Diseño y Administración de Redes

Práctica 3.2

Diseño y Gestión de Tecnologías LANC mediante OvS

Dpto. Ingeniería Electrónica y Comunicaciones
Área de Ingeniería Telemática

Departamento de Ingeniería Electrónica y Comunicaciones
Universidad Zaragoza

Autores:
Profesores del área de Ingeniería Telemática

1. Introducción

1.1. Objetivos

Tras la realización de esta práctica, el alumno deberá ser capaz de:
-Configurar y monitorizar equipos basados en tecnología LAN Conmutada que utilicen OvS.
-Administrar o gestionar en OvS características propias de LANC como son las tablas de conmutación o la creación de LAN virtuales.

1.2. Contenidos

Los objetivos propuestos en esta práctica pretenden afianzar y complementar los **contenidos teóricos sobre SDN vistos en clase**. Por lo tanto, será necesario un estudio previo de los mismos, así como la utilización de los apuntes de clase (y cualquier material adicional que el alumno considere oportuno) como apoyo a la realización práctica.

1.3. Equipos, tecnologías y herramientas

Vamos a trabajar en GNS3 con 5 equipos distintos: switch Ethernet, cloud, TinyCoreLinux6.4, OpenvSwitch-management y Ostinato0.9. El objetivo es configurar una red LAN para gestionar, controlar y transmitir información en un escenario de pruebas SDN (Software Defined Networks) y conectar dicha LAN a Internet a través de un router que haga NAT.

A continuación, recordamos las máquinas presentes en el escenario.

Switch Ethernet. Es un switch ethernet que viene con el propio GNS3.

Cloud. Es el dispositivo que nos va a permitir que nuestro escenario tenga conexión al exterior mediante la interfaz ethernet fija del ordenador donde está instalado GNS3. También viene con el GNS3.

TinyCoreLinux6.4. Por simplificar lo llamaremos TCLinux6.4. Es una distribución ligera de Linux, que se utiliza para equipos con pocos recursos de memoria y disco. La vamos a usar como router de nuestro escenario y nos permitirá la conexión de la red LAN (intranet) con Internet, mediante NAT. También la usaremos para equipo conectado a la red de datos del escenario.

OpenvSwitch-management. Por simplificar lo llamaremos OvS-gestion. Los switch Ethernet virtuales para la SDN utilizarán OpenvSwitch, que es un software que puede instalarse sobre diferentes sistemas operativos. En nuestro caso se ejecuta en una máquina que sigue una distribución Linux Alpine. La virtualización la vamos a hacer sobre docker; como ya viene con el propio GNS3, no habrá que instalarla.

Ostinato0.9. Por simplificar lo llamaremos Ost0.9. Es también una máquina sobre tinyCoreLinux que dispone de un generador de tráfico y software de captura para hacer las pruebas correspondientes de funcionamiento de equipos conectados a la red de datos del escenario.

Un paso previo, que ya hemos explicado en la práctica de introducción de la asignatura es añadir los new template de TinyCoreLinux6.4 y Ostinato0.9. Para OpenvSwitch-management no es necesario porque viene con el propio GNS3.

En cuanto a las **herramientas** necesarias para la verificación y el análisis de los escenarios, utilizaremos el software de captura ***tcpdump*** y el analizador de protocolos ***Wireshark***.

1.4. Escenarios

A continuación, se muestra la disposición de los tres escenarios con los que se trabajará.

La Figura 1.1 nos muestra un escenario complejo para crear redes independientes de control, gestión y datos.

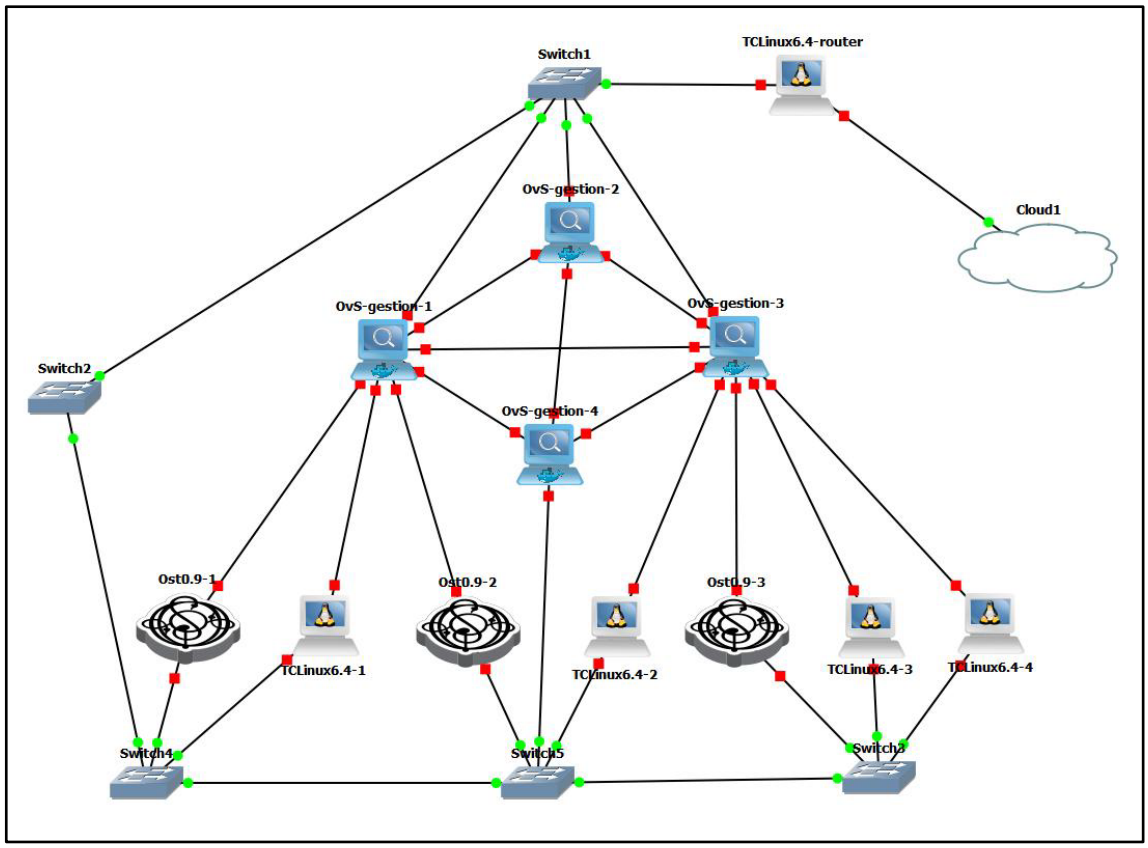

Figura 1.1: Escenario de interconexión de redes para la bancada N.

Seguiremos las siguientes pautas:

Eth0 de las máquinas TCL y Ost se conectan a la red de gestión y control

Eth1 de las máquinas TCL y Ost se conectan a la red de datos que están formadas por las máquinas OVS.

Eth0 de las máquinas OVS se conecta a la LAN donde se encuentra la eth0 de TinyRouter.

Eth1 de TinyRouter de conecta a Cloud para dar salida al exterior.

La Figura 1.2 nos muestra un escenario simple de una red de datos para crear e interconectar VLAN.

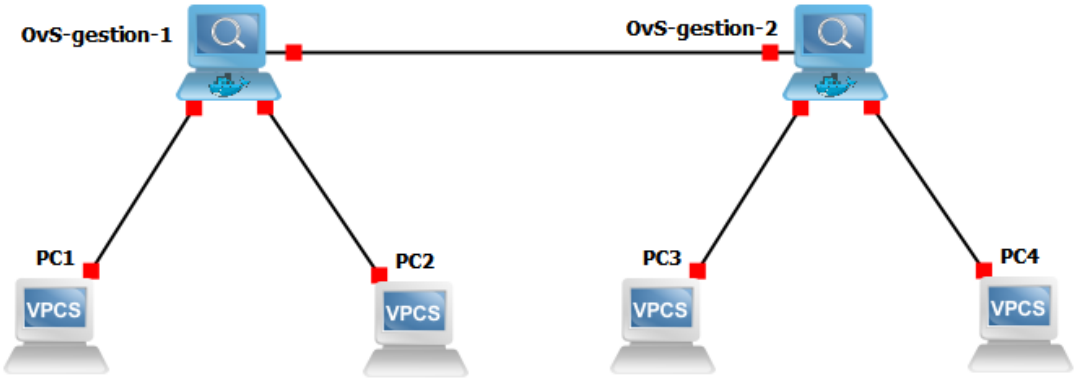

Figura 1.2: Escenario VLAN simple.

La Figura 1.3 nos muestra un escenario IP de una red de datos para crear e interconectar VLAN mediante un túnel GRE.

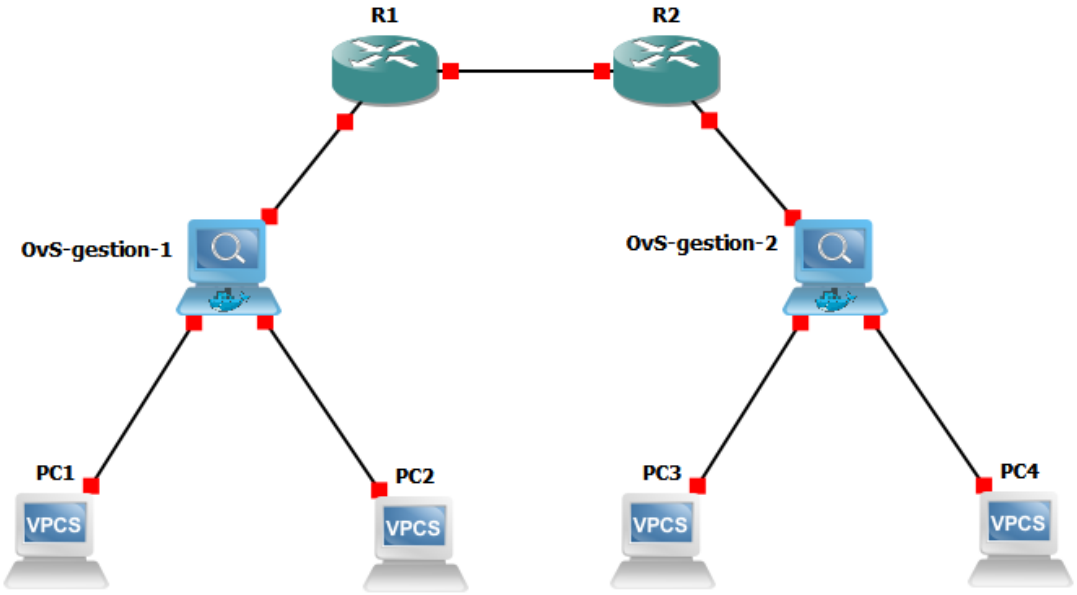

Figura 1.3: Escenario VLAN IP con túnel GRE.

2. Realización práctica en el laboratorio

A continuación, procedemos a realizar la siguiente práctica en GNS3, con las indicaciones que vienen a continuación.

**** La evaluación de esta práctica requiere la entrega del escenario GNS3 con la configuración adecuada, las capturas correspondientes que avalen su correcto funcionamiento y un documento con la explicación oportuna. ****

**** Para facilitar la elaboración del documento explicativo aparecen una serie de cuestiones a lo largo del enunciado de la práctica que deberemos ir contestando. ****

2.1. Configuración de las redes IP de gestión y control y de datos

Este apartado 2.1 no es necesario hacerlo, pero se muestra como ejemplo de cómo sería esta configuración de redes superpuestas. Trabajaremos sobre la figura 1.1.

2.1.1. Configuración inicial:

Máquina TinyRouter:

Es la máquina que actuará como *router* de la red de control y gestión. Hay que configurar los siguientes ficheros:

/opt/bootlocal.sh

```
#!/bin/sh
…
sysctl –w net.ipv4.ip_forward=1 #para activar el forwarding
/opt/eth1.sh& #para activar el interfaz de salida a internet
/opt/eth0.sh& #para activar el interfaz interno en la red de control y gestión
Sudo iptables –t nat –A POSTROUTING –o eth0 –j MASQUERADE #hacer NAT
```

/opt/eth1.sh para la red 192.168.1.0/24 o la que tenga en internet

```
#!/bin/sh
pkill udhcpc
ifconfig eth1 192.168.1.N netmask 255.255.255.0 broadcast 192.168.1.255 up
route add default gw 192.168.1.R
echo nameserver 8.8.8.8 > /etc/resolv.conf
```

/opt/eth0.sh para la red interna de control y gestión, por ejemplo vamos a usar la 192.168.7.0/24

```
#!/bin/sh
pkill udhcpc
ifconfig eth0 192.168.7.R netmask 255.255.255.0 broadcast 192.168.1.255 up
```

Comprobar que los tres ficheros tienen permiso de ejecución y en caso de no tenerlo dárselo con: sudo chmod 755 “file”

Y después, para que haga permanentes los cambios hay que ejecutar

```
filetool.sh -b
```

Máquinas TinyN y ostinato:

Son las máquinas que actuarán como equipos de la red de datos. Hay que configurar los siguientes ficheros:

/opt/bootlocal.sh

```
#!/bin/sh
…
/opt/eth0.sh& #para activar el interfaz de gestión
/opt/eth1.sh& #para activar el interfaz de datos
```

/opt/eth0.sh para la red 192.168.7.0/24 o la que tenga para gestión

```
#!/bin/sh
pkill udhcpc
ifconfig eth0 192.168.7.N netmask 255.255.255.0 broadcast 192.168.7.255 up
route add default gw 192.168.7.R
echo nameserver 8.8.8.8 > /etc/resolv.conf
```

/opt/eth1.sh para la red 192.168.13.0/24 o la que tenga para datos

```
#!/bin/sh
pkill udhcpc
ifconfig eth1 192.168.13.N netmask 255.255.255.0 broadcast 192.168.13.255
up
```

Comprobar que los dos ficheros tienen permiso de ejecución y en caso de no tenerlo dárselo con: sudo chmod 755 “file”

Y después, para que haga permanentes los cambios hay que ejecutar

```
filetool.sh -b
```

En cuanto tengamos conexión a internet podemos comprobar si tenemos el traffic control (tc) que necesitaremos para hacer pruebas. Para eso nos conectamos al repositorio

```
tce-update
tce-load  -iw iproute2.tcz
```

Máquinas OVS-gestion-N

Conectamos eth0 a la red de control y gestión 192.168.7.0/24. Y ethN a los equipos de datos.

Hay que configurar los siguientes ficheros:

/etc/network/interfaces

```
# Static config for eth0
auto eth0
iface eth0 inet static
    address 192.168.7.N
    broadcast 192.168.7.255
    netmask 255.255.255.0
    gateway 192.168.7.R #es la dirección del router
    up echo nameserver 8.8.8.8 > /etc/resolv.conf
```

En cuanto tengamos conexión a internet podemos comprobar si tenemos el traffic control (tc) que necesitaremos para hacer pruebas. Para eso nos conectamos al repositorio

```
apk update
apk add iproute2 #paquete con tc
```

2.1.2. Configuración de la conmutación en la red de datos

Máquinas OVS-gestion-N

Si no existe el fichero /etc/openvswitch/conf.db que es la configuración guardada en la base de datos para openvswitch, el equipo se reinicia por defecto con una configuración tal y como está programado en /bin/boot.sh

Comenzamos comprobando cómo tenemos configurado el OvSwitch

```
ovs-vsctl show
```

Si queremos partir de una situación inicial limpia de configuraciones deberemos borrar todos los *bridge* creados por defecto.

```
ovs-vsctl del-br br0 # repetir este comando para cada bridge
```

Continuamos creando el br0 partiendo de una configuración vacía.

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

Vamos a añadir diferentes puertos al *bridge* que hemos creado. No debemos añadir eth0 porque lo vamos a usar para gestión y control.

```
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth2
ovs-vsctl add-port br0 eth3
…
```

Si cada vez que se reinicia la máquina, ésta toma la configuración por defecto, podemos programar un *script* con las órdenes anteriormente descritas y ejecutarlo en cuanto la arranquemos. También podemos modificar el fichero /bin/boot.sh para que por defecto se inicialice con la configuración deseada.

Para comprobar los diferentes elementos configurados lo haremos con el comando

```
ovs-vsctl list TBL
```

Donde TBL son las posibles tablas de la base de datos. Por ejemplo, si en TBL ponemos Interface nos listará los puertos y si en TBL ponemos Bridge nos listará los switch. Si no queremos ver todos los parámetros de cada elemento sino sólo algunos, debemos indicarlo con la opción --columns= y poner aquellos parámetros deseados.

Por ejemplo, si queremos saber qué índice le ha asignado a cada puerto, vamos a comprobar los parámetros ofport y name, correspondientes a cada interface. El parámetro ofport será el que usemos en la configuración.

```
ovs-vsctl  --columns=name,ofport list Interface
```

Vamos a continuar haciendo algunas pruebas. Por defecto, el bridge que hemos definido trabaja en modo standalone (significa que si no tiene conexión al controlador se configura como learnig switch). Sin embargo, no tiene activado por defecto el protocolo STP y para casos en los que pueda haber múltiples caminos, esto sería conveniente.

Podemos comprobar el correcto funcionamiento haciendo un ping desde tiny1 a tiny2 y debe funcionar.

Vamos a configurar el protocolo STP.

Lo primero que haremos será activarlo en los bridge de cada OvS.

ovs-vsctl set br br0 stp_enable=true

A continuación, damos pesos a los enlaces, por ejemplo:

ovs-vsctl set port eth1 other_config={stp-path-cost=1000}

De esta forma y poniendo pesos distintos para los enlaces veremos como se adaptan los caminos.

También podemos configurar el bridge en modo seguro (significa que si no tiene conexión al controlador no se configure como "learnig switch", y que no tenga ningún flujo definido). También podemos deshabilitar el protocolo STP para que no haya información innecesaria.

ovs-vsctl set br br0 stp_enable=false

ovs-vsctl set-fail-mode br0 secure

Ahora el ping no funciona. Y debemos definir todos los flujos manualmente.

A partir de aquí no es necesario continuar con la configuración y pasamos al siguiente apartado 2.2. No obstante, toda la información siguiente puede ser usada para entender mejor el funcionamiento.

Continuamos definiendo flujos emulando como lo haría un "learning switch". En primer lugar, identificamos todo flujo que tiene dirección destino broadcast y los encaminamos hacia el resto de puertos.

ovs-ofctl add-flow br0 in_port=1,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,…n

…

ovs-ofctl add-flow br0 in_port=n,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:resto de puertos

Para ver todos los flujos que tenemos activados podemos ejecutar

ovs-ofctl dump-flows br0

Habrá que poner especial cuidado con los bucles en la red. Si no lo hacemos bien, corremos el riesgo de provocar tormentas de tráfico. Algo parecido al STP, bloqueando un enlace si fuera necesario. Una vez configurados los flujos broadcast, vemos que sigue sin funcionar el ping. Pero podemos comprobar que sí que se reciben los "arp request" porque tiene como destino la dirección MAC broadcast. Sin embargo no se reciben los "arp reply". Habrá que actuar sobre los tráficos con dirección MAC destino las MAC de las máquinas presentes en la red de datos para que funcione el protocolo ARP.

ovs-ofctl add-flow br0 in_port=2,dl_dst=MACtiny1,actions=output:1

…

ovs-ofctl add-flow br0 in_port=i,dl_dst=MACtiny1,actions=output:1

ovs-ofctl add-flow br0 in_port=1,dl_dst=MACtinyN,actions=output:N

…

ovs-ofctl add-flow br0 in_port=j,dl_dst=MACtinyN,actions=output:N

Estos flujos definidos así, también permiten todo el tráfico IP entre las máquinas presentes en la red de datos. Con esta configuración probamos de nuevo los *ping* y vemos que funcionan, como también lo haría cualquier otra comunicación IP entre extremos.

Si quisiéramos conmutar en función de la dirección MAC origen tendríamos que configurar el comando con dl_src y no dl_dst.

Además de realizar la conmutación a nivel de enlace de datos, podemos hacerlo a nivel IP, a nivel de vlan, a nivel de protocolo ehternet o a nivel de protocolo IP. Para ello, en lugar de poner dl_dst que es data-link_destino, pondríamos nw_dst=direcciónIP (para nivel network), dl_valn=Nºvlan (para VLan, dl_type=0x0800 (para protocolo IP),

dl_type=0x0806 (para protocolo ARP), nw_proto=Nºprotocolo (para los diferentes protocolos de IP).

Ejemplo script SDNL2 máquina OvS-1

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
ovs-vsctl set br br0 stp_enable=true
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2
#ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,dl_dst=0c:06:b3:25:54:01,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,dl_dst=0c:06:b3:40:d6:01,actions=output:1

ovs-ofctl dump-flows br0
```

Ejemplo script SDNL2 máquina OvS-3

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3
ovs-vsctl add-port br0 eth14

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
ovs-vsctl set br br0 stp_enable=true
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface

ovs-ofctl add-flow br0 in_port=2,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:1,3
ovs-ofctl add-flow br0 in_port=1,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,3
ovs-ofctl add-flow br0 in_port=3,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,1
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
```

```
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,dl_dst=0c:06:b3:29:81:01,actions=output:2

ovs-ofctl dump-flows br0
```

Ejemplo script SDNL3 máquina OvS-1

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
ovs-vsctl set br br0 stp_enable=true
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2
#ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.3,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1

ovs-ofctl dump-flows br0
ovs-vsctl show
```

Ejemplo script SDNL3 máquina OvS-3

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3
ovs-vsctl add-port br0 eth14

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
ovs-vsctl set br br0 stp_enable=true
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface

ovs-ofctl add-flow br0 in_port=2,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:1,3
ovs-ofctl add-flow br0 in_port=1,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,3
ovs-ofctl add-flow br0 in_port=3,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,1
```

```
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.3,actions=output:3
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.3,actions=output:3
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:2

ovs-ofctl dump-flows br0
ovs-vsctl show
```

Ejemplo script SDNL3_3 máquina OvS-1

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3
ovs-vsctl add-port br0 eth2

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
ovs-vsctl set br br0 stp_enable=false
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2
#ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:3
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.3,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1

ovs-ofctl dump-flows br0
ovs-vsctl show
```

Ejemplo script SDNL3_3 máquina OvS-2

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
```

```
ovs-vsctl set br br0 stp_enable=false
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface

ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:2
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1

ovs-ofctl dump-flows br0
ovs-vsctl show
```

Ejemplo script SDNL3_3 máquina OvS-3

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth3
ovs-vsctl add-port br0 eth14
ovs-vsctl add-port br0 eth2

#ovs-vsctl set-controller br0 tcp:192.168.13.254:6633
ovs-vsctl set br br0 stp_enable=false
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface

ovs-ofctl add-flow br0 in_port=2,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:1,3
ovs-ofctl add-flow br0 in_port=1,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,3
ovs-ofctl add-flow br0 in_port=3,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,1
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:40:d6:01,actions=output:1
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,dl_dst=0c:06:b3:25:54:01,actions=output:3
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,dl_dst=0c:06:b3:29:81:01,actions=output:2
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:1
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.1,actions=output:4
ovs-ofctl add-flow br0 in_port=4,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:2
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.13.3,actions=output:3
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.13.3,actions=output:3
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_dst=192.168.13.2,actions=output:2

ovs-ofctl dump-flows br0
ovs-vsctl show
```

2.2. Configuración de OvS como Learning Switch

Siguiendo la Figura 1.2, queremos configurar los OvS-gestion como “learning switch”, lo que significa que el switch virtual aprenderá igual que si fuera un switch hardware, como se ha visto en la teoría. Las direcciones IP de los PC las configuraremos en una misma red: 192.168.10.0/24.

Máquinas OVS-gestion-N

Si no existe el fichero /etc/openvswitch/conf.db que es la configuración guardada en la base de datos para openvswitch, el equipo se reinicia por defecto con una configuración tal y como está programado en /bin/boot.sh

Comenzamos comprobando cómo tenemos configurado el OvSwitch

ovs-vsctl show

Si queremos partir de una situación inicial limpia de configuraciones deberemos borrar todos los *bridge* creados por defecto.

ovs-vsctl del-br br0 # repetir este comando para cada *bridge*

Continuamos creando el br0 partiendo de una configuración vacía.

ovs-vsctl add-br br0

Vamos a añadir diferentes puertos al *bridge* que hemos creado. No debemos añadir eth0 porque lo vamos a usar para gestión y control.

ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth2
ovs-vsctl add-port br0 eth3
…

Si cada vez que se reinicia la máquina, ésta toma la configuración por defecto, podemos programar un *script* con las órdenes anteriormente descritas y ejecutarlo en cuanto la arranquemos. También podemos modificar el fichero /bin/boot.sh para que por defecto se inicialice con la configuración deseada.

Para comprobar los diferentes elementos configurados lo haremos con el comando
ovs-vsctl list TBL

Donde TBL son las posibles tablas de la base de datos. Por ejemplo, si en TBL ponemos Interface nos listará los puertos y si en TBL ponemos Bridge nos listará los switch. Si no queremos ver todos los parámetros de cada elemento sino sólo algunos, debemos indicarlo con la opción --columns= y poner aquellos parámetros deseados.

Por ejemplo, si queremos saber qué índice le ha asignado a cada puerto, vamos a comprobar los parámetros ofport y name, correspondientes a cada interface. El parámetro ofport será el que usemos en la configuración.

ovs-vsctl --columns=name,ofport list Interface

Vamos a continuar haciendo algunas pruebas. Por defecto, el bridge que hemos definido no teien definido ningún controlador y trabaja en modo standalone (significa que si no tiene conexión al controlador se configura como learnig switch). Sin embargo, no tiene activado por defecto el protocolo STP y para casos en los que pueda haber múltiples caminos, esto sería conveniente. En nuestra situación no es necesario porque sólo hay un posible camino.

Podemos comprobar el correcto funcionamiento haciendo un ping desde tiny1 a tiny2 y debe funcionar.

Vamos a configurar el protocolo STP.

Lo primero que haremos será activarlo en los bridge de cada OvS.

ovs-vsctl set br br0 stp_enable=true

A continuación, damos pesos a los enlaces, por ejemplo:

ovs-vsctl set port eth1 other_config={stp-path-cost=1000}

De esta forma y poniendo pesos distintos para diferentes enlaces veríamos cómo se adaptaban los caminos.

P1. Deberemos entregar el project que hayamos creado con GNS3. Y las capturas que demuestren que funciona correctamente y en el informe poner las instrucciones que hayamos utilizado en OvS, con su correspondiente explicación. Se propone, que una vez que nos aseguremos de que las tablas ARP de PC1 y PC4 están vacías (si es necesario reiniciamos PC1 y PC4), hacer un ping entre ambos PC y capturar en el enlace entre *switch* comprobando y explicando la existencia y funcionamiento de tramas STP, ARP e ICMP y comprobando los valores MAC e IP (en caso de haberlo).

También podemos configurar el bridge en modo seguro (significa que si no tiene conexión al controlador no se configure como "learnig switch", y que no tenga ningún flujo definido. Si queremos volver al modo ejecutaríamos: *ovs-vsctl set-fail-mode br0 standalone*). También podemos deshabilitar el protocolo STP para que no haya información innecesaria.

ovs-vsctl set br br0 stp_enable=false
ovs-vsctl set-fail-mode br0 secure

Ahora el ping no funciona. Y debemos continuar definiendo todos los flujos manualmente.

2.3. Configuración de flujos en OvS para trabajar con VLAN

Una vez que hemos trabajado con OvS como learning switch, vamos a replicar el escenario de VLAN de la práctica anterior. Pero esta vez, utilizando un Open Virtual Switch y no el switch hardware C3725. Siguiendo la Figura 1.2, queremos configurar PC1 y PC3 en la VLan2 untagged y PC2 y PC4 en Vlan3 untagged. El enlace entre los OvS debe ser tagged con las dos Vlan presentes: 2 y 3. El OvS-2 hará de router para interconectar las Vlan.

Como criterio para definir los flujos en OvS se propone que inicialmente configuremos los OvS como learning-switch y que capturemos el tráfico presente en el escenario. De esta forma sabremos qué flujos debemos definir. A continuación pondremos los OvS como secure y deberemos definir todos los flujos posibles.

Para facilitar la configuración se proporciona el siguiente guión que no está completo, pero sí que tiene todas las instrucciones necesarias:

Ejemplo script SDNVLAN máquina OvS-1

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth2
ovs-vsctl add-port br0 eth15 trunk=100,200
  #si queremos montar un tunel gre podemos hacerlo así.
  #ovs-vsctl add-port br0 eth15 -- set Interface eth15 type=gre options:remote_ip=192.168.7.2
  #ifconfig eth15 192.168.7.1 netmask 255.255.255.0 broadcast 192.168.7.255 up
  #route add default gw 192.168.7.R


ovs-vsctl set br br0 stp_enable=true
ovs-vsctl set-fail-mode br0 secure
```

```
ovs-vsctl -- --columns=name,ofport list Interface

ovs-ofctl add-flow br0 in_port=1,actions=mod_vlan_vid=100,output:3
ovs-ofctl add-flow br0 in_port=2,actions=mod_vlan_vid=200,output:3
ovs-ofctl add-flow br0 in_port=3,dl_vlan=100,actions=strip_vlan,output:1
ovs-ofctl add-flow br0 in_port=3,dl_vlan=200,actions=strip_vlan,output:2

ovs-ofctl dump-flows br0
```

Ejemplo script SDNVLAN máquina OvS-2

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3

ovs-vsctl add-br br0
ovs-vsctl add-port br0 gateway1 -- set interface gateway1 type=internal
ovs-vsctl add-port br0 gateway2 -- set interface gateway2 type=internal
ifconfig gateway1 192.168.100.254 netmask 255.255.255.0 up
ifconfig gateway2 192.168.200.254 netmask 255.255.255.0 up
sysctl -w net.ipv4.ip_forward=1
ovs-vsctl add-port br0 eth3
ovs-vsctl add-port br0 eth4
ovs-vsctl add-port br0 eth15 trunk=100,200
  #si queremos montar un tunel gre podemos hacerlo así.
  #ovs-vsctl add-port br0 eth15 -- set Interface eth15 type=gre options:remote_ip=192.168.7.1
  #ifconfig eth15 192.168.7.2 netmask 255.255.255.0 broadcast 192.168.7.255 up
  #route add default gw 192.168.7.R

ovs-vsctl set br br0 stp_enable=false
ovs-vsctl set-fail-mode br0 secure

ovs-vsctl -- --columns=name,ofport list Interface

ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,nw_dst=192.168.100.3,actions=output:3
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0806,actions=mod_vlan_vid=100,output:5
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,nw_dst=192.168.200.4,actions=output:4
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0806,actions=mod_vlan_vid=200,output:5
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,nw_dst=192.168.100.254,actions=output:1
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0806,actions=mod_vlan_vid=100,output:5
ovs-ofctl add-flow br0 in_port=4,dl_type=0x0806,nw_dst=192.168.200.254,actions=output:2
ovs-ofctl add-flow br0 in_port=4,dl_type=0x0806,actions=mod_vlan_vid=200,output:5
ovs-ofctl add-flow br0 in_port=5,dl_vlan=100,dl_type=0x0806,actions=strip_vlan,output:1,3
ovs-ofctl add-flow br0
in_port=5,dl_vlan=100,dl_type=0x0800,nw_dst=192.168.100.3,actions=strip_vlan,output:3
ovs-ofctl add-flow br0
in_port=5,dl_vlan=100,dl_type=0x0800,nw_dst=192.168.100.254,actions=strip_vlan,output:1
ovs-ofctl add-flow br0
in_port=5,dl_vlan=100,dl_type=0x0800,nw_dst=192.168.200.254/24,actions=strip_vlan,output:1
ovs-ofctl add-flow br0 in_port=5,dl_vlan=200,dl_type=0x0806,actions=strip_vlan,output:2,4
ovs-ofctl add-flow br0
in_port=5,dl_vlan=200,dl_type=0x0800,nw_dst=192.168.200.4,actions=strip_vlan,output:4
ovs-ofctl add-flow br0
in_port=5,dl_vlan=200,dl_type=0x0800,nw_dst=192.168.200.254,actions=strip_vlan,output:2
ovs-ofctl add-flow br0
in_port=5,dl_vlan=200,dl_type=0x0800,nw_dst=192.168.100.254/24,actions=strip_vlan,output:2
```

```
ovs-ofctl add-flow br0
in_port=1,dl_type=0x0800,nw_dst=192.168.100.1,actions=mod_vlan_vid=100,output:5
ovs-ofctl add-flow br0 in_port=1,dl_type=0x0800,nw_dst=192.168.100.3,actions=output:3
ovs-ofctl add-flow br0
in_port=2,dl_type=0x0800,nw_dst=192.168.200.2,actions=mod_vlan_vid=200,output:5
ovs-ofctl add-flow br0 in_port=2,dl_type=0x0800,nw_dst=192.168.200.4,actions=output:4
ovs-ofctl add-flow br0
in_port=3,dl_type=0x0800,nw_dst=192.168.100.1,actions=mod_vlan_vid=100,output:5
ovs-ofctl add-flow br0 in_port=3,dl_type=0x0800,nw_dst=192.168.100.254,actions=output:1
ovs-ofctl add-flow br0
in_port=3,dl_type=0x0800,nw_dst=192.168.200.254/24,actions=strip_vlan,output:1
ovs-ofctl add-flow br0
in_port=4,dl_type=0x0800,nw_dst=192.168.200.2,actions=mod_vlan_vid=200,output:5
ovs-ofctl add-flow br0 in_port=4,dl_type=0x0800,nw_dst=192.168.200.254,actions=output:2
ovs-ofctl add-flow br0
in_port=4,dl_type=0x0800,nw_dst=192.168.100.254/24,actions=strip_vlan,output:2

ovs-ofctl dump-flows br0
```

P2. Deberemos entregar el project que hayamos creado con GNS3. Y las capturas que demuestren que funciona correctamente y en el informe poner las instrucciones que hayamos utilizado en OvS, con su correspondiente explicación. Se propone, que una vez que nos aseguremos de que las tablas ARP de PC1 y PC2 están vacías (si es necesario reiniciamos PC1 y PC2), hacer un ping desde ambos PC (que pertenecen a diferentes redes IP) y capturar en el enlace entre *switch* comprobando y explicando la existencia y funcionamiento de tramas ARP e ICMP y los valores MAC, IP y TTL de estas últimas.

2.4. Configuración de OvS para conectar LAN mediante WAN

Una vez que hemos trabajado con OvS emulando una red SDN (aunque sin controlador, dado que todas las ordenes referentes a los flujos las configuramos manualmente) y hemos replicado el escenario de VLAN de la práctica anterior, vamos a comunicar los OvS mediante un túnel GRE que permite interconectar redes LAN a través de IP. Lo que se consigue encapsulando las tramas Ethernet sobre IP. A esto se le conoce como SDWAN.

Queremos que todos los PC, aunque estén en ubicaciones diferentes trabajen como si estuvieran en una misma LAN.

Las direcciones IP de los PC las configuraremos en una misma red: 192.168.10.0/24. A modo de resumen tendremos 4 redes IP. La primera de ellas (192.168.10.0/24), será en la que estén los 4 PC. En la segunda de ellas (192.168.20.0/24) estarán OvS-1 y R1. En la tercera (192.168.30.0/24) R1 y R2. Y en la cuarta (192.168.40.0/24) R2 y OvS-2.

2.4.1. Configuración OvS como learning switch

Comenzamos Trabajando con la figura 1.2. Por tanto, no aparecen ni R1 ni R2. Además, OvS-1 y OvS-2 estarán en la misma subred 192.168.20.0/24.

En primer lugar, como hemos hecho anteriormente, vamos a trabajar sin definición de flujos, es decir, que sean los OvS los que aprendan en función de las tramas que vayan aparaciendo.

Lo que haremos será volver a definir los bridge como “standalone” para que trabajen como “learning switch”.

A continuación, deberemos cambiar la configuración del interfaz que une los OvS para que implementen un túnel GRE. Como los túneles GRE unen dos máquinas IP deberemos configurar una dirección IP para estos interfaces.
Como en los casos anteriores, para facilitar la configuración se proporciona el siguiente guión que puede no coincidir con nuestra configuración, pero sí que tiene todas las instrucciones:

```
ovs-vsctl del-br br0
ovs-vsctl del-br br1
ovs-vsctl del-br br2
ovs-vsctl del-br br3
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1
ovs-vsctl add-port br0 eth2
ovs-vsctl add-port br0 eth15 -- set Interface eth15 type=gre options:remote_ip=192.168.20.1
ifconfig eth15 192.168.20.2 netmask 255.255.255.0 broadcast 192.168.20.255 up
```

P3. En este caso no será necesario entregar el project que hayamos creado con GNS3. Sino sólo las capturas que demuestren que funciona correctamente y en el informe poner las instrucciones que hayamos cambiado con respecto al project de la P1, con su correspondiente explicación. Se propone, que una vez que nos aseguremos de que las tablas ARP de PC1 y PC4 están vacías (si es necesario reiniciamos PC1 y PC4), hacer un ping entre ambos PC y capturar en el enlace entre *switch* comprobando y explicando la existencia y funcionamiento de tramas GRE donde se encapsulan tramas STP, ARP e ICMP y comprobando los valores MAC e IP (en caso de haberlo).

2.4.2. Configuración OvS con/sin definición de flujos

Continuamos con la configuración de la figura 1.2. Pero a diferencia del apartado anterior deberemos configurar los flujos de OvS mediante las correspondientes órdenes ovs-ofctl. Estas órdenes serán similares a las utilizadas en el apartado 2.3 y 2.4.1 y por tanto no se propone ninguna configuración de ejemplo adicional, sino que habrá que utilizar una combinación de las expuestas anteriormente.

P4. En este caso no será necesario entregar el project que hayamos creado con GNS3. Sino sólo las capturas que demuestren que funciona correctamente y en el informe poner las instrucciones que hayamos cambiado con respecto al project de la P1, con su correspondiente explicación. Se propone, que una vez que nos aseguremos de que las tablas ARP de PC1, PC4, OvS1 y OvS2 están vacías (si es necesario reiniciamos los equipos), hacer un ping entre los PC y capturar en los diferentes enlaces, comprobando y explicando la existencia y funcionamiento de tramas GRE y cómo se encapsulan tramas STP, ARP e ICMP y comprobando los valores MAC e IP (en caso de haberlo).

2.4.3. Configuración OvS y SDWAN con definición de flujos

Ya pasamos a definir un escenario como el presentado en la figura 1.3. Los *router* que vamos a utilizar son los C7200.

Ahora sí, Las direcciones IP de los PC las configuraremos en una misma red: 192.168.10.0/24. A modo de resumen tendremos 4 redes IP. La primera de ellas (192.168.10.0/24), será en la que estén los 4 PC. En la segunda de ellas (192.168.20.0/24) estarán OvS-1 y R1. En la tercera (192.168.30.0/24) R1 y R2. Y en la cuarta (192.168.40.0/24) R2 y OvS-2.
Como ejemplo a seguir para realizar la configuración en los router C7200 se nos facilitan una serie de comandos con todas las funciones necesarias:

```
R1# conf terminal
R1 (config)# int FastEthernet 0/1
R1 (config-if)# no sh
R1 (config-if)# ip address 192.168.7.254 255.255.255.0
R1 (config-if)# exit
R1 (config)# ip route 192.168.8.2 255.255.255.255 192.168.9.8
R1 (config)# exit
R1# write
R1# show ip interface brief (para ver la configuración del direccionamiento)
R1# show ip route (para ver la configuración del encaminamiento)
```

P5. Deberemos entregar el project que hayamos creado con GNS3. También habrá que entragar las capturas que demuestren que funciona correctamente y en el informe poner las instrucciones que hayamos utilizado tanto en los OvS como en los router, con su correspondiente explicación. Se propone, que una vez que nos aseguremos de que las tablas ARP de PC1 y PC4 están vacías (si es necesario reiniciamos PC1 y PC4), hacer un ping entre ambos PC y capturar en el enlace entre *router* comprobando y explicando la existencia y funcionamiento de tramas GRE donde se encapsulan tramas STP, ARP e ICMP y comprobando los valores MAC e IP (en caso de haberlo).