

Proyecto de redes y servicios

Redes definidas por software (SDN)

Índice

Objetivos de aprendizaje	3
1. Introducción a las redes definidas por software	3
1.1 Necesidad	3
1.2 Funcionamiento	3
1.3 ONOS	4
1.4 OpenFlow	5
1.5 Conmutadores virtuales Open vSwitch	7
2. Montaje	7
Paso 1. Incorporación de los dispositivos Open vSwitch	8
Paso 2. Incorporación del dispositivo asociado al controlador ONOS	10
Paso 3. Incorporación del resto de dispositivos	10
Paso 4. Preparar accesos a la gestión del controlador ONOS	10
Paso 5. Configuración de los conmutadores OpenFlow	12
Paso 6. Comprobar la información desde el controlador	14
3. Mejoras a implementar	19
3.1 Segmentación de redes (VLAN)	19
3.2 Esquema alternativo de flujos	19
Referencias	19

Objetivos de aprendizaje

El objetivo de la presente práctica es familiarizarse con la configuración básica de las redes definidas por software (SDN). En concreto, se presentarán sus componentes básicos, describiremos su funcionamiento, y aprenderemos a realizar su manejo básico y emulación.

A la finalización de esta práctica seremos capaces de:

1. Comprender los conceptos básicos asociados a una red definida por software.
2. Instalar y configurar los elementos del plano de control (controlador).
3. Instalar y configurar los elementos del plano de datos (conmutadores).
4. Comprender el funcionamiento de los protocolos más importantes.

1. Introducción a las redes definidas por software

1.1 Necesidad

A pesar de su adopción generalizada, en las redes tradicionales los dispositivos de red integran al mismo tiempo el plano de control (que decide cómo manejar el tráfico de red) y el plano de datos (que reenvía el tráfico de red), lo que reduce la flexibilidad en su configuración, funcionamiento y gestión.

Las redes definidas por software (SDN) vienen a salvar varias de las limitaciones de las redes tradicionales, separando el plano de control del plano de datos, e implementando cada uno por separado. Esta desagregación permite, por ejemplo, que los conmutadores tradicionales se conviertan en simples dispositivos de reenvío y que la lógica de control se implemente en un controlador software centralizado. De esta manera se incrementa la flexibilidad de la red, dividiendo el problema de control de la red en partes tratables e introduciendo nuevas abstracciones que facilitan el desarrollo y evolución de nuevos servicios de red.

1.2 Funcionamiento

La arquitectura funcional de una red definida por software, mostrada en la figura 1, está compuesta por el plano de control y el plano de datos, así como una serie de elementos, interfaces y protocolos incluidos en los mismos.

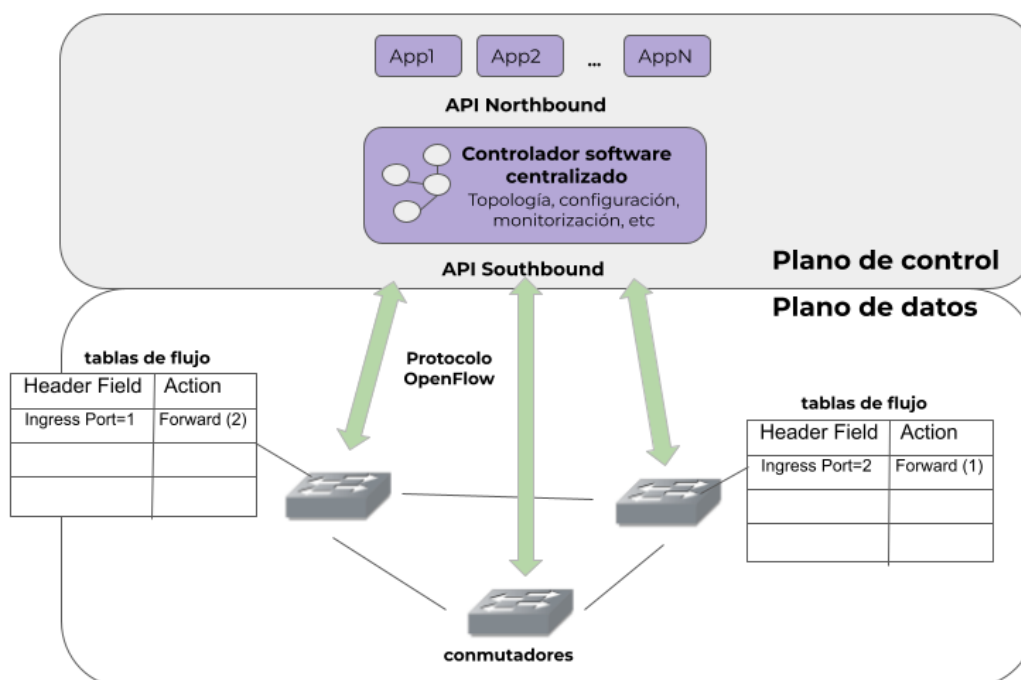


Figura 1. Arquitectura funcional de una red definida por software

El plano de datos es el responsable de tratar directamente la mayor parte de los paquetes que transitan por los conmutadores (dispositivos del plano de datos). Los conmutadores gestionan la recepción, modificación y/o transmisión de paquetes en función de sus tablas de flujos, que se encargan de establecer la lógica asociada en cada caso. Los paquetes que no pueden ser procesados directamente por el plano de datos, por ejemplo, si su información aún no está insertada en la tabla de flujos del conmutador, se reenvían al plano de control para que determine el tratamiento a realizar. De esta manera, la inteligencia de la red se traslada al controlador y los conmutadores únicamente deberán realizar el reenvío de paquetes.

El plano de control se encarga principalmente de mantener las tablas de flujo de los conmutadores. Es responsable de procesar diferentes protocolos de control que afectan a las tablas de flujo en los conmutadores, así como mantener de forma centralizada la información de configuración y monitorización de la red. Para ampliar y/o hacer uso de forma programática de las funciones de la red es habitual que sobre los controladores sea posible la incorporación de aplicaciones específicas.

1.3 ONOS

Un controlador tiene como elementos principales los que veíamos en la figura [1](#):

- Una API hacia el norte (Northbound) para la comunicación con las aplicaciones que se despliegan sobre el controlador con el objetivo de ampliar su funcionalidad (personalizadas o predefinidas).
- Un conjunto de aplicaciones predefinidas y comunes que se instalan habitualmente junto con el controlador, y entre las que se encuentran las asociadas a su interfaz gráfica, enrutamiento básico, conmutación a nivel 2, etc.
- Una API hacia el sur (Southbound) que se utiliza como interfaz de comunicación con los conmutadores que gestiona. Con frecuencia esta API es OpenFlow.
- Un conjunto de funcionalidades core:
 - Detección de dispositivos de usuario y de red.
 - Gestión de la topología de la red: mantener información sobre los detalles de interconexión de dispositivos de red.
 - Gestión de flujos: mantiene una base de datos de los flujos que gestiona el controlador y realizar toda la coordinación necesaria con los dispositivos.
 - Monitorización de estado y tráfico de la red.

El controlador es un servidor al uso que se conecta a cada uno de los conmutadores (con soporte OpenFlow) a través de una red de control (normalmente la gestión se realiza fuera de banda, es decir en una red separada). Dicha red de control permite el envío y la recepción de mensajes OpenFlow que permiten manipular paquetes y añadir entradas a la tabla de flujo. Cuando no se encuentra una coincidencia para un paquete en la tabla de flujo, éste se envía al controlador, para que tome una decisión sobre ese paquete mediante un programa software, que puede añadir entradas a la tabla de flujo, con el fin de que paquetes que coincidan con el mismo patrón no necesiten pasar de nuevo por el controlador (temporal o definitivamente).

El controlador ONOS (Open Network Operating System) es un controlador SDN Open Source gestionado por la Open Networking Foundation (ONF) [\[1\]](#). Es un controlador escalable, de alto rendimiento, con soporte para alta disponibilidad y compatible tanto con dispositivos tradicionales como dispositivos OpenFlow. Está escrito en Java y construido sobre Apache Karaf. Su arquitectura general es la que se muestra en la siguiente figura [2](#).

Conviene resaltar la facilidad que conlleva poner en marcha una red SDN simple con ONOS. Únicamente es necesario activar dos aplicaciones, `org.onosproject.openflow` para poder comunicar los switches OpenFlow con el controlador, y `org.onosproject.fwd` para habilitar el forwarding reactivo. El forwarding reactivo consiste en crear flujos dinámicamente cuando el controlador no tiene ningún flujo específico para manejar ese tráfico. ONOS puede ser administrado mediante su API REST, línea de comandos (CLI) y la interfaz gráfica web.

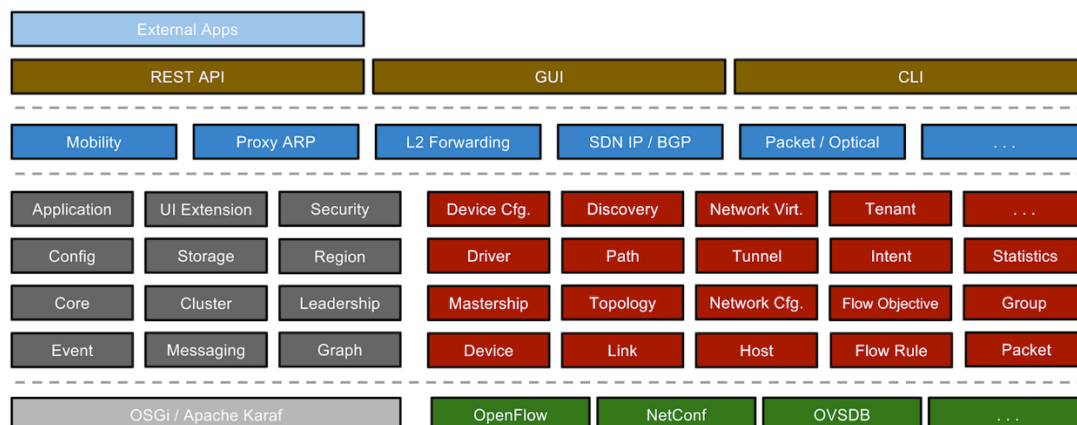


Figura 2. Arquitectura de ONOS (figura tomada de [8])

1.4 OpenFlow

El objetivo principal del protocolo OpenFlow es proporcionar una interfaz abierta para programar la tabla de flujo en los conmutadores de una red definida por software. En una red basada en OpenFlow, cuyo esquema general es el que se muestra en la figura 3, siempre hay un controlador que se comunica con uno o más conmutadores a través de este protocolo. El protocolo define los mensajes específicos, y su formato, intercambiados entre el controlador (plano de control) y el conmutador (plano de datos), así como la descripción de cómo este último debe reaccionar en diversas situaciones y cómo debe responder a los comandos del controlador.

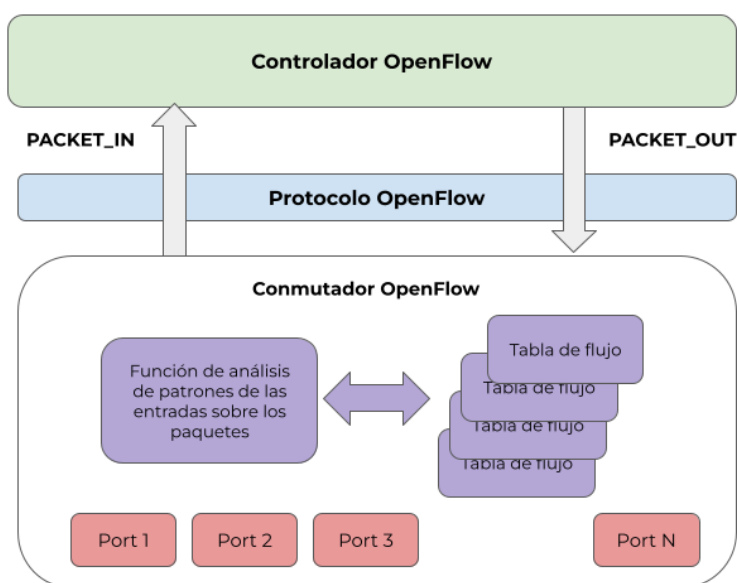


Figura 3. Funcionamiento de una red definida por software con OpenFlow

En un conmutador OpenFlow la función central es tomar los paquetes que llegan a un puerto y tratarlos según la información disponible en sus tablas de flujo. El pipeline de un switch OpenFlow contiene múltiples tablas de flujo, cada una de las cuales puede contener múltiples entradas. El pipeline define cómo interactúan los paquetes con las tablas de flujo. Un conmutador OpenFlow debe tener al menos una tabla de flujo, y opcionalmente puede tener más de una, en cuyo caso el procesamiento se simplifica. El procesamiento siempre comienza en la primera tabla de flujo y se utilizarán las tablas sucesivas dependiendo de la salida de la primera tabla. El proceso se muestra en la figura 4.

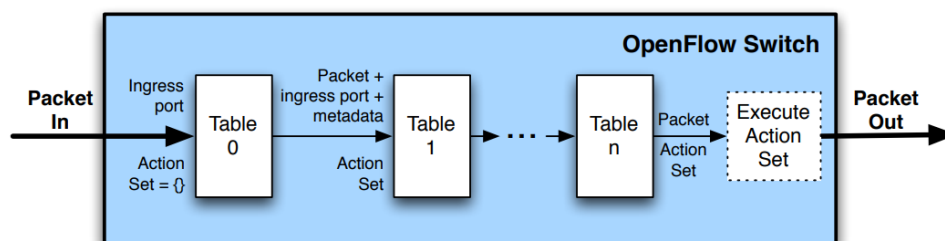


Figura 4. Flujo de paquetes a través del pipeline OpenFlow (figura tomada de [2])

Cuando un paquete llega al conmutador, tal y como se muestra en la siguiente figura 5, el proceso comienza haciendo una búsqueda en la primera tabla de flujo (tabla 0). El paquete se compara con el patrón de cada una de las entradas de la tabla de flujo para seleccionar una entrada. Si en este proceso se encuentra una entrada compatible se guarda el conjunto de instrucciones asociado a esa entrada. Estas instrucciones pueden redirigir el paquete hacia otra tabla de flujo posterior donde se repite el mismo proceso. En la última tabla del pipeline, obviamente, no será posible esta redirección. Cuando el conjunto de instrucciones no contiene una instrucción de redirección, el procesamiento se detiene en la tabla correspondiente y se ejecutan las instrucciones (actions) asociados.

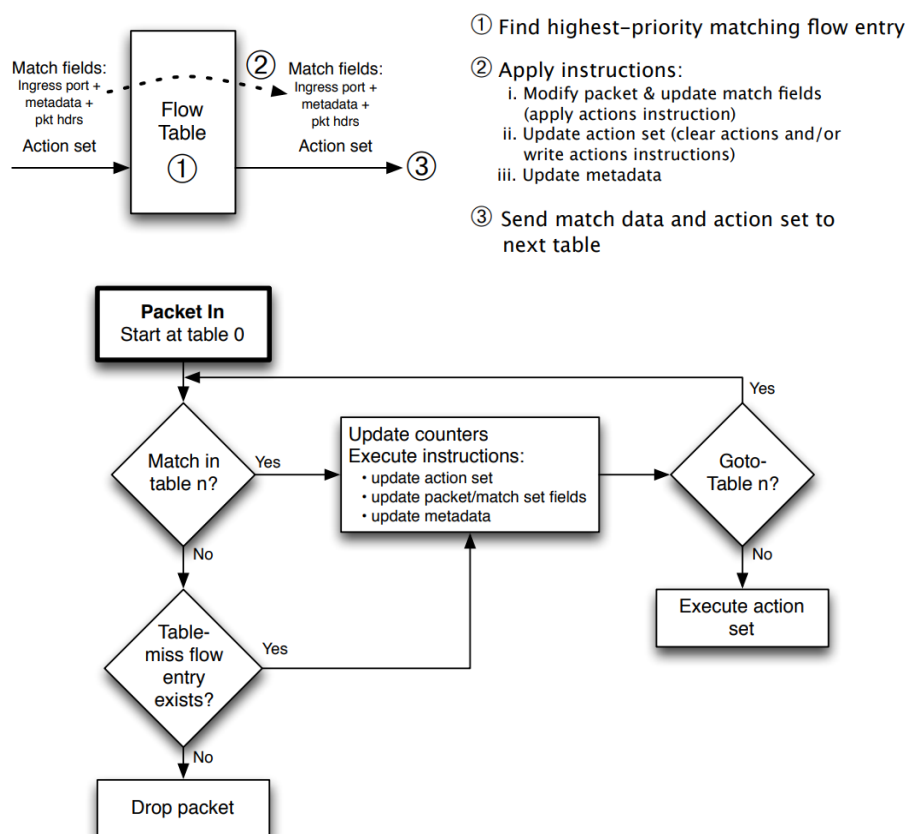


Figura 5. Algoritmo de procesamiento en un pipeline OpenFlow (figura tomada de [2])

Cuando en el plano de datos no concuerda con el paquete entrante, se envía un mensaje OpenFlow PACKET_IN al controlador. El plano de control ejecuta la lógica para determinar cuáles deben ser las tablas de reenvío y la lógica en el plano de datos. En consecuencia, cuando el controlador tiene un paquete de datos para reenviar a través del conmutador, utiliza el mensaje OpenFlow PACKET_OUT.

1.5 Conmutadores virtuales Open vSwitch

Open vSwitch [3] es un software de código abierto diseñado para utilizarse como un conmutador virtual controlable a través de OpenFlow, y pensado para ser utilizado en entornos de virtualización de infraestructuras, de manera que se usa de forma habitual para reenviar el tráfico entre diferentes máquinas virtuales (VMs), en un mismo host físico, y también para comunicar estas VMs con la red física (se emplea en las soluciones de virtualización de servidores habituales en Linux).

2. Montaje

La arquitectura de red que vamos a desplegar y emular (dentro de la solución GNS3) es la que se muestra en la siguiente figura (la primera parte de la arquitectura a emular dentro del proyecto):

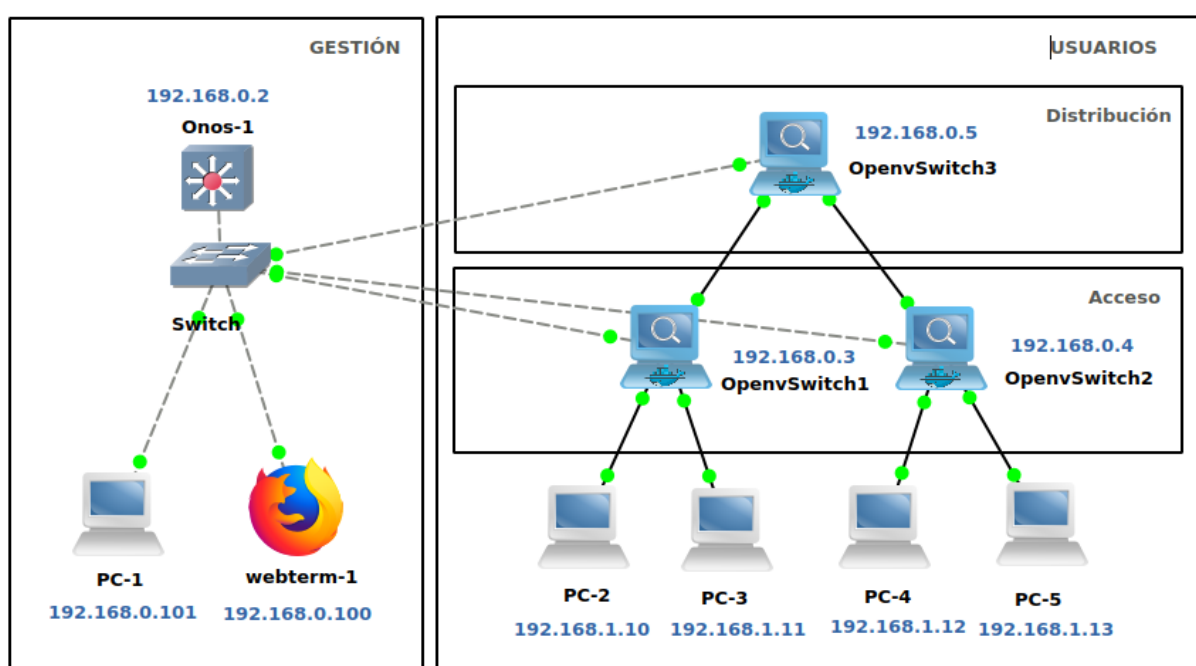


Figura 6. Montaje que se pretende emular en GNS3.

Tanto Open vSwitch como el controlador de ONOS los vamos a implementar a través de contenerización, empleando la solución Docker.

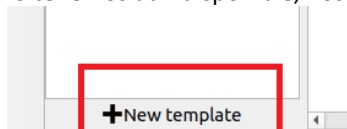
El direccionamiento IP que emplearemos es el que se indica en la siguiente tabla:

Red	Equipo	Dirección IP	Descripción
Gestión	OpenvSwitch1	192.168.0.3	Conmutador virtual Open vSwitch de Acceso
	OpenvSwitch2	192.168.0.4	Conmutador virtual Open vSwitch de Acceso
	OpenvSwitch3	192.168.0.5	Conmutador virtual Open vSwitch de Distribución
	Onos-1	192.168.0.2	Controlador ONOS
	webterm-1	192.168.0.100	Equipo para acceso con navegador para gestión
	PC-1	192.168.0.101	Equipo para acceso por SSH para gestión
Usuarios	PC-2	192.168.1.10	Equipo de usuario en OpenvSwitch1
	PC-3	192.168.1.11	Equipo de usuario en OpenvSwitch1
	PC-4	192.168.1.12	Equipo de usuario en OpenvSwitch2
	PC-5	192.168.1.13	Equipo de usuario en OpenvSwitch3

Tabla 1. Configuración de red a aplicar.

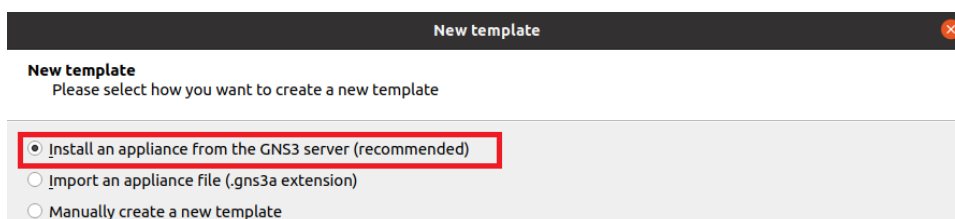
Paso 1. Incorporación de los dispositivos Open vSwitch

Vamos a añadir este tipo de dispositivo a partir del appliance disponible en el servidor de GNS3, para ello, si no lo tenemos aún disponible, nos vamos a nueva plantilla de dispositivo:

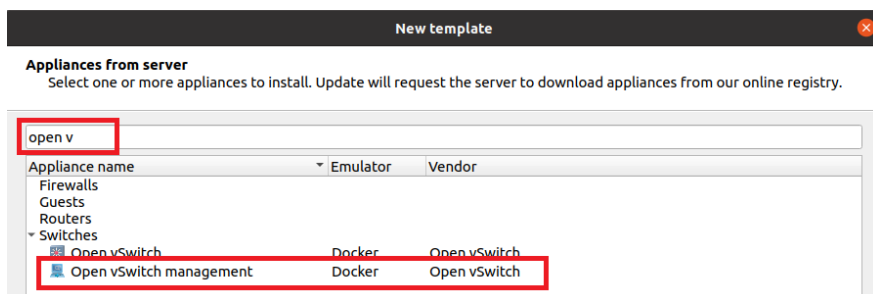


Nota para instalar en casa: estas plantillas van a desplegar los dispositivos en nuestro equipo haciendo uso de Docker, si no lo tenemos instalado podemos hacerlo siguiendo algunos de los tutoriales [9] disponibles en Internet.

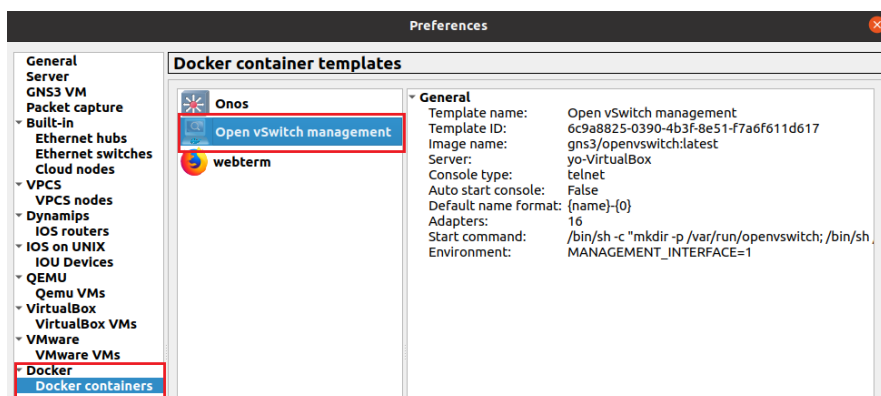
A continuación, indicamos a que deseamos realizar la instalación a partir de un appliance disponible en el servidor de GNS3:

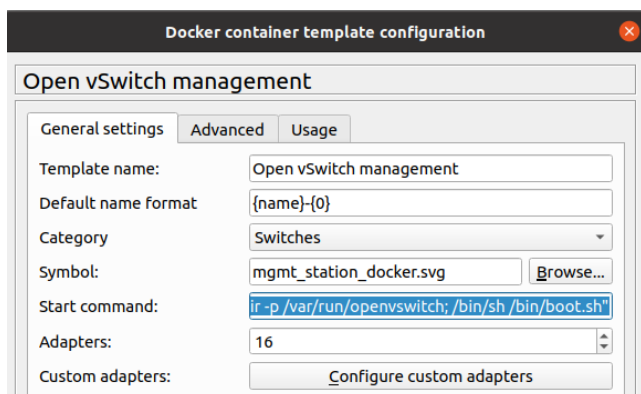


En la siguiente pantalla hacemos la búsqueda poniendo "open v" en el campo del buscador, seleccionamos la plantilla denominada "Open vSwitch management" (ya viene parametrizada para que la interfaz eth0 sea la de management, es decir la que conectaremos con la red de gestión para establecer la comunicación con el controlador) y le damos a instalar:

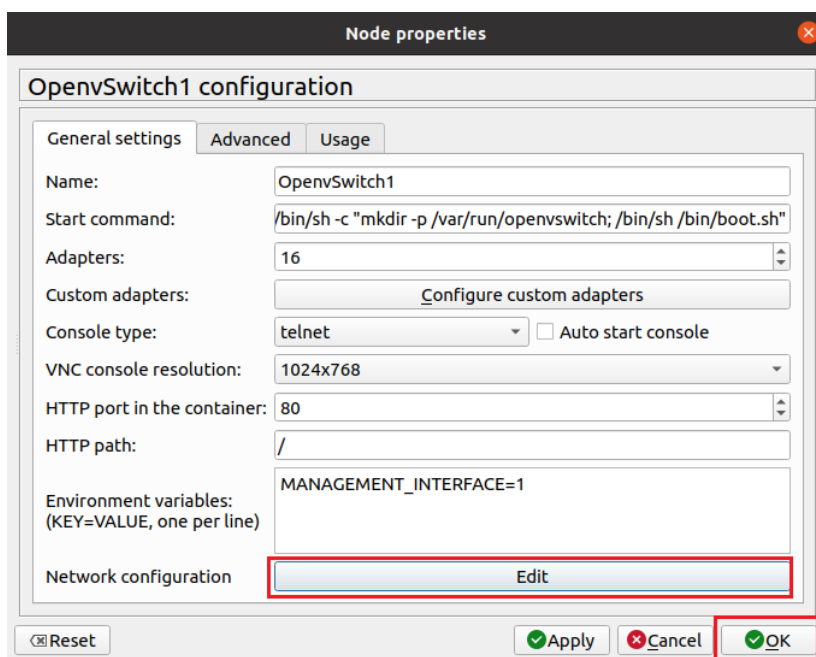


Una vez instalada la plantilla debemos hacer un ajuste para salvar un bug [7] en la imagen docker que GNS3 despliega para OpenvSwitch (añadir en Start Command: `/bin/sh -c "mkdir -p /var/run/openvswitch; /bin/sh /bin/boot.sh"`).

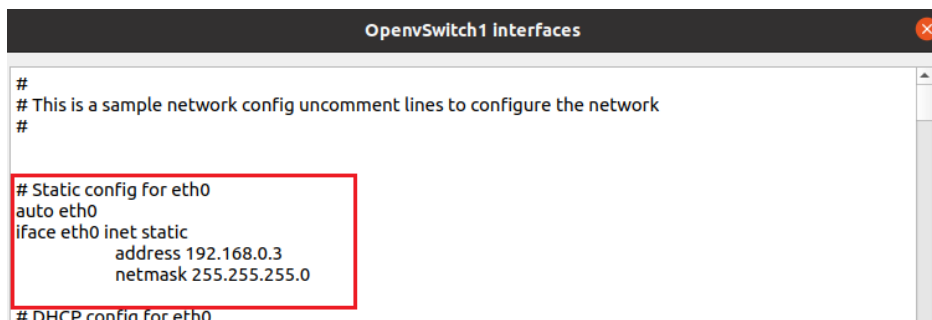




A partir de este momento ya podemos incorporar el dispositivo a nuestro proyecto. En concreto vamos a incorporar 3 dispositivos que llamaremos: OpenvSwitch1, OpenvSwitch2 y OpenvSwitch3, tal y como se mostraba en la figura 6 y a continuación configuramos sus direcciones IP. Para ello, nos vamos a las propiedades de cada dispositivo, y seleccionamos la opción Edit para modificar la configuración de red:

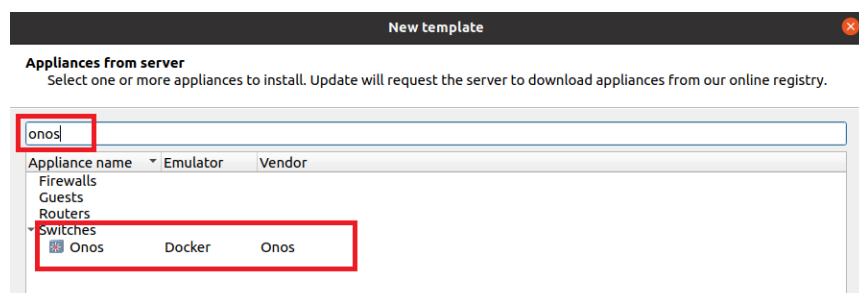


A continuación, especificamos para cada nodo, de forma estática, el direccionamiento que va a emplear en la red de gestión:



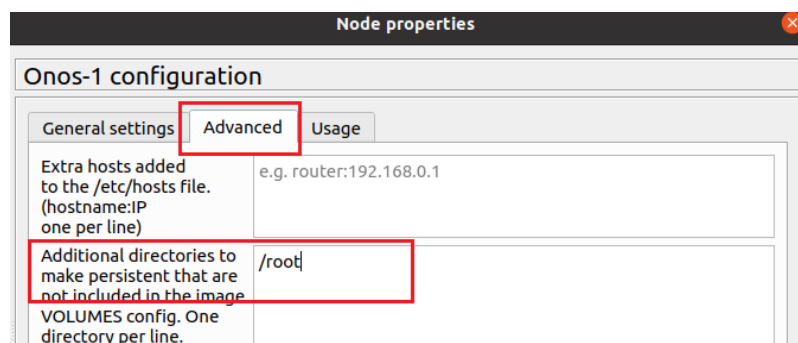
Paso 2. Incorporación del dispositivo asociado al controlador ONOS

De nuevo, vamos a añadir este tipo de dispositivo a partir del appliance disponible en el servidor de GNS3, para ello, si no lo tenemos aún disponible, vamos a proceder de la misma forma que para los conmutadores Open vSwitch. Vamos a añadir una plantilla de dispositivo, y en la pantalla de búsqueda de appliances en el servidor de GNS3, ponemos la palabra “onos”, seleccionamos la plantilla denominada “Onos” y le damos a Instalar:



A partir de este momento ya podemos incorporar el dispositivo a nuestro proyecto, así que incorporamos el dispositivo Onos-1, tal y como se mostraba en la figura 6 y a continuación configuramos su dirección IP. Para ello, nos vamos a las propiedades del dispositivo, y procedemos de la mismo que se indica en el apartado anterior.

Para que los cambios en el controlador sean persistentes, en las propiedades del dispositivo, incluimos la siguiente configuración para hacer persistente la carpeta /root:



Paso 3. Incorporación del resto de dispositivos

A continuación, incorporamos y configuramos el resto de los dispositivos (Switch, PCs y webterm) que se muestran en la figura 6 aplicando la configuración indicada en la tabla 1. Si no tienes instalada la plantilla de alguno de estos dispositivos, investiga la forma de incorporarla y de añadir el dispositivo al proyecto. Una vez que hayas añadido todos los dispositivos al proyecto, realiza las conexiones que se muestran en la figura 6, y procede al arranque de estos.

Nota para instalar en casa: antes de poder usar el dispositivo webterm (despliega en Docker una imagen Linux ligera con entorno gráfico) debemos tener instalado en el equipo un cliente VNC con soporte X11, en Ubuntu 20.04 lo podemos conseguir con el comando:


```
$ sudo apt-get install xtightvncviewer
```

Paso 4. Preparar accesos a la gestión del controlador ONOS

Podremos acceder al controlador a través de una interfaz de línea de comandos (CLI) o a través de una interfaz gráfica (GUI) desde un navegador.

Para el acceso a la CLI, vamos a ir a la consola de PC-1, que está en la red de gestión, y desde ahí vamos a realizar una conexión SSH al puerto 8101 del controlador ONOS (usuario: karaf y contraseña: karaf).

```
# ssh -v -p 8101 karaf@192.168.0.2
```



```
Documentation: wiki.onosproject.org
Tutorials:    tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

karaf@root >
```

Una vez conectados, ya nos encontramos en el CLI de ONOS y podemos lanzar directamente los comandos que tiene disponible el controlador [4]. Por ejemplo, antes de configurar y conectar los conmutadores, podemos lanzar los comandos de la consola de ONOS para consultar los distintos elementos de configuración del controlador (flujos configurados en los conmutadores, dispositivos o conmutadores conectados, enlaces y hosts detectados a partir del tráfico) y comprobaremos que como respuesta no vemos nada aún.



```
karaf@root >
karaf@root > flows
karaf@root > devices
karaf@root > links
karaf@root > hosts
karaf@root >
karaf@root >
```

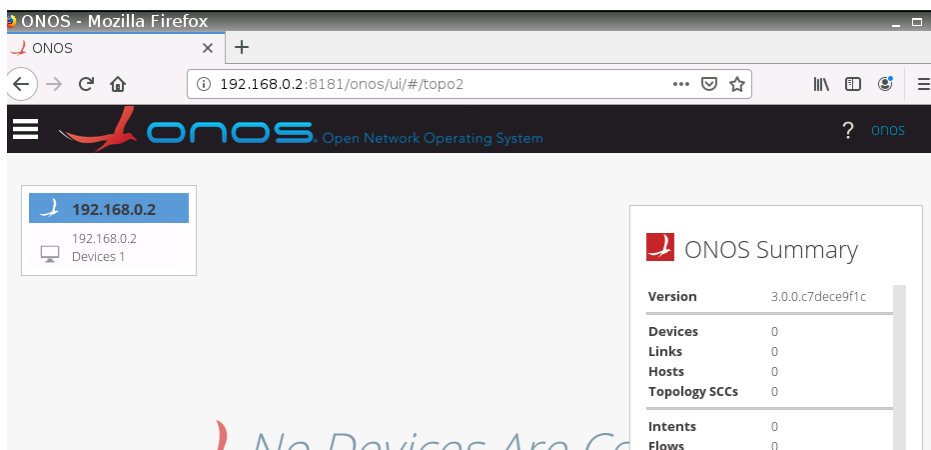
También podremos comprobar por ejemplo las aplicaciones que están activas en el controlador

```
# apps -a -s
```

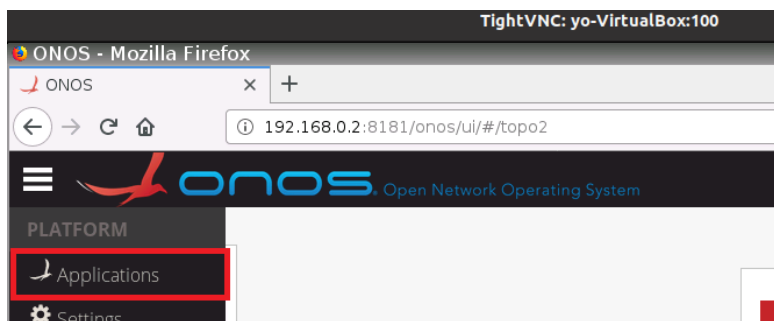
Si quisiéramos activar una aplicación, podríamos hacerlo de la forma que se indica a continuación, en este caso vamos a aprovechar para activar el protocolo OpenFlow en el controlador ONOS (por defecto no está habilitado):

```
# app activate org.onosproject.openflow
```

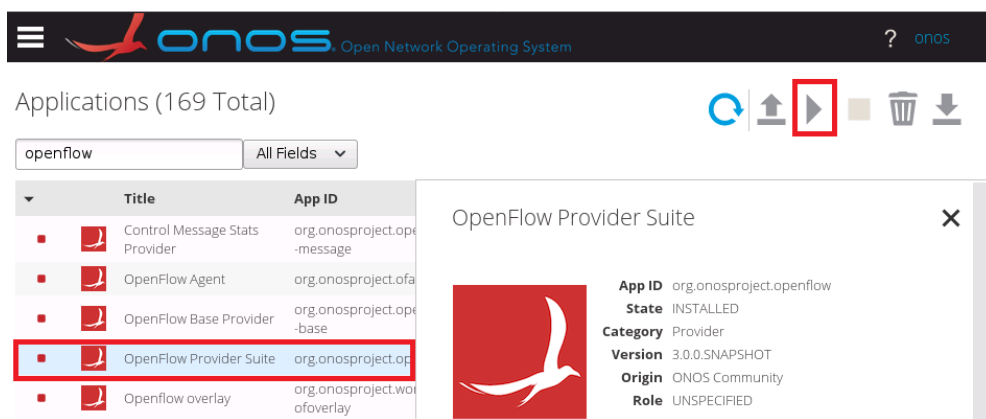
Para el acceso a la GUI, emplearemos el webterm-1, que está en la red de gestión también, y desde su consola, que abrirá una conexión VNC a un entorno gráfico desde el que podemos usar un navegador, accederemos a la URL por defecto del controlador <http://192.168.0.2:8181/onos/ui/index.html> (usuario: onos y contraseña: rocks). Una vez autenticados, veremos la vista de topología en la que solamente veremos de momento al propio controlador.



Si quisiéramos activar el protocolo OpenFlow en el controlador desde su GUI, nos tenemos que ir al menú, y luego a la opción “Applications”:



A continuación, buscaremos por “OpenFlow” para encontrar las App predefinidas, seleccionamos e instalamos la App “OpenFlow Provider Suite” (para instalar al seleccionar se pulsa en el botón de ejecutar de arriba a la derecha).



Podemos investigar más sobre el uso de la consola siguiendo el tutorial básico existente en la wiki del proyecto ONOS [6].

Paso 5. Configuración de los conmutadores OpenFlow

Una vez incorporados y arrancados los dispositivos Open vSwitch en el proyecto, nos conectamos a la consola de cada conmutador, y lo primero que vamos a hacer es ver la configuración existente (lo normal es que veamos varios bridges ya que es la configuración por defecto del appliance de GNS3), ejecutando el siguiente comando:

```
# ovs-vsctl show
```

En la salida de este comando, veremos que por defecto se han creado varios bridges en el conmutador (br0, br1, br2 y br3):

```
#
# ovs-vsctl show
ce68e85-d597-4725-8f9b-b850d9c32a10
  Bridge "br3"
    datapath_type: netdev
    Port "br3"
      Interface "br3"
        type: internal
  Bridge "br0"
    datapath_type: netdev
    Port "br0"
      Interface "br0"
        type: internal
    Port "eth2"
      Interface "eth2"
    Port "eth4"
```

Antes de continuar, vamos a realizar un borrador de estos bridges, con el objetivo de partir de una situación estable, para ello lanzamos el siguiente los siguientes comandos (uno para cada bridge):

```
# ovs-vsctl del-br br0
```

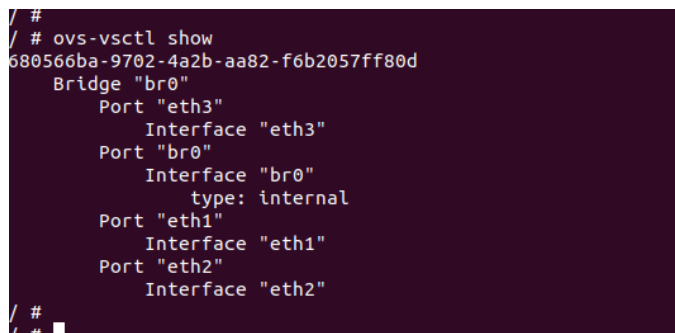
```
# ovs-vsctl del-br br1
# ovs-vsctl del-br br2
# ovs-vsctl del-br br3
```

A continuación, vamos a empezar a aplicar la configuración. Lo primero que tenemos que crear es un bridge, que consiste en una agrupación de puertos, para ello empleamos el siguiente comando (“br0” es el nombre que le vamos a poner a nuestro primer bridge en este conmutador virtual):

```
# ovs-vsctl add-br br0
```

Ahora añadimos los puertos que vamos a usar en el bridge creado (revisar los que podemos necesitar en cada dispositivo) y consultamos luego la configuración:

```
# ovs-vsctl add-port br0 eth1
# ovs-vsctl add-port br0 eth2
# ovs-vsctl add-port br0 eth3
# ovs-vsctl show
```



```
/ #
/ # ovs-vsctl show
680566ba-9702-4a2b-aa82-f6b2057ff80d
    Bridge "br0"
        Port "eth3"
            Interface "eth3"
        Port "br0"
            Interface "br0"
            type: internal
        Port "eth1"
            Interface "eth1"
        Port "eth2"
            Interface "eth2"
/ #
```

El siguiente paso es configurar cada uno de los bridges que acabamos de crear:

- Asociarles un identificador (other-config:datapath-id).
- Establecer que la comunicación con el controlador sea por defecto OpenFlow 1.3 (necesario para poder establecer la comunicación con el controlador ONOS en la versión que estamos usando).
- Establecer la dirección IP del controlador y la forma de conexión (connection-mode=out-of-band, es decir, empleando una red diferente).

En OpenvSwitch1, ejecutamos:

```
# ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000001 \
-- set bridge br0 protocols=OpenFlow13 \
-- set bridge br0 fail_mode=secure \
-- set-controller br0 tcp:192.168.0.2:6633 \
-- set controller br0 connection-mode=out-of-band
```

Una vez ejecutado, comprobamos en varias ocasiones, hasta confirmar que se ha establecido correctamente la conexión con el controlador (el atributo is_connected debe ser true):

```
#
# ovs-vsctl show
680566ba-9702-4a2b-aa82-f6b2057ff80d
Bridge "br0"
    Controller "tcp:192.168.0.2:6633"
    is_connected: true
    fail_mode: secure
    Port "eth3"
        Interface "eth3"
    Port "br0"
        Interface "br0"
        type: internal
    Port "eth1"
        Interface "eth1"
    Port "eth2"
        Interface "eth2"
```

A continuación, seguimos configurando los otros dos conmutadores:

En OpenvSwitch2 ejecutamos:

```
# ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000002 \
-- set bridge br0 protocols=OpenFlow13 \
-- set bridge br0 fail_mode=secure \
-- set-controller br0 tcp:192.168.0.2:6633 \
-- set controller br0 connection-mode=out-of-band
```

En OpenvSwitch3 ejecutamos:

```
# ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000003 \
-- set bridge br0 protocols=OpenFlow13 \
-- set bridge br0 fail_mode=secure \
-- set-controller br0 tcp:192.168.0.2:6633 \
-- set controller br0 connection-mode=out-of-band
```

Si queremos en detalle la configuración y el estado de la conexión con el controlador en cada conmutador podemos ejecutar:

```
# ovs-vsctl list controller
```

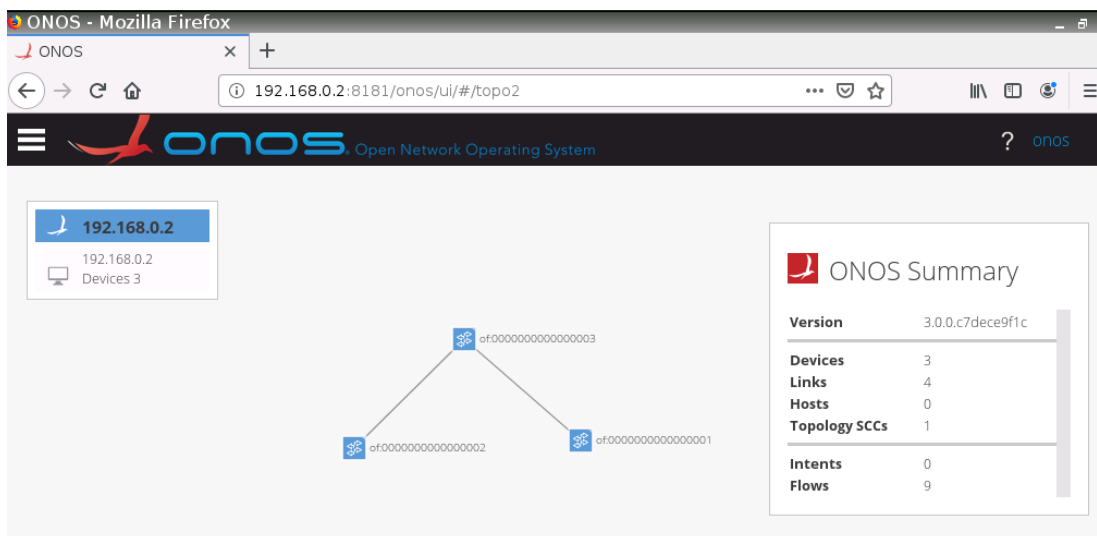
Paso 6. Comprobar la información desde el controlador

Ahora comprobamos la información descubierta en el controlados ONOS, primero lo vamos a hacer por línea de comandos (CLI) y luego por interfaz gráfica (GUI).

Desde la CLI, usamos el comando `devices`, y verificamos que se encuentran registrados los 3 conmutadores.

```
karaf@root > 20:03:21
karaf@root > devices 20:03:21
id=of:0000000000000001, available=true, local-status=connected 6m15s ago, role=MASTER, ty3
id=of:0000000000000002, available=true, local-status=connected 57s ago, role=MASTER, type3
id=of:0000000000000003, available=true, local-status=connected 4s ago, role=MASTER, type=3
karaf@root > 20:03:24
karaf@root > 20:03:25
```

Desde la GUI, vamos al menú de arriba a la izquierda, y luego navegamos a la opción "Topology":



A continuación, vamos a consultar el contenido de las tablas de flujos dentro de cada conmutador (una vez que los mismos se han conectado al controlador ONOS), para ello es importante que a la hora de ejecutar el comando `ovs-ofctl` (hace uso de OpenFlow) en la shell de cada conmutador, forcemos el uso de la versión Openflow 1.3 (por ejemplo, para OpenvSwitch1 ejecutaríamos).

```
# ovs-ofctl dump-flows br0 --protocols=OpenFlow13
```

```
#
# ovs-ofctl dump-flows br0 --protocols=OpenFlow13
cookie=0x100009465555a, duration=999.556s, table=0, n_packets=204, n_bytes=2774
, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
cookie=0x100007a585b6f, duration=999.556s, table=0, n_packets=212, n_bytes=2883
, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
cookie=0x10000ea6f4b8e, duration=999.556s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
#
#
```

En cada una de las entradas de la tabla de flujos OpenFlow que vemos, tenemos los siguientes campos:

- **cookie:** número identificativo (si no se indica al añadir el flujo se pone a 0).
- **duration:** tiempo en segundos que lleva la entrada de flujo activa en esta tabla.
- **priority:** prioridad de la entrada dentro de la tabla.
- **table:** número de tabla.
- **n_packets:** número de paquetes que concordaron con esta entrada de flujo.
- **n_bytes:** número de bytes que coincidieron con esta entrada de flujo.
- **idle_age:** número de segundos que han pasado desde la última concordancia con esta entrada.
- **actions:** acciones a realizar sobre los paquetes que concordaron con esta entrada de flujo.

Si ahora, vamos al CLI de ONOS, deberíamos ver los mismos flujos en el controlador para cada uno de los conmutadores (por ejemplo, para OpenvSwitch1 que tenía el Id acabado en 1 veríamos lo siguiente):

```
karaf@root > 20:37:29
karaf@root > flows added of:0000000000000001 20:37:29
deviceId=of:0000000000000001, flowRuleCount=3
  id=100007a585b6f, state=ADDED, bytes=98600, packets=725, duration=2590, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=100009465555a, state=ADDED, bytes=97512, packets=717, duration=2590, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000ea6f4b8e, state=ADDED, bytes=0, packets=0, duration=2590, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
karaf@root > 20:40:20
karaf@root > 20:40:22
karaf@root > 20:40:22
```

Como podemos ver en ambos casos, el controlador ha instalado por defecto tres (3) flujos por defecto, que se encargan de propagar los mensajes de los siguientes protocolos al controlador:

- Flujo 1 (ETH_TYPE=lldp): propaga los mensajes del protocolo Link Layer Discovery Protocol (LLDP) al controlador ([OUTPUT:CONTROLLER]).
- Flow 2 (ETH_TYPE=bddp): propaga los mensajes del protocolo Broadcast Domain Discovery Protocol (BDDP) al controlador ([OUTPUT:CONTROLLER]).
- Flow 3 (ETH_TYPE=arp): propaga los mensajes del protocolo Address Resolution Protocol (ARP) al controlador ([OUTPUT:CONTROLLER]).

Los flujos anteriores se utilizan por el controlador para el descubrimiento de links y hosts que dependen de sus controladores. Cada entrada de flujo está etiquetada por un appId (id. de aplicación) que identifica qué aplicación la instaló.

Si desde PC2 (192.168.1.10), intentamos hacer ping a PC3 (192.168.1.11), vemos que de momento no llegamos:

```
root@Debian:~#
root@Debian:~#
root@Debian:~# ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
From 192.168.1.10 icmp_seq=1 Destination Host Unreachable
From 192.168.1.10 icmp_seq=2 Destination Host Unreachable
From 192.168.1.10 icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.1.11 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5043ms
pipe 3
root@Debian:~#
```

Esto se debe a que no tenemos los flujos que nos permitan hacer el ping entre los equipos (intercambio de los mensajes ICMP de Echo request y Echo reply). La forma más sencilla de permitir el tráfico ICMP es activando en ONOS la app de Reactive Forwarding (reenvío reactivo). Esta aplicación instala flujos en los conmutados en respuesta a cada paquete IP que no puede ser tratado por un conmutador para que se remita al controlador. La forma más sencilla de activar esta aplicación es a través de la CLI del controlador:

```
karaf@root >
karaf@root > app activate org.onosproject.fwd
Activated org.onosproject.fwd
karaf@root >
```

Si ahora comprobamos, una vez activada la aplicación, y también desde la CLI de ONOS, las entradas en la tabla de flujos del conmutador Open vSwitch1, vemos que se ha añadido una nueva entrada:


```

karaf@root>
karaf@root> flows added of:0000000000000001
deviceId=of:0000000000000001, flowRuleCount=4
  id=100007a585b6f, state=ADDED, bytes=128656, packets=946, duration=3275, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=100009465555a, state=ADDED, bytes=127568, packets=938, duration=3275, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000ea6f4b8e, state=ADDED, bytes=600, packets=10, duration=3275, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000021b41dc, state=ADDED, bytes=196, packets=2, duration=139, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
karaf@root>

```

Este nuevo flujo tiene el selector=ETH_TYPE:ipv4, por tanto se ocupa de los paquete IPv4, y se encarga de reenviarlos al controlador (OUTPUT:CONTROLLER) para que este a su vez añada temporalmente (para mantener la eficiencia de los conmutadores) en las tablas de flujo correspondientes las entradas correspondientes para gestionar temporalmente el tráfico requerido.

Si ahora hacemos ping de nuevo desde PC-2 a PC-3, vemos que ya podemos llegar sin problemas:

```

root@Debian:~#
root@Debian:~# ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=35.7 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=5.46 ms
64 bytes from 192.168.1.11: icmp_seq=3 ttl=64 time=3.42 ms
64 bytes from 192.168.1.11: icmp_seq=4 ttl=64 time=2.00 ms
^C
-- 192.168.1.11 ping statistics --
4 packets transmitted, 4 received, 0% packet loss, time 3012ms
rtt min/avg/max/mdev = 2.008/11.669/35.782/13.975 ms
root@Debian:~#

```

Y además, si dejamos el ping de forma permanente, observamos que se instalan dos nuevos flujos en OpenvSwitch1 para la gestión del tráfico asociado al ping.

```

karaf@root> flows added of:0000000000000001
deviceId=of:0000000000000001, flowRuleCount=6
  id=100007a585b6f, state=ADDED, bytes=268056, packets=1971, duration=6450, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=100009465555a, state=ADDED, bytes=266968, packets=1963, duration=6450, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=10000ea6f4b8e, state=ADDED, bytes=1548, packets=27, duration=6450, liveType=UNKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
  id=45000032991631, state=ADDED, bytes=196, packets=2, duration=3, liveType=UNKNOWN, priority=0, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:2, ETH_DST:0C:61:C3:09:00:00, ETH_SRC:0C:BB:C9:EA:00:00], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], deferred=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
  id=450000c1657f41, state=ADDED, bytes=196, packets=2, duration=3, liveType=UNKNOWN, priority=0, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:1, ETH_DST:0C:BB:C9:EA:00:00, ETH_SRC:0C:61:C3:09:00:00], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:2], deferred=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
  id=10000021b41dc, state=ADDED, bytes=1176, packets=12, duration=3314, liveType=UNKNOWN, priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=null, metadata=null}

```

El primer flujo indica al conmutador que reenvíe los paquetes entrantes en el puerto 2 (IN_PORT:2) al puerto 1 (OUTPUT:1). De manera similar, el segundo flujo indica al conmutador que reenvíe los paquetes del puerto 1 al

puerto 2. Los flujos insertados permiten que el conmutador intercambie paquetes entre los hosts PC2 y PC3 sin retransmitirlos al controlador. Estos dos flujos expiran después de una cierta duración sin cursar tráfico.

Si ahora comprobamos los hosts detectados en el controlador, a través de la CLI de ONOS, vemos que ya aparecen los dos equipos sobre los que se ha intercambiado con éxito el tráfico ICMP (PC-2 y PC-3)

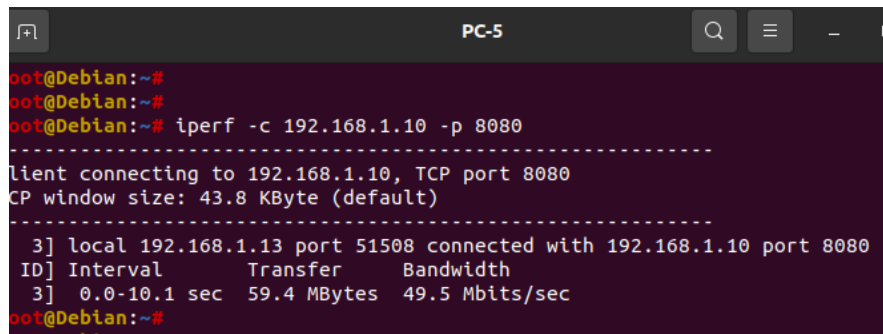
```
karaf@root >
karaf@root >
karaf@root > hosts
id=0C:7F:E7:50:00:00/None, mac=0C:7F:E7:50:00:00, locations=[of:0000000000000001/3], auxLocations
=null, vlan=None, ip(s)=[192.168.1.11], innerVlan=None, outerTPID=unknown, provider=of:org.onospr
object.provider.host, configured=false
id=0C:BB:C9:EA:00:00/None, mac=0C:BB:C9:EA:00:00, locations=[of:0000000000000001/2], auxLocations
=null, vlan=None, ip(s)=[192.168.1.10], innerVlan=None, outerTPID=unknown, provider=of:org.onospr
object.provider.host, configured=false
karaf@root >
karaf@root >
```

Si dejamos lanzado el ping, los flujos se mantienen y podremos ver en la GUI de ONOS la topología actualizada indicando además los enlaces activos y el tráfico cursado en los mismos.

Si hacemos una prueba con iperf entre PC2 y PC5, ejecutando los siguientes comandos:

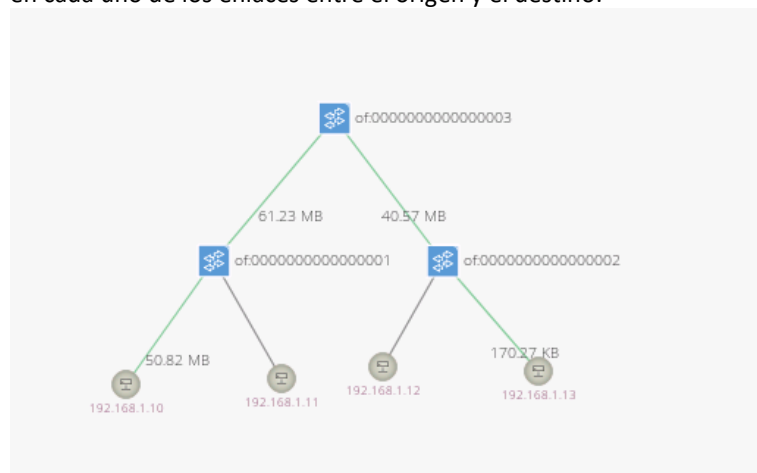
```
PC2# iperf -s -p 8080
```

```
PC5# iperf -c 192.168.1.10 -p 8080
```



```
PC-5
oot@Debian:~#
oot@Debian:~#
oot@Debian:~# iperf -c 192.168.1.10 -p 8080
-----
lient connecting to 192.168.1.10, TCP port 8080
CP window size: 43.8 KByte (default)
-----
 3] local 192.168.1.13 port 51508 connected with 192.168.1.10 port 8080
ID] Interval      Transfer    Bandwidth
 3]  0.0-10.1 sec  59.4 MBytes  49.5 Mbits/sec
oot@Debian:~#
```

Podemos comparar el resultado con el mostrado por ONOS, que además nos da el detalle del ancho de banda en cada uno de los enlaces entre el origen y el destino:



Para más información sobre los comandos en Open vSwitch podemos consultar la guía de referencia de comandos en [\[5\]](#).

3. Mejoras a implementar

3.1 Segmentación de redes (VLAN)

Deberá investigar y aplicar la configuración necesaria en los conmutadores [10] para la implementación de las siguientes LAN virtuales (VLAN) sobre la arquitectura propuesta:

VLAN	Nombre	Dirección de red	Descripción
10	Centrales	192.168.3.0/24	Personal de servicios centrales
20	Oficina	192.168.4.0/24	Personal de oficinas
30	CPD	192.168.5.0/24	Equipamiento en CPD

Tabla 2. LAN virtuales (VLAN) a crear

Red	Equipo	VLAN	Dirección IP	Descripción
Usuarios	PC-2	10	192.168.3.10	Equipo en OpenvSwitch1 de servicios centrales
	PC-3	20	192.168.4.11	Equipo en OpenvSwitch1 de oficinas
	PC-4	20	192.168.4.12	Equipo en OpenvSwitch2 de oficinas
	PC-5	30	192.168.5.13	Equipo en OpenvSwitch2 del cpd

Tabla 3. Direccionamiento asociado

3.2 Esquema alternativo de flujos

Deberá investigar y aplicar la configuración necesaria para que una vez desactivada la app de Reactive Forwarding en el controlador ONOS, sea posible la ejecución con éxito del comando ping entre los equipos PC-2 y PC-4.

Referencias

- [1] Open Networking Foundation (ONF): <https://opennetworking.org/>
- [2] OpenFlow Switch Specification: v1.3.2 <https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf> v1.5 <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.noipr.pdf>
- [3] Open vSwitch: <https://docs.openvswitch.org/en/latest/intro/what-is-ovs/>
- [4] Comandos del CLI de ONOS: <https://wiki.onosproject.org/display/ONOS/Appendix+A+%3A+CLI+commands#AppendixA:CLIcommands-device-remove>
- [5] Guía de referencia de comandos de Open vSwitch: <https://docs.openvswitch.org/en/latest/ref/>
- [6] Tutorial básico de ONOS: <https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial>
- [7] Bug en imagen Docker de Open vSwitch en GNS3: <https://gns3.com/openvswitch-docker-issue>
- [8] ONOS System components: <https://wiki.onosproject.org/display/ONOS/System+Components>
- [9] Tutorial de instalación de Docker en Ubuntu 20.04: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04-es>
- [10] OpenvSwitch Cheat Sheet: <https://adhioutlined.github.io/virtual/Openvswitch-Cheat-Sheet/>