


S08-Lab 2: Introducción a Mininet y Ryu	 Universidad Carlos III de Madrid
Redes Software Última modificación: 20-03-2025	2024-25

Introducción y objetivos

1. 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.

Lea el documento detenidamente. Cuando llegue a un **hito**, llame a un profesor de laboratorio para que lo evalúe.

2. Entorno de trabajo

Utilice la Máquina Virtual (VM) que creó en el laboratorio de virtualización. Es conveniente que haya generado el fichero .ova con la instalación básica para poder desechar una instancia que no funcione correctamente y arrancar desde un estado conocido.

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 con el que se realiza laboratorio). Es posible que tenga que eliminar algún carácter especial y reajustar la indentación.

Nota

En esta práctica, se tiene que utilizar el modo *Python* para todo el código que se edite.

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

```
def function(self,argumento):  
    print('Esto es un ejemplo de mensaje')  
    print('en Python')  
    print('con argumento formateado %s' % argumento)
```

Nota

Intente copiar y pegar este texto e identifique si hay que algún carácter especial y cómo hay que reindentar el código en el editor.

4. Evaluación

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

1. El simulador de redes Mininet

1.1. Introducción

El simulador de redes Mininet está basado en distintas funciones de virtualización del sistema Operativo Linux. El proyecto está disponible en [1].

1.1.1. Configuración preliminar

Para realizar algunas actividades de este laboratorio es necesario tener activado un controlador OpenFlow (OF) de pruebas, que está contenido en el paquete `openvswitch-testcontroller`. Para instalarlo hay que seguir los siguientes pasos:

```
sudo apt install openvswitch-controller
sudo service openvswitch-controller force-stop
cd /usr/bin
ln -sf ovs-testcontroller ovs-controller
```

NOTA: En el segundo paso se desactiva el controlador, porque el proceso de instalación lo activa automáticamente y a continuación se crea un alias, porque el ejecutable se instala con un nombre distinto del que espera mininet.

Una vez finalizado este laboratorio, se recomienda eliminar el paquete de la siguiente manera:

```
cd /usr/bin
sudo rm ovs-controller
cd
sudo apt purge ovs-testcontroller
sudo apt autoremove --purge
```

1.1.2. Primera interacción con Mininet

Al ejecutar Mininet, se activa una topología y un controlador SDN que la controla. Por defecto, la topología se compone de un conmutador(*switch*), dos equipos finales (*hosts*) y un controlador (*controller*)¹

```
student@student-VirtualBox:~$ sudo mn
[sudo] password for student: introducir la clave
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
```

¹el texto en *azul* muestra los comandos que tiene que introducir el usuario. El texto *itálicas azules* representa comentarios.

Introducción a Mininet y Ryu

```
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> Mininet está preparado para interactuar con el usuario
```

Una vez arrancado, Mininet está listo para ejecutar comandos.

Para seguir, arranque Mininet en la máquina virtual y vaya ejecutando los comandos marcados en azul:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

La lista completa de comandos de que dispone Mininet se puede ver con el comando:

```
mininet> help
```

Para listar los nodos de la red el comando es nodes, para ver los enlaces se utiliza net y para ver la información relativa a **todos** los nodos de la red, el comando que se tiene que utilizar es dump.

Cuando se quiera ejecutar un comando dentro de equipo final el formato es

```
nombre host comando
```

Ejecute los siguientes ejemplos:

```
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.189 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.168 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.162 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.164 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.165 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4032ms
rtt min/avg/max/mdev = 0.162/0.169/0.189/0.017 ms
```

```
mininet> h1 ip route
10.0.0.0/8 dev h1-eth0 proto kernel scope link src 10.0.0.1
```

Para salir de Mininet, utilizar el comando quit.

Una vez de vuelta en la línea de comando, ejecutar el comando

```
sudo mn -c
```

para “limpiar” el entorno de cualquier configuración espúrea que no se haya eliminado al salir de Mininet. Esto sera **necesario** si se interrumpe Mininet con la tecla `Control-C` por algún motivo.²

Para finalizar esta parte de la práctica, finalice Mininet con el comando `exit` y limpie el entorno con el comando `sudo mn -c`

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 12 links
.....
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 9 hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Done
completed in 12.709 seconds
student@student-VirtualBox:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
...
```

1.2. Controlando el comportamiento de Mininet desde la línea de comando

A continuación, veremos una serie de opciones que permiten controlar el comportamiento de Mininet desde la línea de comando.

1.2.1. Topología de la red emulada

Mininet incluye una serie de topologías. Pruebe una topología de árbol sencilla:

Salga y, a continuación pruebe la siguiente especificación de topología.

```
sudo mn --topo tree,2,3
```

¿Cual es la diferencia?

Además de la topología `tree`, Mininet ofrece otras topologías simples como `linear` y topologías más complejas como `torus`³.

1.2.2. Controlando la asignación de direcciones MAC en los equipos finales

Ejecute Mininet con el siguiente comando:

²Cuando Mininet está esperando un comando `Control-C` no tiene efecto.

³Algunas topologías tienen **bucles**, con lo que no se deben ejecutar salvo que el controlador que se utilice implemente un mecanismo de supresión de bucles como, p.ej. el protocolo *Spanning Tree*.

```
sudo mn --topo linear,3
```

y averigue las direcciones MAC asignadas a los equipos finales con el comando

```
hX ip addr show
```

para **todos** los equipos finales⁴.

Repita el procedimiento con el siguiente comando:

```
sudo mn --topo linear,3 --mac
```

¿Cuales con las diferencias?

1.2.3. Tipo de conmutador desplegado

También se puede controlar el tipo de conmutador utilizado para las simulaciones. Los dos tipos más frecuentes son el conmutador en espacio de usuario `user` y el controlador basado en Open vSwitch (OVS) y ejecutado en espacio de kernel `ovsk`.

Para seleccionarlos, se incluye `---switch <tipo de switch>` en la línea de comandos. Repita los ejemplos del apartado anterior para los dos tipos de conmutador mencionados.

1.2.4. Versión de OpenFlow utilizada

Mininet también permite controlar la versión de OpenFlow que se utilizará en el plano de control. Esta funcionalidad tiene impacto sobre los comandos que se pueden utilizar en la línea de comando.

Lance Mininet con una topología tipo `linear,1,4`⁵ con el conmutador `ovsk`, incluyendo el parámetro `protocols=OpenFlow13`. Una vez en la línea de comandos, lance los comandos

```
sudo mn --topo linear,1,4 --controller ovsc --switch ovs,protocols=OpenFlow13
pingall
dpctl dump-flows
dpctl dump-flows -O OpenFlow13
```

¿Cuál es la diferencia en la salida de ambos `dpctl dump-flows`?

Repita esta operación **sin** el parámetro `,protocols=OpenFlow13`.

1.2.5. Acceso a los equipos finales desde terminales independientes

Siguiendo con la topología anterior, ejecute el comando:

```
xterm h1s1
```

⁴El comando se puede mejorar incluyendo `| grep ether` al final para aislar la línea que indica la dirección MAC de la interfaz Ethernet del equipo final

⁵`linear,x,y` equivale a una cadena de `x` conmutadores con `y` equipos terminales conectados a cada uno de ellos. Se puede omitir "y" si sólo se requiere un equipo final por conmutador

Verá como se abre un terminal con una línea de comando, ejecute el comando `ip addr show` para identificar la dirección IP de este equipo final.

Repita el proceso con el equipo final h2.

Desde el terminal del equipo final h1, intente lanzar el comando `ping` contra el equipo final h2.

¿Cómo tiene que indicar la dirección del equipo final h2? ¿Porqué?

Hito 1:

Llame a un profesor para que compruebe que ha ejecutado **todos** los comandos y que entiende la mecánica de funcionamiento de Mininet, comentando las preguntas resaltadas en **negrita**.

1.3. Programando Mininet en Python

Mininet define una API para Python que permite automatizar los procesos de interacción o definir topologías nuevas. En este laboratorio se utiliza el lenguaje Python en su versión 2, a pesar de estar siendo desfasada. Estamos a la espera que Mininet se adecue completamente Python3.

1.3.1. Automatización

Empezaremos por utilizar la API de Python para lanzar una serie de pruebas sobre una de las topologías predefinidas en Python. En el directorio `~/Devel`⁶, copie el siguiente código a un fichero y llámelo `mininet1.py`:

```
#!/usr/bin/env python3

from mininet.topo import SingleSwitchTopo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

def simpleTest():
    'Create and test a simple network'
    topo = SingleSwitchTopo(4)
    net = Mininet(topo)
    net.start()
    print('Dumping host connections')
    dumpNodeConnections(net.hosts)
    print('Testing network connectivity')
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

⁶La mejor manera de mantener un control sobre el trabajo es tener la estructura de ficheros organizada. Para evitar problemas, cree el directorio `$HOME/Devel` si no existe en la imagen de la VM y utilícelo para todo el código que tenga que probar

Una vez salvado, aplique permisos de ejecución a este fichero para todos los usuarios y ejecútelo con el comando sudo:

```
sudo ./mininet1.py
```

Debería ver la siguiente salida en el terminal:

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 1 controllers
c0
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

Hito 2:

¿Cómo es la topología que ha creado?

1.3.2. Lanzando la línea de comando de Mininet desde un programa

La API de Mininet permite lanzar la línea de comando de Mininet desde programa. En este segundo ejemplo vamos a utilizar la misma topología del ejemplo anterior y vamos a lanzar la CLI para interactuar con ella desde un programa. Copie el siguiente código a un fichero llamado `mininet2.py`:

```
#!/usr/bin/env python3

from mininet.topo import SingleSwitchTopo
from mininet.net import Mininet
from mininet.log import setLogLevel
from mininet.cli import CLI

def simpleTestCLI():
    topo = SingleSwitchTopo(4)
    net = Mininet(topo)
    net.start()
    CLI(net)
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTestCLI()
```

Aplicúele los permisos de ejecución y ejecútelo como **sudo**. Utilice la CLI para ver la topología generada y para comprobar la conectividad entre los eioqpos finales con el comando `pingall`.

Intente ejecutarlo con el parámetro `--mac` como se sugiere en la sección 1.2.2. ¿Se pueden controlar las direcciones MAC desde la CLI del sistema operativo?

1.3.3. Creación de topologías específicas

Mininet permite la creación de topologías de manera programática. En este ejemplo, creará la topología. Como último ejemplo de programación de Mininet, vamos a crear nosotros una versión de la topología **SingleSwitchTopo** incluida en las librerías de Mininet.

Copie el siguiente código a un fichero llamado `~/Devel/mininet3.py`:

```
#!/usr/bin/env python3

from mininet.net import Mininet
from mininet.topo import Topo
from mininet.log import setLogLevel
from mininet.cli import CLI

class SingleSwitchTopo(Topo):
    'My single switch connected to n hosts.'

    def build(self, N=2):
        switch = self.addSwitch('s1')
        for hn in range(N):
            host = self.addHost(f'h\{hn+1}')
            self.addLink(host, switch)

def simpleTestCLI():
    topo = SingleSwitchTopo(4)
```

```
net = Mininet(topo)
net.start()
CLI(net)
net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTestCLI()
```

Observe que en las línea de importación de librerías aparece únicamente la clase Topo que implementa topologías de manera genérica y que la topología SingleSwitchTopo se deriva de ella en su código (e.d. **no** se importa de las librerías).

La API de Mininet está detallada en <https://mininet.org/api/annotated.html>.

Busque la definición de la función `.addHost()` en la documentación de la clase Topo https://mininet.org/api/classmininet_1_1topo_1_1Topo.html. Observe cómo a continuación del nombre del equipo final, puede incluir más parámetros como la dirección IP o la dirección MAC. Estos parámetros son cadenas de caracteres. Observe en el ejemplo que se utiliza una cadena formateada (`f'h{hn+1}'`) para reear el nombre del equipo final.

Hito 3:

Implemente un programa para Mininet que emule el funcionamiento del parámetro `--mac` de la CLI de Mininet. Discuta su solución con el profesor.

2. El controlador Ryu

En el mundo SDN se separa el plano de control de los elementos de red del plano de datos que se encarga de interconectar, en última instancia, los equipos finales.

Ryu es un controlador SDN. Soporta varios tipos de planos de control como OF (en sus versiones entre 1.0 y 1.5 incluyendo las extensiones propuestas por Nicira) NetConf, OF-config, etc. La documentación completa se encuentra disponible en [2].

Nota

No se olvide de desinstalar el paquete `ovs-testcontroller` como se indica al principio del capítulo 1.

2.1. Interacción básica entre Ryu y la red

La preparación de un entorno básico para trabajar con Ryu en este laboratorio implica la utilización de Mininet para emular la presencia de una red. En nuestro caso, la red está basada en OpenFlow.

Abra dos terminales: en una ventana se ejecutará el controlador Ryu y en la otra Mininet con la red que deseamos emular. El orden de ejecución de los comandos no es crítico, aunque siempre se recomiendan seguir el siguiente orden:

1. Empezar por lanzar el controlador Ryu con la aplicación de control
2. Lanzar después Mininet
3. Al terminal, **no olvidar** limpiar Mininet con `sudo mn --c`

Aplicaciones de control incluidas en Ryu

El controlador Ryu tiene una serie de aplicaciones de base, cuyo código fuente se puede consultar en el directorio `/Install/ryu/ryu/apps`.

```
student@student-VirtualBox:~$ ls Install/ryu/ryu/app
bmpstation.py      rest_router.py      simple_switch_lacp_13.py
cbench.py          rest_topology.py    simple_switch_lacp.py
conf_switch_key.py rest_vtep.py        simple_switch.py
example_switch_13.py simple_monitor_13.py simple_switch_rest_13.py
gui_topology       simple_switch_12.py simple_switch_snort.py
__init__.py        simple_switch_13.py simple_switch_stp_13.py
ofctl              simple_switch_13.pyc simple_switch_stp.py
ofctl_rest.py      simple_switch_14.py simple_switch_websocket_13.py
rest_conf_switch.py simple_switch_15.py wsgi.py
rest_firewall.py   simple_switch_igmp_13.py ws_topology.py
rest_qos.py        simple_switch_igmp.py
```

Para utilizarlas, seleccione la aplicación, quítele la extensión (.py) y antepóngale el prefijo de clase `ryu.app`:

```
student@student-VirtualBox:~/Install/ryu$ ryu-manager ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 f6:97:2e:54:94:91 33:33:ff:54:94:91 1
packet in 1 ee:4d:c9:aa:0e:c6 33:33:00:00:00:16 2
packet in 1 ee:4d:c9:aa:0e:c6 33:33:00:00:00:16 2
packet in 1 ee:4d:c9:aa:0e:c6 33:33:00:00:00:02 2
packet in 1 ee:4d:c9:aa:0e:c6 33:33:00:00:00:16 2
packet in 1 f6:97:2e:54:94:91 33:33:00:00:00:16 1
packet in 1 f6:97:2e:54:94:91 33:33:00:00:00:02 1
packet in 1 f6:97:2e:54:94:91 33:33:00:00:00:16 1
packet in 1 ee:4d:c9:aa:0e:c6 33:33:00:00:00:02 2
packet in 1 f6:97:2e:54:94:91 33:33:00:00:00:02 1
packet in 1 f6:97:2e:54:94:91 ff:ff:ff:ff:ff:ff 1
packet in 1 ee:4d:c9:aa:0e:c6 f6:97:2e:54:94:91 2
packet in 1 f6:97:2e:54:94:91 ee:4d:c9:aa:0e:c6 1
packet in 1 f6:97:2e:54:94:91 ee:4d:c9:aa:0e:c6 1
packet in 1 ee:4d:c9:aa:0e:c6 33:33:00:00:00:02 2
packet in 1 f6:97:2e:54:94:91 33:33:00:00:00:02 1
```

Utilización del controlador Ryu en un escenario Mininet

Como se ha visto en el capítulo ??, Mininet incluye un controlador por defecto. Además, permite la utilización de un controlador externo. En nuestro caso utilizaremos el controlador Ryu instalado en la VM. Para ello hay que incluir los parámetros `--controller remote` en la línea de comando:

```
student@student-VirtualBox:~$ sudo mn --controller remote
[sudo] password for student:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> quit
*** Stopping 1 controllers
```

```
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 18.580 seconds
```

2.2. Interacción de Ryu en entornos Mininet programados

Hasta ahora hemos visto como crear una topología de manera programática a base de crear equipos finales (Host) y conmutadores Switch en una clase que extienda la clase `Topology`.

Para comprobar como hacerlo, tome el fichero **mininet3.py** que creó en la sección 1.3.3 y cópielo en un fichero llamado `mininet4.py`. A continuación, incluya:

1. las líneas: al final de la sección de importación de clases para poder utilizar
 - a) la clase `RemoteController`; esta clase es la utilizada por Mininet cuando se llama con el parámetro `---controller remote` desde la línea de comando;
 - b) la clase `OVSSwitch` para especificar el uso del conmutador OVS; y
 - c) la macro `partial` para simplificar la instanciación de las clases `RemoteController` y `OVSSwitch`.
2. sustituir la creación de la variable `net` por:

```
net = Mininet(topo,
               controller=partial(RemoteController, ip="127.0.0.1"),
               switch=partial(OVSSwitch, protocols="OpenFlow13"))
```

Como se puede apreciar, `partial` se utiliza, por un lado, para indicar que se utiliza el controlador remoto (`RemoteController`) y que está corriendo en el mismo ordenador que la instalcia de Mininet (`ip="127.0.0.1"`) y, por otro, para indicar que los conmutadores son de tipo OVS y que se utiliza el protocolo OpenFlow v1.3.

Lance el controlador Ryu con la clase `ryu.app.simple_switch_13` en una ventana y el script que acaba de crear en la otra.

Compruebe que puede ejecutar el comando `pingall` en la línea de comandos. Salga de la simulación y pare el controlador Ryu.

Nota

El controlador Ryu se parará tanto si se pulsa la tecla `Ctrl-C` en la ventana en que se está ejecutando o como si se procede a purgar la simulación de Mininet con el comando `sudo mn -c` en la ventana que se esté utilizando para Mininet.

A continuación lance el controlador Ryu con la aplicación `ryu.app.simple_switch_12` y observe cómo el controlador no logra establecer la sesión, porque las versiones de OpenFlow

que se utilizan en la red (v1.3) y que espera el controlador (v1.2) son distintas. En la ventana del controlador debería de ver:

```
student@student-VirtualBox:~/Devel$ ryu-manager ryu.app.simple_switch_12
loading app ryu.app.simple_switch_12
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_12 of SimpleSwitch12
instantiating app ryu.controller.ofp_handler of OFPHandler
unsupported version 0x4. If possible, set the switch to use one of the versions [3] on datapath
('127.0.0.1', 55504)
unsupported version 0x4. If possible, set the switch to use one of the versions [3] on datapath
('127.0.0.1', 55506)
unsupported version 0x4. If possible, set the switch to use one of the versions [3] on datapath
('127.0.0.1', 55508)
```

2.3. Programas de Ryu personalizados

Como último paso, vamos a examinar la estructura de una de las aplicaciones de Ryu que hemos utilizado en los apartados anteriores y modificarla para desarrollar nuestro propio programa.

Planificación previa al desarrollo

Dado que va a implementar un programa con funciones de controlador OpenFlow, siempre es conveniente que planifique un escenario (sencillo) para ir probando durante el desarrollo. Se sugiere que se cree una topología sencilla con un conmutador y unos pocos equipos finales (por ejemplo 4). Vamos a trabajar con OpenFlow13.

Utilice la aplicación que creo en la sección 2.2. Alternativamente, créese un script para lanzar Mininet con estas especificaciones y asegurándose que el controlador es remoto (---controller remote¹) para que intente hablar con la aplicación Ryu.

Desarrollo de la aplicación de Ryu

Empiece por copiar el fichero ~/install/ryu/ryu/apps/simple_switch_13.py al directorio ~/Devel. Cámbiele el nombre a my_app.py.

Abra el fichero my_app.py con un editor y elimine todo lo que esté después de la definición de la clase, cambie el nombre de la clase de SimpleSwitch13 a MyApp.

La definición de la clase debería quedar así:

```
MAC_BROADCAST = 'ff'+':ff'*5

class MyApp(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(MyApp, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

¹Esto basta, porque así intentará establecer una sesión TCP contra localhost:6633, que es el puerto por defecto de OF en el que estará escuchando la aplicación Ryu

Compruebe que el código es correcto con el comando:

```
ryu-manager ~/Devel/my_app.py
```

Posteriormente, añada los siguientes dos fragmentos de código al final del fichero y compruebe que se sigue siendo posible ejecutarlo:

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # install table-miss flow entry
    #
    # We specify NO BUFFER to max_len of the output action due to
    # OVS bug. At this moment, if we specify a lesser number, e.g.,
    # 128, OVS will send Packet-In with invalid buffer_id and
    # truncated packet data. In that case, we cannot output packets
    # correctly. The bug has been fixed in OVS v2.1.0.
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]

    match = parser.OFPMatch()
    self.add_flow(datapath, 0, match, actions)
```

```
def add_flow(self, datapath, priority, match, actions,
             buffer_id=None, idle_timeout=0, table_id=0):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    hard_timeout = 0

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    if buffer_id is not None:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                idle_timeout=idle_timeout,
                                hard_timeout=hard_timeout,
                                priority=priority,
                                table_id=table_id,
                                match=match, instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath,
                                idle_timeout=idle_timeout,
                                hard_timeout=hard_timeout,
                                priority=priority,
                                table_id=table_id,
                                match=match, instructions=inst)
```

El primer fragmento lo puede recuperar directamente del fichero `simple_switch_13.py` original del fichero. Su función es asegurar que los paquetes que no sean tratados por las tablas del conmutador sean enviados al controlador.

El segundo toma como base el código original, pero añade parámetros adicionales a la función que le pueden ser útiles en la práctica de desarrollo.

A continuación, crearemos el código que implementa la función de conmutación².

```
# If you hit this you might want to increase
# the "miss_send_length" of your switch
if ev.msg.msg_len < ev.msg.total_len:
    self.logger.debug(
```

²utilice igualmente el código del fichero `simple_switch_13.py` como base

```
f"packet truncated: only \{ev.msg.msg_len\} of \{ev.msg.total_len\} bytes")

msg = ev.msg
datapath = msg.datapath
ofproto = datapath.ofproto
parser = datapath.ofproto_parser
in_port = msg.match['in_port']

pkt = packet.Packet(msg.data)
eth = pkt.get_protocols(ether_types.ethernet)[0]

if eth.ethertype == ether_types.ETH_TYPE_LLDP:
    # ignore lldp packet
    return
dst = eth.dst
src = eth.src

dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})

self.logger.info(
    f"in: \{dpid\} <\{eth.ethertype:04x\}> \{src\} \{dst\} \{in_port\}")
# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = in_port
```

A continuación, implementaremos la funcionalidad de aprendizaje y programación de las tablas de conmutación. Primero, se guarda localmente la MAC origen y se la asocia con el puerto del conmutador por el que ha llegado el paquete:

```
print(self.mac_to_port[dpid])

if dst in self.mac_to_port[dpid]:
    print(" Encontrado un paquete conocido")
if dst == MAC_BROADCAST:
    print(" Encontrado paquete broadcast")

# learn path to source
# send packet to in_port if eth_src is the src MAC
actions = [parser.OFPACTIONOutput(in_port)]
```

Empiece por obtener la información del paquete que acaba de llegar al controlador desde el conmutador y descartar paquetes LLDP e IPv6³:

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug(
            f"packet truncated: \{ev.msg.msg_len\} of \{ev.msg.total_len\} bytes")
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ether_types.ethernet)[0]

    etype = eth.ethertype
    if etype in [ ether_types.ETH_TYPE_LLDP, 0x86dd ]:
        return
    dst = eth.dst
    src = eth.src
```

³Pregunta: ¿Cómo se sabe que el código filtra el tráfico IPv6?


```
dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})
```

Cree la regla para la tabla de conmutación que mandará los paquetes con la MAC destino igual a la MAC origen de este paquete por el puerto por el que llegó el paquete:

```
actions = [parser.OFPACTIONOutput(in_port)]
match = parser.OFPMATCH(eth_dst=src)
self.add_flow(datapath, 1000, match, actions)
```

Y, finalmente, mande el paquete que se recibió por todos los puertos del conmutación en modo FLOOD (inundación):

```
data = None
# verify if we have a valid buffer_id, if yes avoid to send packet_out
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath,
                          buffer_id=msg.buffer_id,
                          in_port=in_port,
                          actions=[parser.OFPACTIONOutput(ofproto.OFPP_FLOOD)],
                          data=data)

datapath.send_msg(out)
```

Hito 4:

Una vez funcione este código, estúdielo y modifíquelo para que se acerque más al comportamiento de un conmutador Ethernet real, asociando a las entradas en la tabla de conmutación un tiempo de vida. (Para poder probar su implementación, elija un tiempo de vida bajo, e.g. 10 segundos).

Pruebe a utilizar esta aplicación con una topología de más de un conmutador (e.g. tree, 3, 2).

¿Cuales son las principales deficiencias de esta implementación?

A. Repaso: ARP y conmutación a nivel 2

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

A.1. ARP

Recordemos el funcionamiento en modo normal de una tarjeta Ethernet. 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. Además, en el sentido de entrada, la tarjeta utiliza este registro para filtrar paquetes y quedarse sólo aquellos que tienen esta dirección en el campo de dirección MAC de destino.

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

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

Por tanto, para que un paquete IP (ver la figura A.1) sea aceptado por un equipo, bien para ser procesado o para ser encaminado hacia otro, se necesita que tenga la dirección MAC correcta (o la dirección de “broadcast” FF:FF:FF:FF:FF:FF).

Para averiguar la dirección MAC de un equipo en una red, existe el Address Resolution Protocol (ARP) o protocolo de resolución de direcciones. 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)

La figura la figura A.2 muestra el intercambio capturado

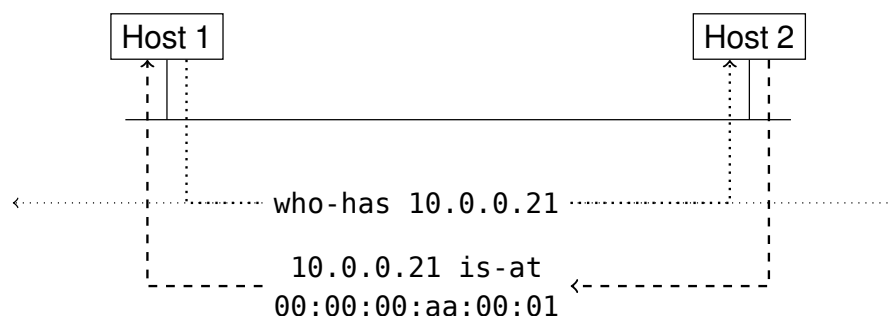


Figura A.2.: Intercambio de paquetes ARP

A continuación se muestran los paquetes capturados en el escenario de la figura la figura A.2. El equipo con dirección IP 10.0.0.20 (campo *psrc*) pregunta por la dirección MAC del equipo que tiene la dirección IP 10.0.0.21 (campo *pdst*). Observar cómo la dirección MAC destino (campo *dst*) es la dirección de *broadcast* (00:00:00:00:00:00):

```
###[ Ethernet ]###
  dst= ff:ff:ff:ff:ff:ff
  src= 00:00:00:aa:00:00
  type= ARP
###[ ARP ]###
  hwtype= 0x1
  ptype= IPv4
  hwlen= 6
  plen= 4
  op= who-has
  hwsrc= 00:00:00:aa:00:00
  psrc= 10.0.0.20
  hwdst= 00:00:00:00:00:00
  pdst= 10.0.0.21
```

En la respuesta capturada el equipo con dirección IP 10.0.0.21 (campo *psrc*) informa que su dirección MAC es 00:00:00:aa:00:01 (campo *hwsrc*):

```
###[ Ethernet ]###
  dst= 00:00:00:aa:00:00
  src= 00:00:00:aa:00:01
  type= ARP
###[ ARP ]###
  hwtype= 0x1
  ptype= IPv4
  hwlen= 6
  plen= 4
  op= is-at
  hwsrc= 00:00:00:aa:00:01
  psrc= 10.0.0.21
  hwdst= 00:00:00:aa:00:00
  pdst= 10.0.0.20
```

A.2. Conmutación a nivel 2

Un conmutador de nivel 2 conmuta paquetes en función de la dirección MAC de destino de un paquete. Es un dispositivo complejo que incluye mecanismos para supresión de bucles lógicos en la red que podrían causar tormentas de paquetes, esto es la replicación descontrolada de paquetes en la red. Al mismo tiempo, permite la existencia de bucles a nivel físico. Esta funcionalidad se implementa con el protocolo **Spanning Tree Protocol (STP)**.

El conmutador tiene una tabla en la que se asocia una dirección MAC al puerto por el que se le tienen que mandar los paquetes y un tiempo de vida para cada entrada. Cuando llega un paquete Ethernet, ocurren las siguientes cosas:

1. Se añade (o refresca) una entrada en la tabla que asocia la **MAC origen** del paquete con el puerto del conmutador por el que llegó
2. Se comprueba si existe una entrada en dicha tabla que asocie la MAC destino con un puerto.

- a) Si existe, se envía el paquete por dicho puerto.
- b) Si no, se mandan réplicas del paquete por todos los puertos activos del conmutador¹.

Los tiempos de vida en la tabla se van decreciendo (normalmente a un ritmo de una unidad por segundo) y cuando llegan a cero, se retira la entrada de la tabla.

¹El protocolo STP es el encargado de activar y desactivar los puertos.

B. Trucos y errores frecuentes

B.1. Errores fáciles de evitar en Mininet

Cuando escriba un escenario de Mininet en un programa Python, evite utilizar el nombre `mininet.py` o simplemente `mininet`. Este nombre confunde al intérprete de Python que piensa que el fichero implementa el módulo `mininet`.

De manera que, por ejemplo, la directiva

```
from mininet.topo import Topo
```

Se interpreta como que en este directorio hay un directorio llamado `topo` en el que hay un fichero `Topo.py` que implementa la clase `Topo`.

B.2. Ajustando las clases de Mininet a nuestras necesidades

La clase base de cualquier nodo en una topología de Mininet es `Host`. Según la documentación, a la hora de la instanciación, a esta clase se le tiene que definir el nombre del nodo y se le pueden ajustar los siguientes parámetros:

Característica	parámetro para <code>.addHost()</code>
dirección IP	<code>ip='10.0.0.10/24'</code>
dirección MAC	<code>mac='70:00:01:02:03:04'</code>
ruta por defecto	<code>defaultRoute='via 10.0.0.1'</code>

Tabla B.1.: Parámetros que se pueden pasar en la creación de un equipo final en Mininet

Para otros parámetros hay que extender la clase. Un ejemplo de ello está en la instalación de mininet de su VM en el fichero `vlanhost.py` en el directorio de ejemplos.

Acrónimos

API	Application Programming Interface
ARP	Address Resolution Protocol
CLI	Línea de comando
OF	OpenFlow
OVS	Open vSwitch
SDN	Software Defined Networking
STP	Spanning Tree Protocol
VM	Máquina Virtual
VirtualBox	

Referencias

- [1] *Mininet: An Instant Virtual Network on your Laptop*. URL: <https://mininet.org>.
- [2] *Ryu documentation*. URL: <https://ryu.readthedocs.io/en>.