Network namespaces, open vSwitch, mininet

Tecnologías y Protocolos de Internet de Nueva Generación

Departamento de Teoría de la Señal y Comunicaciones y
Sistemas Telemáticos y Computación

Octubre de 2017

Contenidos

Contenidos

Linux namespaces

- El propósito de un **namespace** (espacio de nombres) en Linux es recubrir los recursos de una máquina real para que un conjunto de procesos que se encuentran en un namespace los utilicen de forma aislada, pareciendo que se encuentran solos ejecutándose en una máquina real, sin compartir con nadie dichos recursos.
- Uno de los objetivos de un namespace es **dar soporte a contenedores**, una herramienta para *lightweight virtualization* (virtualización ligera), como por ejemplo `mininet`.
- Linux implementa 6 tipos diferentes de namespaces. Para el propósito de este tema se utilizarán 2:
  - **Mount namespaces**: aislar un conjunto de puntos montaje de un sistema de ficheros para que sólo sea accesible desde un grupo de procesos. Procesos en diferentes *mount namespaces* tendrán diferentes vistas de la jerarquía del sistema de ficheros.
  - **Network namespaces**: proporciona aislamiento de los recursos asociados a la red: direcciones IP, tablas de encaminamiento, directorio `/proc/net`, etc.

Contenidos

Network namespaces

- **Network namespaces**: permite tener instancias separadas de interfaces de red y tablas de encaminamiento que trabajan de forma independiente.
  - Esta característica es interesante dentro de una máquina para emular el comportamiento de una red definiendo máquinas virtuales conectadas entre ellas a través de interfaces de red virtuales.
- Todas las máquinas Linux tienen por defecto un *root network namespace* donde están definidas las funcionalidades de red de la máquina, compartidas por todos los procesos que se ejecutan en ella.
- Para crear un network namespace `h1` diferente del `root`:

```
$ ip netns add h1
```

- Para ejecutar un comando dentro de un espacio de nombres, se usa `ip netns exec <networkNamespace> <comando>`. Por ejemplo, hacer un `ping` a la dirección de loopback del espacio de nombres `h1`, se utiliza:

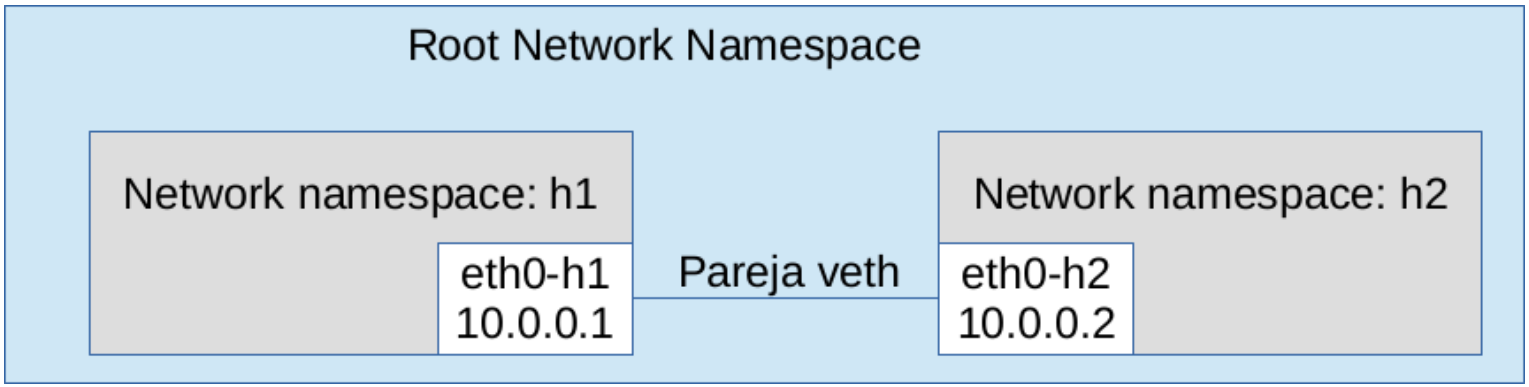
```
$ ip netns exec h1 ping 127.0.0.1
connect: Network is unreachable
```

- Los espacios de nombres no tienen activada la interfaz local. Para activarla:

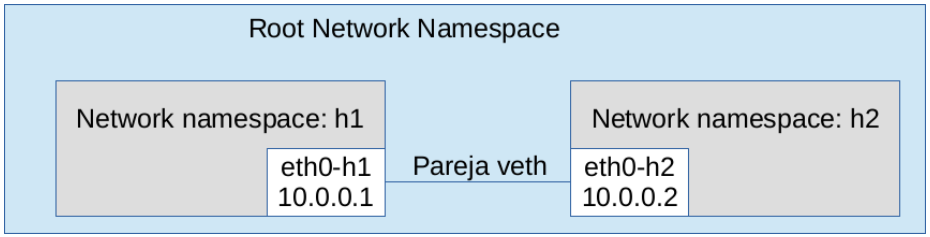
```
$ ip netns exec h1 ip link dev lo up
$ ip netns exec h1 ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmpo_seq=1 ttl=64 time=0.051 ms
...
```

Virtual Ethernet (veth)

- Los *network namespaces* se conectan entre ellos a través de parejas de dispositivos **veth** (*virtual ethernet*). Esto es similar a tener un cable conectando los extremos `veth` al mismo nivel de enlace.
- Cada extremo de un `veth` se coloca en un *network namespace* diferente.



Configuración de Virtual Ethernet (veth)




```
$ ip netns add h1
$ ip netns add h2
$ ip netns exec h1 ip link set dev lo up
$ ip netns exec h2 ip link set dev lo up
$ ip link add eth0-h1 type veth peer name eth0-h2
$ ip link set eth0-h1 netns h1
$ ip link set eth0-h2 netns h2
$ ip netns exec h1 ip address add 10.0.0.1/24 dev eth0-h1
$ ip netns exec h2 ip address add 10.0.0.2/24 dev eth0-h2
$ ip netns exec h1 ip link set dev eth0-h1 up
$ ip netns exec h2 ip link set dev eth0-h2 up
$ ip netns exec h1 ping 10.0.0.2
```

Crea elemento enlace tipo veth con 2 extremos: **eth0-h1** y **eth0-h2**

Lleva a **h1** el extremo **eth0-h1**

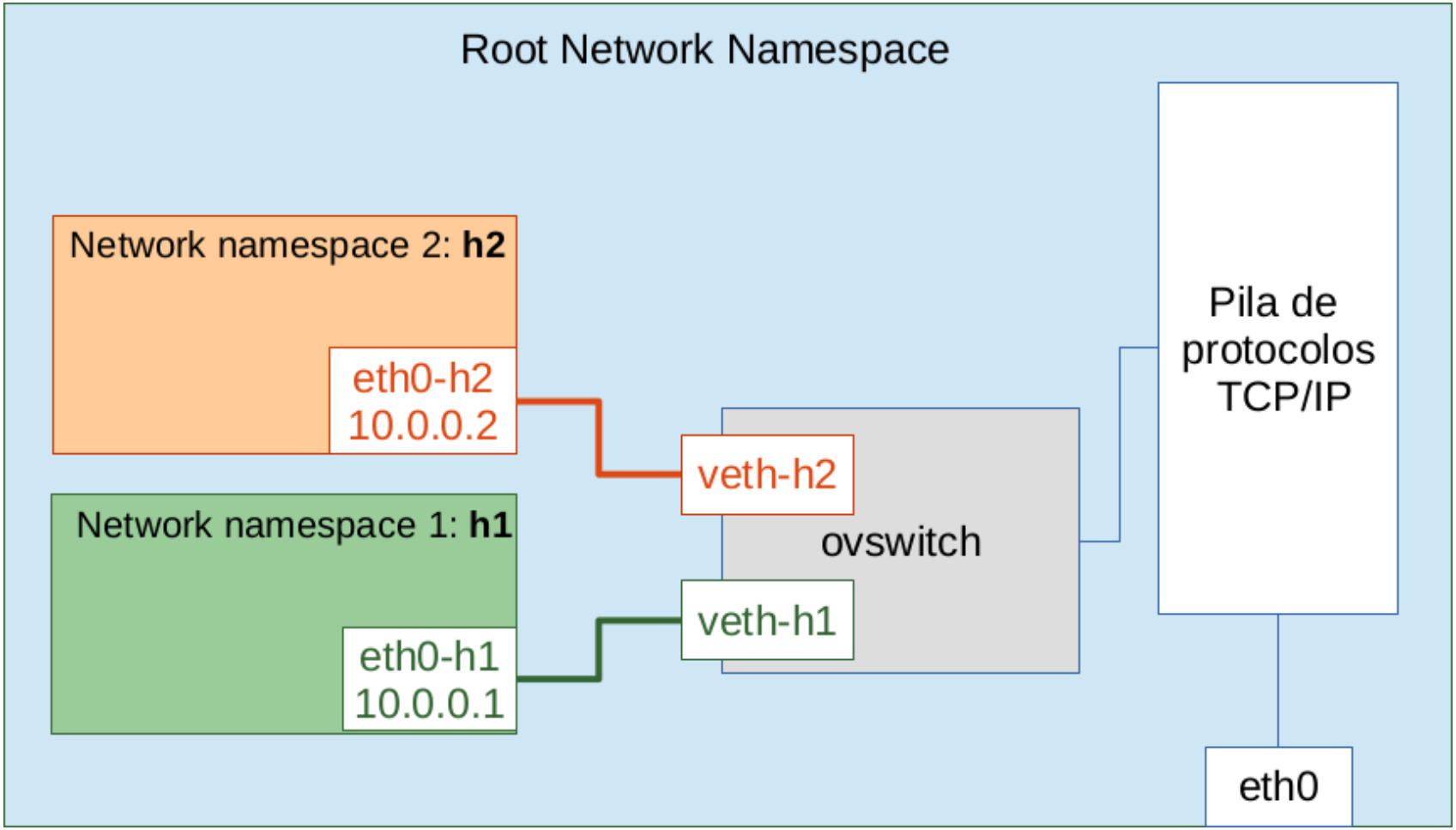
Configura para **eth0-h1** dentro de **h1** una dirección IP

Activa **eth0-h1** dentro de **h1**

Ejecuta un ping a la dirección IP de **eth0-h2** desde **h1**

Contenidos

Open vSwitch para conectar network namespaces



Crear network namespaces conectados a un switch (I)

- Se crean los espacios de nombres h1 y h2:

```
$ ip netns add h1
$ ip netns add h2
```

- Crear el switch software ovswitch

```
$ ovs-vsctl add-br ovs1
```

- Muestra la información del switch

```
$ ovs-vsctl show
6ec74893-3707-4764-bc79-efdd477aa90c
    Bridge "ovs1"
        Port "ovs1"
            Interface "ovs1"
                type: internal
    ovs_version: "2.5.2"
```

Crear network namespaces conectados a un switch (II)

- Crear enlaces

```
$ ip link add eth0-h1 type veth peer name veth-h1
$ ip link add eth0-h2 type veth peer name veth-h2
```

- Mostrar enlaces

```
$ ip link show
5: ovs-system: ...
      link/ether d6:6b:df:1e:c2:43 brd ff:ff:ff:ff:ff:ff
36: ovs1: ...  state DOWN ...
      link/ether 3e:d7:d8:47:52:4e brd ff:ff:ff:ff:ff:ff
39: veth-h1@eth0-h1:  ...  state DOWN ...
      link/ether 92:46:d6:55:69:c4 brd ff:ff:ff:ff:ff:ff
40: eth0-h1@veth-h1:  ...  state DOWN ...
      link/ether c6:2a:64:35:47:67 brd ff:ff:ff:ff:ff:ff
41: veth-h2@eth0-h2:  ...  state DOWN ...
      link/ether 2e:1a:a6:68:16:f6 brd ff:ff:ff:ff:ff:ff
42: eth0-h2@veth-h2:  ...  state DOWN ...
      link/ether 6e:a7:b7:31:d4:1f brd ff:ff:ff:ff:ff:ff
```

- Se han creado las parejas:
  - veth-h1@eth0-h1 (if39) con eth0-h1@veth-h1 (if40)
  - veth-h2@eth0-h2 (if41) con eth0-h1@veth-h1 (if42)

Crear network namespaces conectados a un switch (II)

- Mover las interfaces de máquinas a los namespaces:

```
$ ip link set eth0-h1 netns h1
$ ip link set eth0-h2 netns h2
```

- Las interfaces eth0-h1 y eth0-h2 han desparecido del root namespace. La interfaz veth-h1 y veth-h2 queda asociada al identificador de interfaz que tenían las interfaces cuando fueron creadas:
  - veth-h1@if40: donde if40 se corresponde con eth0-h1 que se encuentra en el identificador de namespace de h1 link-netsid=0)
  - veth-h2@if42: donde if42 se corresponde con eth0-h2 que se encuentra en el identificador de namespace de h2 link-netsid=1)
- Información del root network namespace:

```
$ ip link show
5: ovs-system: ...
      link/ether d6:6b:df:1e:c2:43 brd ff:ff:ff:ff:ff:ff
36: ovs1: ...  state DOWN ...
      link/ether 3e:d7:d8:47:52:4e brd ff:ff:ff:ff:ff:ff
39: veth-h1@if40:  ...  state DOWN ...
      link/ether 92:46:d6:55:69:c4 brd ff:ff:ff:ff:ff:ff link-netsid 0
41: veth-h2@if42:  ...  state DOWN ...
      link/ether 2e:1a:a6:68:16:f6 brd ff:ff:ff:ff:ff:ff link-netsid 1
```

Mostrar la información de los namespaces

- Ver la lista de los namespaces:

```
$ ip netns list
h2 (id: 1)
h1 (id: 0)
```

- Ver las interfaces dentro de los namespaces h1 y h2:

```
$ ip netns exec h1 ip link show
1: lo: ...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
40: eth0-h1@if39: ...  state DOWN ...
    link/ether c6:2a:64:35:47:67 brd ff:ff:ff:ff:ff:ff

$ ip netns exec h2 ip link show
1: lo: ...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
42: eth0-h2@if41: ...  state DOWN ...
    link/ether 6e:a7:b7:31:d4:1f brd ff:ff:ff:ff:ff:ff
```

Conectar los extremos veth al switch

- Conectar los extremos de los veth al switch:

```
$ ovs-vsctl add-port ovs1 veth-h1
$ ovs-vsctl add-port ovs1 veth-h2
```

- Mostrar la configuración del switch

```
$ ovs-vsctl show
6ec74893-3707-4764-bc79-efdd477aa90c
    Bridge "ovs1"
        Port "veth-h1"
            Interface "veth-h1"
        Port "veth-h2"
            Interface "veth-h2"
        Port "ovs1"
            Interface "ovs1"
                type: internal
    ovs_version: "2.5.2"
```

Configurar las direcciones IP en cada namespace

- Es necesario activar las interfaces para el namespace h1:

```
$ ip link set dev veth-h1 up
$ ip netns exec h1 ip link set dev lo up
$ ip netns exec h1 ip link set dev eth0-h1 up
$ ip netns exec h1 ip address add 10.0.0.1/24 dev eth0-h1
```

- Es necesario activar las interfaces para el namespace h2:

```
$ ip link set dev veth-h2 up
$ ip netns exec h2 ip link set dev lo up
$ ip netns exec h2 ip link set dev eth0-h2 up
$ ip netns exec h2 ip address add 10.0.0.2/24 dev eth0-h2
```

Comprobar conectividad entre h1 y h2

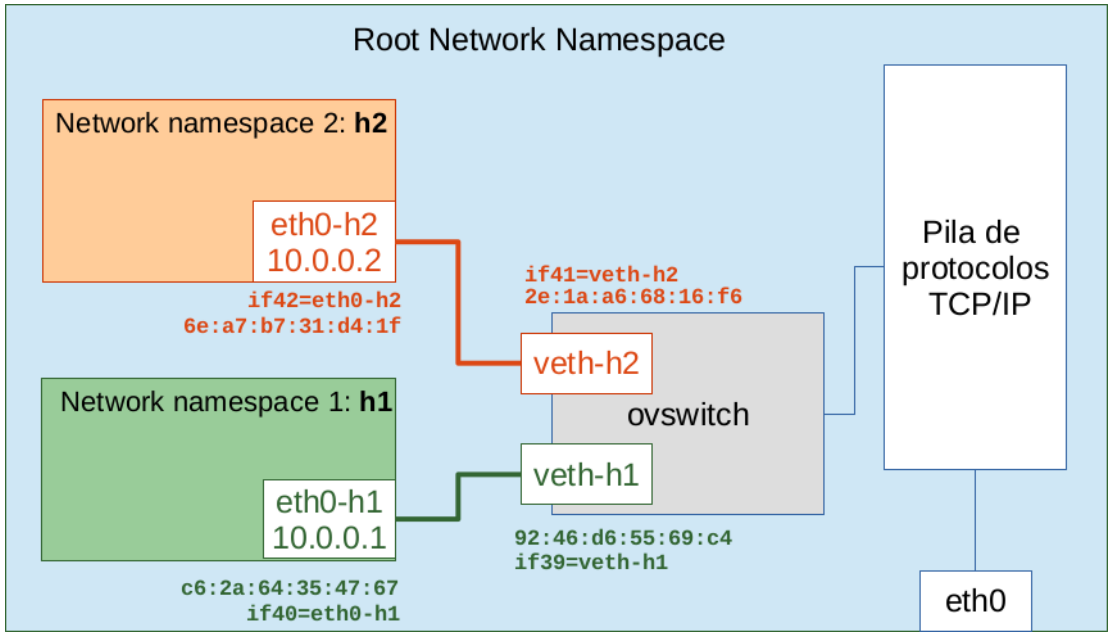
- El switch `ovs1` tiene el **comportamiento por defecto de un switch**: aprendizaje y reenvío.
- Desde namespace h2 se realiza ping a h1:

```
$ ip netns exec h2 ping 10.0.0.1
```

- Se observa la tabla de direcciones aprendidas en el switch:

```
$ ovs-appctl fdb/show ovs1
port    VLAN    MAC                 Age
1       0       c6:2a:64:35:47:67    1
2       0       6e:a7:b7:31:d4:1f    1
```

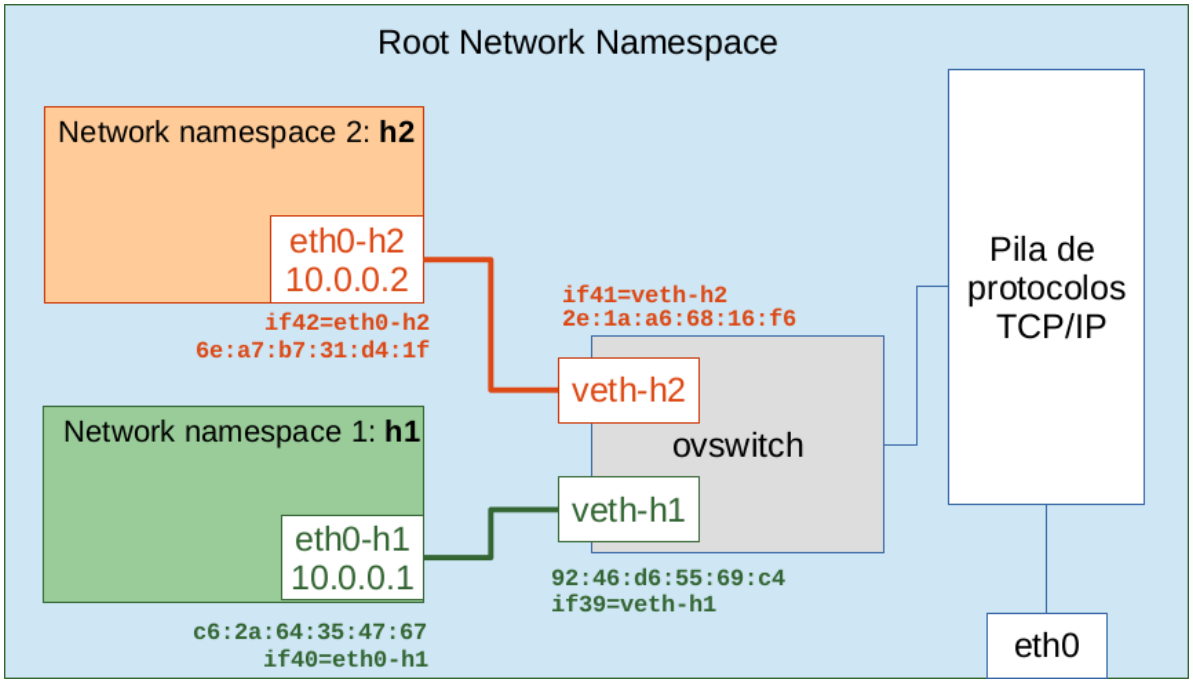
- El switch ha aprendido las direcciones Ethernet de las interfaces eth0-h1 (c6:2a:64:35:47:67) del namespace h1 y eth0-h2 (6e:a7:b7:31:d4:1f) del namespace h2.



Borrar tabla de direcciones aprendidas en open vSwitch

- Para borrar la tabla de direcciones aprendidas en el switch:

```
$ ovs-appctl fdb/flush ovs1
table successfully flushed
$ ovs-appctl fdb/show ovs1
port   VLAN   MAC                Age
```



Contenidos

Mininet

- En vez de arrancar manualmente los network namespaces, podemos usar mininet, un emulador de red que permite arrancar diferentes topologías que involucren gran cantidad de máquinas virtuales.
- Ejemplo:

```
$ sudo mn --topo=single,3 --controller=none --mac
```

  - –topo=single,3: topología de 3 nodos conectados a un switch.
  - –controller=none: no hay controlador para gestionar el comportamiento del switch, por tanto es necesario configurar el **comportamiento del switch de forma manual**.
  - –mac: asignación de direcciones MAC sencillas a las máquinas

Command Line Interface en mininet

- Mininet ofrece una interfaz de línea de comandos (CLI, Command Line Interface) que permite ejecutar comandos útiles de mininet y el arranque de xterm asociados a cada una de las máquinas para poder ejecutar comandos en las mismas, el prompt es mininet>.

```
$ sudo mn --topo=single,3 --controller=none --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

- Se pueden arrancar xterms asociados a las máquinas, por ejemplo, para lanzar un terminal de h1 en una ventana diferente:

```
mininet> xterm h1
```

Interfaces veth y open vSwitch en mininet

- Se puede ver el switch y sus interfaces que se encuentran en el root network namespace:

```
$ ip link show
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state
    DOWN mode DEFAULT group default qlen 1000
    link/ether 98:e7:f4:ef:21:d9 brd ff:ff:ff:ff:ff:ff

22: s1-eth1@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
    ovs-system state UP mode DEFAULT group default qlen 1000
    link/ether 36:c7:28:4d:e0:fa brd ff:ff:ff:ff:ff:ff link-netnsid 0

23: s1-eth2@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
    ovs-system state UP mode DEFAULT group default qlen 1000
    link/ether 06:9b:8b:83:4a:d5 brd ff:ff:ff:ff:ff:ff link-netnsid 1

24: s1-eth3@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
    ovs-system state UP mode DEFAULT group default qlen 1000
    link/ether 42:b9:c0:90:f6:75 brd ff:ff:ff:ff:ff:ff link-netnsid 2

25: s1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
    default qlen 1000
    link/ether aa:9b:8a:f8:83:4d brd ff:ff:ff:ff:ff:ff
```

- Mininet crea los network namespaces asociados a las máquinas virtuales **sin nombre**, con un identificador que solo tiene sentido dentro del namespace actual. Podemos ver que se han creado con los identificadores 0, 1 y 2 dentro del root network namespace:

```
$ ip netns list-id
nsid 0
nsid 1
nsid 2
```

Network namespaces sin nombre en mininet

- Desde Mininet se pueden ver los procesos creados para cada una de las máquinas y sus interfaces:

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=7826>
<Host h2: h2-eth0:10.0.0.2 pid=7828>
<Host h3: h3-eth0:10.0.0.3 pid=7830>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=7835>
```

- Cada máquina de mininet es un proceso bash dentro de un network namespace donde se definen sus interfaces de red:

```
$ ps -fp 7826,7828,7830,7835
UID          PID  PPID  C STIME TTY          TIME CMD
root        7826  7820  0 11:44 pts/20   00:00:00 bash --norc -is mininet:h1
root        7828  7820  0 11:44 pts/21   00:00:00 bash --norc -is mininet:h2
root        7830  7820  0 11:44 pts/22   00:00:00 bash --norc -is mininet:h3
root        7835  7820  0 11:44 pts/23   00:00:00 bash --norc -is mininet:s1
```

- El contenido del namespace creado se puede consultar sabiendo el número de proceso, por ejemplo para h1, su pid=7826:

```
$ nsenter -t 7826 -n ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: h1-eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DE
    group default qlen 1000
    link/ether 00:00:00:00:00:01 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Información de la red en mininet

- La descripción de la red arrancada en mininet se puede ver:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
```

- Desde CLI se pueden ejecutar comandos en cada una de las máquinas, anteponiendo el nombre de la máquina al comando que se desea ejecutar:

```
mininet> h1 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: h1-eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
    group default qlen 1000
    link/ether 00:00:00:00:00:01 brd ff:ff:ff:ff:ff:ff link-netnsid 0

mininet> h2 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: h2-eth0@if23: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
    group default qlen 1000
    link/ether 00:00:00:00:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0

mininet> h3 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: h3-eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
    group default qlen 1000
    link/ether 00:00:00:00:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Información del switch en mininet

- La configuración del switch creado por mininet muestra que el switch tiene 3 puertos s1-eth1, s1-eth2 y s1-eth3 y espera comunicarse con el controlador de openflow en el puerto 6654 (passive listening tcp):

```
$ ovs-vsctl show
8b7e745d-6b6f-448d-a0e3-8ae2f0adc5b6
    Bridge "s1"
        Controller "ptcp:6654"
        fail_mode: secure
        Port "s1-eth3"
            Interface "s1-eth3"
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
    ovs_version: "2.5.2"
```

- En el modo de funcionamiento `fail_mode=secure` sólo el controlador puede tomar decisiones de cómo reenviar los paquetes, si falla la comunicación con el controlador, todos los paquetes se tiran.

Contenidos

Arquitectura de Open vSwitch

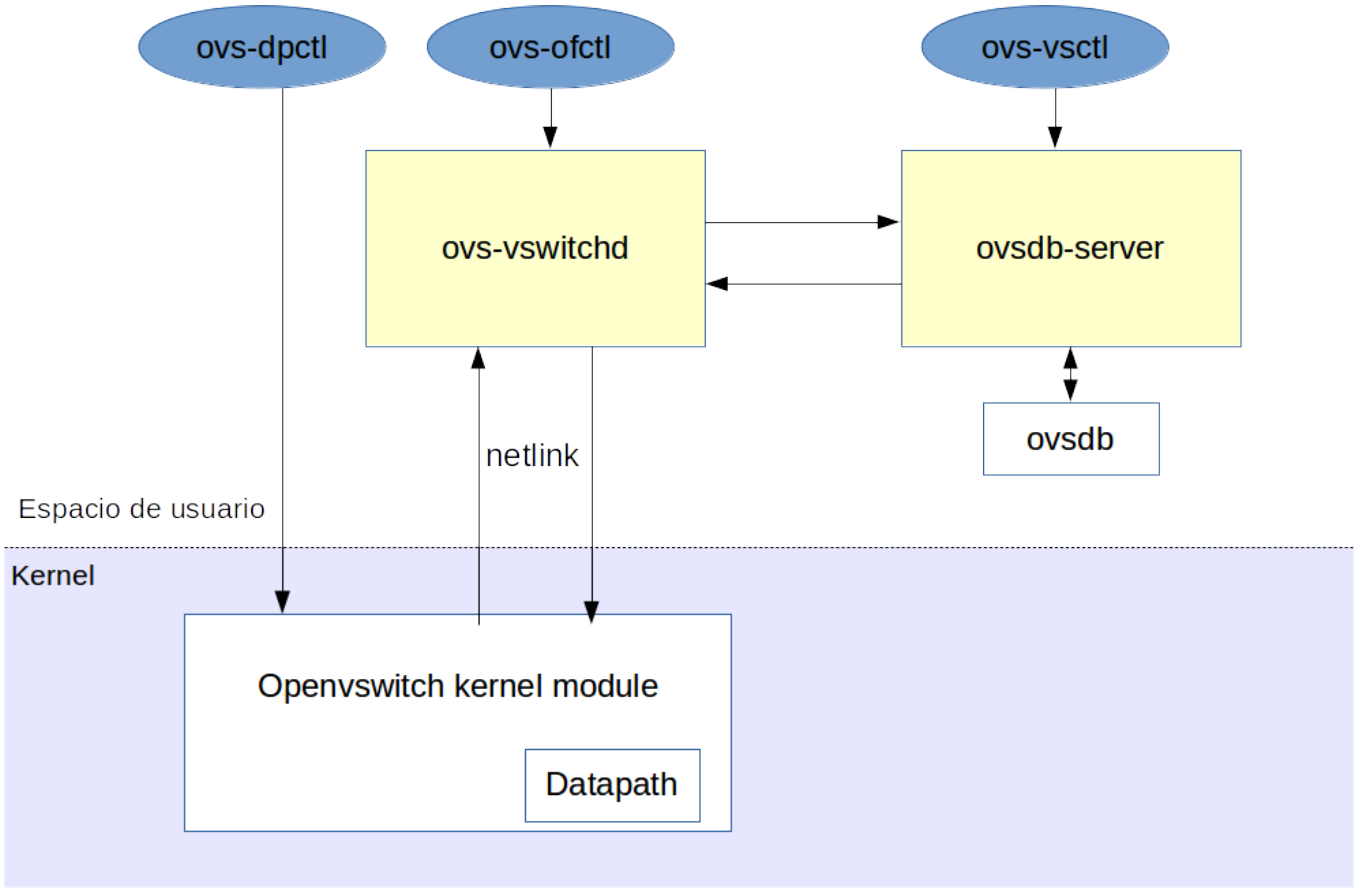
- Los componentes fundamentales de Open vswitch son:
  - OVS kernel module: implementa **datapaths** (similar a un switch) cada uno de los cuáles puede tener varios **vports** (similar a los puertos del switch). Un datapath tiene asociada una tabla de flujos que definen el comportamiento que ha de seguir un paquete en función de sus campos. Si no encuentra una regla en la tabla datapath que aplique al paquete, deberá pasarlo al ovs-vswitchd.
  - ovs-vswitchd: componente central que se ejecuta en espacio de usuario. Procesa los mensajes openFlow y configurará tablas con dicha información, gestiona datapath.
  - ovsdb-server: la configuración del switch es persistente y se guarda en la base de datos ovsdb. Mantiene 16 tablas: Bridge, Queue, QoS, Port, sFlow, Interface, etc.

Flujos en la arquitectura de Open vSwitch

- Open vSwitch maneja 2 tipos de flujos:
  - **Flujos OpenFlow**: en el espacio de usuario dentro de ovs-vswitchd.
  - **Flujos Datapath**: son flujos instalados desde el espacio de usario en el kernel. Forman una "caché" de los flujos OpenFlow dentro de OVS kernel. Colapsan en la definición de un megaflujo todas las acciones que se le aplican a un paquete. El megaflujo se construye a partir de los flujos openFlow.

Comandos para la gestión de Open vSwitch

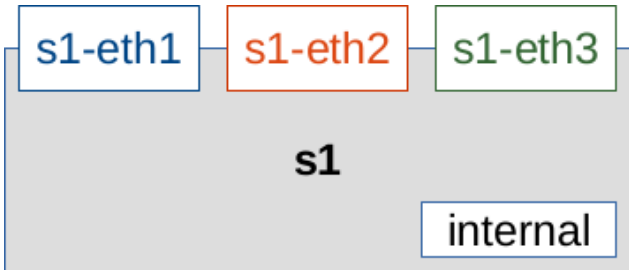
- **ovs-dpctl**: gestiona datapath
- **ovs-ofctl**: gestiona flujos openFlow.
- **ovs-vsctl**: gestiona la DB ovsdb y los switches open vSwitch (también existe la herramienta ovsdb-client).



ovs-vsctl

Información de un switch que mantiene la base de datos: ovsdb

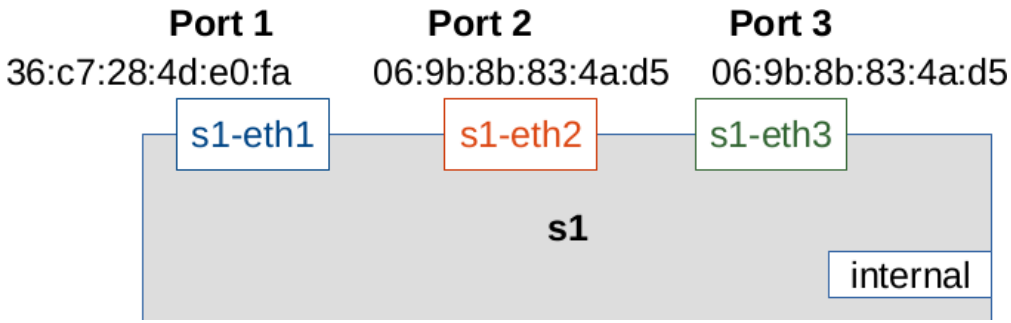
```
$ ovs-vsctl show
8b7e745d-6b6f-448d-a0e3-8ae2f0adc5b6
    Bridge "s1"
        Controller "ptcp:6654"
        fail_mode: secure
        Port "s1-eth3"
            Interface "s1-eth3"
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
    ovs_version: "2.5.2"
```



ovs-ofctl

Información de ovs-switchd consultada a través del protocolo openFlow

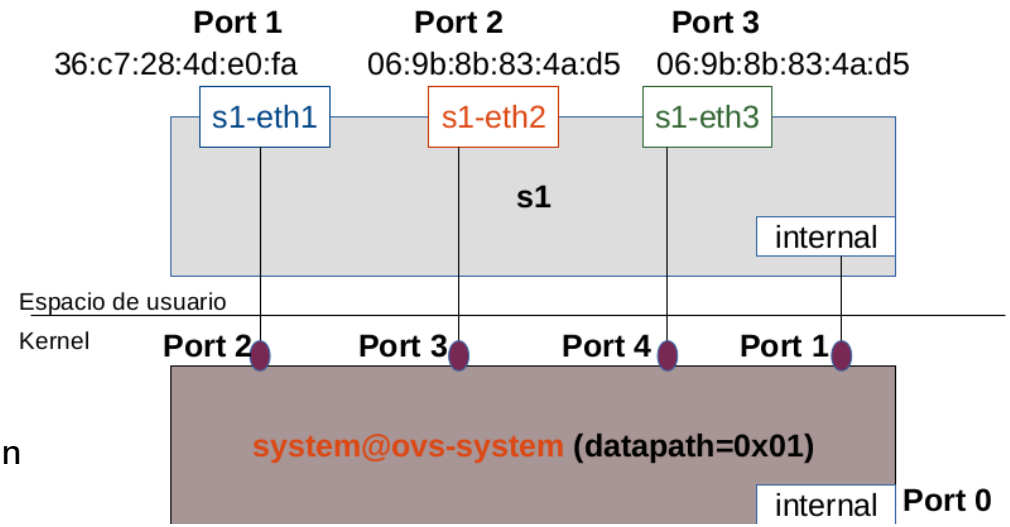
```
$ ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(s1-eth1): addr:36:c7:28:4d:e0:fa
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:06:9b:8b:83:4a:d5
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:42:b9:c0:90:f6:75
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:aa:9b:8a:f8:83:4d
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```



ovs-dpctl

Información de Datapath

```
$ ovs-dpctl show
system@ovs-system:
lookups: hit:18 missed:36 lost:0
flows: 0
masks: hit:32 total:0 hit/pkt:0.59
port 0: ovs-system (internal)
port 1: s1 (internal)
port 2: s1-eth1
port 3: s1-eth2
port 4: s1-eth3
```



- `Hit`: paquetes que han encontrado regla en datapath
- `Missed`: paquetes que se han enviado al espacio de usuario porque no había regla en el datapath
- `Lost`: paquetes que se han descartado antes de alcanzar espacio de usuario.

Contenidos

Mininet sin controlador

- Al arrancar mininet sin controlador, el comportamiento del switch tendrá que configurarse manualmente.
- Si ejecutamos `pingall` dentro del entorno de mininet se enviará un ping desde cada una de las máquinas al resto de las máquinas. En nuestro caso hay 3 hosts, por tanto, desde h1 se enviará ping hacia h2 y h3 y de forma similar se realizará desde el resto. Se muestra "X" que indica que no se puede alcanzar.

```
$ sudo mn --topo=single,3 --controller=none --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:

mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Configuración manual de flujos openFlow (I)

- Se puede configurar un comportamiento de switch en `s1` utilizando el comando `ovs-ofctl` que envía mensajes OpenFlow a `ovs-vswitchd`, simulando un controlador externo.
  - Mininet permite ejecutar comandos en la máquina real, si se antepone el comando `sh` antes del comando real a ejecutar:

    ```
    mininet> sh ovs-ofctl add-flow s1 action=normal
    ```
  - Esto es lo mismo que si desde un terminal de la máquina se ejecuta:

    ```
    $ sudo ovs-ofctl add-flow s1 action=normal
    ```

- Al haber instruido a s1 para que se comporte como switch, se puede observar que hay conectividad entre las máquinas:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

- El switch s1 tiene el funcionamiento normal de un switch: aprendizaje y reenvío:

```
mininet> sh ovs-appctl fdb/show s1
 port  VLAN  MAC                Age
    1     0  00:00:00:00:00:01    2
    2     0  00:00:00:00:00:02    2
    3     0  00:00:00:00:00:03    2
```

Configuración manual de flujos openFlow(II)

- Si se observan los flujos **openFlow** en la tabla de ovs-switchd:

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4);
 cookie=0x0, duration=143.932s, table=0, n_packets=25, n_bytes=1750,
 idle_age=123, actions=NORMAL
```

  - Duration: cuánto tiempo lleva este flujo instalado en la tabla
  - Idle_age: cuánto tiempo hace que un paquete ha satisfecho esta regla

- Si se observan los flujos en **datapath** en el espacio de kernel:

```
mininet> sh ovs-dpctl dump-flows
recirc_id(0),in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:03),
eth_type(0x0800),ipv4(frag=no), packets:1, bytes:98, used:1.952s, actions:3
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:02),
eth_type(0x0800),ipv4(frag=no), packets:1, bytes:98, used:1.954s, actions:1
recirc_id(0),in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:01),
eth_type(0x0800),ipv4(frag=no), packets:1, bytes:98, used:1.963s, actions:2
recirc_id(0),in_port(1),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:03),
eth_type(0x0800),ipv4(frag=no), packets:1, bytes:98, used:1.955s, actions:3
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:01),
eth_type(0x0800),ipv4(frag=no), packets:1, bytes:98, used:1.955s, actions:2
recirc_id(0),in_port(2),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),
eth_type(0x0800),ipv4(frag=no), packets:1, bytes:98, used:1.963s, actions:1
```

Configuración manual de flujos openFlow (III)

- Vamos a eliminar el comportamiento "normal" del switch s1. Para borrar todos los flujos en un switch:

```
mininet> sh ovs-ofctl del-flows s1
```

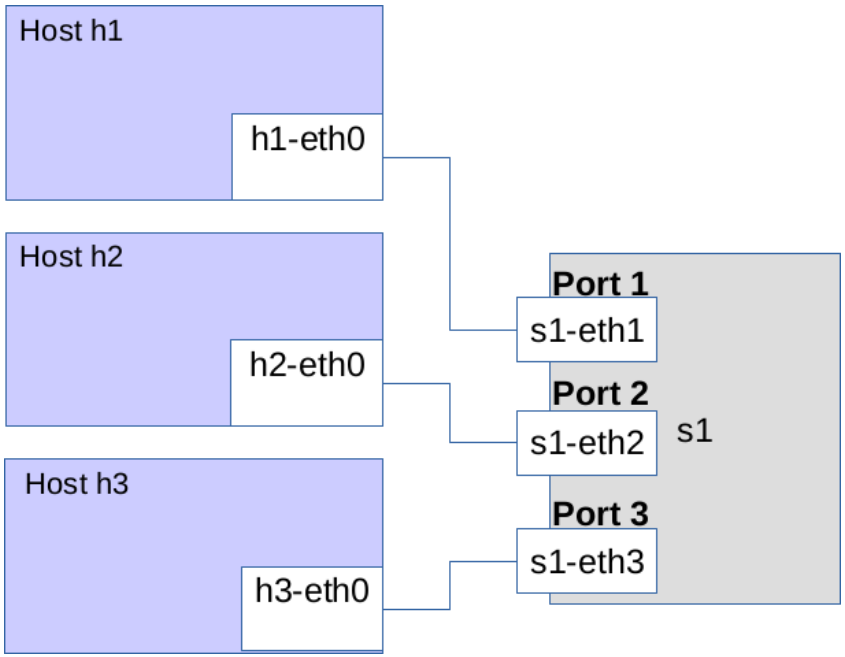
- Se puede comprobar que se han borrado:

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply: *none*
mininet> pingall
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Contenidos

Definición de flujos en el NIVEL FÍSICO (I)

- Partimos de un switch al que hemos borrado su configuración "normal" por tanto, cuando reciba una trama Ethernet por un determinado puerto no va a saber qué hacer y la descartará. **El switch ya no va a aprender**.
- **Para la configuración de flujos a nivel físico es necesario saber en qué puerto del switch está conectada cada máquina**.
- A partir de la información que ofrece mininet con el comando `net` y de la información del switch que devuelve ovs-switchd podemos dibujar la configuración de red tal y como se ve desde el espacio de usuario:



Configuración: flujos en el NIVEL FÍSICO (II)

- Reenvío de puertos del switch entre 1 y 2:

```
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:2
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1
```

- Los flujos openFlow instalados son:

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=27.605s, table=0, n_packets=0, n_bytes=0,
  idle_age=27, priority=500, in_port=1 actions=output:2
  cookie=0x0, duration=15.164s, table=0, n_packets=0, n_bytes=0,
  idle_age=15, priority=500, in_port=2 actions=output:1
```

- Se puede comprobar la conectividad, debido a los flujos que se aplican al tráfico definido no al aprendizaje del switch:

```
mininet> h1 ping -c2 h2
2 packets transmitted, 2 received, 0% packet loss
mininet> h3 ping -c2 h2
2 packets transmitted, 0 received, 100% packet loss
```

- Ahora no hay aprendizaje en switch, pues no se está comportando en modo `normal`, su comportamiento se rige por las reglas openFlow:

```
mininet> sh ovs-appctl fdb/show s1
 port  VLAN  MAC                Age
```

Configuración: flujos en el NIVEL FÍSICO (III)

- Con la configuración de flujos openFlow:

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=27.605s, table=0, n_packets=0, n_bytes=0,
  idle_age=27, priority=500, in_port=1 actions=output:2
  cookie=0x0, duration=15.164s, table=0, n_packets=0, n_bytes=0,
  idle_age=15, priority=500, in_port=2 actions=output:1
```

- Se puede usar el siguiente comando para trazar el camino de un paquete a través del switch:

```
mininet> sh ovs-appctl ofproto/trace s1 in_port=1
Bridge: s1
Flow: in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:00,
      dl_dst=00:00:00:00:00:00,dl_type=0x0000

Rule: table=0 cookie=0 priority=500,in_port=1
OpenFlow actions=output:2

Final flow: unchanged
Megaflow: recirc_id=0,in_port=1,dl_type=0x0000
Datapath actions: 1
```

Configuración: flujos en el NIVEL FÍSICO (IV)

- El resultado de la ejecución `ovs-appctl ofproto/trace` anterior muestra 3 partes:
  - Información del flujo a probar y el switch.
    - **Bridge**: nombre del switch, s1
    - **Flow**: características del flujo a probar, en este caso, recibido en puerto 1 (los campos que no se especifican se quedan a cero)
  - Reglas openFlow que aplican
    - **Rule**: regla que se le aplica, en la tabla 0 la regla de de reenviar por el puerto 2 en openFlow para los paquetes recibidos en puerto 1.
  - Flujo del datapath
    - **Final flow**: el flujo resultante no se modifica
    - **Megaflow**: descripción del flujo dentro del datapath. Esta información se instala en el datapath para consultar rápidamente si se puede aplicar el flujo a los paquetes entrantes.(recirc_id=0 significa que el paquete acaba de entrar en el datapath, no ha habido recirculación).
    - **Datapath actions**: acción tomada reenviar por el puerto 1 en el espacio de kernel

Configuración: flujos en el NIVEL FÍSICO (V)

- Los flujos dentro de una tabla openFlow se consultan desde mayor prioridad a menor, cuando se cumple la condición de un flujo, se ejecuta la acción y no se comprueban más flujos de esa tabla.
- La prioridad por defecto es 32768 (cuanto mayor número, más alta es la prioridad)

```
mininet> sh ovs-ofctl add-flow s1 priority=32768,in_port=1,actions=drop
mininet> h1 ping -c2 h2
  2 packets transmitted, 2 received, 100% packet loss
```

- En la visualización de los flujos la prioridad por defecto no se muestra:

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=297.245s, table=0, n_packets=3, n_bytes=126, idle_age=161,
      priority=500,in_port=1 actions=output:2
  cookie=0x0, duration=276.009s, table=0, n_packets=0, n_bytes=0, idle_age=276,
      priority=500,in_port=2 actions=output:1
  cookie=0x0, duration=171.363s, table=0, n_packets=3, n_bytes=126, idle_age=161,
      in_port=1 actions=drop
```

- Los flujos se pueden instalar asociados a un valor identificador o `cookie` para gestionarlos conjuntamente. Por ejemplo, se pueden borrar todos los flujos que tengan la misma `cookie`. Por defecto, `cookie=0`.

Configuración: flujos en el NIVEL FÍSICO (VI)

- Borrar un flujo, el que estrictamente cumple la condición esa y sólo esa (`--strict`):

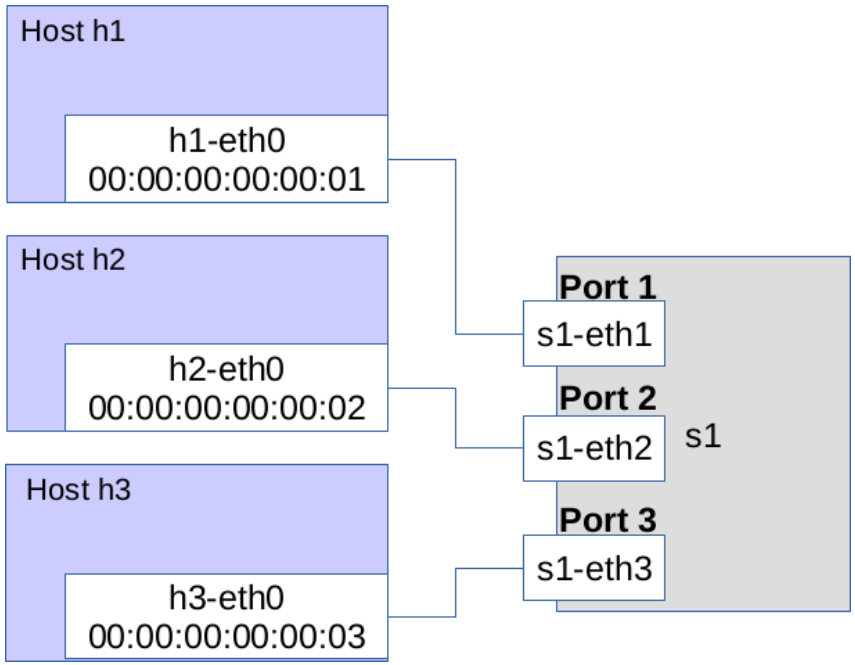
```
mininet> sh ovs-ofctl del-flows s1 --strict in_port=1
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=297.245s, table=0, n_packets=3, n_bytes=126, idle_age=161,
  priority=500,in_port=1 actions=output:2
  cookie=0x0, duration=276.009s, table=0, n_packets=0, n_bytes=0, idle_age=276,
  priority=500,in_port=2 actions=output:1
```

- Vuelve a haber conectividad entre h1 y h2.

Contenidos

Configuración: flujos en el NIVEL DE ENLACE (I)

- El reenvío en el nivel de enlace **se realiza a partir de la información de la cabecera Ethernet**.
- A partir de la configuración de las direcciones Ethernet de las interfaces de las máquinas se puede saber la siguiente información:



Configuración: flujos en el NIVEL DE ENLACE (II)

- Partiendo de la tabla del switch vacía se instalan los siguientes flujos para el reenvío de tramas Ethernet entre h1 y h2:

```
mininet> sh sh ovs-ofctl add-flow s1
 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:2
mininet> sh ovs-ofctl add-flow s1
 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:1
```

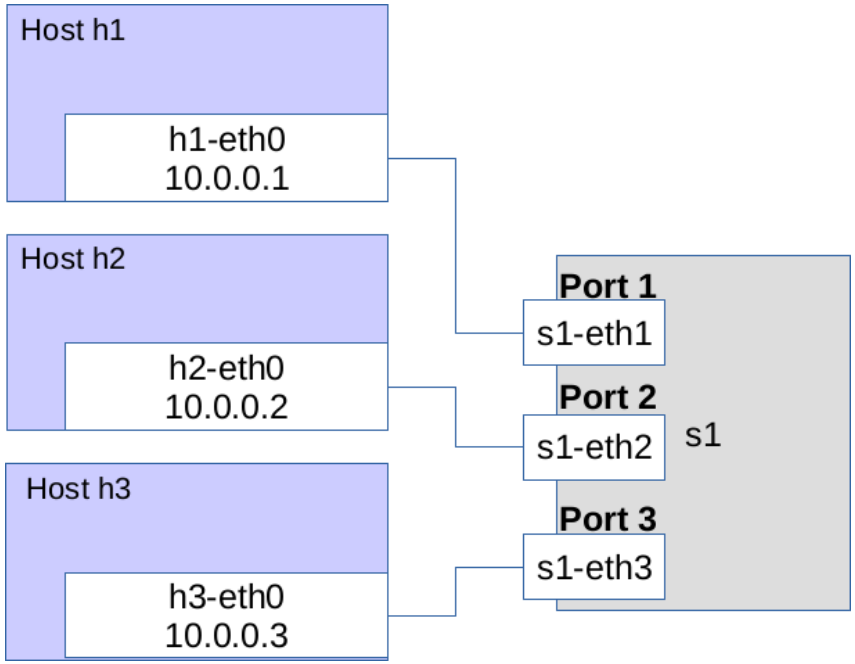
- Para que un ping entre estas máquinas funcione además hay que instalar la regla que permite reenviar la solicitud de ARP. Con la acción FLOOD se envía por todos los puertos salvo por donde se recibe. El parámetro `nw_proto=1` hace referencia a la operación ARP, el valor 1 es solicitud y 2 es respuesta.

```
mininet> sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,actions=flood

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)

mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=543.873s, table=0, n_packets=6, n_bytes=420, idle_age=79,
 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=534.485s, table=0, n_packets=7, n_bytes=462, idle_age=79,
 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=202.094s, table=0, n_packets=13, n_bytes=546, idle_age=61, arp,
 arp_op=1 actions=FLOOD
```

Contenidos

Configuración: flujos en el NIVEL DE RED (I)

- El reenvío en el nivel de red se realiza a partir de la cabecera IP.
- A partir de la configuración de las direcciones IP de las interfaces de las máquinas se puede saber la siguiente información:



Configuración: flujos en el NIVEL DE RED (II)

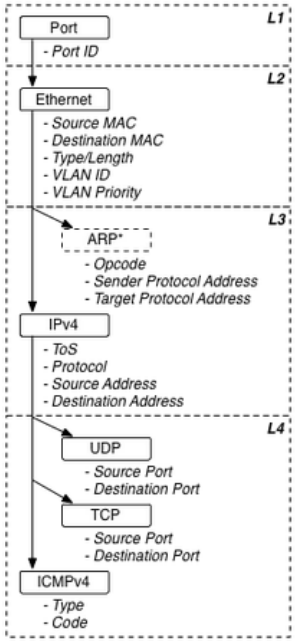
- Partiendo de la tabla del switch vacía se instalan los siguientes flujos para el reenvío de tramas Ethernet entre h1 y h2:

```
mininet> sh ovs-ofctl add-flow s1
 priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=normal
mininet> sh ovs-ofctl add-flow s1
 priority=800,ip,nw_src=10.0.0.3,actions=mod_nw_tos=184,normal
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.1,actions=output:1
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.2,actions=output:2
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.3,actions=output:3

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Contenidos

Campos a comprobar en openFlow 1.1

Classifier Dependencies



Classifier Fields

Protocol	Dependency	Name	Match Field
Port	none	Port ID	in_port
Ethernet	none	Source MAC	dl_src
		Destination MAC	dl_dst
		Type/Length	dl_type
		VLAN ID	dl_vlan
		VLAN Priority	dl_vlan_pcp
⚠ ARP	dl_type = 0x0806	Opcode	nw_proto
		Sender Protocol Address	nw_src
		Target Protocol Address	nw_dst
IPv4	dl_type = 0x0800	Type of Service	nw_tos
		Protocol	nw_proto
		Source Address	nw_src
		Destination Address	nw_dst
TCP	nw_proto = 6	Source Port	tp_src
		Destination Port	tp_dst
UDP	nw_proto = 17	Source Port	tp_src
		Destination Port	tp_dst
ICMPv4	nw_proto = 1	Type	tp_src
		Code	tp_dst

Acciones

- Si no se especifica acción los paquetes serán descartados.
- Además de las acciones básicas, enviar a un puerto, descartar, inundar o enviar al controlador, hay definidas acciones para la modificación de campos del paquete.
- La acción a configurar en un flujo puede ser una lista de acciones separados por comas: `actions=[action][,action...]`
- A continuación se presentan algunas de las más relevantes:

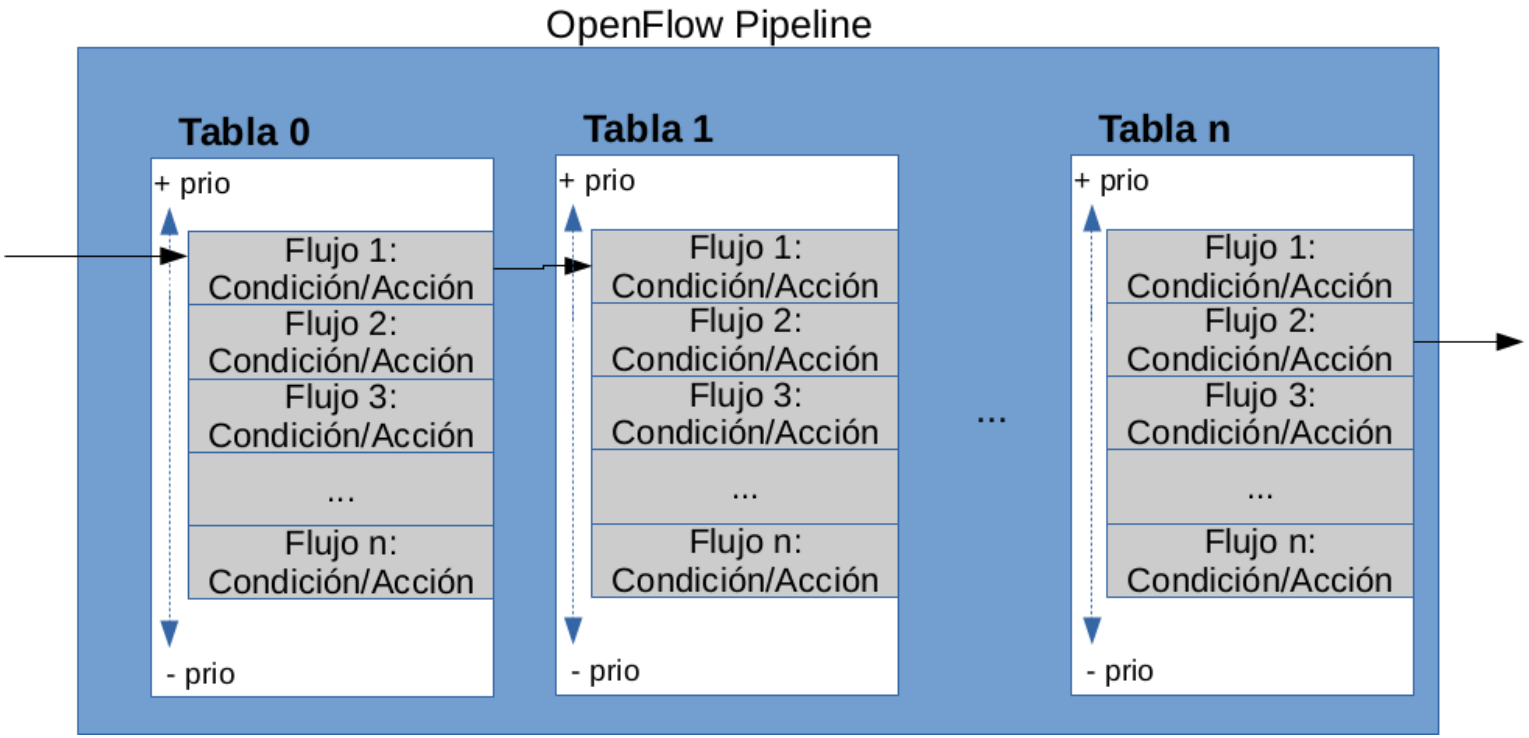
Sintaxis `actions=`	Acción
`output:<nPuerto>`	Reenvía el paquete al puerto `<nPuerto>`
`input`	Reenvía el paquete por el mismo puerto por el que se ha recibido
`all`	Reenvío por todos los puertos, salvo por el recibido
`resubmit(<nPuerto>,<nTabla>)`	Extensión de open vSwitch. Reenvía a la tabla número `<nTabla>`
como si se hubiera recibido por `<nPuerto>`	
`normal`	procesamiento normal de switch, con aprendizaje
`flood`	inundación, salvo en el puerto recibido o que esté desactivado por STP
`drop`	eliminar
`mod_vlan_vid:<vlan_vid>`	modifica VLAN id a `<vlan_vid>`
`mod_vlan_pcp:<vlan_pcp>`	modificar VLAN priority a `<vlan_pcp>`
`strip_vlan`	eliminar la cabecera VLAN, para reenviar por puerto sin etiqueta
`mod_dl_dst:<SrctEthernet>`	modificar dirección Ethernet origen a `<SrcEthernet>`
`mod_dl_dst:<DstEthernet>`	modificar dirección Ethernet destino a `<DstEthernet>`
`mod_nw_src:<SrcIP>`	modificar dirección IPv4 origen a `<SrcIP>`
`mod_nw_dst:<DstIP>`	modificar dirección IPv4 destino a `<DstIP>`
`mod_tp_tos:<ToS>`	modificar ToS de IPv4 a `<ToS>`
`mod_tp_src:<SrcPort>`	modificar puerto origen a `<SrcPort>`
`mod_tp_dst:<DstPort>`	modificar puerto destino a `<DstPort>`

Contenidos

Múltiples tablas

- En general se definen varias tablas dentro de un switch donde se instalan reglas con diferentes objetivos: ACLs (Access Control List), NAT (Network Address Translator), encaminamiento, VLANs (Virtual LAN), etc.
- Cuando se recibe un paquete comienza el procesamiento consultando los flujos de la tabla número 0.
- Un paquete abandona el procesamiento del pipeline cuando la acción es una de las siguientes:
  - Enviar a un puerto determinado (Ej: actions=output:1)
  - Descartar (actions=drop)
  - Inundar (actions=flood)
  - Enviar al controlador (actions=controller)
- Se pueden enviar paquetes del procesamiento de una tabla a otra, siempre que el número de tabla sea mayor del que proviene.

OpenFlow pipeline



Ejemplo de configuración de múltiples tablas con openFlow

- Se desea definir un comportamiento de un switch con las siguientes tablas:
  - Tabla 0: ACL
  - Tabla 1: Encaminamiento

Tabla 0: ACL

- Todos los paquetes cuya dirección Ethernet origen es de multicast (bit menos significativo del byte más significativo de la dirección Ethernet está a 1) se descartan. El resto de paquetes pasan a la siguiente tabla (tabla 1):

```
mininet> sh ovs-ofctl add-flow s1 "table=0, dl_src=01:00:00:00:00:00/01:00:00:00:00:00, actions=drop"
mininet> sh ovs-ofctl add-flow s1 "table=0, priority=0, actions=resubmit(,1)"
```

- Para probar esta tabla:

```
mininet> sh ovs-appctl ofproto/trace s1 in_port=1, dl_src=01:11:11:11:11:11
Bridge: s1
Flow: in_port=1,vlan_tci=0x0000,dl_src=01:11:11:11:11:11,dl_dst=00:00:00:00:00:00,dl_type=0x0000

Rule: table=0 cookie=0 dl_src=01:00:00:00:00:00/01:00:00:00:00:00
OpenFlow actions=drop

Final flow: unchanged
Megaflow: recirc_id=0,in_port=1,dl_src=01:00:00:00:00:00/01:00:00:00:00:00,dl_type=0x0000
Datapath actions: drop
```

  El resultado de la ejecución muestra:
  - **Bridge**: nombre del switch, s1
  - **Flow**: características del flujo a probar, en este caso, recibido en puerto 1 con dirección origen=01:11:11:11:11:11 (los campos que no se especifican s quedan a cero)
  - **Rule**: regla que se le aplica, en la tabla 0 la regla de descarte de tramas con dirección Ethernet con bit multicast a 1.
  - **Final flow**: el flujo resultante no se modifica
  - **Megaflow**: descripción del flujo dentro del datapath. Esta información se instala en el datapath para consultar rápidamente si se puede aplicar el flujo a los paquetes entrantes.(recirc_id=0 significa que el paquete acaba de entrar en el datapath, no ha habido recirculación).
  - **Datapath actions**: acción tomada es descartar.

Tabla 1: Encaminamiento

- Todos los paquetes cuya dirección IP destino sea 10.0.0.1 se modifica la dirección Ethernet destino a 00:00:00:00:00:01 y se envían por el puerto 2.

```
mininet> sh ovs-ofctl add-flow s1 "table=1, ip, nw_dst=10.0.0.1,
 actions=mod_dl_dst=00:00:00:00:00:01,output=2"
```

- Para probar esta tabla:

```
mininet> ovs-appctl ofproto/trace s1 in_port=1,ip,nw_dst=10.0.0.1
Bridge: ovs1
Flow: ip,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:00,dl_dst=00:00:00:00:00:00,
nw_src=0.0.0.0,nw_dst=10.0.0.1,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=0

Rule: table=0 cookie=0 priority=0
OpenFlow actions=resubmit(,1)

Resubmitted flow: ip,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:00,dl_dst=00:00:00:00:00:00,
nw_src=0.0.0.0,nw_dst=10.0.0.1,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=0
Resubmitted regs: reg0=0x0 reg1=0x0 reg2=0x0 reg3=0x0 reg4=0x0 reg5=0x0 reg6=0x0 reg7=0x0
Resubmitted  odp: drop
Resubmitted megaflow: recirc_id=0,ip,in_port=1,dl_src=00:00:00:00:00:00/01:00:00:00:00:00,
nw_dst=10.0.0.1,nw_frag=no
Rule: table=1 cookie=0 ip,nw_dst=10.0.0.1
OpenFlow actions=mod_dl_dst:00:00:00:00:00:01,output:2

Final flow: ip,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:00,dl_dst=00:00:00:00:00:01,
nw_src=0.0.0.0,nw_dst=10.0.0.1,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=0
Megaflow: recirc_id=0,ip,in_port=1,dl_src=00:00:00:00:00:00/01:00:00:00:00:00,
dl_dst=00:00:00:00:00:00,nw_dst=10.0.0.1,nw_frag=no
Datapath actions: set(eth(src=00:00:00:00:00:00/01:00:00:00:00:00,dst=00:00:00:00:00:01)),5
```

El resultado de la ejecución muestra:

- El paquete cumple las condiciones de 2 reglas, una en la tabla 0 y otra en la tabla 1. En la regla de la tabla 0 se reenvía el paquete a la tabla 1. En la regla de la tabla 1 se modifica la dirección Ethernet destino del paquete a 00:00:00:00:00:01 y se reenvía por el puerto 2.

Definición de flujos en un archivo

- Se puede cargar la configuración de los flujos (`add-flows`) desde un fichero donde se encuentran todas las reglas:

```
mininet> sh ovs-ofctl add-flows s1 flows.txt
```

- Donde el contenido del fichero flows.txt es el siguiente:

```
table=0, dl_src=01:00:00:00:00:00/01:00:00:00:00:00, actions=drop
table=0, priority=0, actions=resubmit(,1)
table=1, nw_dst=10.0.0.1, actions=mod_dl_dst=00:00:00:00:00:01,output=2
```

- Para ver los flujos configurados:

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=382.254s, table=0, n_packets=0, n_bytes=0, idle_age=382,
  dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop
 cookie=0x0, duration=382.253s, table=0, n_packets=0, n_bytes=0, idle_age=382,
  priority=0 actions=resubmit(,1)
 cookie=0x0, duration=382.252s, table=1, n_packets=0, n_bytes=0, idle_age=382,
  actions=mod_dl_dst:00:00:00:00:00:01,output:2
```