Se abre en modo lectura el fichero en el que está la configuración antigua
outfile = open(nombre_archivo, 'r')
# Se almacenan todas las líneas del fichero
lines = outfile.readlines()
port_ID = 0

# En este bucle, se buscarán los puertos configurados y se borrarán los perfiles
# de anchos de banda asociados a estos puertos. En las funciones de servicio_Internet
# y servicio_Video que se llamarán posteriormente, se borrarán las entidades MIB, de forma
# que los servicios quedarán borrados por completo.
k=0
for k in range (0,10000):
        j = str(k)
        # Se recorren todas las líneas del fichero hasta encontrar los puertos configurados
        cadena = 'configuration entity port-id ' + j + ' '
        for line in lines:
                if cadena in line:
                        j = int(j)
                        port_ID = j
                        port_ID = str(port_ID)
                        # Cada vez que se encuentra un puerto configurado, se borra el perfil de ancho
                        # de banda asociado
                        borrar_perfil = "configure \n olt-device 0 \n olt-channel 0 \n no policing downstream port-configuration entity port-id " + port_ID + " \n end \n"
                        tn.write(borrar_perfil.encode('ascii') + b"\n")
                        time.sleep(0.1)

outfile.close()
# Finalmente se borra el fichero de configuración
```

```
os.remove(nombre_archivo)


# Actualización del archivo XML con los datos borrados (se actualizarán en las
# funciones correspondientes de Internet o Vídeo)
doc = etree.parse("configuracionGPON.xml")
# Extracción del elemento raíz
redGPON = doc.getroot()


# Obtención del índice correspondiente a la ONU cuya configuración se va a borrar.
# Para ello, se hace una comparación de las diferentes direcciones MAC con
# la de la ONU en cuestión.
index=0
true_index=0
while true_index==0:
        for attr,value in redGPON[index].items():
                if value == MAC_ONU:
                        true_index=1
                        break
                index=index+1


# Se borra la configuración de dicha ONU
num_servicios = len(redGPON[index])
i=0
while i<num_servicios:
        redGPON[index].remove(redGPON[index][0])
        i=i+1


# Finalmente se crea el nuevo árbol modificado y se escribe en el fichero XML
doc = etree.ElementTree(redGPON)
```

```
doc.write("configuracionGPON.xml")


# Una vez borrada la configuración, se pide al usuario que seleccione la nueva
# configuración que desea para la ONU. Dos opciones: servicio de Internet y
# servicio de Internet + Video.
true_opcion = 0
while true_opcion == 0:
        print("\n¿Qué servicio quiere crear ahora en esta ONU? Escoja la opción:")
        print("1 - Servicio de Internet")
        print("2 - Servicio de Internet + Vídeo")
        opcion = int(parametros1[3])+1
        # Si el usuario no selecciona una opción válida, se le vuelve a pedir.
        if opcion == 1 or opcion == 2:
                true_opcion = 1
        else:
                print("Opción no válida. Vuelva a probar.")


# En caso de seleccionar el servicio de Internet, se le pide el nº de servicios.
# Este nº podría ser mayor que 2 pero realmente, no tiene sentido configurar más de 2
# servicios puesto que solo hay 2 VLAN que proporcionen servicio a esta red.
if opcion == 1:
        print("\nConfiguración del servicio de Internet:")
        true_Internet = 1
        num_servicios_Internet = 1
        print("\n")
        # En caso de que el usuario seleccione un número de servicios de Internet que no
sea 0,
        # se llama a la función de servicio_Internet para configurar la ONU con los
nuevos parámetros
        servicio_Internet(ID_ONU,MAC_ONU,num_servicios_Internet)
```

```
                # En caso de seleccionar el servicio de Internet+Video, se le pide el nº de servicios.
                # Este nº podría ser mayor que 2 pero realmente, no tiene sentido configurar más de 2
                # servicios puesto que solo hay 2 VLAN que proporcionen servicio a esta red.
                elif opcion == 2:
                        print("\nConfiguración del servicio de Internet + Vídeo:")
                        true_Video = 1
                        num_servicios_Video = 1
                        print("\n")
                        # En caso de que el usuario seleccione un número de servicios de Internet + Video que no sea 0,
                        # se llama a la función de servicio_Internet para configurar la ONU con los nuevos parámetros
                        servicio_Video(ID_ONU,MAC_ONU,num_servicios_Video)
                return

        # Función borrar_configuracion(ID_ONU,MAC_ONU,nombre_archivo): esta función permite
        # borrar la configuración de una ONU. Toma como argumento el identificador
        # de ONU, la dirección MAC y el nombre del archivo donde está la configuración que
        # será borrada.
        def borrar_configuracion(ID_ONU,MAC_ONU,nombre_archivo):

                # Host y puerto al que se hace la conexión Telnet para acceder al CLI
                host = "172.26.128.38"
                port = "4551"

                # Claves de acceso al CLI
                password1 = "TLNT25"
                password2 = "TLNT145"
                enable = "enable"
```

```
# Acceso al CLI: conexión Telnet al host y puerto indicados anteriormente
tn = telnetlib.Telnet(host,port,1)
# Mediante la función write de telnetlib, escritura de los comandos que permiten
# acceder al menú de privilegios del CLI
tn.write(password1.encode('ascii') + b"\n")
time.sleep(0.1)
tn.write(enable.encode('ascii') + b"\n")
time.sleep(0.1)
tn.write(password2.encode('ascii') + b"\n")
time.sleep(0.1)

# Se abre en modo lectura el fichero en el que está la configuración a borrar.
outfile = open(nombre_archivo, 'r')
# Se almacenan todas las líneas del fichero.
lines = outfile.readlines()
port_ID = 0

# En este bucle, se buscarán los puertos configurados y se borrarán los perfiles
# de anchos de banda asociados a estos puertos. Posteriormente, se borrarán las entidades
# que forman los servicios.
k=0
for k in range (0,10000):
        j = str(k)
        # Se recorren todas las líneas del fichero hasta encontrar los puertos configurados.
        cadena = 'configuration entity port-id ' + j + ' '
        for line in lines:
                if cadena in line:
                        j = int(j)
                        port_ID = j
```

```
                                        port_ID = str(port_ID)

                                        # Cada vez que se encuentra un puerto configurado, se borra el perfil de ancho

                                        # de banda asociado.

                                        borrar_perfil = "configure \n olt-device 0 \n olt-channel 0 \n no policing downstream port-configuration entity port-id " + port_ID + " \n end \n"

                                        tn.write(borrar_perfil.encode('ascii') + b"\n")

                                        time.sleep(0.1)


                outfile.close()

                ID_ONU = str(ID_ONU)

                # Tras borrar los perfiles de ancho de banda, se borrar las entidades MIB presentes en

                # el canal OMCI asociado a la ONU.

                borrar_MIB = "configure \n olt-device 0 \n olt-channel 0 \n onu-local " + ID_ONU + " \n omci-port  " + ID_ONU + "  \n exit \n onu-omci  " + ID_ONU + " \n ont-data mib-reset \n exit \n end \n"

                tn.write(borrar_MIB.encode('ascii') + b"\n")

                time.sleep(2)

                # Finalmente se borra el fichero de configuración.

                os.remove(nombre_archivo)

                # Una vez borrado, se muestra un mensaje informando al usuario.

                print("\nLa configuración de la ONU con MAC " + MAC_ONU + " ha sido borrada.\n")


        # Actualización del archivo XML con los datos borrados

                doc = etree.parse("configuracionGPON.xml")

                # Extracción del elemento raíz

                redGPON = doc.getroot()


                # Obtención del índice correspondiente a la ONU cuya configuración se va a borrar.

                # Para ello, se hace una comparación de las diferentes direcciones MAC con

                # la de la ONU en cuestión.

                index=0
```

```
                true_index=0
                while true_index==0:
                        for attr,value in redGPON[index].items():
                                if value == MAC_ONU:
                                        true_index=1
                                        break
                                index=index+1

                # Se borra la configuración de dicha ONU
                num_servicios = len(redGPON[index])
                i=0
                while i<num_servicios:
                        redGPON[index].remove(redGPON[index][0])
                        i=i+1

                # Finalmente se crea el nuevo árbol modificado y se escribe en el fichero XML
                doc = etree.ElementTree(redGPON)
                doc.write("configuracionGPON.xml")

                return
        def Main():
        #DATOS INICIALES
                flagBW =0
                flagRole=0
                storeRole=0
                extraByte=0
                storeID=int("0xffffffffffffffff",16)
        #ESTABLECIMIENTO DE COMUNICACION----------------------------------------------------------
-------------------------------
                host = '192.168.56.101'
```

```
		port = 6633
		mySocket = socket.socket()
		mySocket.connect((host,port))

		mapa = bytes.fromhex('00000010')
		element = HelloElemVersionbitmap(bitmaps=mapa)
		element.length = 8
		sendHello = Hello(xid = 10, elements = [element])
		message = sendHello.pack()
		mySocket.send(message)
		# data = mySocket.recv(1024)

		# if(len(data)>16):
		#	sendFeaturesReply = FeaturesReply(xid=int.from_bytes(data[20:],byteorder='big'), datapath_id=5, n_buffers=256, n_tables=254, auxiliary_id=0, capabilities=71,reserved=0)
		#	message = sendFeaturesReply.pack()
		#	mySocket.send(message)
	#-----------------------------------------------------------------------------------------------------------------------
		while True:
			extraByte=0
			data = mySocket.recv(1024)
			longReal = len(data)
			while True:
				if (data[1+extraByte] == 5):
					sendFeaturesReply = FeaturesReply(xid=int.from_bytes(data[(4+extraByte):],byteorder='big'), datapath_id=5, n_buffers=256, n_tables=254, auxiliary_id=0, capabilities=71,reserved=0)
					message = sendFeaturesReply.pack()
					mySocket.send(message)
				elif (data[1+extraByte] == 20):
```

```
                                                sendBarrierReply = BarrierReply(xid =
int.from_bytes(data[(4+extraByte):(8+extraByte)],byteorder='big'))

                                                message = sendBarrierReply.pack()

                                                mySocket.send(message)

                                        elif (data[1+extraByte] == 24):

                                                if ((data[(8+extraByte):(12+extraByte)] ==
bytes.fromhex('00 00 00 00')) and (flagRole == 0)):

                                                        flagRole += 1

                                                        storeRole =
ControllerRole.OFPCR_ROLE_EQUAL

        sendRoleReply=RoleReply(xid=int.from_bytes(data[(4+extraByte):(8+extraByte)],byteorder='big'
), role=storeRole, generation_id=storeID)

                                                        message = sendRoleReply.pack()

                                                        mySocket.send(message)

                                                elif (data[(8+extraByte):(12+extraByte)] ==
bytes.fromhex('00 00 00 00')):

                                                        storeID =
int.from_bytes(data[(16+extraByte):(24+extraByte)],byteorder='big')

        sendRoleReply=RoleReply(xid=int.from_bytes(data[(4+extraByte):(8+extraByte)],byteorder='big'
), role=storeRole, generation_id=storeID)

                                                        message = sendRoleReply.pack()

                                                        mySocket.send(message)

                                                elif ((data[(8+extraByte):(12+extraByte)] ==
bytes.fromhex('00 00 00 02')) or (data[(8+extraByte):(12+extraByte)] == bytes.fromhex('00 00 00 03'))or
(data[(8+extraByte):(12+extraByte)] == bytes.fromhex('00 00 00 01'))):

                                                        storeRole =
int.from_bytes(data[(8+extraByte):(12+extraByte)],byteorder='big')

                                                        storeID =
int.from_bytes(data[(16+extraByte):(24+extraByte)],byteorder='big')

        sendRoleReply=RoleReply(xid=int.from_bytes(data[(4+extraByte):(8+extraByte)],byteorder='big'
), role=storeRole, generation_id=storeID)

                                                        message = sendRoleReply.pack()

                                                        mySocket.send(message)

                                        elif (data[1+extraByte] == 18):
```

```
                                        if (data[(8+extraByte):(10+extraByte)] == bytes.fromhex('00 00')):

                                            pruebaDesc=Desc(mfr_desc="GPO, UVA", hw_desc="Agente OpenFlow para configurar OLT", sw_desc="1.0", serial_num="None", dp_desc="None")


        sendMultipartReply=MultipartReply(xid=int.from_bytes(data[(4+extraByte):(8+extraByte)],byteorder='big'), multipart_type=0, flags=0, body=pruebaDesc)

                                            message = sendMultipartReply.pack()

                                            mySocket.send(message)

                                    elif (data[1+extraByte] == 14):

                                        if (int.from_bytes(data[(2+extraByte):(4+extraByte)],byteorder='big') == 120):

                                            parametros1[1] = int.from_bytes(data[(74+extraByte):(76+extraByte)],byteorder='big')-2**12

                                            parametros1[0] = data[(64+extraByte):(70+extraByte)].hex()

                                            parametros1[3] = data[83+extraByte]

                                            if(parametros1[3] == 255):

                                                parametros1[3] = 1

                                            else:

                                                parametros1[3] = 0

                                    elif (data[1+extraByte] == 29):

                                        if (flagBW == 0):

                                            parametros1[2] = int.from_bytes(data[(20+extraByte):(24+extraByte)],byteorder='big')

                                            flagBW += 1

                                        elif (flagBW == 1):

                                            parametros1[4] = int.from_bytes(data[(20+extraByte):(24+extraByte)],byteorder='big')

                                    if(longReal != int.from_bytes(data[(2+extraByte):(4+extraByte)],byteorder='big')):

                                        longReal -= int.from_bytes(data[(2+extraByte):(4+extraByte)],byteorder='big')

                                        extraByte += int.from_bytes(data[(2+extraByte):(4+extraByte)],byteorder='big')
```

```
                                        elif(longReal == int.from_bytes(data[(2+extraByte):(4+extraByte)],byteorder='big')):
                                            break
                                    if ((parametros1[0] != 0) & (parametros1[1] != 0) & (parametros1[2] != 0) & (parametros1[4] != 0) & ((parametros1[3] == 1) or (parametros1[3] == 0))):
                                        break


            # FUNCIÓN MAIN DEL PROGRAMA
            # Desde aquí, se irán llamando a las funciones definidas arriba en función de lo que
            # escoja el usuario.
            # En primer lugar, se le da la bienvenida al programa de gestión y se le pide la dirección MAC
            # de la ONU. Este será siempre el primer paso: el usuario deberá introducir la dirección MAC de
            # la ONU que quiere manejar y a partir irá seleccionando qué hacer.
            print("Bienvenido al programa de gestión de la red GPON.")
            print("A continuación, introduzca la MAC de la ONU y la acción que desea llevar a cabo.\n")
            # La función get_ID_ONU() es llamada al principio del programa para almacenar en el vector
            # MAC las direcciones MAC de todas las ONUs que están conectadas a la red GPON.
            MAC = get_ID_ONU()

            # Este flag marca el fin del programa. Cuando se activa, se finaliza el bucle y
            # se sale del programa.
            true_FIN = 0
            while true_FIN == 0:
                cont=0
                true_MAC = 0
                # Cada vez que se termina de realizar una acción con una ONU, se vuelve a este punto
                # en el que se pide de nuevo una dirección MAC para trabajar con una ONU.
                # La dirección MAC que introduce el usuario se compara con las del vector MAC. Mientras
```

```
                # no introduzca una reconocida por el programa, se le seguirá pidiendo de nuevo
la MAC.
                # Esto es lo que se denominará menú principal.
                while true_MAC == 0:
                        if cont == 0:
                                print("Introduzca la MAC de la ONU que va a configurar: ")
                        else:
                                print("MAC no reconocida. Introduzca la MAC de nuevo: ")
                        print(parametros1[0][8:10])
                        print(parametros1[0][10:])
                        direccion_MAC = "54-4c-52-49-5b-01-"+parametros1[0][8:10]+"-
"+parametros1[0][10:]
                        print(parametros1[0])
                        print(direccion_MAC)
                        cont = cont+1
                        # Bucle que recorre el vector en el que se almacenan las direcciones
MAC
                        for i in range(0,len(MAC)):
                                if direccion_MAC == MAC[i]:
                                        print("\nLa dirección MAC ha sido reconocida
correctamente.")
                                        true_MAC = 1
                                        ID_ONU = i


                # Estos son los nombres que toman los archivos de configuracion, en caso de
existir.
                nombre_archivo1 = 'Servicio_Internet_ONU_MAC_' + direccion_MAC + '.txt'
                nombre_archivo2 = 'Servicio_Internet+Video_ONU_MAC_' + direccion_MAC
+ '.txt'
                # Si existe nombre_archivo1, la ONU tiene configurado un servicio de Internet.
                if path.exists(nombre_archivo1):
                        print("\nLa ONU con dirección MAC " + direccion_MAC + " ya tiene
configurado servicio de Internet.\n")
```

```
                # EL programa permite al usuario ver, modificar o borrar la
configuración
                # del servicio de Internet así volver al menú principal o finalizar el
programa.
                true_opcion = 1
                modificar_configuracion(ID_ONU,direccion_MAC,nombre_archivo1)
                true_FIN = 1

        # Si existe nombre_archivo2, la ONU tiene configurado un servicio de Internet +
Video.
        elif path.exists(nombre_archivo2):
                print("\nLa ONU con dirección MAC " + direccion_MAC + " ya tiene
configurado servicio de Internet y vídeo.\n")
                # EL programa permite al usuario ver, modificar o borrar la
configuración
                # del servicio de Internet + Video así como volver al menú principal o
                # finalizar el programa.
                true_opcion = 1
                modificar_configuracion(ID_ONU,direccion_MAC,nombre_archivo2)
                true_FIN = 1
        # En caso de que no existan los archivos de configuración, la ONU está sin
configurar.
        # Por tanto, se marcará a 0 el flag true_opcion y el programa entrará en el
siguiente bucle.
        else:
                true_opcion = 0

        # Se define la variable opción, que permitirá en el siguiente bucle llamar a una
función u otra
        # dependiendo de la elección del usuario. Se le asigna un valor por defecto que
no
        # coincida con ninguna opción de las que llaman a otras funciones (ahora
mismo, los
        # valores de 0 a 4 están utilizados). Ver opciones más abajo.
        opcion = 5
```

```
                # Si el programa entra en este bucle, quiere decir que la ONU está sin
configurar. Por tanto,
                # el usuario podrá configurar para dicha ONU servicio de Internet, servicio de
Internet + Video,
                # cargar la última configuración activa (antes de que se apagara la red o se
cambiara al TGMS),
                # volver al menú principal o finalizar el programa.
                while true_opcion == 0:
                        opcion = parametros1[3]
                        opcion = int(opcion)+1
                        if opcion == 1 or opcion == 2:
                                true_opcion = 1
                        else:
                                # Si el usuario introduce un nº de opción no válido, se le
vuelve a preguntar.
                                print("Opción no válida. Vuelva a probar.")
                # Si el usuario introduce un 1, se va a configurar servicio de Internet.
                if opcion == 1:
                        print("\nConfiguración del servicio de Internet:")
                        true_Internet = 0
                        # Se le pide el nº de servicios. Este nº podría ser mayor que 2 pero
realmente,
                        # no tiene sentido configurar más de 2 servicios puesto que solo hay 2
VLAN
                        # que proporcionen servicio a esta red.
                        print("\n")
                        # En caso de que el número de servicios sea mayor que 0, se llamará a
la función
                        # servicio_Internet.
                        servicio_Internet(ID_ONU,direccion_MAC,1)
                        # Tanto si se han configurado 1 o más servicios de Internet, como si no
se ha
                        # configurado ninguno, el programa pregunta al usuario si desea seguir
configurando la red.
```

```
                    true_FIN = 1


                # Si el usuario introduce un 2, se va a configurar servicio de Internet+Video.
                elif opcion == 2:
                    print("\nConfiguración del servicio de Vídeo:")
                    true_Video = 0
                    # Se le pide el nº de servicios. Este nº podría ser mayor que 2 pero realmente,
                    # no tiene sentido configurar más de 2 servicios puesto que solo hay 2 VLAN
                    # que proporcionen servicio a esta red.
                    servicio_Video(ID_ONU,direccion_MAC,1)
                    # Tanto si se han configurado 1 o más servicios de Internet+Video, como si no se ha
                    # configurado ninguno, el programa pregunta al usuario si desea seguir configurando la red.
                    true_FIN=1
        if __name__ == '__main__':
            Main()
```