# Mininet Lab 3: WiFi networks

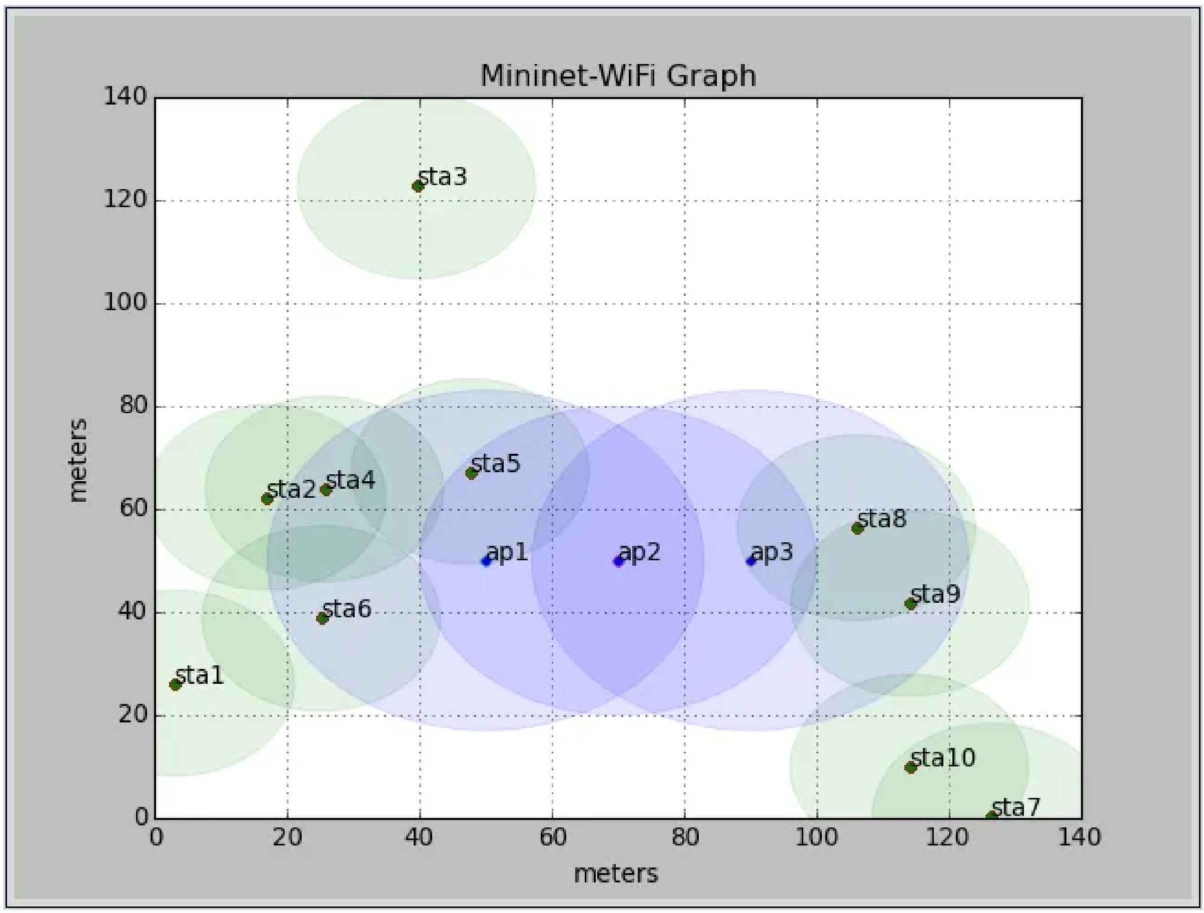
**tags: RSE Labs**

> http://www.brianlinkletter.com/mininet-wifi-software-defined-network-emulator-supports-wifi-networks/ (http://www.brianlinkletter.com/mininet-wifi-software-defined-network-emulator-supports-wifi-networks/)

## Overview on Mininet-WiFi

Mininet-WiFi (https://github.com/intrig-unicamp/mininet-wifi) is a fork of the Mininet project. The Mininet-WiFi developers extended the functionality of Mininet by adding virtualized WiFi stations and access points based on the standard Linux wireless drivers and the 80211_hwsim (https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim) wireless simulation driver.

They also added classes to support the addition of these wireless devices in a Mininet network scenario and to emulate the attributes of a mobile station such as position and movement relative to the access points.



The Mininet-WiFi extended the base Mininet code by adding or modifying classes and scripts. So, Mininet-WiFi adds new functionality and **still supports all the normal SDN emulation** capabilities of the standard Mininet network emulator.

Please refer to the Mininet-WiFi documentation for up-to-date information about this project. At the top of that page is a link to the Mininet-WiFi manual, which is currently hosted at: https://github.com/ramonfontes/manual-mininet-wifi/raw/master/mininet-wifi-draft-manual.pdf (https://github.com/ramonfontes/manual-mininet-wifi/raw/master/mininet-wifi-draft-manual.pdf).

## 802.11 Wireless LAN Emulation

Mininet-wifi incorporates the Linux 802.11 SoftMAC (https://wireless.wiki.kernel.org/en/developers/documentation/glossary#softmac) wireless drivers, the cfg80211 (https://www.kernel.org/doc/html/v4.12/driver-api/80211/cfg80211.html) wireless configuration interface and the mac80211_hwsim (https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim) wireless simulation drivers in its access points.

The **mac80211hwsim** driver is a software simulator for Wi-Fi radios. It can be used to create virtual Wi-Fi interfaces that use the 802.11 SoftMAC wireless LAN driver. Using this tool, it is possible to emulate a Wi-Fi link between virtual machines.
This driver enables to emulate the wifi protocol control messages passing between virtual wireless access points and virtual mobile stations in a network emulation scenario.
By default, it simulates perfect conditions, which means **there is no packet loss or corruption**.

You can use Wireshark to monitor wireless traffic passing between the virtual wireless access point and the virtual mobile stations in the Mininet-wifi network scenarios, but, you will find it is difficult to capture wireless control traffic on standard WLAN interfaces like `ap1-wlan0` because the **Linux kernel strips wireless control messages and headers before making traffic on these interfaces available to user processes** like Wireshark.

This issue is solved using the so-called **hwsim0 interface**. This interface replays communications sent onto the access point's simulated wireless interface(s) such as ap1-wlan0 without stripping any 802.11 headers or control traffic3. We'll see how to do this in the examples below.

## Topics covered in this lab

This lab session will present the basic functionalities of Mininet-WiFi by working through two steps:

**Step 1**: One access point shows how to run the simplest Mininet-WiFi scenario, shows how to capture wireless traffic in a Mininet-Wifi network.

**Step 2**: Multiple access points shows how to create a more complex network topology so we can experiment with a very basic mobility scenario.

## Step 1: One access point

The simplest network is the default topology, which consists of a wireless access point with two wireless stations. The access point is a switch connected to a controller. The stations are hosts.

This simple lab will allow us to demonstrate how to capture wireless control traffic and will demonstrate the way an OpenFlow-enabled access point handles WiFi traffic on the wlan interface.

### Capturing Wireless control traffic in Mininet-WiFi

To view wireless control traffic we must first start Wireshark:

```
$ sudo wireshark
```

Then, start Mininet-WiFi with the default network scenario using the command below:

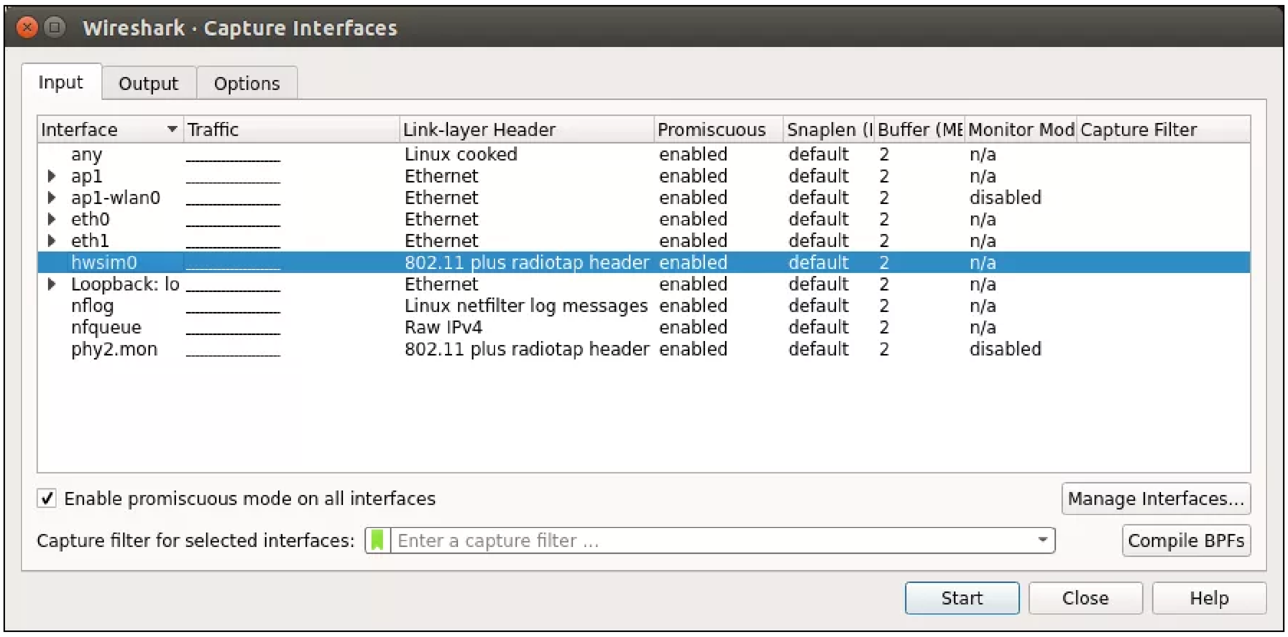
```
$ sudo mn --wifi
```

1.- Usando `dump` y `net` determina la estructura de la red creada.

Next, **enable the hwsim0 interface**. The hwsim0 interface is the software interface created by Mininet-WiFi that copies all wireless traffic to all the virtual wireless interfaces in the network scenario. It is the easiest way to monitor the wireless packets in Mininet-WiFi.

```
mininet-wifi> sh ifconfig hwsim0 up
```

Now, in Wireshark, refresh the interfaces and then start capturing packets on the `hwsim0` interface.



| Interface | Traffic | Link-layer Header | Promiscuous | Snaplen (I | Buffer (ME | Monitor Mod | Capture Filter |
|---|---|---|---|---|---|---|---|
| any | | Linux cooked | enabled | default | 2 | n/a | |
| ap1 | | Ethernet | enabled | default | 2 | n/a | |
| ap1-wlan0 | | Ethernet | enabled | default | 2 | disabled | |
| eth0 | | Ethernet | enabled | default | 2 | n/a | |
| eth1 | | Ethernet | enabled | default | 2 | n/a | |
| hwsim0 | | 802.11 plus radiotap header | enabled | default | 2 | n/a | |
| Loopback: lo | | Ethernet | enabled | default | 2 | n/a | |
| nflog | | Linux netfilter log messages | enabled | default | 2 | n/a | |
| nfqueue | | Raw IPv4 | enabled | default | 2 | n/a | |
| phy2.mon | | 802.11 plus radiotap header | enabled | default | 2 | disabled | |

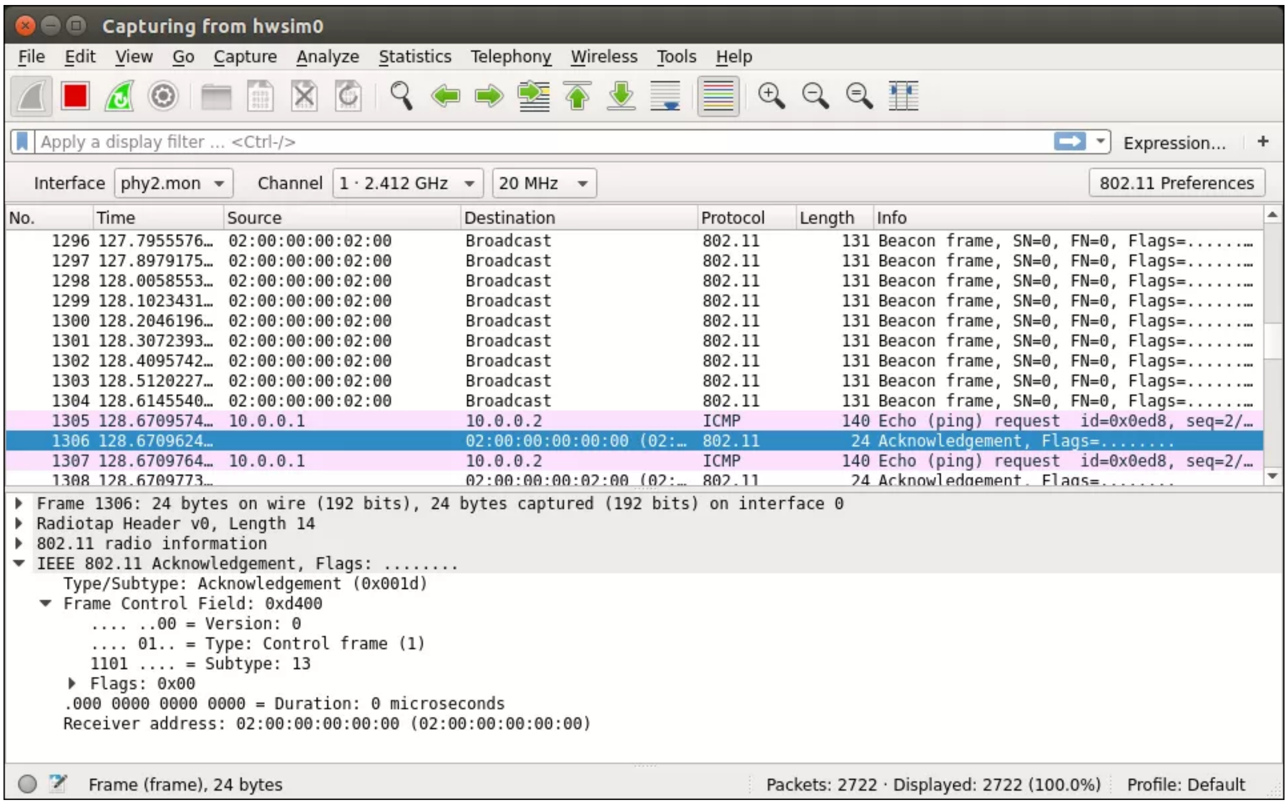
[✓] Enable promiscuous mode on all interfaces

Capture filter for selected interfaces: Enter a capture filter ...

You should see wireless control traffic. Next, execute a ping command:

```
mininet-wifi> sta1 ping sta2
```

In Wireshark, see the wireless frames and the ICMP packets encapsulated in Wireless frames passing through the hwsim0 interface.


Capturing from hwsim0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
802.11 Preferences

Interface phy2.mon   Channel 1 · 2.412 GHz   20 MHz

```
No.   Time              Source              Destination              Protocol  Length Info
 1296 127.7955576…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1297 127.8979175…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1298 128.0058553…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1299 128.1023431…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1300 128.2046196…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1301 128.3072393…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1302 128.4095742…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1303 128.5120227…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1304 128.6145540…  02:00:00:00:02:00   Broadcast                802.11     131 Beacon frame, SN=0, FN=0, Flags=.......…
 1305 128.6709574…  10.0.0.1            10.0.0.2                 ICMP       140 Echo (ping) request  id=0x0ed8, seq=2/…
 1306 128.6709624…                      02:00:00:00:00:00 (02:…  802.11      24 Acknowledgement, Flags=........
 1307 128.6709764…  10.0.0.1            10.0.0.2                 ICMP       140 Echo (ping) request  id=0x0ed8, seq=2/…
 1308 128.6709773…                      02:00:00:00:02:00 (02:…  802.11      24 Acknowledgement, Flags=........

▸ Frame 1306: 24 bytes on wire (192 bits), 24 bytes captured (192 bits) on interface 0
▸ Radiotap Header v0, Length 14
▸ 802.11 radio information
▾ IEEE 802.11 Acknowledgement, Flags: ........
    Type/Subtype: Acknowledgement (0x001d)
  ▾ Frame Control Field: 0xd400
      .... ..00 = Version: 0
      .... 01.. = Type: Control frame (1)
      1101 .... = Subtype: 13
    ▸ Flags: 0x00
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: 02:00:00:00:00:00 (02:00:00:00:00:00)
```

Frame (frame), 24 bytes   Packets: 2722 · Displayed: 2722 (100.0%)   Profile: Default

Stop the ping command by pressing Ctrl-C. .

## Wireless Access Points and OpenFlow

In this simple scenario, the access point has only one interface, ap1-wlan0. By default, stations associated with an access point connect in infrastructure mode so wireless traffic between stations must pass through the access point. If the access point works similarly to a switch in standard Mininet, we expect to see OpenFlow messages exchanged between the access point and the controller whenever the access point sees traffic for which it does not already have flows established.

To view OpenFlow packets, stop the Wireshark capture and **switch to the loopback interface**. Start capturing again on the loopback interface. Use the *openFlow_v1* filter to view only OpenFlow messages.

Then, start some traffic running with the ping command and look at the OpenFlow messages captured in Wireshark.

```
mininet-wifi> sta1 ping sta2
```

You can see some of the messages flowing through.

Stop the Mininet ping command by pressing Ctrl-C.

In the Wireshark window, stop capturing and quit Wireshark.

Stop Mininet-Wifi and clean up the system with the following commands:

```
mininet-wifi> exit
wifi:~$ sudo mn -c
```

## Mininet-WiFi step #2: Multiple access points

When we create a network scenario with two or more wireless access points, we can show more of the functions available in Mininet-WiFi.

In this step, we will create a linear topology with three access points, where one station is connected to each access point.

Run Mininet-Wifi and create a linear topology with three access points:

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

From the output of the command, we can see how the network is set up and which stations are associated with which access points.

```
*** Creating network
*** Adding controller
*** Adding hosts and stations:
sta1 sta2 sta3
*** Adding switches and access point(s):
ap1 ap2 ap3
*** Adding links and associating station(s):
(ap2, ap1) (ap3, ap2) (sta1, ap1) (sta2, ap2) (sta3, ap3)
*** Starting controller(s)
c0
*** Starting switches and access points
ap1 ap2 ap3 ...
*** Starting CLI:
mininet-wifi>
```

We can also verify the configuration using the Mininet CLI commands `net` and `dump`

```
mininet-wifi> net
sta1 sta1-wlan0:None
sta2 sta2-wlan0:None
sta3 sta3-wlan0:None
ap1 lo:  ap1-eth1:ap2-eth1
ap2 lo:  ap2-eth1:ap1-eth1 ap2-eth2:ap3-eth1
ap3 lo:  ap3-eth1:ap2-eth2
c0
```

From the net command above, we see that ap1, ap2, and ap3 are connected together in a linear fashion by Ethernet links. But, we do not see any information about to which access point each station is connect. This is because they are connected over a “radio” interface so we need to run the standard **iw** command at each station to observe to which access point each is associated.

To check which access points are “visible” to each station, use the **iw scan** command:

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan | grep ssid
        SSID: ssid_ap1
        SSID: ssid_ap2
        SSID: ssid_ap3
```

Verify the access point to which each station is currently connected with the **iw link** command. For example, to see the access point to which station sta1 is connected, use the following command:

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
        SSID: ssid_ap1
        freq: 2412
        RX: 1853238 bytes (33672 packets)
        TX: 7871 bytes (174 packets)
        signal: -30 dBm
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi>
```

## A simple mobility scenario

In this example, each station is connected to a different wireless access point. We can use the iw command to change which access point to which each station is connected.

Let’s decide we want sta1, which is currently associated with ap1, to change its association to ap2. Manually switch the sta1 association from ap1 (which is ssid_ap1) to ap2 (which is ssid_ap2) using the following commands:

```
mininet-wifi> sta1 iw dev sta1-wlan0 disconnect
mininet-wifi> sta1 iw dev sta1-wlan0 connect ssid_ap2
```

2.- verifica el cambio con el comando **iw link** ¿Que obtienes?

So we’ve demonstrated a basic way to make stations mobile, where they switch their association from one access point to another.

## OpenFlow flows in a mobility scenario

Now let's see how the Mininet reference controller handles this simple mobility scenario.

We need to get some traffic running from sta1 to sta3 in a way that allows us to access the Mininet-WiFi command line. We'll run the ping command in an xterm window on sta3.

First, check the IP addresses on sta1 and sta3 so we know which parameters to use in our test. The easiest way to see all IP addresses is to run the **dump** command:

```
mininet-wifi> dump
<Host sta1: sta1-wlan0:10.0.0.1 pid=7091>
<Host sta2: sta2-wlan0:10.0.0.2 pid=7094>
<Host sta3: sta3-wlan0:10.0.0.3 pid=7097>
<OVSSwitch ap1: lo:127.0.0.1,ap1-eth1:None pid=7106>
<OVSSwitch ap2: lo:127.0.0.1,ap2-eth1:None,ap2-eth2:None pid=7110>
<OVSSwitch ap3: lo:127.0.0.1,ap3-eth1:None pid=7114>
<Controller c0: 127.0.0.1:6633 pid=7080>
mininet-wifi>
```

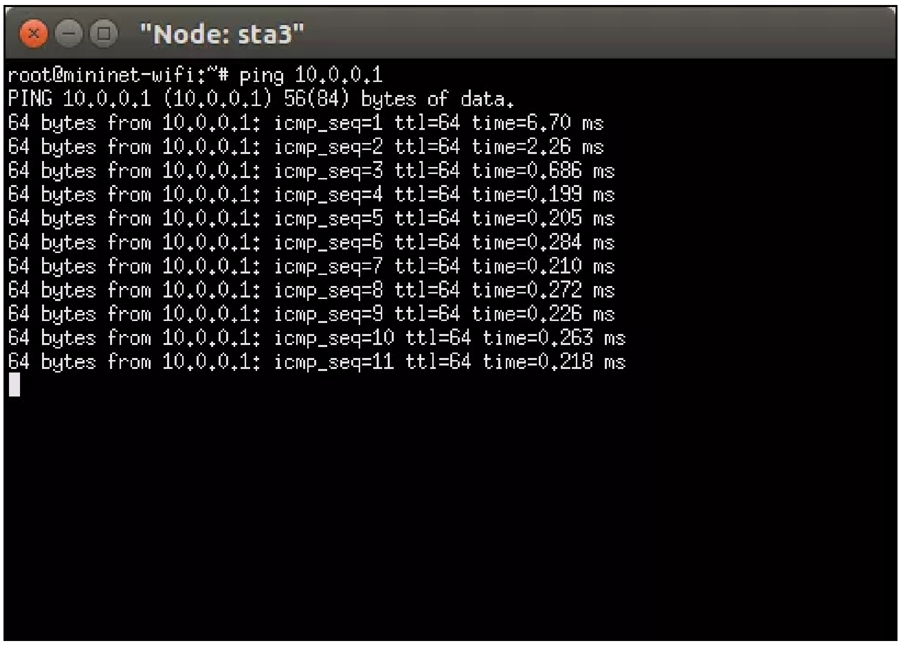
So we see that sta1 has IP address 10.0.0.1 and sta3 has IP address 10.0.0.3.

Next, start an xterm window on sta3:

```
mininet-wifi> xterm sta3
```

In that window, run the following command to send ICMP messages from sta3 to sta1:

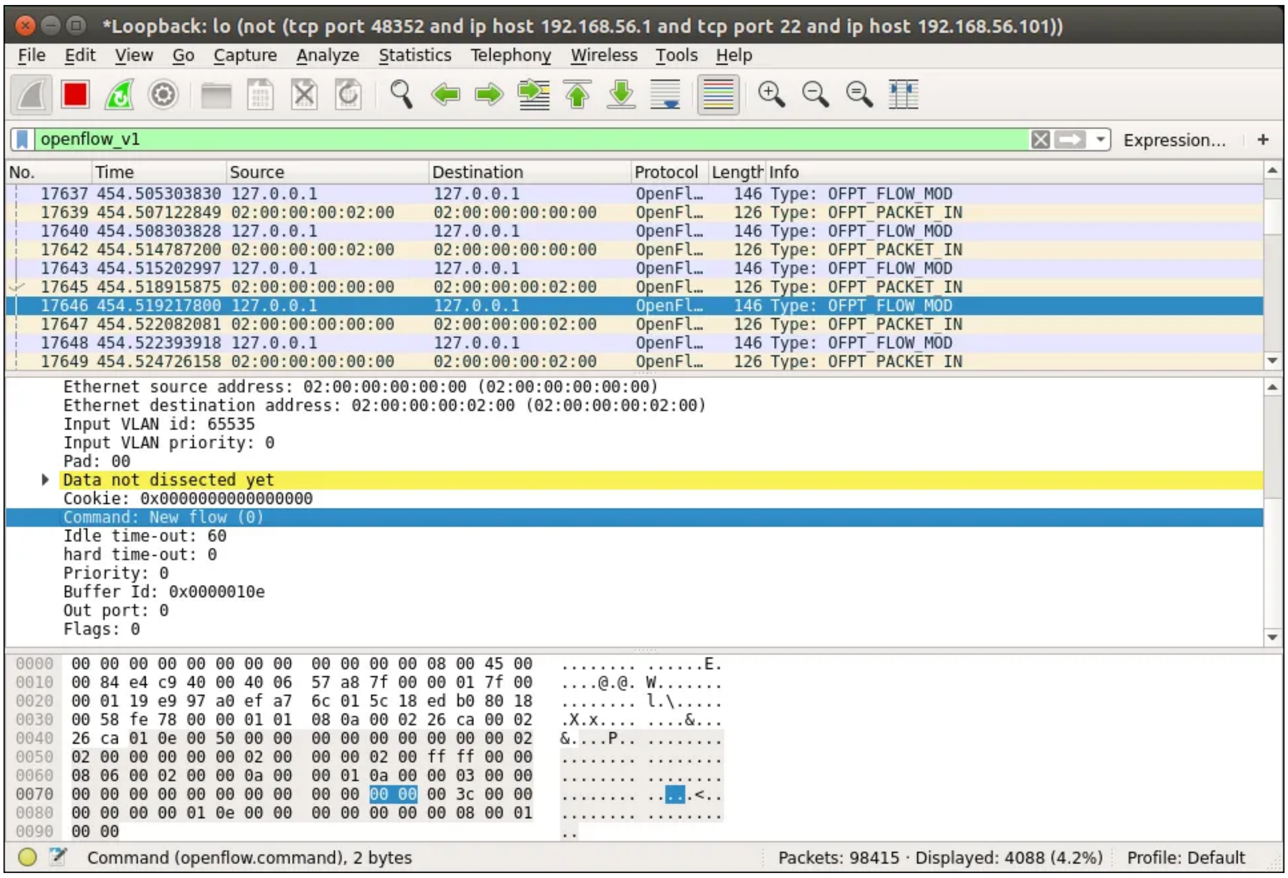
```
# ping 10.0.0.1
```



```
root@mininet-wifi:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=6.70 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=2.26 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.686 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.199 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.205 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.284 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.210 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.272 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.226 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.263 ms
64 bytes from 10.0.0.1: icmp_seq=11 ttl=64 time=0.218 ms
```

Since these packets will be forwarded by the associated access points out a port other then the port on which the packets were received, the access points will operate like normal OpenFlow-enabled switches. Each access point will forward the first ping packet it receives in each direction to the Mininet reference controller. The controller will set up flows on the access points to establish a connection between the stations sta1 and sta3.

If we run Wireshark and enable packet capture on the Loopback interface, then filter using openflow_v1 we will see OpenFlow messages passing to and from the controller.


*Loopback: lo (not (tcp port 48352 and ip host 192.168.56.1 and tcp port 22 and ip host 192.168.56.101))
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
openflow_v1
Expression...

```
No.   Time           Source             Destination        Protocol Length Info
17637 454.505303830 127.0.0.1           127.0.0.1          OpenFl…  146 Type: OFPT_FLOW_MOD
17639 454.507122849 02:00:00:00:02:00   02:00:00:00:00:00  OpenFl…  126 Type: OFPT_PACKET_IN
17640 454.508303828 127.0.0.1           127.0.0.1          OpenFl…  146 Type: OFPT_FLOW_MOD
17642 454.514787200 02:00:00:00:02:00   02:00:00:00:00:00  OpenFl…  126 Type: OFPT_PACKET_IN
17643 454.515202997 127.0.0.1           127.0.0.1          OpenFl…  146 Type: OFPT_FLOW_MOD
17645 454.518915875 02:00:00:00:00:00   02:00:00:00:02:00  OpenFl…  126 Type: OFPT_PACKET_IN
17646 454.519217800 127.0.0.1           127.0.0.1          OpenFl…  146 Type: OFPT_FLOW_MOD
17647 454.522082081 02:00:00:00:00:00   02:00:00:00:02:00  OpenFl…  126 Type: OFPT_PACKET_IN
17648 454.522393918 127.0.0.1           127.0.0.1          OpenFl…  146 Type: OFPT_FLOW_MOD
17649 454.524726158 02:00:00:00:00:00   02:00:00:00:02:00  OpenFl…  126 Type: OFPT_PACKET_IN
```

```
    Ethernet source address: 02:00:00:00:00:00 (02:00:00:00:00:00)
    Ethernet destination address: 02:00:00:00:02:00 (02:00:00:00:02:00)
    Input VLAN id: 65535
    Input VLAN priority: 0
    Pad: 00
  ▸ Data not dissected yet
    Cookie: 0x0000000000000000
    Command: New flow (0)
    Idle time-out: 60
    hard time-out: 0
    Priority: 0
    Buffer Id: 0x0000010e
    Out port: 0
    Flags: 0
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 84 e4 c9 40 00 40 06  57 a8 7f 00 00 01 7f 00   ....@.@. W.......
0020  00 01 19 e9 97 a0 ef a7  6c 01 5c 18 ed b0 80 18   ........ l.\.....
0030  00 58 fe 78 00 00 01 01  08 0a 00 02 26 ca 00 02   .X.x.... ....&...
0040  26 ca 01 0e 00 50 00 00  00 00 00 00 00 00 00 02   &....P.. ........
0050  02 00 00 00 00 00 02 00  00 00 02 00 ff ff 00 00   ........ ........
0060  08 06 00 02 00 00 0a 00  00 01 0a 00 00 03 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 3c 00 00   ........ .....<..
0080  00 00 00 00 01 0e 00 00  00 00 00 00 00 08 00 01   ........ ........
0090  00 00                                              ..
```

Command (openflow.command), 2 bytes    Packets: 98415 · Displayed: 4088 (4.2%)    Profile: Default

Now, in the Mininet CLI, check the flows on each switch with the `dpctl dump-flows` command.

```
mininet-wifi> dpctl dump-flows
*** ap1 ------------------------------------------------
NXST_FLOW reply (xid=0x4):
*** ap2 ------------------------------------------------
NXST_FLOW reply (xid=0x4):
idle_timeout=60, idle_age=0, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:
 cookie=0x0, duration=1068.17s, table=0, n_packets=35, n_bytes=1470, idle_timeout=60, i
 cookie=0x0, duration=1073.174s, table=0, n_packets=1073, n_bytes=105154, idle_timeout=
 cookie=0x0, duration=1073.175s, table=0, n_packets=1073, n_bytes=105154, idle_timeout=
*** ap3 ------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1068.176s, table=0, n_packets=35, n_bytes=1470, idle_timeout=60,
idle_timeout=60, idle_age=0, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:
 cookie=0x0, duration=1073.182s, table=0, n_packets=1073, n_bytes=105154, idle_timeout=
 cookie=0x0, duration=1073.185s, table=0, n_packets=1073, n_bytes=105154, idle_timeout=
mininet-wifi>
```

We see flows set up on ap2 and ap3, but not on ap1. This is because sta1 is connected to ap2 and sta3 is connected to ap3 so all traffic is passing through only ap2 and ap3.

What will happen if sta1 moves back to ap1?

3.- Muove sta1 otra vez al punto de acceso ap1. ¿Que secuencia de comandos tienes que utilizar?

The ping command running on sta3 stops working. We see no more pings completed.

In this case, access points ap2 and ap3 already have flows for ICMP messages coming from sta3 so they just keep sending packets towards the ap2-wlan0 interface to reach where they think sta1 is connected. **Since ping messages never get to sta1 in its new location, the access point ap1 never sees any ICMP traffic so does not request any flow updates from the controller.**

Check the flow tables in the access points again:

```
mininet-wifi> dpctl dump-flows
*** ap1 ------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=40.959s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_
 cookie=0x0, duration=40.958s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_
*** ap2 ------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=40.968s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_
 cookie=0x0, duration=40.964s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_
 cookie=0x0, duration=1214.279s, table=0, n_packets=1214, n_bytes=118972, idle_timeout=
*** ap3 ------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=40.978s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_
 cookie=0x0, duration=40.971s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_
 cookie=0x0, duration=1214.288s, table=0, n_packets=1214, n_bytes=118972, idle_timeout=
mininet-wifi>
```

The controller sees some LLC messages from sta1 but does recognize that sta1 has moved to a new access point, so it does nothing. Since the controller does not modify any flows in the access points, none of the ICMP packets still being generated by sta3 will reach sta1 so it cannot reply. This situation will remain as long as the access points ap2 and ap3 continue to see ICMP packets from sta3, which keeps the old flow information alive in their flow tables.

One “brute force” way to resolve this situation is to delete the flows on the switches. In this simple example, it’s easier to just delete all flows.

Delete the flows in the access points using the command below:

```
mininet-wifi> dpctl del-flows
```

Now the ping command running in the xterm window on sta3 should show that pings are being completed again.

Once all flows were deleted, ICMP messages received by the access points do not match any existing flows so the access points communicate with the controller to set up new flows. If we dump the flows we see that the ICMP packets passing between sta3 and sta1 are now traversing across all three acces points.

```
mininet-wifi> dpctl dump-flows
*** ap1 -----------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=10.41s, table=0, n_packets=11, n_bytes=1078, idle_timeout=60, idl
 cookie=0x0, duration=9.41s, table=0, n_packets=10, n_bytes=980, idle_timeout=60, idle_
*** ap2 -----------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=10.414s, table=0, n_packets=11, n_bytes=1078, idle_timeout=60, id
 cookie=0x0, duration=9.417s, table=0, n_packets=10, n_bytes=980, idle_timeout=60, idle
*** ap3 -----------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=10.421s, table=0, n_packets=11, n_bytes=1078, idle_timeout=60, id
 cookie=0x0, duration=9.427s, table=0, n_packets=10, n_bytes=980, idle_timeout=60, idle
mininet-wifi>
```

We have shown how the Mininet reference controller works in Mininet-WiFi. The Mininet reference controller does not have the ability to detect when a station moves from one access point to another. When this happens, we must delete the existing flows so that new flows can be created. We will need to us a more advanced remote controller, such as OpenDaylight, to enable station mobility but that is a topic outside the scope of this session.

Stop the Mininet ping command by pressing Ctrl-C.

In the Wireshark window, stop capturing and quit Wireshark.

Stop Mininet-Wifi and clean up the system with the following commands:

```
mininet-wifi> exit
wifi:~$ sudo mn -c
```

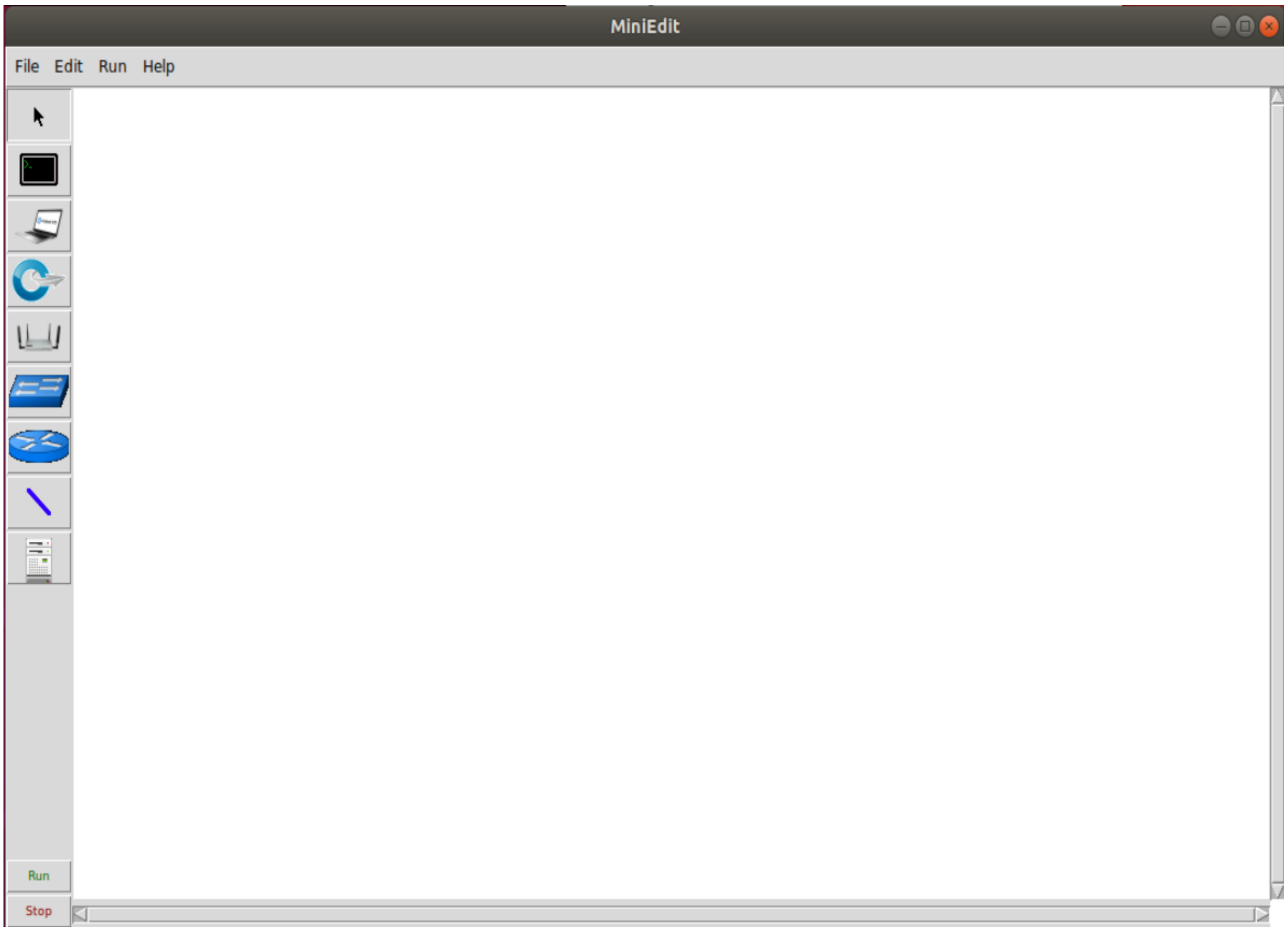
4.- Como ejercio final, ejecuta:

```
$ sudo ~/RSEmininet/mininet-wifi/examples/miniedit.py
```

esto hace que aparezca la version adaptada a WiFi del tool Miniedit que has utilizado en la sesion anterior. Define una topologia similar a la del comando

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

y genera el codigo .py


MiniEdit
File
Edit
Run
Help
Run
Stop