



S09-S12: Desarrollo de una aplicación SDN	 
Redes Software Última modificación: 20-03-2025	2024-25

Introducción y objetivos

1. Objetivo

En este laboratorio desarrollará de manera independiente y basándose en los conocimientos adquiridos en los laboratorios anteriores una aplicación Software Defined Networking (SDN) que implemente la funcionalidad de un encaminador IPv4 y que se ejecute en el controlador RYU. Para demostrar el funcionamiento se apoyará en un entorno de emulación de red sobre Mininet.

2. Normas básicas

Estudie este documento en profundidad antes de empezar el laboratorio. Asegúrese que lo entiende en su totalidad. El laboratorio está pensado para realizarse en grupos de dos estudiantes.

Aclare posibles dudas con los profesores en el laboratorio.

El proceso de desarrollo se ha dividido en cuatro etapas. Cree un subdirectorio **distinto** para el desarrollo de cada una de las etapas. A partir de la segunda etapa, copie los programas que haya desarrollado en la etapa anterior y modifíquelos para satisfacer los criterios de la etapa correspondiente.

El laboratorio se evalúa hito a hito a partir de las respuestas que se incluyan el correspondiente documento que se entregará en Aula Global (AG).

3. Utilización de los ejemplos de código

Este manual incluye ejemplos de código que se deberían utilizar en las prácticas. Se pueden utilizar cortando y pegando el código con un visor PDF arrancado en la máquina virtual o ajustando el funcionamiento de dicha función entre la máquina virtual y el

anfitrión (e.d. el PC de laboratorio). Es posible que tenga que eliminar algún carácter especial y reajustar la indentación.

Nota

Tenga en cuenta que cualquier texto que aparezca en cursiva en el listado o entre flaches, del estilo a *<IPv4 del router>* ó *<IP del router>*, tiene que ser sustituido por el valor indicado en el texto.

Un ejemplo de código que habría que editar después de ser pegado en el editor de textos es:

```
def function(self,argumento):  
    argumento += '<direccion IP>'  
    print('Esto es un ejemplo de mensaje')  
    print(f'con argumento formateado argumento')
```



Intente copiar y pegar este texto e identifique si hay que algún carácter especial y cómo hay que reindentar el código en el editor. La “chincheta” indica que el código está incluido en el fichero PDF. Un visor que soporte adjuntos (*attachments*) permitirá extraer el código a un fichero.

Nota

En esta práctica, se tiene que utilizar el modo *Python* para todo el código que se edite. **Se recomienda** partir de los ejemplos desarrollados en el laboratorio de *Introducción a Mininet y Ryu*.

4. Evaluación de la práctica

Se abrirá un entregador para el documento de práctica. Deberá entregarse **un** documento por grupo de trabajo y en formato **PDF**. Debe contener los listados de los programas desarrollados, capturas de pantalla de los terminales y Wireshark donde sea necesario y texto explicativo, **que se mantendrá lo más resumido posible**.

La práctica consta de tres hitos que se evalúan de la siguiente manera:

Hito	Puntuación	
Hito 1	3 puntos	Conmutación a nivel IP
Hito 2	3 puntos	Router responde a ICMP
Hito 3	4 puntos	Router implementa la funcionalidad ARP

Tabla 1.: Criterios de evaluación

Hito
Los hitos se identifican con este formato. El texto del hito especifica la funcionalidad necesaria para cumplir el hito.

1. Implementación básica de la conmutación de paquetes IPv4

El objetivo de esta práctica es reproducir los procesos de conmutación de paquetes de nivel 3 (IPv4) que tienen lugar en un encaminador. Para implementar el entorno completo, vamos a proceder de manera incremental:

- I. Implementación de la tabla de encaminamiento en el conmutador
 - II. Implementación de las respuestas al protocolo ICMP en el conmutador
 - III. Implementación de las respuestas al protocolo ARP en el conmutador
- Empezamos con el proceso de conmutación de paquetes IPv4 en el *router*.

1.1. Introducción al entorno

En este laboratorio pretendemos emular el siguiente entorno:

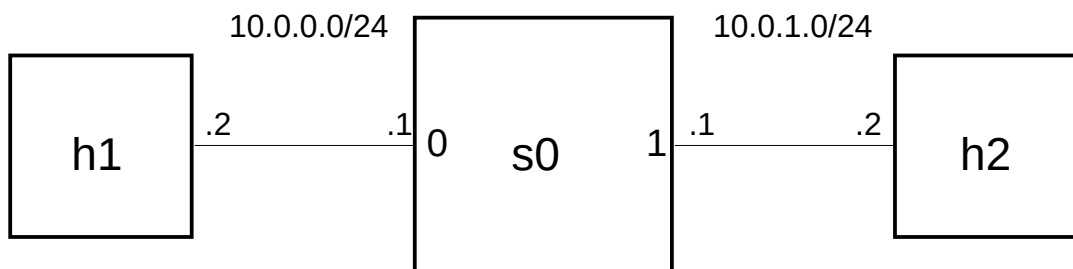


Figura 1.1.: Escenario Mininet del desarrollo

Para que funcione correctamente, se tendrá que mantener la siguiente correspondencia entre direcciones IPv4 y MACs en los equipos del escenario:

Dir. IPv4	MAC	V	Dir. IPv4	MAC	V
10.0.0.1	70:88:99:00:00:01	*	10.0.1.1	70:88:99:10:00:02	*
10.0.0.2	00:00:00:00:00:01		10.0.1.2	00:00:00:00:00:02	

Tabla 1.1.: Correspondencia entre direcciones IPv4 y MACs

Nota: Las direcciones marcadas con una estrella en la columna V son direcciones **virtuales**

Recordemos que para que los equipos finales (*h1* y *h2*) envíen paquetes IPv4, deben de conocer la dirección MAC del destino si está en la misma subred o de la puerta por defecto (*default gateway*). Además, los equipos finales sólo recibirán paquetes cuya MAC destino sea la del equipo o la dirección de *broadcast* (FF:FF:FF:FF:FF:FF)¹

1.2. Infraestructura de desarrollo

Para realizar el desarrollo, copie los ficheros que necesite de los ejemplos de Ryu y Mininet a un directorio **nuevo** y trabaje sobre las copias. Se recomienda que empiece con el siguiente árbol de ficheros:

```
$HOME/Devel
├─ chap1
│   └─ simple_router.py
│      └─ scenario.py
```

1.3. Gestión de MACs de los equipos finales

Como se ha expuesto en la introducción, los equipos finales requieren tener la información MAC del siguiente salto para poder enviar paquetes. Esta información la proporciona normalmente el protocolo de resolución de direcciones o Address Resolution Protocol (Protocolo de resolución de direcciones) (ARP). Para casos excepcionales, Linux permite definir la traducción de direcciones de modo estático.

Para el primer paso, la información ARP se programará de manera estática tanto en los equipos finales como en el encaminador. En los equipos finales, la puerta por defecto será la interface del encaminador a que está conectado.

¹En funcionamiento normal; se puede forzar la recepción de **todos** los paquetes, poniendo el equipo en modo *promiscuo*. Esto queda fuera del alcance de esta práctica.

En un equipo Linux real, la configuración se realizaría con los comandos:

```
ip link set dev eth0 up
ip address add <dirección IPv4 de equipo>/24 dev eth0
ip route add default <dirección IPv4 del router>
arp -s <dirección IPv4 del router> <MAC del router>
```

En el caso de este laboratorio, se tiene que crear una clase de equipo final que extienda la clase `Host` y permita programar de manera estática una entrada en la tabla de ARP. Llámela `ArpHost`.

La tabla de ARP se programará a la hora de instanciar la clase. La información se deberá expresar como una tupla de dos valores, siendo el primero la dirección MAC y el segundo la dirección IP.

Busque la definición de la clase **Host** de Mininet y compruebe qué parámetros admite por defecto para configuración. Cuando se instancia la clase `Host`, los parámetros que se le pasan se transfieren al método `config`. Es este el método que tendrá que redefinir. Puede encontrar información adicional en [1] y en [2] de cómo implementar un método `config` que *extienda* el método original.

Es recomendable fijar el protocolo a `OpenFlow13` en la definición del conmutador. Esto se puede realizar con el parámetro adicional `protocols="OpenFlow13"` a la hora de crearlo.

Recuerde que el entorno necesita que el controlador sea de tipo `RemoteController` para poder utilizar el controlador `Ryu` que tiene instalado en el entorno de desarrollo.

1.4. Proceso de conmutación en el conmutador

El proceso de conmutación en los conmutadores deberá emular el proceso que se realiza en un encaminador de nivel 3.

La selección de paquetes se realiza mediante protocolo y dirección destino. Para ello se tendrá que implementar un objeto `OFPMatch()` que detecte el código de tipo de paquete IPv4 en la cabecera MAC y filtre por prefijo IPv4 en la dirección de destino IPv4 para cada una de las rutas:

```
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                        ipv4_dst=('10.0.0.0', '255.255.255.0'))
```

Los paquetes se emiten una vez ajustados los campos de las cabeceras de la siguiente forma:

- i La dirección MAC origen es la de la interface del encaminador por la que salen
- ii La dirección MAC destino es la del equipo final al que van destinados
- iii El campo TTL de la cabecera IPv4 se decrementa en una unidad

```
actions = [
    parser.OFPACTIONOutput(1),
    parser.OFPACTIONDecNwTtl(),
    parser.OFPACTIONSetField(eth_src='<MAC interface>'),
    parser.OFPACTIONSetField(eth_dst='<MAC host>')
]
```

Nota

Para mantener el código sea legible, es recomendable crear métodos que devuelvan las acciones. Partiendo del ejemplo anterior, se pueden crear los siguientes métodos en la clase que implementa el controlador Ryu para la conmutación y el descarte de paquetes:

```
def forwardActions(self, parser, ofproto, port, src, dst):
    return [
        parser.OFPACTIONOutput(port),
        parser.OFPACTIONDecNwTtl(),
        parser.OFPACTIONSetField(eth_src=src),
        parser.OFPACTIONSetField(eth_dst=dst),
    ]
def dropActions(self, parser, ofproto):
    return [ ]
```

y luego utilizarlas en el código. Observe que se pasan siempre los parámetros parser y ofproto aunque no se utilicen. De esta manera, el código resulta más homogéneo y, por tanto, legible.

Incluya también otro método para la lista que implementa la acción de mandar al controlador.

Nota

Tenga en cuenta que el orden en que se indican las acciones es en el que se ejecutan en el conmutador y, por tanto, **es significativo**. El ejemplo anterior **NO** funciona; hay que arreglarlo.

La tabla de conmutación que se debe implementar es la siguiente:

prioridad	filtro	acción
10000	Paquete LLDP	Descartar
10000	Paquete IPv6	Descartar
1000	prefijo IP == 10.0.0.0/24	Conmutar a la 1
1000	prefijo IP == 10.0.1.0/24	Conmutar a la 2
0	resto	Mandar al controlador

Tabla 1.2.: Tabla de conmutación inicial

Las primeras entradas (las de mayor prioridad) harán que se descarten los paquetes que quedan fuera del objetivo de este laboratorio de manera automática.

En el código del conmutador se recomienda proceder de la siguiente manera:

1. Partir de cualquiera de los ejemplos de la sesión de SDN o del ejemplo del conmutador simple para OpenFlow1.3 de los ejemplos de Ryu, copiando el fichero
`$HOME/Install/ryu/ryu/app/simple_switch13.py`
2. Mantener el envío al controlador como último recurso en la tabla de conmutación.
3. Eliminar el código de respuesta a paquetes recibidos del conmutador salvo la parte que imprime la información básica del paquete recibido (p.ej interface, Ethertype y MACs)

Hito 1: (4 puntos)

Ejecute el entorno y demuestre que se puede realizar ping entre los dos equipos. A partir de este resultado, el hito se valorará de la siguiente manera:

Subhito 1(1 puntos): Para conseguir este subhito se ha de haber creado una clase derivada de Host para los equipos finales, que permita programar su tabla de ARP durante la instanciación en el entorno de simulación de Mininet.

Subhito 2(3 puntos): Capture el tráfico en el extremo de *h1* (con Wireshark) y demuestre que el campo TTL se decrementa en el conmutador. Para ello haga ping de *h1* a *h2* y compare el campo TTL de las preguntas y respuestas ICMP.

2. Gestión del protocolo ICMP

En este paso introducimos la funcionalidad de envío de paquetes desde el controlador. Para ello, haremos que los equipos finales puedan comprobar el estado de las puertas del conmutador enviando mensajes ICMP a la dirección IP correspondiente. Para ello, el controlador deberá enviar paquetes de respuesta ICMP a los paquetes de tipo ICMP_REQUEST que reciba.

Nota

Se recomienda partir del código que se desarrolló en el capítulo anterior. Duplíquelo en un directorio nuevo^a. Idealmente, el contenido del directorio de desarrollo de la Máquina Virtual (VM) se debería parecer a :

```
$HOME/Devel
├── chap1
│   ├── simple_router.py
│   └── scenario.py
└── chap2
    ├── simple_router.py
    └── scenario.py
```

^aDesde \$HOME/Devel puede ejecutar el comando `cp -rv chap1 chap2`.

Para implementar este paso tendrá que modificar el código del controlador (fichero `simple_router.py`) exclusivamente.

2.1. Función de envío de paquetes de respuesta ICMP

Parta de la captura de tráfico que realizó en el paso anterior y observe cómo se derivan los campos del paquete ICMP_REPLY a partir del paquete ICMP_REQUEST. Utilizando la librería de generación de paquetes de Ryu, tendrá que implementar una función que reciba un paquete ICMP_REQUEST y el puerto por el que se recibió y envíe la respuesta ICMP_REPLY correspondiente por el mismo puerto. En la figura 2.1 se ve como hacer los intercambios de *algunos* de los campos para generar el paquete de respuesta ICMP.

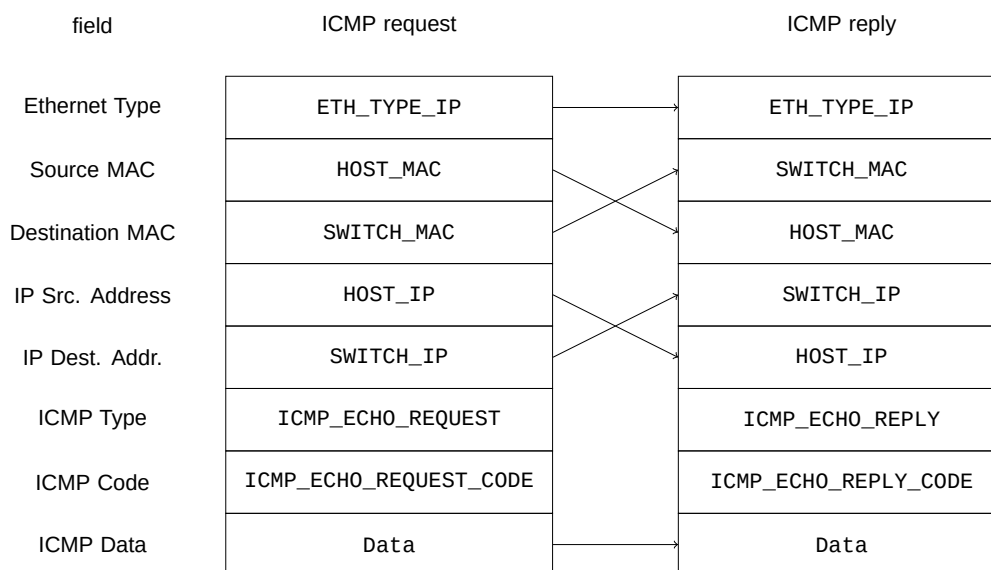


Figura 2.1.: Construcción del paquete de respuesta ICMP

Para implementar esta función, utilice la documentación de la librería de paquetes de Ryu, que se describe en la documentación de Ryu [3] y en el libro de Ryu [4].

2.2. Integración en el código del controlador

Una vez implementada esta función, incluya el filtrado necesario en la tabla de conmutación del controlador, de manera que los paquetes dirigidos a las direcciones asignadas a las puertas del conmutador sean enviados al controlador. Piense la prioridad que deben de tener estas entradas para que este tráfico en concreto sea atendido por el controlador.

Finalmente, modifique la función de procesamiento de eventos para que los paquetes ICMP que lleguen al controlador sean tratados con la función de generación de respuestas **exclusivamente**.

Hito 2: (3 puntos)

Demuestre mediante el comando ping que todos los equipos finales pueden detectar ambos puertos del 'encaminador' mediante el protocolo ICMP.

3. Integración del protocolo ARP

En este paso eliminaremos la programación estática del protocolo ARP en los equipos finales y haremos que el controlador gestione la respuesta a este protocolo. Para ello, añadiremos código para detectar y decodificar paquetes ARP y una función de respuesta a paquetes ARP_REQUEST en la gestión de los paquetes enviados por el conmutador al controlador.

Nota

Se recomienda partir del código que se desarrolló en el capítulo anterior. Duplíquelo en un directorio nuevo. Idealmente, el contenido del directorio de desarrollo de la VM se debería parecer a :

```
$HOME/Devel
├── chap1
│   ├── simple_router.py
│   └── scenario.py
├── chap2
│   ├── simple_router.py
│   └── scenario.py
└── chap3
    ├── simple_router.py
    └── scenario.py
```

Para implementar este paso tendrá que modificar tanto el escenario (`scenario.py`) como el código del controlador (`simple_router.py`).

3.1. Eliminar la configuración estática del protocolo ARP

En el código del escenario, elimine *exclusivamente* el código relacionado con la programación estática de la resolución de direcciones MAC en el escenario¹.

Ejecute el escenario con el código del controlador utilizado en el capítulo anterior **sin modificar**. Compruebe que se ha dejado de poder hacer ping desde cualquier equipo final a cualquier dirección IP remota (e.d. las direcciones de los puertos del

¹Se debe mantener el *gateway* por defecto en los equipos finales

encaminador y la del otro equipo final).

3.2. Decodificación y respuesta al protocolo ARP en el controlador

Para poder trabajar con el protocolo ARP, tendrá que empezar por importar el módulo `arp` de la librería `ryu.lib.packet`. Una posibilidad de mantener el código legible es incluir todos los módulos de protocolos en una única línea en el bloque de importación de módulos:

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

Esto le permitirá detectar la presencia y decodificar la información del protocolo ARP más adelante y reaccionar adecuadamente si está presente:

```
pkt = packet.Packet(msg.data)
```

para mandar el paquete a una función que lo trate adecuadamente.

Respuesta a paquetes ARP

El controlador deberá responder a paquetes de tipo `ARP_REQUEST` con paquetes `ARP_REPLY`. La siguiente figura muestra cómo se debe transformar el paquete `ARP_REQUEST` en un paquete `ARP_REPLY`:

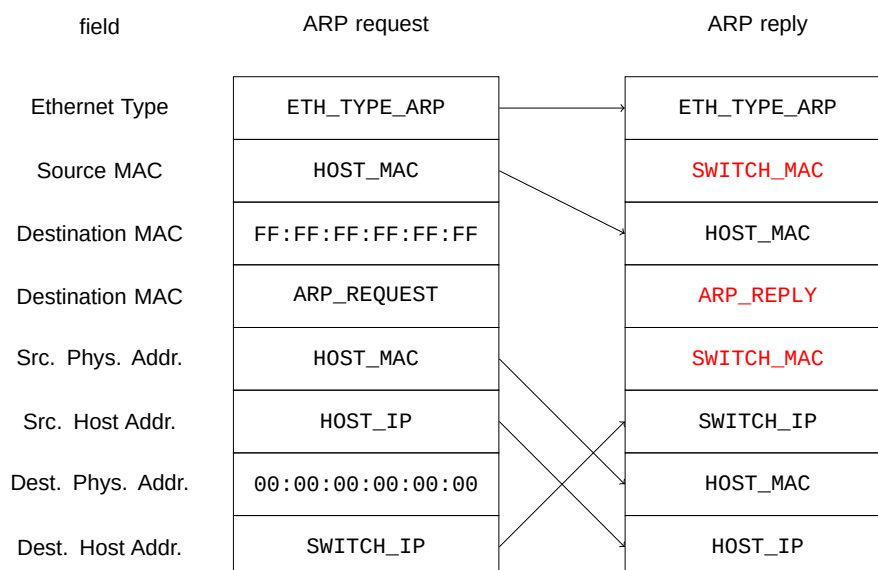


Figura 3.1.: Como crear la respuesta ARP a partir de la petición

Al igual que para la implementación de la respuesta del protocolo ICMP, utilice la documentación de la librería de paquetes de Ryu, que se describe en la documentación de Ryu [3] y en el libro de Ryu [4], para implementar el código que genere y envíe el paquete de respuesta ARP.

Hito 3: (3 puntos)

Como se indicó anteriormente, utilice el programa para el controlador realizado en el capítulo anterior con el escenario Mininet modificado para este hito y demuestre que no hay resolución de direcciones MAC y que, por tanto, tampoco hay tráfico ICMP cuando intente hacer un ping.

A continuación, demuestre que cuando arranca el escenario Mininet y el controlador con el programa que acaba de desarrollar, se puede realizar un ping extremo a extremo y que las direcciones MAC se obtienen automáticamente sin necesidad de tenerlas programadas de manera estática en los equipos finales.

A. La cabecera Ethernet

En este apéndice se repasan los conceptos básicos de redes que se utilizan en las prácticas de SDN de la asignatura.

A.1. La cabecera Ethernet

La siguiente figura muestra los campos en un paquete Ethernet (sin preámbulo):

DMAC	SMAC	EthType	Cabecera	...
------	------	---------	----------	-----

Figura A.1.: Anatomía de un paquete encapsulado en Ethernet

El primer campo es la dirección MAC de destino y el segundo es la dirección MAC origen.

El último campo de la cabecera Ethernet es el campo EtherType, que determina el tipo de paquete que sigue a la cabecera Ethernet. En la siguiente tabla [5] se indican los valores más usuales y que necesita conocer durante la práctica:

EtherType	Tipo de paquete
0x0800	IPv4
0x0806	ARP
0x08dd	IPv6

Tabla A.1.: EtherTypes relevantes

Recordemos el funcionamiento en modo normal de una tarjeta Ethernet¹. En su electrónica tiene un registro (de 6 bytes) en el que se guarda su dirección MAC. Esta dirección se pone en los paquetes que emite la tarjeta hacia la red en el campo de dirección MAC origen (Campo SMAC).

¹Se trata solamente el funcionamiento con IPv4 e IPv6 unicast. El tratamiento de multicast está fuera de los objetivos de este documento.

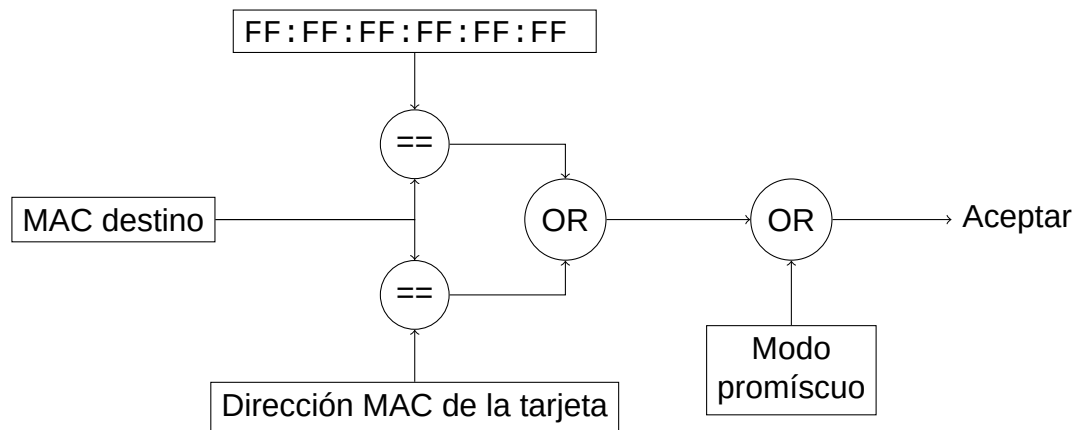


Figura A.2.: Filtro de entrada en una tarjeta Ethernet

Además, en el sentido de entrada (ilustrado en la figura A.2), la tarjeta utiliza este registro para filtrar paquetes y quedarse sólo aquellos que tienen esta dirección o la dirección de “broadcast” FF:FF:FF:FF:FF:FF en el campo de dirección MAC de destino (campo DMAC). Cuando se utiliza la dirección de “broadcast” los paquetes son admitidos por todos los equipos conectados a la red Ethernet a la que está conectada la interface Ethernet por la que son emitidos.

Este es el funcionamiento normal. Cuando la tarjeta se pone en modo *promíscuo*, entrega todos los paquetes al ordenador. Este modo se utiliza en “sniffers” como, p.ej. Wireshark.

B. Address Resolution Protocol (Protocolo de resolución de direcciones)

En un paquete IPv4, las direcciones MAC están en el conjunto de direcciones MAC de los extremos de la red por la que circula en cada momento. En la red a la que se conecta el equipo que origina el paquete, la dirección de MAC origen corresponde con la dirección IP origen. En la red a la que se conecta el equipo destinatario del paquete, la dirección MAC destino corresponde a la dirección IP destino.

Normalmente, estas correlaciones no se conocen *a priori* en los equipos de red.

Para averiguar la dirección MAC de un equipo en una red, existe el protocolo ARP. Este protocolo básicamente funciona con un intercambio de dos paquetes:

1. Pregunta a todos ("broadcast"): ¿Qué MAC tiene el equipo con una dirección IP concreta? Lo pregunta un equipo con dirección IP concreta. (who-has)
2. Respuesta del equipo con dicha dirección IP: La dirección MAC es la del equipo con la dirección IP por la que se pregunta. (is-at)

Cuando un equipo ha establecido la correlación entre una dirección IP y su MAC, intenta realizar periódicamente una confirmación, emitiendo un paquete de pregunta dirigido al equipo al que presume que tiene la dirección IP. Para ello, pone la dirección MAC de dicho equipo en lugar de la dirección "broadcast" en el campo MAC destino.

La siguiente captura muestra un intercambio ARP entre dos equipos en una red de área local:

No.	Time	Source	Destination	Protocol	Length	Info
228	15.973666	Comtrend_dc:58:55	Apple_71:10:c0	ARP	60	Who has 192.168.1.36? Tell 192.168.1.1
229	15.973712	Apple_71:10:c0	Comtrend_dc:58:55	ARP	42	192.168.1.36 is at 78:4f:43:71:10:c0

Figura B.1.: Un intercambio ARP en una red de área local

Acrónimos

AG	Aula Global
ARP	Address Resolution Protocol (Protocolo de resolución de direcciones)
ICMP	Internet Control Message Protocol (Protocolo de mensajes de control de Internet)
MAC	Medium Access Control (Control de acceso al medio)
OF	OpenFlow
SDN	Software Defined Networking
VM	Máquina Virtual

Referencias

- [1] *Host configuration methods*. url: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#config>.
- [2] *Introduction to Mininet: additional examples*. url: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#examples>.
- [3] *Ryu documentation*. url: <https://ryu.readthedocs.io/en>.
- [4] *Ryu SDN Framework*. url: <https://osrg.github.io/ryu-book/en/html/index.html#>.
- [5] *EtherType*. url: <https://en.wikipedia.org/wiki/EtherType>.