Práctica 11. Dispositivos de red

Esta práctica se realizará durante la clase de laboratorio.

Objetivos de aprendizaje

- Comprender el funcionamiento de un switch de capa 2.
- Comprender el funcionamiento de un router.

1. Introducción

Como se vio en la práctica 10, los conmutadores en las redes definidas por software se gobiernan mediante un controlador. En esta práctica se verá cómo utilizar uno de los numerosos conmutadores existentes realizando dos programas de control, uno para que el conmutador OpenFlow opere como un switch de capa 2 y otro para que el conmutador opere como un router.

Switches OpenFlow

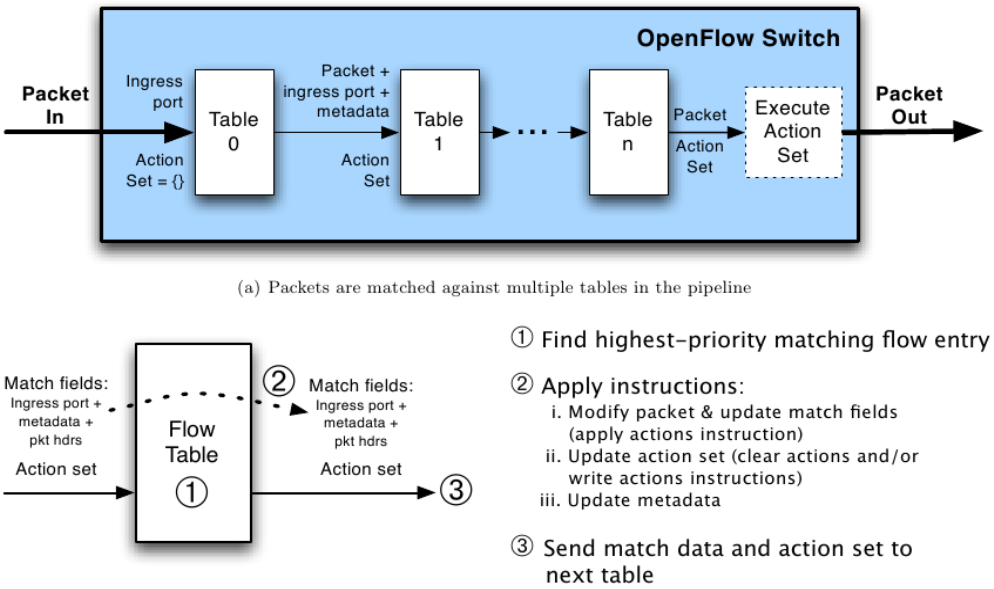
El pipeline de un switch OpenFlow contiene múltiples tablas de flujo, cada una de las cuales puede contener múltiples entradas. El pipeline define como interactúan los paquetes con las tablas de flujo. Un switch OpenFlow debe tener al menos una tabla de flujo, y opcionalmente puede tener más de una, en cuyo caso el procesamiento se simplifica. El procesamiento siempre comienza en la primera tabla de flujo y se utilizarán las tablas sucesivas dependiendo de la salida de la primera tabla. El proceso completo se muestra en la figura 1.

Cuando un paquete se procesa en una tabla de flujo, el paquete se compara con el patrón de cada una de las entradas de la tabla de flujo para seleccionar una entrada. Si en este proceso se encuentra una entrada compatible se ejecuta el conjunto de instrucciones asociado a esa entrada. Estas instrucciones pueden redirigir el paquete hacia otra tabla de flujo posterior donde se repite el mismo proceso. En la última tabla del pipeline, obviamente, no será posible esta redirección. Cuando el conjunto de instrucciones no contiene una instrucción de redirección, el procesamiento se detiene en la tabla correspondiente y normalmente el paquete se reenvía a alguno de los puertos de salida del switch [1].

Entradas de la tabla flujo

Cada entrada de la tabla de flujo consiste en:

- **Campos patrón (match fields)**: Los patrones estarán compuestos por el puerto de ingreso, cabeceras de paquete y opcionalmente metadatos procedentes de una tabla anterior.
- **prioridad**: Orden de precedencia de la entrada.
- **contadores**: Se actualizan cada vez que un paquete casa con el patrón.
- **instrucciones**: Permiten modificar el conjunto de acciones o el procesamiento en pipeline.
- **timeouts**: Tiempo máximo o tiempo en desuso antes de que elimine la entrada de la tabla de flujo.



(a) Packets are matched against multiple tables in the pipeline

(b) Per-table packet processing

Figura 1: Flujo de paquetes a través del pipeline OpenFlow (Figura tomada de [1])

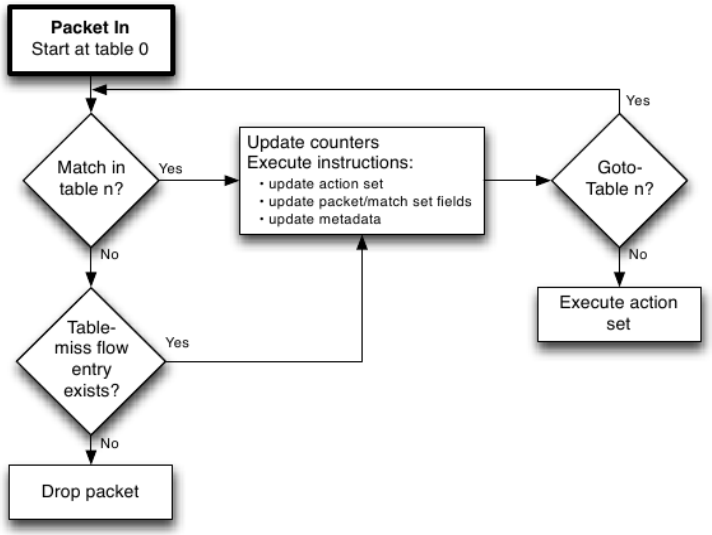


Figura 2: Diagrama de flujo del proceso de *matching* (Figura tomada de [1])

- **cookie**: Es un valor opaco elegido por el controlador. No se usa para procesar los paquetes.

Cuando se recibe un paquete, un switch OpenFlow lleva a cabo las funciones que se muestran en el diagrama de flujo de la figura 3. El switch comienza haciendo una búsqueda en la primera tabla de flujo en base a los *match fields*. Los *match fields* dependen del tipo de paquete y pueden ser, por ejemplo, las direcciones MAC, las direcciones IP. Además se pueden tener en cuenta los metadatos y el puerto de ingreso. Un paquete casa con una entrada de la tabla de flujo cuando los *match fields* coinciden con los valores especificados en esta. Si el valor de la entrada es ANY el campo se omite. El switch también puede soportar máscaras de bits que permiten buscar coincidencias parciales. Una vez que se encuentra una entrada coincidente, se incrementan los contadores y se aplica el conjunto de instrucciones.

Controladores

Para que un switch OpenFlow opere con un comportamiento determinado, por ejemplo como un switch de capa 2, es necesario añadir las entradas adecuadas a las tablas de flujo. Añadir, eliminar o modificar entradas de la tabla de flujo de un conmutador es la misión del controlador.

El controlador es un servidor al uso que se conecta a cada uno de los switches OpenFlow a través de una red de control. Dicha red de control permite el envío y la recepción de mensajes OpenFlow que permiten manipular paquetes y añadir entradas a la tabla de flujo. Cuando no se encuentra una coincidencia para un paquete en la tabla de flujo, éste se envía al controlador, para que tome una decisión sobre ese paquete mediante un programa software, que puede añadir entradas a la tabla de flujo, con el fin de que paquetes que coincidan con el mismo patrón no necesiten pasar por el controlador.

La inteligencia de la red se traslada al controlador y los conmutadores únicamente deberán realizar el reenvío de paquetes, no tomar decisiones.

Existen diversos controladores para dispositivos OpenFlow. Cada uno de los cuales dispone de una API para desarrollar programas de control. Algunos ejemplos de controladores son NOX (C++), POX(Python), OpenDayLight(Java), Ryu (Python), Floodlight (Java), Beacon(Java), etc. En esta práctica se ha optado por utilizar el controlador Ryu, ya que se programa en Python y permite hasta la especificación 1.3 de OpenFlow.

2. Introducción al controlador Ryu

Ryu es una infraestructura basada en componentes para redes definidas por software. Los componentes proveen una API bien definida que permite a los desarrolladores crear nuevas aplicaciones de control. Soporta varios protocolos para manejar dispositivos, entre ellos OpenFlow. Actualmente soporta las versiones 1.0, 1.2, 1.3 y 1.4 de este estándar [2].

La mayoría de los controladores tienen las siguientes características:

- Capacidad para recibir y gestionar eventos (por ejemplo, PACKET_IN y FLOW_REMOVED).
- Capacidad de analizar los paquetes entrantes y de crear paquetes nuevos para enviarlos hacia la red.
- Capacidad para crear y enviar mensajes OpenFlow (PACKET_OUT, FLOW_MOD) para reprogramar los conmutadores.

La forma de proceder de un programa de control normalmente es la siguiente:

1. Recibe un paquete de un conmutador que genera un evento de tipo PACKET_IN.
2. La rutina de atención al evento procesa el paquete, decidiendo las acciones a tomar, que pueden pasar por crear un nuevo paquete y enviarlo, o crear una nueva entrada en la tabla de flujo, para que el paquete entrante y los sucesivos se reenvíen o se eliminen.
3. Enviar el paquete creado o enviar un mensaje de FLOW_MOD para insertar un flujo.

Para crear una nuevas entradas en la tabla debemos especificar el patrón, el conjunto de instrucciones y las acciones. Veamos un ejemplo:

```
1  #!/usr/bin/python
2  #coding=utf-8
3  from ryu.base import app_manager
4  from ryu.controller import ofp_event
5  from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
6  from ryu.controller.handler import set_ev_cls
7  from ryu.ofproto import ofproto_v1_3
8  from ryu.lib.packet import packet
9  from ryu.lib.packet import ethernet
10 from ryu.base import app_manager
11
12 class L2Forwarding(app_manager.RyuApp):
13     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
14     def __init__(self, *args, **kwargs):
15         super(L2Forwarding, self).__init__(*args, **kwargs)
16
17     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
18     def packet_in_handler(self, ev):
19         msg = ev.msg                # Objeto que representa la estuctura de datos PacketIn.
20         datapath = msg.datapath     # Identificador del datapath correspondiente al switch.
21         ofproto = datapath.ofproto # Protocolo utilizado que se fija en una etapa
22                                     # de negociacion entre controlador y switch
23
24         ofp_parser=datapath.ofproto_parser # Parser con la version OF
25             # correspondiente
26
27         in_port = msg.match['in_port'] # Puerto de entrada.
28
29         # Ahora analizamos el paquete utilizando las clases de la libreria packet.
30         pkt = packet.Packet(msg.data)
31         eth = pkt.get_protocol(ethernet.ethernet)
32
33         # Extraemos la MAC de destino
34
35         dst = eth.dst
36
37         # Ahora creamos el match
38         # fijando los valores de los campos
39         # que queremos casar.
40         match = ofp_parser.OFPMatch(in_port=in_port, eth_dst=dst)
41
42         # Creamos el conjunto de acciones: FLOOD
43         actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
44
45         # Creamos el conjunto de instrucciones.
46         inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
47
48         # Creamos el mensaje OpenFlow
49         mod = ofp_parser.OFPFlowMod(datapath=datapath, priority=0, match=match, instructions=
   inst, buffer_id=msg.buffer_id)
50
51         # Enviamos el mensaje.
52         datapath.send_msg(mod)
```

La función `packet_in_handler` contiene cuatro elementos importantes. Primero el uso de la librería `packet` para procesar el paquete entrante, cuyos datos se encuentran en `msg.data`. Dicha librería es bastante extensa por lo que recomendamos consultar la documentación de la API disponible en [3]. Además, la librería permite construir paquetes, lo que será útil más adelante.

Lo siguiente que deberá observar es la construcción del objeto de la clase OPFMatch de la línea 40. Dicha clase representa el patrón que se utilizará posteriormente para insertar una entrada en la tabla de flujo. Los argumentos de la clase OFPMatch indican los valores que deben casar en los paquetes. Algunos de los campos sobre los que se puede aplicar patrones son los muestran en la tabla 1. Aquellos que no se indican podrán ser cualesquiera.

Seguidamente, se crea una lista de acciones que se van a los paquetes que casen con el patrón definido anteri-

Nombre	**Descripción**
in_port	Número del puerto de recepción
in_phy_port	Puerto físico de recepción.
metadata	Metadatos utilizados para el paso de información entre tablas de flujo.
eth_dst	Dirección MAC de destino.
eth_src	Dirección MAC de origen.
eth_type	Tipo de trama Ethernet.
vlan_id	VLAN ID
vlan_pcp	VLAN PCP
arp_op	Opcode del mensaje ARP.
arp_spa	IP de origen de ARP.
arp_tpa	IP de destino de ARP.
arp_sha	MAC de origen de ARP.
arp_tha	MAC de origen de ARP.
ip_dscp	IP DSCP
ip_ecn	IP ECN
ip_proto	Campo de protocolo de la cabecera IP
ipv4_src	Dirección IP versión 4 de origen
ipv4_dst	Dirección IP versión 4 de destino
icmpv4_type	Tipo de mensaje ICMP v4.
icmpv4_code	Código de mensaje ICMP v4.
tcp_src	Puerto de origen de TCP.
tcp_dst	Puerto de destino de TCP.
udp_src	Puerto de origen de UDP.
udp_dst	Puerto de destino UDP.
ipv6_src	Dirección IP de origen, IPv6
ipv6_dst	Dirección IP de destino, IPv6

Tabla 1: Algunos *matching fields* de OpenFlow 1.3

Instrucción	Descripción	Clase en la API de Ryu
`Goto Table`	En OpenFlow 1.1 y posteriores se soportan varias tablas de flujo. Con la acción `GotoTable`, se puede continuar el proceso de *matching* a una tabla de flujo que se especifique. Recuerde que el ID de la tabla de flujo debe ser mayor que el de la actual.	`OFPInstructionGotoTable`
`Write Metadata`	Escribe los metadatos que se pueden utilizar en la siguiente tabla de flujo.	`OFPInstructionWriteMetadata`
`Write Actions`	Añade una acción que se específica en el conjunto de acciones.	
`Apply Actions`	Aplica inmediatamente la acción especificada sin cambiar el conjunto de acciones	`OFPInstructionActions`
`Clear Actions`	Elimina todas las acciones en conjunto de acciones actual	
`Meter`	Aplica el paquete a la medida especificada	`OFPInstructionMeter`

Tabla 2: Instrucciones de OpenFlow

Parámetro	Descripción
`match`	Objeto `OFPMatch`.
`instructions`	Lista de instrucciones.
`priority`	Prioridad de la entrada en la tabla de flujo.
`hard_timeout`	Tiempo para que la entrada se elimine de la tabla de flujo.
`idle_timeout`	Tiempo sin usar hasta que la entrada se elimina de la tabla de flujo.
`buffer_id`	Identificador del buffer al que se debe aplicar esta entrada inmediatamente después de su instalación. Se suele poner textttbuffer_id=msg.buffer_id para que se aplique al paquete recibido por el controlador.

Tabla 3: Parámetros de la clase `OFPFlowMod`.

ormente. Esta lista está definida en la línea 43. Sólo contiene una acción definida por el objeto OFPActionOutput, que se utiliza para especificar el tipo de reenvío de paquetes a utilizar. Como argumento suele recibir el número del puerto físico del conmutador. Además pueden especificarse algunos valores prdefinidos. Por ejemplo, `OFPP_IN_PORT`, reenviaría el paquete por el puerto de ingreso; `OFPP_FLOOD`, reenviará los paquetes por todos los puertos pertenecientes a la misma VLAN excepto los bloqueados y los de recepción; y `OFPP_CONTROLLER` envía el paquete hacia el controlador como un mensaje de tipo `PacketIn`. Existen otras acciones, por ejemplo para decrementar el TTL de un paquete, que se pueden aplicar que se encuentran documentadas en [4]. Nótese también que la lista de acciones puede contener más de una acción.

El siguiente elemento es la lista de instrucciones definida en la línea 46. Las instrucciones definen qué ocurre cuando un paquete casa con el patrón especificado. Se contemplan las instrucciones definidas en la tabla 2, aunque generalmente se suele utilizar la instrucción `OFPInstructionActions`.

A continuación, en la línea 49, se crea un objeto de tipo `OFPFlowMod` que es el mensaje OpenFlow que modifica una entrada de la tabla de flujo. En el constructor, por supuesto se deben pasar el match y las instrucciones. Además se pueden añadir otros parámetros como los descritos en la tabla 3.

Finalmente, en la línea 41 se instala la nueva entrada de la tabla de flujos en el switch mediante el método `send_msg` del objeto *data path*.

Para probar el código anterior inicie el controlador utilizando:

```
sudo ryu-manager <fichero>.py
```

Luego inicie una topología mininet con un único switch y 4 hosts:

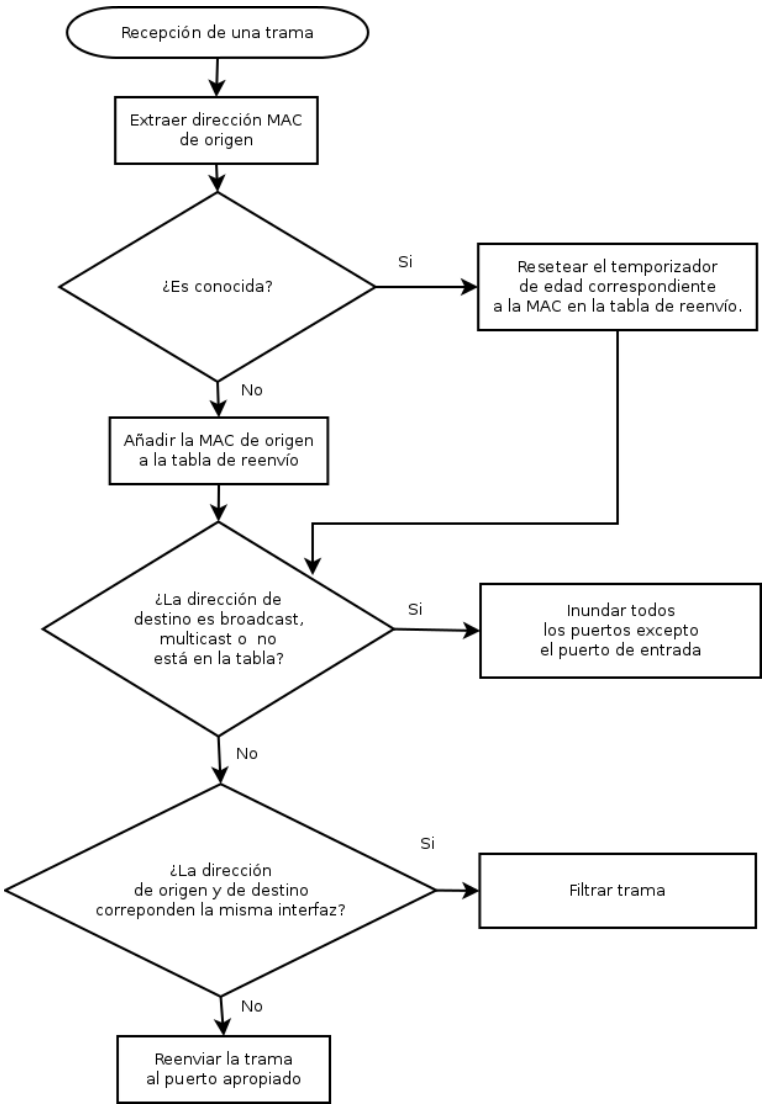


Figura 3: Diagrama de flujo del funcionamiento de un switch de capa 2.

```
sudo mn --topo single,4 --mac --controller remote --switch ovsk,protocols=OpenFlow13
```

Además se indica que se va a utilizar un controlador remoto y un switch con el protocolo OpenFlow 1.3.

3. Implementación de un switch de capa 2

Modifique el ejemplo de la sección anterior para que funcione como se muestra en el diagrama de flujo de la figura 3, es decir, como un switch de capa 2.

A continuación se indican algunos aspectos que pueden ser útiles a la hora de desarrollar el código:

- Para crear la tabla de reenvío del switch, puede añadir un atributo a la clase que sea un diccionario de Python. Para inicializarlo utilice una instrucción similar a esta:

  ```
  mac_to_port = dict()
  ```

  que inicializa el diccionario a vacío.

  Luego podrá determinar si una dirección MAC se encuentra en la tabla de reenvío del siguiente modo:

```
if dst in self.mac_to_port.keys():
```

Para asignar una entrada al diccionario puede utilizar algo similar a esto:

```
self.mac_to_port[src] = in_port
```

- Para determinar si una dirección es broadcast o multicast puede utilizar la siguiente sentencia:

  ```
  if haddr_to_bin(dst) == mac.BROADCAST) or mac.is_multicast(haddr_to_bin(dst)):
  ```

  Para hacer uso de estas funciones deberá importar:

  ```
  from ryu.lib import mac
  from ryu.lib.mac import haddr_to_bin
  ```

- El reseteo del temporizador de edad puede hacerse indicando el valor adecuado en el atributo `idle_timeout` de la clase `OFPFlowMod`.

4. Implementación de un router

Lógica de reenvío de un router es la siguiente:

- Revisar si el paquete está correcto (alcanza el tamaño mínimo y tiene el checksum correcto). Si alguna de estas dos condiciones falla se elimina la trama.
- Decrementar el TTL y recomputar el checksum para la cabecera modificada.
- Encontrar la entrada de la tabla de enrutamiento con el prefijo más largo que casa con la dirección IP de destino del paquete.
- Comprobar la caché ARP para determinar la dirección MAC del siguiente salto. Si la dirección MAC está en caché, se envía construye la trama y se envía el paquete al siguiente salto. En otro caso habrá que hacer una petición ARP para determinar la dirección MAC del siguiente salto y añadir el paquete a la cola hasta que se reciba la respuesta ARP.

Implementación de ARP

Cuando un host realiza una petición ARP para determinar la dirección MAC correspondiente a una MAC es necesario que el host con esa dirección IP responda a la petición, siempre que se encuentre en la misma red. En el caso de un router esto debe ocurrir cuando un host de la red realiza una petición relacionada con una de las interfaces del router. Por ello deberá implementar la respuesta ARP correspondiente a cada una de las interfaces.

Respuestas al ping (ICMP)

Cuando se realiza ping hacia una de las interfaces del router éste deberá responder adecuadamente. Deberá implementar esta respuesta.

Tabla de enrutamiento y reenvío

Implemente una tabla de enrutamiento estático y la lógica de reenvío. Para ello deberá modificar fijar los valores de la dirección MAC de origen y la dirección MAC de destino de la trama reenvíada. Inserte los flujos adecuados en la tabla de flujo y utilice las acciones adecuadas (recuerde decrementar el TTL). Si no es posible reenviar el paquete deberá responderse con un mensaje ICMP de destino inalcanzable.

Peticiones ARP

Con el fin de poder insertar las direcciones MAC adecuadas en las cabeceras de trama, deberá implementar una caché ARP en el controlador y deberá implementer la posibilidad de realizar peticiones ARP desde el router para determinar las direcciones MAC necesarias si éstas no estan en la tabla.

Información de apoyo

- Durante el desarrollo pueden serle útil estas dos funciones.

```
#  Inserta una entrada a la tabla de flujo.
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
    if buffer_id:
      mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
        priority=priority, match=match,
        instructions=inst, idle_timeout=30,command=ofproto.OFPFC_ADD)
    else:
      mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
        match=match, instructions=inst, idle_timeout=30,command=ofproto.OFPFC_ADD)
    print(mod)
    datapath.send_msg(mod)

# Enviar un paquete construido en el controlador
# hacia el switch
def send_packet(self, datapath, port, pkt):
  ofproto = datapath.ofproto
  parser = datapath.ofproto_parser
  pkt.serialize()
  data = pkt.data
  actions = [parser.OFPActionOutput(port=port)]
  out = parser.OFPPacketOut(datapath=datapath,
        buffer_id=ofproto.OFP_NO_BUFFER,
        in_port=ofproto.OFPP_CONTROLLER,
        actions=actions,
        data=data)
  datapath.send_msg(out)
```

- También pueden serle de utilidad las funciones de la librería netaddr, que permiten manejar direcciones IP. Consulte la dirección `https://pythonhosted.org/netaddr/`.
- Para utilizar parsear y construir paquetes utilice la librería `packet`. Para utilizarla deberá realizar los import correspondientes:

```
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import arp
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
```

La documentación de esta librería puede encontrarla en `http://ryu.readthedocs.org/en/latest/library_packet_ref.html`.

Por ejemplo, para analizar si el paquete de entrada es una petción ARP y responder se utilizaría el siguiente código:

```
 if eth.ethertype==ether.ETH_TYPE_ARP:
      arp_msg= pkt.get_protocol(arp.arp)
      if (arp_msg.dst_ip == self.interfaces[in_port][0] and arp_msg.opcode==arp.
    ARP_REQUEST):
```

```
4           e = ethernet.ethernet(dst=src,
5                                 src=self.macs[in_port],
6                                 ethertype=ether.ETH_TYPE_ARP)
7           a = arp.arp(opcode=arp.ARP_REPLY,
8                       src_mac=self.macs[in_port], src_ip=arp_msg.dst_ip,
9                       dst_mac=src, dst_ip=arp_msg.src_ip)
10          p = packet.Packet()
11          p.add_protocol(e)
12          p.add_protocol(a)
13          self.send_packet(datapath, in_port,p)
```

Referencias

[1] OpenFlow Switch Specification. Version 1.3.3 (Protocol version 0x04). September 27, 2013. ONF TS-015 https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf

[2] http://osrg.github.io/ryu/

[3] http://ryu.readthedocs.org/en/latest/library.html

[4] http://ryu.readthedocs.org/en/latest/ofproto_v1_3_ref.html