Práctica 7 – OpenFlow y Open vSwitch

1. Objetivos

El objetivo de esta práctica es ver un ejemplo de conmutadores que emplean OpenFlow para comunicarse con un controlador. Se busca entender la flexibilidad que esta arquitectura software puede ofrecer.

2. Introducción

Open vSwitch (OVS)[1] es una solución de conmutador virtual open source integrado en nuevas versiones del kernel Linux aunque disponible para otros UNIX e incluso Windows y empleado en plataformas de virtualización como Xen o KVM, en sistemas de orquestación como OpenStack y OpenNebula y usado también como plano de control en algunos conmutadores hardware.

En comparación con la funcionalidad clásica de *bridging* en el kernel Linux, OVS ofrece una gran cantidad de funcionalidades, como por ejemplo soporte de VLANs y 802.1Q, agregación de enlaces, NetFlow o de especial interés para esta práctica su soporte de OpenFlow. En esta práctica vamos a ver unos pocos ejemplos de uso de OVS centrados en entender el concepto de la separación entre plano de datos y plano de control, por lo que va a estar muy lejos de ser ni siquiera una introducción a las funcionalidades ofrecidas por OVS.

OVS está formado por varios elementos. Uno de ellos es un módulo en el kernel Linux, incluido desde la versión 3.10 del mismo, encargado de la conmutación de flujos. Este módulo está acompañado por un demonio (ovs-vswitchd) que implementa la lógica del switch y un servidor de base de datos (ovsdb-server) que guarda la configuración de todos los switches creados en la máquina, dando persistencia a los mismos. Varias herramientas acompañan a la instalación, permitiendo gestionar los diferentes elementos; de ellas utilizaremos:

- ovs-vsctl : Permite reconfigurar lo switches controlados por el demonio ovs-vswitchd a base de modificar la base de datos donde se guarda su configuración.
- ovs-appctl : Permite comunicarse con otros demonios de OVS.
- ovs-ofctl : Permite mostrar y modificar configuración de switches referida al protocolo OpenFlow.

Hay más comandos y todos tienen una gran cantidad de opciones y flexibilidad. Se irán describiendo en la práctica los que se necesiten a medida que los escenarios los requieran.

En Ubuntu puede instalar los elementos necesarios instalando el paquete *openvswitch-switch*.

Una vez instalado verá un interfaz de red de nombre ovs-system que no será de utilidad en esta práctica y que puede ignorar. Una vez cree un puente verá un interfaz de red con el nombre del puente. Este interfaz emula el comportamiento del interfaz de puente de las utilidades de *bridging* clásicas de Linux, de modo que permite por ejemplo configurar un interfaz IP para el host en la LAN del puente. Este tipo de interfaz será etiquetado como "*internal*" por el puente. Verá también en los siguientes comandos tanto el concepto de "Port" como de "Interface". En esta práctica no habrá diferencia entre ellos y solo se

[1] https://www.openvswitch.org

distinguirán cuando se haga agregación de enlaces, donde más de un interfaz formará parte del mismo puerto[2].

En los diferentes escenarios planteados en la práctica se hablará de "hosts" para referirse a los extremos de comunicación que se interconectan a través de conmutadores implementados mediante Open vSwitch. Puede llevar a cabo los escenarios de red de muy diversas formas. Puede por ejemplo emplear VMs (Figura 1) para cada uno de los elementos, interconectándolos mediante *internal networks* de VirtualBox (en tal caso recuerde activar el modo promiscuo en los interfaces).

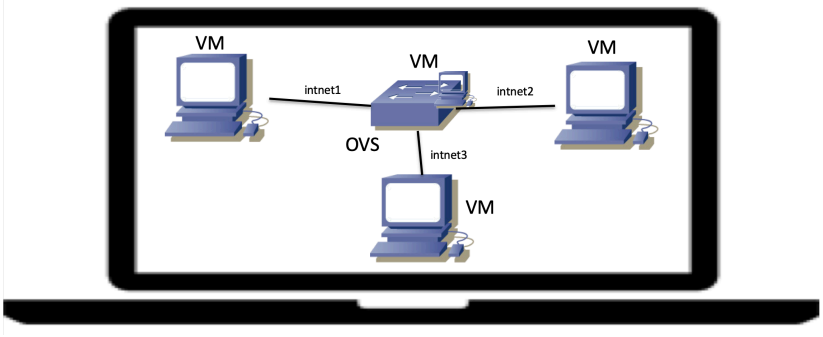

Figura 1 - Implementación con máquinas virtuales

Puede crear todos los conmutadores en la misma máquina Linux (Figura 2) y emplear VMs para actuar como los hosts (en este caso necesita privilegios en esa máquina Linux para trabajar con la pila de red y con los conmutadores).

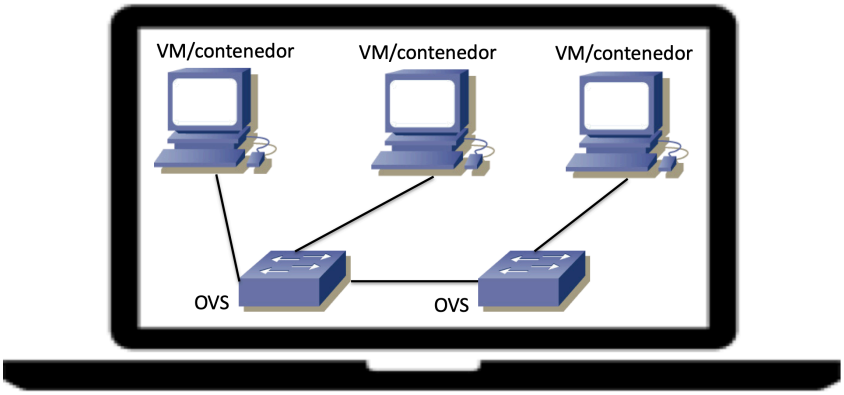

Figura 2 - Implementación con switches en el host

Puede crear todos los conmutadores en la misma máquina Linux (que puede ser una VM) y emplear contenedores o *network namespaces* para implementar los hosts.

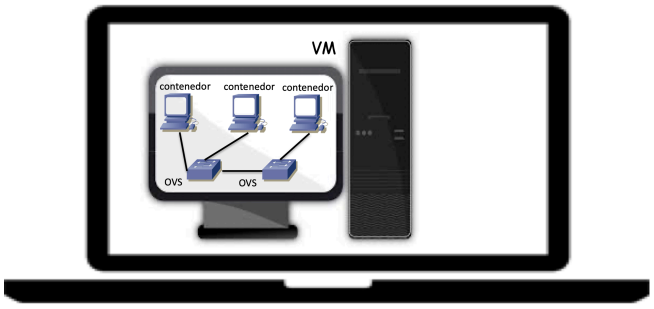

Figura 3 - Implementación dentro de una VM

[2] https://docs.openvswitch.org/en/latest/topics/bonding/

Y por supuesto puede llevar a cabo una combinación de las anteriores opciones (Figura 4), donde por ejemplo algunos hosts son VMs, algunos conmutadores se implementan solos en otra VM y otras VMs hospedan tanto conmutadores como contenedores. Queda a su elección cómo abordar la práctica.

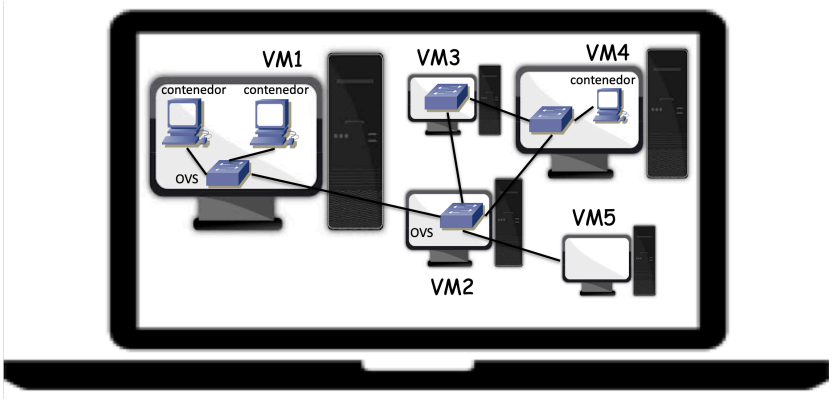

Figura 4 - Implementación con mezcla de VMs y contenedores

3. OVS como un puente tradicional

En primer lugar veremos el funcionamiento de un switch OVS como conmutador tradicional (*learning bridge*). Para ello crearemos la topología de la Figura 5, con los 3 hosts con dirección IP en la misma subred.

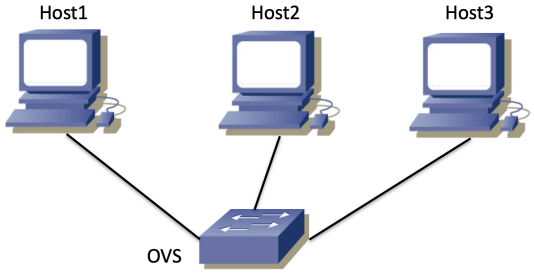

Figura 5 - Escenario de red con 3 hosts a un switch, todo en la misma subred IP

Para crear el conmutador emplee el comando ovs-vsctl :[3]

```
ovs-vsctl add-br <nombreSwitchOVS>
```

El conmutador que ha creado no tiene todavía ningún puerto a los hosts y solo el puerto interno. Puede ver todos los conmutadores OVS con:

```
ovs-vsctl show
```

Los conmutadores OVS vienen por defecto con una regla creada que los lleva a funcionar como un *learning bridge*. Puede verla mostrando toda la programación de flujos en el switch mediante el comando:

```
ovs-ofctl dump-flows <nombreSwitchOVS>
```

Cree los 3 hosts. Por ejemplo, si todo esto lo está llevando a cabo en una VM Linux puede crear 3 contenedores. Instale *tcpdump* en esos hosts/contenedores para que podamos comprobar el funcionamiento de conmutador.

[3] En los ejemplos de esta práctica de comandos, la simbología <> representa que se debe sustituir todo eso por lo que corresponda en cada caso, incluidos los símbolos < y >

Cree interfaces virtuales para conectar los hosts al switch. Por ejemplo, puede emplear Virtual Ethernet Devices (veths). Para añadir un interfaz como puerto del switch emplee el comando:

```
ovs-vsctl add-port <nombreSwitchOVS> <interfaz>
```

Puede ver los puertos en el switch de nuevo con *ovs-vsctl show*.

Compruebe el funcionamiento de switch. Para ello puede poner un ping entre el host1 y el host2 y un tcpdump en host3 para comprobar que no ve en este último los paquetes.

Puede ver la tabla de direcciones MAC aprendida por el switch mediante:

```
ovs-appctl fdb/show <nombreSwitchOVS>
```

Puede comprobar que el switch deja de reenviar paquetes según el comportamiento de un *learning bridge* si elimina el flujo especial (con la acción "NORMAL") que tiene programado mediante el comando (que elimina todos los flujos):

```
ovs-ofctl del-flows <nombreSwitchOVS>
```

Puede volver a introducir esa regla mediante:

```
ovs-ofctl add-flow <nombreSwitchOVS> actions=NORMAL
```

4. Programación manual de flujos en el switch

En este escenario configuraremos en el switch reglas para flujos concretos que permitan que funcione la comunicación entre los hosts pero sin que sea por un aprendizaje tradicional. La topología es la de la Figura 5.

Ponga un ping continuo del host1 al host2 mientras tiene *tcpdump* corriendo en host2 y host3. Elimine el flujo que hace que el conmutador funcione como un *learning switch*. Debería dejar de recibir paquetes en host2 y host3.

En los comandos para configurar flujos en el switch podrá emplear el nombre de los interfaces o el número que ha dado OpenFlow a los mismos. Estos números los puede obtener del comando:

```
ovs-ofctl show <nombreSwitchOVS>
```

Detenga el ping de host1 a host2 y ponga un ping de host1 a una dirección IP de su subred para la que no haya ningún interfaz configurado. Con ello pretendemos que el host envíe paquetes *ARP request* dirigidos a la dirección MAC de broadcast.

Para que los broadcast inunden la LAN programaremos la siguiente regla:

```
ovs-ofctl add-flow <nombreSwitchOVS> dl_dst=ff:ff:ff:ff:ff:ff,actions=flood
```

Con esta regla le estamos indicando que todas las tramas Ethernet que reciba y que se dirijan a la dirección MAC FF:FF:FF:FF:FF:FF deben seguir la acción de inundación, es decir, que se reenvíen por todos los puertos menos por el que se hayan recibido.

Con esta regla deberíamos estar recibiendo los broadcast en host2 y host3. Para que funcione correctamente el ARP necesitamos no solo que llegue la pregunta sino también la respuesta.

Abra una nueva shell en el host1. Ponga a correr un ping del host1 al host3 mientras en la otra shell de host1 tiene un *tcpdump*. A causa del ping el host1 debería enviar el ARP request

para averiguar la dirección MAC del host3[4]; este ARP llegará al host3 gracias al proceso de inundación por la regla que hemos configurado y el host3 contestará (lo verá en la salida de su *tcpdump)* pero la respuesta no llegará al host1 porque no hay una regla en el switch que le indique qué hacer con esta trama Ethernet (no va dirigida a broadcast).

Introduzca una regla que le indique al switch qué hacer con las tramas que van dirigidas a la dirección MAC del host1:

```
ovs-ofctl add-flow <nombreSwitchOVS> dl_dst=<MAChost1>,action=output:
<puertoHost1enSwitch>
```

Con esta regla el *ARP reply* debería ser reenviado a host1, sin embargo, cuando ahora éste envíe el paquete IP del mensaje ICMP del ping irá dirigido a la dirección MAC del host2, para la cual el switch no tiene ninguna regla y de nuevo lo descartará.

Programe las reglas que puedan faltar para que funcione la comunicación IP entre los 3 hosts (todos con todos).

Punto de control 1 (20%): Muestre las reglas configuradas y la comunicación entre todos los hosts.

Puede eliminar una entrada en la tabla de flujos mediante el comando:

```
ovs-ofctl del-flows <match>
```

Deberá sustituir <match> por las condiciones que debían cumplir los paquetes en la regla que quiere eliminar (no debe incluir las acciones que se toman en la regla).

5. Configuración de controlador OpenFlow

En vez de programar las reglas manualmente vamos a configurarle al conmutador que se comunique mediante el protocolo OpenFlow con un controlador que será quien indique al conmutador lo que tiene que hacer con los paquetes o que programe las reglas de reenvío. Esto lo podrá hacer el controlador cuando él desee o puede ser como reacción ante la llegada de ciertos paquetes al switch controlado.

En este primer ejemplo el conmutador delegará en el controlador la decisión de qué hacer con cada paquete y el controlador indicará la acción para el paquete sin programar reglas.

El controlador (Figura 6) es un simple programa que acepta conexiones TCP a un puerto que suele ser el puerto reservado para OpenFlow (puerto 6653) y que emplea este protocolo en la comunicación con el otro extremo del stream TCP. Para que el conmutador OVS pueda comunicarse con el controlador hace falta que en el *network namespace* en el que esté creado el switch OVS haya algún interfaz IP a través del cual pueda llevarse a cabo esta comunicación.

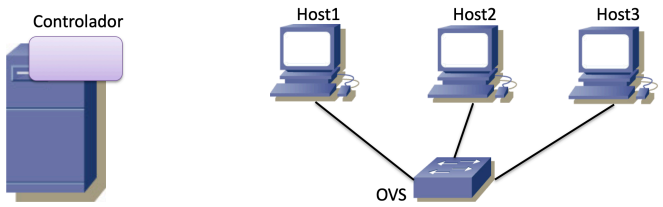

Figura 6 - Controlador y conmutador OVS

[4] Si no ve el ARP de host1 preguntando por host3 puede que sea porque haya llevado a cabo comunicación entre ellos con anterioridad y host1 conozca ya la dirección MAC de host3. Puede eliminar ese conocimiento de host1 con el comando *arp*

En la Figura 7 se muestra un ejemplo de cómo llevar a cabo un escenario de este estilo. En ella el controlador es un programa corriendo en una VM VirtualBox mientras que el switch OVS está en otra VM junto con los hosts, que son tres contenedores o *network namespaces*. Las dos VMs tienen interfaces en una *internal network* de VirtualBox con dirección IP en una subred específica (independiente y aislada de la LAN del switch OVS).

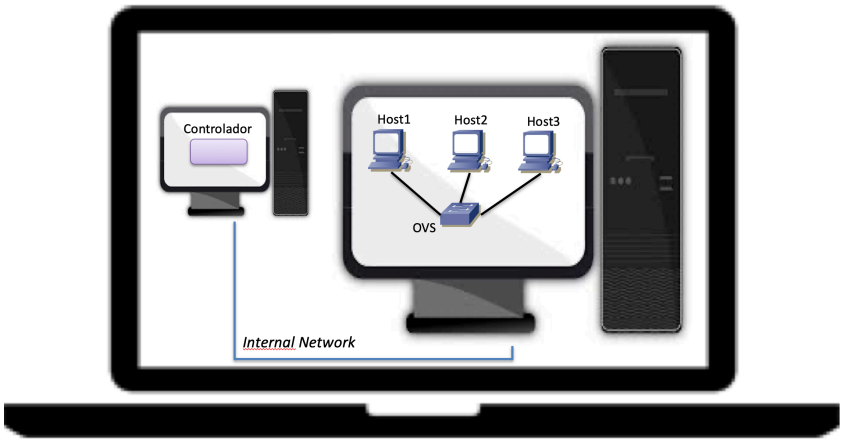

Figura 7 - Controlador en VM separada

Otra opción sencilla sería correr el software de controlador en la misma VM en la que se encuentra el switch OVS y llevar a cabo la comunicación entre ambos a través del interfaz de loopback de dicha máquina. Esta opción es más simple pero deja menos clara la posible separación del plano de datos y de control en diferentes máquinas.

Como software de controlador vamos a emplear POX[5], un controlador escrito en Python y en el que será sencillo modificar las acciones del mismo. Si tiene *git* instalado en la máquina donde va a correr el controlador, para descargar las fuentes tiene que hacer:

```
git clone http://github.com/noxrepo/pox
```

Con eso tendrá un directorio *pox* que contiene el código del controlador. En esta práctica partiremos de los ejemplos de un tutorial. Puede encontrar una descripción de POX en el tutorial de Mininet[6]. Mininet ofrece un sistema de creación de topologías de red mediante contenedores y es muy utilizado para la creación de prototipos con OpenFlow. En esta práctica no emplearemos Mininet para evitar describir otro sistema de gestión de contenedores, pero toda la práctica se podría hacer con él.

Para lanzar el controlador con el código del tutorial puede hacer:

```
pox/pox.py log.level --DEBUG misc.of_tutorial openflow.of_01 --
port==6653
```

Si no indica el puerto es probable que el servidor espere en uno diferente del estándar (es probable que el código del controlador sea anterior a que se reservara un puerto para OpenFlow).

Elimine todos los flujos programados en el conmutador. Por defecto los conmutadores OVS pueden funcionar como un puente tradicional cuando pierde contacto con el controlador. Desactivaremos por si acaso esta funcionalidad con:

```
ovs-vsctl set bridge <OVS> fail-mode="secure"
```

Para configurarle al switch la dirección del controlador debe hacer:

---

[5] https://noxrepo.github.io/pox-doc/html/

[6] https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch#Controller_Choice_POX_Python

```
ovs-vsctl set-controller <OVS> tcp:<IPcontrolador>
```

En caso de que el puerto del controlador no fuera el 6653 puede configurar el puerto con el que debe establecer la conexión TCP el conmutador añadiendo ":<puerto>" al final del comando anterior.

El código por defecto del tutorial de POX hace que actúe de forma similar a un hub, en el sentido de que hace inundación de todos los paquetes. Compruebe que es así.

Inspeccione el tráfico OpenFlow entre el conmutador y el controlador empleando Wireshark para ver los mensajes del protocolo.

Punto de control 2 (20%): Muestre el escenario en funcionamiento y el tráfico OpenFlow en Wireshark.

6. Learning switch mediante controlador

Siguiendo las indicaciones del tutorial, modifique el fichero pox/pox/misc/of_tutorial.py para que el controlador actúe ahora como un *learning bridge*, donde el controlador debe tomar todas las decisiones para todos los paquetes pero aprende direcciones MAC asociadas a puertos del switch.

Punto de control 3 (25%): Muestre este escenario en funcionamiento, así como las modificaciones al código del controlador.

7. Programación de reglas desde el controlador

En los escenarios anteriores todo el tráfico que llega al conmutador se envía al controlador para que tome las decisiones de reenvío. Aunque se puede enviar paquete recortados (sin todos los datos) al controlador, sigue siendo una cantidad de trabajo y tráfico excesiva.

Normalmente un controlador va a programar reglas de reenvío de flujos a priori en el conmutador o va a programarlas ad-hoc para los flujos que vea que aparezcan.

En esta sección se quiere que cuando el conmutador vea un nuevo paquete de un flujo (definido por dirección MAC origen y destino, ambas unicast) programe reglas de reenvío. En concreto haremos que el controlador aprenda la localización (puerto) de cada dirección MAC y cuando vea un paquete entre dos direcciones MAC cuya ubicación conozca, que programe las reglas de reenvío para el flujo en cada sentido, de forma que ya no se enviarán al controlador más paquetes de ese flujo bidireccional, sino que serán reenviados directamente por el conmutador.

El BUM seguirá sufriendo flooding.

Punto de control 4 (20%): Muestre este escenario en funcionamiento, así como las modificaciones al código del controlador.

8. Ingeniería de tráfico

Cree un escenario con 4 conmutadores OVS (Figura 8). Dos de ellos representan switches virtuales en ordenadores donde corren VMs o contenedores mientras que los otros dos representan conmutadores físicos controlables también mediante OpenFlow. El controlador correría en una máquina independiente.

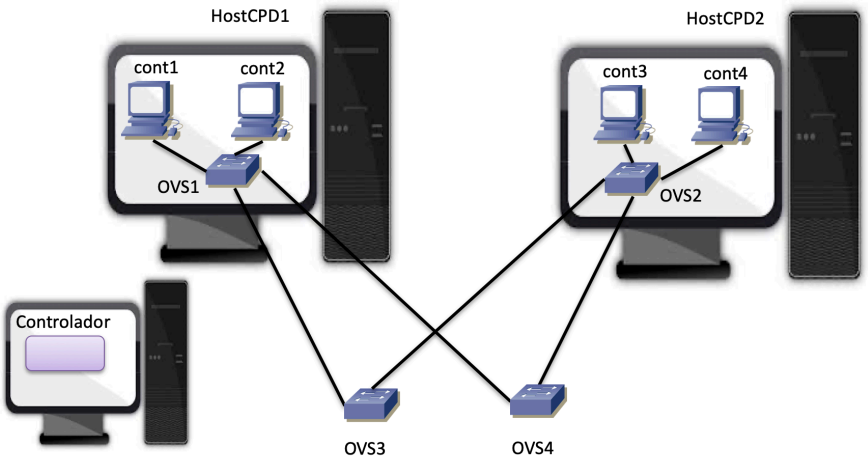

Figura 8 - Escenario para ingeniería de tráfico

Puede implementar este escenario con VMs para las máquinas HostCPD1 y HostCPD2 que hospedan los contenedores, otras dos VMs para los conmutadores OVS3 y OVS4 y una quinta VM para correr en ella el software del controlador. En este caso, para la comunicación de los switches con el controlador puede crear una *internal network* en la que estén todas las VMs.

En realidad no vamos a entrar en mucho detalle sobre cómo implementar la comunicación entre los conmutadores y el controlador, ya que si eso nos preocupara deberíamos preocuparnos también de ofrecer redundancia de dicho controlador. Todo esto quedará fuera del alcance de esta práctica. De hecho, si quiere puede crear todo el escenario en un solo host (o una sola VM) creando en él e interconectando todos los switches virtuales y lanzando ahí todos los contenedores, así como el software del controlador. A efectos de networking (protocolos, packetes, etc) no habrá diferencia.

El controlador en este escenario configurará reglas a-priori en los conmutadores cuando reciba las conexiones de los mismos. Para esto deberá poder identificar a los diferentes conmutadores (es decir, debe saber cuál es la dirección de OVS1 por ejemplo para distinguirlo de OVS2), lo cual se lo configuraremos de manera estática (por ejemplo con las direcciones MAC de los conmutadores en el propio código del controlador).

Cuando el controlador reciba las conexiones de los conmutadores les configurará las reglas necesarias para conseguir que el tráfico entre cont1 y cont2 sea local en OVS1, el tráfico ente cont3 y cont4 sea también local en OVS2, el tráfico entre cont1 y cont3 pase por OVS3 y el tráfico entre cont2 y cont4 pase por OVS4.

No habrá reglas para el broadcast, así que éste será descartado por los conmutadores. Para evitar que esto dé problemas con el funcionamiento de ARP deberá configurar a mano en cada contenedor la correspondencia de dirección IP a MAC de los demás contenedores, de forma que nunca necesiten hacer ARP.

Punto de control 5 (15%): Muestre este escenario en funcionamiento, así como las modificaciones al código del controlador.