TecnoCampus Escola Superior Politècnica

Centre adscrit a la upf. Universitat Pompeu Fabra Barcelona

Grau en Enginyeria Informàtica de Gestió i Sistemes d'Informació

GESTIÓN CENTRALIZADA DE LA RED SDN

Memoria

ERIK VARGA DUDASH

TUTOR: PERE BARBERAN AGUT

2020-2021

TecnoCampus Mataró-Maresme

Abstract

SDN technologies are advancing to create a new era in the network. This paper presents an analysis of the main existing theories, how current networks change and what benefits they provide, as well as their management, using tools that allow emulating the behavior of these large-scale networks, in a reduced and controlled environment. In this way, it is concluded that despite all the theoretical benefits that these technologies seem to present, there is still a long way to go for their wide deployment, and even more time, for the replacement of current technologies.

Resum

Les tecnologies SDN avancen per crear una nova era en la xarxa. En aquest treball es fa una anàlisi sobre les principals teories existents, com canvien les xarxes actuals i que beneficis aporten, així com la seva gestió, utilitzant eines que permeten emular el comportament d'aquestes xarxes de gran escala, en un entorn reduït i controlat. D'aquesta manera, es conclou que malgrat tots els beneficis teòrics que semblen presentar aquestes tecnologies, encara queda un llarg camí per al seu ampli desplegament, i encara més temps, per a la substitució de les tecnologies actuals.

Resumen

Las tecnologías *SDN* avanzan para crear una nueva era en la red. En este trabajo se hace un análisis sobre las principales teorías existentes, como cambian las redes actuales y que beneficios aportan, así como su gestión, utilizando herramientas que permiten emular el comportamiento de estas redes de gran escala, en un entorno reducido y controlado. De esta manera, se concluye en que pese a todos los beneficios teóricos que parecen presentar estas tecnologías, aún queda un largo camino para su amplio despliegue, y aún más tiempo, para la sustitución de las tecnologías actuales.

Índice

Índice de figuras

Índice de tablas

Glosario de términos

SDN	Software Defined Network-ing
OF	OpenFlow
OVS	OpenVSwitch
ODL	OpenDayLight
OFM	OpenFlowManager
DLUX	OpenDayLight User Interface

1. Introducción

Durante los últimos años, las tendencias en el mundo de las redes han comenzado a adoptar nuevas ideas, una de ellas siendo *SDN*.

El problema que existe hoy en día con las redes tradicionales puede llegar a ser grande y complejo. La rápida expansión de los servicios en internet hace que las demandas en las redes sean muy elevadas, la clara desventaja de las redes tradicionales respecto a los servidores solo hace que estos servicios se vean limitados por las redes que recorren.

La elección de esta temática viene dada para dar a conocer la importancia de los problemas a los que se enfrentan las redes actuales, así como mostrar los caminos más populares o mejor aceptados hoy en día, para así promover el interés y concienciar de los cambios que se tienen que producir en un futuro, o al menos, la dirección que tienen que tomar.

La primera parte del trabajo se centra en explicar el funcionamiento y los componentes, de las tecnologías utilizadas en la parte práctica. Empezando por OpenFlow y un pequeño glosario introductorio para dar a conocer la terminología especifica. Continuando con los diferentes puertos que se utilizan dentro de un switch específico para OpenFlow, para posteriormente adentrarse más aun y explicar cómo se procesan los paquetes, y como se ejecutan las instrucciones, una vez los paquetes acceden dentro de uno de estos switch especializados. Posteriormente se explica cómo funcionan las conexiones, es decir, el tipo de comunicación que existe a la hora de hablar de un switch OpenFlow, como se conecta a otros dispositivos, por ejemplo, un controlador, y como lleva a cabo el procesado de estos mensajes entre uno y otro.

Una vez se entiende como trabaja internamente el protocolo OpenFlow, y cada switch que lo implementa, se explica *OpenDayLight*, el controlador para redes SDN más utilizado y polivalente. Se explica primero los principales componentes de los que está formado, para posteriormente explicar cómo se realiza la comunicación entre los dispositivos. Aquí es donde se separa en dos, el primer tipo de comunicación se realiza desde los dispositivos de red hasta el controlador, y el segundo, comunica a las aplicaciones de usuario con el mismo controlador. Por último, se habla brevemente sobre el módulo central de OpenDayLight, el cual se encarga de almacenar los datos de las configuraciones y comunicar su estado.

A partir de aquí, se procede a configurar todas las herramientas necesarias para poder emular topologías SDN. Se comienza instalando el programa de emulación de redes GNS3, este junto a su máquina virtual, la cual permite emular un mayor número de dispositivos diferentes, son la base para la creación de topologías SDN. Esta máquina virtual se virtualiza sobre VMware, que, a su vez, virtualiza también a otra máquina virtual, la de Mininet, que se utiliza para la rápida simulación de topologías, prescindiendo así de la necesidad de configurar múltiples dispositivos uno a uno.

Una vez las herramientas están instaladas y preparadas para su uso, se pasa a la configuración interna. Aquí se explica cómo configurar el controlador, los switch de OpenFlow físicos y aquellos simulados por Mininet. En esta parte, comienzan a aparecer ciertos problemas. Las versiones más modernas del controlador no permiten utilizar aquellos módulos, que son necesarios para demostrar la gestión de las redes mediante *SDN*, desde los entornos configurados previamente. Por lo tanto, se utilizan versiones anteriores, y es seleccionada la última de ellas que parece disponer de estos módulos.

Después de configurar la parte interna de las herramientas, es decir, la configuración básica de las topologías, se procede a explicar el uso de las herramientas que permiten realizar esta gestión de la red. Se comienza con la gestión desde los mismos switch, utilizando una línea de comandos para configurar y consultar los estados. Después se procede con las aplicaciones proporcionadas por el mismo controlador, ambas ellas permiten visualizar el estado completo de la topología, así como ver la configuración de cada dispositivo y crear cambios.

En el momento que se conocen los componentes de cada herramienta y sus funciones básicas, se procede a explicar cómo programar cambios en las topologías, mediante cuatro formas diferentes. La programación más básica comienza con la creación de nuevos flujos, mediante la línea de comandos, en el propio switch, siendo esta, la forma menos dinámica y real. Las tres formas restantes, hacen uso de peticiones directamente al controlador, el cual, a su vez, escribe estos cambios en un switch en concreto. De estas tres formas mencionadas, dos de ellas hacen uso de las aplicaciones del controlador, *DLUX* y *OFM*, para crear estos cambios mediante una interfaz de usuario fácil de utilizar. La última de las formas presentadas, hace uso de un programa de creación de peticiones personalizadas, *Postman*, para la programación de estos flujos.

2. Marco teórico

El mundo de las redes ha estado sufriendo un cambio en sus paradigmas con el intento de adoptar un mayor control sobre la programabilidad de estas.

A día de hoy, muchas redes están formadas por tipologías en las que cada dispositivo de enrutado, switch o router, dispone de su propio software, en el que se toman decisiones a partir de la vista que tiene dicho dispositivo, estas decisiones de enrutado se guardan en tablas internas llamadas *routing tables*, que son programadas manualmente por administradores de sistemas, o bien, vienen por defecto con software privado del fabricante, además de integrar protocolos estándar que usan la mayoría de dispositivos.

Si se requiere hacer algún cambio en las configuraciones de uno o muchos dispositivos, hay que hacerlo manualmente en cada uno de ellos, lo que supone múltiples inconvenientes, como por ejemplo una mayor dificultad a la hora de realizar experimentos a gran escala, lo cual promueve un menor ratio de innovación respecto a otros avances [1]. También, el incremento en la complejidad de estos dispositivos hace que el coste de estas redes sea más elevado.

El inconveniente más grande es que a la hora de necesitar incorporar algún nuevo dispositivo o servicio en la red con el objetivo de escalar esta según las necesidades, se convierta en un trabajo tedioso y costoso en tiempo y recursos.

Todos estos problemas han impulsado el estudio de diferentes alternativas a las redes tradicionales. Se puede considerar al primero de estas alternativas al *Active Networking*, una tecnología desarrollada en los años noventa, con la idea de reducir los costes de computación, que cada vez eran mayores a la hora de desplegar grandes servicios. Este se basaba en la programabilidad del plano de datos, siendo así el precursor del *Network Function Virtualization* (*NFV*).

A Pesar de las ventajas teóricas que prometía, la falta de rendimiento y seguridad, sumado a la falta de una necesidad inmediata en la práctica, han hecho que esta tecnología no fuese extensamente utilizada, pero sí importante para el impulso de ideas posteriores.

Con el paso de los años, los requerimientos han ido aumentando, el tamaño de las redes de las proveedoras de internet y la rápida expansión de los servidores en cuanto a capacidad de computación empezaban a superar las limitaciones impuestas por las redes tradicionales.

Estas tendencias han desencadenado en la separación del plano de control de él de datos, un nuevo modelo que tenía como objetivo un plano de control programable, diferenciándose así del *Active Networking*. Además, apostaba por una vista y configuración completa de la red, a diferencia de la visión específica de un dispositivo de las redes hasta entonces. [2]

Los avances de los diferentes estudios y aplicaciones desarrolladas en la separación del *data* y *control plane* han servido como base para la creación de un protocolo llamado OpenFlow. Este era un avance en varios sentidos respecto a las anteriores tecnologías. En primer lugar, proporcionaba una mayor funcionalidad, siendo así un precursor del *Policy-based routing* (*PBR*), el cual tiene como objetivo de separar el enrutado de paquetes por diferentes políticas, es decir, según el distinto tipo de aplicación, este puede ser por ejemplo voz, video, email, … algo que se consigue usando *Access Control Lists* (*ACL*) que definen distintos criterios para cada paquete. Una de las razones por las que *PBR* no acabó de encajar tan bien como se esperaba, es que, además de tardar un par de años en ofrecer rendimiento similar al de las redes tradicionales, seguían siendo, tal y como las redes tradicionales, tecnología privada del fabricante, y por lo tanto poco flexible a las necesidades específicas de cada usuario. [3]

Por otro lado, *OpenFlow* ofrece un despliegue inmediato sobre el hardware de los switches existentes. Con esto marcaba la principal diferencia con los demás competidores hasta ahora, que fallaban a la hora de una rápida implementación.

Aquí es donde nacen las tecnologías *SDN*, más que una tecnología específica, es un estándar, el cual tiene como objetivo separar el plano de control del plano de datos en un switch, es decir, separar la toma de decisiones del hardware a un controlador único, el cual ve la topología completa y puede tomar estas decisiones a partir de esta vista comunicando cambios directamente al hardware, sin la necesidad de configurar uno a uno.

Durante los últimos años, *OpenFlow* se ha convertido ya en el protocolo estándar para la implementación de tecnologías *SDN*, su rápida implementación y el hecho de ser open source han integrado *OpenFlow* como protocolo de muchos de los switches disponibles de *SDN*

Por otro lado, a la hora de interactuar con aplicaciones de usuario para controlar las redes de la manera más flexible posible se encuentra *OpenDayLight*, el cual se ha posicionado como el controlador más popular para gestionar redes definidas por software. Concretamente se encarga de la comunicación de usuario a protocolo *SDN*, siendo este *OpenFlow*, proporcionando una interfaz cómoda y fácil de usar. Ya que *OpenDayLight* también es open source, los clientes pueden implementar sus propios servicios en *Java*, que es el lenguaje en el que este está escrito.

Otros controladores aparte del *OpenDayLight* son por ejemplo las plataformas *SD-Access* y *SD-WAN*, el intento de sumarse a moda de las redes definidas por software por parte de *Cisco*. *Hewlett Packard* proporciona ya en su línea de switches el soporte a *OpenFlow* y un controlador virtual de redes. Siendo estas alternativas menos explotadas hoy en día, ya sea por el rápido éxito de *OpenFlow* y *OpenDayLight*, siendo estas open source, o bien por la falta de madurez de estas últimas opciones. [4]

Tal y como se ha comentado antes, uno de los problemas con las redes tradicionales es la experimentación a gran escala. En este caso cabe destacar sobre todo *Mininet*, el cual este permite emular topologías con switches *OpenFlow* con simples comandos y desplegarlos sobre hardware de manera rápida. [5,6,7]

3. Objetivos y alcance

- Entender el funcionamiento del protocolo *OpenFlow*.
- Entender el funcionamiento del controlador *OpenDayLight*.
- Entender el funcionamiento de *Mininet*.
- Mostrar la configuración del software de virtualización para topologías *GNS3*.
- Mostrar el uso de una topología *SDN* con *GNS3*.
- Mostrar los beneficios aportados al aplicar las tecnologías *SDN* respecto a topologías tradicionales.
- Entender otras alternativas a las tecnologías expuestas y posibles futuras tendencias.

Con este proyecto se pretende estudiar las tecnologías *SDN* actuales, por lo tanto, entender el funcionamiento teórico de las partes presentadas en los objetivos es uno de los primeros límites. Como parte final de los apartados teóricos se presentará con unas tendencias a futuro, estas limitadas a un repaso simple sin entrar en gran detalle.

La parte práctica de este proyecto está limitada solamente a configuraciones y pruebas. Esto quiere decir que, en un principio, no se va a realizar ningún desarrollo de nuevas herramientas. Dicho esto, incluido en el apartado de configuración, está previsto la utilización de código para poder automatizar algunos procesos.

4. Análisis de referentes

A la hora de realizar este trabajo, se ha visto, en algunos casos, una falta de referentes actualizados y completos.

Haciendo referencia a los apartados más teóricos, la información proporcionada por las fuentes, en la mayoría de los casos, se ha presentado de una manera adecuada. Los temas referentes a *OpenFlow*, sobre todo, están completos de información, la principal de ellas disponible directamente en la documentación oficial. Aunque completa, esta información no está del todo actualizada, las versiones más nuevas de esta documentación constan del año 2015, y aunque versiones más nuevas se han prometido a lo largo de los años, a día de hoy no hay más noticias, y por ello, aunque sin gran impacto en el trabajo, toda la teoría explicada y herramientas utilizadas, se basan en un versión de *OpenFlow* con más de seis años. Por otra parte, en cuanto a *OpenDayLight*, la información disponible tanto en las documentaciones oficiales, como fuera de estas, está bastante más actualizada respecto a *OpenFlow*. Pese a esto, la información encontrada no está del todo completa, existen módulos de *OpenDayLight* que aún no están documentados. También, en general, la información está bien presentada, pero es poco extensa. En cuanto al impacto que ha producido al desarrollo del trabajo, para el apartado teórico, el impacto ha sido bajo. En cuanto a la documentación presentada sobre *Mininet*, no se ha encontrado ningún problema. La información ha sido suficiente y presentada de una manera clara.

En cuanto a los referentes de los apartados prácticos, existen fuentes que han proporcionado información respecto a configuraciones básicas de las herramientas. Algunas de estas fuentes, han sido útiles para el progreso de los temas relacionados con la configuración de *GNS3* y la composición básica de la topología presentada. La demás información, o bien se ha encontrado a trozos en distintas páginas, o encontrada manualmente durante el desarrollo. Sobre todo, los temas referentes a la programación de flujos, se ha visto una falta de calidad de información. Todo esto acompañado de que, la mayoría de las fuentes, son del año 2016 en los mejores casos.

5. Metodología

El proyecto consta de dos partes, siendo esta la parte teórica y la práctica. Los métodos de búsqueda de información para el apartado teórico están basados en varias fuentes distintas. La primera de ellas son las documentaciones oficiales, estas ofrecen los mejores detalles por parte de los desarrolladores de la tecnología, así ofreciendo una fuente confiable y oficial de información. Otra de las fuentes de información son las páginas de primeros resultados de las búsquedas por Google, ya que estas nos ofrecen información con la mayor tendencia de búsquedas, así permitiendo contrastar las tecnologías más populares. Por último, se va a hacer referencia a libros especializados, estos permiten tener el punto de vista de los mayores expertos de este ámbito, eso sí, desde otra perspectiva a la de los desarrolladores, así mostrando los mejores casos de uso y ejemplos que funcionan en la práctica.

Por otro lado, para la parte práctica, se va a recurrir a las documentaciones oficiales, igual que en el apartado teórico, para realizar las configuraciones y pruebas de la forma en la que estas han sido pensadas y desarrolladas. En el momento de no poder progresar con documentaciones oficiales, se va a recurrir también a los tutoriales disponibles en internet, siendo estos en muchos casos escritos por personas que siguen utilizando dicha tecnología pese a la falta de documentación oficial, o, falta de soporte por parte del desarrollador.

En el caso de no poder consultar información referente al problema, se va a intentar resolver estos a base de conocimiento aprendido hasta la fecha y con ayuda de documentación teórica.

Como metodología de desarrollo se va a utilizar el modelo en cascada, o en este caso, un modelo en cascada en paralelo. Esto quiere decir que tanto la parte teórica como la práctica, van a ser desarrolladas en paralelo a partir de cierto punto.

Se va a comenzar con unas fases de teoría planeadas, cada una de estas sirviendo como precursor para la siguiente fase, y la parte teórica en su totalidad va a servir como precursor para que la parte práctica también se desarrolle en un formato de cascada, estando cada fase planteada y sirviendo para mover hacia delante la siguiente fase. Una vez ambas partes estén a punto de finalizar, en paralelo, se van a juntar para hablar sobre las conclusiones y análisis de los resultados.

6. OpenFlow

El protocolo *OpenFlow* es una de las principales propuestas de redes programables en el campo del *SDN*. Su objetivo principal es permitir la separación de la toma de decisiones dentro un switch físico y delegarlo a un controlador centralizado.

En este tema se explica las partes que forman *OpenFlow*, desde los componentes físicos y virtuales, como *OpenFlow* procesa la información y cómo se comunica con los controladores que lo implementan.

6.1. Glosario

Ingress Port: Puerto de entrada a un switch OpenFlow.

Output Port: Puerto de salida de un switch OpenFlow.

Match: Emparejamiento según un criterio especificado.

Flow: También llamado flujo, hace match con paquetes y realiza acciones.

Flow table: Tabla de flujos, almacena de manera ordenada los flujos.

6.2. Puertos OpenFlow

Los puertos *OpenFlow* son las interfaces responsables de pasar paquetes entre los controladores de *OpenFlow* y el resto de los dispositivos en la red.

Los switches *OpenFlow*, se conectan de manera lógica mediante sus puertos *OpenFlow*, en estos, pueden estar asignados varios puertos para el procesamiento de paquetes *OpenFlow*.

En cada switch *OpenFlow* hay dos clases de puertos, los de entrada (*ingress port*) y los de salida (*output port*). En el primero, los paquetes entran por el switch, y son llevados por una serie de pasos donde son modificados y posteriormente enviados hacia el siguiente destino por el puerto de salida. [8]

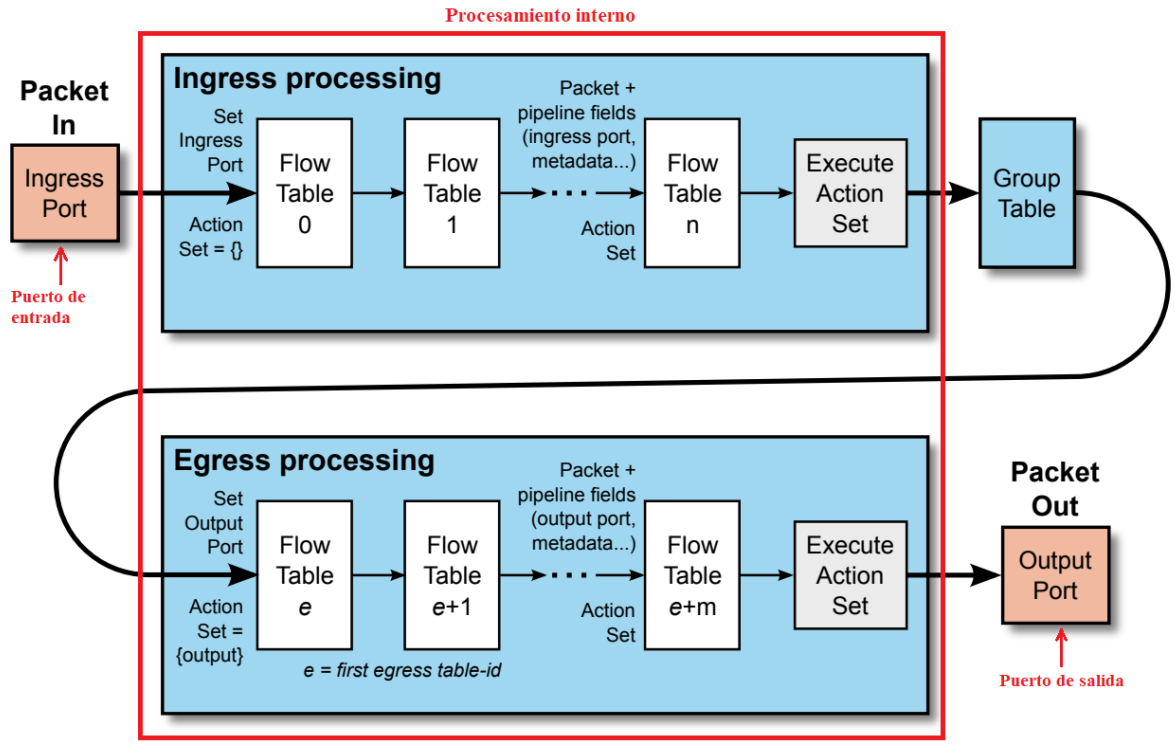


Fig. 6.2.1. Puertos de entrada y salida en un switch OpenFlow. Fuente: [19]

Los puertos *OpenFlow* pueden ser añadidos, modificados o quitados. Las *flow tables* de un switch nunca pueden ser modificados por estas acciones. Los paquetes enrutados a puertos no existentes son simplemente descartados, y los puertos borrados son manejados por el controlador, mediante una limpieza de sus referencias.

Cada una de las dos clases de puertos puede ser de tres tipos, los puertos físicos, los lógicos y los reservados. Entre estos, están también los puertos estándar, los cuales pueden ser tanto físicos, lógicos o reservados localmente (en algunos casos).

6.2.1. Puertos Físicos

Los puertos físicos *OpenFlow* son aquellos puertos que están reservados en el switch, correspondiendo a una entrada de interfaz del hardware del switch. Es decir, una entrada en el switch, está reservada para su uso por *OpenFlow*, un ejemplo de este es una entrada *Ethernet*.

6.2.2. Puertos Lógicos

Los puertos lógicos *OpenFlow* son aquellos, que no corresponden directamente a una entrada física en la interfaz del hardware del switch, sino que pueden llegar a ser mapeados a diferentes puertos, mediante procesos no ligados a *OpenFlow*, por ejemplo, un puerto VLAN.

A la hora de reconocer que la entrada se ha producido por un puerto lógico o no, por parte del controlador, la información sobre el puerto lógico y el puerto físico al que pertenece, tiene que ser enviado a este mismo controlador.

6.2.3. Puertos Reservados

Los puertos reservados en *OpenFlow* son aquellos que definen acciones específicas de enrutado dentro del switch, y normalmente, estas acciones están relacionadas con las tablas de flujos. Entre estas acciones, están las requeridas y las opcionales. Algunas de estas solo están disponibles en un switch *OpenFlow* híbrido.

Entre las acciones requeridas están:

- **ALL**: Todos los puertos que un switch puede usar para enviar fuera un paquete.
- **CONTROLLER**: Canal de comunicación con el controlador OpenFlow.
- **TABLE**: Envía el paquete al principio de la línea de procesado OpenFlow.
- **IN_PORT**: Envía un paquete fuera, mediante el puerto de entrada.
- **ANY**: Cuando ningún puerto está especificado.

- **UNSET**: El puerto de salida no ha sido especificado.

Entre las acciones opcionales están:

- **LOCAL**: Representa la red local del switch con el que puede interactuar.
- **NORMAL**: Modo tradicional de enrutado sin procesamiento *OpenFlow*.
- **FLOOD**: Modo tradicional de enviar el paquete por todas los puertos de salida.

6.3. Procesado de flujos en tablas OpenFlow

Los switch con soporte a *OpenFlow*, pueden ser de procesamiento único con *OpenFlow* o híbridos, es decir, aquellos que soportan el enrutado convencional, además de tener un procesamiento de instrucciones únicas de *OpenFlow*. Por ejemplo, soporta enrutado L2, L3, VLAN, ACLs, entre otros. Aquí, dependiendo del flag del paquete, este va a ser procesado por una línea de procesado u otra.

Cada switch *OpenFlow* tiene una o más tablas de flujos, cuantas menos, más simple el procesado. Cada una de estas tablas de flujo contiene múltiples entradas de flujo.

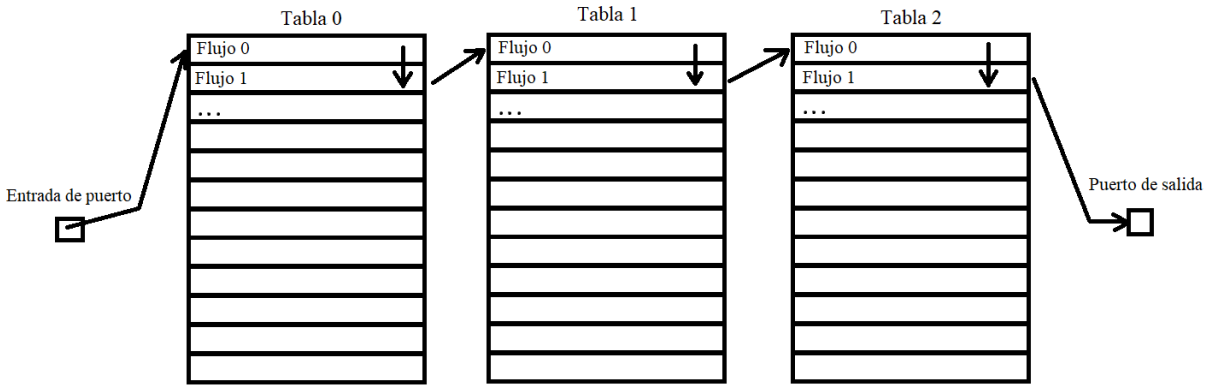


Fig. 6.3.1. Tablas y flujos en switch *OpenFlow*. Fuente: Propia.

La única tabla obligatoria, es la de entrada (*ingress flow table*), la cual comienza por el número de ordenación *0*, y así, las tablas son ordenadas numéricamente dependiendo del orden en el que procesan los paquetes.

La separación entre las tablas de entrada y salida (*egress flow table*) es indicado por la primera tabla de salida, así, ninguna tabla de entrada puede tener un número igual o mayor a esta.

El procesado comienza por la primera tabla, la *0*, aquí va a ser emparejado con una entrada de flujo (*flow entry*). Opcionalmente, puede pasar por otras tablas de entrada, las cuales dependen del resultado de la primera tabla, o directamente enviado a un puerto de salida, aquí también, siendo opcional el procesado de salida.

Cuando el paquete coincide con una *flow entry* en la tabla de flujos, el conjunto de instrucciones de ese flujo es ejecutado, de la misma manera, entre estas instrucciones puede estar la de enviar el paquete a la siguiente tabla, pero nunca hacia atrás.

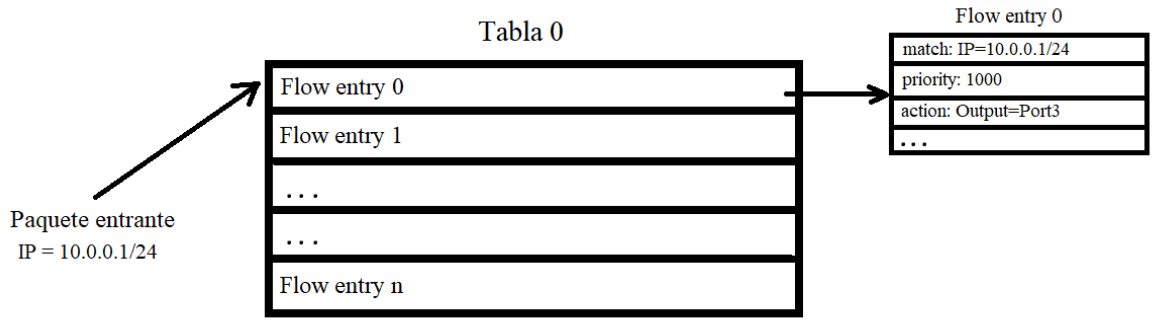


Fig. 6.3.2. Procesado de *flow entries*. Fuente: Propia.

En el caso de que no se encuentre ninguna entrada para este paquete, puede acabar descartando el paquete, en el caso de que la tabla no tenga un flujo de fallo de emparejado, flujo que indica qué hacer con los paquetes que no coinciden con ninguna *flow entry*, en otros casos, el paquete se envía al controlador.

La línea de procesamiento *OpenFlow* en un switch es mapeada directamente sobre el hardware del switch, lo cual requiere de ciertas consistencias para poder tener un correcto funcionamiento. La primera consistencia es la de las tablas, todos los paquetes deben coincidir de la misma manera con todas las tablas de flujo, lo único que varía es el contenido. La consistencia de una *flow entry* requiere que el flujo pueda modificar la cabecera del paquete correcto, y no otra cabecera diferente. Para la consistencia de grupo, todos los paquetes del grupo deben de ser tratados tal y como si estuvieran en una tabla de flujos. En el caso de la

consistencias de puertos, un paquete que sale de un switch por un puerto, debe ser el mismo, si vuelve a entrar por el mismo puerto y siendo el mismo paquete.

Una tabla de flujos se compone de *flow entries*, estas se identifican por sus campos de emparejado (*match fields*), las cuales se usan para identificar los paquetes, junto a la prioridad de este. Otros componentes de una entrada de flujo son los contadores, que se actualizan cuando se empareja un paquete, las instrucciones, las cuales describen acciones, los *timeouts*, que controlan cuando un flujo debe ser retirado, las *cookies*, que se usan para filtrar flujos más que para procesar paquetes, y las etiquetas (*flags*), las cuales se usan para alterar el comportamiento de los flujos.

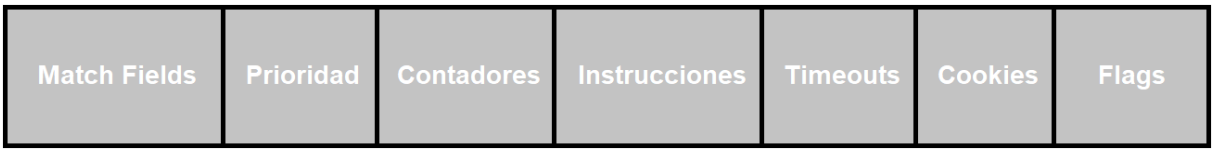


Fig. 6.3.3. *Flow Entry* en una tabla OpenFlow. Fuente: Propia.

6.3.1. Procesado de paquetes

El procesado de un paquete entrante en un switch *OpenFlow* comienza con un emparejado en la primera tabla de flujos.

Las cabeceras son extraídas del paquete, y se utilizan para hacer el emparejado con un flujo específico de la primera tabla de flujos. Estas cabeceras pueden incluir información de varios protocolos, L2 o L3. Además de estas cabeceras, la información puede ser comprobada contra los puertos de entrada, los campos de metadatos, que pueden ser usados para pasar información entre tablas, y otros campos de la línea de procesado.

Las cabeceras de un paquete y los campos de la línea de procesado representan el estado actual del paquete, si las acciones aplicadas en una tabla previa cambian la información de estas, estos cambios se verán reflejados.

Un paquete se empareja con una entrada de flujo si todos los *match fields* del flujo se emparejan con las correspondientes cabeceras y campos de la línea de procesado del paquete.

Solamente la *flow entry* con prioridad más alta es seleccionada, los contadores que están asociados al flujo son actualizados y el conjunto de instrucciones del flujo es ejecutado. En el caso de que haya múltiples flujos con la misma prioridad más alta, el flujo seleccionado es explícitamente indefinido.

Esto solamente ocurre cuando el controlador añade entradas solapadas sin establecer el bit **OFPFF_CHECK_OVERLAP**.

Todas las tablas de flujo deben soportar un flujo de fallo de emparejamiento, el cual define como procesar paquetes no emparejados por otros flujos en la tabla. Estas acciones principalmente suelen ser descartar el paquete o enviarlo al controlador.

El flujo de fallo de emparejamiento es definido por el *match field* y la prioridad, este omite todos los *match fields* y tiene la prioridad más baja (0).

Las tablas de flujo pueden ser usadas para procesado de entrada y salida. Cuando empieza el procesado de entrada, el conjunto de acciones está vacío. Las tablas de flujo usadas para procesado de entrada solo pueden mover paquetes mediante la instrucción *Goto-Table* a otras tablas de entrada, pero no de salida.

En el caso del procesado de salida, todas las acciones de salida, se realizan solo por el puerto de salida, cualquier tipo de puerto es válido como salida, tanto físico como lógico. Las tablas de flujo usadas para el procesado de salida sólo pueden usar la instrucción *Goto-Ta*ble para enviar paquetes a otras tablas de salida, y a diferencia de las tablas de entrada, deben soportar el campo *Action Set Output* para poder enviar paquetes por el puerto de salida.

En cuanto a las diferencias entre ambas, las entradas de flujo tienen los mismos componentes, el emparejamiento es el mismo y la ejecución de instrucciones también. Los flujos de fallo de emparejamiento también son los mismo. Una tabla de flujos puede ser reconfigurada para el procesado de entrada o salida cambiando la primera tabla de salida.

6.3.2. Instrucciones y acciones

Una instrucción realiza cambios en el paquete después de que se haya hecho un correcto emparejamiento con el flujo correspondiente.

El conjunto de instrucciones en un flujo, solo permite una instrucción de cada tipo, y en el caso de que una instrucción no pueda ser ejecutada por el switch, esta debe ser descartada. Existen dos tipos de instrucciones, las obligatorias y las opcionales.

Entre las instrucciones obligatorias están:

- *Clear-Actions:* Borra todo el conjunto de acciones en un flujo de fallo de emparejamiento.
- *Write-Actions:* Añade acciones al conjunto actual de acciones.
- *Goto-Table:* Indica la siguiente tabla de la línea de procesado.

Entre las instrucciones opcionales están:

- *Apply-Actions:* Aplica acciones a un flujo.
- *Clear-Actions:* Borra todo el conjunto de acciones en otros tipos de flujos.
- *Write-Metadata:* Escribe metadatos en el campo de metadatos.
- *Stat-Trigger:* Crea un evento para el controlador a partir de un valor especificado.

Un conjunto de acciones está asociado a cada paquete, el cual está vacío por defecto y se envía entre tablas de flujos. Cada conjunto de acciones se compone de un máximo de una acción de un mismo tipo, si una acción es añadida a un conjunto en el que ya existe este tipo de acción, esta última la sobrescribe. El conjunto de acciones para procesado de entrada siempre comienza vacío, mientras que el de salida dispone desde el inicio de una acción de salida por un puerto.

Existe un orden específico para las acciones dentro de un conjunto de acciones, este es:

1. *copy TTL inwards:* Aplica acciones de etiqueta TTL interna al paquete.

2. *pop:* Aplica todas las etiquetas de acciones al paquete.

3. *push-MPLS:* Aplica acciones de etiqueta *MPLS* al paquete.

4. *push-PBB:* Aplica acciones de etiqueta *PBB* al paquete.

5. *push-VLAN:* Aplica acciones de etiqueta *VLAN* al paquete.

6. *copy TTL outwards:* Aplica acciones de etiqueta TTL externa al paquete.

7. *decrement TTL:* Aplica acciones de etiqueta de disminución del TTL al paquete.

8. *set*: Aplica acciones de etiqueta *set-field* al paquete .

9. *qos*: Aplica acciones de *QoS* al paquete

10. *group*: Aplica acciones de grupo, si este está especificado.

11. *output*: Envía el paquete por el puerto especificado, si no hay una acción de grupo.

Se llama lista de acciones a acciones que siempre se deben de ejecutar en un mismo orden, las especificaciones de *OpenFlow* definen este orden y en caso de que todas las acciones de la lista no puedan ser ejecutadas, no se ejecutará ninguna acción de esta lista.[9]

Cada acción de una lista de acciones o un conjunto de acciones especifica que hay que hacer con el paquete, en el momento que se produce un emparejamiento con un flujo.

6.3.3. Contadores

Los contadores se encargan de mantener las estadísticas dentro de un switch *OpenFlow*, para ello, guardan información sobre el número de paquetes y *bytes* que circulan a través de distintos elementos de la línea de procesado.

Todos los elementos necesitan mantener un contador, incluso si los paquetes no van a ser usados, ya sean flujos sin acciones, puertos caídos o inexistentes.

Existen variedad de contadores, la mayoría de ellos opciones, pero entre los requeridos están:

- *Reference Count* (entradas activas) - 32 bits [en tablas de flujo]
- *Duration* (segundos) - 32 bits [en entradas de flujo, en colas, en grupos y en métricas]
- *Received Packets* - 64 bits [en puertos]
- *Transmitted Packets* - 64 bits [en puertos y en colas]

Entre los contadores opcionales más importantes se encuentran por ejemplo el número de paquetes emparejados, *bytes* enviados y recibidos, errores, número de flujos, entre otros.

6.4. Canales de comunicación

El canal de comunicación de *OpenFlow* conecta cada switch *OpenFlow* a uno o varios controladores, de esta manera, pudiendo realizar configuraciones del switch y responder a eventos.

6.4.1. Tipos de comunicación

Existen tres tipos diferentes de mensajes entre un switch *OpenFlow* y un controlador.

Los mensajes controlador a switch se inician por el controlador, usados principalmente para el manejo y revisión del estado del switch. Entre los principales mensajes que puede solicitar un controlador están:

- *Features:* El controlador pide al switch sus características.
- *Configuration:* El controlador modifica la configuración del switch.
- *Modify-State*: Mensajes que modifican el estado del switch.
- *Read-State:* El controlador solicita la configuración actual del switch.
- *Packet-out:* El controlador especifica por qué puerto enviar paquetes.
- *Barrier:* El controlador se asegura de operaciones completadas.

Los mensajes asíncronos son enviados al controlador por el switch para notificar eventos específicos. Entre los principales mensajes asíncronos están:

- *Packet-in:* Se notifica al controlador sobre el paquete, ya sea por fallo de emparejamiento o una instrucción que le transfiera el control al controlador.
- *Flow-Removed:* Notifica al controlador de que un flujo ha sido removido. En la mayoría de los casos, es removido por el TTL especificado por el controlador.
- *Port-status:* El switch envía el estado de un puerto cada vez que este cambia.

Los mensajes simétricos son enviados tanto por el switch como por el controlador sin haber sido solicitados, los principales entre estos siendo:

- *Hello:* Mensaje de prueba de conexión entre switch y controlador.
- *Echo:* Notifica que la conexión sigue viva.
- *Error:* Notificación de errores por parte del switch o del controlador.

6.4.2. Conexiones

El canal de comunicación *OpenFlow* es abierto para el intercambio de mensajes, cada controlador puede tener abiertas múltiples conexiones a múltiples switch, y un switch pudiendo tener establecidas múltiples conexiones con varios controladores.

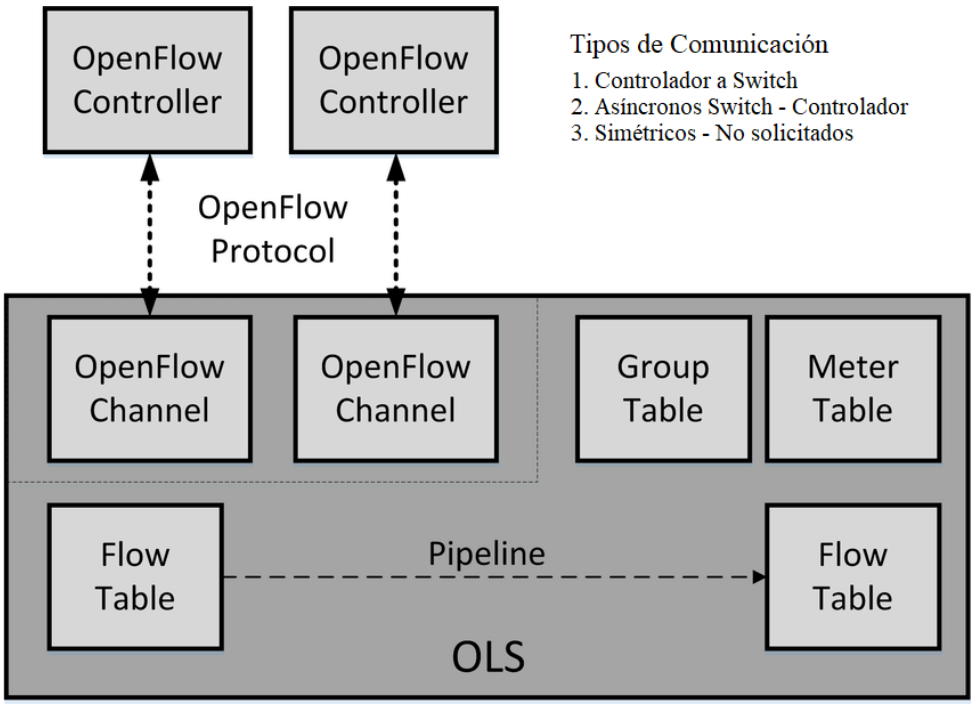


Fig. 6.4.2.1.Canal de comunicación *OpenFlow*. Fuente: [20]

Los protocolos usados para la correcta comunicación entre controlador y switch son *TLS* o *TCP*, evitando así realizar conexiones no seguras.

El estándar usado para crear la conexión con un controlador es *procolo:dirección:puerto,* este identificador siempre representa la vista que tiene el switch del controlador.

Los posibles valores para el protocolo son *tls* o *tcp*, opcionalmente *dtls*, y *udp* como valor no recomendado. La dirección es el nombre del host, en el caso de disponer de un *DNS*, o la dirección IP.

Para el puerto, el valor por defecto en *OpenFlow* es el 6633 o 6653, dependiendo de la versión de *OpenFlow*, aunque puede ser cambiado.

Para que un protocolo de comunicación se considere válido para el uso con *OpenFlow,* debe cumplir con varios requisitos:

- *Multiplexing:* Debe ser capaz de crear múltiples conexiones desde el mismo origen o hacia el mismo destino.

- *Fiabilidad:* Debe prevenir la pérdida de paquetes.

- *Ordenación:* Debe enviar los paquetes en el orden correcto.

- *Fragmentación:* Debe poder fragmentar y recomponer los mensajes OpenFlow.

- *Control de flujos:* Debe de poder controlar el flujo de los mensajes.

- *Seguridad:* Debe de implementar toda la seguridad, ya que el protocolo OpenFlow no lo hace.

Una vez establecida la conexión entre switch y controlador mediante *TLS* o *TCP*, la dirección IP y el puerto *OpenFlow* usado por el controlador, se envían mutuamente el mensaje *OFPT_HELLO,* en conjunto con la versión más alta de *OpenFlow* que soporta cada uno, todo esto, y algunos elementos opcionales más para facilitar la correcta comunicación. Una vez acabada esta fase, el controlador pide al switch sus características mediante el mensaje *OFPT_FEATURES_REQUEST*.

Un switch puede realizar conexiones con múltiples controladores a la vez, de esta manera mejorando la fiabilidad en el caso de que uno de ellos falle, también ofreciendo un mejor balanceo de cargas. Los controladores conectados a un switch pueden tener diferentes roles, entre ellos están *MASTER, EQUAL y SLAVE*.

Estos roles son intercambiados entre los controladores mismos, y cualquiera de estos puede asignarse el rol de *MASTER*, rol que otorga al controlador la prioridad de controlador primario. El rol de *SLAVE* no puede modificar el estado de un switch, solo recibir eventos de su estado.

6.4.3. Procesamiento y modificación de mensajes

Los mensajes *OpenFlow* deben estar garantizados, es decir, en el caso de que la conexión falle y ciertos mensajes no lleguen al controlador, este se abstendrá de realizar cambios o interpretar el estado del switch. Este a su vez, va a enviar mensajes de error al controlador, en el caso de no poder procesar correctamente algunos de los mensajes recibidos por él.

Cuando un controlador añade mensajes a un grupo, o todos se aplican o ninguno, al cual también se puede añadir ordenaciones.

En el caso de aplicar ordenaciones, se debe de tener en cuenta que, a no ser que aplique un mensaje de barrera, el controlador no asumirá ninguna ordenación, así, primero se procesarán los mensajes anteriores a la barrera, y después de procesar la barrera, lo harán los mensajes no ordenados.

Los mensajes de modificación de las tablas de flujo son usados para crear, modificar o borrar flujos específicos en tablas específicas. Cuando un controlador quiere añadir un flujo a una tabla de flujos, primero tiene que mirar si la etiqueta de solapamiento está activada, y en el caso de encontrar otra entrada de flujo igual en la tabla el switch, devuelve un mensaje de error. En el caso de que la etiqueta de solapamiento no esté activada, la nueva entrada de flujo sobrescribe a la antigua. Si el mensaje del controlador es de modificación, y concuerda con un flujo en una tabla, el campo de instrucciones cambia a los nuevos valores, así como los contadores, pero el resto de los campos no varían. Por último, si el mensaje es de eliminar un flujo específico, este debe de ser borrado si existe en la tabla especificada, en caso contrario no varía nada.

Otros métodos de eliminar flujos existen en cada switch *OpenFlow*, aparte del mensaje especificando dicha acción por parte del controlador. Todos los flujos disponen de campos de *timeout, idle_timeout* y *hard_timeout*. Cuando el flujo no recibe ningún paquete durante el tiempo especificado en el *idle_timeout*, esta va a ser removido de la tabla. Cuando el campo *hard_timeout* llega a cero, independientemente de los procesos en marcha, el flujo es removido.

7. OpenDayLight

OpenDayLight es un proyecto open source cuyo objetivo principal es la aceleración de la implementación de soluciones *SDN*. Para ello cuenta con múltiples proyectos y múltiples compañías soporte, los cuales desarrollan y mantienen aplicaciones de virtualización de redes, uno de los principales siendo el controlador que permite implementar el protocolo *OpenFlow*.

El proyecto ha comenzado en 2013, con la *Linux Foundation* como principal fundador. Muchas de las grandes empresas del sector de las redes se han sumado a lo largo de los años, con nombres como *Cisco*, *Brocade*, *IBM*, entre otros, uniéndose al principio de todo. A día de hoy, existen catorce versiones, la primera de estas, siendo del año 2014.[10]

En este capítulo se explican las partes principales de las que se compone la plataforma y su funcionamiento.

7.1. Controlador

El controlador *OpenDayLight* es una aplicación multiplataforma ejecutada sobre la *JVM* y desarrollada en *Java*, que ofrece gran modularidad y modificación por parte del usuario, siendo esta una herramienta open source. Todo esto es posible incorporando tecnologías ya disponibles y permitiendo la fácil incorporación de otras muchas.

Algunas de estas tecnologías que permiten a *OpenDayLight* ser tan flexible son *OSGI* y *YANG*. *OSGI* es un *framework* de *Java* que permite crear y gestionar *bundles*, los cuales son aislados en *containers*. Esto permite que la creación de nuevos bundles sea rápido y sencillo. Un ejemplo que está incluido en el propio *OpenDayLight* es *Karaf*, el cual es en sí mismo un *bundle* que permite instalar diversos paquetes dentro de un controlador.

Por otro lado, *YANG* es un lenguaje de modelado de datos para manipular configuraciones y estados en aplicaciones [11], usado principalmente para el correcto funcionamiento de unos de los componentes principales de *OpenDayLight*, *MD-SAL*, el cual actúa como un middleware entre las aplicaciones de usuario y los protocolos usados en los dispositivos de red.

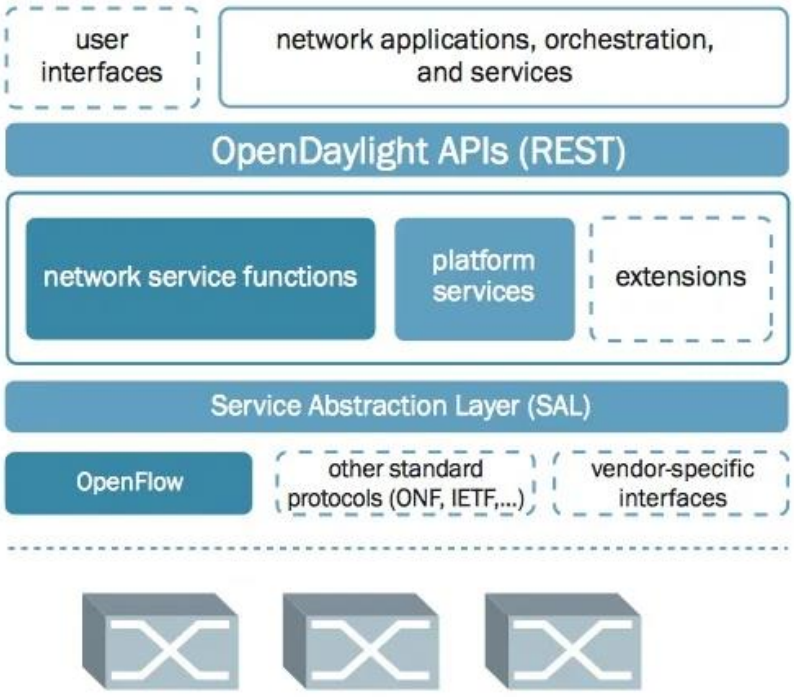


Fig. 7.1.1. Arquitectura de OpenDayLight. Fuente: [15]

7.2. Comunicación

El controlador de *OpenDayLight* actúa como un intermediario, comunicando las aplicaciones de usuario con los dispositivos de red. Esto es posible gracias a la *interfaz northbound* y *southbound*, que comunican el controlador con las aplicaciones de usuario (*northbound*) y con los protocolos de manejo de dispositivos de red (*southbound*).

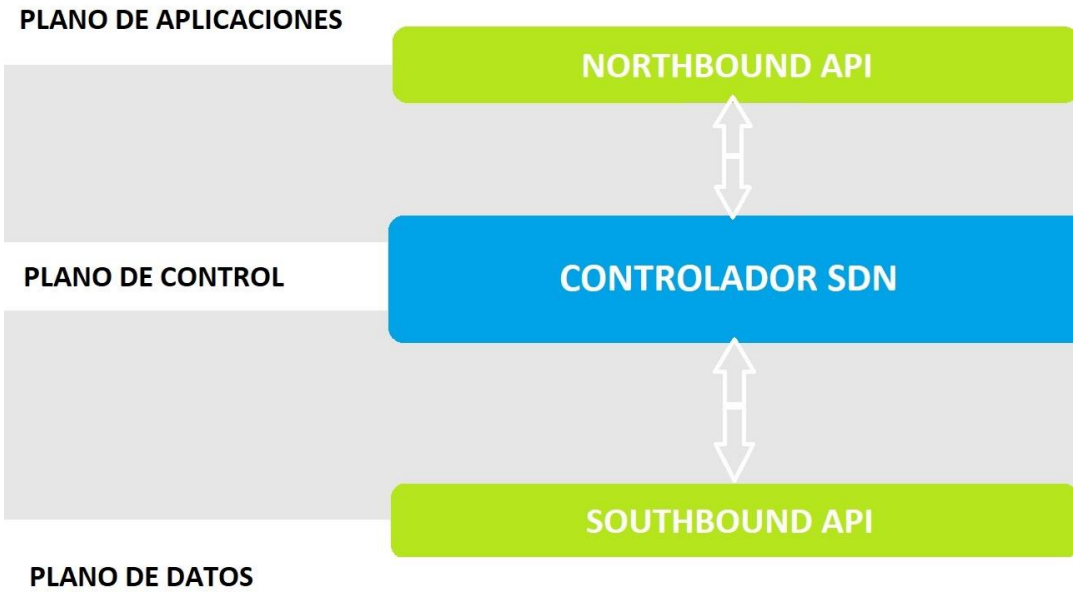


Fig. 7.2.1. Diferentes niveles de abstracción en *ODL*. Fuente: Propio.

7.2.1. NETCONF

NETCONF es un protocolo basado en *XML* que permite manipular la configuración en dispositivos de red [12]. Este protocolo crea una API, a través de la cual un controlador puede configurar dispositivos de red, aunque solamente aquellos dispositivos que soportan este protocolo.

Utiliza un método de comunicación llamado *Remote Procedure Call* (RPC). Tanto las peticiones como las respuestas, están formateadas en *XML*. Esto permite que una gran variedad de dispositivos puedan interpretar estos mensajes. Mediante las comunicaciones entre el cliente y los dispositivos, este puede concretar las configuraciones de estos dispositivos, y a su vez,

configurarlos. Entre el contenido que puede presentar estos mensajes, se pueden encontrar el estado de topologías, los servicios, las políticas, los consumidores, entre otros.

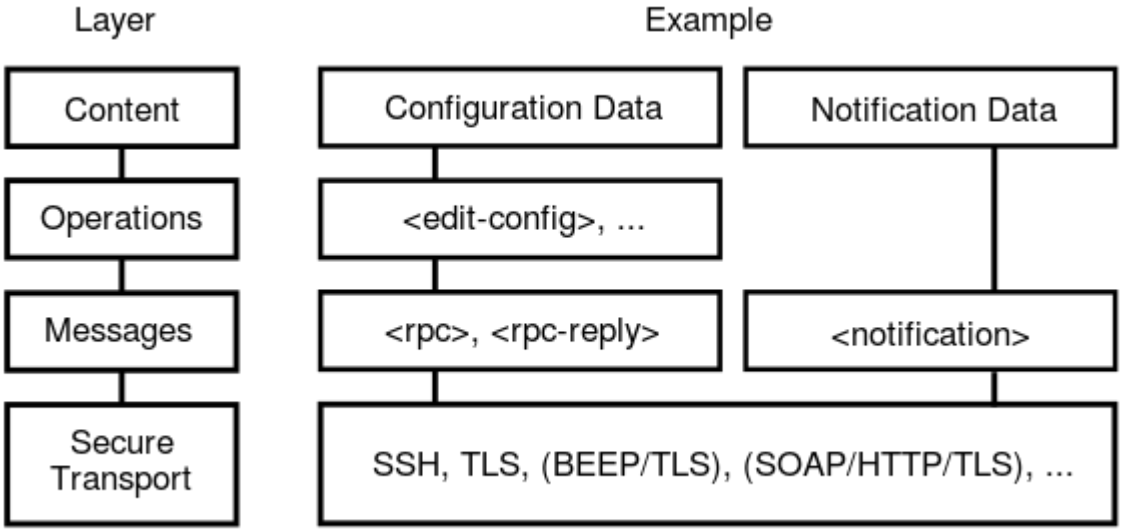


Fig. 7.2.1.1. Niveles de componentes en NETCONF. Fuente: [13].

Para el protocolo de seguridad usado en conjunto con *NETCONF,* se recomienda el uso mínimo de *SSH* o *TLS,* dependiendo del uso.

7.2.2. RESTCONF

RESTCONF es un protocolo que está basado en *http* que proporciona acceso al estado y configuración de dispositivos de red a aplicaciones de usuario. Permite interactuar con modelos *YANG,* pero a diferencia de *NETCONF*, interactúa con este mediante peticiones *http* basadas en APIs *REST*. Esto significa que las peticiones no tienen que ser codificadas en *XML*, sino que también en *JSON*. Para especificar la localización del modelo de datos, este requiere de una dirección URL, de la misma manera que cualquier otra petición *http*. [14]

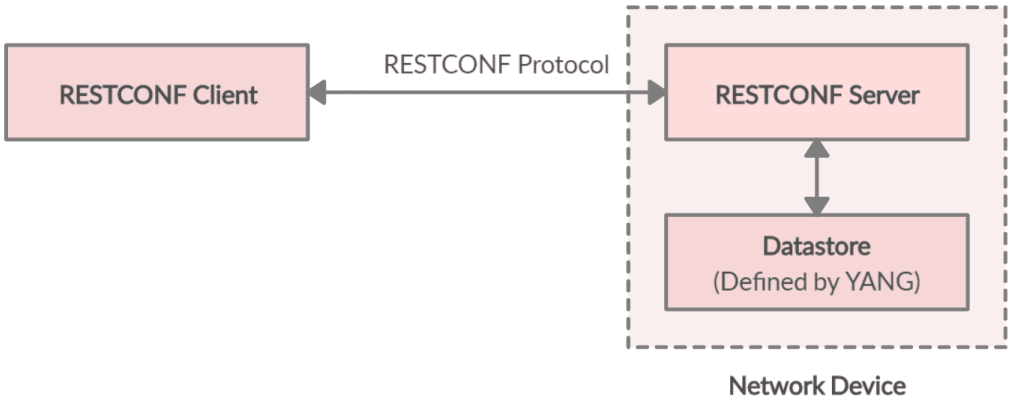


Fig. 7.2.2.1. Esquema de una conexión *RESTCONF*. Fuente: [15].

Cada una de estas peticiones hechas por el cliente es enviada al servidor de *RESTCONF*, el cual, en este caso, está situado dentro del controlador *ODL*. Una vez la petición llega al servidor, establece conexión directa con el *datastore* de *MD-SAL*, una vez aquí, la petición extrae o modifica los datos dentro de esta.

7.3. MD-SAL

Model-Driven Servicie Adaptation Layer (*MD-SAL*) es un componente del controlador de *OpenDayLight.* Proporciona una infraestructura para mensajería y almacenamiento de datos, basado en los modelos generados por las aplicaciones.

Los modelos que representan el estado de las aplicaciones, se crean en forma de árbol. Existen dos tipos de modelos de almacenamiento de estados, *Operational* y *Configuration*.

El primero representa el estado actual del sistema, esta información es recogida desde los dispositivos mediante las *southbound* APIs, este posteriormente es representado a usuarios mediante las aplicaciones en las *northbound* APIs. El segundo de ellos, *Configuration*, representa el estado deseado, es decir, aquel estado del sistema, que los usuarios de las *northbound* APIs crean o modifican, que posteriormente va a ser representado en los dispositivos de red mediante las *southbound* APIs.[16]

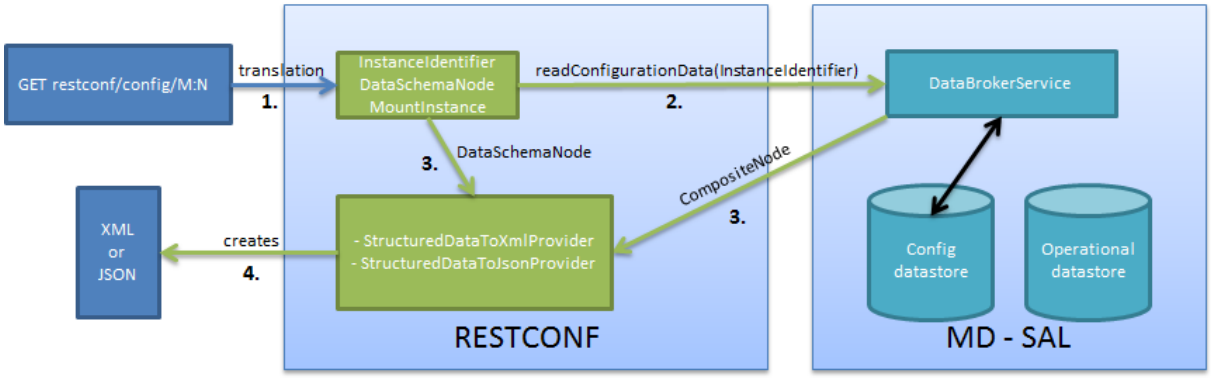


Fig. 7.3.1. Petición *REST* a *MD-SAL*. Fuente: [17].

Los modelos de datos usados en *MD-SAL* están creados mediante *YANG*. Este es un lenguaje de modelado de datos usados para configuraciones y mantenimiento de estados en las redes.

MD-SAL también ofrece varios modos diferentes de comunicación. Entre ellos están los *Remote Procedure Calls* (*RPC*), que son aquellas peticiones en las que un consumidor o una aplicación envía un mensaje a un proveedor, es decir, aquella entidad que contiene información, responde a este mensaje de manera asíncrona. Otro modo de comunicación son las suscripciones, en este caso, existe una entidad a la que suscriben otras múltiples. Cada vez que hay un cambio de estado, la entidad que publica envía un mensaje a todos los suscriptores informando sobre los eventos. Por último, también existe la comunicación transaccional, en la que cada transacción está separada de otra, las de lectura solamente pueden leer, y las de modificación las únicas que pueden el estado.

8. Configuración SDN

Para poder comenzar con las implementaciones de topologías SDN, se necesita configurar primero ciertas aplicaciones. Estas pueden ayudar a explorar las posibilidades que las futuras redes, y algunas ya hoy en día, van a implementar a gran escala de una forma muy personalizada, virtualizando todo el proceso desde un ordenador.

En este apartado se configura todo lo necesario para poder experimentar con las tecnologías explicadas en los apartados anteriores. Para la demostración se utiliza un equipo con *Windows 10* instalado, un procesador de 4 núcleos físicos (8 núcleos virtuales) y 16GB de RAM.

8.1. Emulación de redes con GNS3

GNS3 es un programa de emulación de redes que permite crear topología combinando tanto dispositivos virtuales como reales.

GNS3 se compone de dos partes principales, una GUI en la que se puede diseñar y configurar las topologías, y un servidor que ejecuta los dispositivos de las topologías. Las opciones disponibles como servidores para las topologías en GNS3 incluyen, un servidor GNS3 local, o la máquina virtual GNS3, local o remota.

A continuación, se procede con la configuración de GNS3. En este caso, se usa la máquina virtual de GNS3 de manera local como servidor para las topologías.

Desde la página oficial de GNS3 se descarga el ejecutable para el sistema operativo correspondiente, en este caso va a ser *Windows* (https://www.gns3.com/software/download).

En cuanto a los requisitos recomendados para un buen funcionamiento de las topologías, estos son los siguientes:

- **Sistema operativo**: *Windows 7* o superior, de 64 bits.
- **Procesador**: Al menos 4 núcleos lógicos con virtualización.
- **Memoria**: 8 GB de RAM, cuantos más dispositivos más se necesita.

Lo primero, al darle a descargar la última versión de *GNS3*, hay que registrarse y acceder a la cuenta personal, posteriormente, va a llevar a la última página de descarga. El proceso de instalación es intuitivo hasta el final, solo hay que fijarse en los detalles que se preguntan, no hay ninguna configuración específica en este paso.

Ya que no se trabaja ni en Linux ni en Mac, la terminal que se dentro de GNS3 para *Windows* es *Solar-Putty*, la cual va a salir como opción de descarga durante la instalación de GNS3.

Una vez se tiene instalado el *GNS3*, se abre una interfaz como la de la Fig. 8.1.1.

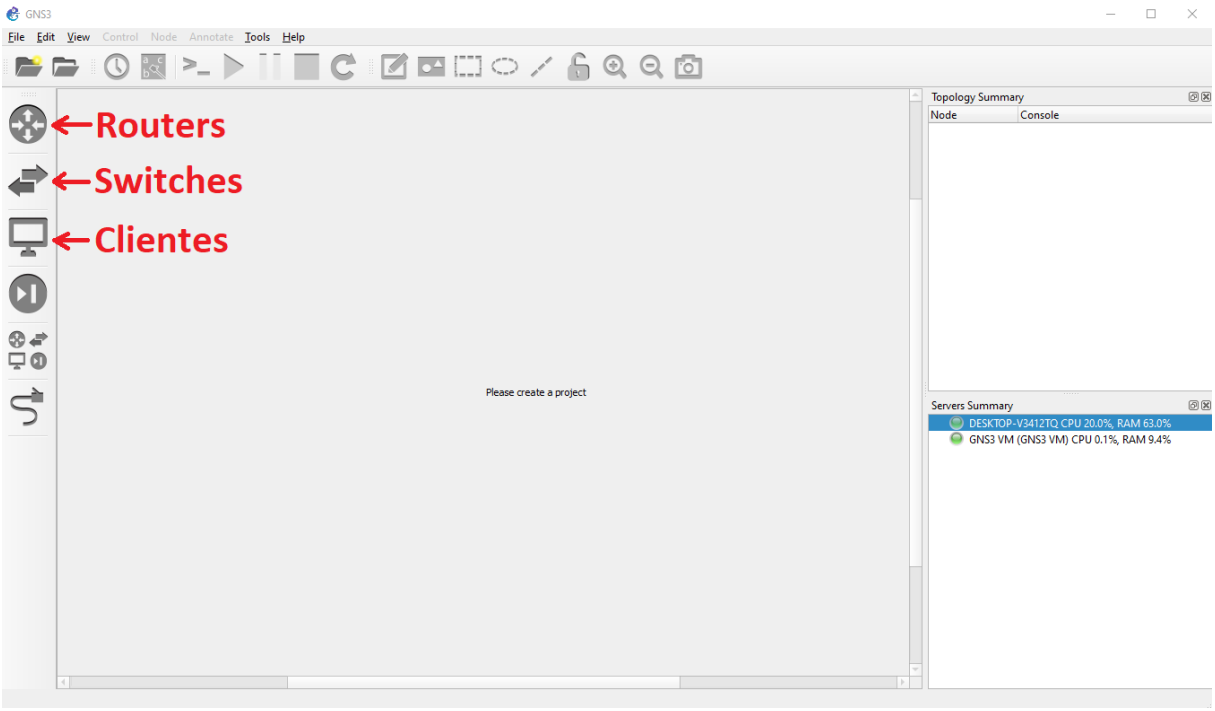


Fig. 8.1.1. Interfaz *GNS3*. Fuente: Propia.

Al arrancar por primera vez, sale una pestaña preguntando el tipo de servidor que se va a utilizar. Ya que no se ha configurado nada aún, solo se puede utilizar la opción de ejecutarlo sobre el propio ordenador.

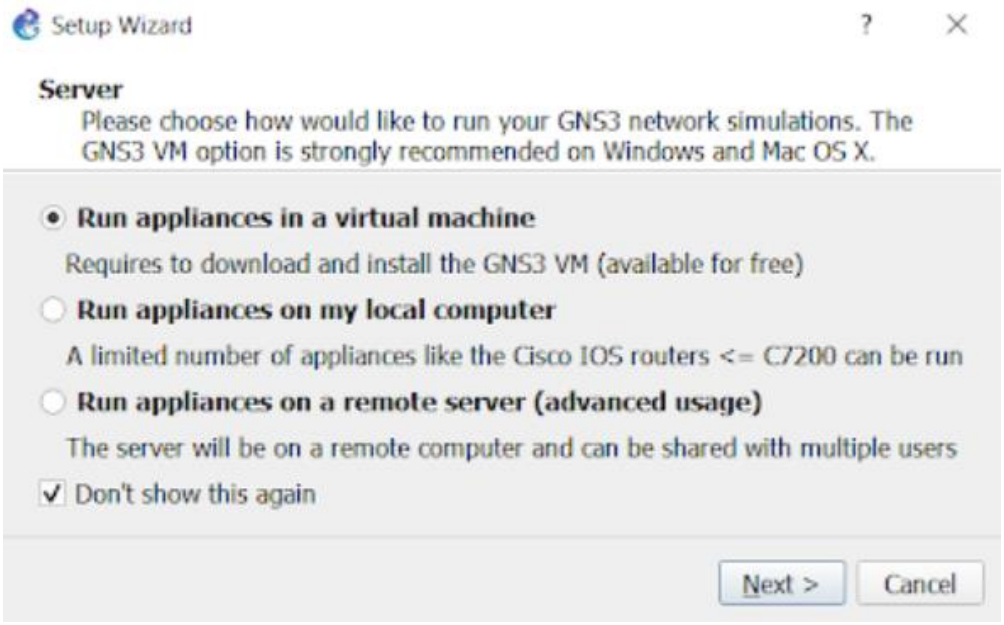


Fig. 8.1.2. Menú de selección de servidor en *GNS3*. Fuente: Propia.

Si se elige ejecutar las aplicaciones sobre el propio ordenador de manera local, la siguiente pestaña va a preguntar información sobre la configuración de este servidor. Se pueden cambiar, pero las opciones por defecto ya son correctas.

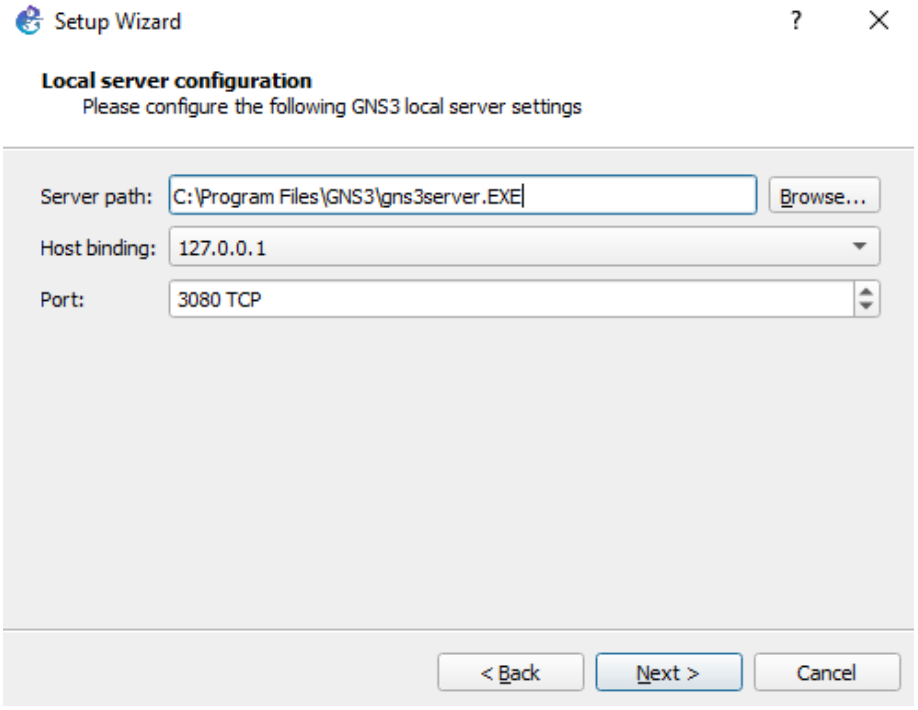


Fig. 8.1.3. Configuración servidor en *GNS3*. Fuente: Propia.

En este momento ya se puede comenzar a configurar topologías, aunque esto solo se va a hacer sobre el propio ordenador, por lo tanto, se tiene que instalar la máquina virtual de GNS3 para poder montar las topologías.

8.2. Virtualización con VMware

Se necesita el programa de virtualización *VMware Workstation Pro*, ya que este soporta *nested virtualization*, es decir, las aplicaciones que son virtualizadas dentro de la máquina virtual de *GNS3* que se va a utilizar, van a ser aceleradas por la CPU. Por esta razón es recomendable usar *VMware* sobre *VirtualBox*.

VMware Workstation Pro requiere de una licencia para ser usado, pero se puede pedir una prueba gratuita de 30 días. En el caso de requerir un uso más prolongado, se tiene que pagar la licencia. Otros programas gratuitos como VirtualBox o VMware (versión gratuita) pueden servir, eso sí, siempre con una desventaja en rendimiento.

Se procede a descargar *VMware Workstation Pro* desde la página oficial (https://www.vmware.com/go/getworkstation-win). De la misma manera que con la instalación de *GNS3*, el proceso de instalación del *VMware* no requiere ninguna configuración específica. Al acabar la instalación va a salir una ventana para introducir la licencia del producto, se le da a la opción de prueba gratuita de 30 días, en el caso de que no se disponga de una clave.

Fig. 8.2.1. Menú licencias *VMware*. Fuente: Propia.

8.3. Máquina virtual GNS3

Anteriormente, se ha mencionado que se tiene que instalar una máquina virtual para poder ejecutar dispositivos en ella, la primera de estas aplicaciones es *GNS3 VM*.

La razón principal para el uso de la máquina virtual de *GNS3* es que permite ejecutar sistemas operativos que alternativamente no se pueden ejecutar, como en el caso de *Windows* o *Mac*, ya que carecen del kernel de *Linux*.

Primero de todo se descarga *GNS3 VM* para *VMware Workstation* desde la página oficial (https://www.gns3.com/software/download-vm).

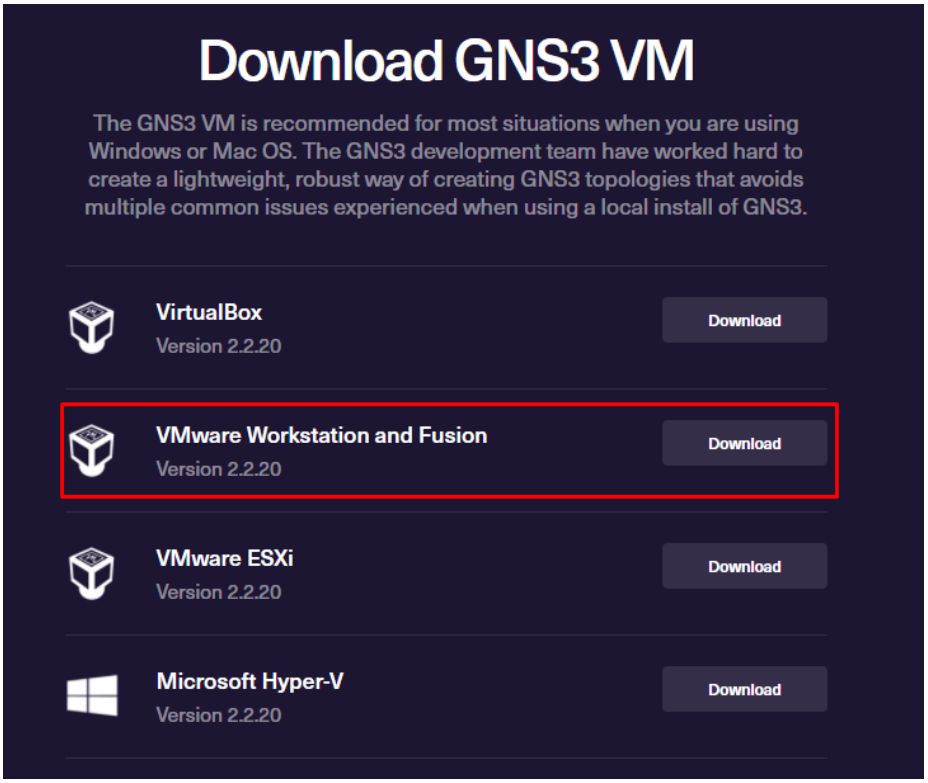


Fig. 8.3.1. Descarga *GN3VM* para *VMware*. Fuente: Propia.

Una vez el archivo *GNS3 VM.ova* está descargado, se puede importar en *VMware*. En la pantalla de inicio se le da a abrir máquina virtual y se selecciona el archivo descargado.

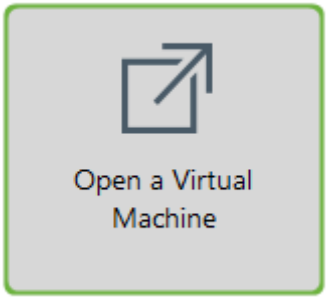


Fig. 8.3.2. Botón para abrir máquina virtual en *VMware*. Fuente: Propia.

Una vez importada *GNS3 VM* se abren las opciones de la máquina virtual. Por defecto, se tiene una configuración de hardware bastante baja para los requerimientos, por lo tanto, se tienen que subir. Una cantidad que proporciona un rendimiento adecuado es a partir de 4GB de RAM, siendo recomendable unos 8GB. En cuanto al número de procesadores, se recomienda utilizar un mínimo de 4 núcleos virtuales, cuantos más, más fluida va a ser experiencia a la hora de configurar las topologías. Otro elemento importante es tener un adaptador de red seleccionado como NAT para el correcto funcionamiento de las aplicaciones.

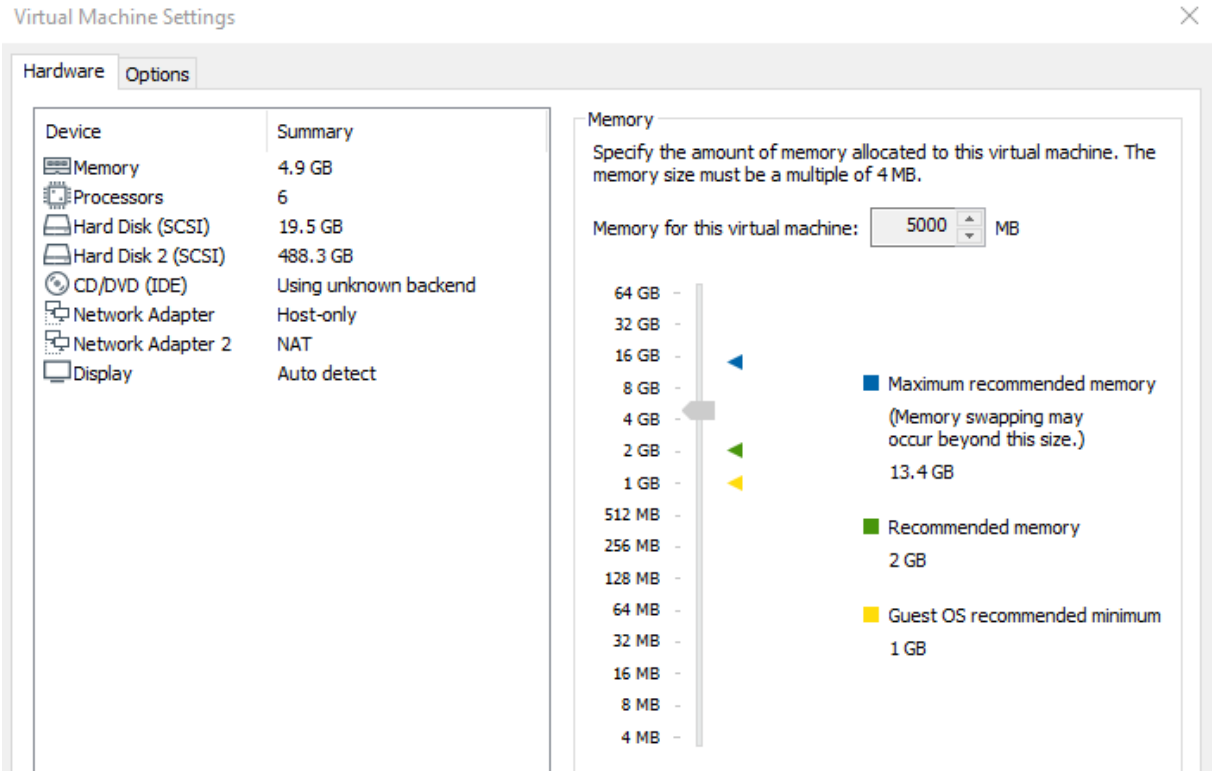


Fig. 8.3.3. Ajustes de máquina virtual en *VMware*. Fuente: Propia.

En este punto se apaga la máquina virtual, ya que tiene que ser ejecutada por el propio *GNS3*.

8.4. Configuración GNS3

Una vez se tiene la máquina virtual lista, en el menú de selección de servidor, se elige la opción del *GNS3 VM*. Esto va a llevar a un menú que va a permitir elegir las características de este.

GNS3 va a realizar una búsqueda en las máquinas virtuales disponibles, y si la conexión con *VMware* es correcta, va a mostrar *GNS3 VM* para seleccionar. También permite seleccionar las características de hardware de la máquina virtual, por lo tanto, se utilizan las mismas especificaciones, teniendo en cuenta de que la RAM debe ser múltiplo de cuatro.

A continuación, en los ajustes de *GNS3*, en la pestaña de *GNS3 VM* se le da a la opción de *Activar GNS3 VM*, y se aplican los cambios. La máquina virtual de *GNS3 VM* tiene que arrancar sola a partir de ahora.

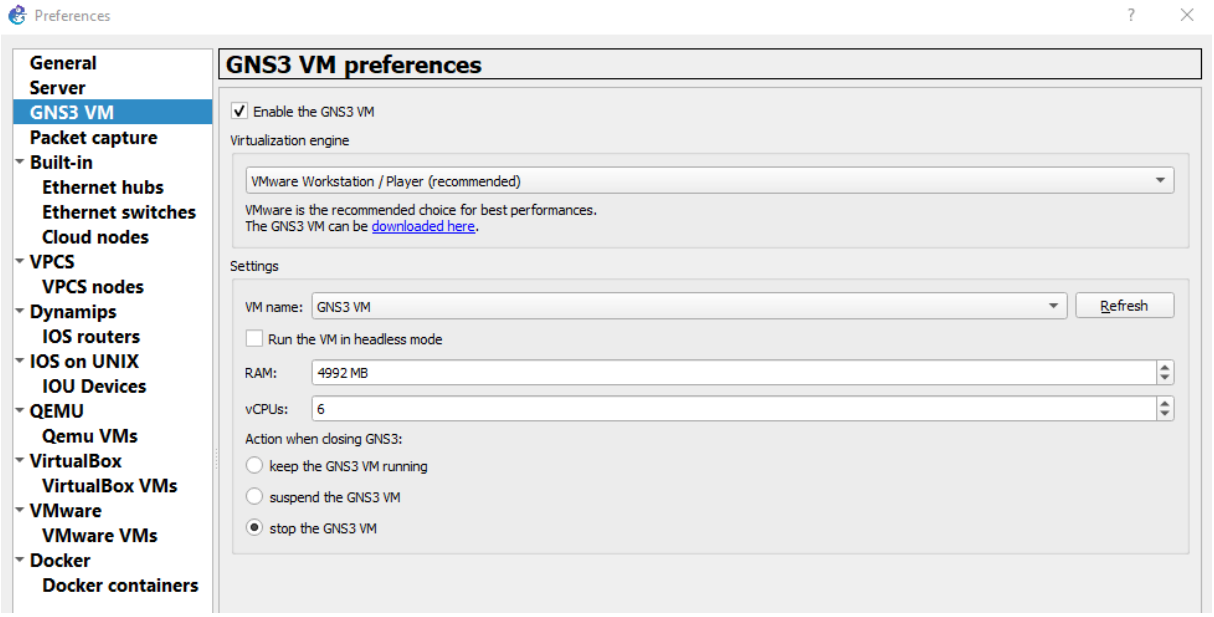


Fig. 8.4.1. Ajustes de *GNS3VM* en *GNS3*. Fuente: Propia.

Se puede distinguir que todo está bien configurado, si en el recuadro de la derecha, aparecen en verde los siguientes iconos, el primer servidor siendo el propio ordenador, y el segundo, la máquina virtual de *GNS3*.

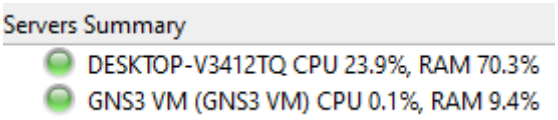


Fig. 8.4.2. Servidores en funcionamiento *GNS3*. Fuente: Propia.

8.5. Máquina virtual Mininet

La máquina virtual de *Mininet* es una herramienta muy útil para la simulación de grandes topologías. De la misma manera que la máquina virtual de *GNS3*, se ejecuta sobre *VMware*. Para ello lo primero que se necesita es descargar la imagen de *Mininet*, la cual se puede encontrar en el repositorio oficial de *GitHub* de *Mininet* (https://github.com/mininet/mininet/releases/).

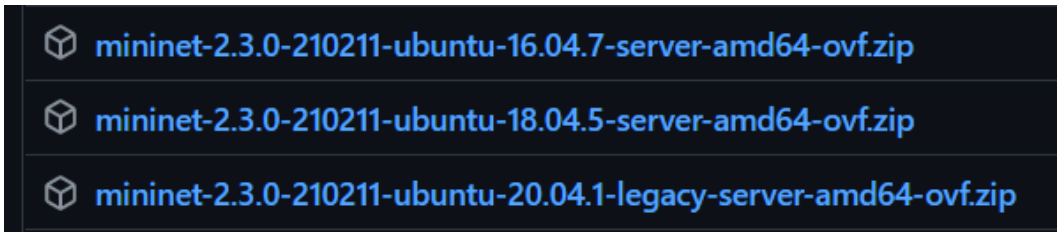


Fig. 8.5.1. Diferentes versiones de *Ubuntu* para *Mininet*. Fuente: Propia.

Según la versión que se necesite, se aconseja tener versiones del *Ubuntu* de *Mininet* semejantes a las versiones de los controladores que se van a usar, de esta manera, toda la configuración que se realice en torno a *OpenFlow*, no varía entre dispositivos. Además, para el correcto funcionamiento de todos los componentes de las topologías, se recomienda utilizar una versión de *Mininet* igual o superior a la 2.2.2.

Una vez importada la máquina virtual de *Mininet* a *VMware*, de la misma manera que la máquina virtual de *GNS3*, se procede a importar esta a GNS3. Para ello se va en *GNS3* a las preferencias, una vez allí, en el apartado de *VMware VMs* se le da a *New*.

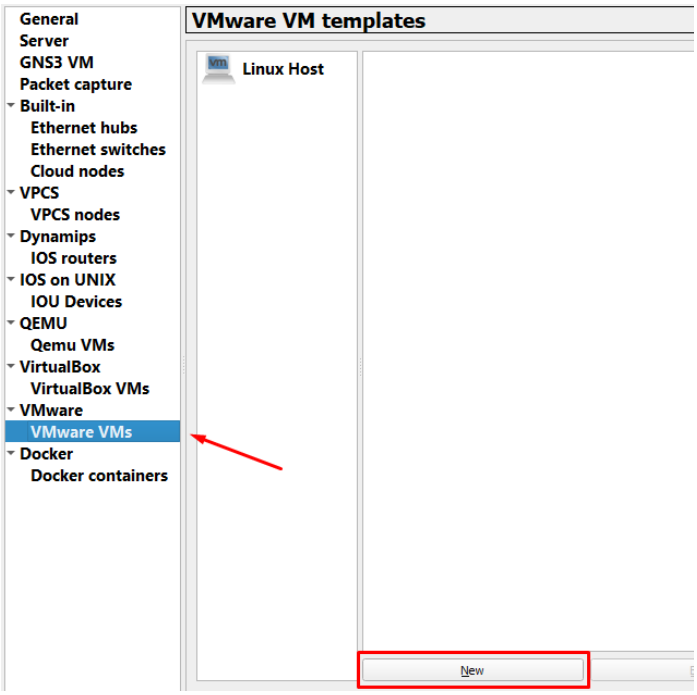


Fig. 8.5.2. Menú importar máquina virtual a *GNS3*. Fuente: Propia.

Se continua hacia delante, hasta que sale el menú de selección de la VM, aquí es donde debe salir la VM de *Mininet*, en el caso de estar bien importada a *VMware*. A partir de aquí, ya se tiene una máquina virtual de *Mininet* lista para su uso.

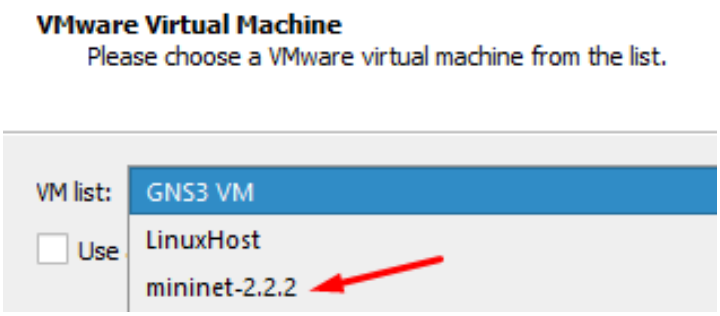


Fig. 8.5.3. Seleccionable de máquina virtual en *GNS3*. Fuente: Propia.

9. Creación de topologías SDN

En este apartado se utilizan las aplicaciones configuradas en el tema anterior, para crear topologías simples con las cuales, se muestra cómo se trabaja con redes *SDN*, el proceso de configuración de una topología.

La topología usada a partir de este tema, está compuesta de dos controladores, un switch *OVS* que actúa como un switch normal, una máquina virtual de Mininet, dos switch *OVS* que se conectan al controlador, y conectados a estos switch, un host en cada uno. El controlador *Controller1* toma el papel de controlador para la topología simulada desde *Mininet*. El otro controlador, *Controller2*, se conecta directamente con los switch *OVS* físicos en *GNS3*. De esta manera, cada controlador maneja su topología, sin interferencias entre uno y otro.

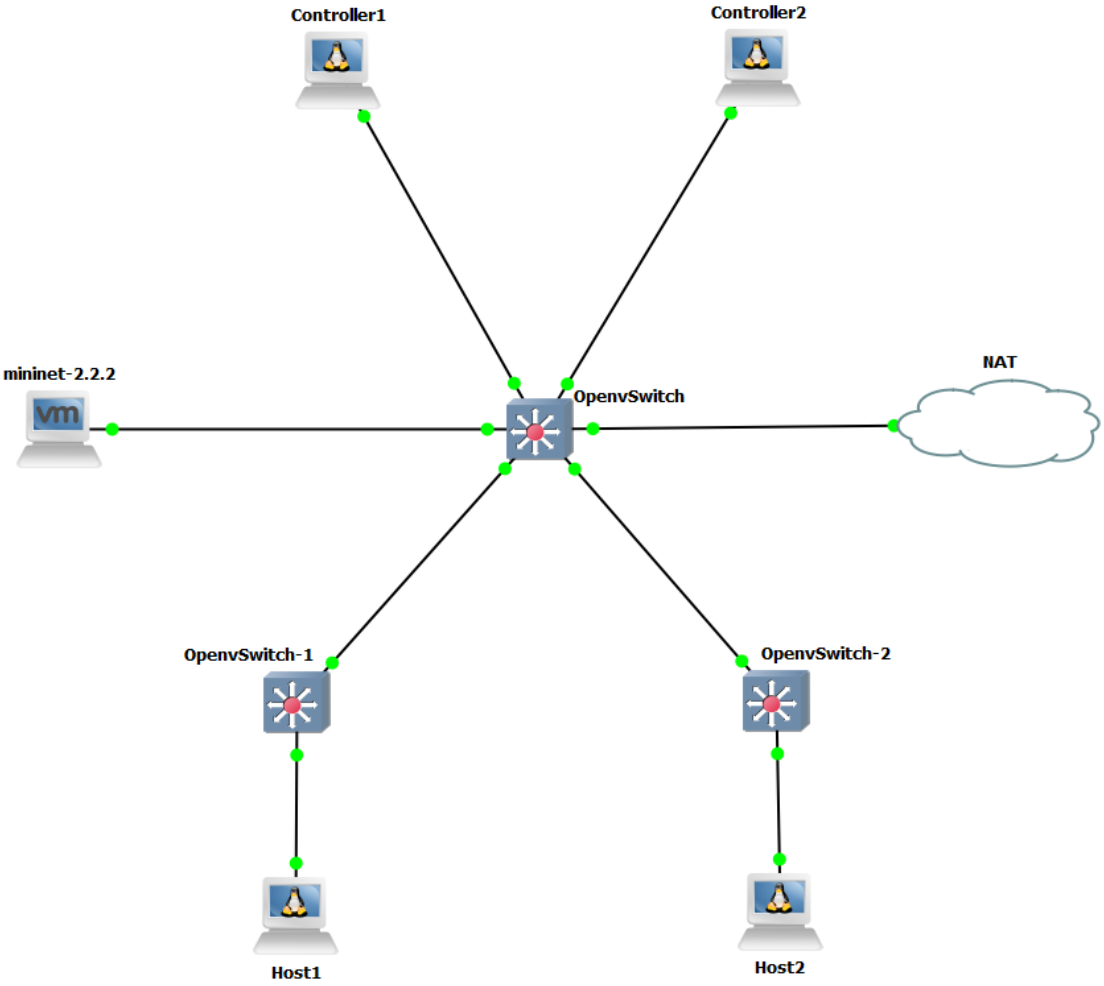


Fig. 9.1. Topología montada en *GNS3*. Fuente: Propia.

9.1. Controlador OpenDayLight

El controlador de *OpenDayLight* se va a instalar sobre un docker de *Ubuntu*, aunque técnicamente, cualquier sistema con una línea de comandos *Linux* va a permitir realizar esta instalación. Se recomienda empezar con versiones de *Ubuntu* 16 y superiores para mayor compatibilidad con las herramientas a instalar sobre este.

En la parte superior de *GNS3*, se le da a *File*, *New Template*, *Install an appliance from the GNS3 server*. Aquí es donde *GNS3* permite elegir descargar un catálogo extenso de dispositivos, aquí se elige el *Ubuntu Docker Guest.*[18]

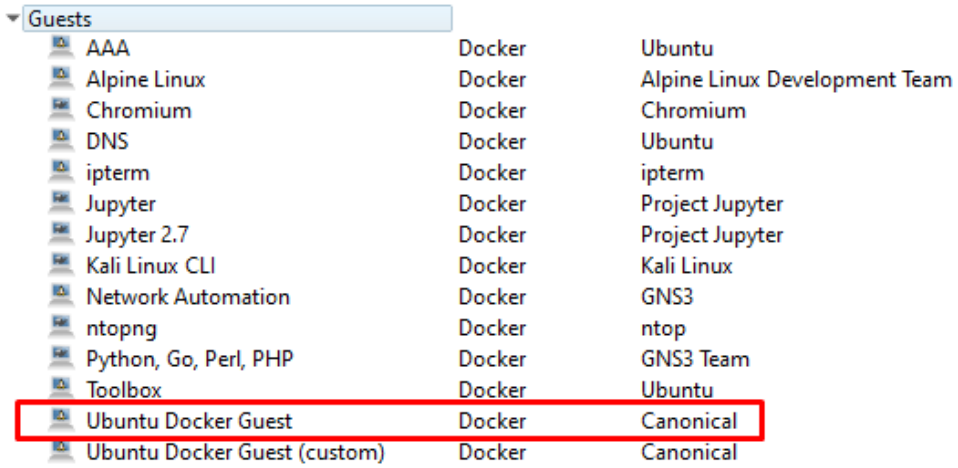


Fig. 9.1.1. Opciones de *dockers* en *GNS3*. Fuente: Propia.

Una vez elegido el dispositivo a instalar, abajo del todo, se le da a *Install*, aquí, da a elegir dónde se quiere instalar este dispositivo. Ya que se ha configurado la *GNS3 VM*, se va a instalar sobre esta.

Este Docker de *Ubuntu* viene con la versión 16.04, en el caso de necesitar una versión superior de *Ubuntu*, por ejemplo, la 18.04, como es el caso para versiones más nuevas de *ODL* (versiones a partir del año 2018 aproximadamente), se procede a instalar un Docker diferente.

En la misma pestaña de instalación de aplicaciones del *GNS3* se selecciona el Docker con el nombre *AAA*, aunque el nombre no indique ninguna información sobre él, en realidad es un *Ubuntu 18.04* totalmente operativo, con la diferencia de que no está distribuido oficialmente por el equipo que desarrolla *Ubuntu*.

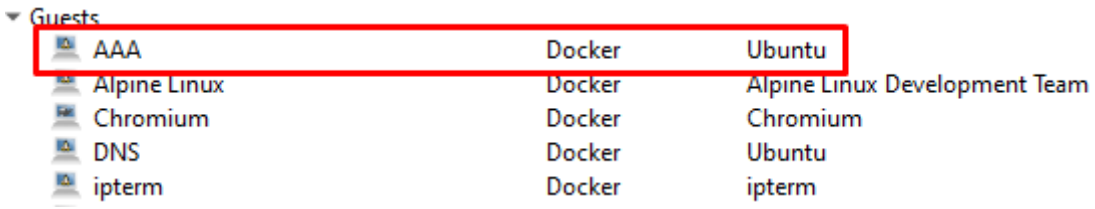


Fig. 9.1.2. *Docker* alternativo en *GNS3*. Fuente: Propia.

Una vez instalado correctamente el Docker de *Ubuntu*, tiene que salir en la parte izquierda, en la zona de dispositivos, tal y como muestra la Fig. 9.1.3.

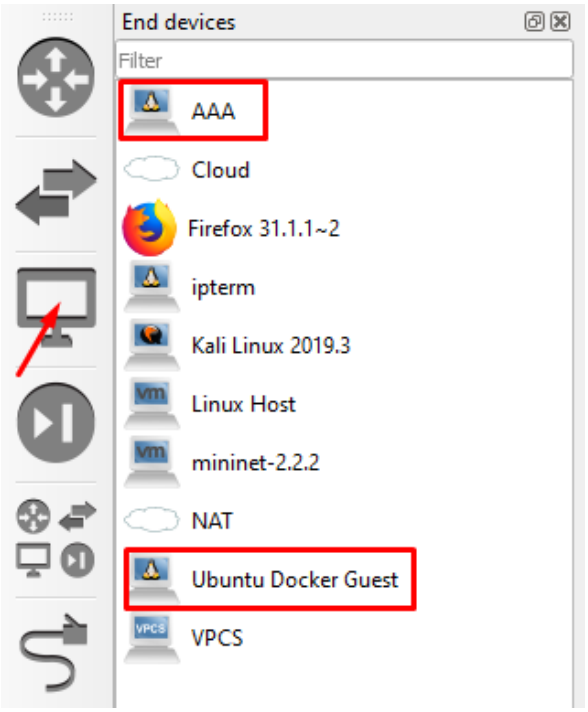


Fig. 9.1.3. *Dockers* instalados en el menú de *GNS3*. Fuente: Propia.

9.2. Switch OpenFlow - OpenVSwitch

Para poder controlar un switch con *ODL* mediante *OpenFlow*, se necesita que este, sea capaz de interpretar estos mensajes, por lo tanto, se utiliza un *OVS* (OpenVSwitch) físico, instalado desde la propia aplicación de *GNS3*.

De la misma manera que en la instalación del Docker para el controlador, en la parte de aplicaciones, se abre la pestaña de *Switches* y se selecciona el *Open vSwitch*, tal como se muestra en la Fig. 9.2.1.

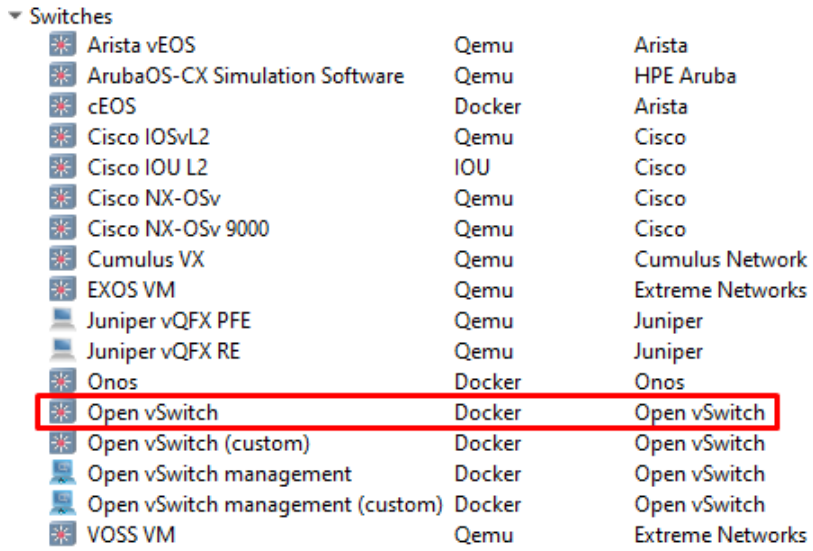


Fig. 9.2.1. *Docker* de *OVS* en *GNS3*. Fuente: Propia.

A continuación, se elige la opción de instalar sobre la *GNS3 VM*. Una vez finalizada la instalación, este va a aparecer en la sección de switches en *GNS3*.

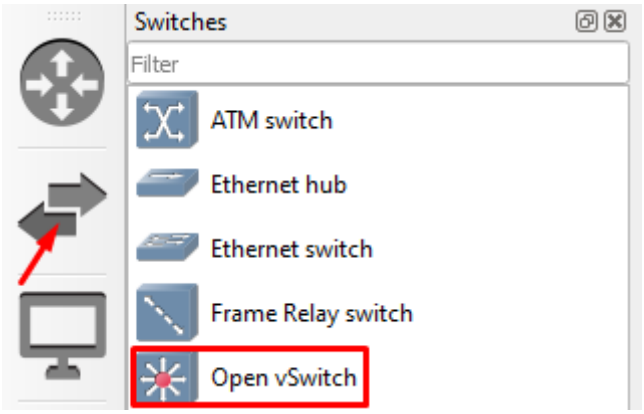


Fig. 9.2.2. *Docker* de *OVS* en el menú de *GNS3*. Fuente: Propia.

9.3. Configuración de topologías

Toda topología en *GNS3* requiere de conexión a internet para su correcto funcionamiento. Esta conexión se realiza conectándose a la red local mediante el propio ordenador, el cual hace una emulación de todos los dispositivos dentro de *GNS3*. Para ello, se necesita de un nodo *NAT*. El nodo *NAT* de *GNS3* reemplaza el nodo de *conexión estándar a internet* y el nodo *Cloud*, proporcionando como ventaja respecto a estos un control de licencias a la hora de instalar aplicaciones desde internet. Este requiere de la *GNS3 VM* para su correcto funcionamiento, ya que necesita poder crear un nodo virtual. En el caso de necesitar conectarse a la topología desde fuera (internet o área local), será necesario usar el nodo *Cloud*, que a día de hoy sigue presente en *GNS3*.

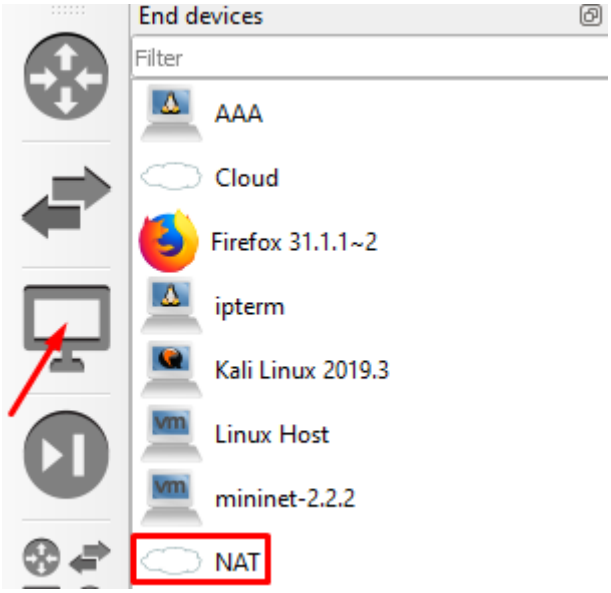


Fig. 9.3.1. Selección de *NAT* desde *GNS3*. Fuente: Propia.

Durante las pruebas realizadas se ha podido comprobar, de que conectando el nodo *NAT* directamente al ordenador, y no a *GNS3 VM*, el comportamiento no cambia para las configuraciones necesarias, presentando como ventaja de que se puede acceder desde fuera a las topologías, evitando así tener que utilizar interfaces gráficas en el propio *GNS3* para interactuar con el controlador, las cuales ofrecen una velocidad muy inferior respecto al propio ordenador.

Al arrastrar dispositivos ya instalados en *GNS3*, va a salir una ventana para seleccionar en que servidor se quieren ejecutar, como se ha mencionado antes, el nodo *NAT* se va a ejecutar sobre el propio ordenador. En cambio, los demás dispositivos, van a ir directamente a la *GNS3 VM*.

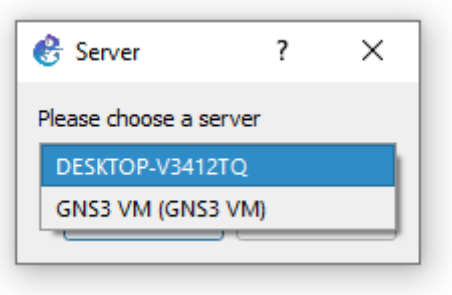


Fig. 9.3.2. Selección de servidor en *GNS3*. Fuente: Propia.

Para poder unir múltiples elementos dentro de una topología se usa un switch. Ya que se ha instalado un *OpenVSwitch*, es el que se va a usar para ello. Para unir los elementos, se le da al último elemento de la columna de la izquierda, como nuestra la Fig. 9.3.3.

Fig. 9.3.3. Botón añadir conector en *GNS3*. Fuente: Propia.

Se hace click en los elementos a unir y se seleccionan los puertos.

Fig. 9.3.4. Conexión *NAT* a *OVS* en *GNS3*. Fuente: Propia.

Se procede a conectar los elementos principales de la topología. Una topología básica, pero funcional puede ser de este tipo. Un controlador y dos *OVS* que pueden ser conectados mediante *OpenFlow*, aunque también, pueden ser utilizados como un switch normal. Una vez aquí, se puede proceder a añadir complejidad extra a los *OVS*, como por ejemplo hosts o más switches.

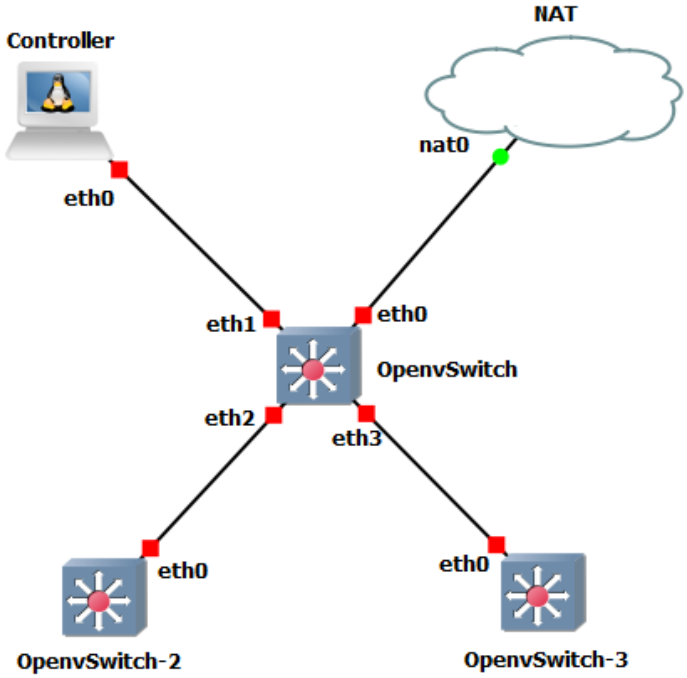


Fig. 9.3.5. Topología básica con solo *OVS* físico en *GNS3*. Fuente: Propia.

Antes de poder ejecutar los dispositivos, necesitan ser configurados. Para ello, y siempre apagados, se le da click derecho y *edit config*. Aquí, dependiendo del dispositivo, hay más o menos puertos que elegir, pero normalmente, se configura solamente el puerto *eth0*. Para ello se borra la # delante de las líneas de configuración que se necesitan activar.

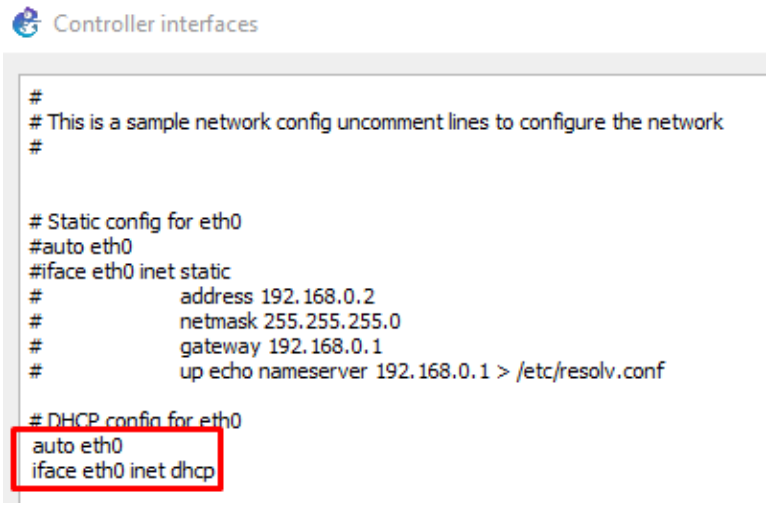


Fig. 9.3.6. Configuración DHCP . Fuente: Propia.

Las líneas borradas en la foto anterior indican que se está configurando la conexión mediante *DHCP*, es decir, la asignación de IP se va a hacer automáticamente por el router, en este caso, por el nodo *NAT*. Una vez guardados los cambios de la configuración, se ejecuta la topología.

Una de las maneras más rápidas de desplegar grandes topologías, es mediante *Mininet*. En el tema anterior se ha configurado *Mininet* en torno a *GNS3*, ahora, para añadirlo a la topología, solamente es necesario arrastrarlo, este se va a conectar directamente a la *GNS3 VM*. Este, siendo una máquina virtual, no necesita configuración. Si todo se ha configurado correctamente hasta ahora, se le va a dar al botón verde que está arriba y todos los dispositivos se van a encender con la configuración modificada previamente. En cuanto a *Mininet*, de la misma manera que al encender *GNS3*, automáticamente se abre la *GNS3 VM*, al encender el nodo de *Mininet* en la topología, se va a encender la máquina virtual de esta. Para entrar a la máquina virtual, se piden las credenciales, siendo ambas *mininet*.

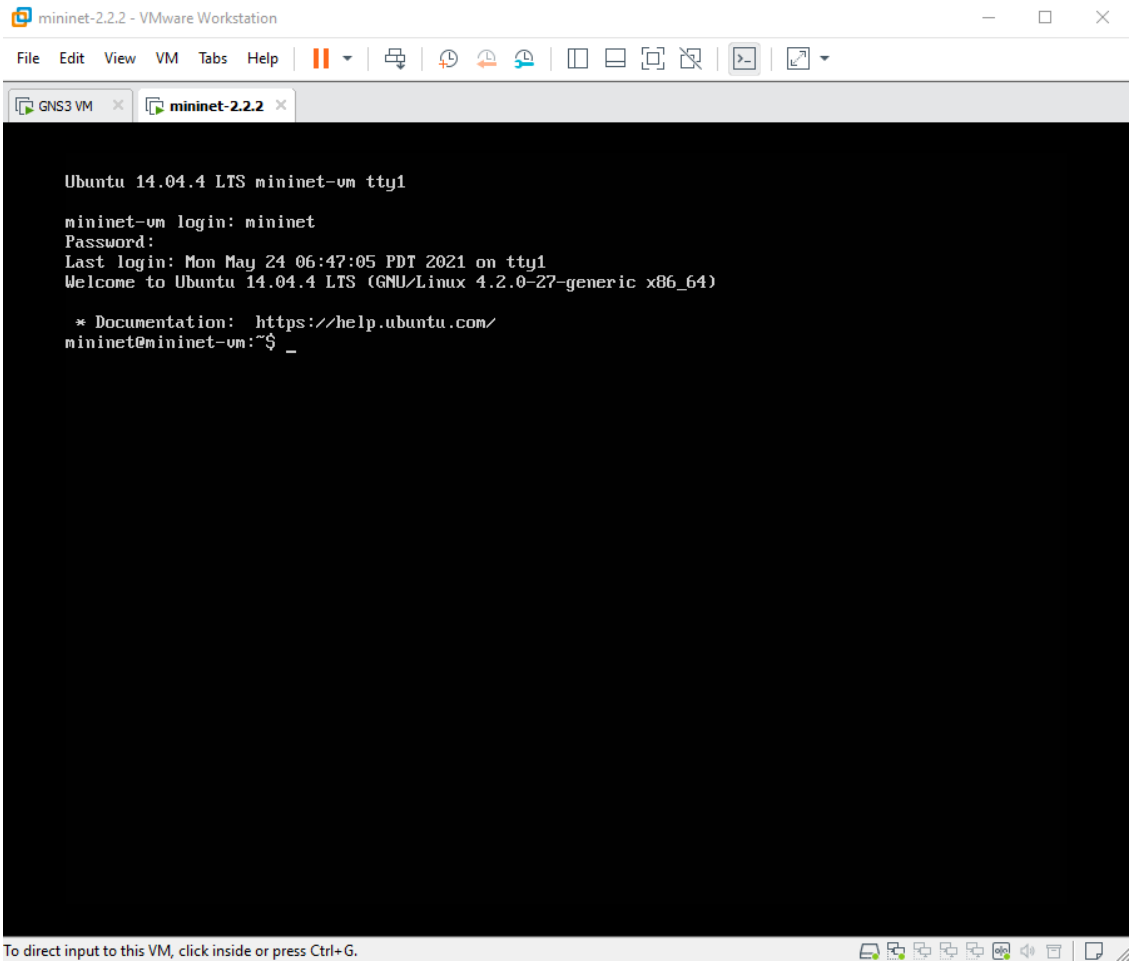


Fig. 9.3.7. Pantalla principal de *Mininet*. Fuente: Propia.

Una vez llegados a este punto, la topología está totalmente preparada para la instalación del software necesario para hacer de esta, una topología *SDN*.

9.4. Configuración del controlador

El controlador de ODL necesita una serie de programas y configuraciones concretas para su correcto funcionamiento. Se va a ver en este apartado toda la configuración necesaria para el correcto funcionamiento del controlador dentro de *GNS3*.

9.4.1. Programas base

Para comenzar con la configuración, se necesita abrir la consola de los dispositivos, click derecho sobre el dispositivo y se hace click a *Console*. Como el sistema operativo es *Windows 10*, la consola es la de *Solar-PuTTY*, la cual se ha instalado a la hora de instalar *GNS3*, pero en

caso de estar en Linux, se puede hacer con la consola por defecto. Una manera alternativa y más práctica de interactuar con la consola desde Windows 10, está disponible en el *Anexo1*.

Cada vez que se apaga y se enciende el controlador o cualquier dispositivo, la configuración aplicada dentro de él se borra, pero los archivos descargados no.

Para empezar, es necesario actualizar los paquetes del sistema, haciendo primero un *apt update*, seguido de un *apt upgrade*, asegurándose así de que todos los paquetes están al día.

Ya que *ODL* está basado en *Java*, es necesario tener este instalado. Para la versión que se va a utilizar, es suficiente *Java 8*, pero en el caso de usar una versión de *ODL* posterior a 2018, hay que instalar *Java 11*, para el cual, a su vez, es recomendable utilizar el docker de *Ubuntu* con la versión 18. Para la instalación de *Java 8* se utiliza el siguiente comando

- ***apt install openjdk-8-jre-headless***

seguido de

- ***export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64***

Este último sirve para especificar cuál es el *Java* por defecto. Ahora, si se hace un *echo $JAVA_HOME*, va a salir un mensaje de la versión por defecto de *Java*.[19]

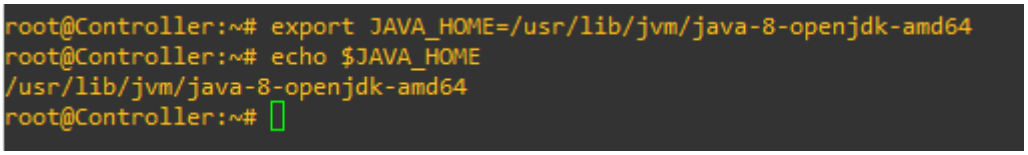


Fig. 9.4.1.1. Exportación del *JAVA_HOME*. Fuente: Propia.

Seguidamente, otras utilidades necesarias son *wget* para descargar archivos de internet y *unzip*, para descomprimirlos. Para ello se usa

- ***apt install wget unzip***

9.4.2. Configuración OpenDayLight

Para poder usar *ODL*, es necesario primero descargar la versión correcta desde internet.

Versión OpenDayLight	**Link al repositorio**
0.3. Lithium - 2015	https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.3.0-Lithium/distribution-karaf-0.3.0-Lithium.zip
0.6.4. Carbon - 2018	https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.6.4-Carbon/distribution-karaf-0.6.4-Carbon.zip
14.0. Silicon - 2021	https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/opendaylight/14.0.0/opendaylight-14.0.0.zip

Tabla 9.4.2.1. Versiones *ODL* más utilizadas. Fuente: Propia.

Para descargar el controlador, se hace un *wget* seguido del link al repositorio de la versión de *ODL*. En este caso se utiliza la versión *0.6.4. Carbon*. Esta versión, tiene disponibles los paquetes que necesitaremos más adelante, en cambio, versiones futuras a esta, comienzan a perder soporte. Estas nuevas versiones de *ODL* traen nuevas funcionalidades, pero ninguna de ellas son útiles para las pruebas de este trabajo. A día de hoy, y dependiendo de los paquetes con los que se trabaje, parece haber un compromiso entre versiones, es decir, no todas las versiones de *ODL* traen o funcionan bien con los mismos paquetes. Por ejemplo, existen paquetes que solamente están disponibles para la versión *0.3. Lithium*, pero al mismo tiempo, en esta versión, otros paquetes fundamentales no funcionan lo suficientemente bien. Pero para el caso de las pruebas de este trabajo, para la versión *0.6.4. Carbon*, se ha encontrado un conjunto de paquetes que funcionan correctamente.

Una vez descargada la versión de *ODL*, se hace un *unzip* seguido del nombre del archivo comprimido.

```
root@Controller:~# ls
distribution-karaf-0.6.4-Carbon.zip
root@Controller:~# unzip distribution-karaf-0.6.4-Carbon.zip
```

Fig. 9.4.2.1. Descomprimir el controlador. Fuente: Propia.

En este punto, se puede ejecutar *ODL*, navegando dentro de la carpeta descomprimida.

```
root@Controller:~# cd distribution-karaf-0.6.4-Carbon/bin
root@Controller:~/distribution-karaf-0.6.4-Carbon/bin# ls
aaa-cli-jar.jar  configure-cluster-ipdetect.sh  instance      karaf.bat          setenv.bat  start      status.bat
client           configure_cluster.sh           instance.bat  set_persistence.sh  shell       start.bat  stop
client.bat       custom_shard_config.txt        karaf         setenv              shell.bat   status     stop.bat
root@Controller:~/distribution-karaf-0.6.4-Carbon/bin#
```

Fig. 9.4.2.2. Archivos en la carpeta de *ODL*. Fuente: Propia.

En la carpeta *bin*, está el ejecutable de *ODL*, *./karaf*.

En el caso de requerir salir de *ODL* y ejecutarlo de nuevo, vacío de paquetes instalados previamente, se utiliza el comando *./karaf clean*. Una vez dentro, tiene que salir la interfaz del controlador.

```
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]

Karaf started in 4s. Bundle stats: 64 active, 64 total

    ________                ________                .__  .__       .__    __
    \_____  \ ______ ____   ____ \_____\ _____  ___.__.|  | |__| ____ |  |__/  |_
     /   |   \\____ \_/ __ \ /    \ |  |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|  `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/                 /_____/      \/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Fig. 9.4.2.3. Pantalla principal de *ODL*. Fuente: Propia.

Aquí existen varios comando a utilizar, los principales siendo, *feature:list*, para mostrar la lista de paquetes disponibles a instalar, y, *feature:install*, para instalar paquetes especificados. Los paquetes a instalar son los siguientes.

- ***feature:install odl-restconf odl-l2switch-switch odl-mdsal-all***
- ***feature:install odl-dlux-core odl-dluxapps-topology odl-dluxapps-nodes odl-dluxapps-yangui odl-dluxapps-yangman***

Este conjunto de paquetes es el básico para el correcto funcionamiento del controlador de *ODL*, y de la interfaz gráfica que se va a utilizar más adelante. En la primera línea, se encuentran los paquetes relacionados con las conexiones *Rest*, la configuración de un switch *L2*, y toda la colección de paquetes de *MDSAL*, respectivamente. La segunda línea corresponde a la interfaz gráfica, cada uno de ellos, siendo un módulo diferente.

9.4.3. Configuración OpenFlowManager

El *OpenFlowManager* (*OFM*) es una herramienta desarrollada por *Cisco*, que permite gestionar redes *SDN*. Gracias a su interfaz, facilita el control de flujos dentro un switch.

Lo primero, se abre una nueva consola sobre el controlador, haciendo click derecho sobre él y se le da a *Auxiliary console*, esto va a abrir una nueva consola, sin interrumpir la que tiene *ODL* ya ejecutándose.

Aquí se van a instalar unas herramientas más específicas para poder ejecutar *OFM*. Se instalan *npm*, *git*, *nodejs* y *grunt*, ya que *OFM* es un servidor que utiliza *Javascript* para su funcionamiento, utilizando el comando

- ***apt install npm git nodejs***

Una vez instalados estos, mediante *npm*, se instala el programa de automatización llamado *grunt*, con el cual se ejecuta todo el servidor, mediante el comando

- ***npm install -g grunt-cli***

Una vez instalado, se descarga el código del *OFM*, el cual está disponible en el repositorio oficial de *CiscoDevNet*.

- ***git clone https://github.com/CiscoDevNet/OpenDaylight-Openflow-App.git***

En este momento, ya hay dos carpetas en el controlador, la primera con *ODL*, y la segunda con *OFM*.

```
~ # ls
OpenDaylight-Openflow-App  distribution-karaf-0.6.4-Carbon
~ #
```

Fig. 9.4.3.1. Carpetas necesarias en el controlador. Fuente: Propia.

Para continuar con la configuración de *OFM*, es necesario primero mirar la dirección IP del controlador, ya que va a ser necesaria más adelante.

Para la configuración de *OFM*, es necesario cambiar un archivo dentro de la carpeta del mismo, para ello se utiliza el siguiente comando

- ***nano ./ofm/src/common/config/env.module.js***

Aquí se abre un editor de texto con la configuración del *OFM*, en el cual hay que cambiar la dirección IP por la del controlador. Se aplica en línea de *baseURL*, cambiando localhost por la IP del controlador.[20]

```
define(['angularAMD'], function(ng) {
    'use strict';

    var config = angular.module('config', [])
        .constant('ENV', {
          baseURL: "http://localhost:",
          adSalPort: "8181",
          mdSalPort : "8181",
          ofmPort : "8181",
          configEnv : "ENV_DEV",
          odlUserName: 'admin',
          odlUserPassword: 'admin',
          getBaseURL : function(salType){
              if(salType!==undefined){
                  var urlPrefix = "";
                  if(this.configEnv==="ENV_DEV"){
                      urlPrefix = this.baseURL;
                  }else{
                      urlPrefix = window.location.protocol+"//"+window.location$
```

Fig. 9.4.3.2. Archivo configuración de *OFM*. Fuente: Propia.

Una vez guardados los cambios en el archivo, y dentro de la carpeta del OFM, se ejecuta el servidor mediante el comando *grunt*.

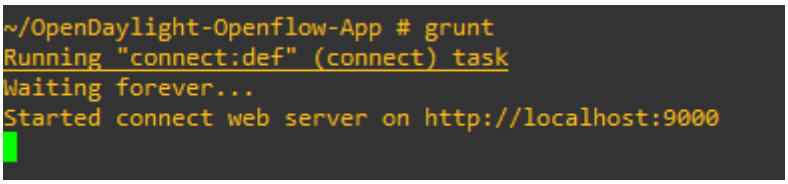


Fig. 9.4.3.3. *OFM* en funcionamiento. Fuente: Propia.

Antes de finalizar la configuración de *OFM*, es necesario instalar un paquete más en *ODL*. En la versión *0.3. Lithium* este paquete se llama *odl-openflowplugin-all*, en versiones más nuevas, este paquete ya no existe. Un paquete que cumple básicamente todos los roles de este, en la versión *0.6.4. Carbon*, es *odl-openflowplugin-flow-services-ui*.

- ***feature:install odl-openflowplugin-flow-services-ui***

En este momento, tanto *ODL* como *OFM* están totalmente configurados. Para acceder a la interfaz gráfica de ODL, hay que ir desde una ventana del navegador, a la dirección

- ***IP_DEL_CONTROLADOR:8181/index.html#/login*** *(Ejemplo: 192.168.127.133:8181/...)*

Aquí, la pantalla de login va a pedir las credenciales, las cuales son *admin* tanto para usuario como contraseña.

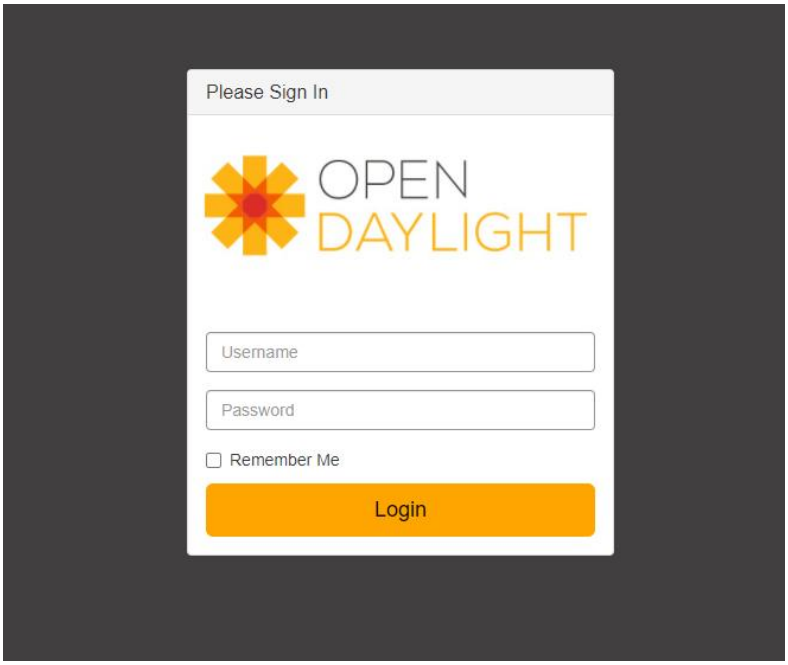


Fig. 9.4.3.4. Pantalla principal de *DLUX*. Fuente: Propia.

Para acceder a la aplicación de *OFM* la dirección es

- ***IP_DEL_CONTROLADOR:9000*** *(ejemplo: 192.168.127.133:9000)*

Este no requiere de autentificación para el acceso.

Dado que se ha configurado el nodo *NAT* para que sea ejecutado sobre el ordenador y no desde *GNS3 VM*, el acceso a estas direcciones se puede hacer desde el propio ordenador, así evitando el uso de aplicaciones dentro de *GNS3* que ralentizan exponencialmente el uso de estas interfaces.

9.5. Configuración OpenFlow

Para que un switch *OpenFlow* sea visto como tal por el controlador, es necesario especificar esto dentro del switch. A continuación, se muestran dos formas diferentes de registrar un switch, la primera mediante un switch físico, y la segunda, mediante *Mininet*.

9.5.1. Configuración OpenFlow - OVS

Una vez los dispositivos están configurados mediante *DHCP* y tienen una IP, se puede proceder a configurar *OpenFlow* en ellos.

De la misma manera que con un controlador, hay que abrir una consola a uno de los *OVS*, y se comprueba de que tiene IP mediante *ifconfig eth0*, para asignarle al controlador.

- ***ovs-vsctl set-controller br0 tcp:[IP_DEL_CONTROLADOR]:6633***

Con este comando, se le especifica al switch de que se conecte mediante *tcp* a la dirección especificada en el puerto 6633, el cual es el puerto por defecto en *OpenFlow*. Cada switch que se quiera registrar para ser visto por el controlador, debe de configurarse con este comando.

En este momento, en el apartado de *Topology* dentro de *ODL*, se puede observar que este switch ya se ha registrado.

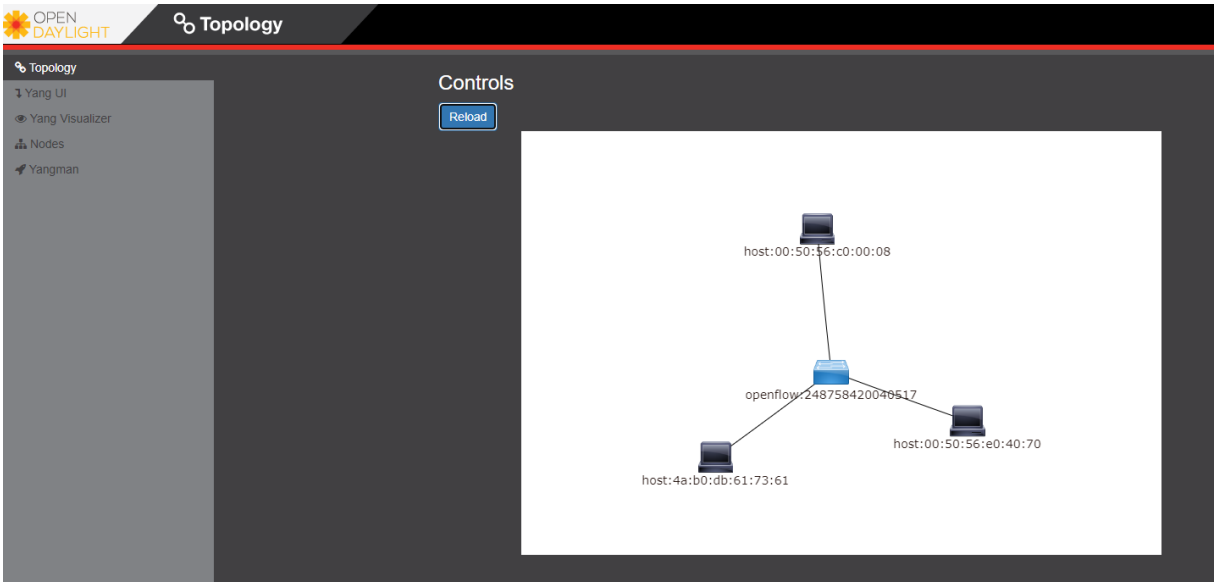


Fig. 9.5.1.1. Topología configurada en funcionamiento desde *DLUX*. Fuente: Propia.

9.5.2. Configuración OpenFlow - Mininet

En apartados anteriores se ha configurado *Mininet*. Una vez se enciende el nodo de *Mininet*, y se ponen las credenciales dentro de la máquina virtual, se puede pasar a crear topologías

Existen dos formas diferentes de hacerlo, la primera con un simple comando, y la segunda, mediante scripts de *Python*.

Para la creación de topologías mediante comando, hay que utilizar el siguiente,

- ***sudo mn --controller=remote,ip=ip_controlador --switch=ovsk,protocols=OpenFlow13 --mac --topo=tree,2***

Si se sigue el comando paso a paso, se puede apreciar que se define un controlador remoto, con una IP (la IP del controlador), también se especifica que el tipo de switch es un *OVS* con el protocolo *OpenFlow 1.3*, todos ellos con una MAC, y la topología en forma de árbol con dos nodos cada uno.

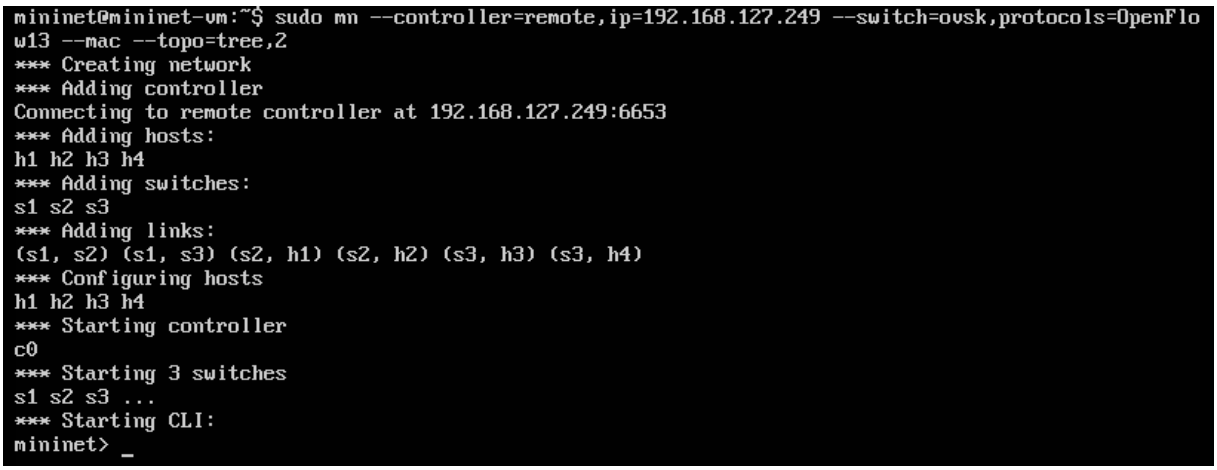


Fig. 9.5.2.1. Arranque de topología en *Mininet*. Fuente: Propia.

Después de hacer un *pingall*, la topología va a quedar de la siguiente manera, tal y como se muestra en la Fig. 9.5.2.2.

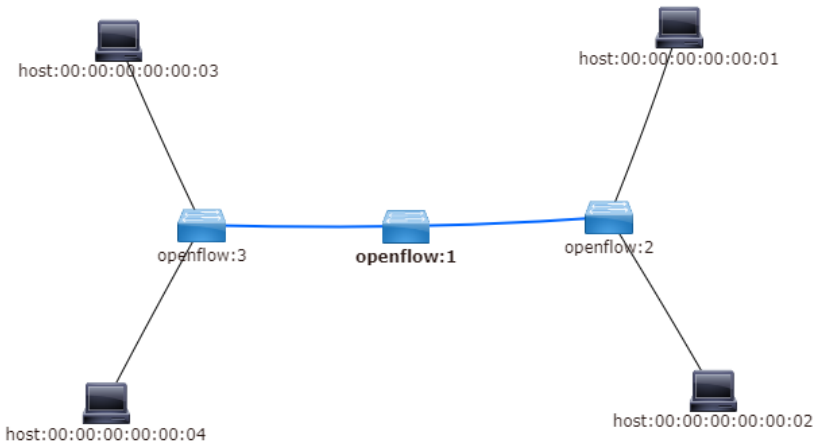


Fig. 9.5.2.2. Topología creada en *Mininet* vista desde *DLUX*. Fuente: Propia.

Las topologías creadas con *Python* pueden llegar a ser mucho más personalizadas, ofreciendo el control total al usuario, a la hora de la configuración. Esto es posible gracias a que *Mininet* presenta una API específicamente diseñada para trabajar con *Python*. La manera de ejecutar las topologías, respecto a la versión con la línea de comandos, también es diferente.

- ***sudo python nombre_del_archivo.py***

Para ello lo primero que debemos hacer es importar las librerías disponibles creadas por *Mininet*.[21]

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
```

A partir de aquí se crea la topología. El código completo está disponible en el *Anexo2*.

```
topo = ... // variable que contiene la topología
net = Mininet(topo) // objeto con la configuración completa
net.start() // ejecución de la topología
CLI(net) // comando que permite seguir en la consola de Mininet
```

10. Herramientas SDN

En este apartado se muestran varias formas de trabajar con las herramientas instaladas en apartados anteriores. Es posible ver la información acerca de las topologías de diferentes maneras, cada una de ellas con sus ventajas e inconvenientes. Para las demostraciones se va a utilizar una topología con dos controladores, uno de ellos está conectado a *OVS* físicos dentro de la topología, cada uno con un host. El otro controlador está conectado a Mininet, desde el cual se crea una topología en árbol, con tres *OVS*, y cuatro hosts. Todo esto para demostrar que independientemente de la manera de crear las topologías, los resultados va a ser similares.

10.1. OVS

Cada switch *OVS* tiene información dentro de él que se puede utilizar para ver fragmentos de la configuración. Esto puede ayudar a ver si se han producido cambios aplicados en otras herramientas, por ejemplo.

Existen varios comandos para ello, habiendo diferencias entre *OVS* físicos y aquellos *OVS* que se encuentran dentro de *Mininet*.

Versión	*OVS Físico*	*OVS Mininet*
Puertos	*ovs-ofctl -O OpenFlow13 dump-ports br0*	*sh ovs-ofctl dump-ports -O OpenFlow13*
Flujos	*ovs-ofctl -O OpenFlow13 dump-flows br0*	*sh ovs-ofctl dump-flows -O OpenFlow13*
Conexión	*ovs-vsctl show*	*sh ovs-ofctl show -O OpenFlow13*

Tabla 10.1.1. Comandos más utilizados en un OVS. Fuente: Propia.

Ambas versiones cumplen la misma función, la diferencia se encuentra en que, mientras el primero es ejecutado por el switch mismo, el segundo, por *Mininet*. Así, con el comando de la primera columna, se puede ver todos los puertos de los que dispone el switch.

```
/ # ovs-ofctl -O OpenFlow13 dump-ports br0
OFPST_PORT reply (OF1.3) (xid=0x2): 17 ports
  port 10: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=2615, errs=0, coll=0
           duration=1285.073s
```

Fig. 10.1.1. Puertos vistos desde un switch *OVS*. Fuente: Propia.

Ambas versiones muestran la misma información. Por cada switch, se indica la versión del protocolo *OpenFlow* y el número de puertos. Por cada puerto, se ve el número de paquetes recibidos, descartados, enviados, errores, etc. En el caso de la imagen de arriba, se puede ver que, por este puerto, no se han recibido paquetes, han sido descartados, esto se debe a que en este puerto no hay dispositivos conectados. Se puede ver un ejemplo de un puerto activo desde *Mininet*.

```
mininet> dpctl dump-ports -O OpenFlow13
*** s1 ------------------------------------------------------------------------
OFPST_PORT reply (OF1.3) (xid=0x2): 3 ports
  port  1: rx pkts=215, bytes=17615, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=217, bytes=17811, drop=0, errs=0, coll=0
           duration=922.882s
```

Fig. 10.1.2. Puertos vistos desde un switch *OVS* en *Mininet*. Fuente: Propia.

Se puede ver que la segunda línea delimita el switch número uno, aquí llamado como *s1*, este tiene tres puertos, 215 paquetes recibidos y 217 enviados.

Por el número bastante parejo entre paquetes enviados y recibidos, se puede llegar a la conclusión de que esto es resultado de un ping entre ellos. Mirando el comando anterior a este, se ve esta información verificada.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Fig. 10.1.3. Conectividad entre nodos en *Mininet*. Fuente: Propia.

Si miramos el comando de información de flujos, podemos ver información línea por línea, de cada flujo en un switch. Entre esta información podemos encontrar en qué tabla está el flujo, el número de paquetes que han hecho *match*, la prioridad del flujo y la acción a realizar, entre otros opcionales.

En el resultado de consola de abajo, se aprecia que el primer flujo pertenece a la tabla cero, han hecho *match* 44 paquetes, tiene una prioridad de 2 (baja), su condición de *match* es, aquellos paquetes que entren por el puerto uno, y la acción es enviar esos paquetes por el puerto dos. El tercer flujo, por ejemplo, tiene una prioridad más alta, también muchos más paquetes han hecho *match*, y como acción, tiene especificado enviar paquetes al controlador.

```
mininet> dpctl dump-flows -O OpenFlow13
*** s1 ------------------------------------------------------------
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000017, duration=998.225s, table=0,
n_packets=44,    n_bytes=3080, priority=2,in_port=1
actions=output:2
 cookie=0x2b00000000000016, duration=998.225s, table=0,
n_packets=46, n_bytes=3276, priority=2,in_port=2 actions=output:1
 cookie=0x2b00000000000008, duration=1004.11s, table=0,
n_packets=642, n_bytes=54570, priority=100,dl_type=0x88cc
actions=CONTROLLER:65535
```

Otro comando muy útil en *OVS* es la conexión, con este se ve la conexión del switch con el controlador. Tal y como se puede ver en la Fig. 10.1.4, sale que en *br0*, el switch está conectado por *TCP*, y el estado de la conexión, en este caso es *true* (conectado)

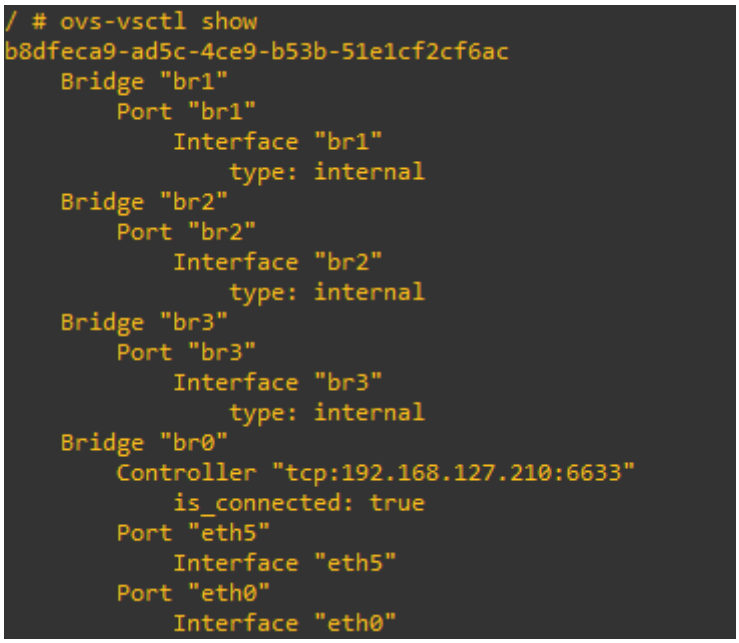


Fig. 10.1.4. Conexiones físicas de un switch *OVS*. Fuente: Propia.

10.2. DLUX

La aplicación web que permite interactuar con el controlador *ODL*, se llama *DLUX*. Tal y como se ha visto a la hora de configurar el controlador *ODL*, se tienen que instalar una serie de paquetes para el correcto funcionamiento de este. Desde *DLUX*, se puede ver el modelo de la topología (el esquema), la información de los nodos de cada uno de los switches y ver el inventario de peticiones, desde el que se puede consultar información de los elementos de la topología, y hacer modificaciones.

10.2.1. Módulo Topología

El módulo de la topología permite ver un esquema del diseño físico actual de la topología. Esta herramienta es útil ya que fuera de programas de emulación como *GNS3*, no es posible ver en todo momento como es el esquema de la topología.

Se puede observar la diferencia entre el esquema simulado por *DLUX*, y el esquema diseñado en *GNS3*, mostrado en la Fig. 10.2.1.1 y Fig. 10.2.1.2.

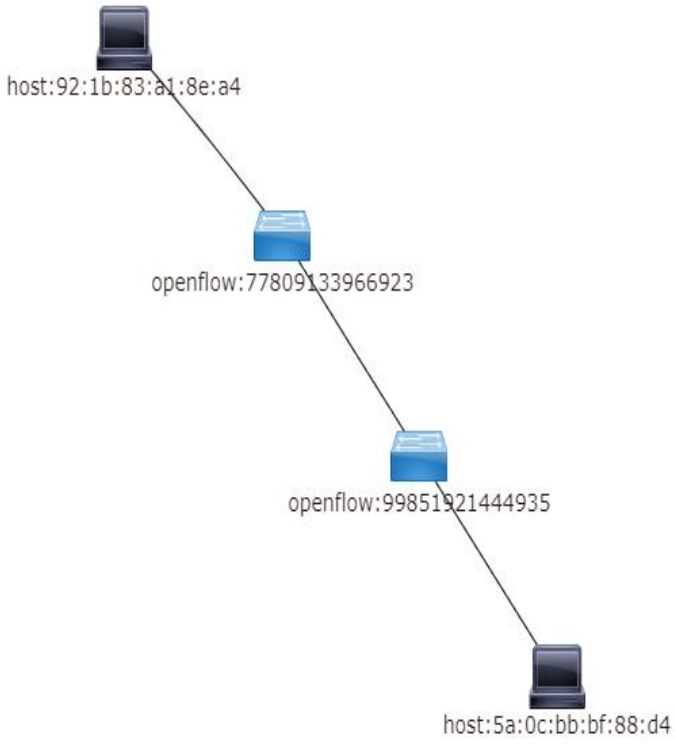


Fig. 10.2.1.1. Topología creada mediante switch físicos. Fuente: Propia.

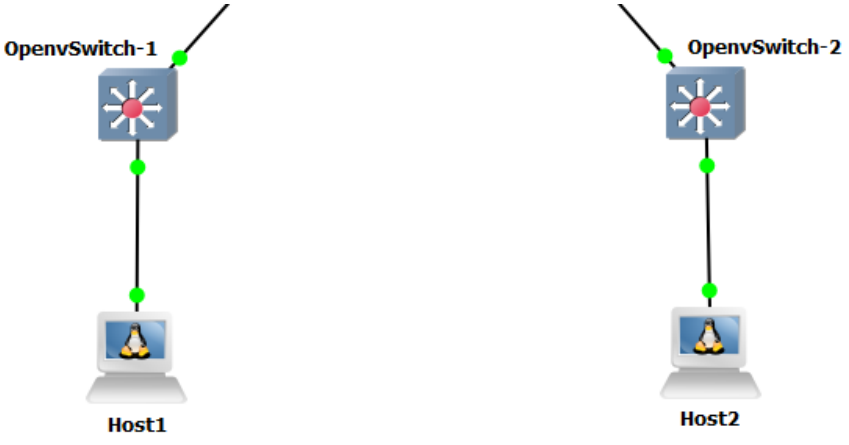


Fig. 10.2.1.2. Topología real de switch físicos en *GNS3*. Fuente: Propia.

En la topología de *DLUX*, se puede observar que ambos switch están conectados, pero en GNS3, no están conectados entre sí directamente, sino mediante otro switch en medio. La razón para ello es que este switch que está entre ellos, no está configurado para conectarse con el controlador, y por tanto es omitido. Pese a todo, el comportamiento que describen es el mismo.

Otra información que se puede encontrar en esta pantalla es, al pasar con el ratón por encima de cada dispositivo y conector, se ve información de estos.

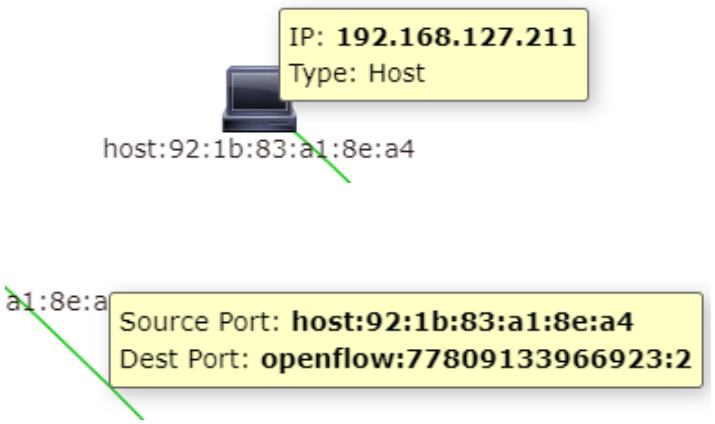


Fig. 10.2.1.3. IP y conectores vistos desde *DLUX*. Fuente: Propia.

10.2.2. Módulo Nodos

El módulo de los nodos permite ver información sobre todos los puertos disponibles en cada switch.

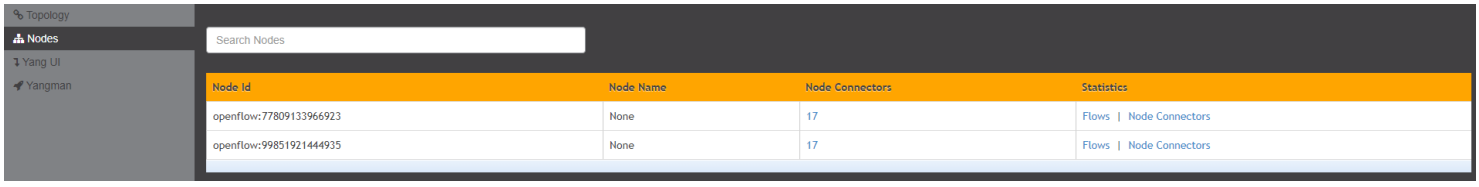
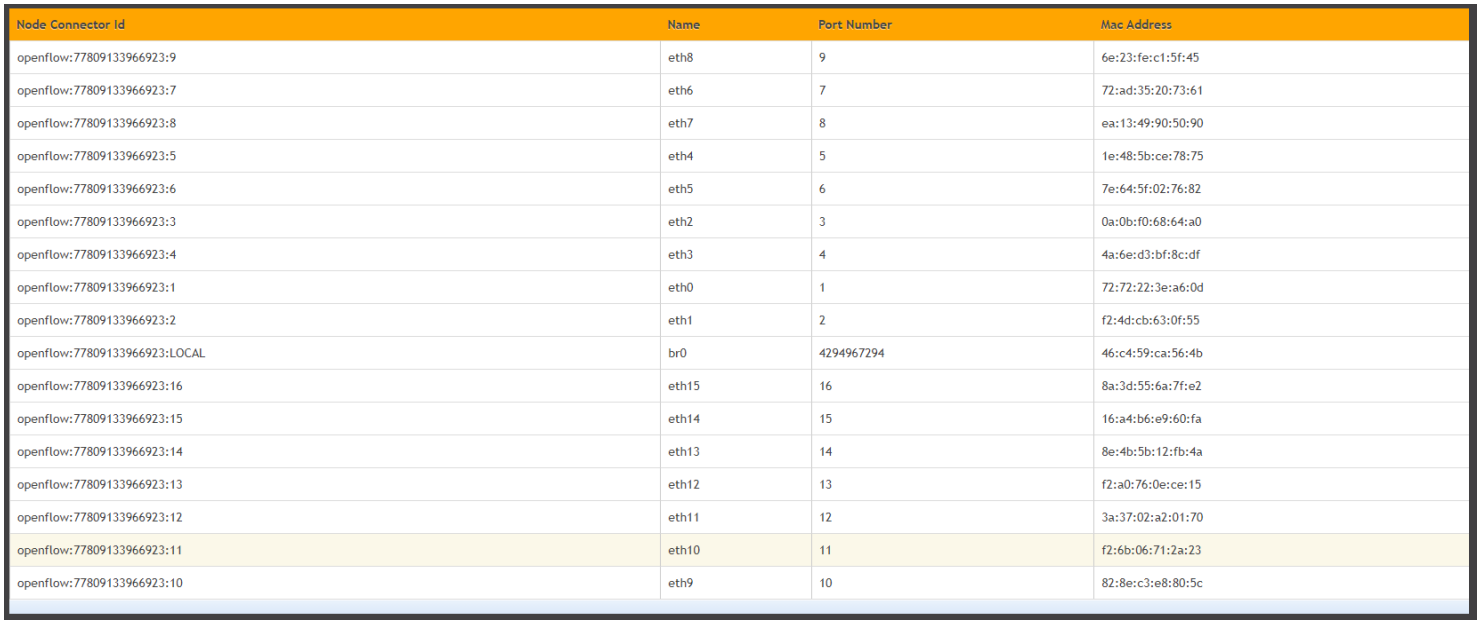


Fig. 10.2.2.1. Todos los switch OVS vistos desde *DLUX*. Fuente: Propia.

A Partir de la primera ventana, se puede ver información como el nombre y el número de puertos. Si se hace click en la columna de *Node Connectors*, se pueden ver todos los puertos del switch.



Node Connector Id	Name	Port Number	Mac Address
openflow:77809133966923:9	eth8	9	6e:23:fe:c1:5f:45
openflow:77809133966923:7	eth6	7	72:ad:35:20:73:61
openflow:77809133966923:8	eth7	8	ea:13:49:90:50:90
openflow:77809133966923:5	eth4	5	1e:48:5b:ce:78:75
openflow:77809133966923:6	eth5	6	7e:64:5f:02:76:82
openflow:77809133966923:3	eth2	3	0a:0b:f0:68:64:a0
openflow:77809133966923:4	eth3	4	4a:6e:d3:bf:8c:df
openflow:77809133966923:1	eth0	1	72:72:22:3e:a6:0d
openflow:77809133966923:2	eth1	2	f2:4d:cb:63:0f:55
openflow:77809133966923:LOCAL	br0	4294967294	46:c4:59:ca:56:4b
openflow:77809133966923:16	eth15	16	8a:3d:55:6a:7f:e2
openflow:77809133966923:15	eth14	15	16:a4:b6:e9:60:fa
openflow:77809133966923:14	eth13	14	8e:4b:5b:12:fb:4a
openflow:77809133966923:13	eth12	13	f2:a0:76:0e:ce:15
openflow:77809133966923:12	eth11	12	3a:37:02:a2:01:70
openflow:77809133966923:11	eth10	11	f2:6b:06:71:2a:23
openflow:77809133966923:10	eth9	10	82:8e:c3:e8:80:5c

Fig. 10.2.2.2. Todos los puertos de un switch en *DLUX*. Fuente: Propia.

Desde la columna *Statistics*, se puede ver la información de las tablas de flujos, el cual cumple la misma función que el comando de ver flujos visto en apartados anteriores, y ver información de los puertos, el cual, también, proporciona la misma información que el comando utilizado desde el switch que muestra los puertos con la información de paquetes enviados, recibidos, etc.

10.2.3. Módulo Inventory

El módulo inventario, también conocido como *Yang UI* o *Yangman*, el cual pertenece a los dos últimos nodos dentro de la aplicación, permite consultar o modificar el estado de cada switch de la topología. Se puede ver desde información detallada de cada nodo, hasta configurar los flujos que modifican el comportamiento de la topología.

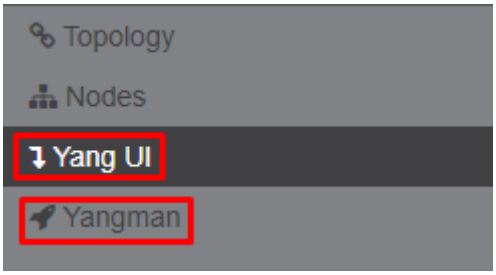


Fig. 10.2.3.1. Módulos *Yang UI* y *Yangman* en *DLUX*. Fuente: Propia.

Mirando dentro de uno de ellos, se puede ver que disponen de una multitud de módulos o paquetes con diferentes objetivos cada uno.

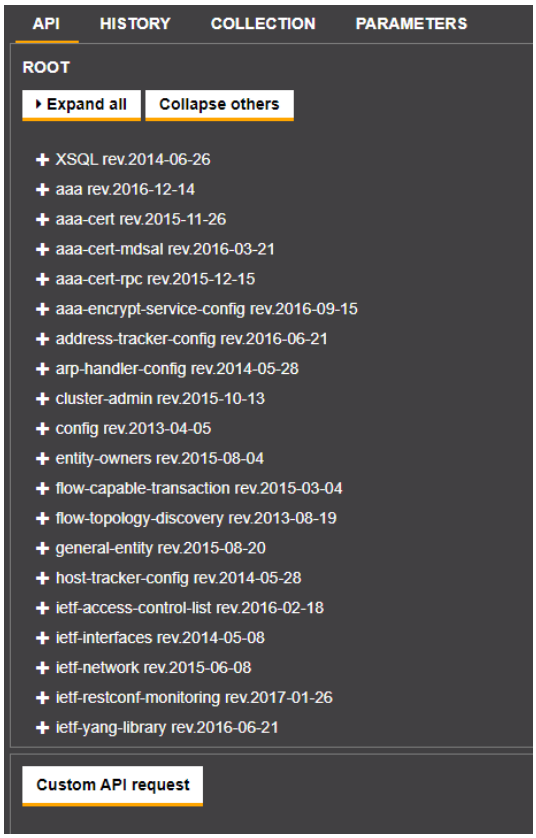


Fig. 10.2.3.2. Componentes de *ODL* disponibles desde *DLUX*. Fuente: Propia.

Muchos de estos módulos, disponen dentro de dos modos, *operational* o *config*, como se muestra en Fig. 10.2.3.3.

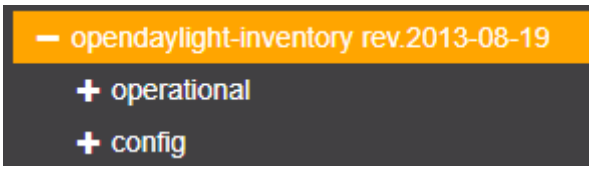


Fig. 10.2.3.3. Módulos *Operational* y *Config* de *ODL*. Fuente: Propia.

Con el modo operational, se ve la configuración actual de la topología. Dependiendo del módulo, esta información es diferente. Por ejemplo, si está instalado el paquete de *AAA* (autenticación), en este se puede consultar los diferentes certificados, políticas, etc. que se hayan configurado en cada switch, o en su ausencia, configurar nuevos. Bajando por las pestañas, se comienza a ver información cada vez más específica.

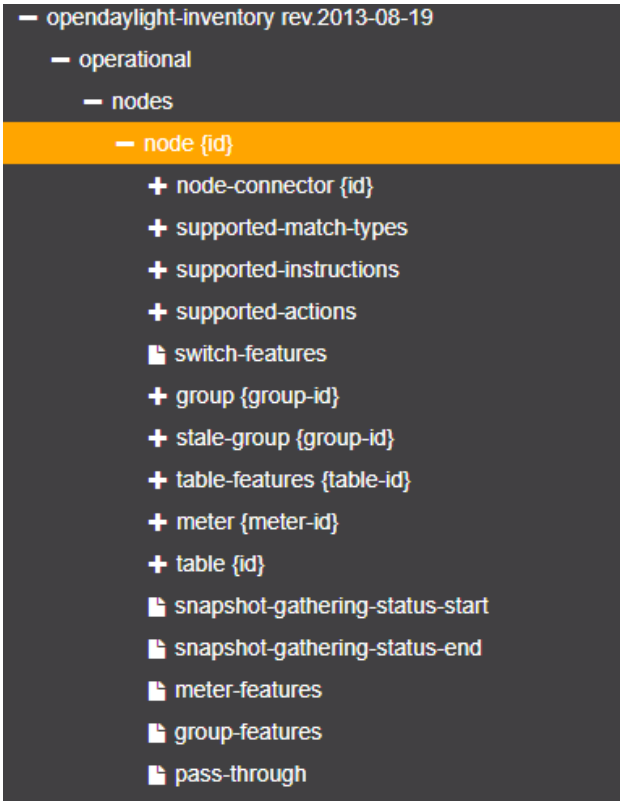


Fig. 10.2.3.4. Submódulos de cada componente de *ODL*. Fuente: Propia.

Cada una de estas líneas, es en realidad una llamada a la API de *ODL*, con criterios diferentes en la petición. En la parte de abajo está la zona de envío de peticiones y vista de resultados.

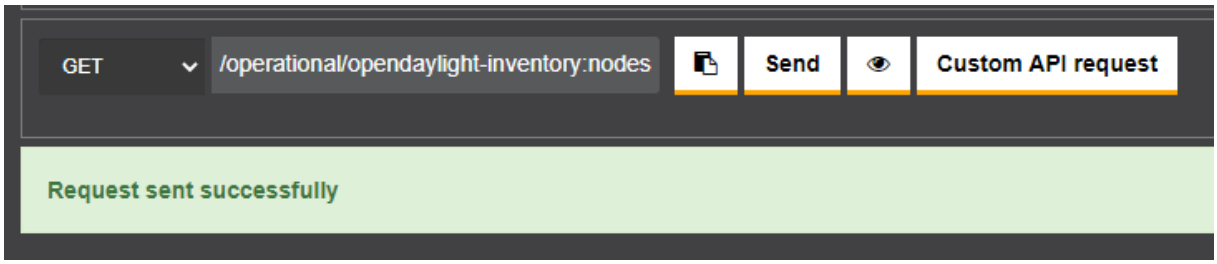


Fig. 10.2.3.5. Petición *operational* desde *Yang UI*. Fuente: Propia.

Como se puede ver, ya que el modo actual es el *operational*, solo se puede hacer un GET, es decir, solo se puede consultar información, pero no modificar. Si está seleccionado *nodes* y se envía la petición, llega la respuesta maquetada por la aplicación, la cual por defecto está en un formato *XML* o *JSON*.

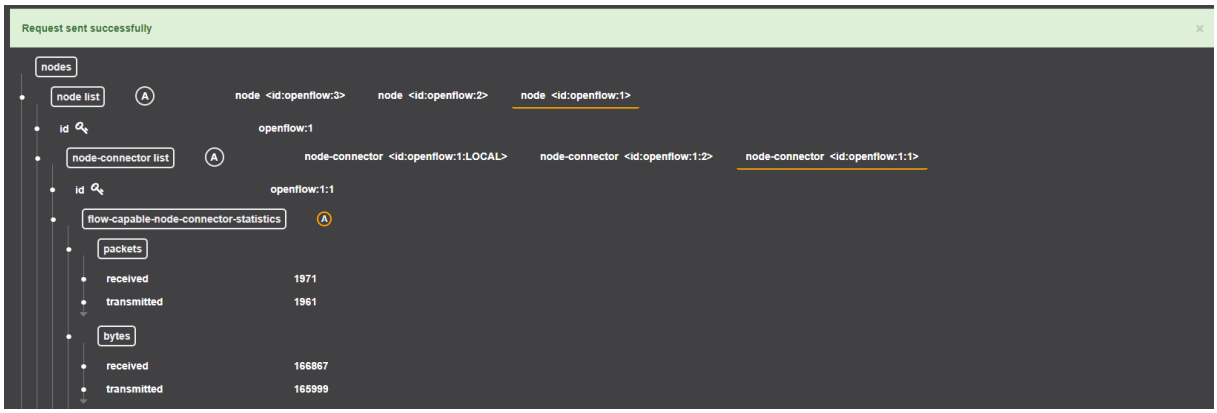


Fig. 10.2.3.6. Respuesta a petición en *Yang UI*. Fuente: Propia.

Si se hace lo mismo desde *Yangman*, se puede ver la información en formato *JSON*.

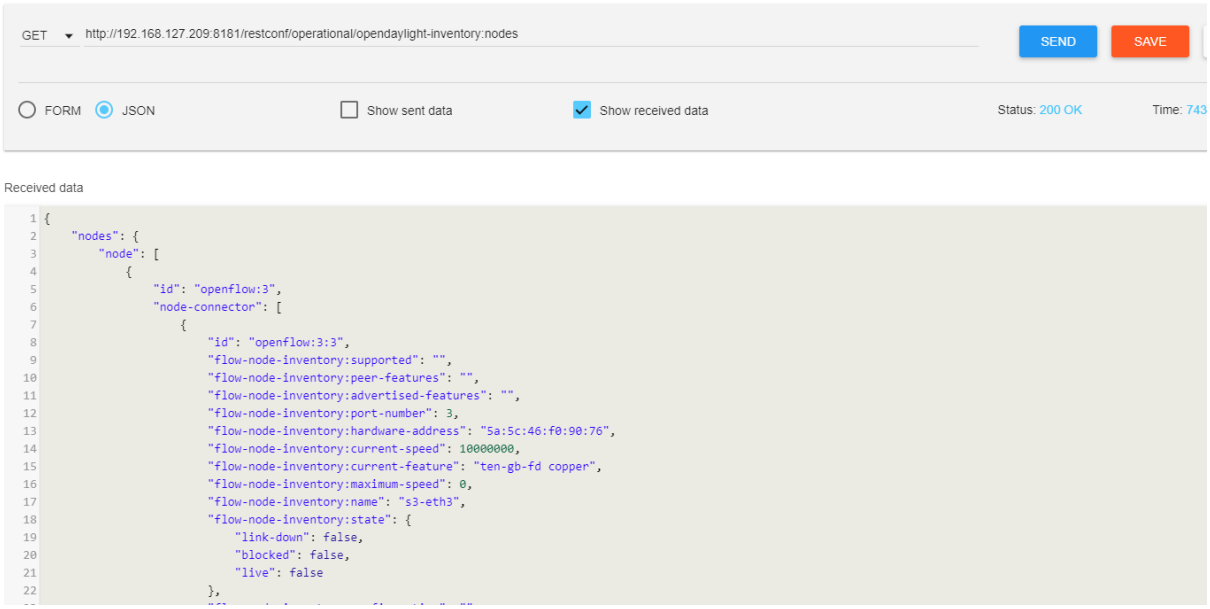


Fig. 10.2.3.7. Respuesta a petición en *Yangman*. Fuente: Propia.

10.3. OpenFlowManager

Cisco DevNet OpenDayLight OpenFlow Manager (*OFM*) es una aplicación desarrollada por Cisco, que permite comunicarse con el controlador *ODL* mediante una *REST API* a través de *RESTCONF*. Permite a los usuarios visualizar topologías, ver estadísticas e implementar flujos para alterar el comportamiento de estas.

10.3.1. Estructura del OFM

La aplicación dispone de cuatro pestañas. La primera, *Basic View*, de la misma forma que el módulo *Topology* de *DLUX*, muestra el esquema de la topología.

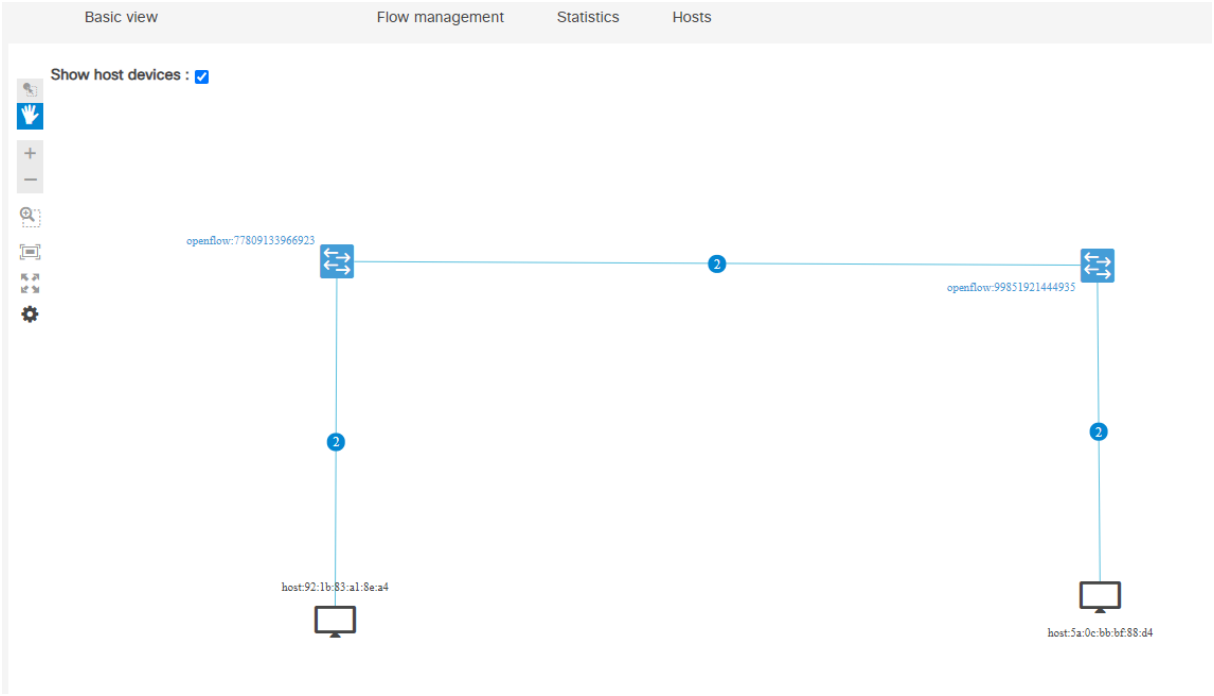


Fig. 10.3.1.1. Topología vista desde *OFM*. Fuente: Propia.

Desde la pestaña de Hosts, se puede consultar información relevante sobre los dispositivos conectados a los switch de la topología.

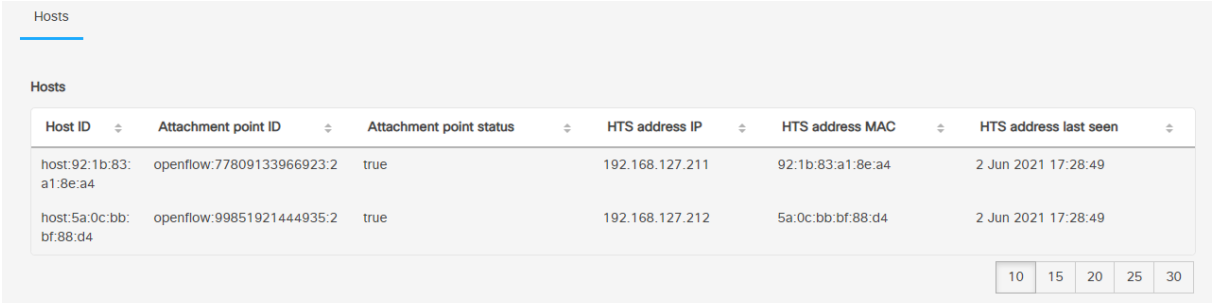


Fig. 10.3.1.2. Hosts vistos desde *OFM*. Fuente: Propia.

El apartado de Statistics contiene información muy parecida a la que proporcionan las herramientas mostradas previamente, no siendo esta pestaña la principal razón para la instalación de esta herramienta.

10.3.2. Flow Management

La pestaña de *flow management* del *OFM* es una gran herramienta que permite controlar los flujos de manera rápida y sencilla.

Están disponibles dos vistas al entrar a esta pestaña. En la primera, se muestra información básica sobre cada switch, como el tipo de dispositivo, versión del protocolo y el número de flujos configurados.

Fig. 10.3.2.1. Resumen de cada switch OVS en *OFM*. Fuente: Propia.

La siguiente vista permite ver información muy específica de cada flujo. A su vez, permite filtrar por diferentes parámetros, haciendo la búsqueda de determinados criterios más rápida.

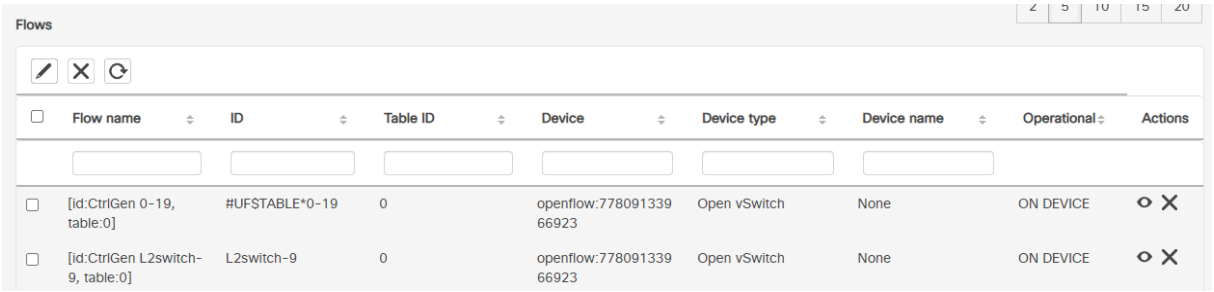


Fig. 10.3.2.2. Resumen de flujos en *OFM*. Fuente: Propia.

Una característica interesante, es que permite ver si el flujo está operacional o no, gracias a la última columna. También se puede ver información adicional de flujo, si se hace click sobre el icono en forma de ojo en la última columna. Presenta una interfaz muy intuitiva a la hora de ver la configuración del flujo, pudiendo también ver la mayoría de las opciones de configuración que presenta dicho flujo.

11. Programación de flujos

Como se ha visto hasta ahora, existen diferentes maneras de trabajar con herramientas *SDN*, como aquellas que ayudan a ver el estado actual de la topología. Una de las razones principales para ver el estado actual, es la necesidad de crear un nuevo estado, así, cambiando todo el comportamiento de la topología. De la misma manera, existen diferentes posibilidades a la hora de realizar una nueva configuración. En este tema se va a ver las mejores formas de configurar flujos para topologías *GNS3*, cada uno con sus ventajas e inconvenientes.

11.1. Programación desde el switch

La forma más sencilla de programar un flujo a las tablas de flujos de un switch, es mediante comandos, directamente en el mismo switch *OVS*. De la misma manera que con la configuración, los comandos para programar flujos varían entre un *OVS* y *Mininet*.

Para los ejemplos de este y futuros apartados, los flujos van a ser específicos para hacer una acción de *drop*, de todo paquete que vaya hasta un *host* en concreto. La forma básica para crear un flujo desde comando es la siguiente.

- ***OVS: ovs-ofctl add-flow br0 ip,nw_dst=192.168.0.1,actions=X***
- ***Mininet: sh ovs-ofctl add-flow s1 action=X -O OpenFlow13***

Los comandos en sí, son muy semejantes a los demás en estructura, la diferencia está en que se especifica que el comando es del tipo *add-flow*, y más adelante, las características del flujo. Una vez aplicado el flujo, se puede ver a este en la tabla de flujos.

```
/ # ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=13.245s, table=0, n_packets=0, n_bytes=0, idle_age=13, ip,nw_dst=192.168.127.206 actions=drop
```

Fig. 11.1.1. Flujos vistos desde switch *OVS* físico. Fuente: Propia.

En este caso, todos los paquetes que van a pasar por el *switch1*, no puedan llegar al *host2*, para ello la acción del flujo ha sido especificada como *drop*. Se puede verificar esto haciendo un ping desde *host1* a *host2*.

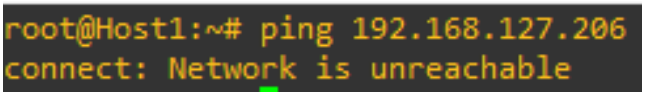


Fig. 11.1.2. Demostración de flujo en acción, acción *drop*. Fuente: Propia.

Ahora bien, si se hace ping desde *host2* hasta *host1*, van a funcionar los pings, aunque esto no queda del todo reflejado. Esto se debe a que el paquete sabe cómo ir, pero no como volver.

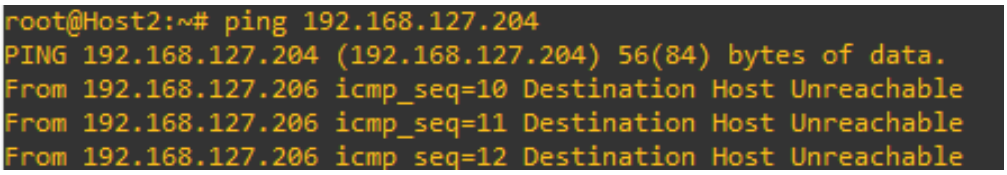


Fig. 11.1.3. Demostración de flujo en acción, acción drop. Fuente: Propia.

Para más información sobre otros posibles comandos para la programación de flujos en *OpenFlow*, consultar el *Anexo3*.

11.2. Programación desde DLUX

DLUX ofrece una interfaz más simple e intuitiva a la hora de programar flujos, comparado a la línea de comandos. También, permite guardar en una colección las configuraciones, así, permitiendo desplegar de una forma más rápida consultas guardadas previamente.

Una vez accedemos a *DLUX*, tanto si se utiliza el módulo de *Yang UI* como el de *Yangman*, se dispone de los mismos módulos de *ODL*. Tal y como se ha comentado en apartados anteriores, cada módulo dispone de un apartado *operational* y otro de *config*. En este caso, cada módulo que contiene el apartado de *config*, puede ser programado.

Se va a utilizar *Yangman* a la hora de programar estos flujos, ya que su interfaz ofrece mayor facilidad de uso y permite ver las características que pueden llevar estos flujos mediante seleccionables. Volviendo al módulo de *OpenDayLight-Inventory* desde *Yang UI*, se puede ver que esta interfaz se compone básicamente de la dirección con la que hacer un PUT o un POST, pero no ofrece la posibilidad de ver cada parámetro, tal y como se puede ver en la parte *operational* del mismo módulo de *Yang UI*.

Fig. 11.2.1. Programación de flujos desde *Yang UI*. Fuente: Propia.

Una vez en *Yangman*, y en el módulo correspondiente, se puede comenzar a configurar el nuevo flujo.

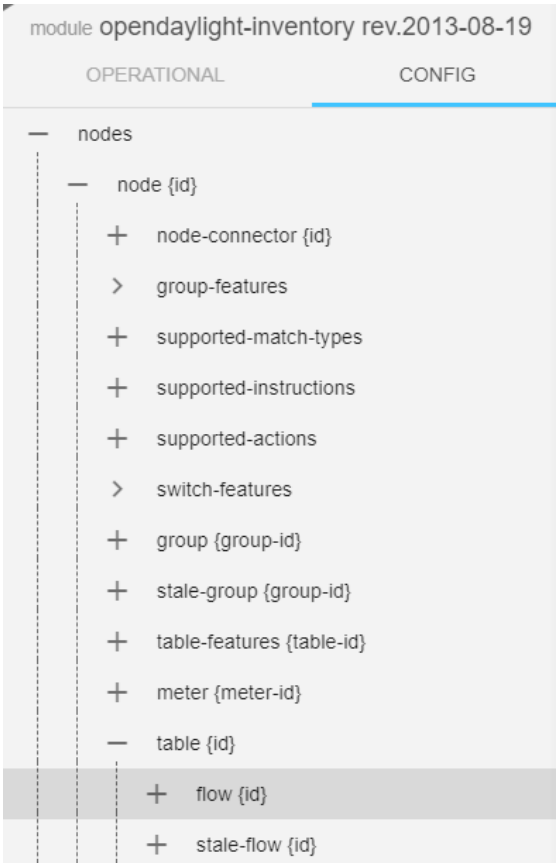


Fig. 11.2.2. Programación de flujos desde *Yangman*. Fuente: Propia.

Dentro de la parte de configuración, aparecen dos vistas, una por defecto para ver la configuración en formato *JSON*, y otra, en formato formulario. Se marca la petición como PUT o POST, y se rellena la dirección. Después de */node/*, hace especificar el id del switch, también llamado aquí como nodo. Después de */table/*, hay que especificar la tabla de flujos en la que va a ir el flujo que se está programando. Aunque las tablas de flujo comienzan por la tabla cero, el comportamiento no va a ser alterado si se le especifica una tabla de nivel superior, pero el orden en el que se ejecutan los flujos sí. Solo queda rellenar la parte posterior a */flow/*, en la que se

especifica el id del flujo. Cada una de estas etiquetas deben ser iguales aquí y en la parte de abajo, en el formulario. Si se rellena el formulario, es decir, se envía una petición a la API de *ODL* con un body en formato *JSON* o *XML*, en el que uno de los identificadores no coincide con los especificados en el header, el flujo va a dar error, o bien al momento de creación, o bien al ver el estado de este flujo.

Una de las principales funciones que tienen los flujos de *OpenFlow*, es la capacidad de hacer un match sobre un paquete, en base a criterios. Desde los formularios de *Yangman*, hay una gran variedad de opciones disponibles para elegir.

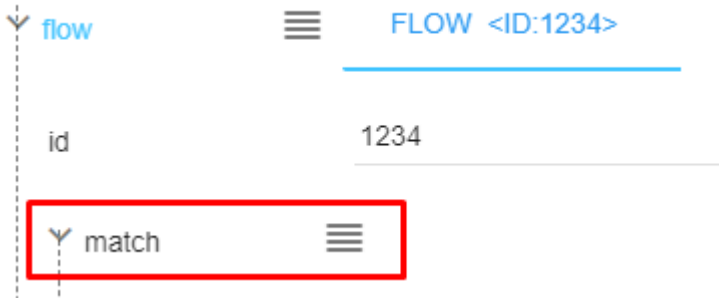


Fig. 11.2.3. Selección de opciones de match desde *Yangman*. Fuente: Propia.

Para continuar con el ejemplo de la línea de comandos en apartados anteriores, se va a hacer que el *switch1*, haga un drop de todos los paquetes que vayan del *host1* hacia el *host3*, basándose en la IP. Para especificar este tipo de match, hay que rellenar el apartado de *layer-3-match*. En el caso de un match mediante IP, es necesario especificar el *ethernet type*, el cual es el 2048 paraIPv4.[22]

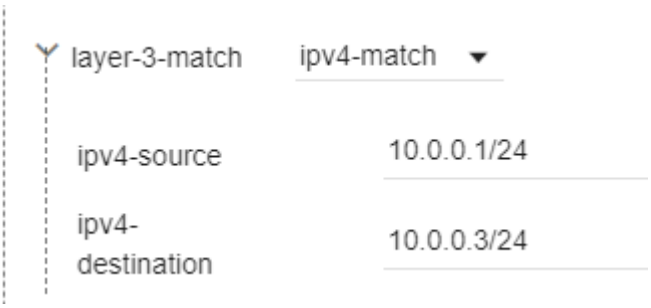


Fig. 11.2.4. Selección de opciones de match de IP desde *Yangman*. Fuente: Propia.

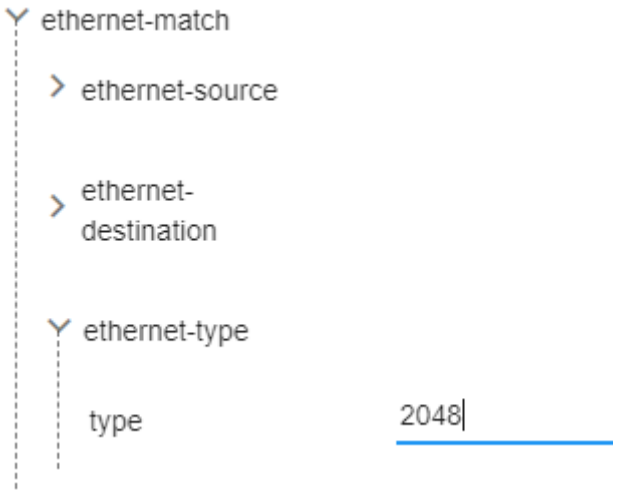


Fig. 11.2.5. Selección de opciones de match de IP y tipo ethernet desde *Yangman*. Fuente: Propia.

Más abajo en la lista, el siguiente apartado permite crear instrucciones para el flujo. Tal y como se ha explicado en el tema de *OpenFlow*, cada instrucción, puede contener una o más acciones. En este caso, solamente interesa hacer un drop. También se puede añadir instrucciones nuevas o acciones, si se le da al icono al lado del nombre de cada uno.

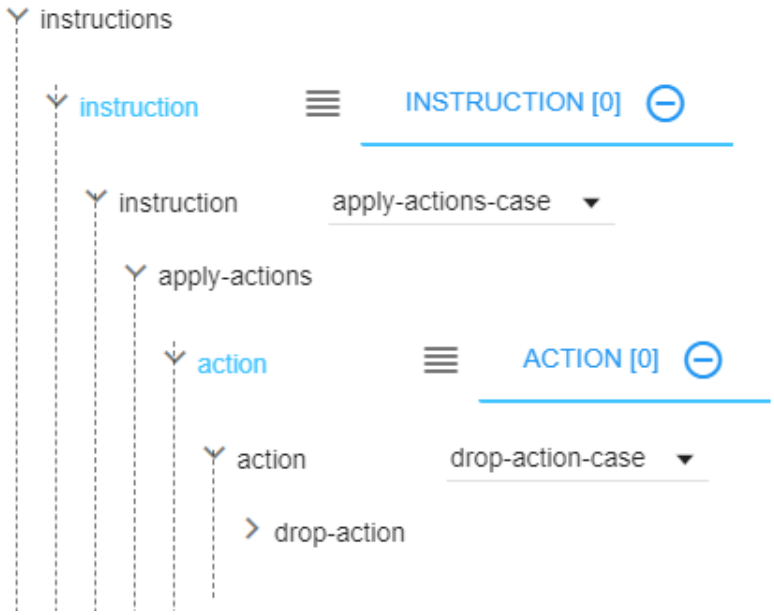


Fig. 11.2.6. Selección de acciones desde *Yangman*. Fuente: Propia.

Bajando en la lista, existe una gran cantidad de opciones que añadir, la mayoría de ellas opcionales. Entre los que faltan por rellenar, se encuentran el nombre del flujo, indicado por la etiqueta *flow-name*, el cual, también es opcional, aunque recomendable para posteriormente reconocerlo con mayor facilidad. Abajo del todo, se encuentran los campos de prioridad identificado como *priority* y el id de la tabla del flujo, con el nombre de *table_id*. El primero, aunque opcional, es altamente recomendable, ya que va a ser el primero en ser ejecutado por encima de los flujos guardados por defecto en el switch. El segundo de ellos, es obligatorio para el correcto funcionamiento del flujo.

Una vez finalizada la configuración, arriba a la derecha, está el botón de *save*, que permite guardar el flujo en nuestra colección, y el botón de *send*, el cual ejecuta el flujo. Al darle a *send*, va a llegar una respuesta, el llamado *status code* de *http*. Todo valor que sea 200-201, se considera una petición correcta. Esto quiere decir que el flujo ha sido programado con éxito.

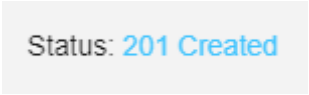


Fig. 11.2.7. Status de flujo creado en *Yangman*. Fuente: Propia.

Para comprobar el correcto funcionamiento del flujo, se hace un ping entre el *host1* y el *host3*.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
49 packets transmitted, 0 received, 100% packet loss, time 47999ms
```

Fig. 11.2.8. Flujo creado en *Yangman* en acción. Fuente: Propia.

Como se puede observar, los pings desde *host1* a *host3* no llegan. Ahora bien, si se hace un ping desde *host3* hasta *host1*, estos tampoco van a llegar, en realidad, ningún ping entre IPs 10.0.0.0 (como es el caso de todos los host) van a hacer ping entre sí, esta información la podemos comprobar desde el propio switch.

```
mininet> sh ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=3.782s, table=0, n_packets=0, n_bytes=0,
priority=1222,ip,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=drop
```

Según el resultado del comando de arriba, aunque se ha especificado una IP concreta, el controlador solo reconoce el rango de IPs completo. Una forma fácil de solucionar este problema, es mediante el uso de la MAC. De este modo, en vez de seleccionar el match por IP, se hace por MAC.

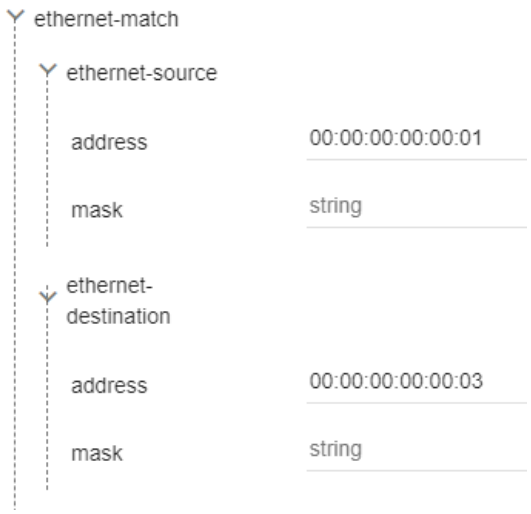


Fig. 11.2.9. Selecciones de opciones de match por MAC desde *Yangman*. Fuente: Propia.

De nuevo se comprueba hacer ping a diferentes host entre sí.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3025ms
```

Fig. 11.2.10. Demostración de flujo con acción drop desde *Yangman*. Fuente: Propia.

```
mininet> h3 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=13 Destination Host Unreachable
From 10.0.0.3 icmp_seq=14 Destination Host Unreachable
From 10.0.0.3 icmp_seq=15 Destination Host Unreachable
^C
--- 10.0.0.1 ping statistics ---
17 packets transmitted, 0 received, +3 errors, 100% packet loss, time 15998ms
```

Fig. 11.2.11. Demostración de flujo con acción de drop desde *Yangman*. Fuente: Propia.

Ahora, se puede ver que el *host3* en realidad llega al *host1*, pero, tal y como no puede volver, estos paquetes no llegan. Como muestra Fig. 11.2.12, al llegar al *switch1*, el flujo se activa, inhabilitando todo el tráfico saliente de *host1*, independientemente si la petición ha comenzado en el *host3*.

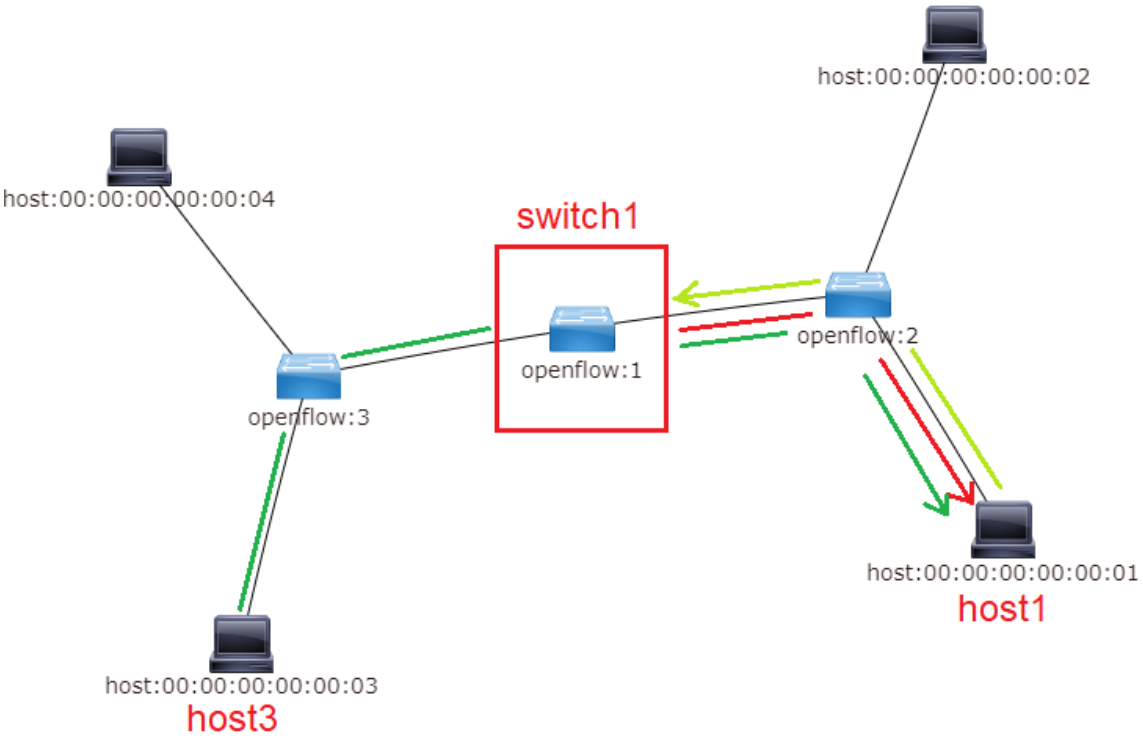


Fig. 11.2.12. Diagrama del flujo de paquetes en la topología. Fuente: Propia.

11.3. Programación desde OFM

OFM ofrece una interfaz muy útil a la hora de gestionar flujos. De la misma manera que con *DLUX*, es posible la creación de flujos mediante una interfaz sencilla. También, se puede ver rápidamente toda la información básica necesaria para ello.

Una vez dentro de *OFM*, se le da al botón de editar, identificado por un lápiz.

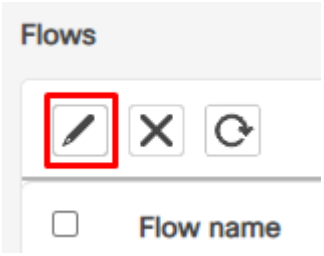


Fig. 11.3.1. Crear nuevo flujo desde *OFM*. Fuente: Propia.

La siguiente pantalla lo primero que va a pedir es seleccionar el switch de la topología a programar. Una vez seleccionado, aparecen tres filas para rellenar, el id de la tabla, el id del flujo y su prioridad. Todas las demás opciones, aparecen como seleccionables en la columna de la izquierda. En el apartado de *General Properties* se marca también la propiedad *Flow Name*. En el siguiente apartado, permite seleccionar el tipo de match que se le va a especificar al flujo. Siguiendo los mismos pasos que en los apartados anteriores, se va a hacer que el match sea por MAC, pero para hacerlo por IP, basta con seleccionar los mismos criterios de match que previamente. El último paso es seleccionar el tipo de acción que va a realizar el flujos, en este caso va a ser *drop*.

Al igual que en *DLUX*, se dispone de varios botones útiles, el primero *Show preview*, permite ver el body de la petición, esto puede ayudar a la hora de hacer peticiones de creación de flujos a mano, algo que se va a ver en el siguiente apartado. El otro botón es *Send request*, el cual permite crear el mismo flujo.

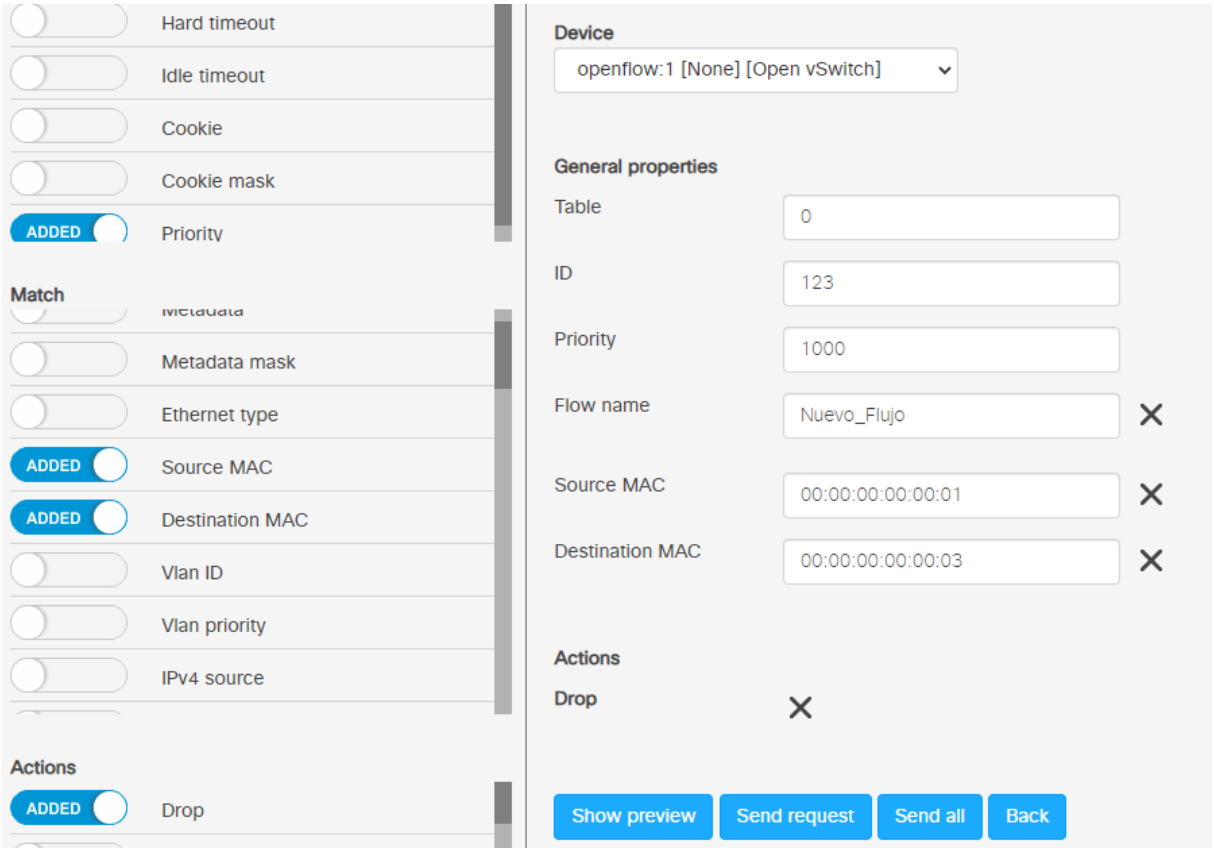


Fig. 11.3.2. Configuración básica de nuevo flujo en *OFM*. Fuente: Propia.

Una vez enviada la petición, arriba de todo sale una notificación, informando si el flujo se ha programado bien o no. A pesar del mensaje, esto no siempre quiere decir que el flujo es correcto, sino que la petición de crear el flujo ha sido correcta, pese a esto, el flujo puede no funcionar correctamente, por lo que siempre hay que comprobarlo.

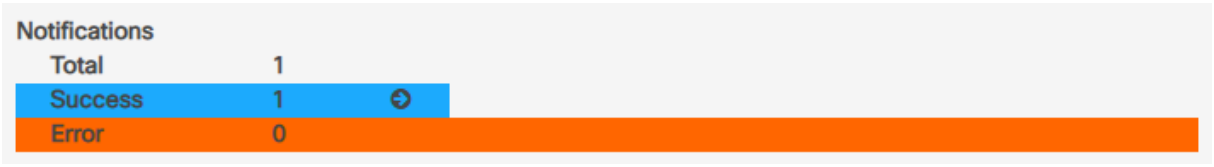


Fig. 11.3.3. Configuración básica de nuevo flujo correcta en *OFM*. Fuente: Propia.

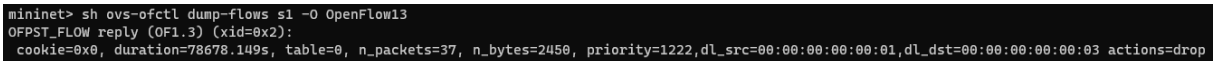


Fig. 11.3.4. Comprobación de nuevo flujo en switch programado en *OFM*. Fuente: Propia.

Según indica la tabla de flujos del *switch1*, el flujo está correctamente registrado en el switch, si se comprueba esto haciendo ping desde *host1* a *host3*, se ve que el flujo está correcto. Por otra parte, en el caso de hacer un ping de *host2* a *host4*, por ejemplo, el flujo no debe interferir.

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 1998ms
```

Fig. 11.3.5. Comprobación de nuevo flujo en acción, programado en *OFM*. Fuente: Propia.

```
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.251 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.174 ms
^C
--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
```

Fig. 11.3.6. Ping entre hosts sin match en flujo, programado en *OFM*. Fuente: Propia.

La ventaja más importante de *OFM* es su rápida visualización de flujos. Al instante de incorporar un nuevo flujo, se pueden ver los cambios aplicados. Es también capaz de reconocer si un flujo está registrado en el switch o no, es decir, el flujo ha sido mal programado, o por algún error entre la conexión con el controlador con el switch, este no ha podido ser registrado, por ello solo se encuentra registrado en el controlador, pero no en el switch. Este tipo de información, no puede ser vista directamente desde el switch, y tampoco desde *DLUX*.

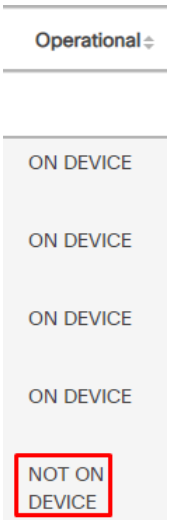


Fig. 11.3.7. Estado de flujo reflejado en *OFM*. Fuente: Propia.

11.4. Automatización desde Postman

Postman es una herramienta para realizar peticiones a APIs. Permite guardar colecciones de peticiones, crear nuevas peticiones definiendo parámetros con una alta granularidad, y, analizar los resultados mediante todas las herramientas que ofrece.

Una forma práctica de utilizar *Postman*, es mediante su uso en conjunto a *DLUX* y *OFM*. Estas dos aplicaciones ofrecen mucha información de manera sencilla, desde *DLUX*, se puede ver la IP que va a tener la petición, también, desde *Yangman* o el mismo *OFM* es posible configurar un flujo desde la interfaz, y pegarlo como body de la petición en *Postman*. Los beneficios que ofrece respecto a las otras opciones mostradas anteriormente, es que no requiere de una interfaz de usuario, no requiere contacto directo con un controlador específico. Gracias a esto, se puede realizar peticiones ya guardadas en diferentes colecciones, además, fáciles de cambiar.

Una de las principales desventajas de las opciones anteriores, es que requieren cada uno su propia interfaz, esto quiere decir que es necesario tener al menos una ventana abierta por controlador, en cambio, con *Postman* podemos trabajar con un número ilimitado de controladores desde la misma interfaz. También, permite tener una granularidad de parámetros superior, cabe mencionar que con *Postman*, es posible adaptarse a cualquier tipo de peticiones,

con tal de que el controlador funcione mediante APIs. Esto quiere decir que no obliga a utilizar un formato predeterminado, como por ejemplo *Yangman* con *JSON*, si no que acepta cualquier formato moderno de programación de APIs.

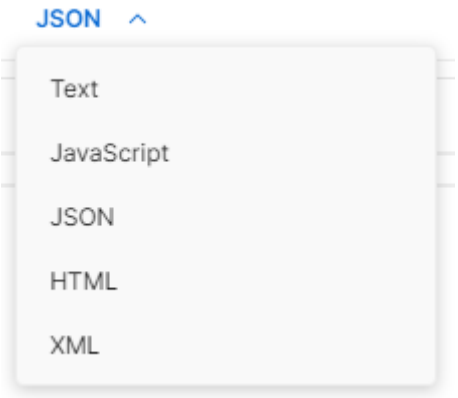


Fig. 11.4.1. Diferentes tipos de *body* disponibles para una petición en *Postman*. Fuente: Propia.

Continuando con el ejemplo de apartados anteriores, se configura desde *Postman* el mismo flujo de match por MAC, con la acción de drop.

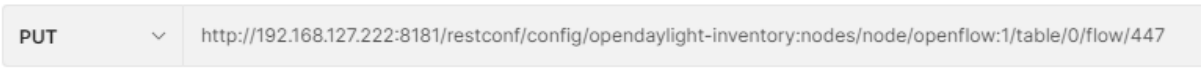


Fig. 11.4.2. Petición tipo *config* en *Postman*. Fuente: Propia.

Desde la barra de navegación de arriba, se introduce la cabecera de la petición. Una vez más, *Yangman* o *Yang UI* aquí pueden ser de gran ayuda. Cada uno de los paquetes disponibles en *ODL*, tienen su propia dirección, por ello, navegando por estos módulos se puede copiar las peticiones que más interesan. De nuevo se va a recurrir al módulo de *opendaylight-inventory* para la configuración del flujo. De la misma manera que se hace login desde *DLUX*, todas estas peticiones que se hacen al controlador, van a requerir de autorización. Para ello, desde el apartado de *Authorization*, se marca *Basic Auth* con las mismas credenciales.

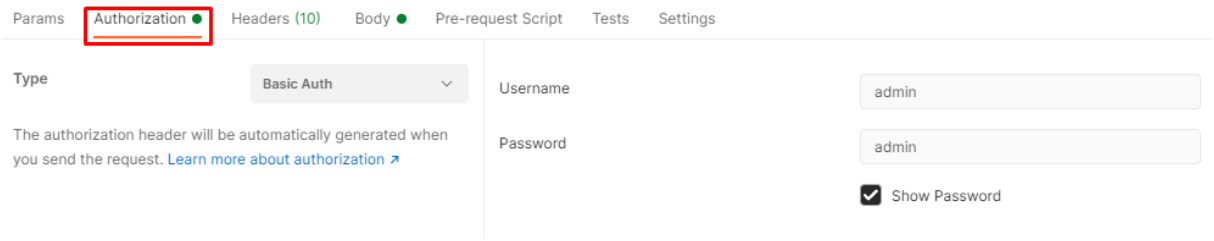


Fig. 11.4.3. Autorización en *Postman*. Fuente: Propia.

A Partir de aquí, solo falta rellenar el body, para ello, también se puede utilizar la ayuda de *Yangman* o *OFM*. En este caso, va a ser un body *raw* en formato *JSON*. Más ejemplos de peticiones OpenFlow mostradas en el *Anexo4*.

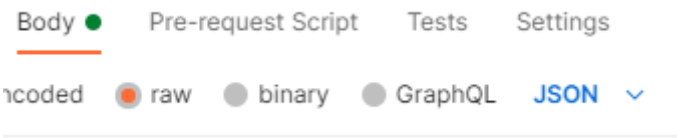


Fig. 11.4.4. Selección de *body* en *Postman*. Fuente: Propia.

Si se utilizan en conjunto todas las herramientas, se puede ver en *Postman* la misma efectividad de creación de flujos que en las demás herramientas. Desde el mismo *OFM*, se ve que el flujo ha sido añadido correctamente. La misma información es verificable desde el mismo switch.

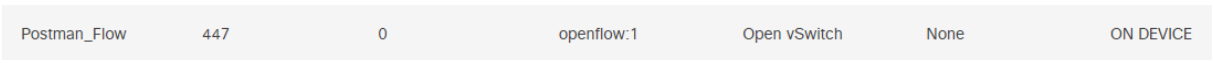


Fig. 11.4.5. Nuevo flujo creado desde *Postman*. Fuente: Propia.

```
mininet> sh ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x3, duration=17.525s, table=0, n_packets=0, n_bytes=0,
priority=2000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=drop
 cookie=0x2b00000000000007, duration=87893.545s, table=0, n_packets=80,
n_bytes=5600, priority=2,in_port=1 actions=output:2
 cookie=0x2b00000000000006, duration=87893.545s, table=0, n_packets=87,
n_bytes=6174, priority=2,in_port=2 actions=output:1
 cookie=0x2b00000000000001, duration=87897.463s, table=0,
n_packets=35172, n_bytes=2989620, priority=100,dl_type=0x88cc
actions=CONTROLLER:65535
 cookie=0x2b00000000000000, duration=87897.463s, table=0, n_packets=0,
n_bytes=0, priority=0 actions=drop
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4032ms
```

Como se puede observar, el primer flujo de todos, es el programado desde *Postman*. Haciendo ping entre los dispositivos, se puede observar que el flujo funciona correctamente.

12. Conclusiones

En este trabajo se han visto todas las posibilidades que ofrecen las herramientas de gestión de las redes *SDN*.

Comenzando por los apartados teóricos, se ha visto las diferencias que existen entre estas nuevas tecnologías y las tradicionales. Grandes avances se han ido haciendo en los últimos veinte años, y algunos de ellos han ayudado el desarrollo de las tecnologías presentadas en este trabajo. *OpenFlow* incorpora ideas ya planteadas previamente a este, como la separación de los planos, o el control de la granularidad de los paquetes, como en el caso del *PBR*. Con esto, se quiere decir que se espera que este protocolo siga evolucionando, probablemente nutriéndose de otras tecnologías que vayan a salir por el camino, ya que, por el momento, *OpenFlow* intenta solucionar todos los problemas existentes en las redes, en un solo paquete, hasta el punto en el que el rendimiento de este, a la hora de escalar la red, se ve gravemente afectado.

Por la parte de *OpenDayLight*, pese a utilizar *Java* para su desarrollo, lo que no favorece su rendimiento, por otro lado, favorece la distribución y la facilidad de desarrollo de las aplicaciones para este. Aunque no es el único controlador disponible hoy en día, parece tener un camino más seguro de cara a futuro. Su gran compatibilidad con la mayoría de los protocolos de red, y el echo de ser open source, hacen que sea una gran opción para la gestión de las redes *SDN*. La única parte negativa encontrada, al menos de cara a usuarios convencionales, es que, con el tiempo, comienza a perder soporte de módulos más básicos, centrándose más en requerimientos de grandes organizaciones.

Por último, el conjunto de todas las herramientas utilizadas en el trabajo, permiten crear y gestionar redes, con una granularidad y practicidad antes no vista, al menos con herramientas open source. Por otro lado, el coste necesario para que estas topologías funcionen de manera automatizada, y ahorren tiempo a la hora de producir cambios, es muy elevado. El grado de complejidad e infraestructura necesarios para el correcto funcionamiento, es muy elevado comparado con las redes tradicionales, por lo tanto, es comprensible el poco éxito que están teniendo las redes *SDN* a la hora de sustituir las redes tradicionales de tamaño pequeño. Este gran coste, por otro lado, si que es justificable en redes de gran tamaño, donde un pequeño cambio de decisión, conlleva una cantidad de trabajo rápidamente salvable con la utilización de estas nuevas tecnologías.

13. Ampliaciones

Las ampliaciones posibles para este trabajo pueden ir en varias direcciones diferentes.

Por una parte, existe la posibilidad de que las versiones más modernas de *OpenDayLight* ofrezcan módulos, que, pese a no replicar el comportamiento de los utilizados en el trabajo, permitan aun así realizar topologías programables, con una interfaz diferente, o bien, sin ningún tipo de interfaz, simplemente con peticiones básicas manuales. Existe la posibilidad también de que, al llegar futuras versiones de *ODL* y *OpenFlow*, estas venir con módulos, para una fácil gestión de las redes desde *GNS3*.

En el trabajo, no se ha puesto una énfasis muy grande a la hora de montar topologías muy complejas, añade complejidad extra a la hora de entender las funcionalidades básicas, pero a partir de estas, es relativamente simple ir añadiendo. Una de las ampliaciones posibles, manteniendo la misma estructura del trabajo, es la de montar topologías grandes, separándolas en *VLANs*. Esto permite replicar las topologías de la vida real, donde cada organización y zonas geográficas cuentan con sus propias IPs.

Otra posible ampliación recae en el uso de controladores diferentes. Además de *OpenDayLight* y *OpenFlow*, existen otras alternativas en cuanto a gestión en la red, como por ejemplo *OpenStack* o *Neutron*.

Por último, *GNS3* también permite conectar dispositivos reales a las topologías, y por ello, otra posible ampliación puede hacer uso de dispositivos reales, sean un host, un switch o el mismo controlador, para llevar las topologías fuera de los entornos de emulación.

14. Bibliografía

[1] Manar Jammala, Taranpreet Singha, Abdallah Shamia, Rasool Asalb, Yiming Lic: Software-Defined Networking: State of the Art and Research Challenges (https://arxiv.org/ftp/arxiv/papers/1406/1406.0124.pdf)

[2] Nick Feamster, Jennifer Rexford, Ellen Zegura: The Road to SDN: An Intellectual History of Programmable Networks (http://www.sigcomm.org/sites/default/files/ccr/papers/2014/April/0000000-0000012.pdf)

[3] Paul Goransson; Chuck Black; Timothy Culver: Software Defined Networks: A Comprehensive Approach

[4] SDN Protocols (https://www.dclessons.com/sdn/sdn-protocols)

[5] Scott S. Lowe, Matt Oswalt, Jason Edelman: Network Programmability and Automation: Skills for the Next-Generation Network Engineer

[6]OpenFlow Switch Specification (https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf)

[7] Zhang Fu: An Introduction to Software-Defined Networking (http://www.cse.chalmers.se/edu/year/2018/course/EDA344DIT423/SLIDESNOTES/13.SDN-lecture_ZF.pdf)

[8] OpenFlow Switch Specification: Version 1.5.1 (https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf)

[9] OVS Actions - (https://man7.org/linux/man-pages/man7/ovs-actions.7.html)

[10] OpenDayLight Project – (https://en.wikipedia.org/wiki/OpenDaylight_Project)

[11] OpenDaylight Project: OpenDaylight Controller (https://docs.opendaylight.org/_/downloads/controller/en/v2.0.3/pdf/)

[12] IETF - Network Configuration Protocol (NETCONF) (https://www.rfc-editor.org/rfc/pdfrfc/rfc6241.txt.pdf)

[13] NETCONF – (https://es.wikipedia.org/wiki/NETCONF)

[14] A. Bierman, YumaWorks, M. Bjorklund, Tail-f Systems, K. Watsen, Juniper Networks - RESTCONF Protocol (https://www.rfc-editor.org/rfc/pdfrfc/rfc8040.txt.pdf)

[15] RESTCONF Tutorial – (https://ultracon-Fig.com.au/blog/restconf-tutorial-everything-you-need-to-know-about-restconf-in-2020/)

[16] OpenDaylight Project – MD-SAL (https://docs.opendaylight.org/_/downloads/mdsal/en/latest/pdf/)

[17] OpenDayLight Controller – (https://docs.opendaylight.org/projects/controller/en/latest/_images/Get.png)

[18] David Bombal – GNS3 Talks: Ubuntu Docker Container, OpenDaylight, Python, SDN

(https://www.youtube.com/watch?v=rrja9gJjkvQ)

[19] John Sobanski - How to install OpenDaylight as a Service on Ubuntu 18.04 LTS (https://john.soban.ski/how-to-install-opendaylight-as-a-service-on-ubuntu.html)

[20] CisvoDevNet – OpenDaylight Openflow App (https://github.com/CiscoDevNet/OpenDaylight-Openflow-App)

[21] Mininet – (https://github.com/mininet/mininet)

[22] OpenDayLight – OpenFlow Plugin (https://docs.opendaylight.org/projects/openflowplugin/en/latest/users/operation.html)