Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

ESTUDIO DE LAS REDES DEFINIDAS POR SOFTWARE MEDIANTE EL DESARROLLO DE ESCENARIOS VIRTUALES BASADOS EN EL CONTROLADOR OPENDAYLIGHT

TRABAJO FIN DE MÁSTER

Raúl Álvarez Pinilla

2015

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

ESTUDIO DE LAS REDES DEFINIDAS POR SOFTWARE MEDIANTE EL DESARROLLO DE ESCENARIOS VIRTUALES BASADOS EN EL CONTROLADOR OPENDAYLIGHT

Autor
Raúl Álvarez Pinilla

Director
David Fernández Cambronero

Departamento de Ingeniería de Sistemas Telemáticos

2015

Resumen

El surgimiento de nuevas tendencias tecnológicas, tal como la llegada de servicios en la nube y el gran aumento de dispositivos conectados a la red, repercuten en las redes de comunicación mediante la demanda de mayores requisitos. Asimismo, hasta ahora los protocolos utilizados en las redes han sido definidos de forma independiente, originando con ello una falta de flexibilidad, y en ciertos casos la utilización de protocolos propietarios ha limitado la infraestructura de red a equipos de un mismo fabricante, creando así una fuerte dependencia con el vendedor. En consecuencia ha surgido el enfoque conocido como Redes Definidas por Software, o por sus siglas SDN (Software Defined Networking), que proporciona una arquitectura de red flexible y escalable para adaptarse a las necesidades, y así poder reaccionar de forma dinámica a los recursos demandados por las aplicaciones.

Las Redes Definidas por Software proponen el desacoplamiento del plano de control respecto del plano de datos, y de esta forma centralizar la dirección de la red en un controlador SDN, que se comunicará con cada uno de los dispositivos que componen la red. Esta abstracción permite al controlador tener una visión general de la topología de la red y realizar un reenvío óptimo del tráfico según las exigencias de cada momento.

Dada la amplia variedad de controladores para abarcar las SDN, el presente trabajo se ha centrado sobre el proyecto colaborativo basado en código abierto OpenDaylight. Se encarga de la construcción de una completa plataforma SDN basada en diversos estándares que permite ser desplegada en una amplia variedad de entornos. De esta manera pretende acelerar el desarrollo y la adopción de las Redes Definidas por Software, y convertirse en un referente en este ámbito.

Por ello, este trabajo consiste en el análisis de la arquitectura que proponen las Redes Definidas por Software a través del desarrollo y prueba de escenarios virtuales que muestren casos de uso relevantes basados en el controlador OpenDaylight. Esto incluye como objetivos el desarrollo de aplicaciones que interactúen con la plataforma para la gestión de la red, y la integración en un entorno de computación en la nube basado en el proyecto OpenStack.

Palabras clave: Redes Definidas por Software, SDN, OpenDaylight, OpenStack, OpenFlow, VNX, Mininet, DevStack.

Abstract

The appearance of new technology trends, such as the arrival of cloud services and the increase of connected devices, affect communication networks by demanding higher requirements. Furthermore, so far the protocols used on networks are defined independently, thereby causing a lack of flexibility, and in some cases the use of proprietary protocols has limited network infrastructure equipment from the same manufacturer, creating in this way a strong dependence with the vendor. Consequently, there has emerged the approach known as Software Defined Networking, or by its acronym SDN, that provides a flexible and scalable architecture to adapt to the network needs, and thus the network can dynamically react to the demanded resources by applications.

Software Defined Networking proposed the decoupling of control plane from the data plane, and thus the network management is centralized in an SDN controller that will communicate with each of the network devices. This abstraction allows the controller to have an overview of the network topology and make an optimal traffic forwarding to the requests of each moment.

Given the wide variety of controllers to cover the SDN, this project has focused on the collaborative and open source OpenDaylight project. It is responsible for building a complete SDN platform based on several standards that allows it to be deployed in a wide variety of environments. Thus, it aims to accelerate the development and adoption of Software Defined Networking and become a benchmark.

Therefore, this project consists of analyzing the architecture proposed by the Software Defined Networking paradigm through the development and testing of virtual scenarios that show relevant use cases based on the OpenDaylight controller. It includes as objectives the development of applications that interact with the platform for network management and the integration into a cloud computing environment based on the OpenStack project.

Key words: Software Defined Networking, SDN, OpenDaylight, OpenStack, OpenFlow, VNX, Mininet, DevStack.

Índice de contenidos

Índice de figuras

Siglas

AD-SAL	API-Driven Service Abstraction Layer
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
BYOD	Bring Your Own Device
COW	Copy-On-Write
DHCP	Dynamic Host Configuration Protocol
DLUX	OpenDaylight User Experience
DNAT	Destination Network Address Translation
DNS	Domain Name System
DVR	Distributed Virtual Routing
EPL	Eclipse Public License
FSF	Free Software Foundation
GRE	Generic Routing Encapsulation
IaaS	Infrastructure as a Service
IANA	Internet Assigned Numbers Authority
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KVM	Kernel-based Virtual Machine
LLDP	Link Layer Discovery Protocol
LXC	Linux Containers
MAC	Media Access Control
MD-SAL	Model-Driven Service Abstraction Layer
ML2	Modular Layer 2
NAT	Network Address Translation
NBI	Northbound Interface

NFV	Network Functions Virtualization
ODL	OpenDaylight
OF	OpenFlow
ONF	Open Networking Foundation
OPNFV	Open Platform for NFV
OSGi	Open Services Gateway Initiative
OSI	Open System Interconnection
OSI	Open Source Initiative
OSPF	Open Shortest Path First
OVS	Open vSwitch
OVSDB	Open vSwitch Database Management Protocol
OXM	OpenFlow Extensible Match
PBB	Provider Backbone Bridging
PCEP	Path Computation Element Protocol
REST	Representational State Transfer
RPC	Remote Procedure Call
SAL	Service Abstraction Layer
SBI	Southbound Interface
SDK	Software Developer Kit
SDN	Software Defined Networking
SNAT	Source Network Address Translation
SOA	Service Oriented Architecture
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to live
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VNX	Virtual Networks over linuX
VPN	Virtual Private Network
VXLAN	Virtual eXtensible Local Area Network
XML	eXtensible Markup Language

1 Introducción

En el día de hoy están surgiendo nuevas tendencias que repercuten en las redes de comunicación mediante la demanda de mayores requisitos. La llegada de servicios en la nube y el gran aumento de dispositivos conectados a la red son algunos ejemplos que plantean a la industria modificar la arquitectura de las redes de comunicación. Como resultado, se está produciendo un mayor nivel de tráfico que las redes deben ser capaces de responder, en tiempo y forma, por las demandas de los consumidores para no perder valor en el mercado.

Las aplicaciones que antes se alojaban en una única máquina física y proporcionaban un servicio a una serie de clientes han sufrido una migración hacia entornos de computación en la nube, y ahora se pueden encontrar distribuidas en diferentes máquinas virtuales con un gran intercambio de información entre ellas. Con ello, el patrón de comportamiento del tráfico en los centros de datos se ha visto modificado, y las aplicaciones cliente-servidor han evolucionado su arquitectura necesitando comunicación con equipos que disponen de otros recursos, como bases de datos y servidores. Esta tendencia proporciona un importante dinamismo que mejora, entre otras cosas, la disponibilidad y el consumo de recursos. Sin embargo, en entornos de estas características no se puede considerar una infraestructura de red estática, ya que no abastecería de forma óptima las necesidades que se precisaran en cada momento.

Los usuarios de todo el mundo están incrementando el empleo de dispositivos móviles, tales como smartphones o tablets, y es habitual ver a personas conectadas con varios de estos dispositivos simultáneamente. Además, en entornos corporativos el personal IT se encuentra el desafío de permitir el acceso de tales dispositivos personales con la protección de la información y propiedad intelectual de la empresa, lo que se conoce como el movimiento Bring Your Own Device (BYOD). De igual manera, el Internet de las Cosas (IoT), que se basa en la interconexión digital de objetos cotidianos con Internet pretendiendo crear un mundo en el que las personas y dispositivos interactúen entre sí, también incrementa el número de dispositivos conectados a Internet, y con ello la complejidad en la gestión de todas sus conexiones.

Además, tendencias como Big Data que pretenden manejar grandes conjuntos de datos para su tratamiento solicitan una constante demanda en la capacidad de red. Esto ocasiona que los operadores se enfrenten a escalar sus redes a tamaños inimaginables hasta ahora, manteniendo que ninguna conexión de la red se descuide.

De todas las tendencias mencionadas se origina un importante crecimiento en el tráfico de datos en las redes de comunicación. Cisco, a través de su informe Visual Networking Index [1], predice diferentes datos sobre las redes de comunicación. Su última publicación, en mayo de 2015, trata la evolución prevista desde el año 2014 hasta 2019. En él se prevé que en el año 2019 haya 3.9 millardos de usuarios en Internet (más del 51% de la población mundial), con una subida de 2.8 millardos respecto al año 2014. Si se tiene en cuenta el número de dispositivos conectados, se prevé que en 2019 haya 24 millardos de dispositivos conectados globalmente, lo que supone 3 dispositivos por persona en el mundo. Además, la velocidad de los accesos de banda ancha a Internet se verá mejorada hasta una velocidad media de 42.5 Mbps. En cuanto al tráfico IP global, se transportarán de media 5.5 exabytes por día, del cual un 80% se corresponderá a tráfico de video. Dados estos datos, se necesita mejorar la arquitectura de las redes para hacer frente al futuro que se espera.

Figura 1.1. Evolución global de las redes de comunicación [1]

Asimismo, hasta ahora los protocolos utilizados en las redes han sido definidos de forma independiente unos de otros, ya que cada uno de ellos intenta resolver una problemática en particular. Debido a este desarrollo, en el momento de interaccionar varios protocolos, se origina una importante falta de flexibilidad que repercute negativamente en las redes de comunicación. Además hay que añadir en ciertos casos la utilización protocolos propietarios, que limitan la infraestructura de la red a equipos del mismo fabricante, ya que no se permite la interacción con otros dispositivos de terceros, y crea una importante dependencia con el vendedor.

Dada esta situación ha surgido un enfoque que pretende mostrar una nueva arquitectura para las redes actuales que resuelve inconvenientes que poseen las redes tradicionales. Este enfoque conocido como Redes Definidas por Software, o por sus siglas SDN (Software Defined Networking), proporciona una arquitectura de red flexible y escalable que permite adaptarse a las necesidades, y así poder reaccionar de forma dinámica a los recursos demandados por las aplicaciones. Como resultado, en entornos de red tan complejos como los existentes en centros de datos, proporciona una simplificación de la gestión de la red, y por el consiguiente una mejor utilización de la infraestructura de la que se dispone.

1.1 Motivación

La elección del tema del presente trabajo se atribuye al importante crecimiento que está experimentando el ecosistema alrededor de las Redes Definidas por Software. Se ha convertido en una tendencia en el campo de la investigación, y dado su futuro prometedor desempeñarán un importe papel en el modo en que se construyan las redes.

Con dicho tópico surge la oportunidad de aplicar los conocimientos relacionados con el campo de las SDN, que se adquieran a lo largo del trabajo, en escenarios virtuales basados en el controlador OpenDaylight.

Asimismo, la amplia relevancia de las Redes Definidas por Software permite trabajar en escenarios utilizados hoy en día por proveedores de servicios, como son entornos de computación en la nube.

1.2 Objetivos

El propósito general de la investigación es profundizar en el conocimiento de las Redes Definidas por Software analizando soluciones existentes actualmente que pretenden incrementar su adopción sobre las redes tradicionales. Para ello se llevará a cabo un estudio de estándares y de diversos proyectos relacionados, que tras una valoración de ellos se orientará el trabajo en una plataforma específica, OpenDaylight.

La comprensión de diversos casos de uso de las Redes Definidas por Software se realizará mediante el desarrollo de escenarios controlados por la plataforma OpenDaylight. Permitirá evaluar el desarrollo de este enfoque que está en camino de convertirse en una nueva realidad para las redes de comunicación.

Los objetivos técnicos del proyecto consisten en la creación de escenarios virtuales que permitan entender la arquitectura SDN, así como sus ventajas, a través de las funcionalidades que proporciona OpenDaylight. Esto incluye el desarrollo de aplicaciones que interactúen con la plataforma para la gestión de la red y la integración en un entorno de computación en la nube basado en el proyecto OpenStack.

1.3 Estructura de la memoria

El trabajo está compuesto por un total de seis capítulos y dos apéndices, que muestran los siguientes contenidos:

- En el *Capítulo 1: Introducción* se muestra el contexto de la investigación y los principales objetivos a conseguir.
- En el *Capítulo 2: Redes Definidas por Software* se presenta el enfoque SDN sobre las redes de comunicación y se incluye información sobre el estándar OpenFlow. Además se realiza un estudio comparativo entre diferentes controladores, tanto comerciales como de código abierto, que existen en el mercado.
- En el *Capítulo 3: OpenDaylight* se analiza tal proyecto, así como la visión técnica, la arquitectura software y las diferentes versiones de la plataforma.
- En el *Capítulo 4: Escenarios de demostración* se aplican los conocimientos adquiridos sobre escenarios virtuales desarrollados controlados por OpenDaylight.
- En el *Capítulo 5: Resultados y conclusiones* se ofrece una visión final sobre el trabajo realizado y posibles líneas de trabajo futuro.
- En la sección de *Apéndices*, que se adjunta al final del documento, se proporcionan los archivos utilizados en los escenarios de demostración creados para el trabajo.

Se ha utilizado un código de colores en los diferentes cuadros que aparecen a lo largo de la memoria, donde un relleno anaranjado muestra el contenido total o parcial de un fichero específico, y un relleno de color gris claro indica comandos ejecutados en un terminal, así como la salida por pantalla de los mismos.

```
CONTENIDO DE FICHEROS
```

```
COMANDOS EN UN TERMINAL
SALIDA POR PANTALLA
```

2 Redes Definidas por Software

A lo largo de este capítulo se pretende analizar la arquitectura que adoptan las Redes Definidas por Software, así como el estándar OpenFlow que es ampliamente reconocido en el entorno SDN. La evolución que está experimentando este campo ha originado que se disponga de un gran número de soluciones, tanto comerciales como de código abierto, que se detallará más adelante.

2.1 Arquitectura

El enfoque que plantean las Redes Definidas por Software se fundamenta en desacoplar el plano de control de la red del plano de datos. De esta forma es posible la optimización del funcionamiento de la red creando una red programable al abstraer el control a aplicaciones.

En las redes tradicionales, tanto el plano de control como el plano de datos se encuentran situados en el propio dispositivo de red, y en este caso es el plano de control de cada nodo el responsable de conocer que acción tomar al recibir un cierto flujo de tráfico. El plano de control se puede encargar del procesamiento de protocolos de enrutamiento, como pueden ser OSPF y BGP, y debe realizar una gran cantidad de tareas para determinar el camino hacia donde tienen que reenviar los paquetes para llegar a un cierto destino. A esto hay que añadir otras exigencias que se han ido añadiendo con el paso del tiempo, como puede ser la calidad de servicio, que incrementan la complejidad en los dispositivos de una infraestructura de red.

Con la separación que proponen las Redes Definidas por Software, el plano de control se centraliza en un controlador SDN que dispone comunicación con el plano de datos de cada uno de los dispositivos que componen la red. Esta abstracción permite al controlador tener una visión general de la topología de la red y realizar un reenvío óptimo del tráfico según las necesidades de cada momento.

En la figura 2.1 se muestra una comparativa entre las diferentes arquitecturas, donde se aprecia la migración del plano de control y el surgimiento de un agente de control en cada uno de los dispositivos de la red.

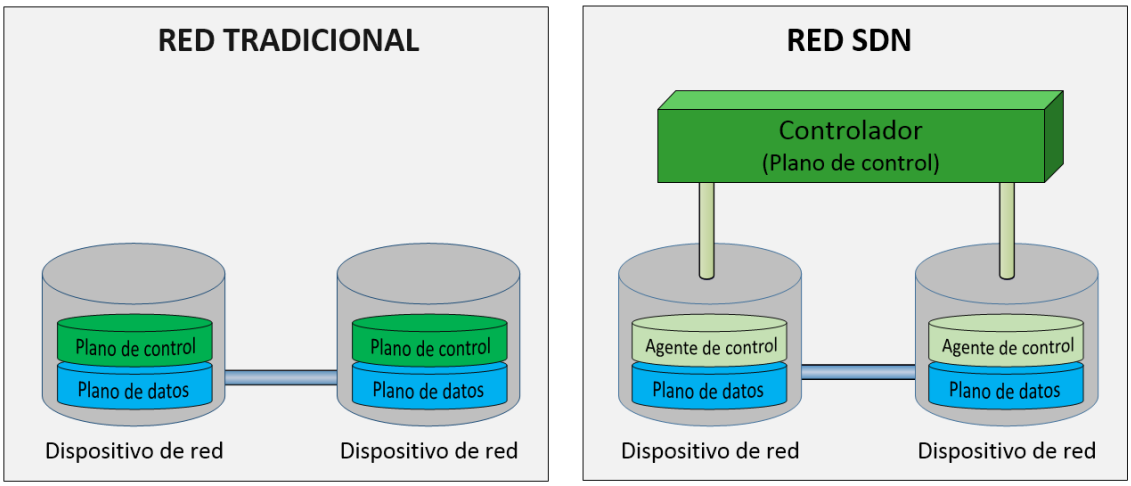

Figura 2.1. Comparativa entre redes tradicionales y redes SDN

Además del desacoplamiento del plano de control respecto al plano de datos, que origina una arquitectura divida en dos capas, hay que sumar un nuevo nivel superior de abstracción que permite la comunicación de aplicaciones con el plano de control..

De esta forma, la arquitectura SDN se encuentra dividida en tres capas. La capa de infraestructura que se compone por los diferentes nodos de la red que realizan la conmutación de paquetes dada por el plano de datos que contienen. En la capa de control se sitúa el controlador SDN que posee la inteligencia de la red para dirigir de manera óptima los flujos de tráfico. Y la capa de aplicación, que supone el mayor nivel de abstracción y se compone por aplicaciones de negocio que permiten la gestión de la red mediante servicios conectados al controlador a través de una interfaz.

Dada dicha arquitectura, desde el plano de control se diferencian dos interfaces de programación (API) en cada extremo que permiten la comunicación con las restantes capas:

- Interfaz northbound: Es utilizada para la interacción entre el controlador SDN y las aplicaciones que proveen un mayor nivel de abstracción de la red. Puede ser utilizada para innovar en aplicaciones que permitan una adecuada y automática gestión de la red gracias a la programabilidad de las redes SDN. Es posible la integración con plataformas como Puppet, Chef, Ansible u OpenStack.
- Interfaz southbound: Proporciona la conexión entre el controlador SDN y los diferentes nodos que componen la red. A través de ella el controlador descubre la topología de la red y permite realizar cambios dinámicamente en el tráfico según las demandas que surjan en tiempo real. Un ejemplo de esta interfaz es OpenFlow, que se describe más adelante.

A continuación se muestra la arquitectura SDN completa con las tres capas mencionadas y las interfaces que permiten la comunicación entre ellas:

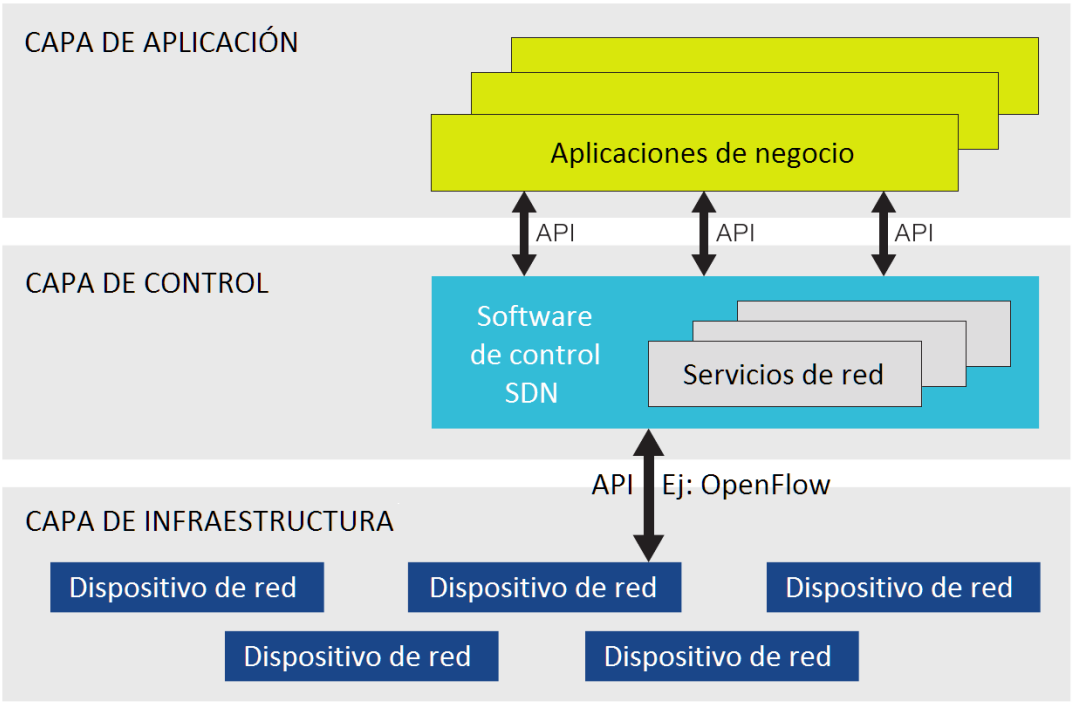

Figura 2.2. Arquitectura SDN [2]

La arquitectura propuesta pretende solucionar diferentes aspectos negativos que presentan las redes tradicionales, como son la complejidad, la escasa flexibilidad, la imposibilidad de escalabilidad y la fuerte dependencia con los vendedores. A lo largo de los años la industria ha ido produciendo nuevos protocolos de red más seguros y eficientes, pero todos ellos han sido definidos de manera aislada con el objetivo de resolver un problema específico y sin obtener un beneficio de forma conjunta. Esta complejidad ha originado una red estática, y hoy en día se necesitan redes convergentes que sean capaces de operar con distintos tipos de tráfico a la vez de manera dinámica y flexible.

Además, la aplicación de políticas en una red grande, en la que se encuentra descentralizado el control, implica la configuración de múltiples dispositivos que como resultado puede ocasionar inconsistencias.

A todo ello hay que sumar que las demandas de proveedores y empresas de nuevas capacidades en la red se ven frenadas por la producción de los equipamientos por parte de los vendedores, que puede llegar a durar hasta más de tres años.

Para hacer frente a las limitaciones mencionadas que existen en las redes actuales, las Redes Definidas por Software proponen un cambio en la arquitectura, de forma que se puedan conseguir redes capaces de abordar con éxito los requerimientos exigidos. Por ello, la arquitectura SDN se define como:

- Programable, porque posibilita manejar el control de la red mediante la implementación de programas que optimizan rápidamente los recursos.
- Flexible, al permitir ajustar dinámicamente el tráfico de la red para satisfacer los diferentes cambios que se puedan originar.
- Gestionada centralizadamente, ya que dispone de un controlador SDN que contiene una visión global de la red y tiene la inteligencia para su control.
- Basada en estándares abiertos, los cuales permiten simplificar la red porque las instrucciones son proporcionadas por controladores SDN en lugar de por múltiples dispositivos específicos de un cierto fabricante.

A pesar de existir el concepto de Redes Definidas por Software desde hace años, no se ha dado a conocer tan ampliamente hasta 2011, que fue cuando se constituyó la Open Networking Foundation (ONF) [3].

La ONF es un consorcio industrial sin fines de lucro que está liderando el desarrollo y avance de las SDN, y entre otras cosas se encarga de la estandarización de OpenFlow. Algunos de los miembros fundadores de la ONF son Google, Facebook y Microsoft, y a estos se han ido añadiendo otros como Cisco, Dell, HP, F5 Networks, IBM, NEC, Huawei, Juniper Networks, Oracle y VMware, todos ellos de gran relevancia en el sector de las telecomunicaciones.

2.2 OpenFlow

Es el primer estándar que define la interfaz de comunicación entre el controlador y los diferentes dispositivos de la red de la arquitectura SDN con el objetivo de ajustar el comportamiento del plano de datos. Dicho estándar es abierto y ha sido ampliamente implementado por los fabricantes, por lo que lo convierte en un referente en la interfaz southbound.

Surgió a partir del proyecto de investigación *"OpenFlow: Enabling Innovation in Campus Networks"* de 2008 en la Universidad de Stanford [4], con el objetivo de permitir a los investigadores experimentar e innovar con protocolos usados diariamente en sus redes. La primera especificación surgió a finales de 2009, y desde entonces ha seguido evolucionando con la supervisión de la ONF, ya que desde el año 2011 tiene el control sobre la especificación.

La arquitectura que propone OpenFlow, como se muestra en la figura 2.3, se compone de los siguientes elementos: switch, controlador y protocolo.

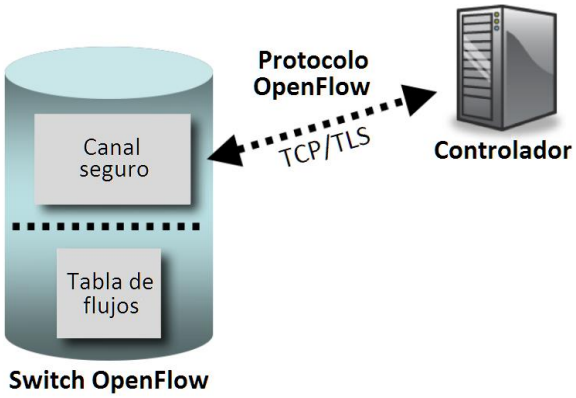

Figura 2.3. Arquitectura OpenFlow (versión 1.0) [4]

2.2.1 Switch

El switch es el dispositivo encargado del procesamiento directo de los paquetes que recorren la red. Está compuesto por un agente OpenFlow, situado en el extremo de un canal seguro, que recibe e interpreta los mensajes que ordena el controlador, y los traduce a un conjunto de instrucciones que implementa en el plano de datos del switch, donde se sitúa la tabla de flujos [5].

La tabla de flujos es la parte principal del switch, y en ella se encuentran un conjunto de entradas que proporcionan instrucciones de reenvío que indican que acción se debe realizar con un determinado flujo de tráfico. Se define como flujo a la agrupación de paquetes que comparten una serie de características comunes, como puede ser una cierta pareja de dirección IP origen y dirección IP destino. Cada entrada de flujo se compone de los siguientes elementos:

- Campos de correspondencia: Definen un cierto flujo mediante el establecimiento de un conjunto de campos que se comparan con los paquetes recibidos en el switch.
- Prioridad: Implanta un orden de preferencia en las entradas de una tabla de flujos.
- Contadores: Muestran estadísticas sobre el flujo correspondiente a una cierta entrada, como por ejemplo el número de paquetes o bytes identificados.
- Instrucciones: Indican una acción o conjunto de acciones que se deben ejecutar con un cierto flujo.
- Temporizadores: Permiten la eliminación de una cierta entrada de forma automática transcurrido un cierto periodo de tiempo.
- Cookie: Es una información añadida por el controlador que permite la identificación de la entrada de flujo.

Con tal estructura, al recibir un nuevo paquete entrante por alguno de los puertos del switch, se realiza un proceso de correspondencia, conocido como *packet matching*, que consiste en comparar los campos de las cabeceras del paquete y su puerto origen con los campos de correspondencia de las entradas de la tabla de flujos. En la primera versión de OpenFlow existen 12 posibles campos para hacer este proceso, que son:

- Puerto origen
- Dirección Ethernet de origen
- Dirección Ethernet de destino
- Tipo de trama Ethernet
- Identificador de VLAN
- Prioridad de VLAN
- Dirección IP de origen
- Dirección IP de destino
- Protocolo de la capa superior a IP
- Tipo de Servicio IP (ToS)
- Puerto origen TCP/UDP
- Puerto destino TCP/UDP

Cuando un determinado paquete coincide con los campos de correspondencia de una entrada de la tabla de flujos, se realiza la acción o acciones que se describen en dicha entrada y se actualizan las estadísticas. Las acciones que se pueden realizar son:

- Reenviar el flujo a un puerto o conjunto de puertos específico: Encamina los paquetes a través de la red.
- Reenviar el flujo al controlador: De esta forma el controlador recibe, a través del canal seguro, los paquetes de un determinado flujo para procesarlos y tomar una cierta decisión al respecto.
- Modificar campos del paquete: Permite añadir, editar o eliminar campos de las cabeceras de los paquetes recibidos de un cierto flujo.
- Descartar el flujo de paquetes: Puede ser utilizado para implementar medidas de seguridad en la red, y de esta forma utilizar el switch OpenFlow como un firewall.

Mediante la combinación de los campos de correspondencia y las posibles acciones se puede gestionar ampliamente el comportamiento de un switch OpenFlow, y de esta forma implementar funcionalidades como conmutación a nivel 2, a nivel 3, traducción de direcciones de red, firewall, balanceo de carga, etc.

Cuando un paquete recibido no encuentra ninguna coincidencia con las entradas de la tabla de flujos se debe crear una nueva entrada para recoger al flujo desconocido. Para ello el paquete se envía al controlador, se procesa y se informa al switch sobre un nuevo flujo a crear en su tabla. Finalmente, el paquete se envía de vuelta al switch y a partir de entonces se realizarán las acciones definidas en la nueva entrada de flujo creada.

Las entradas creadas en la tabla de flujos son acompañadas por unos temporizadores que permiten el borrado automático de la entrada pasado un cierto tiempo. Uno de los temporizadores es el denominado *IDLE_TIMEOUT*, que permite el borrado de la entrada tras un cierto tiempo de inactividad en el flujo. El valor de este temporizador es comparado con un campo recogido en las estadísticas, que proporciona el tiempo desde que se produjo la última coincidencia de un paquete, y de esta forma es posible saber la inactividad de un cierto flujo. El otro temporizador que puede ser utilizado es *HARD_TIMEOUT*, que indica el tiempo invariable en el que expirará una entrada desde su creación. El uso de estos temporizadores es opcional, por lo que una entrada puede permanecer de forma permanente en una tabla de flujos. Sin embargo, el controlador siempre tiene el control de la red y puede eliminar o renovar una cierta entrada en el momento que considere oportuno.

Actualmente existen multitud de dispositivos de red que tienen implementado OpenFlow. Se pueden encontrar en fabricantes como Cisco, que han incorporado el estándar en switches y routers que ofrecen a sus clientes, y en proyectos como Open vSwitch, que es ampliamente utilizado.

Open vSwitch [6], abreviado como OVS, es un proyecto de código abierto que proporciona un switch virtual soportado por la mayoría de hipervisores. Se encarga de reenviar el tráfico entre diferentes máquinas virtuales que se encuentran en el mismo equipo o a través de una red física. Utiliza OpenFlow para el control de flujos y OVSDB [7] como protocolo de gestión para administrar las implementaciones de OVS.

2.2.2 Controlador

Consiste en una aplicación software que se encarga de programar en todos los dispositivos de la red las entradas de flujo necesarias para gestionar el tráfico adecuadamente. Es capaz de dirigir el plano de datos de diversos dispositivos de diferentes fabricantes mediante un único protocolo estándar, OpenFlow.

Dada la amplia aceptación del estándar, actualmente la mayoría de controladores SDN disponen de OpenFlow en la interfaz southbound, y algunos de ellos incorporan otras interfaces de comunicación de forma adicional.

El controlador se encuentra permanentemente a la escucha de cualquier mensaje proveniente de los switches OpenFlow de la red. Desde la especificación 1.4 de OpenFlow, la Autoridad de Números Asignados en Internet (IANA) ha adjudicado a la ONF el puerto TCP 6653 con el objetivo de homogeneizar el uso de un único puerto, ya que en las primeras implementaciones de OpenFlow se han utilizado otros números de puertos TCP como el 975, 976 y 6633.

2.2.3 Protocolo

La comunicación entre el controlador y los switches OpenFlow de la red se produce sobre el nivel de transporte mediante el protocolo TCP. Es posible asegurar el canal utilizando la capa TLS (Transport Layer Security) si la comunicación se transfiere por un medio hostil. Este caso no se contempla en entornos completamente controlados, como puede ser un centro de datos, ya que la utilización de cifrado basado en TLS afecta al rendimiento.

Cada mensaje OpenFlow comienza con la misma estructura en la cabecera, lo que permite su utilización independientemente de la versión que se esté manejando. Los campos que la componen son:

- Version (8 bits): Indica la versión de OpenFlow que está siendo utilizada. Actualmente el valor que puede tomar se comprende entre 1 (versión 1.0) y 5 (versión 1.4).
- Type (8 bits): Informa sobre el tipo de mensaje que se presenta en el cuerpo del paquete. El valor que contiene depende de la versión utilizada, y en la última versión puede tomar valores entre 0 y 34, ya que existen 35 tipos de mensajes [8].
- Length (16 bits): Indica donde termina el mensaje OpenFlow proporcionando su longitud. El valor mínimo que puede tomar es 8, ya que 8 bytes es el tamaño de la cabeza fija que contienen todos los paquetes OpenFlow.
- Xid (32 bits): Proporciona un identificador de la transacción que se está realizando. Es un valor único que permite enlazar solicitudes con respuestas.

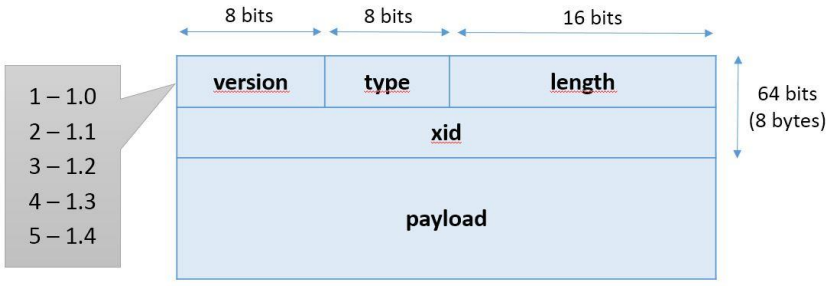

Figura 2.4. Estructura de un paquete OpenFlow

Además, dependiendo de la intención de comunicación de los mensajes OpenFlow se diferencian tres tipos:

- Controlador a Switch: Son mensajes iniciados por el controlador y dirigidos al switch, que pueden requerir una respuesta en ciertos casos, y tienen el objetivo de gestionar y examinar el estado de los switches de la red.
- Asíncronos: Son mensajes iniciados por el switch y dirigidos al controlador sin que este los haya solicitado. Se utilizan para notificar al controlador de eventos que han surgido en la red, como por ejemplo la llegada de un paquete o el cambio de estado en un switch.
- Simétricos: Se transmiten tanto desde el switch como desde el controlador sin que haya surgido ninguna solicitud previa. Dentro de este tipo se incluyen mensajes para establecer y verificar la conexión entre el controlador y el switch.

2.2.4 Evolución

Desde el lanzamiento de la primera especificación en diciembre de 2009 se han ido añadiendo numerosas mejoras para el perfeccionamiento del estándar. Además, con el apoyo de la ONF desde la versión 1.2, se ha conseguido un mayor impacto en los vendedores consiguiendo ser implementado en los dispositivos. A continuación se muestran las principales mejoras que se han producido en la especificación hasta la última publicación, en abril de 2015, con la versión 1.5.1:

- OpenFlow 1.1 (febrero de 2011): Destaca la utilización de varias tablas de flujos en un mismo switch y la tabla de grupos, que proporcionan una mayor flexibilidad de acciones a ejecutar dado un paquete entrante en el switch.

  Al disponer de varias tablas de flujos es posible realizar el proceso de correspondencia por las entradas de múltiples tablas de flujos conectadas, lo que se conoce como *pipeline*. Cuando no coincide ninguna entrada de una tabla de flujos con un cierto paquete se puede continuar examinando las tablas de flujos siguientes, ya que se encuentran en un cierto orden. Si por el contrario, en algún momento existe una coincidencia con una entrada, se dispone de una acción adicional que permite continuar el proceso de correspondencia en una tabla específica y así formar un conjunto de acciones a realizar, conocido como *action set*. Esto añade el concepto de *metadata*, que consiste en un registro de información utilizado durante el *pipeline*. En la figura 2.5 se muestra el proceso de correspondencia que se sigue con un paquete entrante en el switch.

La tabla de grupos, conocida como *group table*, está compuesta por entradas que identifican las acciones a realizar por un determinado grupo. Este concepto de grupo permite indicar de forma eficiente que un mismo conjunto de acciones deben ser procesadas por múltiples flujos, siendo útil para implementar multicast.

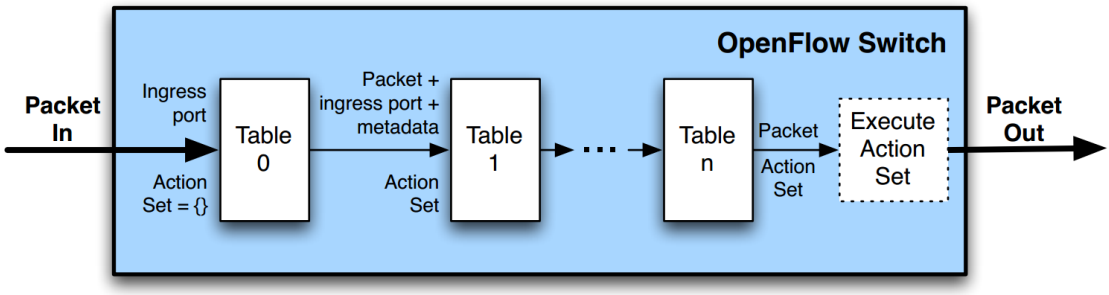

Figura 2.5. Proceso de correspondencia OpenFlow con múltiples tablas de flujos [9]

- OpenFlow 1.2 (diciembre de 2011): Se añade soporte para IPv6 y se crea una expresión flexible para el *packet matching* que permite la comparación con nuevos campos en los actuales y futuros protocolos. Esto reemplaza a la estructura fija que se estaba utilizando por una nueva estructura denominada *OpenFlow Extensible Match* (OXM). Con este nuevo avance se puede llegar a realizar el proceso de correspondencia con campos que se encuentren dentro del cuerpo del paquete, creando entradas de flujos sin límites

  Además, se especifica la conexión de un switch con varios controladores. El controlador podrá asumir el rol de maestro o de esclavo, y dependiendo de ello tendrá mayor o menor privilegio en la toma de acciones sobre el switch.

- OpenFlow 1.3 (junio de 2012): Destaca la aparición de una nueva tabla en el plano de datos del switch, la tabla de medidas, del inglés *meter table*. Consiste en una serie de entradas que definen un conjunto de métricas por flujo que facilitan el control de la tasa de paquetes asignadas a ellas. Permiten implementar operaciones simples de calidad de servicio, como pueden ser limitaciones en el tráfico, o pueden ser combinadas con colas para implementar configuraciones de calidad de servicio más complejas como DiffServ.

  Cabe mencionar otras mejoras como el añadido del etiquetado PBB (Provider Backbone Bridging) utilizado en las redes troncales de los operadores, y un mayor soporte a IPv6. Ahora se podrá realizar *packet matching* con otros campos de las cabeceras de extensión de IPv6.

- OpenFlow 1.4 (octubre de 2013): Aparece un nuevo mecanismo parar realizar modificaciones en los switches garantizando atomicidad en ellas, *bundles mechanism*. Consiste en agrupar un conjunto de mensajes OpenFlow en uno solo y garantizar el éxito de todos ellos. De esta forma se realizarán todas las modificaciones juntas, o si falla alguna todas las demás fallarán. Asimismo, proporciona una mejor sincronización en los cambios producidos.

  Se agrega la posibilidad de sincronizar tablas de flujo, tanto de manera unidireccional como bidireccional. Cuando una entrada de flujo es añadida, modificada o eliminada en una tabla origen, su correspondiente entrada sincronizada, en otra tabla de flujos, debe ser añadida, modificada o eliminada automáticamente. Este método permite tener varios flujos con la misma información en diferentes puntos del *pipeline*.

  Además, a partir de este momento el puerto TCP utilizado por OpenFlow es el 6653, aprobado por la IANA.

- OpenFlow 1.5 (diciembre de 2014): Se extiende el pipeline hacia un conjunto de tablas de flujos que se basan en el contexto del puerto de salida del switch. Hasta ahora las entradas de flujo de las tablas dependían fundamentalmente de los campos que provienen en el paquete y del puerto origen en el switch. Esto añadía una limitación si se quería realizar el proceso de correspondencia teniendo en cuenta el puerto de salida de un cierto flujo. Así, ahora cuando un paquete sea redirigido hacia un puerto de salida, comenzará a procesar la primera tabla de flujos de salida y continuará por un *pipeline* similar al utilizado en versiones anteriores. La siguiente figura muestra la arquitectura actual de OpenFlow:

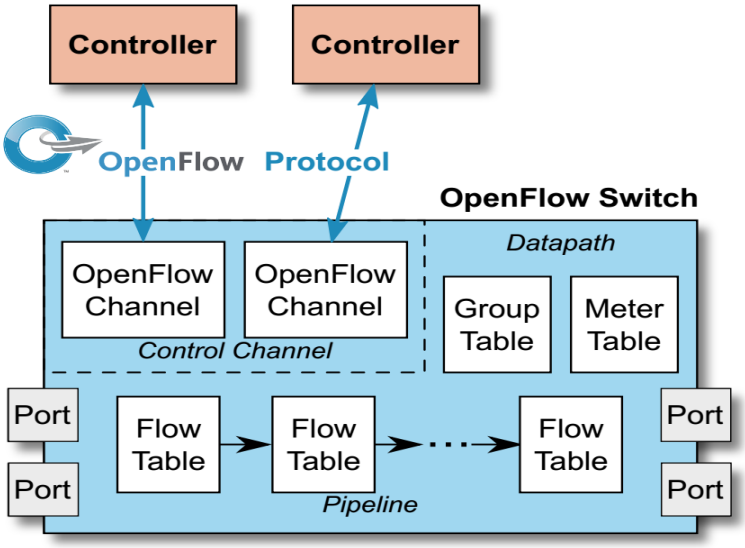

Figura 2.6. Arquitectura OpenFlow (versión 1.5) [10]

2.3 Controladores SDN

En la comunidad científica existe un gran trabajo en las Redes Definidas por Software, y hoy en día existen diferentes iniciativas que han liberado software implementando diferentes funcionalidades necesarias para desplegar un entorno real. En el mercado existen diversos controladores SDN que pueden ser seleccionados para llevar la gestión de la red y resulta complicado elegir el adecuado para unas ciertas necesidades [11].

Como se ha mencionado, OpenFlow es un estándar bastante reconocido como interfaz southbound en la arquitectura SDN, por lo que elegir un controlador con soporte OpenFlow ofrece muchas ventajas. Asimismo, es recomendable que proporcione soporte a nuevas extensiones del estándar. Esta recomendación es importante ya que es la alternativa del interfaz southbound en la que más información disponible hay hasta el momento para abordar las SDN.

El rendimiento es otra de las características importantes a evaluar mediante el establecimiento de flujos. La medida que se utiliza en este caso es el tiempo de instalación de un flujo, y con ello el número de flujos por segundo que el controlador puede instalar. Se debe garantizar que el rendimiento del controlador no produzca un cuello de botella en la red implantada.

La inclusión de ciertas funcionalidades de red permite un mejor manejo del tráfico, y por ello algunos controladores SDN llevan incorporadas aplicaciones que pueden ser útiles en la red, como software de seguridad, balanceadores de carga, servicios de computación en la nube, etc.

Un controlador SDN debe manejar un gran número de switches, por lo que el concepto de escalabilidad en el momento de seleccionar un controlador es importante. Depende de las necesidades actuales y futuras de la red, pero un solo controlador SDN debería ser capaz de administrar al menos 100 switches sin problemas de rendimiento.

Además, las implementaciones de los controladores son muy variadas y se encuentran controladores en diversos lenguajes de programación, como C/C++, Java, Python o Ruby entre otros.

Dada la amplia diversidad de controladores SDN, se ha realizado un estudio sobre las principales soluciones comerciales y de código abierto que se describe a continuación.

2.3.1 Controladores comerciales

Algunos controladores SDN comerciales son estrictamente basados en software, como propone el enfoque SDN, pero otros mantienen la idea de seguir utilizando hardware propietario. La principal ventaja de estos frente a los controladores de código abierto es el soporte que ofrecen los fabricantes. Destacan muchos vendedores que contribuyen activamente en el proyecto de código abierto OpenDaylight, que se trata en el *Apartado 3. OpenDaylight*, y a partir de él han añadido características propietarias para ofrecer un controlador comercial. Los actores más importantes que existen en el mercado hoy en día son:

- Big Switch: Su controlador Big Cloud Fabric combina un número de módulos para lograr una transición comprensible de redes tradicionales a redes SDN. En este caso, se encuentra enfocado en switches de Big Switch que están ejecutando su sistema operativo Switch Light.
- Cisco: Dispone de varias plataformas para proporcionar software de control en las SDN. XNC es la versión comercial del controlador OpenDaylight que utilizan como referencia en implementación, y está soportado en la plataforma que proporcionan llamada Cisco One Platform Kit, abreviada como onePK. También disponen de Application Policy Infrastructure Controller (APIC) que ha sido diseñado para incrementar el acceso a políticas en entornos northbound y southbound.
- HP: Han desarrollado su propio controlador, HP VAN SDN Controller Software, con el que pretenden crear un ecosistema basado en OpenFlow y proporcionar un kit de desarrollo de software SDN (SDK), que fomente la creación de aplicaciones en el controlador.
- Juniper: Ofrecen una solución comercial que se compone del controlador de código abierto OpenContrail en la que proporciona soporte adicional para los usuarios. También contribuyen en el proyecto OpenDaylight pero su solución ha sido desarrollada de forma independiente.
- NEC: Ha producido la primera solución comercial SDN basada en OpenFlow, que introdujo en 2011. Desde entonces ha mejorado y actualmente es uno de los controladores más maduros y robusto en el mercado que utilizan varias organizaciones. Además, también es participante del proyecto OpenDaylight y ha contribuido parte del código de su controlador relacionada con la creación de redes virtuales.

2.3.2 Controladores de código abierto

Se caracterizan por tener una rápida evolución debido a una amplia comunidad que los respaldan. Algunas de las soluciones más utilizadas han surgido como una bifurcación de proyectos de controladores pioneros en las SDN, e incluso existen soluciones comerciales que se han basado en ellos. Destacan los siguientes controladores de código abierto:

- NOX: Es el controlador original OpenFlow en el campo de las SDN. Fue donado a la comunidad investigadora en 2008 por Nicira, y desde entonces ha sido utilizado por decenas de grupos de investigación y ha servido como base para otros proyectos relacionados con las SDN. Está escrito en C++ y es soportado solamente por sistemas Linux. Desafortunadamente no ha sido fuertemente implementado debido a deficiencias en su entrono de desarrollo y se ha parado su desarrollo.
- POX: Se desarrolló a partir de NOX, está implementado en Python y es soportado por sistemas Windows, Mac OS y Linux. Su uso está pensado principalmente para investigación y educación. Tiene un entorno de desarrollo para trabajar fácil, dispone de un conjunto razonable de APIs y existe documentación sobre su utilización.
- Ryu: Está escrito en Python. Dispone de un conjunto de APIs bien definidas, que permite el desarrollo de nuevas aplicaciones de gestión de red, y de varios protocolos para la gestión de los dispositivos de red, como OpenFlow y Netconf. Está escrito en Python soporta completamente desde la versión 1.0 hasta la 1.4 de OpenFlow.
- Beacon: Es un controlador modular y multiplataforma compatible con la programación basada en threads y eventos. Está implementado en Java y aunque su influencia en el campo de las SDN ha decaído, ha servido como referencia para proyectos muy activos actualmente como Floodlight y OpenDaylight.
- Floodlight: Está escrito en Java, y ha sido realizado partiendo del controlador Beacon. Cuenta con un sistema de módulos que lo hace extensible, fácil de configurar y de alto rendimiento. Está respaldado por una gran comunidad de desarrolladores que ha generado suficiente documentación sobre la plataforma.
- OpenContrail: Se trata de un proyecto dirigido por Juniper que también dispone de una versión comercial que se diferencia únicamente por el soporte que proporciona el fabricante. Está enfocado en proporcionar virtualizaciones de funciones de red.

- ONOS: Tal controlador se encuentra enfocado particularmente para casos de uso de proveedores de servicio con grandes redes WAN. Está implementado en Java, y su arquitectura se centra en la tolerancia a fallos y la distribución de la red mediante múltiples controladores. Además, dispone de diversas interfaces northbound y southbound.

- OpenDaylight: Ha surgido como un proyecto de la fundación Linux en el que contribuyen importantes empresas. Es un controlador multipropósito escrito en Java y originado a partir de Beacon. Además, es ampliamente reconocido y dispone de un gran número de módulos que pueden ser utilizados según las necesidades de la organización. Destaca sobre otros controladores al disponer de numerosas interfaces southbound, incluyendo OpenFlow, y de una capa de abstracción situada encima de dichas interfaces.

Dada la amplia variedad de controladores para abarcar las SDN, el presente trabajo se ha centrado sobre la plataforma OpenDaylight. Se ha elegido dicha opción porque es un controlador de código abierto que está ganando mucha importancia en el campo de las SDN con el objetivo de ser un referente. Además, existe una amplia comunidad que lo soporta, incluyendo desarrolladores de importantes empresas, que están trabajando en relevantes casos de uso, como la integración en un entorno de computación en la nube que se analizará más adelante.

3 OpenDaylight

Es un proyecto colaborativo basado en código abierto que pretende acelerar el desarrollo y la adopción de las Redes Definidas por Software, con el objetivo de proporcionar un enfoque más transparente que fomente la innovación. Forma parte de los proyectos de la Fundación Linux, y desde su origen en abril de 2013 se han unido importantes empresas del sector de las telecomunicaciones, como Brocade, Cisco, Citrix, Dell, Ericsson, HP, IBM, Intel, Juniper, Microsoft, NEC, Red Hat y VMware entre otras.

OpenDaylight se centra en la construcción de una completa plataforma SDN de código abierto basada en diversos estándares que permita ser desplegada en una amplia variedad de entornos. Se trata de una infraestructura de control modular de la que se espera incluir en ella soporte para estándares y protocolos SDN emergentes, servicios de red, nuevas aplicaciones, y elementos del plano de datos que incluyen interfaces de dispositivos físicos y virtuales. Un ejemplo de ello es OpenFlow, gracias a la colaboración con organizaciones como la Open Networking Foundation (ONF). Sin embargo, la plataforma no se limita a ciertos estándares en concreto, sino que tiene como objetivo soportar una amplia variedad de interfaces definidas en estándares abiertos, como pueden ser I2RS, VxLAN o PCEP, o incluso adoptar otros creados en un futuro y avanzar junto con la comunidad SDN.

La plataforma que proporciona el proyecto OpenDaylight puede ser desplegada directamente sin requerir ningún otro componente en un amplio ámbito, incluido los vendedores. Esto es posible debido a la arquitectura de la plataforma, que provee un conjunto de funciones básicas para las aplicaciones, y a la gran variedad de colaboradores que contribuyen en el proyecto.

No obstante, para utilizar el código del proyecto se debe colaborar aportando mejoras a OpenDaylight que proporcionen un valor adicional. El proyecto se encuentra en pleno desarrollo gracias a las contribuciones por una amplia comunidad bajo la Licencia Pública Eclipse v1.0 (EPL), la cual es una opción común para proyectos basados en Java. Tal licencia de software está aprobada por la Open Source Initiative (OSI) y es considerada como licencia de software libre por la Free Software Foundation (FSF). De esta forma se permite maximizar la compatibilidad de OpenDaylight con el gran ecosistema de bibliotecas y componentes de terceras partes que han sido publicadas bajo la licencia EPL también.

El desarrollo colaborativo y el software de código abierto son estrategias y acciones que aceleran la creación y adaptación de nuevas tecnologías, así como la evolución y el despliegue de entornos innovadores. Además se establece un marco de trabajo en el que todos los vendedores pueden aprovechar y desarrollar un valor adicional

3.1 Visión técnica

La arquitectura de la plataforma OpenDaylight mantiene la misma estructura que la vista en el enfoque de las SDN, tres capas principales separadas por dos interfaces que permiten la comunicación entre ellas. Además, en OpenDaylight hay que sumar un nuevo nivel que determina como responder a un cierto servicio independientemente de la comunicación utilizada en la interfaz southbound. En la siguiente figura se muestra la arquitectura al completo:

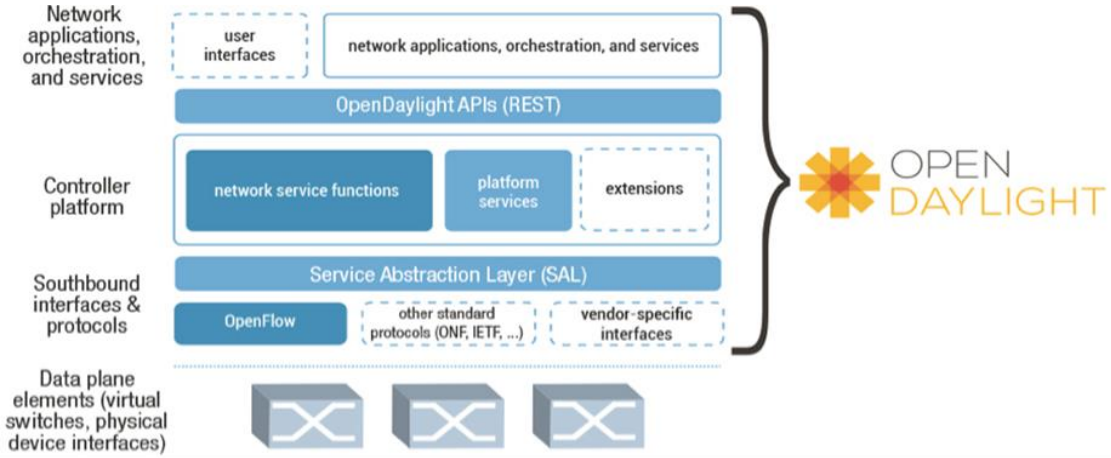

Figura 3.1. Arquitectura de la plataforma OpenDaylight [12]

En los siguientes subapartados se presenta en mayor detalle cada uno de los niveles que componen la arquitectura de la plataforma OpenDaylight.

3.1.1 Servicios de gestión y aplicaciones

La capa superior de la arquitectura está compuesta por aplicaciones de negocio y lógica de red que pueden controlar y supervisar el comportamiento de la red mediante la interacción con el plano de control. Para ello es necesario conectividad con el controlador a través de una interfaz situada en la interfaz northbound.

3.1.2 Interfaz northbound

Proporciona una diversidad de servicios del controlador a través de un conjunto de APIs REST que las aplicaciones pueden aprovechar para la gestión de la infraestructura de red. Para su utilización existe una capa de protección que aporta seguridad añadida a la plataforma.

3.1.3 Plataforma de controlador

Se trata de una plataforma modular que proporciona un conjunto de servicios de red. Entre ellos se encuentran los servicios básicos de red que incluyen los siguientes elementos:

- Topology Manager: Maneja la información sobre los dispositivos de red. Se compone de los siguientes servicios: Topology Discovery, Topology Update, Topology Database, Topology Northbound APIs.
- Statictics Manager: Proporciona un conjunto de estadísticas que son obtenidas a través de peticiones a los nodos de la red.
- Switch Manager: Mantiene el registro de los nodos de la red y detalles sobre sus conexiones. En el momento en que el controlador descubre un nuevo nodo en la red, recoge su información y es guardada por este gestor.
- Forwarding Rules Manager (FRM): Gestiona la programación de flujos en la red. Además mantiene un repositorio centralizado con todas las reglas instaladas en los nodos. Se puede considerar como un pre-compilador para la instalación de flujos.
- Host Tracker: Almacena información sobre los equipos finales, así como su dirección física y lógica. Puede ser configurado dinámicamente, escuchando los mensajes ARP que se envían por la red o enviando solicitudes de ARP si fuera necesario, o estáticamente utilizando la interfaz northbound que permite añadir y eliminar equipos finales.

Además, existen otros servicios en el mismo nivel de la arquitectura que interactúan con los servicios básicos de red y proporcionan características específicas a la red mejorando las funcionalidades SDN de la plataforma.

3.1.4 Capa de abstracción de servicios

Es uno de los conceptos centrales de la arquitectura de OpenDaylight que se encarga de la conexión entre los módulos de funciones básicas de red y los plugins de protocolo de la interfaz southbound. Determina como se debe cumplir un cierto servicio solicitado independientemente del protocolo subyacente utilizado entre el controlador y los dispositivos de red.

Desde un principio se desarrolló un enfoque API-Driven SAL (AD-SAL), pero dada su inefectividad se ha decidido migrar los proyectos dependientes de esta capa a un enfoque Model-Driven SAL (MD-SAL). En la figura 3.2 se muestra un diagrama básico de ambos enfoques.

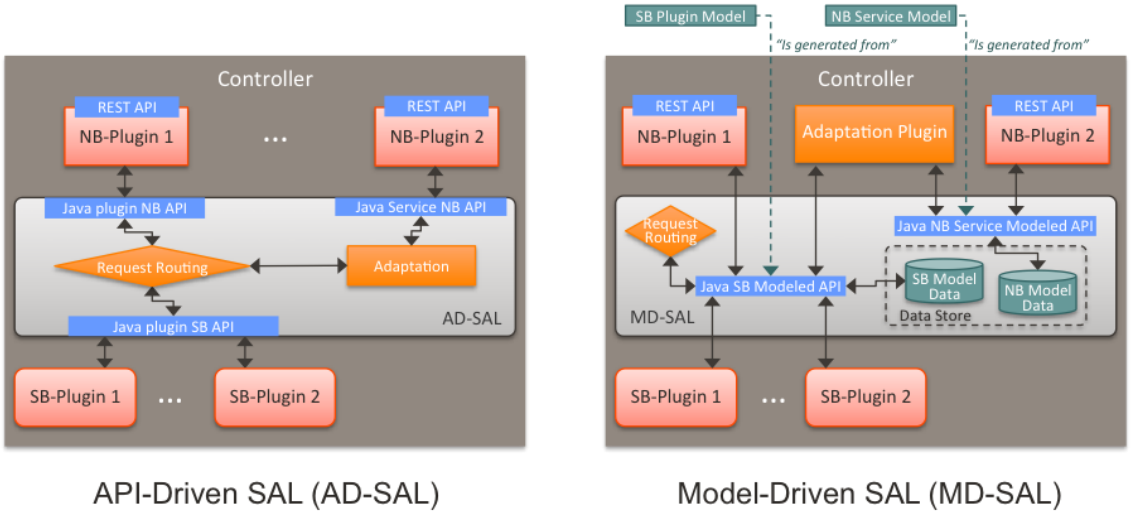

Figura 3.2. Enfoques de la capa de abstracción de servicios [13]

3.1.5 Interfaz southbound

En dicha capa se encuentran diferentes plugins que implementan ciertos protocolos con el propósito de controlar la infraestructura de red de la que se dispone. Entre la variedad de plugins que dispone la plataforma, se encuentran OpenFlow, OVSDB, NetConf o SNMP, ya que son ampliamente utilizados en la gestión de redes. Tal diversidad de interfaces permite la integración en entornos de varios fabricantes.

3.1.6 Dispositivos de red

En la capa inferior de la arquitectura se encuentran todos los dispositivos, tanto físicos como virtuales, que componen la red que gestiona OpenDaylight.

3.2 Arquitectura software

La plataforma OpenDaylight se encuentra implementada plenamente en software y se ejecuta a través de una Máquina Virtual Java (JVM). De esta forma permite ser utilizada en cualquier hardware que disponga un sistema operativo que soporte Java.

Para la construcción, ejecución y comunicación de las aplicaciones de la plataforma se utilizan un conjunto de herramientas que facilitan dichas tareas. La construcción de los proyectos software se lleva a cabo mediante Apache Maven, y la gestión de la plataforma se basa en un entorno de ejecución OSGi denominado Karaf, que facilita la comunicación entre aplicaciones que se ejecutan en el mismo espacio de direcciones. En los siguientes subapartados se proporciona más información sobre cada una de ellas.

3.2.1 Maven

Es una herramienta que permite automatizar el proceso de compilación y generación de ejecutables a partir del código fuente de proyectos Java [14]. Tiene un modelo de configuración de construcción basado en un formato XML mediante un archivo POM.xml (Project Object Model), que es utilizado para la definición del proyecto software, así como sus dependencias con otros módulos y componentes externos.

Maven está construido como una arquitectura basada en plugins que se pueden descargar dinámicamente desde repositorios mantenidos por Maven o por terceros. A través de ellos se puede utilizar cualquier aplicación controlable mediante la entrada estándar, y así mantener una interfaz con herramientas, por ejemplo, de compilación.

Los proyectos de software Maven se basan en una secuencia bien definida de fases que definen el orden en que diferentes objetivos deben ser logrados. A esta secuencia se denomina ciclo de vida, y de forma predeterminada se compone de las siguientes etapas:

1. Validación: Se encarga de la comprobación de la definición del proyecto y que toda la información necesaria se encuentre disponible.
2. Compilación: Compila los archivos fuentes .java del proyecto.
3. Test: Ejecuta test automáticos sobre el código compilado utilizando un marco de pruebas unitarias adecuadas.
4. Empaquetado: Con el código compilado genera un fichero distribuible, como .jar.
5. Pruebas de integración: Se realiza el despliegue en un entorno donde se ejecutan pruebas de integración.
6. Verificación: Comprueba la validez del código empaquetado, así como su calidad.
7. Instalación: Copia el fichero distribuible en el repositorio local de Maven, y de esta forma los artefactos generados pueden utilizarse en otros proyectos.
8. Despliegue: Copia el fichero distribuible a un servidor remoto para compartir con otros desarrolladores y proyectos Maven con acceso a ese servidor remoto.

Además se sigue una determinada convención en la estructura de directorios que permite la homogenización proyectos y facilitar la implementación. La estructura básica se compone de dos directorios, *src* que mantiene todo el código fuente para la construcción del proyecto, y *target* que es utilizado para almacenar los ejecutables obtenidos tras la construcción del proyecto. Dentro del directorio *src* se encuentran dos subdirectorios, *main* que mantienen el código principal del proyecto, y *test* que almacena las pruebas. En ambos subdirectorios se almacena por separado el código Java de los recursos que se necesiten. En el siguiente diagrama se muestra la estructura de un proyecto Maven:

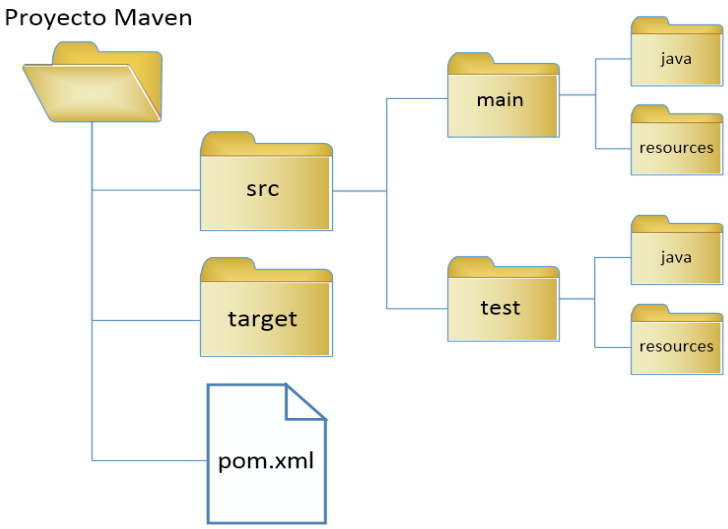

Figura 3.3. Estructura de directorios Maven

3.2.2 OSGi

La alianza OSGI fue creada en marzo de 1999 con el objetivo de definir especificaciones abiertas de software para el desarrollo de plataformas compatibles que proporcionen múltiples servicios. Fue pensada inicialmente para la aplicación en sector domótico y automovilístico, aunque su uso se ha extendido a otras plataformas como es el caso de OpenDaylight.

OSGi se basa en la utilización de componentes modulares, denominados bundles, de los que se especifica su ciclo de vida y las interacciones de los mismos. Además proporciona un registro de servicios que los bundles pueden utilizar para publicar, descubrir y unirse a otros en una arquitectura orientada a servicios (SOA).

Las dos principales implementaciones de OSGi son Eclipse Equinox [15] y Apache Felix [16]. La primera de ellas ha sido desarrollada por la Fundación Eclipse y es utilizada como entorno de ejecución de los diferentes componentes que forman el entorno de desarrollo integrado Eclipse. Por otro lado, Apache Felix se originó como un esfuerzo de la comunidad para implementar el marco de trabajo OSGi bajo la licencia Apache.

En la primera versión de OpenDaylight, Hydrogen, se utilizaban de forma conjunta las dos implementaciones mencionadas. Equinox era utilizado como lanzador y se encarga de la ejecución de los bundles, y Felix gestionaba la creación. A partir de la publicación de Helium se utiliza Equinox únicamente como implementación OSGi por defecto.

En definitiva, el controlador OpenDaylight se puede considerar como un conjunto de bundles apoyados sobre la implementación que proporciona Eclipse Equinox como marco de trabajo OSGi.

3.2.3 Karaf

Es un entorno de ejecución basado en OSGi que proporciona un contenedor ligero en el que varios componentes pueden ser desplegados. Permite la integración con las implementaciones de OSGi mencionadas en el apartado anterior, Apache Felix y Eclipse Equinox, y proporciona un conjunto de servicios al entorno simplificando aspectos como el empaquetado e instalación de aplicaciones.

Fue adoptado en OpenDaylight a partir del lanzamiento de Helium y se sigue manteniendo en la versión Lithium utilizando Eclipse Equinox como implementación OSGi.

Las principales características que proporciona Karaf son: configuración dinámica, gestión remota, consola de ejecución, despliegue dinámico, sistemas de registros, y seguridad entre otras.

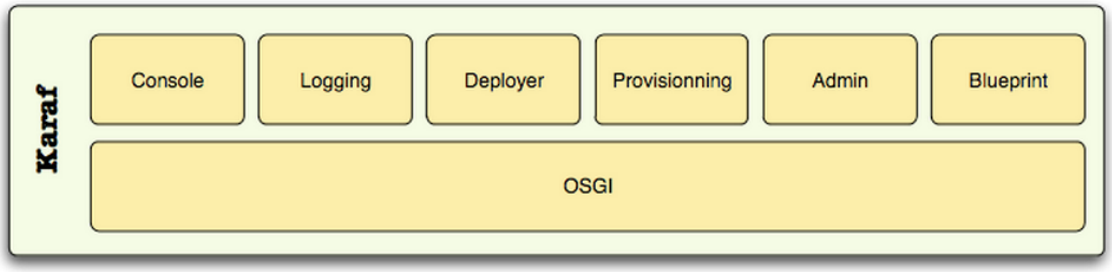

Figura 3.4. Arquitectura de Apache Karaf [17]

En las distribuciones Karaf de OpenDaylight se sigue la siguiente estructura de directorios:

- *bin:* Contiene los scripts que permiten el inicio de Karaf tanto en un sistema Unix como Windows. Además se encuentran algunos otros scripts que permiten configurar el entorno.
- *configuration:* Los archivos de dicha carpeta permiten cambiar los niveles de registro de varios bundles en tiempo de ejecución.
- *data:* Se almacenan ficheros que son modificados en tiempo de ejecución. Aquí se encuentran cacheados bundles que han sido desplegados y registros que guardan los eventos el entorno Karaf.
- *deploy:* Este directorio se reserva para el desarrollo y despliegue de bundles. No debe ser utilizado en entornos de producción.
- *etc:* Contiene toda la información que permite configurar los diferentes módulos del controlador.

- *lib*: Se almacenan los archivos ejecutables, como .jar, necesarios para ejecutar instancias de Karaf.
- *system*: Se utiliza como repositorio local Maven. Es utilizado por Karaf cuando se quiere instalar un nuevo bundle.

Además Karaf añade el concepto de feature, que consiste en la agrupación de diferentes bundles dando como resultado una aplicación. Este concepto es ampliamente utilizado en la instalación de componentes de OpenDaylight ya que permite instalar fácilmente todos los bundles necesarios para un cierto componente según desee el usuario.

3.3 Herramientas de gestión

Para la gestión del proyecto OpenDaylight se utilizan un gran número de herramientas que permiten ayudar a los desarrolladores en su participación. Entre sus funcionalidades se encuentran la gestión óptima del código, la resolución de errores o la comunicación entre desarrolladores con el objetivo de resolver problemas y dudas que surgen en relación al proyecto. La mayoría de estas herramientas son también utilizadas en otros proyectos de desarrollo software basados en integración continua.

En los siguientes subapartados se encuentran analizadas en mayor detalle las herramientas más destacadas que son utilizadas en OpenDaylight.

3.3.1 Ask

Es un foro de pregunta-respuesta utilizado por usuarios y desarrolladores de OpenDaylight. Actualmente está compuesto por más de 700 miembros y hay publicadas más de 1.000 preguntas, las cuales cubren un amplio número de temáticas, desde publicaciones relacionadas con el funcionamiento de la plataforma hasta errores con ciertos módulos.

Se encuentra disponible en: https://ask.opendaylight.org

3.3.2 Identity

Permite la creación y gestión de una cuenta de usuario OpenDaylight. Dicha cuenta es necesaria para la contribución en el proyecto, así como para la utilización de algunas de sus herramientas.

Se encuentra disponible en: https://identity.opendaylight.org

3.3.3 Bugzilla

Es una herramienta que facilita el registro y seguimiento de errores que aparecen a lo largo de un desarrollo de software. Originalmente fue concebida para la supervisión del proyecto Mozilla, pero puede aplicarse a cualquier proyecto que funcione con un modelo de anotaciones y notificaciones automáticas. Mediante una página web el usuario podrá realizar las acciones básicas de Bugzilla: consultar errores existentes o publicar un nuevo error. En este último caso será necesario disponer de una cuenta OpenDaylight.

Se encuentra disponible en: https://bugs.opendaylight.org

3.3.4 GitHub

Es una plataforma que proporciona el alojamiento de proyectos que utilizan como sistema de control de versiones Git. En él se encuentran almacenados todos los proyectos que componen OpenDaylight. Además de proporcionar un servicio de alojamiento, también destacan otras herramientas que permiten el seguimiento de problemas en el software, la revisión de código, y la visualización de ramas donde se pueden comparar los progresos realizados en cada una de ellas.

Se encuentra disponible en: https://github.com/opendaylight

3.3.5 Gerrit

Es un sistema de revisión de código basado en web que facilita la verificación online de proyectos que utilizan Git como sistema de control de versiones. Simplifica el mantenimiento del código ya que proporciona un marco de trabajo para la verificación de los cambios antes de que sean aceptados definitivamente en el código del proyecto. Cualquier usuario puede producir cambios en el código y subirlo a Gerrit, pero hasta que no se ha completado la revisión de forma satisfactoria los cambios no son actualizados como parte del proyecto. También es una buena herramienta de colaboración ya que permite almacenar las conversaciones que han surgido alrededor de la actualización de código.

Se encuentra disponible en: https://git.opendaylight.org/gerrit

3.3.6 Jenkins

Es un software de código abierto que facilita la integración continua para el desarrollo de software. Es posible la programación de tareas que se encargan de compilar y ejecutar un conjunto de pruebas a las actualizaciones de código que se producen en el desarrollo de software. Además de ayudar en la integración de código periódicamente, permite obtener métricas de calidad y visualizar los resultados dentro de la herramienta.

Se encuentra disponible en: https://jenkins.opendaylight.org

3.3.7 Nexus

Es un gestor de repositorios que almacena y comparte los artefactos de software. Se trata de un servidor proxy de librerías necesarias para la construcción de software. Ofrece una descarga de librerías más rápida, posibilita el desarrollo modular y permite tener una gestión centralizada de los artefactos a nivel del proyecto.

Se encuentra disponible en: https://nexus.opendaylight.org

3.3.8 Listas de correo

Es un servicio que permite la distribución de mensajes de correo electrónico entre múltiples usuarios interesados en una temática de OpenDaylight, y ayuda a estar informado sobre las discusiones que mantienen los desarrolladores en un cierto proyecto. Para ello es necesario realizar una subscripción a la lista deseada y esperar la confirmación por parte del administrador. Los mensajes enviados a dirección de correo de la lista son reenviados a las direcciones de correo electrónico de los suscriptores. Además, es posible recibir un resumen diario que incluye los mensajes incorporados a la lista cada día.

Se encuentra disponible en: https://lists.opendaylight.org

3.3.9 Sonar

Es una plataforma que se encarga de evaluar el software producido en el proyecto OpenDaylight, con el objetivo de mejorar la calidad del código. Cubre los siguientes aspectos: arquitectura y diseño, duplicaciones, pruebas unitarias, complejidad, errores potenciales, reglas de codificación y comentarios. Sonar utiliza para su estudio otras herramientas de análisis de código fuente como Checkstyle, PMD o FindBugs, y presenta de manera unificada la información obtenida a través de su interfaz.

Se encuentra disponible en: https://sonar.opendaylight.org

3.3.10 Wiki

En este portal se encuentra toda la información del proyecto OpenDaylight para desarrolladores. Está compuesta por el conjunto de proyectos que componen OpenDaylight y a medida que cada uno de ellos va madurando se va completando la documentación por los contribuyentes. Además contiene la planificación de desarrollo de las diferentes versiones de la plataforma OpenDaylight y muestra detalles como la dependencia que existe entre los diferentes proyectos.

Se encuentra disponible en: https://wiki.opendaylight.org

3.4 Versiones

El proyecto OpenDaylight ha experimentado un gran crecimiento desde el anuncio de su fundación el 8 de abril de 2013, siendo así uno de los proyectos de código abierto más grandes y exitosos. Se han publicado un total de tres versiones, Hydrogen, Helium y Lithium, y ya se encuentra una propuesta sobre el cuarto lanzamiento, Beryllium. Actualmente cuenta con el apoyo de 466 desarrolladores y 47 organizaciones, que han dado como resultado 2,3 millones de líneas de código en la última versión publicada en junio de 2015, Lithium.

A continuación se muestra un resumen sobre las versiones estables que se encuentran publicadas del proyecto OpenDaylight.

3.4.1 Hydrogen

El primer lanzamiento del proyecto OpenDaylight fue Hydrogen en febrero de 2014. Se publicaron un total de tres versiones, una básica con componentes mínimos de la plataforma y dos más especializadas según el usuario al que esté dirigido que consisten en añadir nuevos componentes a la versión base.

- Base *(Base Edition)*: Está pensada para aquellos usuarios que están explorando SDN para una prueba de conceptos tanto en entornos físicos como virtuales, o para el desarrollo de iniciativas académicas. En la interfaz southbound solo se incluyen las interfaces de OpenFlow, OVSDB y NetConf.
- Virtualización *(Virtualization Edition)*: Incluye una serie de componentes que permiten la creación y gestión de redes virtuales multiusuario, aparte de aplicaciones de seguridad y gestión de redes. Está pensada para la incorporación en centros de datos, e incorpora servicios como Affinity, VTN, DOVE y OpenStack.
- Proveedor de servicios *(Service Provider Edition)*: Permite facilitar la evolución de las redes existentes que poseen los proveedores de servicios hacia redes SDN. Incluye diferentes módulos que proporcionan soporte a protocolos comúnmente utilizados en las redes de los proveedores de servicios, así como aplicaciones de seguridad y gestión de redes. Incluye una amplia variedad de interfaces southbound, como SNMP, BGP-LS, PCEP y LISP, y los servicios Affinity y LISP.

La arquitectura que ofrece el conjunto de las diferentes versiones de Hydrogen se muestra a continuación:

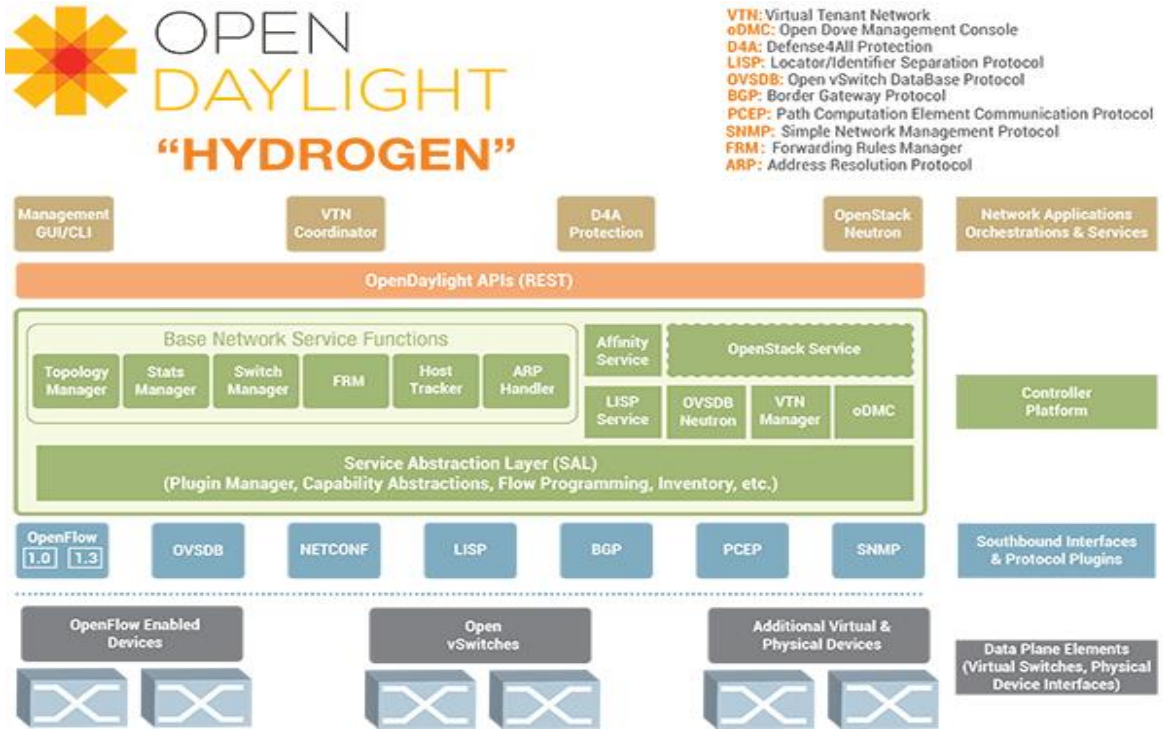

Figura 3.5. OpenDaylight Hydrogen [12]

3.4.2 Helium

Es el segundo lanzamiento del proyecto OpenDaylight, en septiembre de 2014, se centra en mejorar aspectos relacionados con la seguridad, escalabilidad y estabilidad necesarios en los entornos de producción. Entre las nuevas características que se añaden se encuentran una nueva interfaz de usuario, *openDayLight User eXperience (DLUX)*, una integración más profunda con OpenStack y un proceso de instalación más simple y personalizable a través de Apache Karaf. Con esta nueva característica se elimina el lanzamiento de varias versiones, como sucedió con Hydrogen, y el usuario es capaz de elegir que componentes instalar según las necesidades que disponga partiendo todos ellos de la misma distribución.

Helium está pensado para trabajar con redes razonablemente grandes pudiendo manejar un conjunto de controladores y de esta forma despreocuparse de las necesidades requeridas por las aplicaciones. Además, la plataforma ha evolucionado en otras áreas que incluyen alta disponibilidad, clustering y seguridad.

La siguiente figura muestra la arquitectura de Helium, en la que se han integrado nuevos componentes:

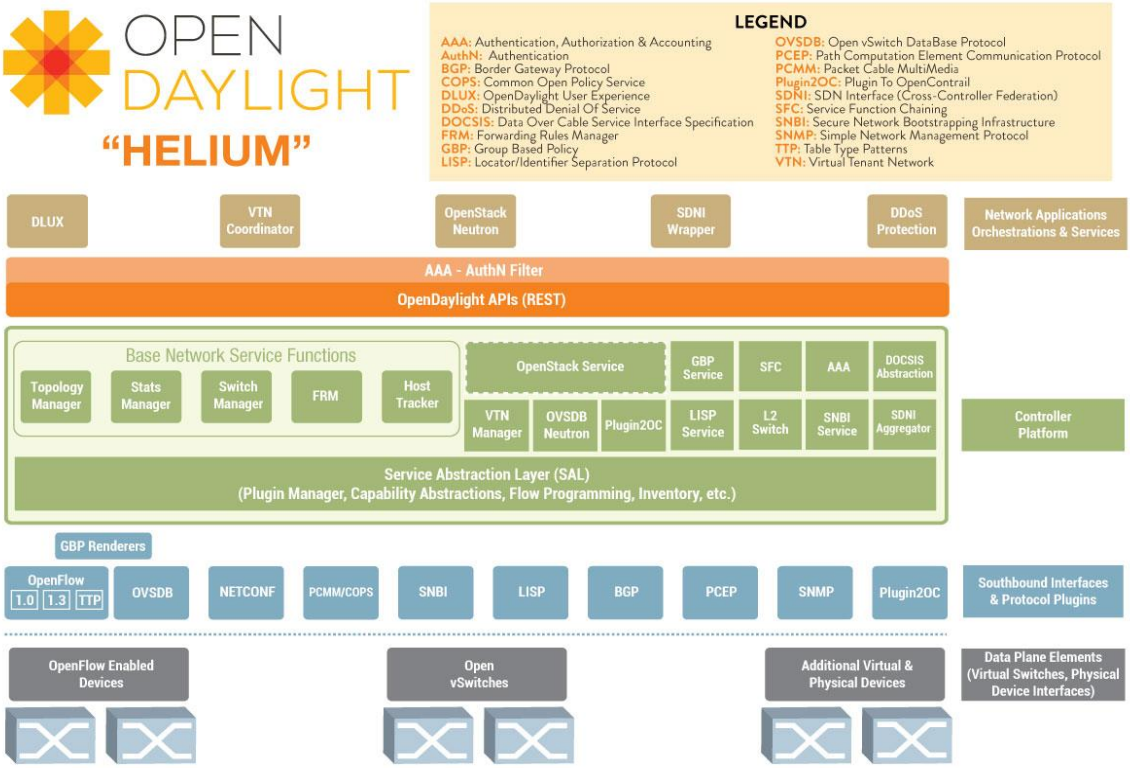

Figura 3.6. OpenDaylight Helium [12]

3.4.3 Lithium

El tercer lanzamiento del proyecto OpenDaylight ha sido Lithium, en junio de 2015, y se mantiene hasta la actualidad como la versión estable más reciente. En su plan de proyecto se plantea la generación de cuatro *Service Release* más para perfeccionar la distribución corrigiendo posibles bugs que se encuentren, con la intención de publicar la última versión de ellas en marzo de 2016.

En esta nueva versión se han añadido nuevas características y mejoras a los proyectos existentes. Se ha incrementado la escalabilidad y rendimiento de la plataforma, existen nuevos servicios para entornos de computación en la nube, hay nuevas características que mejoran la seguridad, se han añadido y mejorado APIs, y existen seis nuevas interfaces southbound para ampliar la conexión con dispositivos en el plano de datos.

Con Lithium se espera que proveedores de servicios y empresas puedan realizar la transición hacia las SDN con un particular enfoque en la ampliación de la capacidad de programación de las redes de comunicación. Y además, se espera ser utilizado sobre 20 soluciones comerciales, así como en el proyecto Open Platform for NFV (OPNFV) [18].

Lithium, la última versión estable del proyecto, presenta la siguiente arquitectura:

Figura 3.7. OpenDaylight Lithium [12]

3.5 Módulos utilizados

Durante la evolución de la plataforma OpenDaylight se han sumado proyectos que proporcionan nuevas características. Para ello es necesario la aprobación del Comité de Dirección Técnica de OpenDaylight, y hasta la fecha se encuentra un total de más de 40 proyectos aprobados.

Con un número tan alto de proyectos dentro de una única plataforma existen varias relaciones de dependencia entre ellos. En el último lanzamiento publicado, Lithium, se han establecido tres niveles de dependencia para establecer un orden en la entrega de artefactos de cada proyecto, dando como resultado el siguiente grafo:

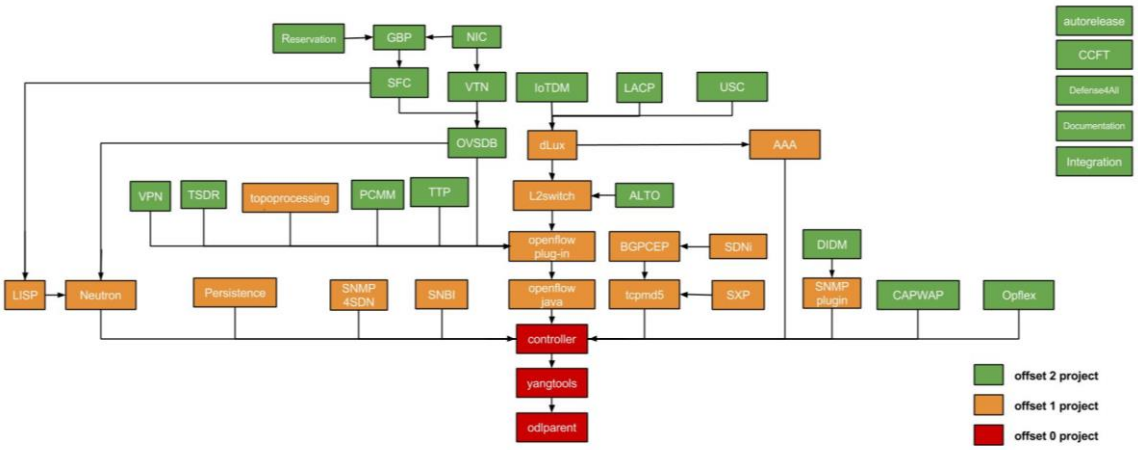

Figura 3.8. Grafo de dependencia de proyectos en OpenDaylight [12]

La Wiki de OpenDaylight proporciona un listado completo[1] de todos los proyectos aprobados, así como una página web por proyecto que incluye relevante información sobre su utilización. En los siguientes subapartados se trata en mayor detalle un conjunto de proyectos que serán utilizados en los escenarios de demostración desarrollados en el presente trabajo.

3.5.1 AAA

El proyecto AAA proporciona servicios de seguridad a la plataforma de OpenDaylight, protegiendo entre otras cosas el acceso de las aplicaciones a las APIs de la interfaz northbound. Se compone de los siguientes mecanismos:

- Autenticación: Proceso en el que un usuario realiza una propuesta sobre su identidad, y la demostración de estar en posesión de las credenciales permiten comprobarlo. Permite tanto una autenticación directa como federada.
- Autorización: Consiste en la concesión de privilegios al usuario basado en su identidad autenticada. Dependiendo de ello puede tener acceso a ciertos recursos, como llamadas RPC, y a acciones, como la lectura o escritura de datos sobre una topología en concreto.
- Contabilización: Se refiere al seguimiento de los recursos de la plataforma que han tenido acceso los usuarios. La información que un proceso de contabilización registra habitualmente es la identidad del usuario, el tipo de servicio que se le ha proporcionado, cuando comenzó a usarlo, y cuando terminó.

3.5.2 Controller

Tal proyecto se encarga de la construcción de un controlador modular, extensible y multiprotocolo que garantice escalabilidad y disponibilidad a los usuarios.

Originalmente su desarrollo partió del controlador Beacon, que introdujo la utilización de OSGi dotando de modularidad a la plataforma. A esta idea se añadieron otras como la arquitectura MD-SAL, que proporciona la abstracción necesaria para soportar múltiples protocolos a través de plugins en el interfaz southbound. Por otro lado, en la interfaz northbound se ha extendido un conjunto de aplicaciones que interactúan mediante servicios web REST.

Este proyecto es sobre el que se soporta el resto de proyectos que componen OpenDaylight, ya que proporciona la arquitectura de toda la plataforma.

[1] https://wiki.opendaylight.org/view/Project_list

3.5.3 DLUX

OpenDayLight User eXperience (DLUX) es una interfaz de usuario situada en el plano de aplicaciones de red que proporciona una gestión simple, moderna e intuitiva de la topología de red. Está creada a partir del marco de trabajo que proporciona AngularJS, y su apariencia se compone de un panel lateral situado a la izquierda en el que se muestran los módulos cargados y un marco central que recoge las acciones de cada módulo.

Para acceder a ella es necesario conectarse a través de un navegador a la URL http://<ODL-IP>:8181/index.html. Las credenciales por defecto son identificador de usuario y contraseña *admin*.

El módulo principal de DLUX carga el módulo *Topology* que muestra una representación gráfica de la topología de red creada, y a este se pueden añadir funcionalidades adicionales a la aplicación instalando nuevos módulos como:

- Node: Proporciona un conjunto de estadísticas de red e información sobre los puertos de los switches de la topología.
- Yang UI: Permite la interacción con el controlador OpenDaylight a través de un conjunto de APIs.

3.5.4 L2 Switch

Se encarga de proporcionar la funcionalidad de switching de nivel 2 a los switches de la topología. Cuando un paquete se recibe, este módulo aprende la dirección MAC origen, y si el destino es conocido lo reenvía hacia él. En caso contrario, enviará un mensaje broadcast hacia todos los puertos externos de la red. Solamente se encarga de los paquetes Ethernet que no se traten de paquetes LLDP, ya que se encargan de descubrir la topología. El siguiente cuadro resume las principales acciones:

DIRECCION MAC		ACCIÓN
Origen	Destino	
Desconocida	Desconocida	1. Aprende dirección origen 2. Reenvía el paquete por todos los puertos, excepto por el de entrada
Desconocida	Conocida	1. Aprende dirección origen 2. Reenvía el paquete por el puerto donde se encuentra el destino
Conocida	Desconocida	1. Reenvía el paquete por todos los puertos, excepto por el de entrada
Conocida	Conocida	1. Reenvía el paquete por el puerto donde se encuentra el destino

Figura 3.9. Cuadro resumen de acciones en el módulo L2 Switch

La arquitectura del proyecto se compone de los siguientes elementos:

- Packet Handler: Maneja los paquetes que llegan al controlador, examina la información relativa a las direcciones MAC, y los dirige hacia el correcto destino para ser tratados.
- Loop Remover: Se encarga de descubrir y remover bucles de la red actualizando el estado STP que corresponda a determinados puertos de los switches en la topología.
- ARP Handler: Gestiona los paquetes ARP, ya sea mediante la instalación de flujos de inundación en modo proactivo en los switches o mediante su tratamiento en el controlador, según la configuración.
- Address Tracker: Mantiene el control de direcciones MAC y direcciones IP de los dispositivos conectados en la red.
- Host Tracker: Monitoriza la situación de los equipos finales en la red.
- L2Switch Main: Instala flujos en cada switch dependiendo del tráfico que se transfiere por la red y las direcciones MAC aprendidas por *Address Tracker*.

3.5.5 Neutron Northbound

Se centra en la comunicación entre el proyecto OpenStack Neutron, encargado de la gestión de la red en OpenStack, y la plataforma OpenDaylight. Para ello se expone una API REST en la interfaz northbound de la plataforma OpenDaylight que recibe llamadas de un controlador instalado en OpenStack Neutron para ejecutar acciones como la creación de redes.

Los principales objetivos del proyecto consisten en evolucionar junto con las APIs de OpenStack Neutron, mantener la transparencia en la comunicación entre OpenStack y OpenDaylight, mejorar los métodos para el transporte de información y atraer más participación en el desarrollo.

3.5.6 OpenFlow Plugin

Es un componente de OpenDaylight situado en el interfaz southbound que dirige la comunicación entre el controlador y los dispositivos de red físicos o virtuales de la infraestructura de red que soportan el protocolo OpenFlow para la gestión de sus flujos. El plugin OpenFlow pretende desarrollar una interfaz que permita soportar las diferentes versiones actuales y futuras de OpenFlow.

Consiste en una implementación de la especificación de switches OpenFlow que genera la Open Networking Foundation. Se comenzó con la generación de un plugin basado en la especificación 1.0, y desde la versión Helium se encuentra disponible una implementación de la versión 1.3.3 de la especificación basada en la arquitectura MD-SAL.

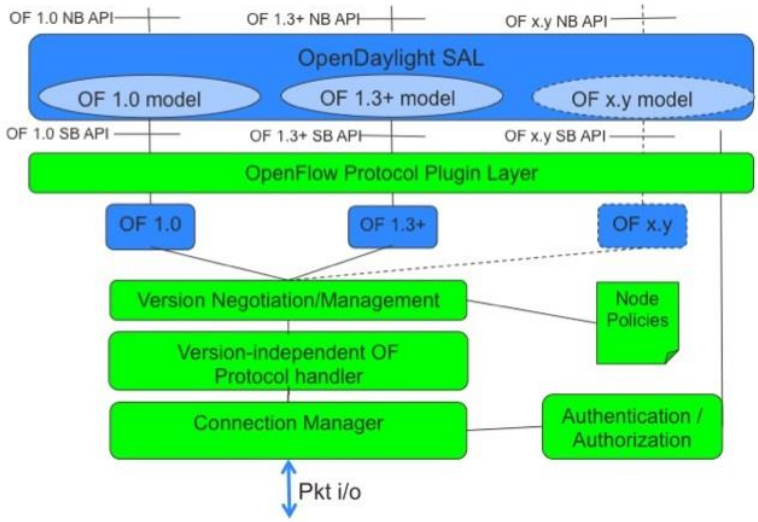

Figura 3.10. Plugin OpenFlow [12]

3.5.7 OVSDB MD-SAL Southbound Plugin

Se basa en una implementación del protocolo de gestión OVSDB [7], que permite la configuración de Open vSwitch a través de la interfaz southbound. El protocolo utiliza llamadas RPC codificadas en JSON para la gestión remota, y de esta forma OpenDaylight es capaz de obtener, crear, modificar y eliminar objetos como puertos de un Open vSwitch.

Una instancia de Open vSwitch se compone de un servidor ligero de base de datos (*ovsdb-server*), un proceso ejecutado en segundo plano (*ovs-vswitchd*), y opcionalmente de un módulo que proporciona reenvío de paquetes rápido (*forwarding-path*). El controlador es capaz de gestionar el servidor de base de datos a través del protocolo OVSDB y a través de OpenFlow se instalan entradas de flujo en los switches. En la figura 3.11 situada en el siguiente módulo se representa la interacción con los componentes de Open vSwitch.

Comparado con OpenFlow, OVSDB utiliza operaciones que ocurren en una escala de tiempo mayor, y entre sus acciones permite:

- Creación, modificación y eliminación de Open vSwitch.
- Configuración de controladores a los que se debe establecer la conexión, tanto a través del protocolo OpenFlow como OVSDB.
- Creación, modificación y eliminación de puertos, así como interfaces de túnel.
- Creación, modificación y eliminación de colas.
- Configuración de políticas de calidad de servicio, y asignación de ellas a determinadas colas.
- Recolección de estadísticas.

3.5.8 OVSDB Neutron

Una vez recibidas las llamadas a la API de Neutron Northbound, el componente *net-virt* de OVSDB se encarga de configurar y gestionar los flujos de los Open vSwitches de la red situados en los nodos del entorno OpenStack a través de plugins en la interfaz southbound, como OVSDB y OpenFlow.

En la siguiente figura se muestra la interacción de OpenStack con el servicio OVSDB Neutron, que se ejecuta dentro de OpenDaylight, y el reparto de acciones destinadas a las interfaces de Open vSwitch:

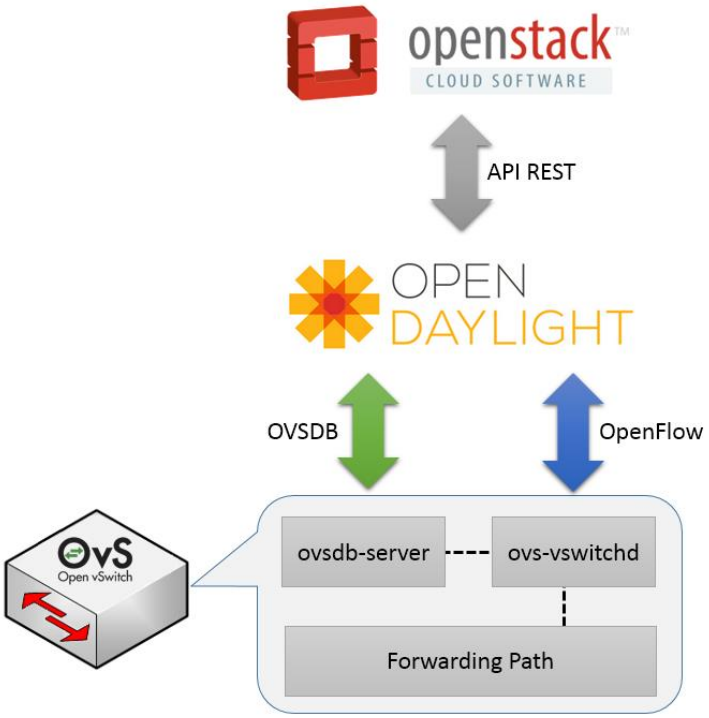

Figura 3.11. Interfaces OpenStack - OpenDaylight - Open vSwitch

3.6 Integración con OpenStack

La utilización de la plataforma OpenDaylight en un entorno de computación en la nube, como OpenStack, es uno de los casos de uso más relevantes del controlador ya que simplifica la gestión de la red.

La amplia variedad de interfaces southbound de OpenDaylight le permite gestionar, tanto infraestructura virtual como física de un centro de datos, y proporcionar servicios de red avanzados. De esta forma se pretende que con la utilización de un controlador SDN en un centro de datos se proporcione escalabilidad y flexibilidad a las redes, y sean prestadas como un servicio más en un entorno de computación en la nube.

3.6.1 OpenStack

Es un sistema de computación en la nube que permite el control de recursos de computación, almacenamiento y red en un centro de datos [19], proporcionando una infraestructura como servicio (IaaS). Está compuesto por un conjunto de proyectos que se encargan de realizar una cierta función para soportar el despliegue de la infraestructura en su conjunto. Cada proyecto realiza un rol específico en la arquitectura y la comunicación entre los diferentes módulos se realiza a través de una API de acceso al servicio.

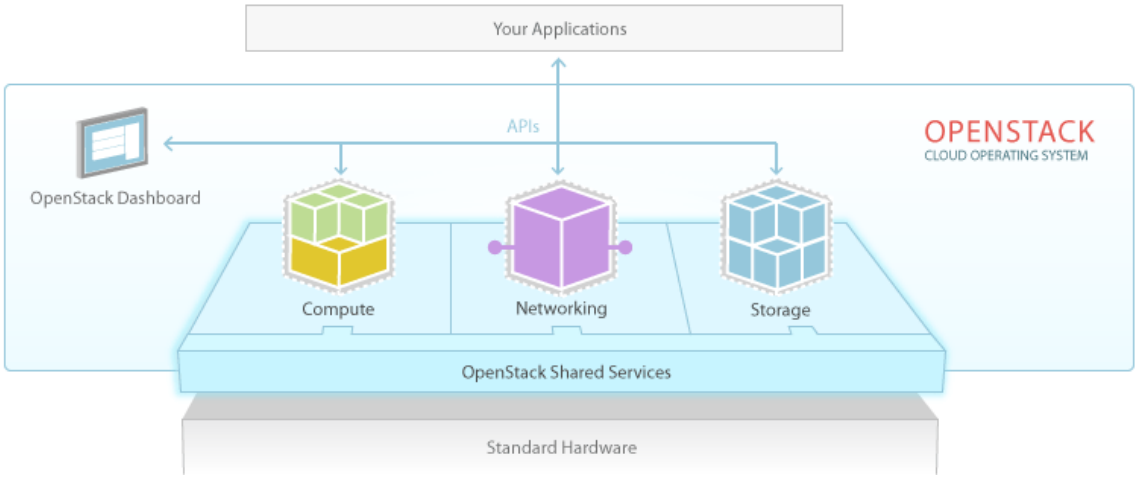

Figura 3.12. Diseño de la arquitectura de OpenStack [19]

Desde su lanzamiento en 2010 promovido por Rackspace Cloud y la NASA se han añadido nuevos proyectos continuamente hasta la última versión publicada en abril de 2015 conocida como Kilo, que está compuesta en total por 16 proyectos. El objetivo de esta plataforma es construir un referente de código abierto que responda a las necesidades en la creación de nubes públicas y privadas independientemente de su tamaño.

Los principales componentes del proyecto OpenStack son siete, donde se muestran sus interacciones en la figura 3.13, y a continuación una descripción de cada uno de ellos:

- Identity (Proyecto Keystone): Proporciona autenticación y autorización para la utilización de los diferentes proyectos.
- Compute (Proyecto Nova): Es el sistema de nodos de computación que se encarga del aprovisionamiento de máquinas virtuales. Soporta diferentes tecnologías de virtualización como KVM, Xen o Hyper-V entre otras.
- Image Management (Proyecto Glance): Almacena un catálogo de imágenes virtuales que serán utilizadas por los nodos de computación
- Dashboard (Proyecto Horizon): Es la aplicación web que permite realizar la mayor parte de las actividades dentro del entorno OpenStack de forma gráfica.
- Object Store (Proyecto Swift): Es el responsable de asegurar la información a través de un sistema de almacenamiento redundante y escalable.
- Block Storage (Proyecto Cinder): Proporciona el almacenamiento de dispositivos de bloque a los servidores virtuales de la nube.
- Networking (Proyecto Neutron): Es el sistema encargado de la gestión de la red en la nube OpenStack.

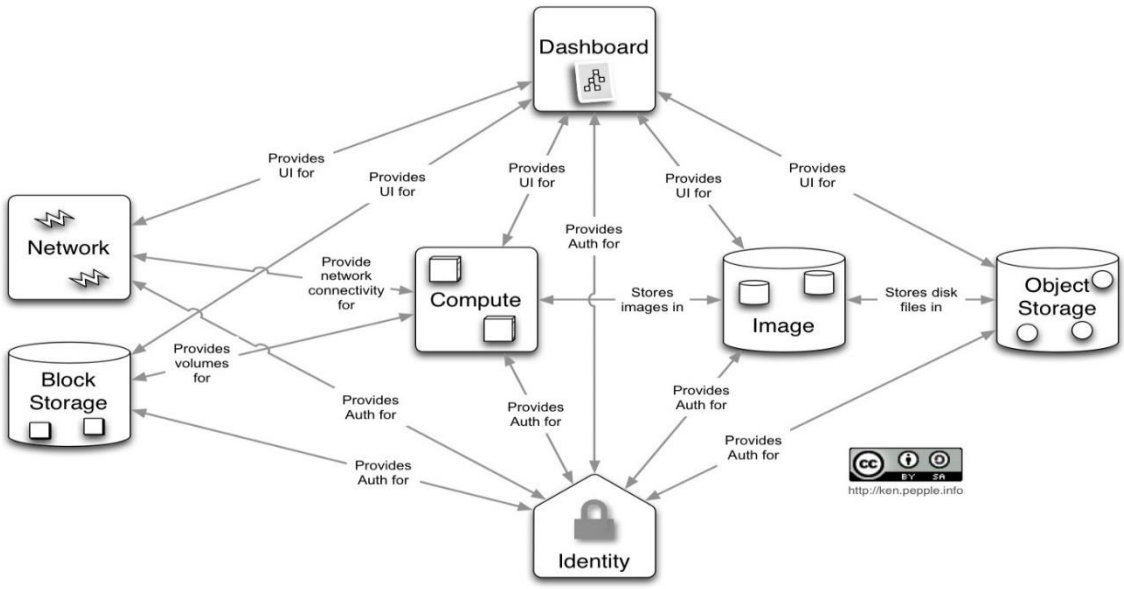

Figura 3.13. Interacciones entre los componentes de OpenStack [20]

Cada uno de los componentes anteriormente mencionados dispone de una API REST que permite la comunicación entre ellos y dota OpenStack como un entorno extensible y adaptable a los diferentes entornos. La excepción de estos componentes es Horizon, que a través del uso de un navegador web abstrae el uso de dichas APIs a los usuarios.

3.6.2 Gestión de la red

El presente trabajo se ha centrado en el proyecto Neutron que es el encargado de la gestión de la red y el responsable de la comunicación con controladores SDN como OpenDaylight. Dicho proyecto está compuesto por un conjunto de plugins que proporcionan servicios de red como la gestión de tráfico en nivel 2 y nivel 3 del modelo OSI, y otros más complejos como firewall, balanceo de carga o VPN.

El plugin ML2 (Modular Layer 2) permite a OpenStack Neutron la utilización de varias tecnologías de red de nivel 2 simultáneamente. Fue implantado en la versión OpenStack Havana publicada en octubre de 2013, y hasta entonces no era posible manejar a la vez varios plugins de nivel 2, como era el caso de los plugins Open vSwitch, Linux bridge e Hyper-V. Dada su arquitectura modular ha depreciado tales plugins reemplazándolos por controladores que soporta ML2 y ha simplificado el añadido de soporte a nuevas tecnologías, como es el caso de la comunicación con dispositivos hardware y controladores SDN. Se disponen de los siguientes controladores:

- Controladores de tipo de red *(Type Drivers)*: Mantienen el estado de la red, llevan a cabo la asignación de redes de los usuarios, y validan los proveedores de red. Actualmente se incluyen controladores para redes de tipo:
  - Local: Consiste en un segmento de conectividad entre máquinas virtuales y otros dispositivos que se encuentran en el mismo nodo de computación. No permite la conexión de dispositivos en diferentes nodos.
  - Flat: Proporciona conectividad entre dispositivos a través de una red física sin ninguna opción de segmentación, como el etiquetado VLAN o técnicas de tunelado. Solo puede existir una red flat por cada red física disponible.
  - VLAN: Permite la segmentación de una red física a través del etiquetado que proporciona la cabecera IEEE 802.1Q. De esta forma es posible la existencia de hasta 4094 redes VLAN en una única red física.
  - GRE: Consiste en una técnica de tunelado que encapsula el tráfico entre máquinas de diferentes nodos de computación. Para la distinción de las diferentes redes se utiliza un identificador de túnel.
  - VXLAN: Es otra posible técnica de tunelado en las redes físicas como sucede con el tipo de red anterior. VXLAN realiza la encapsulación de tramas de nivel 2 dentro de paquetes UDP que utilizan el puerto destino 4789.

- Controladores de mecanismo de red *(Mechanism Drivers)*: Son los responsables de tratar la información suministrada por los controladores de tipo de red y asegurarse que es aplicada según el mecanismo seleccionado. Proporcionan el acceso a la red y para ello utilizan agentes situados en los nodos de computación a través de llamadas RPC o interactúan con dispositivos externos o controladores. Pueden ser utilizados simultáneamente para acceder a diferentes puertos de una misma red virtual, y tratan tecnologías basadas en agentes (Hyper-V, Linux bridge, Open vSwitch), basadas en controladores (Tail-f NCS, OpenDaylight), y de dispositivos hardware de fabricantes (Arista, Cisco Nexus).

El controlador de mecanismo de red OpenDaylight (OpenDaylight ML2 Mechanism Driver) consiste en un proxy REST que reenvía las llamadas de la API de Neutron al controlador OpenDaylight, que contiene un servicio REST en la interfaz northbound denominado OpenStack Neutron. Este es el encargado de trasladar las llamadas realizadas a la API a los necesarios servicios de OpenDaylight.

Además de ML2 en Neutron, existen plugins que proporcionan otras funciones de red, como gestión de tráfico en nivel 3, balanceo de carga o firewall como servicio. Dichos plugins también se encuentran en desarrollo para que puedan interactuar con OpenDaylight y sea este quien gestione dichos servicios.

En la siguiente figura se muestra la arquitectura completa de comunicación entre las dos plataformas, desde un plugin situado en OpenStack Neutron con OpenDaylight:

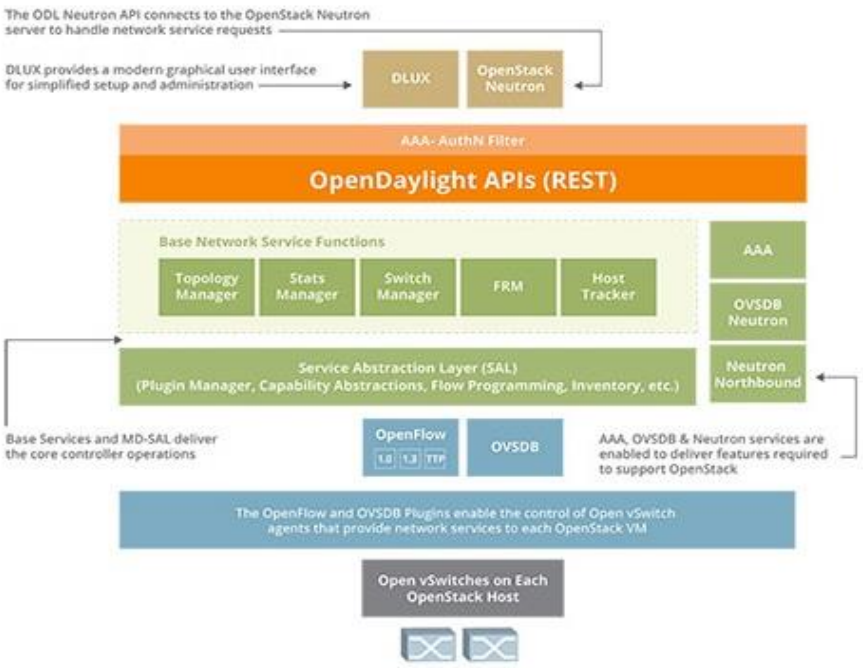

Figura 3.14. Arquitectura de comunicación entre OpenDaylight y OpenStack [12]

Para el caso de uso mencionado es necesario que se implementen como mínimo los servicios de AAA, Neutron Northbound y OVSDB Neutron. Este último se encarga desde el aislamiento de las redes de los diferentes usuarios, utilizando túneles GRE o VXLAN mediante el protocolo OVSDB, hasta la programación de flujos a través de OpenFlow en los switches distribuidos por el entorno OpenStack.

En un entorno OpenStack en el que no se utiliza OpenDaylight como controlador SDN es necesario gestionar numerosas interfaces y dispositivos virtuales de red en los nodos, como se aprecia en la siguiente imagen:

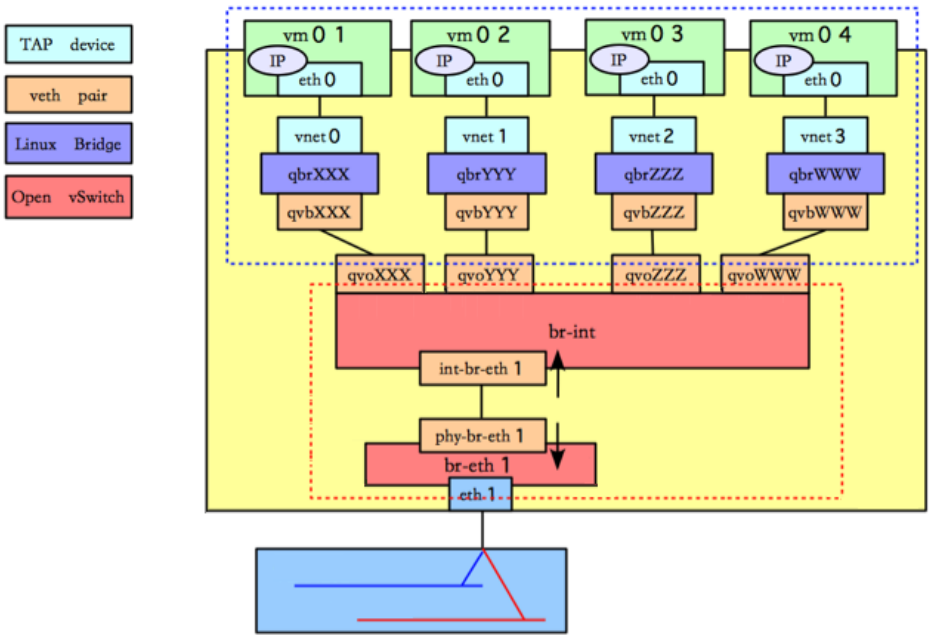

Figura 3.15. Diagrama de red en un nodo de computación [20]

Se utilizan dos interfaces de red (*TAP* y *veth*) y dos switches virtuales (Linux Bridge y Open vSwtich): Las interfaces *TAP* son utilizadas por hipervisores, como KVM, para implementar una interfaz de red en un sistema operativo huésped y las interfaces *veth* son un par de interfaces directamente conectadas que son utilizadas principalmente para conexiones entre switches virtuales. Los switches Linux Bridge se comportan como un switch de nivel 2 que realiza aprendizaje a través de la dirección MAC, y los Open vSwitch permiten una mayor configuración en sus puertos como si de un switch físico se tratara, y así aplicar configuraciones de VLAN o túneles.

Por cada máquina virtual creada es necesaria la creación de un Linux Bridge para aplicar reglas *iptables* gestionadas por los grupos de seguridad a las interfaces TAP, ya que si se encuentran conectadas al bridge de integración *br-int* directamente no es posible al tratarse de un Open vSwitch.

En definitiva, la gestión de la red se convierte en algo complejo para controlar si se posee un entorno OpenStack con varios nodos de computación y numerosos usuarios. Por ello las SDN es una solución a este posible problema, y el diagrama de red anterior visto en un nodo de computación evolucionaría a algo más flexible y sencillo como se muestra en la siguiente figura:

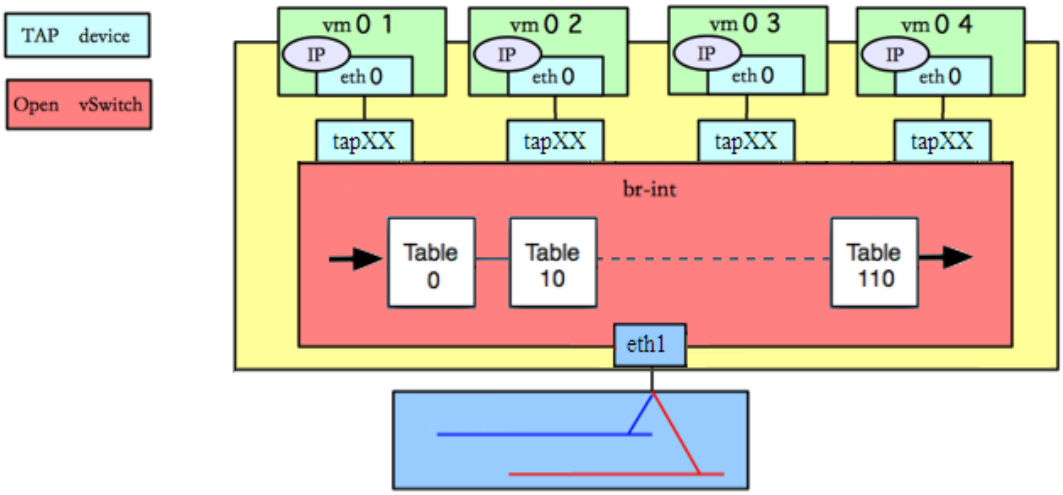

Figura 3.16. Diagrama de red en un nodo de computación con OpenDaylight

En este nuevo modelo se pretende que un único Open vSwitch por nodo de computación compuesto por diferentes tablas de flujo gestione todo el tráfico local, y sus flujos sean controlados por OpenDaylight a través de OpenFlow. De esta forma se consigue que todos los flujos pasen a través de un *pipeline* en el switch, como se mencionó en el *Apartado 2.2: OpenFlow*, y se realicen un conjunto de acciones según se considere. Desde el primer lanzamiento de OpenDaylight se ha trabajado en esta idea, y en Helium ha evolucionado a un conjunto de doce tablas [21]. Cada una de las tablas de flujos creadas se corresponde con un servicio de red, dando como resultado:

- Tabla 0 - Clasificador: Se encarga de añadir un campo de metadatos al flujo, denominado *REG0*, que permite identificar durante el *pipeline* si proviene de una máquina virtual local al nodo de computación *(REG0=1)* o exterior a él *(REG0=2)*. Si es un tráfico local, conocidas la dirección MAC y puerto, se agrega el campo *tun_id* que identifica red virtual.
- Tabla 10 - Director: Se encarga de alterar el natural flujo dentro del *pipeline* para mejorar el tratamiento de flujos. El funcionamiento de esta tabla se encuentra todavía por implementar en el futuro.
- Tabla 20 - Contestador ARP: Con esta tabla se optimiza la utilización de los enlaces entre nodos ya que se evita enviar paquetes ARP broadcast por la red.

- Tabla 30 - DNAT: Se encarga de modificar la dirección IP destino donde se dirige la conexión. Es utilizado en el tráfico de entrada a máquinas virtuales que disponen de una IP flotante.
- Tabla 40 - Control de acceso de salida: Aplica reglas de filtrado al tráfico saliente de máquinas virtuales.
- Tabla 50 - Balanceador de carga: Proporciona el servicio de balanceo de carga entre máquinas virtuales a partir de la creación de un virtual IP.
- Tabla 60 - Enrutamiento virtual distribuido (DVR): Permite a los paquetes intercambiarse entre diferentes redes virtuales del entorno OpenStack. A efectos prácticos se comporta como un router ya que interconecta redes, decrementa TTL y reescribe la dirección MAC.
- Tabla 70 - Reenvío L3: Junto con la tabla anterior se encarga de completar la acción de enrutamiento. En este caso se modifica la dirección MAC destino dependiendo de la dirección IP destino del paquete.
- Tabla 80 - Reescritura de L2: Proporciona un servicio de modificación de la cabecera de nivel 2. La implementación de este servicio es utilizado como prototipo para el desarrollo de los demás, ya que muestra las relaciones entre clases Java y el registro de servicios OSGi.
- Tabla 90 - Control de acceso de entrada: Aplica reglas de filtrado al tráfico entrante a las máquinas virtuales.
- Tabla 100 - SNAT: Se encarga de modificar la dirección IP origen de donde proviene la conexión. Se utiliza para el acceso a redes externas.
- Tabla 110 - Reenvío L2: A partir del campo de metadatos *REG0* y de la dirección MAC destino se reenvía el paquete a los destinos apropiados. Se manejan tanto flujos de paquetes dirigidos a un único destino (unicast) como a varios destinos (multicast y broadcast).

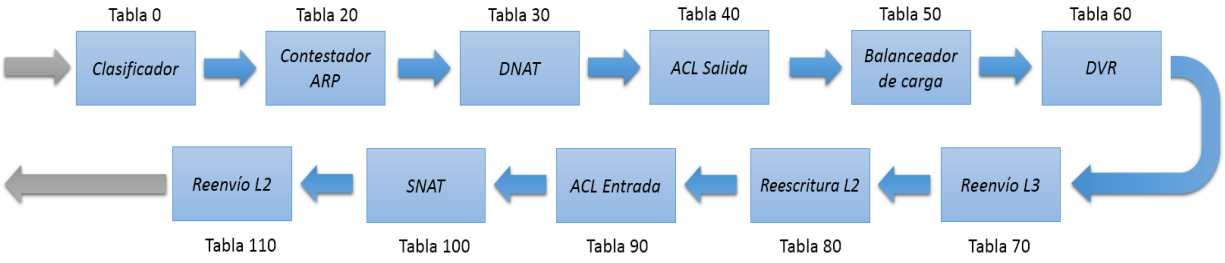

Figura 3.17. Pipeline OpenFlow en OpenStack

Por ejemplo, para el envío de paquetes entre máquinas de una misma una red local solamente es necesario utilizar la tabla 0 de clasificación y la tabla 110 para el reenvío en L2. En el siguiente diagrama de flujos se muestran los diferentes estados del *pipeline*:

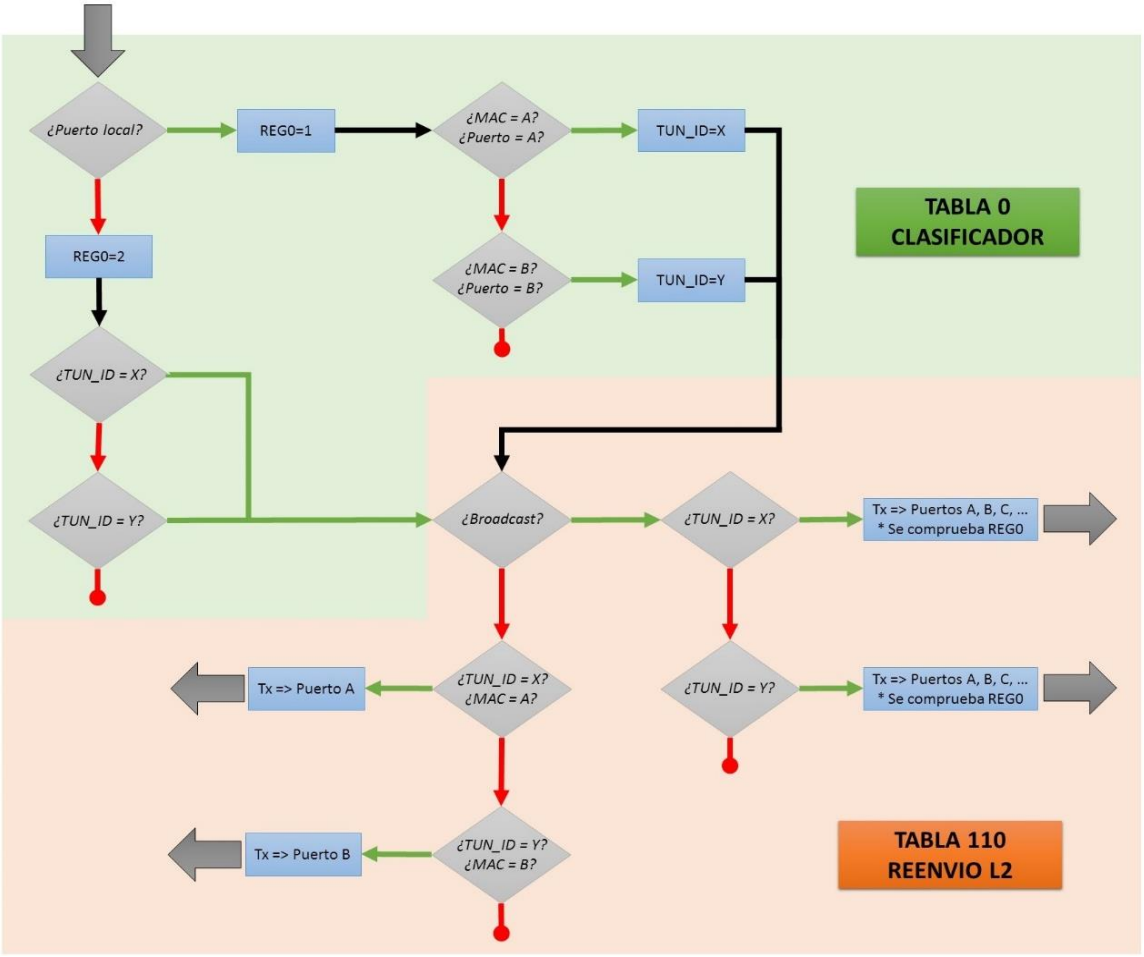

Figura 3.18. Diagrama de flujos del *pipeline* para el reenvío L2

En el *Apartado 4.4 Escenario 2: Entorno OpenStack* se muestra un escenario de ejemplo en el que se comprueban los conceptos explicados, entre ellos el reenvío a nivel 2, y se puede comprender en mayor detalle el funcionamiento del *pipeline* dentro de los switches situados en los nodos de computación de un entorno de computación en la nube basado en el proyecto OpenStack.

4 Escenarios de demostración

A lo largo de este capítulo se pretende aplicar los conocimientos adquiridos sobre las Redes Definidas por Software y OpenDaylight, y desarrollar un conjunto de escenarios virtuales que muestren casos de uso relevantes. El primero de ellos ha consistido en el desarrollo de una aplicación que interactúe con la plataforma a través de la interfaz northbound, y el otro se ha centrado en la automatización del despliegue de un entorno de computación en la nube basado en OpenStack con OpenDaylight como gestor de red.

4.1 Herramientas utilizadas

Para el despliegue de los escenarios propuestos más adelante se han utilizado un conjunto de herramientas que automatizan la creación de topologías de red, como Virtual Networks over LinuX (VNX) o Mininet, y el desarrollo de un entorno OpenStack, mediante DevStack.

Asimismo, para el desarrollo de los escenarios se ha trabajado sobre el sistema operativo Ubuntu Vivid Vervet (15.04) en un ordenador portátil que posee un procesador Intel Core i7-4710HQ @ 2.50 Ghz y 16 GB de memoria RAM.

4.1.1 Virtual Networks over Linux

Es una herramienta de virtualización de propósito general para la creación de escenarios de forma automática. Es de código abierto y ha sido desarrollada por el Departamento de Ingeniería Telemática de la Universidad Politécnica de Madrid [22]. Permite la definición de máquinas virtuales con los sistemas operativos más usuales, como Linux (en sus distintas distribuciones: Ubuntu, Fedora, etc.), Windows o FreeBSD, así como la capacidad de emular routers comerciales (Cisco mediante Dynamips y Juniper mediante Olive). Adicionalmente VNX proporciona la posibilidad de desplegar los escenarios de red virtuales sobre clústeres de servidores federados, distribuyendo la carga entre los distintos servidores y permitiendo con ello la creación de escenarios de gran tamaño.

Para su utilización es necesario definir una plantilla en lenguaje XML con la descripción del escenario virtual a crear mediante el lenguaje de especificación VNX, incluyendo características de máquinas virtuales y especificaciones de las redes virtuales. Una vez originada, el programa VNX la analizará y creará el entorno virtual descrito. La especificación de la plantilla seguirá el siguiente formato:

```
<?xml version="1.0" encoding="UTF-8"?>
<vnx>
      definiciones globales: <global>
      definiciones de redes virtuales: <net>
      definiciones de máquinas virtuales: <vm>
      definiciones del equipo anfitrión: <host>
</vnx>
```

Una vez realizada la instalación de VNX, que se encuentra descrita en detalle en el portal de la herramienta[2], será necesario descargar alguno de los sistemas de ficheros preconfigurados desde su repositorio[3], que siguen la siguiente convención de nombres: vnx_rootfs_<virtplatform>_<osname>-<osversion>-[gui-]<rootfsversion>.<ext>

Es importante recordar a lo largo del trabajo que los sistemas de ficheros descargados disponen de los usuarios "*root*" y "*vnx*", con contraseña "*xxxx*" en ambos.

Las máquinas virtuales pueden utilizar el sistema de ficheros que se ha descargado de dos formas posibles, en modo directo o en modo COW (Copy-On-Write). Cuando una máquina virtual utiliza el sistema de ficheros en modo directo solo podrá ser usado un sistema de ficheros por máquina virtual, por lo que para un escenario con numerosas máquinas virtuales es necesario disponer de numerosas copias de sistemas de ficheros.

La opción de Copy-On-Write, que será la utilizada en los escenarios de demostración, permite que varias máquinas virtuales puedan utilizar el mismo sistema de ficheros. En este caso se devuelve punteros a un mismo sistema de ficheros, y en el momento en que un proceso intenta modificar algún fichero del sistema, se crea una copia para prevenir que los cambios producidos por dicho proceso sean visibles por todos los demás. Todo ocurre de forma transparente para los procesos, y si ninguno de ellos realiza modificaciones en el sistema de ficheros, no se creará ninguna copia adicional del recurso. De esta forma se dispone de un sistema de ficheros de sólo lectura, y del directorio (read and write) donde se encontrarán las modificaciones del sistema de ficheros. Además se dispondrá de un directorio en el que se encuentran punteros del sistema de ficheros completo de la máquina virtual. En el siguiente diagrama se muestra un ejemplo de la arquitectura de directorios del sistema de ficheros en modo COW:

[2] http://web.dit.upm.es/vnxwiki/index.php/Vnx-install

[3] http://vnx.dit.upm.es/vnx/filesystems/

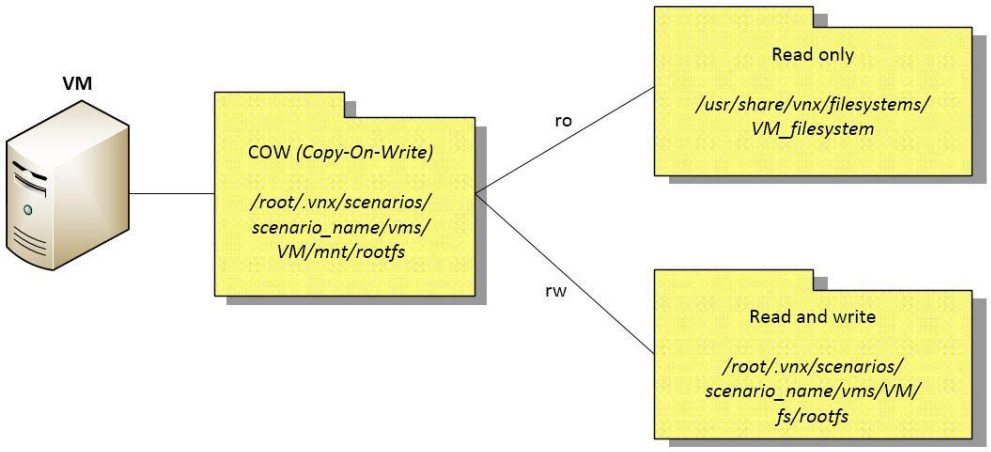

Figura 4.1. Sistema de ficheros en modo COW (Copy-On-Write)

El último paso para el funcionamiento del escenario es arrancarlo a través de la herramienta VNX, la cual ofrece múltiple funcionalidades. Los comandos a ejecutar tendrán la siguiente estructura:

```
# vnx -f VNX_file --ACTION [options][-M VM_LIST]
```

- *VNX_FILE* es la plantilla en lenguaje XML creada para el escenario.
- *ACTION* es una de las diferentes acciones disponibles de la herramienta.
- *VM_LIST* es una de las posibles opciones en el que la acción a ejecutar sólo se producirá en la lista de máquinas virtuales indicadas del escenario.

A continuación se muestra el diagrama de estados en el que pueden estar las diferentes máquinas virtuales de un escenario en concreto a través de la herramienta VNX, y sus diferentes acciones de transición:

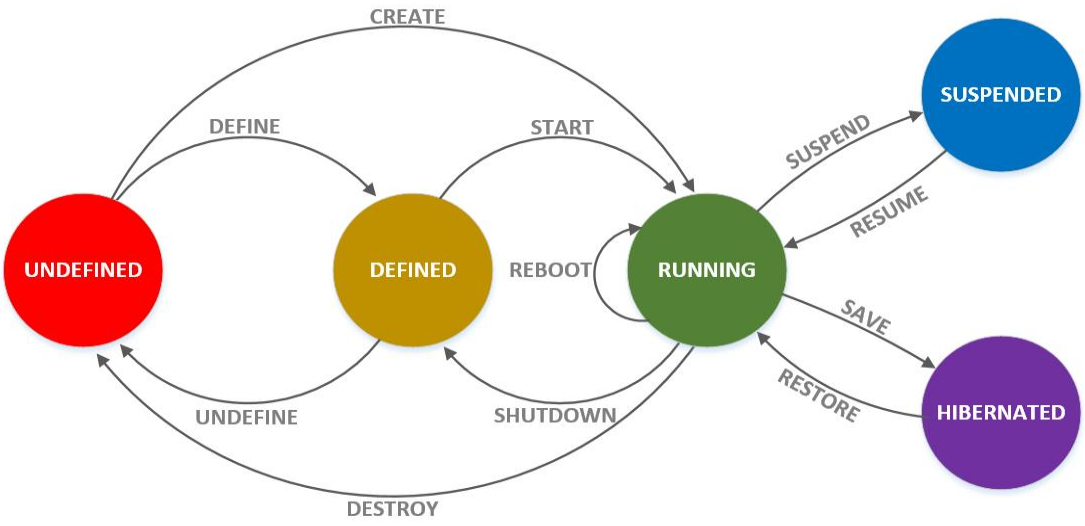

Figura 4.2. Diagrama de estados de las máquinas virtuales en VNX

Para cambiar de estado las máquinas virtuales del escenario, y realizar acciones como *create*, *destroy*, *shutdown*, *start* y *reboot*, se utilizarán los siguientes comandos:

```
# vnx –f VNX_file --create
# vnx –f VNX_file --destroy
# vnx –f VNX_file --start
# vnx –f VNX_file --shutdown
# vnx –f VNX_file --reboot
```

Cabe mencionar la opción *–n* al utilizar la acción *create* o *start*, que permite ocultar las consolas de las distintas máquinas virtuales arrancadas.

También existen otros comandos de interés de la herramienta VNX a utilizar cuando se encuentra el escenario arrancado, como por ejemplo:

- Ejecutar comandos indicados con una cierta etiqueta de la plantilla.

  ```
  # vnx –f VNX_file --execute TAG [-M VM_LIST]
  ```

- Mostrar las consolas de un conjunto de máquinas virtuales del escenario.

  ```
  # vnx –f VNX_file --console [-M VM_LIST]
  ```

- Mostar el estado en el que se encuentran las máquinas virtuales.

  ```
  # vnx –f VNX_file --show-status [-M VM_LIST]
  ```

- Mostrar un diagrama con las conexiones de red construidas en el escenario.

  ```
  # vnx –f VNX_file --show-map
  ```

4.1.2 Mininet

Es una plataforma que permite la creación de topologías de red virtuales compuestas por equipos finales, switches y controladores en una única máquina física o virtual [23]. La principal ventaja de utilizar esta herramienta es la rápida formación de ciertas topologías utilizando únicamente un comando. Es una herramienta fácil de manejar y permite crear un entorno en cuestión de segundos para experimentar con Redes Definidas por Software basadas en OpenFlow.

Desde su página oficial[4] se explican diferentes alternativas para su instalación, desde la descarga de una máquina virtual preconfigurada hasta la instalación nativa partiendo del código fuente.

Aunque también es posible crear topologías personalizadas según lo desee el usuario, algunas topologías preconfiguradas que pueden ser reproducidas rápidamente son:

- Topología *single*: Se trata de un único switch conectado a N equipos finales.

```
# mn --topo=single,N
```

- Topología *linear*: Se forma una cadena con N switches, de los cuales cada uno de ellos tiene conectado un equipo final.

```
# mn --topo=linear,N
```

- Topología *tree*: Crea una topología de árbol en la que es posible especificar la profundidad N de la misma.

```
# mn --topo=tree,N
```

Dada una cierta topología se le puede añadir un controlador que será el encargado de administrar los flujos de la red. Es posible utilizar un controlador gestionado por Mininet o indicar la dirección IP y puerto de un controlador gestionado por el usuario. En este último caso se utilizará el siguiente comando:

```
# mn --controller=remote,ip=127.0.0.1,port=6653
```

4.1.3 DevStack

DevStack es un conjunto de scripts escritos en Bash que permite automatizar el despliegue de un entorno OpenStack de desarrollo en una máquina física o virtual [24]. Es una excelente opción para aquellos usuarios que quieren realizar una toma de contacto con OpenStack y para desarrolladores que quieren construir prototipos y realizar pruebas en un entorno completo con las últimas actualizaciones.

Desde su comienzo en 2011 por la OpenStack Foundation, DevStack ha ido evolucionando, y en la actualidad permite un gran número de opciones de configuración, así como la utilización de servicios externos, como es el caso de OpenDaylight. Para ello pone a disposición un mecanismo de plugins que permite añadir fácilmente el apoyo a proyectos y características adicionales.

[4] http://mininet.org/download

Para descargar DevStack es necesario disponer del paquete Git instalado en el sistema operativo, y realizar la clonación de su repositorio:

```
$ git clone https://git.openstack.org/openstack-dev/devstack
```

La rama master del repositorio de DevStack utiliza las versiones de desarrollo de los componentes de OpenStack, por lo que si se desea crear un entorno con versiones estables es necesario cambiar de rama a *stable/[release]*, por ejemplo:

```
$ git checkout stable/kilo
```

DevStack trata de ser funcional con una mínima configuración, pero dada la gran cantidad de características que componen los diferentes proyectos de OpenStack, es posible personalizar el archivo de configuración *local.conf* con las variables de entorno que son utilizadas en la herramienta. Tal fichero de configuración seguirá una estructura de variable-valor por secciones, como se muestra en el siguiente ejemplo:

```
[[local|localrc]]
VARIABLE_1 = value_1
VARIABLE_2 = value_2

[[post-config|<config-file-name>]]
VARIABLE_3 = value_3
VARIABLE_4 = value_4
```

Una de las configuraciones que puede llevarse a cabo mediante la personalización del fichero *local.conf*, es la configuración multi-nodo. Con la activación / desactivación de ciertos servicios, en diferentes nodos que están ejecutando DevStack, es posible crear un nodo controlador conectado con múltiples nodos de computación que albergarán las máquinas virtuales. Esta será la configuración que se lleve a cabo en el *Apartado 4.4 Escenario 2: Entorno OpenStack*.

Una vez definida la configuración de DevStack solamente falta ejecutar el script *stack.sh,* sin permisos de *root*, y esperar a que finalice mostrando un mensaje satisfactorio como el siguiente:

```
$ ./stack.sh

This is your host ip: 10.0.0.11
Horizon is now available at http:// 10.0.0.11/
Keystone is serving at http:// 10.0.0.11:5000/v2.0/
The default users are: admin and demo
The password: password
```

4.2 Instalación de OpenDaylight

Las distribuciones estables de la plataforma OpenDaylight pueden ser descargadas desde la página de descarga de software [5]. La última versión disponible, publicada en junio de 2015, es Lithium y se puede descargar a través de un archivo comprimido de 220 MBytes de tamaño. En este caso la distribución se encuentra sin ningún módulo instalado por defecto, pero se podrán instalar más adelante aquellos que se deseen.

También existe la opción de descargar el código fuente de un determinado proyecto desde su repositorio y compilarlo. En este caso se puede clonar el repositorio con una cuenta OpenDaylight o de forma anónima. Para más información sobre esta alternativa es recomendable seguir los capítulos *1. Getting started with Git and Gerrit* y *2. Pulling and Pushing the Code from the CLI* de la Guía de Desarrollo [13].

Una vez descomprimido el archivo descargado desde la página de descarga de software es necesario disponer de Java 7 JDK y Apache Maven para la construcción y ejecución de la plataforma. Para ello hay que instalar los siguientes paquetes:

```
# apt-get install openjdk-7-jre openjdk-7-jdk maven
```

Para utilizar la distribución descargada se ejecuta Karaf:

```
# ./bin/karaf

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

La primera vez que se ejecuta Karaf es necesario disponer de conexión a Internet, y una vez arrancado se pueden instalar los módulos que se deseen. En la Guía de Instalación [25] se muestran los módulos que pueden ser instalados, así como cuestiones de compatibilidad entre ellos. Para instalarlos hay que ejecutar el siguiente comando:

```
opendaylight-user@root> feature:install <feature1> <feature2> ...
```

[5] http://www.opendaylight.org/software/downloads

Se puede mostrar una lista completa con los módulos de que pueden instalar a través del comando *feature:list*. Para comprobar los módulos instalados se debe añadir la opción *-i*, y para especificar la búsqueda es posible utilizar el comando *grep* como se muestra a continuación:

```
opendaylight-user@root> feature:list [-i][|grep <feature>]
```

Para mostrar logs en la consola de Karaf es necesario activarlos a través de alguno de los siguientes comandos:

```
opendaylight-user@root> log:display
opendaylight-user@root> log:tail
```

Además de arrancar el controlador a través de Karaf en modo interactivo, también es posible ejecutarlo en segundo plano e interactuar con él a través de:

```
# ./bin/start    => Arranca Karaf como demonio
# ./bin/status   => Muestra el estado de Karaf
# ./bin/client   => Accede a la consola de Karaf
# ./bin/stop     => Detiene Karaf
```

4.3 Escenario 1: Ejemplo de uso de APIs REST

En este escenario se pretende utilizar la plataforma OpenDaylight a través del nivel de aplicación de la arquitectura SDN. Para ello se hará uso de la interfaz gráfica de gestión DLUX, y se desarrollará una aplicación que controle el acceso de los equipos finales a la red, así como con capacidad para bloquear equipos no deseados.

Hasta ahora, en las redes tradicionales era necesario mantener una extensa documentación de la red con información sobre routers, switches y equipos conectados a cada puerto que facilitara la resolución de problemas. De esta forma, si se disponía de una red con un gran número de dispositivos la tarea de documentación era compleja, así como el entendimiento por parte de un nuevo administrador de la red. En una Red Definida por Software, donde el control se encuentra centralizado, esta tarea se simplifica, y es inmediato disponer de un mapa con la topología de la red que se está gestionando e información sobre todos sus dispositivos en tiempo real.

Si no se tiene un adecuado control de acceso de los equipos finales a la red también puede ser un problema. Con un enfoque SDN el controlador tiene toda la información sobre los dispositivos conectados a la red, por lo que puede permitir o denegar el tráfico de ciertos equipos según crea conveniente.

Para resolver el control de acceso caso se ha desarrollado una aplicación, implementada como un script en Python, que mediante la utilización de las APIs REST situadas en la interfaz northbound de la plataforma OpenDaylight puede obtener información sobre todos los dispositivos de la red y denegar el tráfico de aquellos equipos no deseados.

4.3.1 Descripción del escenario

Para las pruebas realizadas se ha creado un escenario a través de la herramienta VNX con un sistema de ficheros basado en LXC (LinuX Containers) [26], que es una tecnología de contenedores que permite crear sistemas Linux contenidos dentro de otro aplicándole algunos límites de CPU, memoria o incluso asignarle interfaces de red, tratándolo como si fuera un equipo independiente. La ventaja de LXC respecto a otras tecnologías de virtualización es su ligereza, ya que utiliza recursos mínimos dando así respuestas mucho más rápidas.

OpenDaylight se ejecutará en el equipo anfitrión, y será necesario instalar los siguientes módulos a través de la consola Karaf:

```
opendaylight-user@root> feature:install odl-dlux-all odl-restconf
                        odl-l2switch-switch
```

Los switches virtuales creados por la topología se comunicarán con el controlador a través de 127.0.0.1:6653. En el *Apéndice A.1: Plantilla del escenario* se muestra la plantilla completa del escenario, que puede ser ejecutada para crear el escenario con el siguiente comando:

```
# vnx –f Demo_Scenario.xml -t -n
```

4.3.2 DLUX

Con el escenario creado y OpenDaylight ejecutándose, se puede acceder a la interfaz gráfica de gestión DLUX a través de: http://localhost:8181/index.html. El nombre de usuario y contraseña por defecto son ambos *admin*.

La primera visión que se ofrece es la topología de red creada con todos sus nodos. Para descubrir los equipos finales conectados es necesario generar tráfico desde ellos, por lo que se ha incorporado a la plantilla del escenario un comando que envía pings desde cada equipo. Para ello basta con ejecutar el siguiente comando:

```
# vnx –f Demo_Scenario.xml -x pingall
```

De esta forma es posible ver la topología al completo, como se muestra en la figura 4.3, y cualquier administrador de red es capaz de descubrir gráficamente nodos y equipos finales en la red.

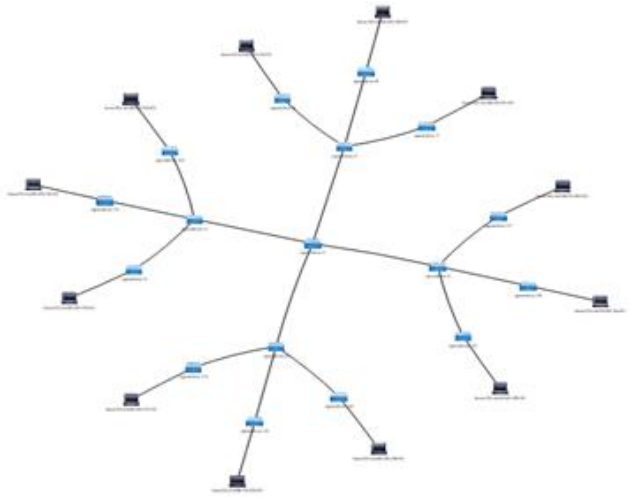

Figura 4.3. Topología de red de estrella extendida

DLUX proporciona otras opciones a través del panel izquierdo, como información sobre los nodos de la topología en el apartado *Nodes*. En dicho apartado se muestra un listado con todos los nodos de la topología y se ofrece la posibilidad de listar los puertos de cada nodo y recopilar estadísticas sobre ellos.

4.3.3 Aplicación de control de acceso a la red

La interfaz northbound de la plataforma OpenDaylight permite la comunicación con aplicaciones a través de llamadas REST, las cuales pueden gestionar la red. Aprovechando esta característica de las SDN se ha desarrollado una aplicación, a través de un script en Python, que puede obtener información sobre los nodos y equipos finales de la red y permitir o denegar el tráfico de aquellos equipos no autorizados. El código de la aplicación se puede encontrar adjunto en *Apéndice A.2 Aplicación de control de acceso a la red*. Para ejecutar la aplicación es necesario tener instalado Python en el equipo:

```
$ python NetworkAccessControlApplication.py

|-----------------------------------------|
|--- Network Access Control Application ---|
|-----------------------------------------|

[1] Show topology information
[2] Show blocked hosts
[3] Block a specific host
[4] Unblock a specific host
[q] Quit

Select an option:
```

La aplicación proporciona en su inicio un menú en el que el usuario puede mostrar información sobre la topología o sobre los equipos finales bloqueados, así como realizar acciones de bloqueo y desbloqueo de ciertos equipos finales.

Si se selecciona la opción 1, *Show topology information*, se indica al usuario el número de switches y equipos finales en la red, así como información más detallada sobre ellos en forma de tabla, como se muestra a continuación:

```
Select an option: 1

   => Number of switches:  17
   => Number of hosts:  12

   |--------------------|
   |---- SWITCH LIST ----|
   |--------------------|
   |     openflow:16     |
   |     openflow:17     |
   |     openflow:12     |
   |     openflow:13     |
   |     openflow:14     |
   |     openflow:15     |
   |     openflow:11     |
   |     openflow:10     |
   |     openflow:4      |
   |     openflow:3      |
   |     openflow:2      |
   |     openflow:1      |
   |     openflow:7      |
   |     openflow:8      |
   |     openflow:5      |
   |     openflow:6      |
   |     openflow:9      |
   |--------------------|

   |-------------------------------------------------------------|
   |-------------------------- HOST LIST -------------------------|
   |-------------------------------------------------------------|
   |         MAC         |      IP      |    SWITCH         |   PORT  |
   |                     |              |                   |         |
   |  02:fd:00:05:04:01  |  10.0.0.5    |  openflow:10      |    2    |
   |  02:fd:00:05:01:01  |  10.0.0.2    |  openflow:7       |    2    |
   |  02:fd:00:05:09:01  |  10.0.0.10   |  openflow:15      |    2    |
   |  02:fd:00:05:06:01  |  10.0.0.7    |  openflow:12      |    2    |
   |  02:fd:00:05:05:01  |  10.0.0.6    |  openflow:11      |    2    |
   |  02:fd:00:05:02:01  |  10.0.0.3    |  openflow:8       |    2    |
   |  02:fd:00:05:03:01  |  10.0.0.4    |  openflow:9       |    2    |
   |  02:fd:00:05:08:01  |  10.0.0.9    |  openflow:14      |    2    |
   |  02:fd:00:05:0b:01  |  10.0.0.12   |  openflow:17      |    2    |
   |  02:fd:00:05:0a:01  |  10.0.0.11   |  openflow:16      |    2    |
   |  02:fd:00:05:00:01  |  10.0.0.1    |  openflow:6       |    2    |
   |  02:fd:00:05:07:01  |  10.0.0.8    |  openflow:13      |    2    |
   |-------------------------------------------------------------|
```

La opción 2, *Show blocked hosts*, rastrea los flujos instalados en los switches de la red para mostrar aquellos que se han denegado por el usuario. Si no se ha bloqueado ninguno de ellos se mostrará el siguiente resultado:

```
Select an option: 2

   => Number of blocked hosts:  0

   |-----------------------------------------------------------------|
   |---------------------- BLOCKED HOST LIST -----------------------|
   |-----------------------------------------------------------------|
   |         MAC         |      IP     |    SWITCH         |   PORT  |
   |                     |             |                   |         |
   |-----------------------------------------------------------------|
```

Para comprobar el funcionamiento de la aplicación se selecciona como objetivo el equipo h1, que dispone de las siguientes características en la interfaz de red:

```
root@h1:~# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 02:fd:00:05:00:01
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::fd:ff:fe05:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1909 errors:0 dropped:99 overruns:0 frame:0
          TX packets:114 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:330969 (330.9 KB)  TX bytes:9356 (9.3 KB)
```

Es posible bloquear un equipo no autorizado a la red con la opción 3, *Block a specific host*, proporcionando la dirección MAC, que puede extraerse de la información sobre la topología.

```
Select an option: 3

   Enter the target host to block [mac]: 02:fd:00:05:05:01

   => Host founded in network topology
   ======>    MAC:  02:fd:00:05:00:01
   ======>     IP:  10.0.0.1
   ======> SWITCH:  openflow:6
   ======>   PORT:  2
   => Request to block host has been sended
```

Si se comprueba en este caso el estado sobre los equipos finales bloqueados, se mostrará el equipo h1 como bloqueado:

```
Select an option: 2

   => Number of blocked hosts:  1

   |-----------------------------------------------------------------|
   |---------------------- BLOCKED HOST LIST -----------------------|
   |-----------------------------------------------------------------|
   |         MAC         |      IP      |     SWITCH        |  PORT  |
   |                     |              |                   |        |
   |  02:fd:00:05:05:01  |  10.0.0.6    |   openflow:6      |    2   |
   |-----------------------------------------------------------------|
```

De esta forma se ha creado un flujo en la tabla de flujos del switch donde se encuentra conectado el equipo final que bloqueará todo el tráfico procedente con la MAC objetivo. Se puede comprobar la tabla de flujos del switch openflow:6 (Net6):

```
# ovs-ofctl dump-flows Net6 -O Openflow13
 OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x2b00000000000014, duration=210.631s, table=0, n_packets=42,
  n_bytes=4277,  priority=0 actions=drop
  cookie=0x0, duration=20.116s, table=0, n_packets=14, n_bytes=1036,
  priority=20,dl_src=02:fd:00:05:00:01 actions=drop
  cookie=0x2b00000000000012, duration=202.777s, table=0, n_packets=1656,
  n_bytes=312861, priority=2,in_port=1 actions=output:2
  cookie=0x2b00000000000013, duration=202.773s, table=0, n_packets=30,
  n_bytes=2548, priority=2,in_port=2 actions=output:1,CONTROLLER:65535
  cookie=0x2b00000000000014, duration=210.631s, table=0, n_packets=43,
  n_bytes=3655, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
```

Para desbloquear un equipo final bloqueado previamente es necesario seleccionar la opción 4, *Unblock a specific host*, e indicar la dirección MAC:

```
Select an option: 4

   Enter the target host to unblock [mac]: 02:fd:00:05:00:01

   => Host founded in network topology
   ======>    MAC:  02:fd:00:05:00:01
   ======>     IP:  10.0.0.1
   ======> SWITCH:  openflow:6
   ======>   PORT:  2
   => Request to unblock host has been sended
```

Para verificar que el flujo de bloqueo instalado funciona como es debido, se puede enviar un ping continuo desde el equipo bloqueado momentos previos y posteriores al bloqueo. En el siguiente cuadro se muestra el resultado junto con un posterior desbloqueo a partir de la secuencia 66 del ping:

```
root@h1:~# ping 10.0.0.10

PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=0.300 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.066 ms
From 10.0.0.10 icmp_seq=63 Destination Host Unreachable
From 10.0.0.10 icmp_seq=64 Destination Host Unreachable
From 10.0.0.10 icmp_seq=65 Destination Host Unreachable
64 bytes from 10.0.0.10: icmp_seq=66 ttl=64 time=2000 ms
64 bytes from 10.0.0.10: icmp_seq=67 ttl=64 time=1000 ms
64 bytes from 10.0.0.10: icmp_seq=68 ttl=64 time=0.839 ms
```

Si se utiliza un analizador de tráfico como Wireshark es posible capturar las peticiones y respuestas HTTP que se han utilizado para comunicarse con la APIs de OpenDaylight, que se sitúan en el puerto 8181.

En las opciones 1 y 2, donde se recopila información relativa a la topología de la red y sus flujos, se utilizan peticiones GET a diferentes URL que proporciona la API de OpenDaylight. Las respuestas son proporcionadas en formato XML, aunque también existe la posibilidad de un formato JSON, y la aplicación es la encargada del tratamiento de dicho datos. En estos casos el envío de información es grande, ya que se está solicitando datos sobre toda la topología de la red y sobre los flujos de todos los nodos. En el escenario planteado, en el primer caso, donde se recibe información sobre la topología, se reciben 44.521 bytes, y en el segundo caso que se recopilan los flujos se recibe un total de 2.119.462 bytes. En las siguientes capturas se muestran más detalles:

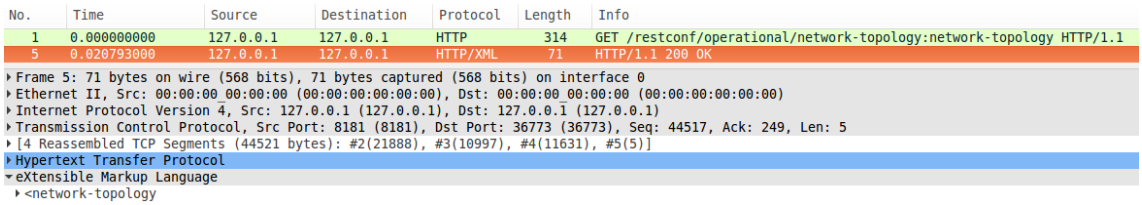

Figura 4.4: Captura de petición GET sobre topología de red

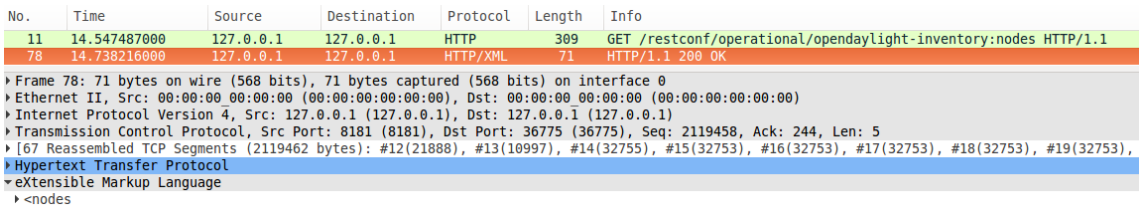

Figura 4.5. Captura de petición GET sobre flujos en la red

Para crear un flujo que bloquee el tráfico generado desde un cierto equipo final es necesario enviar una solicitud PUT a la interfaz northbound de OpenDaylight. En el cuerpo de la solicitud se incorpora un conjunto de datos en formato XML, también es posible en JSON, que indica información relativa al flujo. En la siguiente captura se muestra dicho tráfico:

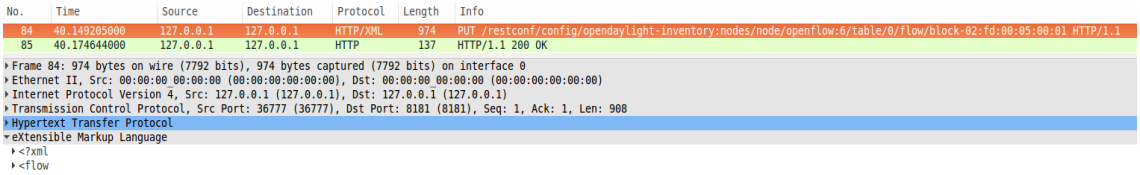

Figura 4.6. Captura de petición PUT para creación de flujo

Cuando se ha producido la creación de un flujo en un cierto nodo, es posible eliminarlo a través de una acción DELETE a la URL apropiada, que en este caso es /restconf/config/opendaylight-inventory:nodes/node/openflow:6/table/0/flow/block-02:fd:00:05:00:01, como se muestra en la siguiente captura recogida:

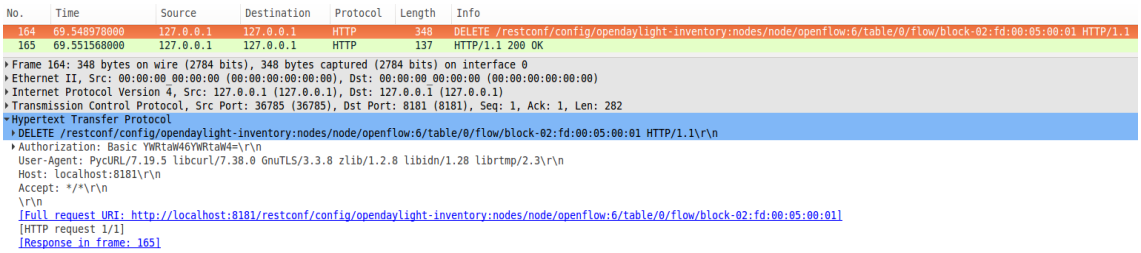

Figura 4.7. Captura de petición DELETE para eliminación de flujo

Para obtener la lista completa de interfaces soportadas por el controlador, junto con información sobre cada uno de ellas, se debe acceder a la siguiente dirección: http://localhost:8181/apidoc/explorer/index.html.

4.3.4 Pruebas de rendimiento

Durante el transcurso de las pruebas se han desarrollado varios escenarios, y a medida que estos aumentaban el número de dispositivos se ralentizaba el sistema por la alta utilización del procesador. Esto sucede porque se están ejecutando numerosas tareas en un solo equipo, como son la virtualización de equipos finales y switches de tipo Open vSwitch, además de la ejecución de la plataforma OpenDaylight.

De la misma manera, aumentar el número de switches en la red implica aumentar el tráfico que se transmite entre el controlador y los switches, y por ello este ha sido objeto de análisis en el presente apartado.

A través de la herramienta de análisis de tráfico Wireshark es posible obtener gráficas sobre el número de paquetes transmitido a lo largo de la captura. Se ha decidido utilizar en este análisis con diferentes topologías aumentando el número de dispositivos de red.

Se creará la topología a los 10 segundos de iniciar la captura, y 50 segundos más tarde se enviará tráfico entre los equipos finales.

Utilizándola en el escenario de demostración visto, que consistía en una topología de estrella extendida con 17 switches y 12 equipos finales, se obtiene el siguiente resultado:

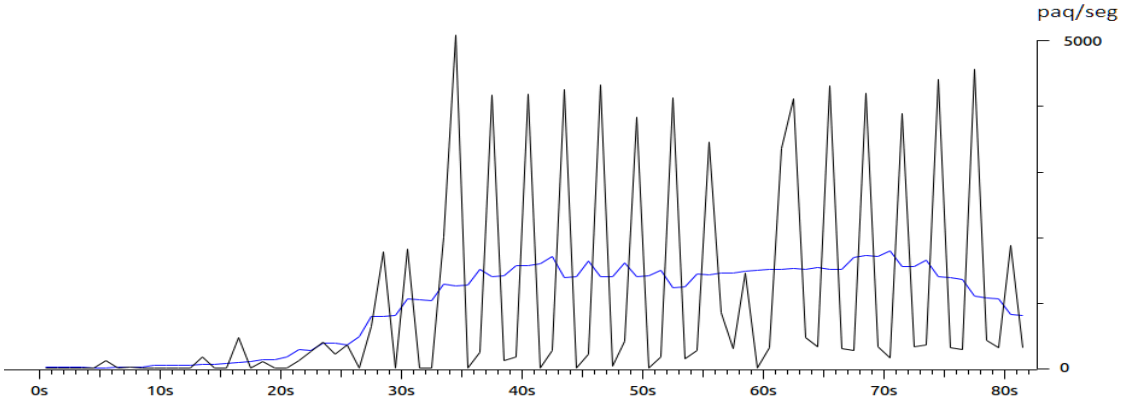

Figura 4.8. Gráfico de análisis de tráfico en topología de estrella extendida

A primera vista se aprecia el crecimiento de paquetes transferidos entre el controlador y los switches de la red a los pocos segundos de crear la topología. Una vez creada hay una media de unos 1.500 paquetes por segundo capturados.

Utilizando Mininet para crear otras topologías de prueba rápidamente se confirma la fuerte dependencia que implica tener más dispositivos de red con el tráfico transferido. Se ha utilizado la opción que proporciona Mininet de crear topologías de árbol especificando la profundidad: *mn --switch=ovsk,protocols=OpenFlow13 --controller =remote,ip=127.0.0.1,port=6633 --topo=tree,N*.

Con una profundidad de cinco niveles, se crean 32 switches y 31 equipos finales, y aunque no ralentiza el equipo en profundidad si se nota el aumento de paquetes recogido en la captura mediante Wireshark:

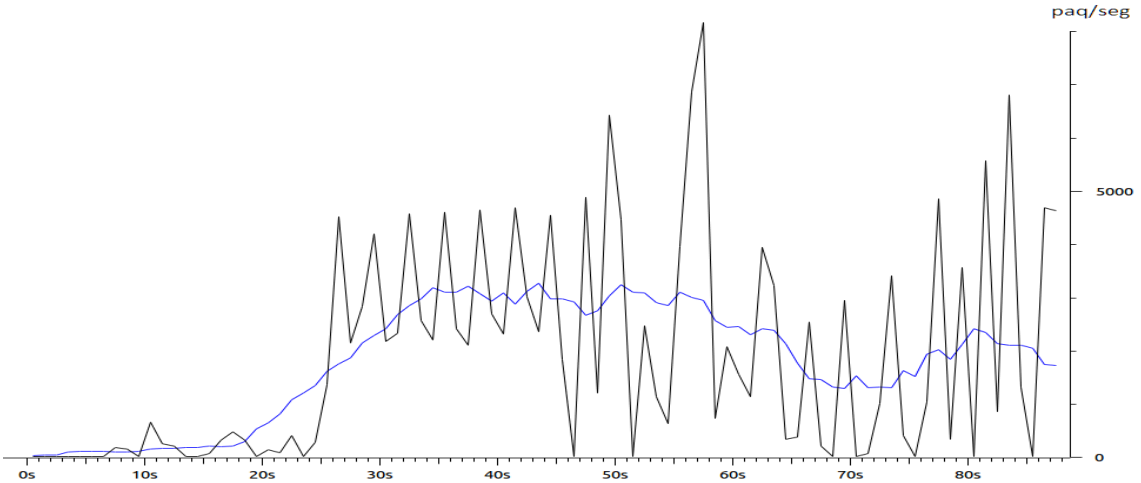

Figura 4.9. Gráfico de análisis de tráfico en topología de árbol cinco

El número de paquetes por segundo ha aumentado y hay una media aproximadamente de 2.500 paquetes por segundo, superior al caso anterior. Si se accede a DLUX es posible observar la topología al completo:

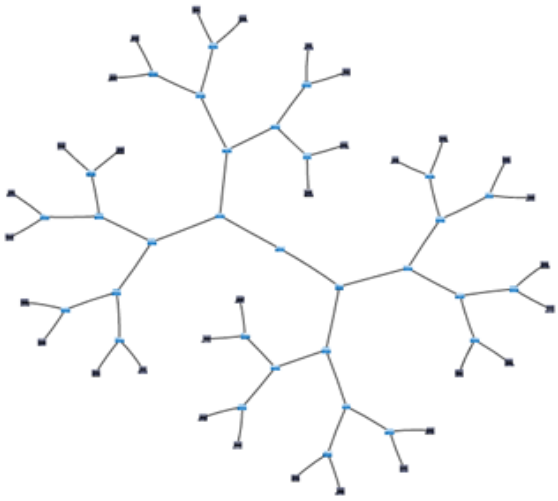

Figura 4.10. Topología de árbol de profundidad cinco

Si se aumenta la profundidad de la topología de árbol creada a seis, se crean 64 switches y 63 equipos finales. En esta situación el equipo se ralentiza e incluso llega a bloquearse en varias ocasiones. Además, el analizador de tráfico no es capaz de capturar todos los paquetes que se están transfiriendo por la red.

De esta forma, se puede extraer como conclusión que el enfoque centralizado que plantea las SDN tiene el inconveniente de la constante comunicación que hay que soportar entre el controlador y los dispositivos de la red, que deberá ser supervisada para omitir cierta frecuencia de mensajes innecesarios según las necesidades, además de que la virtualización del entorno trabajado en un solo equipo no es suficiente.

4.4 Escenario 2: Entorno OpenStack

En este escenario se pretende automatizar la integración de OpenDaylight en un entorno de computación en la nube basado en el proyecto OpenStack y analizar las ventajas de la aplicación de las SDN. Como parte del desarrollo de este trabajo se ha creado un repositorio público en GitHub que pretende compartir los escenarios de prueba utilizados con aquellos usuarios que quieren realizar una toma de contacto en un entorno OpenStack con OpenDaylight. Los escenarios se encuentran lo suficientemente automatizados para que en unos sencillos pasos se despliegue un escenario virtual operativo. El enlace del repositorio es el siguiente:

https://github.com/ralvarep/odl-openstack

4.4.1 Descripción del escenario

De los escenarios desarrollados y subidos en el repositorio se ha elegido mostrar en este trabajo el más complejo de ellos, que está compuesto por varios nodos de computación. De esta forma se pretende tener una mejor visión de un entorno OpenStack, con su último lanzamiento Kilo, en el que OpenDaylight es el gestor de red. La plantilla del escenario, que se encuentra adjunta en el *Apéndice B.1 Plantilla del escenario*, crea la siguiente topología:

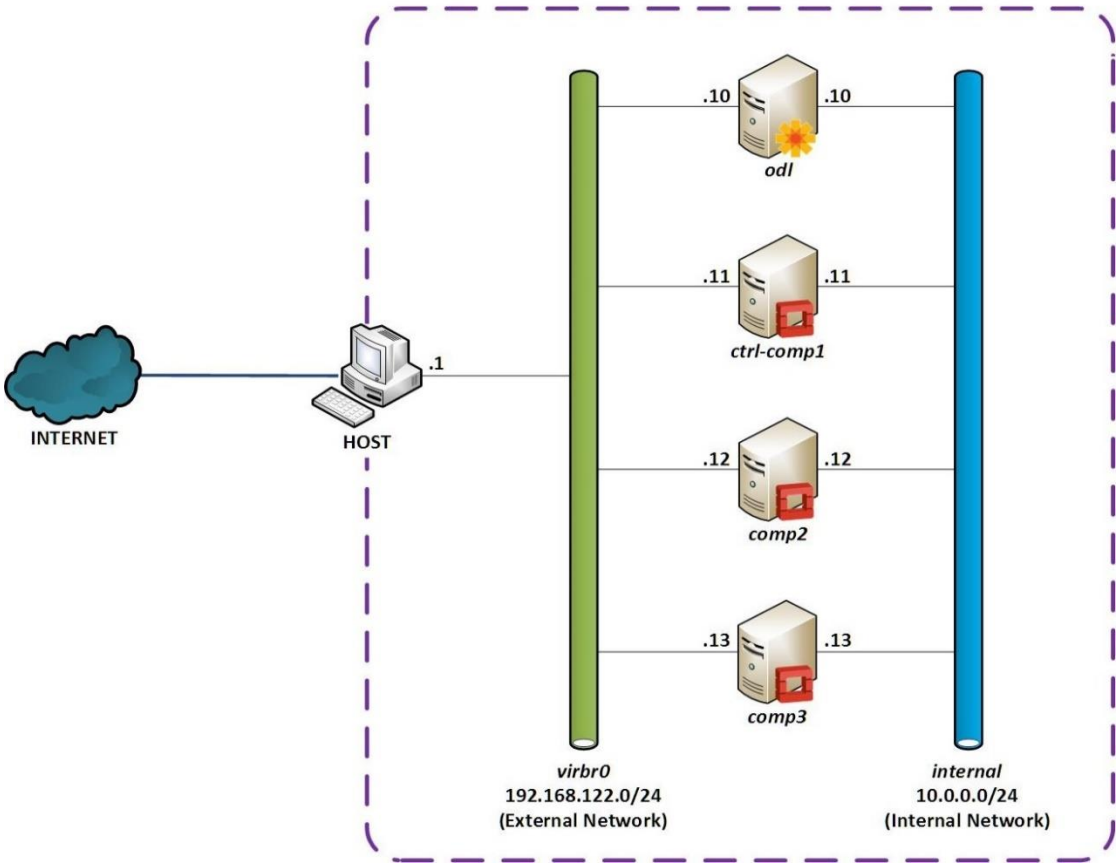

Figura 4.11. Escenario de demostración OpenStack - OpenDaylight

Para recrear la función de una red externa dentro del entorno virtualizado se ha decidido utilizar la red creada por defecto en libvirt [27], *virbr0*. Para la conexión a tal red es necesario solicitar desde la máquina virtual deseada una petición desde el cliente de DHCP. De esta forma se enviarán peticiones broadcast por la interfaz creada solicitando información de configuración, y así se conseguirá una dirección IP junto con otros parámetros relevantes para el correcto funcionamiento del sistema en la red, como la máscara de red, el router por defecto y los servidores de DNS. El servidor DHCP otorgará a la máquina virtual una dirección IP del rango 192.168.122.2 – 192.168.122.254 y tendrá acceso a Internet comportándose el equipo anfitrión como un router. En nuestro caso se solicitará al servidor DHCP una cierta dirección IP desde cada una de las máquinas virtuales para tener IP conocidas.

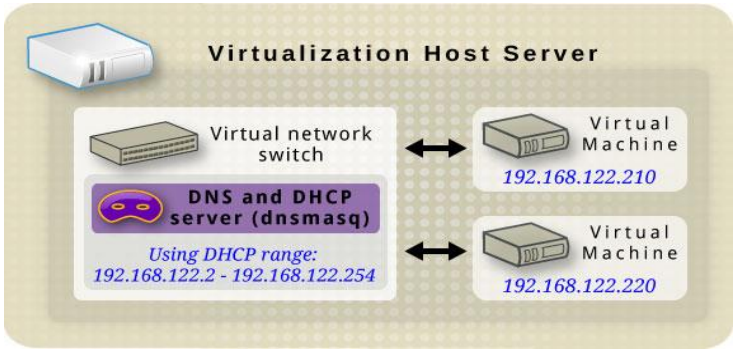

Figura 4.12. Redes virtuales con libvirt

Se puede comprobar que está activo el switch virtual *virbr0* mediante el comando *ifconfig* o *brctl show* en el que se obtendrán los siguientes resultados:

```
# ifconfig virbr0

Link encap:Ethernet  direcciónHW fe:35:18:d7:76:81
Direc.inet:192.168.122.1   Difus.:192.168.122.255 Másc:255.255.255.0
ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
Paquetes RX:27 errores:0 perdidos:0 overruns:0 frame:0
Paquetes TX:18 errores:0 perdidos:0 overruns:0 carrier:0
colisiones:0 long.colaTX:0
Bytes RX:2884 (2.8 KB)  TX bytes:1934 (1.9 KB)
```

```
# brctl show virbr0

bridge name       bridge id              STP enabled    interfaces
virbr0            8000.000000000000      yes
```

También existen unas funciones disponibles a través del comando virsh mediante el cual se obtiene más información, como por ejemplo:

- Mostrar el listado de redes (en nuestro caso la red que utilizada es *default*, que deberá venir preconfigurada):

```
# virsh net-list
Nombre               Estado     Inicio automático Persistente
----------------------------------------------------------
 default              activo          si                si
```

- Mostrar información general sobre la red solicitada:

```
# virsh net-info default

Nombre          default
UUID            70a01cdb-e771-462c-8ec1-8a7f64c57088
Activar:        si
Persistente:    si
Autoinicio:     si
Puente:         virbr0
```

- Mostrar el fichero de configuración en lenguaje XML de la red solicitada:

```
# virsh net-dumpxml default
<network>
  <name>default</name>
  <uuid>70a01cdb-e771-462c-8ec1-8a7f64c57088</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

Si en algunas de las comprobaciones no se obtiene un resultado verificando la existencia del switch virtual *virbr0* o se desea crear una nueva red, se puede definir según se indica en el *Apéndice B.2 Creación de redes con libvirt*.

En el escenario planteado se utilizan sistemas de ficheros KVM ya que se necesita una virtualización completa. En los nodos donde se están ejecutando servicios OpenStack se producirá una virtualización anidada de máquinas virtuales KVM, una primera virtualización para la creación del escenario y una segunda virtualización para alojar las máquinas virtuales creadas por OpenStack. En el *Apéndice B.3 Construcción del sistema de ficheros* del documento se proporciona más información sobre la creación o descarga del sistema de ficheros. Para arrancar el entorno es necesario seguir 4 pasos:

1) Clonar el repositorio

```
$ git clone https://github.com/ralvarep/odl-openstack.git
```

2) Construir el sistema de ficheros

```
$ filesystems/download-rootfs.sh
```

3) Crear el escenario virtual [6]

```
$ sudo vnx -f multi-node_4nodes.xml -t
```

4) Arrancar el entorno

```
$ sudo vnx -f multi-node_4nodes.xml -x start-odl-l3 --h2vm-timeout 0
```

[6] Para acceder a las máquinas virtuales, usuario: vnx contraseña: xxxx

Tras ejecutar el último comando, el despliegue de OpenStack con DevStack durará entre 5 y 10 minutos, dependiendo de la capacidad de cómputo de la máquina. Una vez finalizado se mostrarán los siguientes mensajes en cada una de las máquinas virtuales:

```
Starting OpenDaylight…
```

```
This is your host ip: 10.0.0.11
Horizon is now available at http://10.0.0.11/
Keystone is serving at http://10.0.0.11:5000/
The default users are: admin and demo
The password: password
```

```
This is your host ip: 10.0.0.12
```

```
This is your host ip: 10.0.0.13
```

En este momento ya es posible acceder a la interfaz gráfica de gestión de OpenStack, Horizon, a través de la dirección 192.168.122.11 desde el equipo anfitrión.

4.4.2 Servicios de L2

Cuando finaliza el despliegue del entorno OpenStack, se pueden comprobar los switches creados en los nodos de computación para la gestión de la red:

```
vnx@ctrl-comp1:~$ sudo ovs-vsctl show
    Manager "tcp:10.0.0.10:6640" is_connected: true
    Bridge br-int
        Controller "tcp:10.0.0.10:6633" is_connected: true
        fail_mode: secure
        Port br-int
            Interface br-int
    ovs_version: "2.3.1"
```

En la salida por pantalla se comprueba que existe un switch de integración, *br-int*, controlado por 10.0.0.10, donde se encuentra OpenDaylight, a través de los puertos 6640 (OVSDB) y 6633 (OpenFlow). En los otros nodos de computación se debe repetir la misma situación.

El despliegue de switches de integración es llevado a cabo tras el descubrimiento de los nodos por parte de OpenDaylight. En ese momento se establece una conexión OpenFlow-OVSDB con cada nodo y a partir de ahora, cualquier petición recibida por OpenDaylight a través de su API REST, será transformada en ciertas acciones en el switch de integración adecuado. El siguiente diagrama muestra la relación de conexiones:

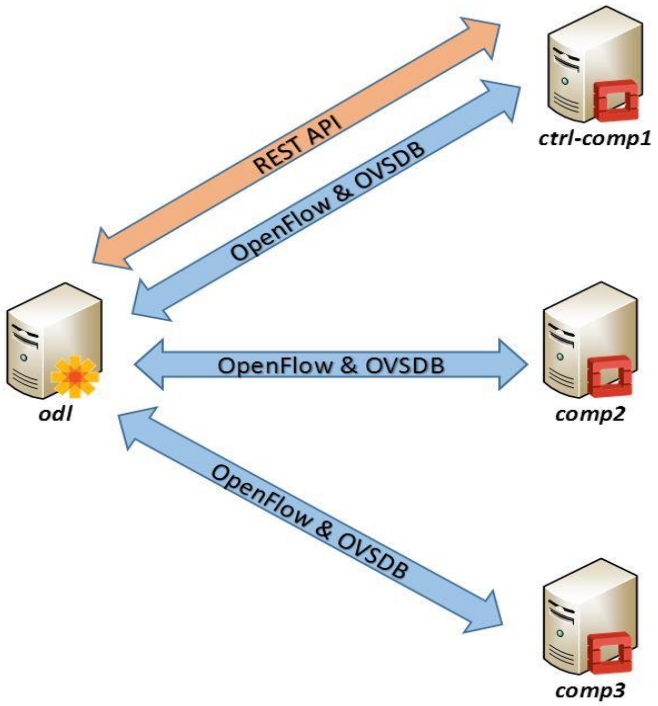

Figura 4.13. Comunicaciones entre los nodos del entorno OpenStack

Asimismo, es posible comprobar el estado inicial de las tablas de flujos de los switches de integración de los nodos, que son iguales de momento:

```
vnx@ctrl-comp1:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, table=0,   priority=0 actions=goto_table:20
cookie=0x0, table=0,   dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, table=20,  priority=0 actions=goto_table:30
cookie=0x0, table=30,  priority=0 actions=goto_table:40
cookie=0x0, table=40,  priority=0 actions=goto_table:50
cookie=0x0, table=50,  priority=0 actions=goto_table:60
cookie=0x0, table=60,  priority=0 actions=goto_table:70
cookie=0x0, table=70,  priority=0 actions=goto_table:80
cookie=0x0, table=80,  priority=0 actions=goto_table:90
cookie=0x0, table=90,  priority=0 actions=goto_table:100
cookie=0x0, table=100, priority=0 actions=goto_table:110
cookie=0x0, table=110, priority=0 actions=drop
```

Para la creación de una cierta topología en OpenStack se han creado un conjunto de scripts que se encuentran en la carpeta *scripts-devstack* del repositorio público [7]. A continuación se ejecutarán paso a paso las líneas del script *createL2Scenario_multi-node_4nodes.sh* para analizar las interacciones en el entorno.

[7] https://github.com/ralvarep/odl-openstack/tree/master/scripts-devstack

Se comenzará con la creación de una nueva red, de tipo GRE con identificador 101, a través de los siguientes comandos en el nodo de control de OpenStack, ctrl-comp1:

```
vnx@ctrl-comp1:~$ cd /home/vnx/devstack

vnx@ctrl-comp1:~/devstack$ source openrc admin admin

vnx@ctrl-comp1:~/devstack$ TENANT_ID=$(keystone tenant-list 2>
/dev/null | grep '\s'admin'' | awk '{print $2}')

vnx@ctrl-comp1:~/devstack$ IMAGE_ID=$(nova image-list | grep 'cirros-
0.3.4-x86_64-uec\s' | awk '{print $2}')

vnx@ctrl-comp1:~/devstack$ neutron net-create NET1 --tenant-id
$TENANTider:network_type gre --provider:segmentation_id 101

Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 82ea6f4d-4501-43f9-a89f-27554ed53d8c |
| mtu                       | 0                                    |
| name                      | NET1                                 |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 101                                  |
| router:external           | False                                |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | a592d509e04745d5bcda8401168ecaa8     |
+---------------------------+--------------------------------------+
```

Para la creación de la red se han establecido varias comunicaciones entre el controlador de OpenStack y la API de OpenDaylight. En dichas conexiones se comprueba que la red creada no exista, y en tal caso se procederá a la creación mediante llamadas POST a través de HTTP, como se muestra en la siguiente captura de tráfico:

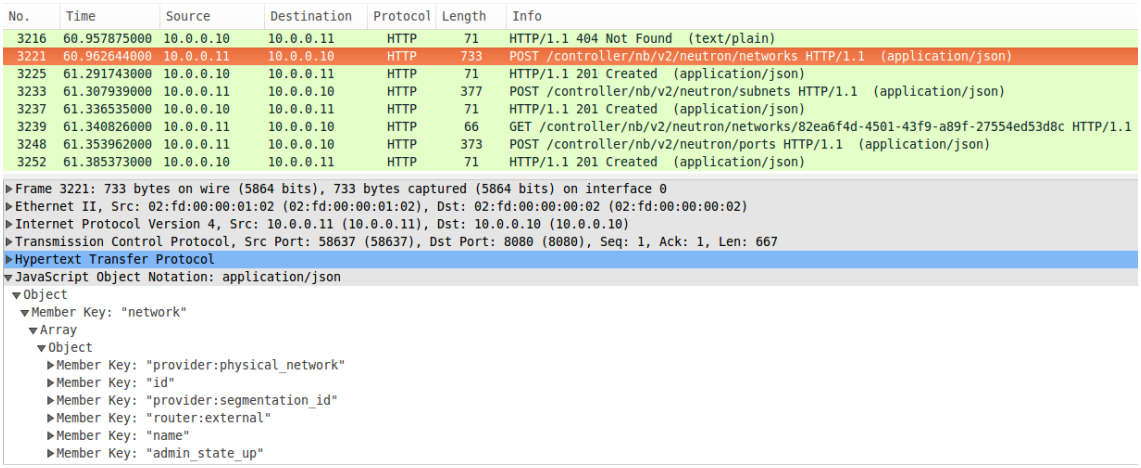

Figura 4.14. Captura de conexiones HTTP para la creación de una red

Si se comprueban las conexiones establecidas en el nodo odl, donde se encuentra OpenDaylight ejecutándose, se han establecido dos conexiones (OVSDB y OpenFlow) por nodo de computación y las llamadas a la API para la creación a la red:

Dirección A	Puerto A	Dirección B	Puerto B
10.0.0.11	58636	10.0.0.10	http-alt
10.0.0.11	58637	10.0.0.10	http-alt
10.0.0.11	58638	10.0.0.10	http-alt
10.0.0.11	58639	10.0.0.10	http-alt
10.0.0.13	60780	10.0.0.10	ovsdb
10.0.0.11	43538	10.0.0.10	ovsdb
10.0.0.12	37798	10.0.0.10	ovsdb
10.0.0.11	56356	10.0.0.10	6633
10.0.0.12	55490	10.0.0.10	6633
10.0.0.13	37031	10.0.0.10	6633

Figura 4.15. Conexiones establecidas desde el nodo OpenDaylight

Hasta ahora, la tabla de flujos de los switches no se ha visto modificada, ya que es necesario configurar alguna subred dentro de la red para la creación de instancias.

Para ello se creará una subred con la numeración 10.101.0.0/24, puerta de enlace 10.101.1 y servidor DNS para las máquinas que se alojen localizado en 8.8.8.8, que es un servidor público de Google. El comando a insertar es el siguiente:

```
vnx@ctrl-comp1:/home$ neutron subnet-create --name SUBNET1 --tenant-
id $TENANT_ID NET1 10.101.0.0/24 --gateway=10.101.0.1 --dns-
nameserver 8.8.8.8

Created a new subnet:
+-------------------+----------------------------------------------------+
| Field             | Value                                              |
+-------------------+----------------------------------------------------+
| allocation_pools  |{"start": "10.101.0.2", "end": "10.101.0.254"} |
| cidr              | 10.101.0.0/24                                      |
| dns_nameservers   | 8.8.8.8                                            |
| enable_dhcp       | True                                               |
| gateway_ip        | 10.101.0.1                                         |
| host_routes       |                                                    |
| id                | 7e85ae89-f1db-42ba-9d9c-d087b7456808               |
| ip_version        | 4                                                  |
| ipv6_address_mode |                                                    |
| ipv6_ra_mode      |                                                    |
| name              | SUBNET1                                            |
| network_id        | 82ea6f4d-4501-43f9-a89f-27554ed53d8c               |
| subnetpool_id     |                                                    |
| tenant_id         | a592d509e04745d5bcda8401168ecaa8                   |
+-------------------+----------------------------------------------------+
```

Capturando el tráfico intercambiado entre el nodo de control OpenStack y OpenDaylight se observa la utilización de la API REST para ordenar la creación de la subred:

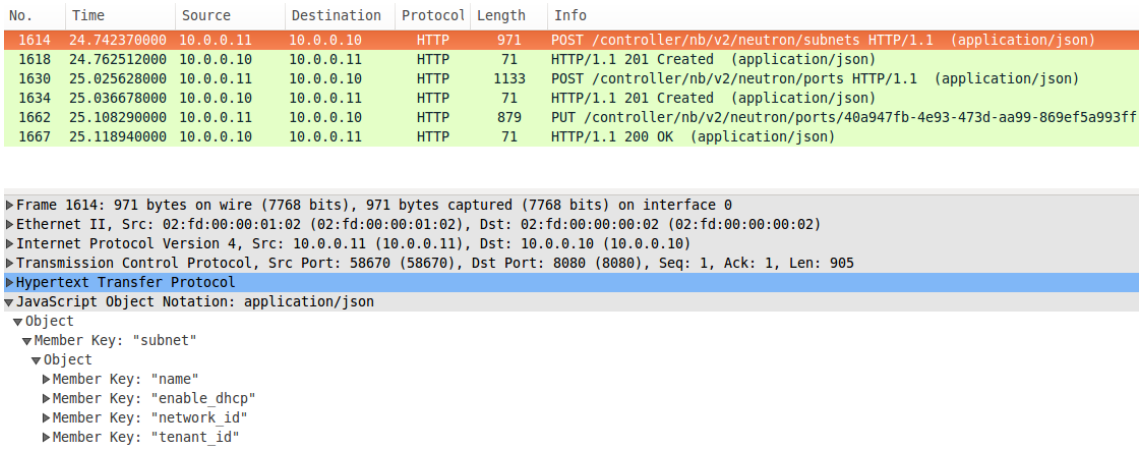

Figura 4.16. Captura de conexiones HTTP para la creación de una subred

Cuando se produce la creación de la subred se establecen un conjunto de túneles entre los nodos de computación con el objetivo de permitir la comunicación entre instancias alojadas en diferentes nodos. Si se analiza el tráfico producido en tal momento, es posible capturar los mensajes que envía OpenDaylight a los procesos Open vSwitch que se ejecutan en los nodos de computación para crear interfaces GRE:

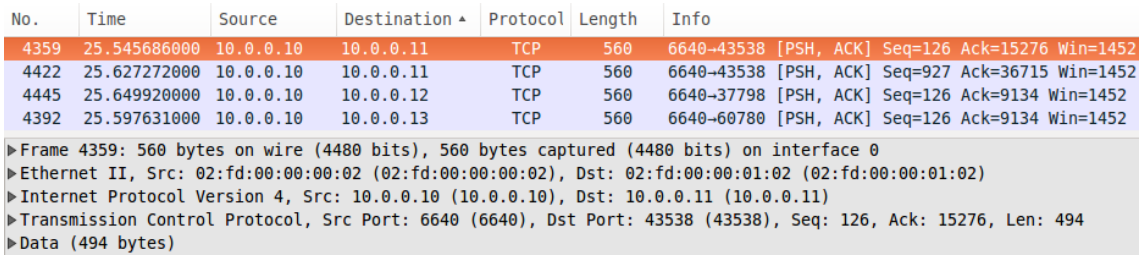

Figura 4.17. Creación de túneles GRE a través de OVSDB desde OpenDaylight

Como resultado, en los switches de integración aparecerán nuevas interfaces que conectan con otros switches. Si se verifica el estado de los switches en los nodos:

```
vnx@ctrl-comp1:/home$ sudo ovs-vsctl show
    Manager "tcp:10.0.0.10:6640" is_connected: true
    Bridge br-int
        Controller "tcp:10.0.0.10:6633" is_connected: true
        fail_mode: secure
        Port "tap40a947fb-4e"
            Interface "tap40a947fb-4e"
                type: internal
        Port br-int
            Interface br-int
        Port "gre-10.0.0.13"
            Interface "gre-10.0.0.13"
                type: gre
                options: {key=flow, local_ip="10.0.0.11",
                 remote_ip="10.0.0.13"}
        Port "gre-10.0.0.12"
            Interface "gre-10.0.0.12"
                type: gre
                options: {key=flow, local_ip="10.0.0.11",
                 remote_ip="10.0.0.12"}
```

De esta forma se comprueba cómo se han creado dos interfaces de tipo GRE para la comunicación con los nodos de computación externos, y una interfaz adicional (tap40a947fb-4e) que se corresponde con el servidor DHCP de la red. Para ello se ha creado un espacio de nombres con una interfaz que se corresponde al servidor donde está ejecutándose un servidor DHCP a través de *dnsmasq* [28]. Se puede comprobar de la siguiente forma:

```
vnx@ctrl-comp1:/home$ ip netns
qdhcp-82ea6f4d-4501-43f9-a89f-27554ed53d8c
```

```
vnx@ctrl-comp1:~$ sudo ip netns exec qdhcp-82ea6f4d-4501-43f9-a89f-
27554ed53d8c bash
root@ctrl-comp1:~# ifconfig
tap40a947fb-4e
  Link encap:Ethernet  HWaddr fa:16:3e:fa:23:23
  inet addr:10.101.0.2 Bcast:10.101.0.255 Mask:255.255.255.0
  inet6 addr: fe80::f816:3eff:fefa:2323/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

Esta última interfaz, correspondiente al servidor DHCP, solo se crea en el nodo de control, por lo que en los demás switches de integración del entorno no aparecerá:

```
vnx@comp2:/home$ sudo ovs-vsctl show

    Manager "tcp:10.0.0.10:6640" is_connected: true
    Bridge br-int
        Controller "tcp:10.0.0.10:6633" is_connected: true
        fail_mode: secure
        Port "gre-10.0.0.11"
            Interface "gre-10.0.0.11"
                type: gre
                options: {key=flow, local_ip="10.0.0.12",
                 remote_ip="10.0.0.11"}
        Port br-int
            Interface br-int
```

Revisando la tabla de flujos se comprueba que se han añadido nuevas entradas relacionadas con la subred creada y con el servidor DHCP:

```
vnx@ctrl-comp1:/home$ sudo ovs-ofctl dump-flows br-int -O Openflow13

OFPST_FLOW reply (OF1.3) (xid=0x2):

cookie=0x0, duration=4009.079s, table=0, n_packets=8, n_bytes=648,
 in_port=1,dl_src=fa:16:3e:fa:23:23 actions=set_field:0x65->tun_id,
 load:0x1->NXM_NX_REG0[],goto_table:20

[...]

cookie=0x0, duration=4005.477s, table=0, n_packets=0, n_bytes=0,
 tun_id=0x65,in_port=2 actions=load:0x2->NXM_NX_REG0[],goto_table:20

cookie=0x0, duration=4003.965s, table=0, n_packets=0, n_bytes=0,
 tun_id=0x65,in_port=3 actions=load:0x2->NXM_NX_REG0[],goto_table:20

[...]

cookie=0x0, duration=4005.989s, table=110, n_packets=0, n_bytes=0,
 priority=8192,tun_id=0x65 actions=drop

[...]

cookie=0x0, duration=4007.492s, table=110, n_packets=0, n_bytes=0,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:
 00:00:00:00 actions=output:1

cookie=0x0, duration=4006.998s, table=110, n_packets=3, n_bytes=230,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:
 00:00:00:00 actions=output:1,output:2,output:3

cookie=0x0, duration=4008.079s, table=110, n_packets=0, n_bytes=0,
 tun_id=0x65,dl_dst=fa:16:3e:fa:23:23 actions=output:1
```

En los nodos de computación comp2 y comp3 se ha añadido una nueva línea en la última tabla de flujos para reenviar el tráfico al servidor DHCP:

```
vnx@ctrl-comp2:/home$ sudo ovs-ofctl dump-flows br-int -O Openflow13

cookie=0x0, duration=4084.656s, table=110, n_packets=0, n_bytes=0,
 tun_id=0x65,dl_dst=fa:16:3e:fa:23:23 actions=output:1
```

Una vez creada la subred, se pueden iniciar máquinas virtuales en el entorno OpenStack a través del componente nova y conectarlas a una cierta red a ellas. Para ello hay que ejecutar los siguientes comandos:

```
vnx@ctrl-comp1:~/devstack$ NET1_ID=$(neutron net-list | grep -w NET1
| awk '{print $2}')

vnx@ctrl-comp1:~/devstack$ nova boot VM-11 --poll --flavor m1.tiny --
image $IMAGE_ID --nic net-id=${NET1_ID} --availability-zone
nova:ctrl-comp1

Server building... 100% complete
```

Una vez creada la instancia de máquina virtual, en este caso en el nodo ctrl-comp1, se habrá añadido una nueva interfaz al switch de integración (tap8756223d-6b):

```
vnx@ctrl-comp1:~$ sudo ovs-vsctl show

    Manager "tcp:10.0.0.10:6640" is_connected: true
    Bridge br-int
        [...]
        Port "tap8756223d-6b"
            Interface "tap8756223d-6b"
        [...]
```

Si se examinan las entradas de las tablas de flujos de los switches de integración, se encuentran nuevos flujos con información de nivel 2 de la máquina virtual creada:

```
vnx@ctrl-comp1:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13

[...]

cookie=0x0, duration=1255.734s, table=0, n_packets=29, n_bytes=2024,
 in_port=4,dl_src=fa:16:3e:b7:c6:35 actions=set_field:0x65->tun_id,
 load:0x1->NXM_NX_REG0[],goto_table:20

[...]

cookie=0x0, duration=1254.731s, table=110, n_packets=2, n_bytes=717,
 tun_id=0x65,dl_dst=fa:16:3e:b7:c6:35 actions=output:4
```

Asimismo, en las tablas de flujos de los nodos de computación comp2 y comp3, también aparece una nueva entrada para enviar tráfico a la instancia creada:

```
vnx@ctrl-comp1:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13

cookie=0x0, duration=1343.225s, table=110, n_packets=0, n_bytes=0,
 tun_id=0x65,dl_dst=fa:16:3e:b7:c6:35 actions=output:1
```

Una vez analizada la creación de redes, subredes e instancias en el entorno OpenStack, se crearán dos nuevas instancias, cada una en los nodos de computación restantes con la opción *--availability-zone*:

```
vnx@ctrl-comp1:~/devstack$ nova boot VM-12 --poll --flavor m1.tiny --
image $IMAGE_ID --nic net-id=${NET1_ID} --availability-zone nova:comp2

Server building... 100% complete

vnx@ctrl-comp1:~/devstack$ nova boot VM-13 --poll --flavor m1.tiny --
image $IMAGE_ID --nic net-id=${NET1_ID} --availability-zone nova:comp3

Server building... 100% complete
```

En este caso OpenDaylight se encarga de gestionar la creación de un túnel GRE entre ambos nodos de computación, comp2 y comp3, para comunicar ambas máquinas virtuales sin necesidad de enviar el tráfico por el nodo de control, evitando con ello sobrecargar los enlaces de la red física:

```
vnx@comp2:~$ sudo ovs-vsctl show

    Manager "tcp:10.0.0.10:6640" is_connected: true
    Bridge br-int
        [...]
        Port "gre-10.0.0.13"
            Interface "gre-10.0.0.13"
                type: gre
                options: {key=flow, local_ip="10.0.0.12",
                 remote_ip="10.0.0.13"}
```

La lista completa de entradas de flujo de todos los nodos del entorno OpenStack se adjunta en el *Apéndice B.6 Tablas de flujos de los nodos de computación (L2)*.

Como resultado de las acciones anteriores se ha creado una red local con tres instancias conectadas a ella, como se muestra en Horizon a través del apartado *Topología de red*:

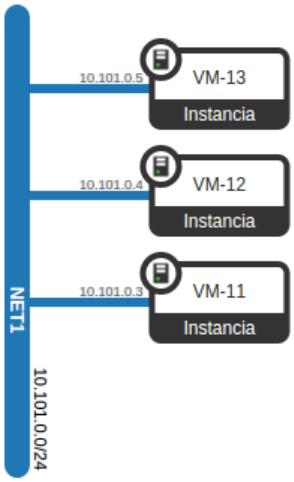

Figura 4.18. Topología de red en OpenStack desde Horizon (L2)

Como muestra en la anterior figura, las tres máquinas virtuales se encuentran en la misma red local y tienen conectividad entre ellas, pero físicamente se encuentran alojadas en los diferentes nodos de computación de la arquitectura:

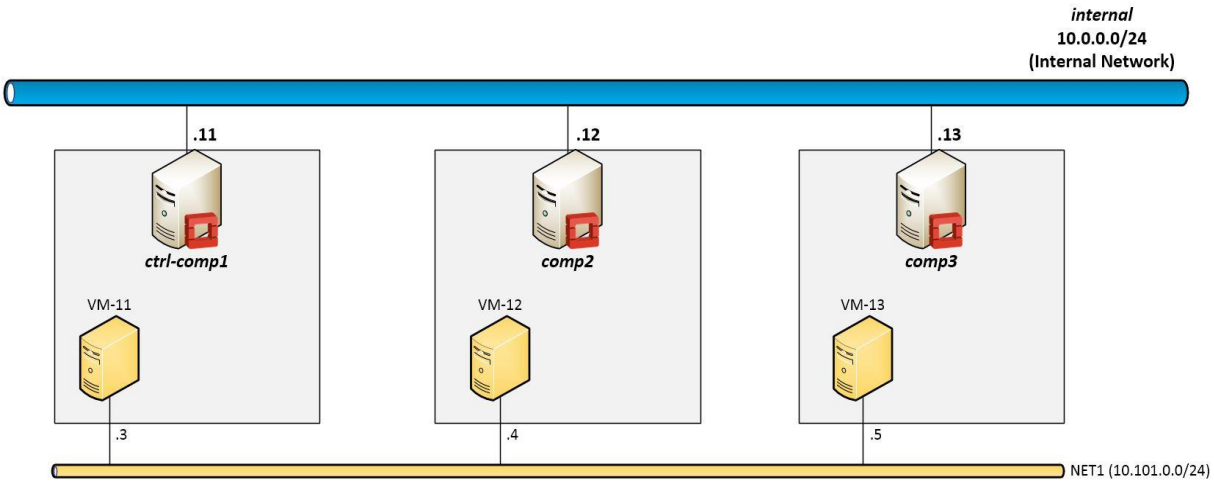

Figura 4.19. Arquitectura del escenario OpenStack-OpenDaylight (L2)

Si se prueba a realizar un ping entre máquinas virtuales para comprobar su conectividad se obtendrá un resultado satisfactorio. Por ejemplo, la prueba desde VM-11 hasta VM-13 se muestra a continuación:

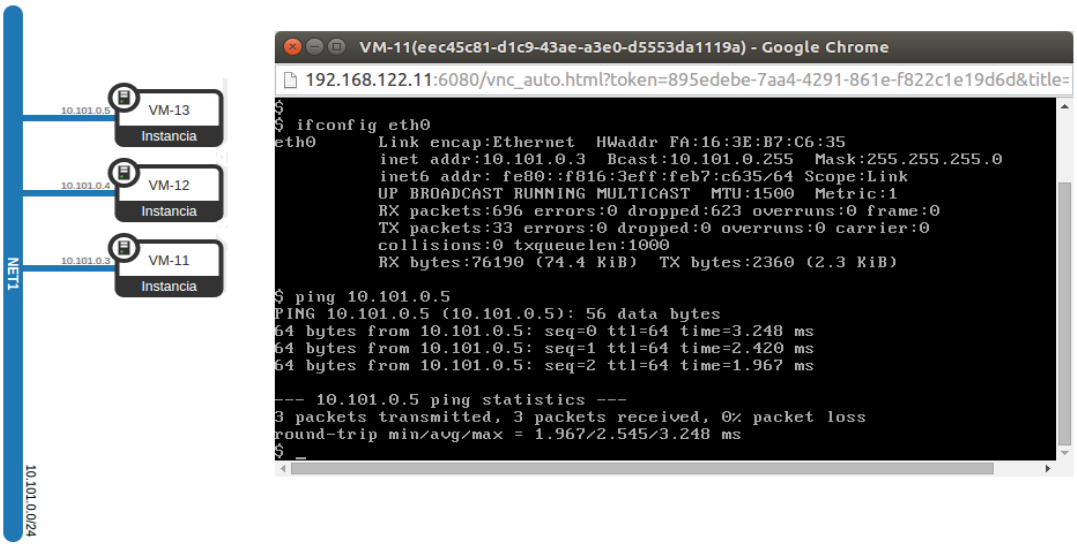

Figura 4.20. Prueba de conectividad entre máquinas virtuales (L2)

En el caso anterior, si se captura el tráfico intercambiado a través de la red se observa cómo se transfiere encapsulado a través de GRE:

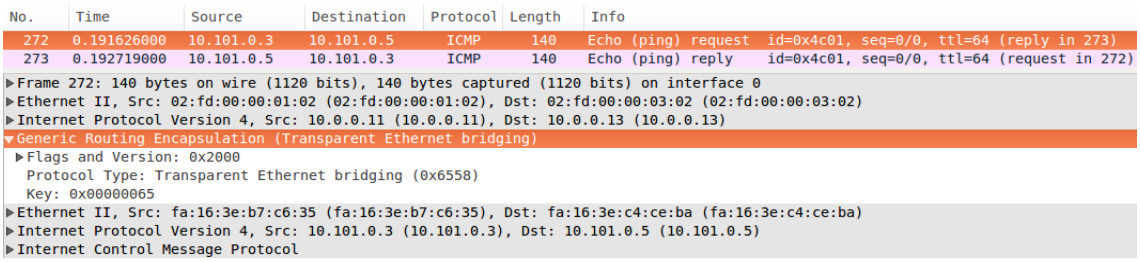

Figura 4.21. Captura de tráfico encapsulado con GRE

De la misma forma que se han creado redes GRE, es posible crear redes de tipo VXLAN, y el resultado sería el mismo. Internamente se cambiarían las interfaces de los switches de integración y el tráfico se encapsularía sobre UDP, como se muestra a continuación:

```
vnx@comp3:~$ sudo ovs-vsctl show

    Bridge br-int
        [...]
        Port "vxlan-10.0.0.12"
            Interface "vxlan-10.0.0.12"
                type: vxlan
                options: {key=flow, local_ip="10.0.0.11",
                 remote_ip="10.0.0.12"}
        Port "vxlan-10.0.0.13"
            Interface "vxlan-10.0.0.13"
                type: vxlan
                options: {key=flow, local_ip="10.0.0.11",
                 remote_ip="10.0.0.13"}
```

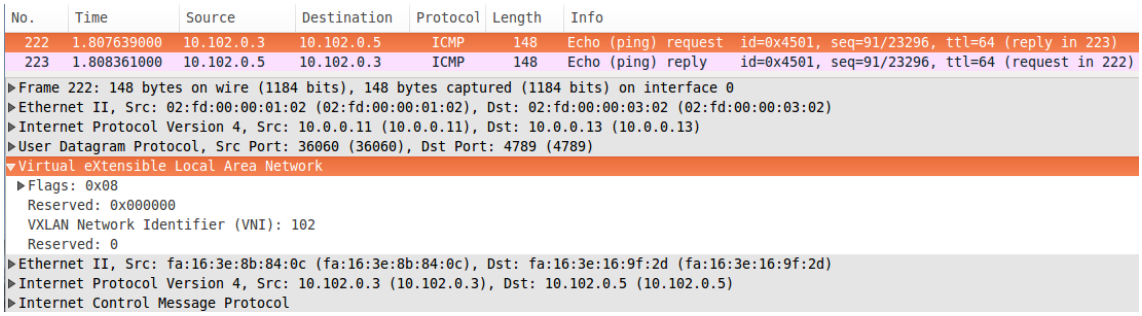

Figura 4.22. Captura de tráfico encapsulado con VLXAN

4.4.3 Servicios de L3

Para comprobar la función de enrutamiento que es posible gestionar a través de OpenDaylight, se añadirá una nueva red al entorno OpenStack y se interconectará con la red creada en el apartado anterior. En este caso se analizará el script *createL3Scenario_multi-node_4nodes.sh*[8] del repositorio público.

Dadas varias redes en la topología de OpenStack, es posible conectarlas entre sí a través de un router. Esta acción es llevada a cabo mediante la creación de un router, que gestiona OpenDaylight a través de las tablas de flujos, y la conexión de interfaces virtuales del router a las subredes creadas previamente. Los comandos a ejecutar para llevar a cabo estas tareas son los siguientes:

```
vnx@ctrl-comp1:~/devstack$ neutron router-create R1 --tenant-id
$TENANT_ID

Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| distributed           | False                                |
| external_gateway_info |                                      |
| id                    | 4c049c6e-58a4-48f4-acdc-f603bb140585 |
| name                  | R1                                   |
| routes                |                                      |
| status                | ACTIVE                               |
| tenant_id             | a592d509e04745d5bcda8401168ecaa8     |
+-----------------------+--------------------------------------+

vnx@ctrl-comp1:~/devstack$ neutron router-interface-add R1 SUBNET1
Added interface ae2e640c-b622-4791-afd6-9c783e04bb4d to router R1.

vnx@ctrl-comp1:~/devstack$ neutron router-interface-add R1 SUBNET2
Added interface c99c72e2-99da-4e24-b845-6aa58487a10e to router R1.
```

[8] https://github.com/ralvarep/odl-openstack/blob/master/scripts-devstack

Como resultado, en Horizon se mostrarán las dos redes conectadas a través de una nueva entidad:

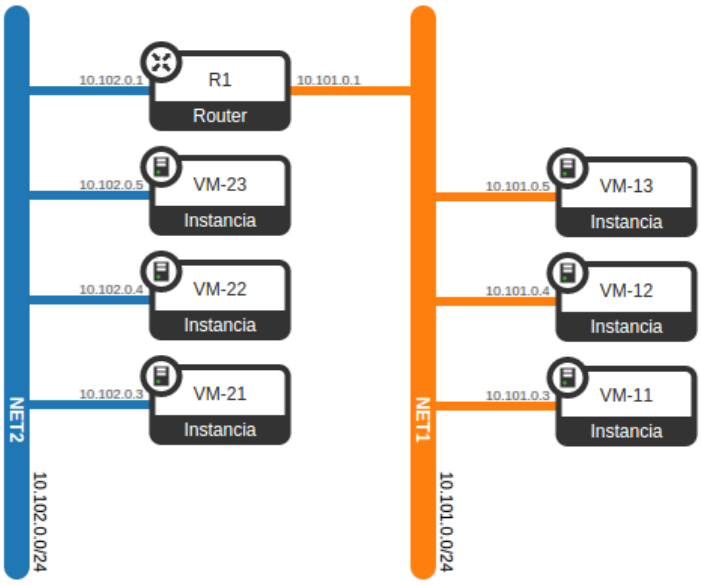

Figura 4.23. Topología de red en OpenStack desde Horizon (L3)

Las seis máquinas virtuales se encuentran repartidas en dos redes locales diferentes, pero entre sí tienen conectividad a través del router. Asimismo, físicamente se encuentran alojadas en los distintos nodos de computación de la arquitectura:

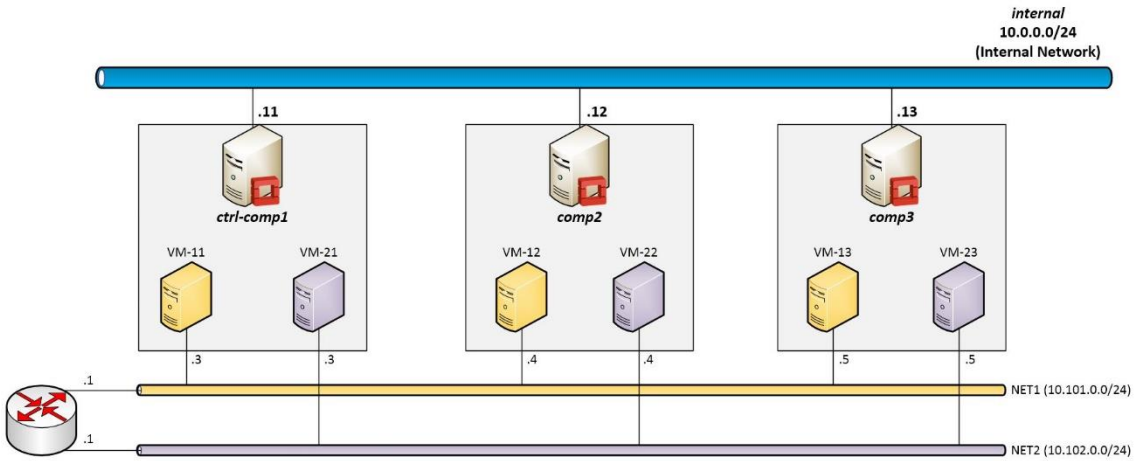

Figura 4.24. Arquitectura del escenario OpenStack-OpenDaylight (L3)

Con el plugin L3 activado a través de OpenDaylight se hace uso de las siguiente tablas que componen el *pipeline*: tabla 20 (Contestador ARP), tabla 60 (Enrutamiento Virtual Distribuido) y tabla 70 (Reenvío L3).

La tabla 20 está formada por tantas entradas como direcciones IP en la topología se estén utilizando, y permite obtener la dirección MAC de un equipo sin necesidad de inundar la red con un mensaje broadcast. El formato de la entrada es:

```
cookie=0x0, duration=3654.137s, table=20, n_packets=3, n_bytes=126,
priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.1
actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:19
:89:21->eth_src,load:0x2->NXM_OF_ARP_OP[],move:NXM_NX_ARP_SHA[]->NXM_
NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e198
921->NXM_NX_ARP_SHA[],load:0xa650001->NXM_OF_ARP_SPA[],IN_PORT
```

Con esta entrada configurada con OpenFlow es posible generar un mensaje de respuesta ARP al nodo solicitando de la dirección MAC. En el caso de enrutamiento entre redes es de especial relevancia, ya que permite proporcionar una MAC virtual del router para que los equipos finales envíen tráfico con destino otras redes.

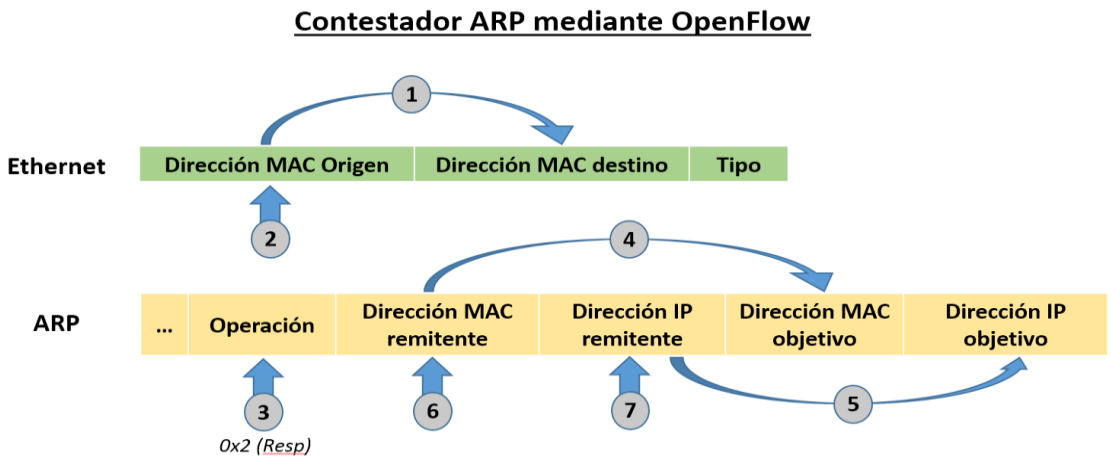

Figura 4.25. Contestador ARP mediante OpenFlow

Los pasos que se siguen para generar la respuesta, dado un mensaje ARP solicitando la dirección MAC de una determinada dirección IP, son los siguientes:

1. Se modifica la dirección MAC destino con la dirección MAC origen del paquete.
2. Se inserta la dirección MAC origen en la trama Ethernet. Puede tratarse de la dirección MAC de una máquina virtual o de una dirección MAC virtual de un router que gestiona OpenDaylight.
3. Se modifica la operación del mensaje ARP a respuesta (0x2).
4. La dirección MAC remitente del mensaje ARP se utiliza como dirección MAC objetivo.
5. La dirección IP remitente del mensaje ARP se utiliza como dirección IP objetivo.
6. La dirección MAC remitente es modificada con la dirección MAC que ha sido insertada en el paso 2.
7. Se modifica la dirección IP remitente con la dirección lógica de la interfaz de una máquina virtual o la dirección IP virtual de un router.

La última acción que se añade en las entradas de flujo relativas a la tabla 20 es *IN_PORT*, y permite enviar directamente la respuesta ARP por el puerto de entrada de la solicitud, sin necesidad de continuar procesando el *pipeline* del switch.

La tabla 60 es la que se encarga de enrutar los paquetes de una red a otra. A través de la tabla de flujos se realiza cambiando el campo de identificador de túnel al identificador de la red destino y colocando la dirección MAC origen que se corresponde a la dirección MAC virtual del router. Un ejemplo de entrada es el siguiente:

```
cookie=0x0, duration=3793.076s, table=60, n_packets=3, n_bytes=294,
priority=2048,ip,tun_id=0x65,nw_dst=10.102.0.0/24
actions=set_field:fa:16:3e:04:35:cb->eth_src,dec_ttl,set_field:0x66->
tun_id,goto_table:70
```

En la tabla 70 se finaliza el proceso de enrutamiento y se modifica la dirección MAC destino en el paquete según la dirección IP destino, como se muestra a continuación:

```
cookie=0x0, duration=3785.452s, table=70, n_packets=7, n_bytes=686,
priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.3
actions=set_field:fa:16:3e:87:c1:43->eth_dst,goto_table:80
```

La lista completa de entradas de flujo de todos los nodos del entorno OpenStack se adjunta en el *Apéndice B.7 Tablas de flujos de los nodos de computación (L3)*.

Dada la estructura de las tablas de flujos, es posible enrutar paquetes entre instancias de diferentes redes que se encuentran en el mismo nodo de computación sin necesidad de utilizar los enlaces de la red. Esto es lo conocido como *Routing Virtual Distribuido*, o por sus siglas DVR.

Para comprobar la conectividad entre redes se puede ejecutar un ping desde una máquina virtual hasta otra situada en la otra red, y se completará correctamente:

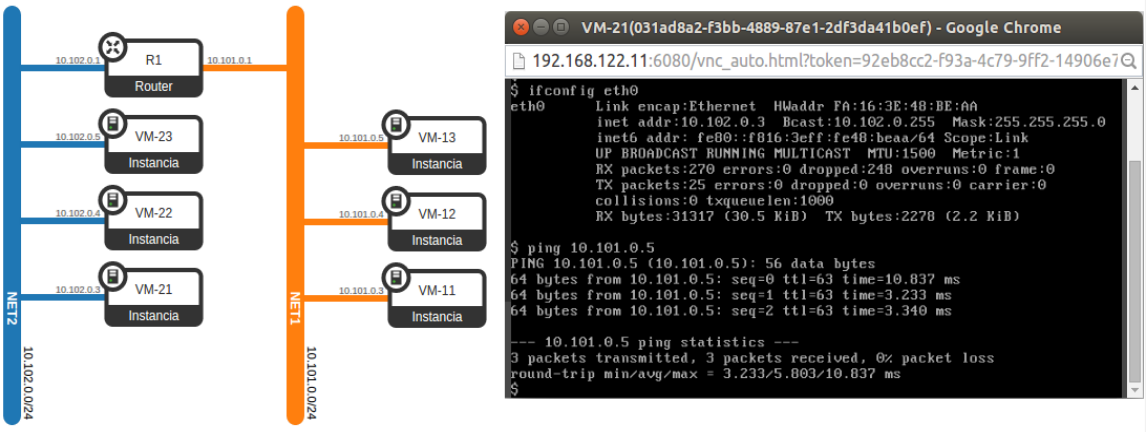

Figura 4.26. Prueba de conectividad entre máquinas virtuales (L3)

5 Resultados y conclusiones

Una vez realizada la descripción del trabajo desarrollado, en este capítulo se revisan los resultados obtenidos, extrayendo las conclusiones oportunas, y se plantean posibles líneas de trabajo futuras.

5.1 Resultados

Durante el trascurso del presente trabajo se ha alcanzado un amplio nivel de conocimiento sobre las Redes Definidas por Software, tanto en conceptos relacionados con la arquitectura que se propone como en soluciones específicas que se presentan en diferentes proyectos.

OpenDaylight acerca el entendimiento de un controlador SDN, y me ha permitido descubrir aspectos técnicos como su arquitectura y soporte software, y otras cuestiones que no hay que descuidar en un proyecto de esta envergadura, como son las herramientas de gestión, que me han permitido comunicarme con desarrolladores.

Como se explicó al inicio de la memoria, los objetivos técnicos del proyecto, que consistían en la creación de escenarios virtuales que muestren diferentes casos de uso, se han cumplido satisfactoriamente. En el primer escenario de demostración se ha desarrollado una aplicación que interactúa con el controlador a través de sus APIs REST, y muestra la ventaja del software para el control de acceso a la red, y en el segundo escenario se ha integrado OpenDaylight con la plataforma OpenStack.

El segundo escenario de demostración me ha permitido adentrarme en un entorno de computación en la nube y experimentar a través de OpenStack. En este caso ha sido necesario profundizar en la gestión de la red para entender la integración con OpenDaylight. Asimismo, se ha automatizado la creación de un entorno OpenStack con OpenDaylight mediante un conjunto de escenarios VNX, que se encuentran disponibles en el repositorio publicado. Esto permite tener escenarios didácticos en el que cualquier persona interesada puede tener una toma de contacto sin preocuparse por el despliegue del entorno y por cuestiones de la integración OpenStack-OpenDaylight.

En definitiva, el trabajo ha aumentado mi experiencia en el campo de las Redes Definidas por Software, y me ha permitido conocer un proyecto de referencia como es OpenDaylight.

5.2 Conclusiones

Las Redes Definidas por Software proporcionan una arquitectura de red flexible y escalable que permite adaptarse a las necesidades actuales y futuras. Las aplicaciones podrán demandar dinámicamente recursos de red en tiempo real, como ancho de banda o calidad de servicio, y a través del software se podrá gestionar. Debido a su capacidad se optimizará el uso de recursos de red infrautilizados de una forma inteligente, simplificando la complejidad que introducen las nuevas tendencias en las redes de comunicación. Dadas las atractivas características y ventajas de las Redes Definidas por Software, la comunidad investigadora y la industria han impulsado numerosas iniciativas que pretenden proponer soluciones basadas en SDN.

El estándar OpenFlow se cataloga como una de la interfaces de las Redes Definidas por Software del futuro, y mediante su evolución se soportan cada vez más funcionalidades para que las SDN se apliquen en ámbitos más amplios. Su desarrollo y gran usabilidad ha despertado, tanto en fabricantes como en generadores de contenidos, un gran interés.

El proyecto OpenDaylight, que tiene la misión de incrementar la adopción de las SDN, se está convirtiendo en un referente junto a otras organizaciones como la Open Networking Foundation. Proporciona una completa plataforma que se encuentra en constante evolución con diferentes módulos para abastecer las necesidades de los usuarios. Su última versión publicada, Lithium, compuesta por más de 40 proyectos ofrece una gran diversidad de características a la plataforma mediante un conjunto extensible de servicios y protocolos. Asimismo, proveedores comerciales de soluciones SDN ofrecen versiones personalizadas basadas en OpenDaylight enfocadas para usos específicos.

El caso de uso consistente en proporcionar servicios de red en plataformas de computación en la nube como OpenStack mediante la integración de controladores SDN, como OpenDaylight, está tomando gran relevancia en el sector. Es un tema que se encuentra en pleno desarrollo por OpenDaylight y a medida que avanza se incluyen nuevos servicios de red.

En resumen, el futuro de las redes va a depender cada vez más del software, lo que acelerará el ritmo de la innovación para las redes. Con sus numerosas ventajas, las Redes Definidas por Software prometen transformar las redes estáticas de hoy en día, y de esta forma convertirse en una nueva realidad para las redes de comunicación.

5.3 Líneas de trabajo futuras

Las Redes Definidas por Software han impulsado el desarrollo de numerosos proyectos que tratan de mejorar las redes de comunicaciones actuales. Este trabajo se ha centrado en OpenDaylight, pero extender la investigación en otros controladores y herramientas ayudaría a entender en mayor medida el enfoque que se está adoptando por otras organizaciones.

OpenDaylight se compone por un gran conjunto de proyectos que pretenden acelerar la adopción de las SDN. En este caso se ha analizado una pequeña porción de todos ellos, por lo que seguir con dicho análisis y abordar otros proyectos es otra posible línea de investigación.

En el primer escenario de demostración se ha trabajado una topología simple sin redundancia. Con la última versión publicada de OpenDaylight, es posible crear topologías más complejas con enlaces redundantes, ya que el controlador se encarga de reenviar el tráfico y bloquear los bucles que se generen. Esto permitiría hacer un análisis más realista de las infraestructuras de red que existen hoy en día.

Asimismo, el desarrollo de nuevas aplicaciones SDN que interactúen con OpenDaylight es una interesante línea de desarrollo, ya que cualquier idea puede ser desarrollada en software para resolver un cierto problema que se plantee en las redes de comunicación. Para ello se puede realizar una implementación en la capa del controlador y tratarse como un módulo que puede ser instalado en Karaf, igual que sucede con otros proyectos, o separar la aplicación del controlador manteniendo el contacto a través de las APIs que proporciona OpenDaylight en la interfaz northbound.

En el segundo escenario de demostración se ha analizado la integración de OpenDaylight en un entorno de computación en la nube como OpenStack y se han mostrado algunas de las funciones de red capaces a realizar con SDN, como son el reenvío de paquetes en L2 y el enrutamiento en L3. Esto es solo el principio de lo que se está desarrollando, ya que se pretende migrar las funcionalidades de red de OpenStack a OpenDaylight, y poder gestionar otros servicios como la asignación de IP flotantes, Balanceo de Carga, Firewall, etc. La creación de un entorno OpenStack más realista, sin la utilización de DevStack, y la continuación con el desarrollo de esta línea de trabajo proporcionaría conocimientos sobre la aplicación de las SDN en los centros de datos actuales, ya que es un importante caso de uso.

Apéndice A: Escenario 1

En este apéndice se adjunta toda la información relativa al primer escenario de demostración explicado en el trabajo. Se incluye la plantilla VNX para la creación del escenario con topología de estrella extendida, así como la aplicación desarrollada que permite el control de acceso a la red.

A.1 Plantilla del escenario

El escenario virtual utilizado ha sido creado por la herramienta VNX, y la plantilla desarrollada en lenguaje XML que lo define es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      STUDY OF SOTWARE-DEFINED NETWORKING
THROUGH THE DEVELOPMENT OF VIRTUAL SCENARIOS
      BASED ON THE OPENDAYLIGHT CONTROLLER
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Author: Raúl Álvarez Pinilla
Tutor: David Fernández Cambronero
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Telematics Engineering Department (DIT)
Technical University of Madrid (UPM)
SPAIN

Name:        demo_scenario
Description: Extended Start Topology
-->

<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
  <global>
    <version>2.0</version>
    <scenario_name>demo_scenario</scenario_name>
    <automac offset="5"/>
    <vm_mgmt type="none" />
    <!--vm_mgmt type="private" network="10.250.0.0" mask="24"
offset="16">
       <host_mapping />
    </vm_mgmt-->
    <vm_defaults>
        <console id="0" display="no"/>
        <console id="1" display="yes"/>
    </vm_defaults>
  </global>

```

```
36      <net name="Net1" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:01">
37            <connection name='link1' net='Net2'/>
38          <connection name='link2' net='Net3'/>
39          <connection name='link3' net='Net4'/>
40          <connection name='link4' net='Net5'/>
41      </net>
42      <net name="Net2" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:02">
43            <connection name='link5' net='Net6'/>
44            <connection name='link6' net='Net7'/>
45            <connection name='link7' net='Net8'/>
46      </net>
47      <net name="Net3" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:03">
48            <connection name='link8'  net='Net9'/>
49            <connection name='link9'  net='Net10'/>
50            <connection name='link10' net='Net11'/>
51      </net>
52      <net name="Net4" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:04">
53            <connection name='link11' net='Net12'/>
54            <connection name='link12' net='Net13'/>
55            <connection name='link13' net='Net14'/>
56      </net>
57      <net name="Net5" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:05">
58            <connection name='link14' net='Net15'/>
59            <connection name='link15' net='Net16'/>
60            <connection name='link16' net='Net17'/>
61      </net>
62      <net name="Net6" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:06"/>
63      <net name="Net7" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:07"/>
64      <net name="Net8" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:08"/>
65      <net name="Net9" mode="openvswitch" controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:09"/>
66      <net name="Net10" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:0a"/>
67      <net name="Net11" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:0b"/>
68      <net name="Net12" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:0c"/>
69      <net name="Net13" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:0d"/>
70      <net name="Net14" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:0e"/>
71      <net name="Net15" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:0f"/>
```

```
72      <net name="Net16" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:10"/>
73      <net name="Net17" mode="openvswitch"
      controller="tcp:127.0.0.1:6653"
      of_version="OpenFlow13"hwaddr="00:00:00:00:00:11"/>
74
75      <vm name="h1" type="lxc">
76        <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
77        <if id="1" net="Net6">
78          <ipv4>10.0.0.1/24</ipv4>
79        </if>
80        <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.4 -
      c 1</exec>
81      </vm>
82
83      <vm name="h2" type="lxc">
84        <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
85        <if id="1" net="Net7">
86          <ipv4>10.0.0.2/24</ipv4>
87        </if>
88        <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.5 -
      c 1</exec>
89      </vm>
90
91      <vm name="h3" type="lxc">
92        <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
93        <if id="1" net="Net8">
94          <ipv4>10.0.0.3/24</ipv4>
95        </if>
96        <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.6 -
      c 1</exec>
97      </vm>
98
99      <vm name="h4" type="lxc">
100       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
101       <if id="1" net="Net9">
102         <ipv4>10.0.0.4/24</ipv4>
103       </if>
104       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.7 -
      c 1</exec>
105     </vm>
106
107     <vm name="h5" type="lxc">
108       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
109       <if id="1" net="Net10">
110         <ipv4>10.0.0.5/24</ipv4>
111       </if>
112       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.8 -
      c 1</exec>
113     </vm>
114
115     <vm name="h6" type="lxc">
```

```
116       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
117       <if id="1" net="Net11">
118         <ipv4>10.0.0.6/24</ipv4>
119       </if>
120       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.9 -
      c 1</exec>
121     </vm>
122
123     <vm name="h7" type="lxc">
124       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
125       <if id="1" net="Net12">
126         <ipv4>10.0.0.7/24</ipv4>
127       </if>
128       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.10
      -c 1</exec>
129     </vm>
130
131     <vm name="h8" type="lxc">
132       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
133       <if id="1" net="Net13">
134         <ipv4>10.0.0.8/24</ipv4>
135       </if>
136       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.11
      -c 1</exec>
137     </vm>
138
139     <vm name="h9" type="lxc">
140       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
141       <if id="1" net="Net14">
142         <ipv4>10.0.0.9/24</ipv4>
143       </if>
144       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.12
      -c 1</exec>
145     </vm>
146
147     <vm name="h10" type="lxc">
148       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
149       <if id="1" net="Net15">
150         <ipv4>10.0.0.10/24</ipv4>
151       </if>
152       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.1 -
      c 1</exec>
153     </vm>
154
155     <vm name="h11" type="lxc">
156       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
157       <if id="1" net="Net16">
158         <ipv4>10.0.0.11/24</ipv4>
159       </if>
160       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.2 -
      c 1</exec>
161     </vm>
```

```
162
163     <vm name="h12" type="lxc">
164       <filesystem
      type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu</filesystem>
165       <if id="1" net="Net17">
166         <ipv4>10.0.0.12/24</ipv4>
167       </if>
168       <exec seq="pingall" type="verbatim" ostype="exec">ping 10.0.0.3 -
      c 1</exec>
169     </vm>
170
171   </vnx>
```

A.2 Aplicación de control de acceso a la red

La aplicación desarrollada para el escenario que permite controlar el acceso de los equipos a la red ha sido implementada como un script en Python, que se adjunta a continuación:

```
1   #!/usr/bin python
2   # -*- encoding: utf-8 -*-
3
4   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
5   #       STUDY OF SOTWARE-DEFINED NETWORKING
6   #  THROUGH THE DEVELOPMENT OF VIRTUAL SCENARIOS
7   #       BASED ON THE OPENDAYLIGHT CONTROLLER
8   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
9   # Author: Raúl Álvarez Pinilla
10  # Tutor: David Fernández Cambronero
11  # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
12  # Telematics Engineering Department (DIT)
13  # Technical University of Madrid (UPM)
14  # SPAIN
15
16  import os, sys, libxml2, urllib, re, pycurl
17  from StringIO import StringIO
18  from subprocess import check_output, CalledProcessError
19
20  # Show menu options
21  def menu():
22      while True:
23          print "\n[1] Show topology information"
24          print "[2] Show blocked hosts"
25          print "[3] Block a specific host"
26          print "[4] Unblock a specific host"
27          print "[q] Quit\n"
28          option = raw_input("Select an option: ")
29
30          if option == '1':
31              getTopology()
32              showTopology()
33          elif option == '2':
34              getTopology()
35              getBlocks()
```

```
36              showBlockList()
37          elif option == '3':
38              getTopology()
39              blockHost()
40          elif option == '4':
41              getTopology()
42              unblockHost()
43          elif option == 'q':
44              sys.exit(1)
45
46  # Get network topology making HTTP GET request to OpenDaylight API
47  def getTopology():
48      url = 'http://localhost:8181/restconf/operational/network-
    topology:network-topology'
49      storage = StringIO()
50      connection = pycurl.Curl()
51      connection.setopt(connection.URL, url)
52      connection.setopt(connection.USERPWD, 'admin:admin')
53      connection.setopt(connection.HTTPHEADER, ['Accept:
    application/xml'])
54      connection.setopt(connection.WRITEFUNCTION, storage.write)
55      connection.perform()
56      connection.close()
57
58      content = storage.getvalue()
59      topology = libxml2.parseDoc(re.sub(' xmlns="[^"]+"', '',
    storage.getvalue()))
60      nodes=topology.xpathEval('/network-topology/topology/node')
61
62      #Clear previous entries
63      for i in range(len(switch_list)):
64          switch_list.pop()
65      for i in range(len(host_list)):
66          host_list.pop()
67
68      for node in nodes:
69          node_id=str(libxml2.parseDoc(str(node)).xpathEval('/node/node-
    id/text()')[0])
70          if node_id.find("openflow")>= 0:
71              switch_list.append(node)
72          if node_id.find("host")>= 0:
73              host_list.append(node)
74
75  # Show information tables with network topology
76  def showTopology():
77      print "\n   => Number of switches: ",len(switch_list)
78      print "    => Number of hosts: ",len(host_list)
79
80      print "\n    |---------------------|"
81      print "    |---- SWITCH LIST ----|"
82      print "    |---------------------|"
83      for switch in switch_list:
84          print
    "    |     ",libxml2.parseDoc(str(switch)).xpathEval('/node/node-
    id/text()')[0],"\t |"
85      print "    |---------------------|"
86
```

```
87      print "\n   |-------------------------------------------------------
        ---------|"
88      print "   |------------------------ HOST LIST ---------------------
        -------|"
89      print "   |-------------------------------------------------------
        -------|"
90      print
        "   |         MAC         |      IP   \t|    SWITCH  \t|   PORT  |"
91      print
        "   |                     |           \t|            \t|         |"
92      for host in host_list:
93          print "   | ",
94          print str(libxml2.parseDoc(str(host)).xpathEval('/node/node-
        id/text()')[0])[5:22], " | ",
95          print
        libxml2.parseDoc(str(host)).xpathEval('/node/addresses/ip/text()')[0],
        "\t| ",
96          tpID=str(libxml2.parseDoc(str(host)).xpathEval('/node/attachme
        nt-points/tp-id/text()')[0]).split(':')
97          print tpID[0]+':'+tpID[1],"\t|   ",
98          print tpID[2],"   |"
99      print "   |-------------------------------------------------------
        -------|\n"
100
101 # Get blocked hosts making HTTP GET request to OpenDaylight API
102 def getBlocks():
103     url = 'http://localhost:8181/restconf/operational/opendaylight-
        inventory:nodes'
104     storage = StringIO()
105     connection = pycurl.Curl()
106     connection.setopt(connection.URL, url)
107     connection.setopt(connection.USERPWD, 'admin:admin')
108     connection.setopt(connection.HTTPHEADER, ['Accept:
        application/xml'])
109     connection.setopt(connection.WRITEFUNCTION, storage.write)
110     connection.perform()
111     connection.close()
112
113     content = storage.getvalue()
114     flows = libxml2.parseDoc(re.sub(' xmlns="[^"]+"', '',
        storage.getvalue())).xpathEval('//flow')
115
116     #Clear previous entries
117     for i in range(len(block_list)):
118         block_list.pop()
119
120     for flow in flows:
121         flow_id=str(libxml2.parseDoc(str(flow)).xpathEval('/flow/id/te
        xt()')[0])
122         if flow_id.find("block")>= 0:
123             block_list.append(flow_id[6:23])
124
125 # Show information table with blocked hosts
126 def showBlockList():
127     print "\n   => Number of blocked hosts: ",len(block_list)
128
129     print "\n   |-------------------------------------------------------
        ---------|"
```

```
130     print "   |-------------------- BLOCKED HOST LIST ---------------
    -------|"
131     print "   |-----------------------------------------------------
    -------|"
132     print
    "   |         MAC         |      IP   \t|    SWITCH  \t|   PORT  |"
133     print
    "   |                     |           \t|            \t|         |"
134
135     for host in host_list:
136         if str(libxml2.parseDoc(str(host)).xpathEval('/node/node-
    id/text()')[0])[5:22] in block_list:
137             print "   | ",
138             print
    str(libxml2.parseDoc(str(host)).xpathEval('/node/node-
    id/text()')[0])[5:22], " | ",
139             print
    libxml2.parseDoc(str(host)).xpathEval('/node/addresses/ip/text()')[0],
    "\t| ",
140             tpID=str(libxml2.parseDoc(str(host)).xpathEval('/node/atta
    chment-points/tp-id/text()')[0]).split(':')
141             print tpID[0]+':'+tpID[1],"\t|   ",
142             print tpID[2],"   |"
143     print "   |-----------------------------------------------------
    -------|\n"
144
145 # Block a target host knowing the mac making HTTP PUT request to
    OpenDaylight API
146 def blockHost():
147     mac = raw_input("\n   Enter the target host to block [mac]: ")
148
149     founded = False
150     for host in host_list:
151         if str(libxml2.parseDoc(str(host)).xpathEval('/node/node-
    id/text()')[0])[5:22] == mac:
152             founded = True
153             targetHost = host
154             break
155     if founded == False:
156         print "\n   => Host not founded in network topology\n"
157         return
158
159     ip =
    libxml2.parseDoc(str(targetHost)).xpathEval('/node/addresses/ip/text()
    ')[0]
160     tpID=str(libxml2.parseDoc(str(host)).xpathEval('/node/attachment-
    points/tp-id/text()')[0]).split(':')
161     switch = tpID[0]+':'+tpID[1]
162     port = tpID[2]
163
164     print "\n   => Host founded in network topology"
165     print "   ======>    MAC: ", mac
166     print "   ======>     IP: ", ip
167     print "   ======> SWITCH: ", switch
168     print "   ======>   PORT: ", port
169
170     flowId = 'block-'+ mac
```

```
171     url = 'http://localhost:8181/restconf/config/opendaylight-
    inventory:nodes/node/'+switch+'/table/0/flow/'+flowId
172     data  = '<?xml version="1.0" encoding="UTF-8"
    standalone="no"?><flow xmlns="urn:opendaylight:flow:inventory">'
173     data += '<hard-timeout>0</hard-timeout><idle-timeout>0</idle-
    timeout><priority>20</priority><flow-name>blockedHost-'+mac+'</flow-
    name>'
174     data += '<match><ethernet-match><ethernet-
    source><address>'+mac+'</address></ethernet-source></ethernet-
    match></match>'
175     data +=
    '<id>'+flowId+'</id><table_id>0</table_id><instructions><instruction><
    order>0</order><apply-actions><action>'
176     data += '<order>0</order><drop-action/></action></apply-
    actions></instruction></instructions></flow>'
177     connection = pycurl.Curl()
178     connection.setopt(connection.URL, url)
179     connection.setopt(connection.USERPWD, 'admin:admin')
180     connection.setopt(connection.HTTPHEADER, ['Content-type:
    application/xml', 'Accept: application/xml'])
181     connection.setopt(connection.CUSTOMREQUEST, 'PUT')
182     connection.setopt(connection.POSTFIELDS, data)
183     connection.perform()
184     connection.close()
185
186     print "   => Request to block host has been sended\n"
187
188 # Unblock a target host knowing the mac making HTTP DELETE request to
    OpenDaylight API
189 def unblockHost():
190     mac = raw_input("\n   Enter the target host to unblock [mac]: ")
191
192     founded = False
193     for host in host_list:
194         if str(libxml2.parseDoc(str(host)).xpathEval('/node/node-
    id/text()')[0])[5:22] == mac:
195             founded = True
196             targetHost = host
197             break
198     if founded == False:
199         print "\n   => Host not founded in network topology\n"
200         return
201
202     ip =
    libxml2.parseDoc(str(targetHost)).xpathEval('/node/addresses/ip/text()
    ')[0]
203     tpID=str(libxml2.parseDoc(str(host)).xpathEval('/node/attachment-
    points/tp-id/text()')[0]).split(':')
204     switch = tpID[0]+':'+tpID[1]
205     port = tpID[2]
206
207     print "\n   => Host founded in network topology"
208     print "   ======>    MAC: ", mac
209     print "   ======>     IP: ", ip
210     print "   ======> SWITCH: ", switch
211     print "   ======>   PORT: ", port
212
```

```
213     url = 'http://localhost:8181/restconf/config/opendaylight-
    inventory:nodes/node/'+switch+'/table/0/flow/block-'+mac
214     connection = pycurl.Curl()
215     connection.setopt(connection.URL, url)
216     connection.setopt(connection.USERPWD, 'admin:admin')
217     connection.setopt(connection.CUSTOMREQUEST, 'DELETE')
218     connection.perform()
219     connection.close()
220 
221     print "   => Request to unblock host has been sended\n"
222 
223 # Main
224 try:
225     # Lists
226     host_list=[]
227     switch_list=[]
228     block_list=[]
229 
230     os.system('clear')
231     print "\n|-----------------------------------------|"
232     print "|--- Network Access Control Application ---|"
233     print "|-----------------------------------------|"
234     menu()
235 
236 except CalledProcessError as e:
237     print "Error:", e.returncode
238     sys.exit(1)
```

Apéndice B: Escenario 2

En este apéndice se adjunta toda la información relativa al segundo escenario de demostración explicado en el trabajo. Se incluyen ficheros como la plantilla VNX para la creación del escenario que permite desplegar un entorno OpenStack con OpenDaylight, así como ficheros de configuración de DevStack y las tablas de flujos completas de las pruebas realizadas.

B.1 Plantilla del escenario

El escenario virtual utilizado ha sido creado por la herramienta VNX, y la plantilla desarrollada en lenguaje XML que lo define es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      STUDY OF SOTWARE-DEFINED NETWORKING
THROUGH THE DEVELOPMENT OF VIRTUAL SCENARIOS
      BASED ON THE OPENDAYLIGHT CONTROLLER
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Author: Raúl Álvarez Pinilla
Tutor: David Fernández Cambronero
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Telematics Engineering Department (DIT)
Technical University of Madrid (UPM)
SPAIN
-->

<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
  <global>
    <version>2.0</version>
    <scenario_name>multi-node_4nodes</scenario_name>
    <automac/>
    <vm_mgmt type="none" />
    <vm_defaults>
        <console id="0" display="no"/>
        <console id="1" display="yes"/>
    </vm_defaults>
    <cmd-seq seq='start-no-odl'>conf-no-odl,stack</cmd-seq>
    <cmd-seq seq='start-odl-l2'>start-odl,conf-odl-l2,stack</cmd-seq>
    <cmd-seq seq='start-odl-l3'>start-odl,conf-odl-l3,stack</cmd-seq>
    <cmd-seq seq='stop'>unstack,stop-odl</cmd-seq>
  </global>


  <!-- NET DEFINITION -->
```

```
36      <!-- virbr0 (192.168.122.0/24) -->
37      <net name="virbr0" mode="virtual_bridge" managed="no"/>
38      <!-- internal (10.0.0.0/24) -->
39      <net name="internal" mode="virtual_bridge"/>
40
41
42      <!-- OPENDAYLIGHT NODE -->
43      <vm name="odl" type="libvirt" subtype="kvm" os="linux"
    exec_mode="sdisk" arch="x86_64" vcpu="8">
44          <filesystem
    type="cow">filesystems/devstack_odl_ovs_KILO.qcow2</filesystem>
45          <mem>4G</mem>
46          <if id="1" net="virbr0">
47            <ipv4>dhcp,192.168.122.10</ipv4>
48          </if>
49          <if id="2" net="internal">
50            <ipv4>10.0.0.10/24</ipv4>
51          </if>
52
53          <!-- Start OpenDaylight -->
54          <exec seq="start-odl" type="verbatim" ostype="system">
55               echo "\nStarting OpenDaylight..." > /dev/ttyS0;
56               /home/vnx/opendaylight/distribution-karaf-0.2.4-
    SNAPSHOT/bin/start;
57          </exec>
58
59          <!-- Stop OpenDaylight -->
60          <exec seq="stop-odl" type="verbatim" ostype="system">
61               echo "\nStopping OpenDaylight..." > /dev/ttyS0;
62               /home/vnx/opendaylight/distribution-karaf-0.2.4-
    SNAPSHOT/bin/stop;
63          </exec>
64      </vm>
65
66
67      <!-- CONTROLLER NODE (Controller + Network + Compute1) -->
68      <vm name="ctrl-comp1" type="libvirt" subtype="kvm" os="linux"
    exec_mode="sdisk" arch="x86_64" vcpu="8">
69          <filesystem
    type="cow">filesystems/devstack_odl_ovs_KILO.qcow2</filesystem>
70          <mem>4G</mem>
71          <if id="1" net="virbr0">
72            <ipv4>dhcp,192.168.122.11</ipv4>
73          </if>
74          <if id="2" net="internal">
75            <ipv4>10.0.0.11/24</ipv4>
76          </if>
77
78          <!-- Copy DevStack configuration (DevStack, without
    OpenDaylight)-->
79          <filetree seq="conf-no-odl" root="/home/vnx/devstack/local.conf"
    perms="644">conf/no-odl/local.conf.ctrl-comp1</filetree>
80
81          <!-- Copy DevStack configuration (DevStack with OpenDaylight [L2
    services])-->
82          <filetree seq="conf-odl-l2" root="/home/vnx/devstack/local.conf"
    perms="644">conf/odl-l2/local.conf.ctrl-comp1</filetree>
83
```

```
84      <!-- Copy DevStack configuration (DevStack with OpenDaylight [L2
        & L3 services])-->
85      <filetree seq="conf-odl-l3" root="/home/vnx/devstack/local.conf"
        perms="644">conf/odl-l3/local.conf.ctrl-comp1</filetree>
86
87      <!-- Copy DevStack L2 Scenario -->
88      <filetree seq="create-l2"
        root="/home/vnx/devstack/createL2Scenario.sh" perms="777">scripts-
        devstack/createL2Scenario_multi-node_4nodes.sh</filetree>
89
90      <!-- Copy DevStack L3 Scenario -->
91      <filetree seq="create-l3"
        root="/home/vnx/devstack/createL3Scenario.sh" perms="777">scripts-
        devstack/createL3Scenario_multi-node_4nodes.sh</filetree>
92
93      <!-- Start DevStack -->
94      <exec seq="stack" type="verbatim" ostype="system">
95       su vnx -c "/home/vnx/devstack/stack.sh" > /dev/ttyS0;
96      </exec>
97
98      <!-- Stop DevStack -->
99      <exec seq="unstack" type="verbatim" ostype="system">
100          su vnx -c "/home/vnx/devstack/unstack.sh > /dev/ttyS0";
101     </exec>
102
103     <!-- Create DevStack L2 Scenario-->
104     <exec seq="create-l3" type="verbatim" ostype="system">
105          /home/vnx/devstack/createL2Scenario.sh > /dev/ttyS0;
106     </exec>
107
108     <!-- Create DevStack L3 Scenario-->
109     <exec seq="create-l3" type="verbatim" ostype="system">
110          /home/vnx/devstack/createL3Scenario.sh > /dev/ttyS0;
111     </exec>
112   </vm>
113
114
115   <!-- COMPUTE NODE (Compute2)-->
116   <vm name="comp2" type="libvirt" subtype="kvm" os="linux"
      exec_mode="sdisk" arch="x86_64" vcpu="8">
117     <filesystem
      type="cow">filesystems/devstack_odl_ovs_KILO.qcow2</filesystem>
118     <mem>2G</mem>
119     <if id="1" net="virbr0">
120       <ipv4>dhcp,192.168.122.12</ipv4>
121     </if>
122     <if id="2" net="internal">
123       <ipv4>10.0.0.12/24</ipv4>
124     </if>
125
126     <!-- Copy DevStack configuration (DevStack, without
      OpenDaylight)-->
127     <filetree seq="conf-no-odl" root="/home/vnx/devstack/local.conf"
      perms="644">conf/no-odl/local.conf.comp2</filetree>
128
129     <!-- Copy DevStack configuration (DevStack with OpenDaylight [L2
      services])-->
130     <filetree seq="conf-odl-l2" root="/home/vnx/devstack/local.conf"
      perms="644">conf/odl-l2/local.conf.comp2</filetree>
```

```
131
132       <!-- Copy DevStack configuration (DevStack with OpenDaylight [L2
      & L3 services])-->
133       <filetree seq="conf-odl-l3" root="/home/vnx/devstack/local.conf"
      perms="644">conf/odl-l3/local.conf.comp2</filetree>
134
135       <!-- Start DevStack -->
136       <exec seq="stack" type="verbatim" ostype="system">
137            su vnx -c "/home/vnx/devstack/stack.sh > /dev/ttyS0";
138       </exec>
139
140       <!-- Stop DevStack -->
141       <exec seq="unstack" type="verbatim" ostype="system">
142            su vnx -c "/home/vnx/devstack/unstack.sh > /dev/ttyS0";
143       </exec>
144     </vm>
145
146
147     <!-- COMPUTE NODE (Compute3)-->
148     <vm name="comp3" type="libvirt" subtype="kvm" os="linux"
      exec_mode="sdisk" arch="x86_64" vcpu="8">
149       <filesystem
      type="cow">filesystems/devstack_odl_ovs_KILO.qcow2</filesystem>
150       <mem>2G</mem>
151       <if id="1" net="virbr0">
152         <ipv4>dhcp,192.168.122.13</ipv4>
153       </if>
154       <if id="2" net="internal">
155         <ipv4>10.0.0.13/24</ipv4>
156       </if>
157
158       <!-- Copy DevStack configuration (DevStack, without
      OpenDaylight)-->
159       <filetree seq="conf-no-odl" root="/home/vnx/devstack/local.conf"
      perms="644">conf/no-odl/local.conf.comp3</filetree>
160
161       <!-- Copy DevStack configuration (DevStack with OpenDaylight [L2
      services])-->
162       <filetree seq="conf-odl-l2" root="/home/vnx/devstack/local.conf"
      perms="644">conf/odl-l2/local.conf.comp3</filetree>
163
164       <!-- Copy DevStack configuration (DevStack with OpenDaylight [L2
      & L3 services])-->
165       <filetree seq="conf-odl-l3" root="/home/vnx/devstack/local.conf"
      perms="644">conf/odl-l3/local.conf.comp3</filetree>
166
167       <!-- Start DevStack -->
168       <exec seq="stack" type="verbatim" ostype="system">
169            su vnx -c "/home/vnx/devstack/stack.sh > /dev/ttyS0";
170       </exec>
171
172       <!-- Stop DevStack -->
173       <exec seq="unstack" type="verbatim" ostype="system">
174            su vnx -c "/home/vnx/devstack/unstack.sh > /dev/ttyS0";
175       </exec>
176     </vm>
177
178   </vnx>
```

B.2 Creación de redes con libvirt

Libvirt es una API de virtualización que facilita, entre otras cosas, la creación de redes virtuales. En el escenario de demostración donde se despliegue un entorno OpenStack se utiliza dicha API para la simulación de una red externa dentro del escenario virtualizado. En la instalación de libvirt se crea por defecto la red *virbr0*, que es la utilizada en el escenario. Si no se dispone de ella en el equipo o se desea crear una nueva red con su switch virtual correspondiente se puede definir según se indica a continuación:

1. Creación de un fichero .xml especificando la red a crear en el directorio */etc/libvirt/qemu/network*. A continuación se muestra un ejemplo del fichero correspondiente de la red preconfigurada *default*:

```
<network>
  <name>default</name>
  <bridge name="virbr0" />
  <forward/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254" />
    </dhcp>
  </ip>
</network>
```

2. Definición de la red

```
# virsh net-define /etc/libvirt/qemu/network/default.xml
La red default se encuentra definida desde default.xml
```

3. Configuración de auto-inicio de la red (opcional)

```
# virsh net-autostart default
La red default se ha sido marcada para iniciarse automáticamente
```

4. Inicio de la red

```
# virsh net-start default
La red default se ha iniciado
```

Si se desea borrar una red existente se puede hacer uso del siguiente comando:

```
# virsh net-destroy default
La red default ha sido destruida
```

B.3 Construcción del sistema de ficheros

Las máquinas virtuales del escenario de demostración utilizan un único sistema de ficheros que contiene todas las herramientas necesarias, como OpenDaylight, Open vSwitch y DevStack. Los pasos a seguir para la creación del sistema de ficheros, que se encuentra disponible en el repositorio [9], son:

```
1) Descargar KVM Ubuntu 14.04 VNX rootfs (64 bits)

        vnx_download_rootfsvnx_rootfs_kvm_ubuntu64-14.04-v025.qcow2.bz2

2) Modificar el sistema de ficheros descargado

        vnx --modify-rootfs vnx_rootfs_kvm_ubuntu64-14.04-v025.qcow2
        --mem 4G --vcpu 8 --arch x86_64

3) Instalar OpenDaylight (Karaf 0.2.4)

        apt-get install maven openjdk-7-jre openjdk-7-jdk unzip wget

        wget https://nexus.opendaylight.org/content/repositories/
             opendaylight.snapshot/org/opendaylight/integration/distribution
             -karaf/0.2.4-SNAPSHOT/distribution-karaf-0.2.4-20150530.024323-
             26.zip

        unzip distribution-karaf-*.zip

        bin/karaf

        feature:install odl-base-all odl-aaa-authn odl-restconf
                 odl-nsf-all odl-adsal-northbound odl-mdsal-apidocs
                 odl-ovsdb-openstack odl-ovsdb-northbound odl-dlux-core

4) Instalar Open vSwitch 2.3.1

        apt-get install autoconf automake build-essential debhelper dkms
                 fakeroot graphviz ipsec-tools iputils-arping iputils-ping
                 libssl-dev libtool module-assistant python-all python-qt4
                 python-twisted-conch python-twisted-web raccoon

        wget http://openvswitch.org/releases/openvswitch-2.3.1.tar.gz

        tar xvf openvswitch-2.3.1.tar.gz

        DEB_BUILD_OPTIONS='nocheck' fakeroot debian/rules binary

        sudo dpkg -i ../*.deb

5) Instalar DevStack

        apt-get install git

        git clone -b stable/kilo https://git.openstack.org/openstack-
                                  dev/devstack
```

[9] https://github.com/ralvarep/odl-openstack/tree/master/filesystems

B.4 Fichero de configuración del nodo controlador

El fichero *local.conf* que define la creación del nodo de control del escenario OpenStack mediante la herramienta DevStack es el siguiente:

```
[[local|localrc]]

# Prevent refreshing of dependencies and DevStack recloning
OFFLINE=True
RECLONE=No

# Destination path for installation
DEST=/opt/stack

# Log Options
LOGFILE=$DEST/logs/stack.sh.log
SCREEN_LOGDIR=$DEST/logs/stack
LOG_COLOR=True

# Credentials
ADMIN_PASSWORD=password
MYSQL_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=token

# IP Details
HOST_IP=10.0.0.11
HOST_NAME=ctrl-comp1
SERVICE_HOST=$HOST_IP
SERVICE_HOST_NAME=$HOST_NAME

# Services
disable_all_services
ENABLED_SERVICES=key
ENABLED_SERVICES+=,n-api,n-cpu,n-cond,n-sch,n-novnc,n-crt,n-cauth
ENABLED_SERVICES+=,g-api,g-reg
ENABLED_SERVICES+=,horizon
ENABLED_SERVICES+=,rabbit,mysql
ENABLED_SERVICES+=,q-svc,q-dhcp,q-meta

# Images (Cirros 0.3.4)
IMAGE_URLS="http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-uec.tar.gz"

# VNC Server
VNCSERVER_PROXYCLIENT_ADDRESS=$HOST_IP
VNCSERVER_LISTEN=0.0.0.0
NOVNCPROXY_URL="http://192.168.122.11:6080/vnc_auto.html"

# Public Network
FLOATING_RANGE=192.168.122.0/24
PUBLIC_NETWORK_GATEWAY=192.168.122.1

# Neutron
NEUTRON_CREATE_INITIAL_NETWORKS=False
Q_PLUGIN=ml2
```

```
52  ENABLE_TENANT_TUNNELS=True
53  Q_ML2_TENANT_NETWORK_TYPE=vxlan,gre
54  #enable_plugin networking-odl
    http://git.openstack.org/openstack/networking-odl
55  enable_plugin networking-odl https://github.com/flavio-
    fernandes/networking-odl helium
56
57  # OpenDaylight
58  ODL_MODE=externalodl
59  ODL_MGR_IP=10.0.0.10
60  ODL_MGR_PORT=6640
61  ODL_USERNAME=admin
62  ODL_PASSWORD=admin
63  ODL_PORT=8080
64  ODL_ENDPOINT="http://${ODL_MGR_IP}:${ODL_PORT}/controller/nb/v2/neutron
    "
65
66  # L3-OpenDaylight
67  disable_service q-l3
68  Q_L3_ENABLED=True
69  ODL_L3=True
70  [[post-config|$NEUTRON_CONF]]
71  [DEFAULT]
72  service_plugins=networking_odl.l3.l3_odl.OpenDaylightL3RouterPlugin
73
74  # ML2 Configuration
75  [[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
76  [agent]
77  minimize_polling=True
```

B.5 Fichero de configuración de los nodos de computación

El fichero *local.conf* que define la creación de un nodo de computación del escenario OpenStack mediante la herramienta DevStack es el siguiente:

```
1   [[local|localrc]]
2
3   # Prevent refreshing of dependencies and DevStack recloning
4   OFFLINE=True
5   RECLONE=No
6
7   # Destination path for installation
8   DEST=/opt/stack
9
10  # Log Options
11  LOGFILE=$DEST/logs/stack.sh.log
12  SCREEN_LOGDIR=$DEST/logs/stack
13  LOG_COLOR=True
14
15  # Credentials
16  ADMIN_PASSWORD=password
17  MYSQL_PASSWORD=$ADMIN_PASSWORD
18  RABBIT_PASSWORD=$ADMIN_PASSWORD
19  SERVICE_PASSWORD=$ADMIN_PASSWORD
20  SERVICE_TOKEN=token
```

```
21
22  # IP Details
23  HOST_IP=10.0.0.12
24  HOST_NAME=comp2
25  SERVICE_HOST=10.0.0.11
26  SERVICE_HOST_NAME=ctrl-comp1
27
28  # Services
29  disable_all_services
30  ENABLED_SERVICES=n-cpu,n-novnc,rabbit
31
32  # VNC Server
33  VNCSERVER_PROXYCLIENT_ADDRESS=$HOST_IP
34  VNCSERVER_LISTEN=0.0.0.0
35  NOVNCPROXY_URL="http://192.168.122.11:6080/vnc_auto.html"
36
37  # Neutron
38  Q_PLUGIN=ml2
39  ENABLE_TENANT_TUNNELS=True
40  Q_ML2_TENANT_NETWORK_TYPE=gre,vxlan
41  enable_plugin networking-odl https://github.com/stackforge/networking-odl
42
43  # OpenDaylight
44  ODL_MODE=compute
45  ODL_MGR_IP=10.0.0.10
46  ODL_MGR_PORT=6640
47  ODL_USERNAME=admin
48  ODL_PASSWORD=admin
49  ODL_PORT=8080
50  ODL_ENDPOINT="http://${ODL_MGR_IP}:${ODL_PORT}/controller/nb/v2/neutron"
```

B.6 Tablas de flujos de los nodos de computación (L2)

Tras la creación de una red local junto con tres máquinas virtuales en el entorno de OpenStack, como se mostró en la figura 4.18, se obtienen las siguientes tablas de flujo en los diferentes nodos:

```
vnx@ctrl-comp1:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=130.090s, table=0, n_packets=14, n_bytes=2799, in_port=1,dl_src=fa:16:3e:8b:a9:f6
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=116.948s, table=0, n_packets=26, n_bytes=1898, in_port=4,dl_src=fa:16:3e:58:6a:81
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7816.834s, table=0, n_packets=0, n_bytes=0, priority=0 actions=goto_table:20
cookie=0x0, duration=129.600s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=1 actions=drop
cookie=0x0, duration=116.447s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=4 actions=drop
cookie=0x0, duration=126.498s, table=0, n_packets=24, n_bytes=1626, tun_id=0x65,in_port=2
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=124.990s, table=0, n_packets=25, n_bytes=1696, tun_id=0x65,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7817.688s, table=0, n_packets=56, n_bytes=6328, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=7816.327s, table=20, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:30
cookie=0x0, duration=7815.821s, table=30, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:40
cookie=0x0, duration=7815.315s, table=40, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:50
cookie=0x0, duration=7814.807s, table=50, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:60
cookie=0x0, duration=7814.298s, table=60, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:70
cookie=0x0, duration=7813.793s, table=70, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:80
cookie=0x0, duration=7813.282s, table=80, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:90
cookie=0x0, duration=7812.776s, table=90, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:100
cookie=0x0, duration=7812.273s, table=100, n_packets=89, n_bytes=8019, priority=0 actions=goto_table:110
cookie=0x0, duration=127.007s, table=110, n_packets=0, n_bytes=0, priority=8192,tun_id=0x65 actions=drop
cookie=0x0, duration=7811.759s, table=110, n_packets=4, n_bytes=328, priority=0 actions=drop
cookie=0x0, duration=128.511s, table=110, n_packets=49, n_bytes=3322,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:1,output:4
cookie=0x0, duration=128.012s, table=110, n_packets=30, n_bytes=2218,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:1,output:2,output:3,output:4
cookie=0x0, duration=86.198s, table=110, n_packets=2, n_bytes=717, tun_id=0x65,dl_dst=fa:16:3e:02:69:f8 actions=output:2
cookie=0x0, duration=115.946s, table=110, n_packets=2, n_bytes=717, tun_id=0x65,dl_dst=fa:16:3e:58:6a:81 actions=output:4
cookie=0x0, duration=99.316s, table=110, n_packets=2, n_bytes=717, tun_id=0x65,dl_dst=fa:16:3e:71:73:50 actions=output:3
cookie=0x0, duration=129.091s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:8b:a9:f6 actions=output:1
```

```
vnx@comp2:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=824.495s, table=0, n_packets=28, n_bytes=1934, in_port=2,dl_src=fa:16:3e:71:73:50
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=8515.454s, table=0, n_packets=15, n_bytes=1753, priority=0 actions=goto_table:20
cookie=0x0, duration=823.991s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=2 actions=drop
cookie=0x0, duration=820.980s, table=0, n_packets=17, n_bytes=1423, tun_id=0x65,in_port=1
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=819.475s, table=0, n_packets=24, n_bytes=1626, tun_id=0x65,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=8516.436s, table=0, n_packets=337, n_bytes=37739, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=8514.951s, table=20, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:30
cookie=0x0, duration=8514.445s, table=30, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:40
cookie=0x0, duration=8513.938s, table=40, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:50
cookie=0x0, duration=8513.437s, table=50, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:60
cookie=0x0, duration=8512.930s, table=60, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:70
cookie=0x0, duration=8512.428s, table=70, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:80
cookie=0x0, duration=8511.921s, table=80, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:90
cookie=0x0, duration=8511.419s, table=90, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:100
cookie=0x0, duration=8510.913s, table=100, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:110
cookie=0x0, duration=821.483s, table=110, n_packets=1, n_bytes=42, priority=8192,tun_id=0x65 actions=drop
cookie=0x0, duration=8510.407s, table=110, n_packets=17, n_bytes=1949, priority=0 actions=drop
cookie=0x0, duration=822.986s, table=110, n_packets=39, n_bytes=2332,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2
cookie=0x0, duration=822.484s, table=110, n_packets=25, n_bytes=1696,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2,output:1,output:3
cookie=0x0, duration=805.343s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:02:69:f8 actions=output:3
cookie=0x0, duration=831.586s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:58:6a:81 actions=output:1
cookie=0x0, duration=823.488s, table=110, n_packets=2, n_bytes=717, tun_id=0x65,dl_dst=fa:16:3e:71:73:50 actions=output:2
cookie=0x0, duration=844.643s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:8b:a9:f6 actions=output:1
```

```
vnx@comp3:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=854.633s, table=0, n_packets=25, n_bytes=1696, in_port=2,dl_src=fa:16:3e:02:69:f8
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=8533.762s, table=0, n_packets=29, n_bytes=2557, priority=0 actions=goto_table:20
cookie=0x0, duration=854.131s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=2 actions=drop
cookie=0x0, duration=851.106s, table=0, n_packets=7, n_bytes=927, tun_id=0x65,in_port=1
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=849.590s, table=0, n_packets=23, n_bytes=1556, tun_id=0x65,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=8534.735s, table=0, n_packets=353, n_bytes=39529, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=8533.260s, table=20, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:30
cookie=0x0, duration=8532.757s, table=30, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:40
cookie=0x0, duration=8532.255s, table=40, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:50
cookie=0x0, duration=8531.748s, table=50, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:60
cookie=0x0, duration=8531.246s, table=60, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:70
cookie=0x0, duration=8530.741s, table=70, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:80
cookie=0x0, duration=8530.236s, table=80, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:90
cookie=0x0, duration=8529.730s, table=90, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:100
cookie=0x0, duration=8529.226s, table=100, n_packets=84, n_bytes=6736, priority=0 actions=goto_table:110
cookie=0x0, duration=851.612s, table=110, n_packets=1, n_bytes=42, priority=8192,tun_id=0x65 actions=drop
cookie=0x0, duration=8528.722s, table=110, n_packets=29, n_bytes=2585, priority=0 actions=drop
cookie=0x0, duration=853.119s, table=110, n_packets=28, n_bytes=1766,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2
cookie=0x0, duration=852.616s, table=110, n_packets=24, n_bytes=1626,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2,output:1,output:3
cookie=0x0, duration=853.627s, table=110, n_packets=2, n_bytes=717, tun_id=0x65,dl_dst=fa:16:3e:02:69:f8 actions=output:2
cookie=0x0, duration=876.330s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:58:6a:81 actions=output:1
cookie=0x0, duration=861.708s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:71:73:50 actions=output:3
cookie=0x0, duration=889.389s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:8b:a9:f6 actions=output:1
```

B.7 Tablas de flujos de los nodos de computación (L3)

Tras la creación de dos redes local con tres máquinas virtuales cada una e interconectadas entre sí mediante un router en el entorno de OpenStack, como se mostró en la figura 4.23, se obtienen las siguientes tablas de flujo en los diferentes nodos:

```
vnx@ctrl-comp1:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13

OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=6744.114s, table=0, n_packets=32, n_bytes=3218, in_port=6,dl_src=fa:16:3e:87:c1:43
 actions=set_field:0x66->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=6757.190s, table=0, n_packets=10, n_bytes=2375, in_port=5,dl_src=fa:16:3e:9a:7b:72
 actions=set_field:0x66->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=6796.493s, table=0, n_packets=33, n_bytes=2360, in_port=4,dl_src=fa:16:3e:6c:1e:ae
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=6809.649s, table=0, n_packets=14, n_bytes=2799, in_port=1,dl_src=fa:16:3e:35:3a:a9
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7686.028s, table=0, n_packets=10, n_bytes=1077, priority=0 actions=goto_table:20
cookie=0x0, duration=6743.612s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=6 actions=drop
cookie=0x0, duration=6756.696s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=5 actions=drop
cookie=0x0, duration=6809.161s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=1 actions=drop
cookie=0x0, duration=6795.992s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=4 actions=drop
cookie=0x0, duration=6806.070s, table=0, n_packets=11, n_bytes=1156, tun_id=0x65,in_port=2
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=6753.683s, table=0, n_packets=4, n_bytes=870, tun_id=0x66,in_port=2
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=6804.553s, table=0, n_packets=20, n_bytes=1562, tun_id=0x65,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=6752.174s, table=0, n_packets=5, n_bytes=912, tun_id=0x66,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7686.867s, table=0, n_packets=1978, n_bytes=219558, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=7685.519s, table=20, n_packets=131, n_bytes=15993, priority=0 actions=goto_table:30
cookie=0x0, duration=6693.410s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.3
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:6c:1e:ae->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e6c1eae->NXM_NX_ARP_SHA[],
 load:0xa650003->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6699.503s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.5
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:24:c6:a0->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e24c6a0->NXM_NX_ARP_SHA[],
 load:0xa650005->NXM_OF_ARP_SPA[],IN_PORT
```

```
cookie=0x0, duration=6696.456s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.4
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:6c:75:f1->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e6c75f1->NXM_NX_ARP_SHA[],
 load:0xa650004->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6690.366s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.2
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:35:3a:a9->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e353aa9->NXM_NX_ARP_SHA[],
 load:0xa650002->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6670.484s, table=20, n_packets=2, n_bytes=84, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.5
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:05:0a:6f->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e050a6f->NXM_NX_ARP_SHA[],
 load:0xa660005->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6673.535s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.4
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:d9:79:34->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163ed97934->NXM_NX_ARP_SHA[],
 load:0xa660004->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6704.582s, table=20, n_packets=3, n_bytes=126, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.1
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:19:89:21->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e198921->NXM_NX_ARP_SHA[],
 load:0xa650001->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6686.783s, table=20, n_packets=3, n_bytes=126, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.1
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:04:35:cb->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e0435cb->NXM_NX_ARP_SHA[],
 load:0xa660001->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6679.673s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.3
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:87:c1:43->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e87c143->NXM_NX_ARP_SHA[],
 load:0xa660003->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=6676.612s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.2
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:9a:7b:72->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e9a7b72->NXM_NX_ARP_SHA[],
 load:0xa660002->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7685.011s, table=30, n_packets=116, n_bytes=13481, priority=0 actions=goto_table:40
```

```
cookie=0x0, duration=6686.283s, table=30, n_packets=15, n_bytes=2512, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.0/24
 actions=goto_table:40
cookie=0x0, duration=6704.079s, table=30, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.0/24
 actions=goto_table:40
cookie=0x0, duration=7684.499s, table=40, n_packets=131, n_bytes=15993, priority=0 actions=goto_table:50
cookie=0x0, duration=7683.991s, table=50, n_packets=131, n_bytes=15993, priority=0 actions=goto_table:60
cookie=0x0, duration=7683.481s, table=60, n_packets=123, n_bytes=15209, priority=0 actions=goto_table:70
cookie=0x0, duration=6687.811s, table=60, n_packets=3, n_bytes=294, priority=2048,ip,tun_id=0x65,nw_dst=10.102.0.0/24
 actions=set_field:fa:16:3e:04:35:cb->eth_src,dec_ttl,set_field:0x66->tun_id,goto_table:70
cookie=0x0, duration=6687.301s, table=60, n_packets=5, n_bytes=490, priority=2048,ip,tun_id=0x66,nw_dst=10.101.0.0/24
 actions=set_field:fa:16:3e:19:89:21->eth_src,dec_ttl,set_field:0x65->tun_id,goto_table:70
cookie=0x0, duration=7682.970s, table=70, n_packets=117, n_bytes=14621, priority=0 actions=goto_table:80
cookie=0x0, duration=6700.013s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.5
 actions=set_field:fa:16:3e:24:c6:a0->eth_dst,goto_table:80
cookie=0x0, duration=6693.903s, table=70, n_packets=3, n_bytes=294, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.3
 actions=set_field:fa:16:3e:6c:1e:ae->eth_dst,goto_table:80
cookie=0x0, duration=6677.125s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.2
 actions=set_field:fa:16:3e:9a:7b:72->eth_dst,goto_table:80
cookie=0x0, duration=6674.047s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.4
 actions=set_field:fa:16:3e:d9:79:34->eth_dst,goto_table:80
cookie=0x0, duration=6696.966s, table=70, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.4
 actions=set_field:fa:16:3e:6c:75:f1->eth_dst,goto_table:80
cookie=0x0, duration=6690.879s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.2
 actions=set_field:fa:16:3e:35:3a:a9->eth_dst,goto_table:80
cookie=0x0, duration=6680.187s, table=70, n_packets=7, n_bytes=686, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.3
 actions=set_field:fa:16:3e:87:c1:43->eth_dst,goto_table:80
cookie=0x0, duration=6670.994s, table=70, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.5
 actions=set_field:fa:16:3e:05:0a:6f->eth_dst,goto_table:80
cookie=0x0, duration=7682.464s, table=80, n_packets=131, n_bytes=15993, priority=0 actions=goto_table:90
cookie=0x0, duration=7681.963s, table=90, n_packets=131, n_bytes=15993, priority=0 actions=goto_table:100
cookie=0x0, duration=7681.459s, table=100, n_packets=108, n_bytes=12697, priority=0 actions=goto_table:110
cookie=0x0, duration=6685.774s, table=100, n_packets=18, n_bytes=2806, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.0/24
 actions=goto_table:110
```

```
cookie=0x0, duration=6703.573s, table=100, n_packets=5, n_bytes=490, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.0/24
 actions=goto_table:110
cookie=0x0, duration=6806.574s, table=110, n_packets=0, n_bytes=0, priority=8192,tun_id=0x65 actions=drop
cookie=0x0, duration=6754.185s, table=110, n_packets=15, n_bytes=1418, priority=8192,tun_id=0x66 actions=drop
cookie=0x0, duration=7680.923s, table=110, n_packets=15, n_bytes=1475, priority=0 actions=drop
cookie=0x0, duration=6808.075s, table=110, n_packets=29, n_bytes=2634,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:1,output:4
cookie=0x0, duration=6755.692s, table=110, n_packets=4, n_bytes=1348,
 priority=16384,reg0=0x2,tun_id=0x66,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:5,output:6
cookie=0x0, duration=6755.189s, table=110, n_packets=7, n_bytes=1016,
 priority=16384,reg0=0x1,tun_id=0x66,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:5,output:2,output:3,output:6
cookie=0x0, duration=6807.578s, table=110, n_packets=33, n_bytes=2344,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:1,output:2,output:3,output:4
cookie=0x0, duration=6765.756s, table=110, n_packets=2, n_bytes=717, tun_id=0x65,dl_dst=fa:16:3e:24:c6:a0 actions=output:2
cookie=0x0, duration=6713.290s, table=110, n_packets=4, n_bytes=913, tun_id=0x66,dl_dst=fa:16:3e:05:0a:6f actions=output:2
cookie=0x0, duration=6756.194s, table=110, n_packets=1, n_bytes=42, tun_id=0x66,dl_dst=fa:16:3e:9a:7b:72 actions=output:5
cookie=0x0, duration=6743.108s, table=110, n_packets=9, n_bytes=1403, tun_id=0x66,dl_dst=fa:16:3e:87:c1:43 actions=output:6
cookie=0x0, duration=6795.490s, table=110, n_packets=5, n_bytes=1011, tun_id=0x65,dl_dst=fa:16:3e:6c:1e:ae actions=output:4
cookie=0x0, duration=6808.659s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:35:3a:a9 actions=output:1
cookie=0x0, duration=6778.847s, table=110, n_packets=4, n_bytes=913, tun_id=0x65,dl_dst=fa:16:3e:6c:75:f1 actions=output:3
cookie=0x0, duration=6726.452s, table=110, n_packets=3, n_bytes=759, tun_id=0x66,dl_dst=fa:16:3e:d9:79:34 actions=output:3
```

```
vnx@comp2:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=7486.155s, table=0, n_packets=19, n_bytes=2176, in_port=4,dl_src=fa:16:3e:d9:79:34
 actions=set_field:0x66->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7538.539s, table=0, n_packets=32, n_bytes=2874, in_port=2,dl_src=fa:16:3e:6c:75:f1
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7593.978s, table=0, n_packets=30, n_bytes=2731, priority=0 actions=goto_table:20
cookie=0x0, duration=7538.035s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=2 actions=drop
cookie=0x0, duration=7485.649s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=4 actions=drop
cookie=0x0, duration=7482.617s, table=0, n_packets=6, n_bytes=1475, tun_id=0x66,in_port=1
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7535.010s, table=0, n_packets=16, n_bytes=1417, tun_id=0x65,in_port=1
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7533.495s, table=0, n_packets=11, n_bytes=1156, tun_id=0x65,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7481.097s, table=0, n_packets=2, n_bytes=674, tun_id=0x66,in_port=3
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7594.967s, table=0, n_packets=1223, n_bytes=135753, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=7593.475s, table=20, n_packets=114, n_bytes=12419, priority=0 actions=goto_table:30
cookie=0x0, duration=7446.536s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.3
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:6c:1e:ae->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e6c1eae->NXM_NX_ARP_SHA[],
 load:0xa650003->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7452.626s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.5
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:24:c6:a0->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e24c6a0->NXM_NX_ARP_SHA[],
 load:0xa650005->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7449.576s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.4
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:6c:75:f1->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e6c75f1->NXM_NX_ARP_SHA[],
 load:0xa650004->NXM_OF_ARP_SPA[],IN_PORT
```

```
cookie=0x0, duration=7443.492s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.2
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:35:3a:a9->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e353aa9->NXM_NX_ARP_SHA[],
 load:0xa650002->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7423.582s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.5
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:05:0a:6f->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e050a6f->NXM_NX_ARP_SHA[],
 load:0xa660005->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7426.653s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.4
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:d9:79:34->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163ed97934->NXM_NX_ARP_SHA[],
 load:0xa660004->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7456.700s, table=20, n_packets=1, n_bytes=42, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.1
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:19:89:21->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e198921->NXM_NX_ARP_SHA[],
 load:0xa650001->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7436.867s, table=20, n_packets=1, n_bytes=42, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.1
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:04:35:cb->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e0435cb->NXM_NX_ARP_SHA[],
 load:0xa660001->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7432.785s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.3
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:87:c1:43->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e87c143->NXM_NX_ARP_SHA[],
 load:0xa660003->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7429.703s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.2
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:9a:7b:72->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e9a7b72->NXM_NX_ARP_SHA[],
 load:0xa660002->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7592.969s, table=30, n_packets=106, n_bytes=11635, priority=0 actions=goto_table:40
cookie=0x0, duration=7436.361s, table=30, n_packets=6, n_bytes=588, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.0/24
 actions=goto_table:40
cookie=0x0, duration=7456.198s, table=30, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.0/24
 actions=goto_table:40
cookie=0x0, duration=7592.464s, table=40, n_packets=114, n_bytes=12419, priority=0 actions=goto_table:50
```

```
 cookie=0x0, duration=7591.964s, table=50, n_packets=114, n_bytes=12419, priority=0 actions=goto_table:60
 cookie=0x0, duration=7591.458s, table=60, n_packets=112, n_bytes=12223, priority=0 actions=goto_table:70
 cookie=0x0, duration=7437.878s, table=60, n_packets=2, n_bytes=196, priority=2048,ip,tun_id=0x65,nw_dst=10.102.0.0/24
  actions=set_field:fa:16:3e:04:35:cb->eth_src,dec_ttl,set_field:0x66->tun_id,goto_table:70
 cookie=0x0, duration=7437.370s, table=60, n_packets=0, n_bytes=0, priority=2048,ip,tun_id=0x66,nw_dst=10.101.0.0/24
  actions=set_field:fa:16:3e:19:89:21->eth_src,dec_ttl,set_field:0x65->tun_id,goto_table:70
 cookie=0x0, duration=7590.936s, table=70, n_packets=110, n_bytes=12027, priority=0 actions=goto_table:80
 cookie=0x0, duration=7453.136s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.5
  actions=set_field:fa:16:3e:24:c6:a0->eth_dst,goto_table:80
 cookie=0x0, duration=7447.049s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.3
  actions=set_field:fa:16:3e:6c:1e:ae->eth_dst,goto_table:80
 cookie=0x0, duration=7430.218s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.2
  actions=set_field:fa:16:3e:9a:7b:72->eth_dst,goto_table:80
 cookie=0x0, duration=7427.167s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.4
  actions=set_field:fa:16:3e:d9:79:34->eth_dst,goto_table:80
 cookie=0x0, duration=7450.086s, table=70, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.4
  actions=set_field:fa:16:3e:6c:75:f1->eth_dst,goto_table:80
 cookie=0x0, duration=7443.995s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.2
  actions=set_field:fa:16:3e:35:3a:a9->eth_dst,goto_table:80
 cookie=0x0, duration=7433.301s, table=70, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.3
  actions=set_field:fa:16:3e:87:c1:43->eth_dst,goto_table:80
 cookie=0x0, duration=7424.110s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.5
  actions=set_field:fa:16:3e:05:0a:6f->eth_dst,goto_table:80
 cookie=0x0, duration=7590.453s, table=80, n_packets=114, n_bytes=12419, priority=0 actions=goto_table:90
 cookie=0x0, duration=7589.950s, table=90, n_packets=114, n_bytes=12419, priority=0 actions=goto_table:100
 cookie=0x0, duration=7589.447[ 8873.023030] serial8250: too much work for irq4
s, table=100, n_packets=105, n_bytes=11537, priority=0 actions=goto_table:110
 cookie=0x0, duration=7435.855s, table=100, n_packets=7, n_bytes=686, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.0/24
  actions=goto_table:110
 cookie=0x0, duration=7455.693s, table=100, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.0/24
  actions=goto_table:110
 cookie=0x0, duration=7535.518s, table=110, n_packets=8, n_bytes=687, priority=8192,tun_id=0x65 actions=drop
 cookie=0x0, duration=7483.123s, table=110, n_packets=15, n_bytes=1418, priority=8192,tun_id=0x66 actions=drop
 cookie=0x0, duration=7588.944s, table=110, n_packets=28, n_bytes=2619, priority=0 actions=drop
```

```
cookie=0x0, duration=7537.026s, table=110, n_packets=22, n_bytes=1618,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2
cookie=0x0, duration=7484.638s, table=110, n_packets=5, n_bytes=1390,
 priority=16384,reg0=0x2,tun_id=0x66,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:4
cookie=0x0, duration=7484.132s, table=110, n_packets=2, n_bytes=674,
 priority=16384,reg0=0x1,tun_id=0x66,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:4,output:1,output:3
cookie=0x0, duration=7536.522s, table=110, n_packets=23, n_bytes=2061,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2,output:1,output:3
cookie=0x0, duration=7519.397s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:24:c6:a0 actions=output:3
cookie=0x0, duration=7466.926s, table=110, n_packets=0, n_bytes=0, tun_id=0x66,dl_dst=fa:16:3e:05:0a:6f actions=output:3
cookie=0x0, duration=7506.320s, table=110, n_packets=1, n_bytes=42, tun_id=0x66,dl_dst=fa:16:3e:9a:7b:72 actions=output:1
cookie=0x0, duration=7493.238s, table=110, n_packets=2, n_bytes=196, tun_id=0x66,dl_dst=fa:16:3e:87:c1:43 actions=output:1
cookie=0x0, duration=7545.611s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:6c:1e:ae actions=output:1
cookie=0x0, duration=7558.699s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:35:3a:a9 actions=output:1
cookie=0x0, duration=7537.529s, table=110, n_packets=5, n_bytes=955, tun_id=0x65,dl_dst=fa:16:3e:6c:75:f1 actions=output:2
cookie=0x0, duration=7485.143s, table=110, n_packets=3, n_bytes=759, tun_id=0x66,dl_dst=fa:16:3e:d9:79:34 actions=output:4
```

```
vnx@comp3:~$ sudo ovs-ofctl dump-flows br-int -O Openflow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=7781.599s, table=0, n_packets=28, n_bytes=2960, in_port=3,dl_src=fa:16:3e:24:c6:a0
 actions=set_field:0x65->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7729.159s, table=0, n_packets=22, n_bytes=2414, in_port=4,dl_src=fa:16:3e:05:0a:6f
 actions=set_field:0x66->tun_id,load:0x1->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7833.704s, table=0, n_packets=44, n_bytes=3451, priority=0 actions=goto_table:20
cookie=0x0, duration=7781.096s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=3 actions=drop
cookie=0x0, duration=7728.654s, table=0, n_packets=0, n_bytes=0, priority=8192,in_port=4 actions=drop
cookie=0x0, duration=7776.553s, table=0, n_packets=16, n_bytes=1262, tun_id=0x65,in_port=2
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7725.608s, table=0, n_packets=7, n_bytes=1629, tun_id=0x66,in_port=1
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7724.086s, table=0, n_packets=2, n_bytes=674, tun_id=0x66,in_port=2
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7778.069s, table=0, n_packets=4, n_bytes=801, tun_id=0x65,in_port=1
 actions=load:0x2->NXM_NX_REG0[],goto_table:20
cookie=0x0, duration=7834.692s, table=0, n_packets=2173, n_bytes=241203, dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=7833.202s, table=20, n_packets=120, n_bytes=13065, priority=0 actions=goto_table:30
cookie=0x0, duration=7703.702s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.3
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:6c:1e:ae->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e6c1eae->NXM_NX_ARP_SHA[],
 load:0xa650003->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7709.790s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.5
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:24:c6:a0->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e24c6a0->NXM_NX_ARP_SHA[],
 load:0xa650005->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7706.742s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.4
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:6c:75:f1->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e6c75f1->NXM_NX_ARP_SHA[],
 load:0xa650004->NXM_OF_ARP_SPA[],IN_PORT
```

```
cookie=0x0, duration=7700.651s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.2
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:35:3a:a9->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e353aa9->NXM_NX_ARP_SHA[],
 load:0xa650002->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7680.772s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.5
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:05:0a:6f->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e050a6f->NXM_NX_ARP_SHA[],
 load:0xa660005->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7683.827s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.4
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:d9:79:34->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163ed97934->NXM_NX_ARP_SHA[],
 load:0xa660004->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7714.373s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x65,arp_tpa=10.101.0.1
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:19:89:21->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e198921->NXM_NX_ARP_SHA[],
 load:0xa650001->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7695.555s, table=20, n_packets=2, n_bytes=84, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.1
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:04:35:cb->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e0435cb->NXM_NX_ARP_SHA[],
 load:0xa660001->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7689.963s, table=20, n_packets=1, n_bytes=42, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.3
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:87:c1:43->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e87c143->NXM_NX_ARP_SHA[],
 load:0xa660003->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7686.881s, table=20, n_packets=0, n_bytes=0, priority=1024,arp,tun_id=0x66,arp_tpa=10.102.0.2
 actions=move:NXM_OF_ETH_SRC[]->NXM_OF_ETH_DST[],set_field:fa:16:3e:9a:7b:72->eth_src,load:0x2->NXM_OF_ARP_OP[],
 move:NXM_NX_ARP_SHA[]->NXM_NX_ARP_THA[],move:NXM_OF_ARP_SPA[]->NXM_OF_ARP_TPA[],load:0xfa163e9a7b72->NXM_NX_ARP_SHA[],
 load:0xa660002->NXM_OF_ARP_SPA[],IN_PORT
cookie=0x0, duration=7832.696s, table=30, n_packets=107, n_bytes=11270, priority=0 actions=goto_table:40
cookie=0x0, duration=7695.046s, table=30, n_packets=12, n_bytes=1697, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.0/24
 actions=goto_table:40
cookie=0x0, duration=7713.868s, table=30, n_packets=1, n_bytes=98, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.0/24
 actions=goto_table:40
cookie=0x0, duration=7832.194s, table=40, n_packets=120, n_bytes=13065, priority=0 actions=goto_table:50
```

```
cookie=0x0, duration=7831.691s, table=50, n_packets=120, n_bytes=13065, priority=0 actions=goto_table:60
cookie=0x0, duration=7831.185s, table=60, n_packets=120, n_bytes=13065, priority=0 actions=goto_table:70
cookie=0x0, duration=7696.567s, table=60, n_packets=0, n_bytes=0, priority=2048,ip,tun_id=0x65,nw_dst=10.102.0.0/24
 actions=set_field:fa:16:3e:04:35:cb->eth_src,dec_ttl,set_field:0x66->tun_id,goto_table:70
cookie=0x0, duration=7696.059s, table=60, n_packets=0, n_bytes=0, priority=2048,ip,tun_id=0x66,nw_dst=10.101.0.0/24
 actions=set_field:fa:16:3e:19:89:21->eth_src,dec_ttl,set_field:0x65->tun_id,goto_table:70
cookie=0x0, duration=7830.679s, table=70, n_packets=116, n_bytes=12673, priority=0 actions=goto_table:80
cookie=0x0, duration=7710.302s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.5
 actions=set_field:fa:16:3e:24:c6:a0->eth_dst,goto_table:80
cookie=0x0, duration=7704.206s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.3
 actions=set_field:fa:16:3e:6c:1e:ae->eth_dst,goto_table:80
cookie=0x0, duration=7687.407s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.2
 actions=set_field:fa:16:3e:9a:7b:72->eth_dst,goto_table:80
cookie=0x0, duration=7684.330s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.4
 actions=set_field:fa:16:3e:d9:79:34->eth_dst,goto_table:80
cookie=0x0, duration=7707.253s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.4
 actions=set_field:fa:16:3e:6c:75:f1->eth_dst,goto_table:80
cookie=0x0, duration=7701.162s, table=70, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.2
 actions=set_field:fa:16:3e:35:3a:a9->eth_dst,goto_table:80
cookie=0x0, duration=7690.471s, table=70, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.3
 actions=set_field:fa:16:3e:87:c1:43->eth_dst,goto_table:80
cookie=0x0, duration=7681.279s, table=70, n_packets=2, n_bytes=196, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.5
 actions=set_field:fa:16:3e:05:0a:6f->eth_dst,goto_table:80
cookie=0x0, duration=7830.176s, table=80, n_packets=120, n_bytes=13065, priority=0 actions=goto_table:90
cookie=0x0, duration=7829.673s, table=90, n_packets=120, n_bytes=13065, priority=0 actions=goto_table:100
cookie=0x0, duration=7829.170s, table=100, n_packets=108, n_bytes=11368, priority=0 actions=goto_table:110
cookie=0x0, duration=7694.539s, table=100, n_packets=12, n_bytes=1697, priority=1024,ip,tun_id=0x66,nw_dst=10.102.0.0/24
 actions=goto_table:110
cookie=0x0, duration=7713.364s, table=100, n_packets=0, n_bytes=0, priority=1024,ip,tun_id=0x65,nw_dst=10.101.0.0/24
 actions=goto_table:110
cookie=0x0, duration=7778.575s, table=110, n_packets=14, n_bytes=1277, priority=8192,tun_id=0x65 actions=drop
cookie=0x0, duration=7726.113s, table=110, n_packets=15, n_bytes=1418, priority=8192,tun_id=0x66 actions=drop
cookie=0x0, duration=7828.664s, table=110, n_packets=47, n_bytes=3978, priority=0 actions=drop
```

```
cookie=0x0, duration=7780.084s, table=110, n_packets=17, n_bytes=1304,
 priority=16384,reg0=0x2,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:3
cookie=0x0, duration=7727.640s, table=110, n_packets=5, n_bytes=1390,
 priority=16384,reg0=0x2,tun_id=0x66,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:4
cookie=0x0, duration=7727.131s, table=110, n_packets=2, n_bytes=674,
 priority=16384,reg0=0x1,tun_id=0x66,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:4,output:1,output:2
cookie=0x0, duration=7779.580s, table=110, n_packets=11, n_bytes=1156,
 priority=16384,reg0=0x1,tun_id=0x65,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:3,output:1,output:2
cookie=0x0, duration=7780.592s, table=110, n_packets=3, n_bytes=759, tun_id=0x65,dl_dst=fa:16:3e:24:c6:a0 actions=output:3
cookie=0x0, duration=7728.146s, table=110, n_packets=4, n_bytes=913, tun_id=0x66,dl_dst=fa:16:3e:05:0a:6f actions=output:4
cookie=0x0, duration=7763.981s, table=110, n_packets=0, n_bytes=0, tun_id=0x66,dl_dst=fa:16:3e:9a:7b:72 actions=output:1
cookie=0x0, duration=7750.898s, table=110, n_packets=2, n_bytes=196, tun_id=0x66,dl_dst=fa:16:3e:87:c1:43 actions=output:1
cookie=0x0, duration=7803.271s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:6c:1e:ae actions=output:1
cookie=0x0, duration=7816.362s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:35:3a:a9 actions=output:1
cookie=0x0, duration=7788.644s, table=110, n_packets=0, n_bytes=0, tun_id=0x65,dl_dst=fa:16:3e:6c:75:f1 actions=output:2
cookie=0x0, duration=7736.241s, table=110, n_packets=0, n_bytes=0, tun_id=0x66,dl_dst=fa:16:3e:d9:79:34 actions=output:2
```

Bibliografía

[1] Cisco Visual Networking Index: Forecast and Methodology (2014-2019), Cisco, 2015.

[2] SDN Architecture Overview. White paper, Open Networking Foundation, 2013.

[3] Software Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, 2012.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, Computer Communication Review (CCR), vol. 38, pp. 69-74, 2008.

[5] P. Göransson, C. Black, "The OpenFlow Specification" en Software Defined Networks - A comprehensive Approach, pp. 81-117, Elsevier, 2014.

[6] Open vSwitch, http://openvswitch.org/.

[7] IETF RFC 7047, Open vSwitch Database Management Protocol, 2013.

[8] Flowgrammable - OpenFlow Message Layer, http://flowgrammable.org/sdn/openflow/message-layer/.

[9] OpenFlow Switch Specification, Version 1.1.0, Open Networking Foundation, 2011.

[10] OpenFlow Switch Specification, Version 1.5.1, Open Networking Foundation, 2015.

[11] T.D. Nadeau, K. Gray, "SDN Controllers" en SDN: Software Defined Networks, pp. 71-115, O'Reilly, 2013.

[12] OpenDaylight Developer Wiki, https://wiki.opendaylight.org.

[13] OpenDaylight Developer Guide, http://go.linuxfoundation.org/l/6342/2015-06-28/2l76qr/6342/128124/bk_developers_guide_20150629.pdf.

[14] Apache Maven, https://maven.apache.org/.

[15] Eclipse Equinox, http://www.eclipse.org/equinox/.

[16] Apache Felix, http://felix.apache.org/.

[17] Apache Karaf, http://karaf.apache.org/.

[18] Open Platform for NFV (OPNFV), https://www.opnfv.org/.

[19] OpenStack - Open Source Cloud Computing Software, https://www.openstack.org/.

[20] OpenStack Documentation, http://docs.openstack.org/.

[21] FlavioBlog, http://www.flaviof.com/.

[22] Virtual Networks over linuX (VNX), http://vnx.dit.upm.es/.

[23] Mininet - An Instant Virtual Network on your Laptop, http://mininet.org/.

[24] DevStack - an OpenStack Community Production, http://docs.openstack.org/developer/devstack/.

[25] OpenDaylight Getting Started Guide, http://go.linuxfoundation.org/l/6342/2015-06-28/2l76qt/6342/128122/bk_getting_started_guide_20150629.pdf.

[26] Linux Containers, https://linuxcontainers.org/.

[27] Libvirt: The virtualization API, http://libvirt.org/.

[28] DHCP server: dnsmasq, http://www.thekelleys.org.uk/dnsmasq/doc.html.