RETO 1: Verificación del comportamiento de un hub.

En esta práctica no vamos a utilizar el controlador de referencia, el cual es el controlador por defecto que Mininet utiliza durante su simulación. Por lo tanto, hemos de asegurarnos de que no se está ejecutando:

```
$ ps -A | grep controller
```

Si resulta que sí se está ejecutando, hemos de matar el proceso, bien pulsando Ctrl-C en la ventana donde se está ejecutando el programa controlador o desde una ventana ssh ejecutando:

```
$ sudo killall controller
```

Asimismo, también deberíamos ejecutar sudo mn –c y reiniciar Mininet para asegurarnos que el entorno esté limpio. Desde la consola de Mininet:

```
$ sudo mn -c
```

A continuación, abrimos otra terminal y ejecutamos el ejemplo de hub básico:

```
$ cd ~/pox
$ ./pox.py log.level --DEBUG forwarding.hub
```

Finalmente, ejecutamos mininet

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

Este comando le indica a POX que se ejecute mostrando los informes de errores y que inicie el componente hub.

Los switches pueden necesitar algo de tiempo para conectarse. Cuando un switch OpenFlow pierde su conexión con el controlador, generalmente incrementará el espacio de tiempo con el que intenta contactar con el controlador, hasta un máximo de 15 segundos. Como el switch OpenFlow aún no se ha conectado, este retardo puede tomar cualquier valor entre 0 y 15 segundos.

Si se considera que es un espacio de tiempo demasiado elevado para esperar, puede configurarse el switch para que no espere más de N segundos, utilizando el parámetro –max-backoff.

Otra alternativa es salir de Mininet, eliminar los switches, iniciar el controlador, y después iniciar Mininet para que se conecte inmediatamente.

Una vez que la aplicación nos indica que el switch OpenFlow se ha conectado, el controlador POX nos mostrará un mensaje de este tipo:

```
INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
DEBUG:samples.of_tutorial:Controlling [Con 1/1]
```

A continuación, verificamos que todos los hosts pueden enviarse un ping entre sí, y que todos los hosts ven exactamente el mismo tráfico. Es decir, que el hub realmente se comporta como un hub. Para ello, crearemos xterms para cada host y visualizaremos el tráfico en cada uno.

En la consola de Mininet, iniciar tres xterms:

```
mininet> xterm h1 h2 h3
```

Organiza cada xterm para que se puedan ver todas a la vez en la pantalla. Esto puede requerir reducir su tamaño.

En las xterms para h2 y h3, ejecuta tcpdump, una utilidad para imprimir los paquetes que ve un host:

```
# tcpdump -XX -n -i h2-eth0
```

y:

```
# tcpdump -XX -n -i h3-eth0
```

En la xterm para h1, envía un ping:

```
# ping -c1 10.0.0.2
```

Los paquetes ping van hasta el controlador, el cual indica al switch que los envíen por todos los interfaces excepto por el interfaz por el que se están enviando. Deberían verse paquetes ARP e ICMP idénticos correspondientes al ping en las dos terminales en las que se está ejecutando tcpdump. Así es como funciona un hub: envía todos los paquetes por todos los puertos de la red.

A continuación, veremos lo que pasa cuando un host que no existe no responde.

En la xterm de h1 ejecuta:

```
# ping -c1 10.0.0.5
```

Deberías ver tres peticiones ARP sin respuesta en las xterms en las que se está ejecutando tcpdump. Si más adelante tienes errores en tu código, ten en cuenta que tres paquetes ARP sin respuesta puede ser una señal de que se están descartando paquetes accidentalmente.

Cierra todas las xterms.

Con el comando “ovs-ofctl” podemos echar un vistazo a los flujos que el controlador POX ha instalado en el switch OpenFlow s1 con el siguiente comando:

```
$ sudo ovs-ofctl dump-flows s1
```

A continuación, vamos a echar un vistazo al código del hub ubicado en ~/pox/pox/forwarding/hub.py:

```
from pox.core import core

import pox.openflow.libopenflow_01 as of
```

```
from pox.lib.util import dpidToStr



log = core.getLogger()



def _handle_ConnectionUp (event):

  msg = of.ofp_flow_mod()

  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))

  event.connection.send(msg)

  log.info("Hubifying %s", dpidToStr(event.dpid))



def launch ():

  core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)



  log.info("Hub running.")
```

APIs útiles de POX:

- **connection.send()**: esta función envía un mensaje OpenFlow a un switch. Cuando se arranca una conexión con un switch se dispara el evento ConnectionUp. El código anterior invoca una función _handle_ConnectionUp() que implementa la lógica de hub.

- Clase **ofp_action_output**: Esta acción se utiliza junto con ofp_packet_out y ofp_flow_mod. Especifica un puerto del switch por el cual se quiere enviar el paquete. También puede tomar varios números de puerto “especiales”. Un ejemplo es el mostrado en el código anterior, en el cual se utiliza el puerto OFPP_FLOOD para enviar el paquete por todos los puertos excepto por el puerto por el que originalmente llegó el paquete.

- Clase **ofp_match** (no utilizada en el código anterior, pero que puede ser de utilidad en la práctica): los objetos de esta clase describen campos de la cabecera del paquete y un puerto de entrada para llevar a cabo la comparación con la tabla de flujos.
  Todos los campos son opcionales. Los elementos que no se especifican se consideran “comodines” y se corresponden con cualquier valor.

  Algunos campos importantes de los objetos ofp_match son:
  - dl_src: la dirección de origen a nivel de enlace (MAC).
  - dl_dst: la dirección de destino a nivel de enlace (MAC).
  - in_port: el puerto de entrada del paquete al switch.

Ejemplo: crear un “match” que se corresponda con paquetes entrantes por el puerto 3:

```
match = of.ofp_match()

match.in_port=3
```

- Mensaje OpenFlow **ofp_packet_out** (no utilizado en el código anterior, pero que puede ser de utilidad en la práctica): el mensaje ofp_packet_out indica a un switch que envíe un paquete. El paquete a enviar puede ser uno construido en el controlador, o puede ser un paquete que el switch haya recibido, almacenado en un buffer y reenviado al controlador (y ahora se referencia a través de un buffer_id).

  Algunos campos importantes son:
  - buffer_id: El identificador del buffer que se quiere enviar. No hay que especificarlo si se quiere enviar un paquete construido en el controlador.
  - data: conjunto de bytes que se quiere que envíe el switch. No hay que especificarlo si se está enviando un paquete almacenado en un buffer.
  - actions: una lista de acciones a aplicar. Para esta práctica únicamente consideraremos la acción ofp_action_output.
  - in_port: si se está enviando un paquete almacenado en un buffer, el número del puerto por el que inicialmente se recibió el paquete. En cualquier otro caso, especificar OFPP_NONE.

- Mensaje OpenFlow **ofp_flow_mod**: este mensaje indica al switch que ha de instalar una entrada en la "flow table". Las entradas de la "flow table" se comparan con los paquetes entrantes y cuando se produce una coincidencia se ejecuta una lista de acciones sobre los paquetes afectados. Las acciones son las mismas que las de ofp_packet_out, aunque tal y como se ha indicado previamente, para esta práctica únicamente es necesaria la acción ofp_action_output. El "match" se describe mediante un objeto ofp_match.

  Algunos campos importantes son:
  - idle_timeout: Número de segundos en estado inactivo antes de que se elimine la entrada de flujo. Por defecto, no se elimina después de ningún tiempo inactivo.
  - hard_timeout: Número de segundos antes de que se elimine la entrada de flujo. Por defecto, no se elimina.
  - actions: una lista de acciones a llevar a cabo sobre los paquetes que se correspondan con la entrada de flujo (ofp_action_output).
  - priority: cuando se utilizan entradas no exactas (con comodines), este campo especifica la prioridad de entradas que se superponen. Valores mayores indican una prioridad mayor. No es importante en el caso de entradas exactas o que no se superponen.
  - buffer_id: buffer_id de un buffer sobre el que aplicar las acciones inmediatamente. Si no han de ejecutarse acciones sobre ningún buffer, dejar sin especificar.
  - in_port: si se utiliza buffer_id, este campo indica el puerto de entrada asociado.
  - match: un objeto ofp_match. Por defecto "matchea" cualquier cosa, por lo que es recommendable establecer alguno de sus campos…

Ejemplo: creación de un flow_mod que envía los paquetes recibidos por el puerto 3 por el puerto 4.

```
        fm = of.ofp_flow_mod()
        fm.match.in_port = 3
        fm.actions.append(of.ofp_action_output(port = 4))
```