

# Memoria Mininet-Wifi

David Carrascal Acebrón

Mayo del 2019



# Índice general

<b>1. Mininet-Wifi</b>	<b>5</b>
1.1. Enlaces útiles	5
1.2. Introducción teórica	5
1.2.1. ¿Qué es?	5
1.2.2. ¿Qué es el término RSSI?	5
1.3. Instalación	6
1.4. Mininet-Wifi CLI	6
1.4.1. Comando: EOF + quit + exit	6
1.4.2. Comando: distance	7
1.4.3. Comando: dpctl	7
1.4.4. Comando: dump + net	7
1.4.5. Comando: xterm y gterm	7
1.4.6. Comando: help	8
1.4.7. Comando: intfs + nodes + ports	8
1.4.8. Comando: iperf + iperfudp	9
1.4.9. Comando: links	11
1.4.10. Comando: noecho	11
1.4.11. Comando: pingall + pingallfull	11
1.4.12. Comando: pingpair + pingpairfull	12
1.4.13. Comando: px + py	12
1.4.14. Comando: sh	12
1.4.15. Comando: source	13
1.4.16. Comando: start + stop	13
1.4.17. Comando: switch	14
1.4.18. Comando: time	14
1.4.19. Comando: x	14
1.5. Test 1: Escenario simple, unico punto de acceso	15
1.5.1. Capturando tráfico de control wireless en Mininet-Wifi	15
1.5.2. Puntos de acceso Wireless y OpenFlow	17
1.6. Test 2: Multiples puntos de acceso	18
1.6.1. Asociaciones con los puntos de acceso	20
1.6.2. Flows OpenFlow en un escenario de Handover	21
1.7. Test 3: Python API y scripts	25
1.7.1. Métodos básicos API Mininet-Wifi	25
1.7.2. Métodos básicos API Mininet	26
1.7.3. Ejemplo script Mininet-Wifi	27
1.7.4. Cambiando la red durante el tiempo de ejecución	29
1.8. Test 4	30
1.8.1. Python API y métodos de movilidad	30
1.8.2. Ejemplo script movilidad	31
1.8.3. Prueba con la herramienta Iperf	32
<b>Anexos</b>	<b>37</b>

<b>A. Herramienta IPerf</b>	<b>39</b>
A.1. Opciones . . . . .	39
A.2. Ejemplo de uso . . . . .	40

# Capítulo 1

## Mininet-Wifi

Por lo que he leído Mininet-Wifi no incorpora el estándar que usábamos para el estudio de las redes de área personal con tasa baja de transmisión de datos, 802.15.4 .

### 1.1. Enlaces útiles

- Articulos sobre Mininet y Mininet-Wifi: <http://www.brianlinkletter.com/tag/mininet/>

### 1.2. Introducción teórica

#### 1.2.1. ¿Qué es?

Mininet-Wifi es un fork del proyecto Mininet con el que podías emular redes SDN, y se han extendido sus funcionalidades al ámbito de las redes wireless. Por lo que con el se pueden combinar ambas funcionalidades, las anteriores comunes a Mininet y las nuevas que incorpora Mininet-Wifi.

#### 1.2.2. ¿Qué es el término RSSI?

El indicador de fuerza de la señal recibida (RSSI por las siglas del inglés Received Signal Strength Indicator), es una escala de referencia **en relación a 1 mW** para medir el nivel de potencia de las señales recibidas por un dispositivo en las redes inalámbricas (típicamente WIFI o telefonía móvil).

RSSI	Descripción
0	Señal ideal
-40	Señal idónea con tasas de transferencia estables.
-60	Enlace bueno, ajustando la transmisión (Tx) se puede lograr una conexión estable al 80 %
-70	Enlace medio-bajo, es una señal medianamente buena aunque se pueden sufrir problemas con lluvia y viento.
-80	Es la señal mínima aceptable para establecer la conexión. Pueden ocurrir caídas que se traducen en corte de comunicación

### 1.3. Instalación

Instalar Mininet-Wifi me ha resultado muy sencillo. Se lleva a cabo vía un shellsript que te dan ya hecho al clonar el repositorio de Mininet-Wifi. Los pasos que he seguido en la instalación de Mininet-Wifi son:

- Instalar una máquina virtual con Ubuntu 16.04
- Añadir git, sudo apt-get install git
- Clonar el repositorio, git clone https://github.com/intrig-unicamp/mininet-wifi
- cd mininet-wifi
- Completar la instalación: sudo **util/install.sh -Wlnfv**
  - -W: wireless dependencies
  - -n: mininet-wifi dependencies
  - -f: OpenFlow
  - -v: OpenvSwitch
  - -l: wmediumd
- De forma adicional hemos instalado Wireshark para poder analizar los test realizados

### 1.4. Mininet-Wifi CLI

En esta sección se va a recorrer todas las funcionalidades que nos ofrece la CLI ( Command Line Interface) de Mininet-Wifi en la medida que sea posible. A continuación, listamos todos los comandos que están disponibles y documentados en la CLI de Mininet-Wifi.

<b>EOF</b>	<b>gterm</b>	<b>links</b>	<b>pingallfull</b>	<b>py</b>	<b>stop</b>
<b>distance</b>	<b>help</b>	<b>net</b>	<b>pingpair</b>	<b>quit</b>	<b>switch</b>
<b>dpctl</b>	<b>intfs</b>	<b>nodes</b>	<b>pingpairfull</b>	<b>sh</b>	<b>time</b>
<b>dump</b>	<b>iperf</b>	<b>noecho</b>	<b>ports</b>	<b>source</b>	<b>x</b>
<b>exit</b>	<b>iperfudp</b>	<b>pingall</b>	<b>px</b>	<b>start</b>	<b>xterm</b>

#### 1.4.1. Comando: EOF + quit + exit

Estos tres comandos se utilizan para lo mismo, salir de la CLI de Mininet-Wifi y terminar la emulación. El código de estos tres comandos no difieren mucho, son comandos heredados de la CLI de Mininet. EOF y quit terminan haciendo uso de exit al final, por lo que podríamos decir que son un poco repetitivos.

```
def do_exit( self, _line ):
    "Exit"
    assert self # satisfy pylint and allow override
    return 'exited by user command'

def do_quit( self, line ):
    "Exit"
    return self.do_exit( line )

def do_EOF( self, line ):
    "Exit"
    output( '\n' )
    return self.do_exit( line )
```

### 1.4.2. Comando: distance

Este comando sirve para medir la distancia entre dos entes de la topología emulada. El uso es muy sencillo, únicamente hay que indicar el nombre dado a cada ente y nos dará la distancia en metros.

```
mininet-wifi> distance stal sta2
The distance between stal and sta2 is 40.00 meters
mininet-wifi> █
```

Figura 1.1: Comando distance output.

### 1.4.3. Comando: dpctl

El comando dpctl es un comando heredado de la CLI de Mininet, pero redefinido en el for de Mininet-Wifi con pocas variaciones. Dpctl es un comando de utilidad de administración que permite cierto control sobre el switch OpenFlow (ovs-ofctl en el switch OpenvSwitch). Con este comando es posible agregar flujos a la tabla de flujo, consultar las características y el estado de los switches y cambiar otras configuraciones, limpiar la tabla entre otras cosas.

Como se ha dicho este comando tiene bastantes funcionalidades, por no entrar una a una se deja aquí donde se debe ir a consultar cada funcionalidad del comando en función del tipo de switch.

- Ofsoftswitch: <https://github.com/CPqD/ofsoftswitch13/wiki/Dpctl-Documentation>
- OpenvSwitch: <http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt>

Por lo que se ha podido apreciar en el código del comando, la instrucción se replicará en todos los switches de la topología.

```
def do_dpctl(self, line):
    """Run dpctl (or ovs-ofctl) command on all switches.
    Usage: dpctl command [arg1] [arg2] ..."""
    args = line.split()
    if len(args) < 1:
        error('usage: dpctl command [arg1] [arg2] ...\n')
        return
    nodesL2 = self.mn.switches + self.mn.aps
    for sw in nodesL2:
        output('*** ' + sw.name + ' ' + ('-' * 72) + '\n')
        output(sw.dpctl(*args))
```

*#Nota: Nosotros utilizaremos sobretodo dpctl dump-flows y  
# dpctl del-flows para consultar o limpiar las tablas de flujo.*

### 1.4.4. Comando: dump + net

Estos comandos nos arrojarán información sobre la topología emulada. El comando net nos indicará los nombres de los entes que hay en la topología a emular y sus interfaces. El comando dump además nos indicará el tipo de ente, dirección IP, puerto cuando corresponda, interfaz y el identificador de proceso (pid) del ente.

### 1.4.5. Comando: xterm y gterm

Estos dos comandos nos permitirán abrir terminales en el nodo indicado. El comando **xterm** nos permitirá abrir una terminal simple, y el comando **gterm** nos permitirá abrir una gnome-terminal. Podemos abrir varias terminales a la vez indicando todos los nodos que queremos abrir una terminal en ellos.

Uso xterm/gterm [node1] [node2] ...

```

mininet-wifi> net
c1
sta1 sta1-wlan0:wifi
sta2 sta2-wlan0:wifi
ap1 lo: ap1-wlan1:wifi ap1-eth2:ap2-eth2
ap2 lo: ap2-wlan1:wifi ap2-eth2:ap1-eth2
mininet-wifi> dump
<Controller c1: 127.0.0.1:6653 pid=8355>
<Station sta1: sta1-wlan0:10.0.0.2 pid=8328>
<Station sta2: sta2-wlan0:10.0.0.3 pid=8333>
<OVSAp ap1: lo:127.0.0.1,ap1-wlan1:None,ap1-eth2:None pid=8341>
<OVSAp ap2: lo:127.0.0.1,ap2-wlan1:None,ap2-eth2:None pid=8347>
mininet-wifi>

```

Figura 1.2: Comando net y dump output.

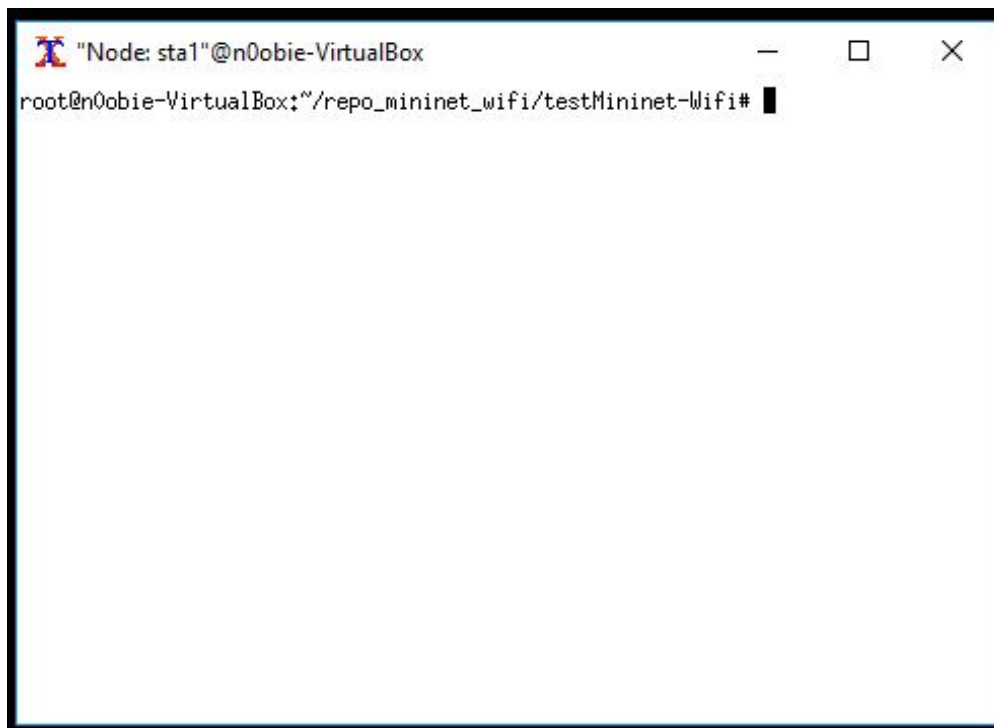


Figura 1.3: Ejemplo comando xterm.

#### 1.4.6. Comando: help

El comando por excelencia para visualizar las indicaciones o posible ayuda sobre la operativa de funcionamiento de un programa o comando. Es un comando heredado desde Mininet, al que Mininet-Wifi ha añadido sus nuevas características. Podemos hacer uso de él únicamente escribiendo **help**. Además podemos obtener información extra de otros comandos utilizando **help comando**.

#### 1.4.7. Comando: intfs + nodes + ports

Estos comandos listarán información relativa a los nodos de la topología. El comando **intfs** listará toda la información relativa a las interfaces de los nodos. El comando **nodes** listará todos los nodos de la topología.



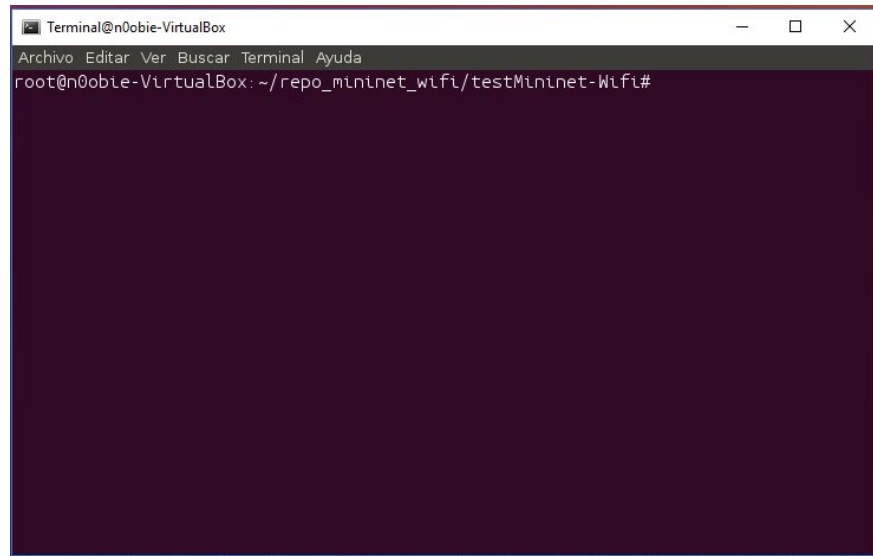


Figura 1.4: Ejemplo comando gterm.

y el comando, **ports** utilizado más bien en Mininet se utilizaba para listar los puertos e interfaces de los switches de la topología.

```
mininet-wifi> intfs
c0:
sta1: sta1-wlan0
sta2: sta2-wlan0
ap1: lo,ap1-wlan1
mininet-wifi> █
```

Figura 1.5: Ejemplo comando intfs.

```
mininet-wifi> nodes
available nodes are:
ap1 c0 sta1 sta2
mininet-wifi> █
```

Figura 1.6: Ejemplo comando nodes.

#### 1.4.8. Comando: **iperf** + **iperfudp**

Estos comandos nos ayudarán a hacer un benchmark de las capacidades de la topología. Podemos indicar la parte cliente y al parte servidor o dejar a elección de Mininet-Wifi la decisión de entre que extremos hacer el test de **iperf**. La única diferencia entre el comando **iperf** y **iperfudp** es que uno emplea tráfico TCP para hacer el test y el otro hace uso de tráfico UDP para hacer el test. Una vez completado el test podremos apreciar el resultado en Mbits/s. Para más información sobre la herramienta **iperf** consulte el punto **A de los anexos**.

```
mininet-wifi> iperf
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['9.41 Mbits/sec', '10.1 Mbits/sec']
mininet-wifi> █
```

Figura 1.7: Ejemplo de benchmark con Iperf.

### 1.4.9. Comando: links

Este comando nos arrojará información sobre el estado de los enlaces de la topología. Este comando, aunque ha sido heredado de Mininet, éste ha sido adaptado a los nuevos enlaces de características Wireless de Mininet-Wifi.

```
mininet-wifi> links
sta1-wlan0<->wifi (OK wifi)
sta2-wlan0<->wifi (OK wifi)
ap1-eth2<->ap2-eth2 (OK OK)
mininet-wifi> █
```

Figura 1.8: Ejemplo comando links.

### 1.4.10. Comando: noecho

Comando completamente heredado de Mininet. Su funcionalidad reside en ejecutar un comando en uno de los nodos de forma silenciosa, sin sacar por pantalla ningún tipo de información sobre el comando. Para hacer test de este comando y comprender bien bien la salida estándar se ha leído este artículo, muy interesante para entender el concepto de tty y sus opciones: <http://www.linusakesson.net/programming/tty/index.php>

```
def do_noecho( self, line ):
    """Run an interactive command with echoing turned off.
    Usage: noecho [cmd args]"""
    if self.isatty():
        # Modo silencioso
        quietRun( 'stty -echo' )

    self.default( line )

    if self.isatty():
        # Lo volvemos a poner de manera estándar
        quietRun( 'stty echo' )
```

### 1.4.11. Comando: pingall + pingallfull

Estos comandos nos permitirán hacer ping entre todos los host y las estaciones wifi. Por lo que se ha estado testeado pingall funciona según la descripción de su uso. Por el contrario el comando pingallFull no funcionaba, se ha estado investigando y depurando el código, y hemos llegado a la conclusión de que el código estaba obsoleto, ya que al ser un fork de Mininet únicamente incluía a Host con una interfaz alámbrica y no a las estaciones Wifi. Por lo que se ha modificado el código en un fork de Mininet-Wifi para así dotar de la funcionalidad que supuestamente debía tener. Acto seguido se ha abierto otro pull-request para añadir esta modificación en el master de Mininet-Wifi, aquí pueden comprobar el seguimiento de esta modificación.

<https://github.com/intrig-unicamp/mininet-wifi/pull/230>

```
mininet-wifi> pingallfull
*** Ping: testing ping reachability
sta1 -> sta2
sta2 -> sta1
*** Results:
  sta1->sta2: 1/1, rtt min/avg/max/mdev 2.910/2.910/2.910/0.000 ms
  sta2->sta1: 1/1, rtt min/avg/max/mdev 3.084/3.084/3.084/0.000 ms
mininet-wifi> █
```

Figura 1.9: Posible futuro funcionamiento del comando pingallfull.

#### 1.4.12. Comando: pingpair + pingpairfull

Los comandos **pingpair** y **pingpairfull** son análogos a los expuestos anteriormente. La diferencia reside en el hecho de que ya no es un ping entre todos los host y estaciones Wifi, si no, por parejas.

#### 1.4.13. Comando: px + py

Estos comandos pueden ser similares, pero la diferencia reside en el hecho de que el comando px ejecuta código python y el comando py evalúa una sentencia de código python. Pero, ¿Cual es la diferencia entre evaluar y ejecutar? Hay dos diferencias importantes:

- La primera de ellas, es que a la hora de evaluar, evaluamos un única expresión. Mientras que si hablamos de ejecución podemos ejecutar un bloque entero de código (Una función entera, un bloque try/catch, una clase). Ojito! O.o una declaración no es una expresión.
- La segunda diferencia reside en el hecho de que una evaluación de código siempre retorna el valor/resultado de la expresión dada. Por el contrario la ejecución ignora el valor/resultado de una expresión deberemos obligarle a retornarlo por ejemplo haciendo print de la expresión a ejecutar.

Expuesto esta diferencias se debería entender el resultado y la forma en la que se han ejecutado ambos comandos para obtener el mismo resultado.

```
mininet-wifi> px print sta1.cmd("echo Esto es una prueba n.n, todo OK!")
Esto es una prueba n.n, todo OK!

mininet-wifi> py sta1.cmd("echo Esto es una prueba n.n, todo OK!")
Esto es una prueba n.n, todo OK!

mininet-wifi> █
```

Figura 1.10: Ejemplo funcionamiento comandos px y py.

#### 1.4.14. Comando: sh

Este comando tiene una gran importancia ya que es nuestra puerta a ejecutar código fuera de la CLI de Mininet-Wifi. El output de dicho comando es redirigido a la CLI de Mininet-Wifi, su uso es muy sencillo únicamente hay que poner el comando sh y el comando a ejecutar. Ejemplo: **sh ifconfig**.

```
mininet-wifi> sh ls
mn12545_ap1-wlan1.apconf  test_movilidad_1.py  test_movilidad_3.py  test_topo_2.py  traza.txt
mn12545_ap2-wlan1.apconf  test_movilidad_2.py  test_topo_1.py      test.zip        traza.zip
```

Figura 1.11: Ejemplo funcionamiento comando sh.

### 1.4.15. Comando: source

El comando source tiene como funcionalidad de ejecutar un fichero con comandos de la CLI de Mininet-Wifi. Por así decirlo es un comando que nos permite lanzar Mininet-Wifi CLI scripts (Podríamos llamar a los ficheros con ordenes por ejemplo **MWCLI-scripts**). Puede que esto se vea un poco abstracto por lo que vamos a poner ejemplo de uso. En un fichero llamado por nosotros comando.txt vamos a introducir linea a linea comandos que queremos ejecutar en la CLI de Mininet-Wifi. Esto nos ahorrará el tiempo de escribirlos cuando lancemos la CLI de Mininet-Wifi y agotaremos menos tiempo de la emulación.

```
n0obie@n0obie-VirtualBox:~/repo_mininet_wifi/testMininet-Wifi/scripts$ cat comando.txt
py sta1.cmd("echo Ejecutamos un comando desde un fichero externo!!!")
dump

n0obie@n0obie-VirtualBox:~/repo_mininet_wifi/testMininet-Wifi/scripts$ █
```

Figura 1.12: Contenido del fichero comandos.txt.

Una vez dentro de CLI de Mininet-Wifi podemos llamar a nuestro MWCLI script con el comando source, el resultado es el siguiente. (El echo sale corrupto aun no sé a ciencia cierta el por qué, pero sospecho que se debe a que había un mensaje en sta1 en cola para ser dirigido por la salida estándar y a salido con nuestra orden de echo).

```
mininet-wifi> source comando.txt
echo Ejecutamos un comando desde un fichero externoip link set lo up!
Ejecutamos un comando desde un fichero externoip link set lo up!

<Controller c1: 127.0.0.1:6653 pid=13869>
<Station sta1: sta1-wlan0:10.0.0.2 pid=13842>
<Station sta2: sta2-wlan0:10.0.0.3 pid=13847>
<OVSAP ap1: lo:127.0.0.1,ap1-wlan1:None,ap1-eth2:None pid=13855>
<OVSAP ap2: lo:127.0.0.1,ap2-wlan1:None,ap2-eth2:None pid=13861>
mininet-wifi> █
```

Figura 1.13: Ejemplo funcionamiento comando source.

### 1.4.16. Comando: start + stop

En teoría estos comandos, añadidos en Mininet-Wifi tienen como funcionalidad la de parar/iniciar la movilidad en los nodos de la topología. No funciona. Se ha hecho debug y se ha visto que lo único que hacen es modificar una variable booleana llamada pause\_simulation pero no hacen nada con ella después. Si tengo tiempo arreglaré esto y presentaré el tercer pull-request para que lo añadan al master.

```
# cli.py

def do_stop(self, line):
    "stop mobility for a while"
    self.mn.stop_simulation()

def do_start(self, line):
    "pause mobility for a while"
    self.mn.start_simulation()
```

```
# net.py clase Mininet_Wifi (obj mn)

    @staticmethod
    def stop_simulation():
        "Pause the simulation"
        mob.pause_simulation = True

    @staticmethod
    def start_simulation():
        "Start the simulation"
        mob.pause_simulation = False

# mobility.py

class mobility(object):
    ...
    pause_simulation = False
    ...
```

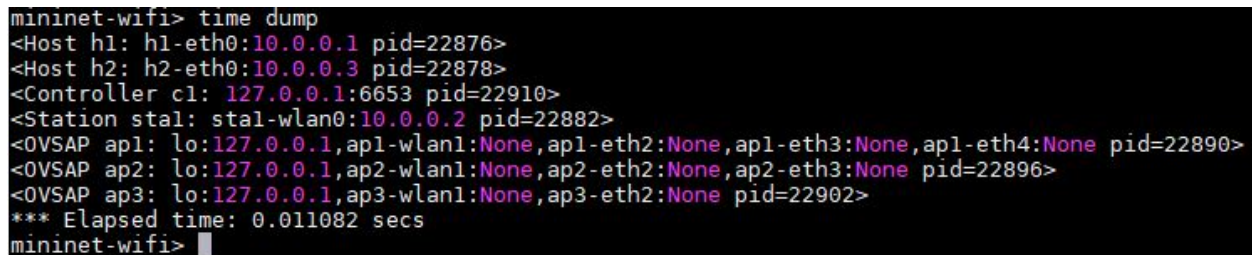
#### 1.4.17. Comando: switch

El comando switch es un comando heredado de la CLI de Mininet. Su funcionalidad era la de parar o reanudar un switch dado el nombre y la operación de start/stop. Sintaxis:

```
switch <switch name> {start,stop}
```

#### 1.4.18. Comando: time

El comando time es similar al comando time de la terminal de Linux. Se utiliza para sacar el tiempo que se consume en ejecutar una orden dada.

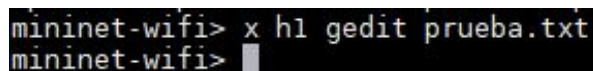


```
mininet-wifi> time dump
<Host h1: h1-eth0:10.0.0.1 pid=22876>
<Host h2: h2-eth0:10.0.0.3 pid=22878>
<Controller c1: 127.0.0.1:6653 pid=22910>
<Station stal: stal-wlan0:10.0.0.2 pid=22882>
<OVSAp ap1: lo:127.0.0.1,ap1-wlan1:None,ap1-eth2:None,ap1-eth3:None,ap1-eth4:None pid=22890>
<OVSAp ap2: lo:127.0.0.1,ap2-wlan1:None,ap2-eth2:None,ap2-eth3:None pid=22896>
<OVSAp ap3: lo:127.0.0.1,ap3-wlan1:None,ap3-eth2:None pid=22902>
*** Elapsed time: 0.011082 secs
mininet-wifi> █
```

Figura 1.14: Ejemplo funcionamiento comando time.

#### 1.4.19. Comando: x

El comando x se utiliza para crear un canal x11 a un nodo dado. Es muy útil para abrir aplicaciones con una interfaz gráfica. Ejemplo:



```
mininet-wifi> x h1 gedit prueba.txt
mininet-wifi> █
```

Figura 1.15: Ejemplo funcionamiento comando x.

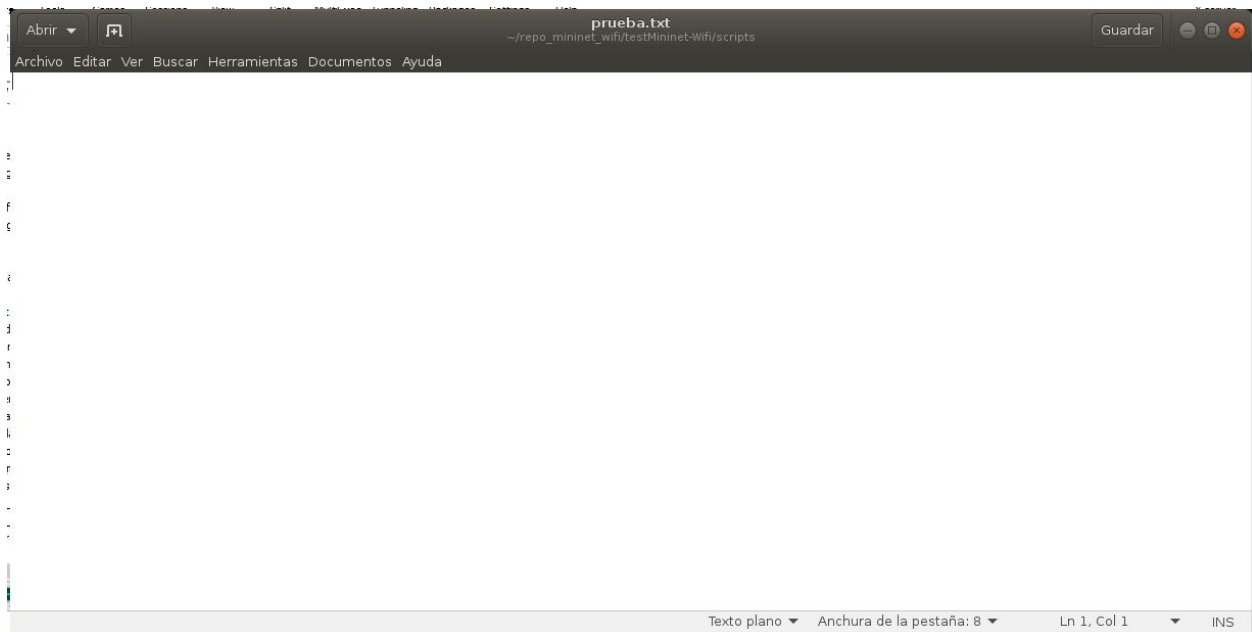


Figura 1.16: GUI de gedit en el Host 1.

## 1.5. Test 1: Escenario simple, unico punto de acceso

La red más sencilla es por defecto la red de un punto de acceso y dos estaciones wireless. El punto de acceso está conectado directamente al controlador, y las estaciones wireless serán los host. La idea de este test es comprobar que hay fluctuación de tráfico de control OpenFlow en el punto de acceso y además hay tráfico de usuario en la interfaz de wlan.

### 1.5.1. Capturando tráfico de control wireless en Mininet-Wifi

- Lanzamos Wireshark para analizar tráfico: **wireshark &**
- Lanzamos Mininet-Wifi con el siguiente comando, además por defecto cargará la topología por defecto (Valga la redundancia): **sudo mn -wifi**
- Lo siguiente, es activar la interfaz hwsim0. Pero, ¿Qué es la interfaz hwsim0? La interfaz hwsim0 es una interfaz software creada por Mininet-Wifi que copia todo el tráfico wireless dirigido a todas las interfaces wireless virtuales de la topología a emular. Según la documentación seguida es la forma más sencilla de monitorizar los mensajes wireless en Mininet-Wifi. Desde la Mininet-Wifi CLI: **sh ifconfig hwsim0 up**
  - El comando **sh** en la CLI de Mininet tiene la funcionalidad de ejecutar un comando fuera de la Interfaz de Mininet-Wifi.
- Una vez levantada la interfaz podemos poner Wireshark a escuchar en la interfaz.
- Hacemos Ping desde la estación sta1, a la estación sta2: **sta1 ping sta2**
- Comprobamos que hay tráfico escuchando en la interfaz:



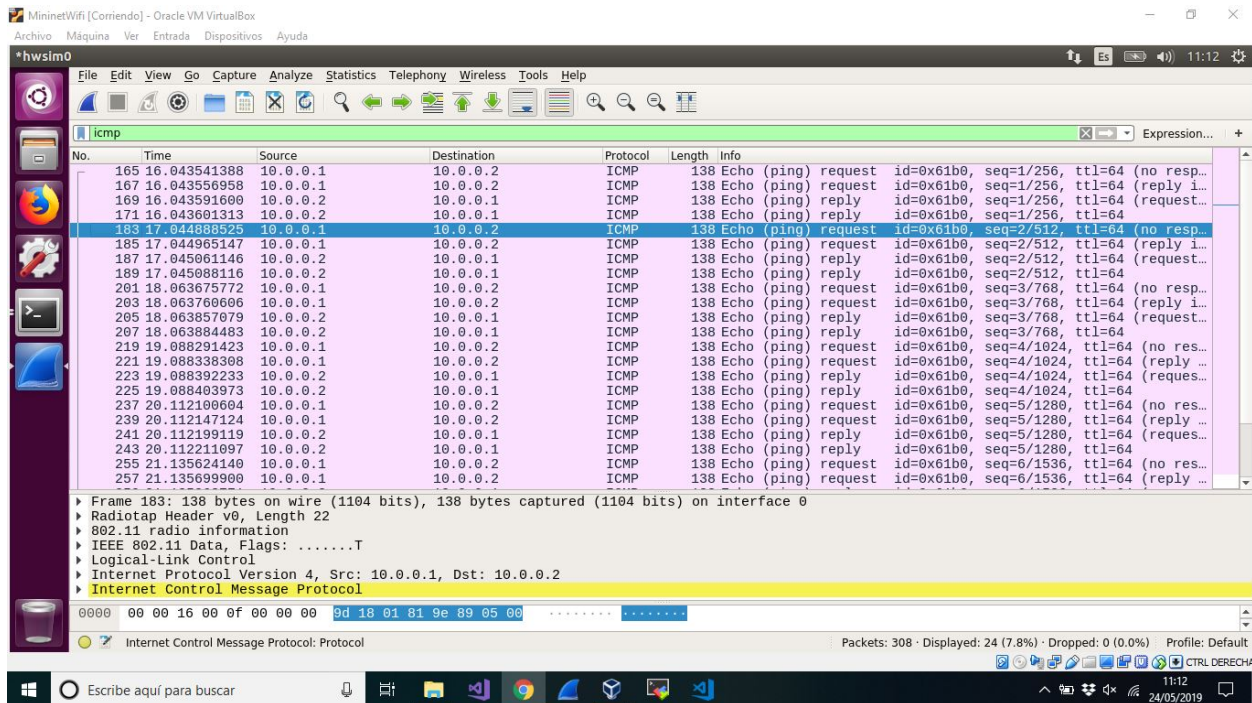


Figura 1.17: Comprobación que hay tráfico a través de la interfaz.

Como puede verse el punto de acceso tiene una interfaz asociada llamada `ap1-wlan1`. Por defecto, las estaciones wireless asociadas con el punto de acceso se conectan en modo **"infrastructure"**. Esto significa que el tráfico wireless entre dos estaciones asociadas al punto de acceso debe pasar siempre a través de este. Sabiendo que los puntos de acceso funcionan de forma similar a los switch en Mininet, esperaríamos observar tráfico de control entre el punto de acceso y el controlador, cuando el punto de acceso, observe tráfico para el cual no se ha establecido una regla (No pertenece a un flujo para el cual haya asignada una acción en la tabla de flujos).

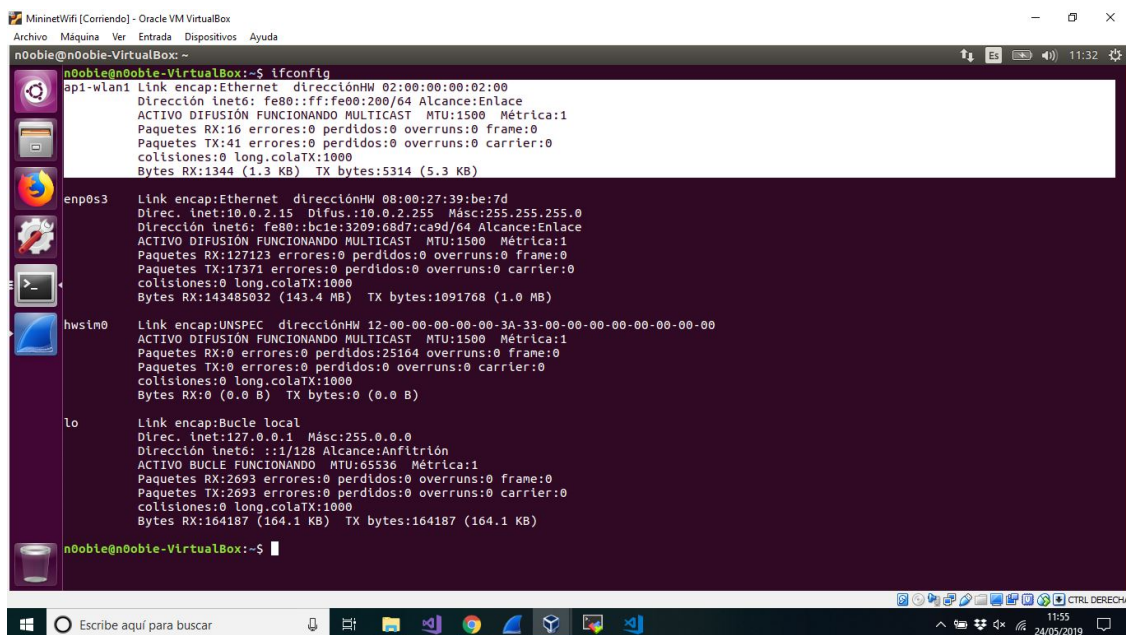


Figura 1.18: Interfaz `ap1-wlan1` del Punto de acceso.



### 1.5.2. Puntos de acceso Wireless y OpenFlow

Si deseamos ver **paquetes OpenFlow**, debemos poner a escuchar Wireshark en la interfaz de **loopback**. Podemos además utilizar el filtro: **OpenFlow\_1.0** para visualizarlo de una forma más clara. Dicho esto, ponemos a capturar Wireshark y repetimos el ping entre sta1 y sta2.

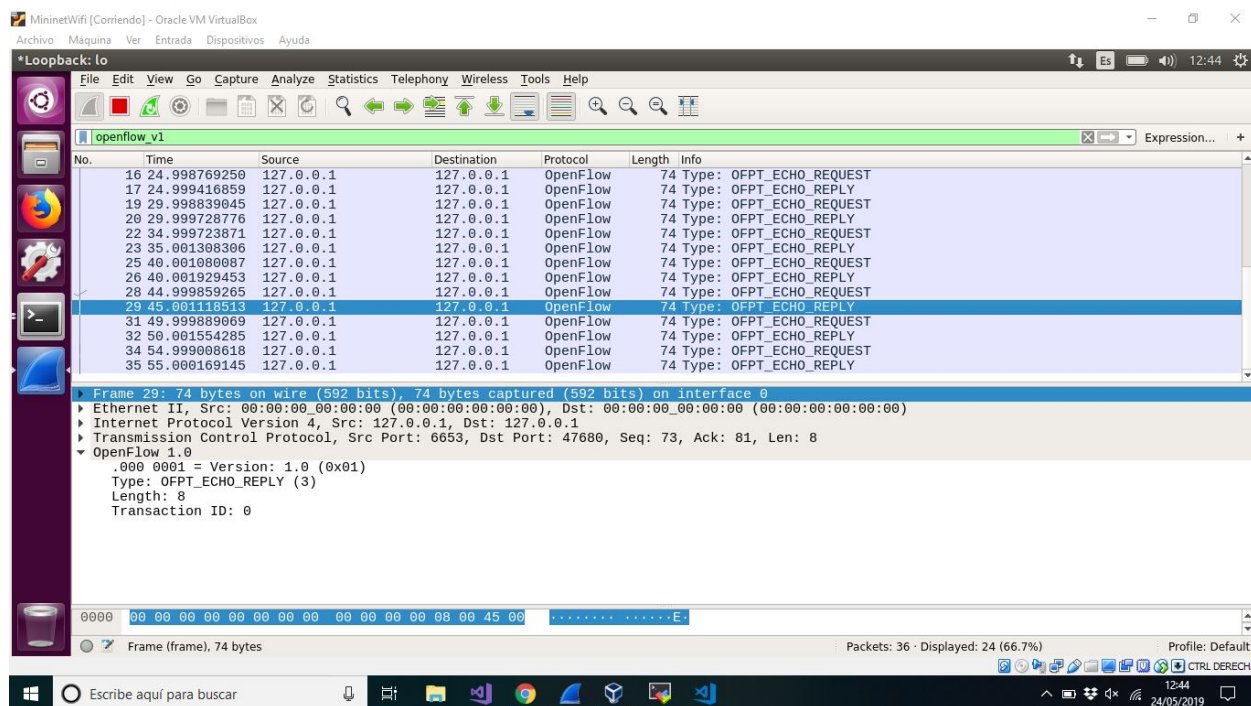


Figura 1.19: Tráfico OpenFlow.

Nota:

- Consultar comando dpctl: <http://ranosgrant.cocolog-nifty.com/openflow/dpctl1.8.html>
- xID: Identificador de transacción, es aleatorio, identificador controlador - switch, va cambiando a lo largo del tiempo.
- El estándar es OpenFlow 1.3, pero Mininet-Wifi funciona con OpenFlow 1.1

Como esperábamos el paquete ICMP debería ser redirigido al controlador para decidir que hacer con el paquete, al no tener un Flow con una regla establecida. De esta manera el controlador cuando le llegue el paquete instanciará una serie de reglas en los switch para que el paquete ICMP sea encaminado de un host a otro. En cambio, encontramos que las dos estaciones son capaces de intercambiar paquetes inmediatamente sin redirigir el primer paquete al controlador. Solo una trama ARP, que es de tipo broadcast, es llevada hacia el controlador y es ignorada.

Para comprobar si hubiera algún flujo con una acción instanciada en alguna de las estaciones bases hacemos: **dpctl dump-flows**

Al hacer esto en esta altura del tutorial, vemos que no hay ninguna regla instanciada en la estación. ¿Pero, tiene esto sentido? ¿Cómo se han comunicado las dos puntos de acceso?

Según la guía, podemos apreciar que los switches con OpenFlow activado, van a rechazar las "Hairpin connections". Las cuales son flows que hacen que el tráfico, sea enviado por el mismo puerto que ha sido recibido.

- **Hairpin connection:** Comunicación entre dos dispositivos host en el mismo dominio NAT utilizando sus puertos finales (Ya traducidos)

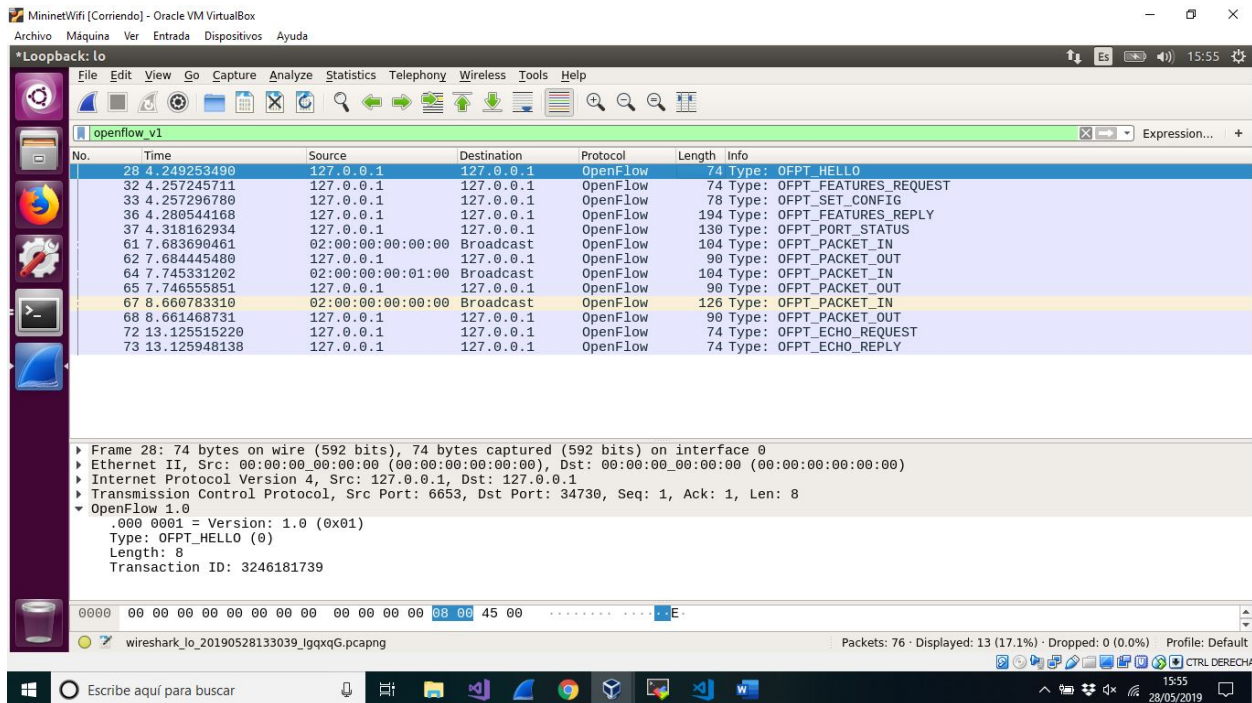


Figura 1.20: No hay tráfico OpenFlow hacia el controlador.

Un punto de acceso wireless, por diseño, recibe y manda paquetes en la misma interfaz. Las estaciones wifi conectadas al mismo punto de acceso requerirán a "Hairpin connection" para comunicarse entre ellas. El autor de la guía supone que Linux, maneja a la interfaz WLAN en cada punto de acceso como un enlace radio sta1-ap1-sta2 como si fuera un HUB, donde ap1-wlan0 proporciona la funcionalidad de "HUB" para los datos que pasan entre sta1 y sta2. ap1-wlan0 cambia los paquetes en el dominio inalámbrico y lo hará no introduzca un paquete en la parte del conmutador Ethernet" del punto de acceso ap1 a menos que deba cambiarse a otra interfaz en ap1 que no sea ap1-wlan0.

- Podemos detener el ping con CTRL+C
- Podemos detener Mininet-Wifi vía comando **exit**
- Podemos limpiar los archivos residuales de Mininet-Wifi con: **sudo mn -c**

## 1.6. Test 2: Multiples puntos de acceso

En este test vamos a crear una topología lineal con tres puntos de acceso, y tres estaciones wifi. Donde cada una está conectada a cada punto de acceso.

Podemos crear la topología: **sudo mn -wifi -topo linear,3**

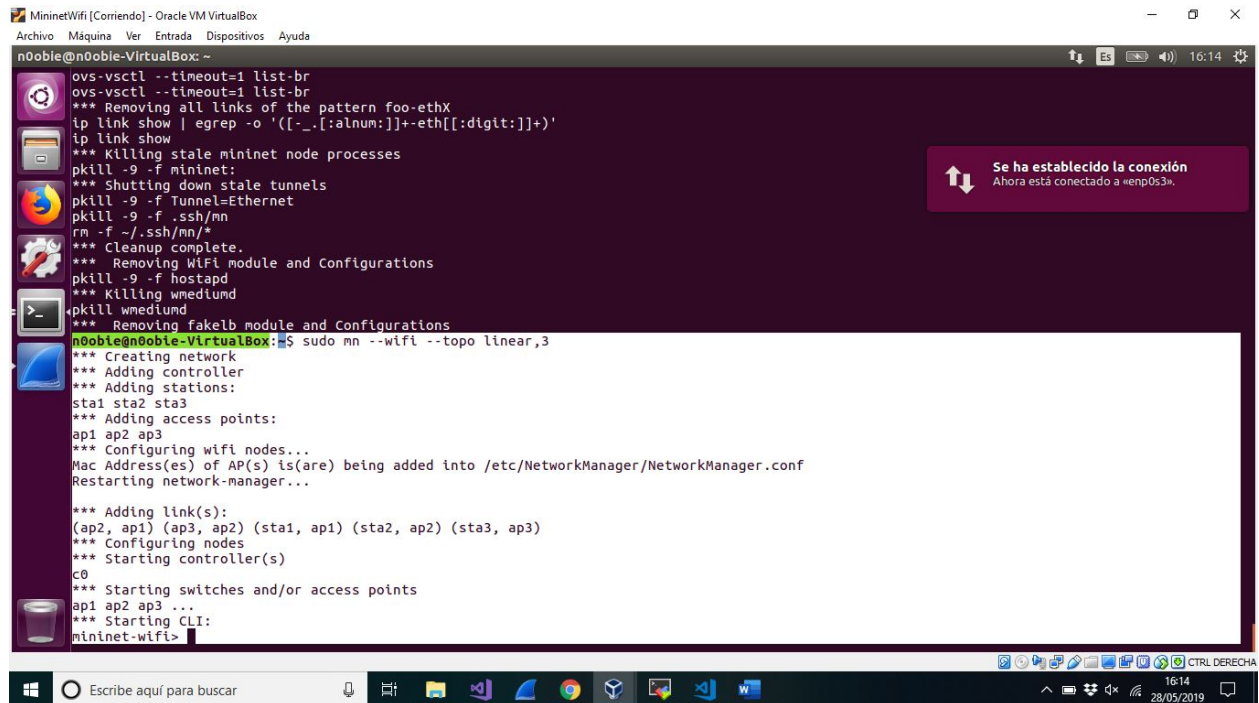


Figura 1.21: Montar topología lineal en Mininet-Wifi.

Además, podemos verificar la configuración de la topología haciendo uso de los comandos **net** y **dump**. Con el comando **net** podemos ver la conexión entre los nodos. Con el comando **dump** podemos ver la conexión entre los nodos y además información extra como el PID.

Podemos contemplar que tenemos los tres puntos de acceso conectados juntos entre ellos uno a uno de una manera lineal mediante enlaces de Ethernet. Pero, no vemos ningún tipo de información sobre las estaciones Wifi, en lo relativo a que punto de acceso están conectadas. Esto se debe a que esa conexión se lleva a cabo mediante un enlace de tipo radio. Para poder consultar esta información tendremos que hacer uso del comando **iw** (Interface Wireless) en cada estación Wifi para poder contemplar a que punto de acceso está conectada.

Para poder verificar que puntos de acceso son alcanzables por una estación Wifi usaremos **staX iw dev staX-wlan0 scan**. Para afinar la búsqueda podemos hacer un grep del ssid para quitarnos información extra.

```
sta1 iw dev sta1-wlan0 scan — grep ssid
```

```

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

Figura 1.22: Comprobar que puntos de accesos son alcanzables en Mininet-Wifi.

Para comprobar a que punto de acceso está conectado en un momento dado una estación Wifi podemos hacerlo mediante el comando **iw link**. Por ejemplo: **sta1 iw dev sta1-wlan0 link**

```
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: 1973604 bytes (44742 packets)
  TX: 10930 bytes (277 packets)
  signal: -36 dBm
  tx bitrate: 54.0 MBit/s

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

Figura 1.23: Comprobar el punto de acceso actual en Mininet-Wifi.

Con este comando podemos apreciar distintas estadísticas como pueden ser la potencia de señal recibida, la cantidad de bytes transmitidos y recibidos, el régimen binario de transmisión, SSID, y la frecuencia expresada en MHz, en este caso se trata del canal 1 Wifi. Debemos recordar que el espectro Wifi se extiende desde 2402 MHz hasta los 2494 Mhz (La banda de los 2.4GHz).

En este caso cada estación Wifi está conectada a un punto de acceso según se pudo ver con los anteriores comandos de **net** y **dump**. Se puede hacer uso del comando **iw** para conmutar de un punto de acceso por otro.

Nota de la guía: *Los comandos iw pueden usarse en escenarios estáticos como este, pero no deben usarse cuando Mininet-WiFi asigna automáticamente asociaciones en escenarios de movilidad más realistas. Discutiremos cómo Mininet-WiFi maneja la movilidad real y cómo usar los comandos iw con Mininet-WiFi más adelante esta publicación.*

### 1.6.1. Asociaciones con los puntos de acceso

En este caso vamos a comprobar como podemos cambiar la conexión existente entre la estación Wifi 1 (sta1) con el punto de acceso 1 (ap1). En este caso vamos a conectar la estación Wifi 1 (sta1) con el punto de acceso 2 (ap2). Debemos recordar los SSID de cada punto de acceso (ssid\_apX).

- Para desconectarnos del actual punto de acceso: **sta1 iw dev sta1-wlan0 disconnect**
- Para conectarnos a ap2: **sta1 iw dev sta1-wlan0 connect ssid\_ap2**

Podemos verificar el funcionamiento de este comando usando el comando iw link que ya se explico anteriormente. Hemos podido comprobar como se puede hacer un simple *Handover* entre dos puntos de acceso.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:04:00 (on sta1-wlan0)
  SSID: ssid_ap2
  freq: 2412
  RX: 7307 bytes (168 packets)
  TX: 94 bytes (2 packets)
  signal: -36 dBm
  tx bitrate: 1.0 MBit/s

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

Figura 1.24: *Handover* completado en Mininet-Wifi.



Ahora vamos a estudiar como gestiona el controlador OVSAP de Mininet-Wifi el Handover entre dos células Wifi. Para llevar a cabo este experimento vamos hacer ping desde sta3 hacia sta1. Entraremos en la terminal de sta3 con el comando **xterm**. Para hacer ping debemos saber las distintas IPs de las distintas estaciones Wifi, podemos consultarlas con el comando **dump**.

```
mininet-wifi> dump
<Controller c0: 127.0.0.1:6653 pid=7410>
<Station sta1: sta1-wlan0:10.0.0.1 pid=7417>
<Station sta2: sta2-wlan0:10.0.0.2 pid=7419>
<Station sta3: sta3-wlan0:10.0.0.3 pid=7421>
<OVSAP ap1: lo:127.0.0.1,ap1-wlan1:None,ap1-eth2:None pid=7426>
<OVSAP ap2: lo:127.0.0.1,ap2-wlan1:None,ap2-eth2:None,ap2-eth3:None pid=7429>
<OVSAP ap3: lo:127.0.0.1,ap3-wlan1:None,ap3-eth2:None pid=7432>
mininet-wifi>
```

Figura 1.25: Consultar IPs en Mininet-Wifi.

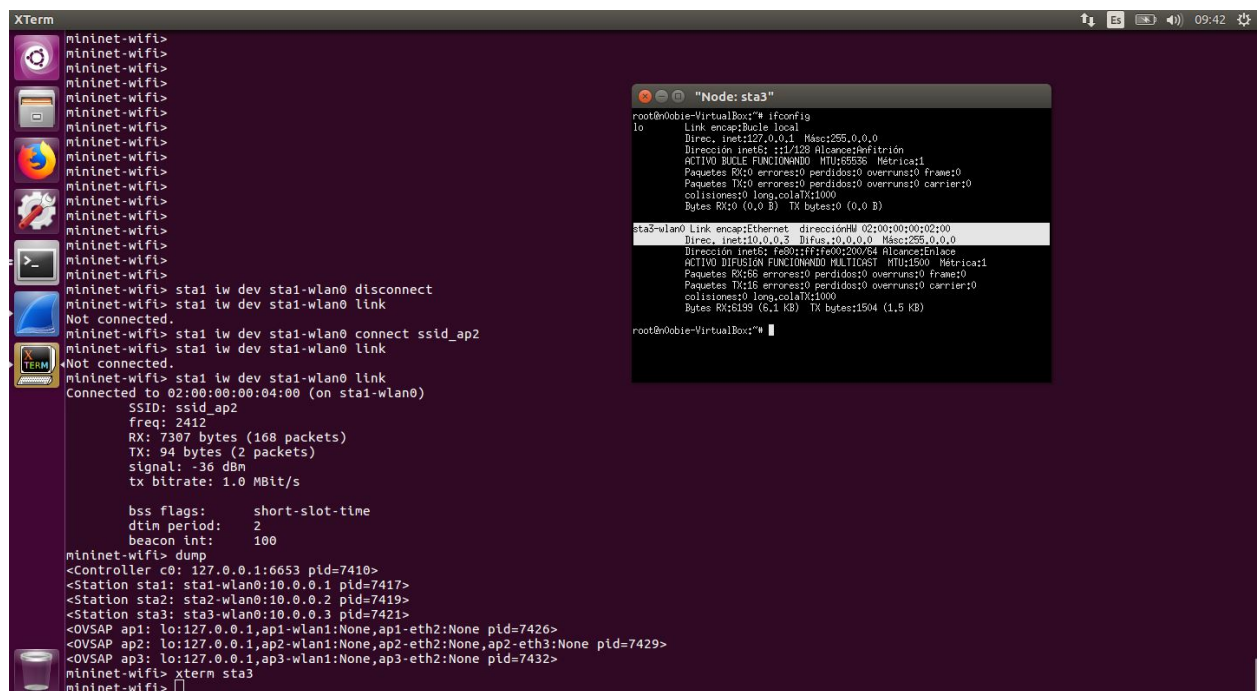


Figura 1.26: Entrar en la consola de *Sta3*.

Vamos hacer ping desde sta3 a sta1 (**Ip: 10.0.0.1**). Ahora desde que estos paquetes serán reenviados por los puntos de acceso asociados, en un puerto distinto al puerto en el que se recibieron los paquetes, los puntos de acceso funcionarán normalmente en cuanto a OpenFlow se refiere. Cada punto de acceso va a reenviar el primer ping que le llegue en dirección al controlador. El controlador va a instanciar en las tablas de flows de cada punto de acceso las reglas correspondientes para que se pueda instanciar una conexión entre sta1 y sta3.

### 1.6.2. Flows OpenFlow en un escenario de Handover

Podemos comprobar el intercambio de paquetes OpenFlow escuchando la interfaz de loopback.

```

"Node: sta3"

colisiones:0 long.colaTX:1000
Bytes RX:0 (0.0 B) TX bytes:0 (0.0 B)

sta3-wlan0 Link encap:Ethernet direcciónHW 02:00:00:00:02:00
Direc. inet:10.0.0.3 Difus.:0.0.0.0 Másc:255.0.0.0
Dirección inet6: fe80::ff:fe00:200/64 Alcance:Enlace
ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST MTU:1500 Métrica:1
Paquetes RX:66 errores:0 perdidos:0 overruns:0 frame:0
Paquetes TX:16 errores:0 perdidos:0 overruns:0 carrier:0
colisiones:0 long.colaTX:1000
Bytes RX:6199 (6.1 KB) TX bytes:1504 (1.5 KB)

root@n0obie-VirtualBox:~# 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=30.7 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.31 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.248 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.368 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.393 ms
^C
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4057ms
rtt min/avg/max/mdev = 0.248/6.616/30.754/12.075 ms
root@n0obie-VirtualBox:~#

```

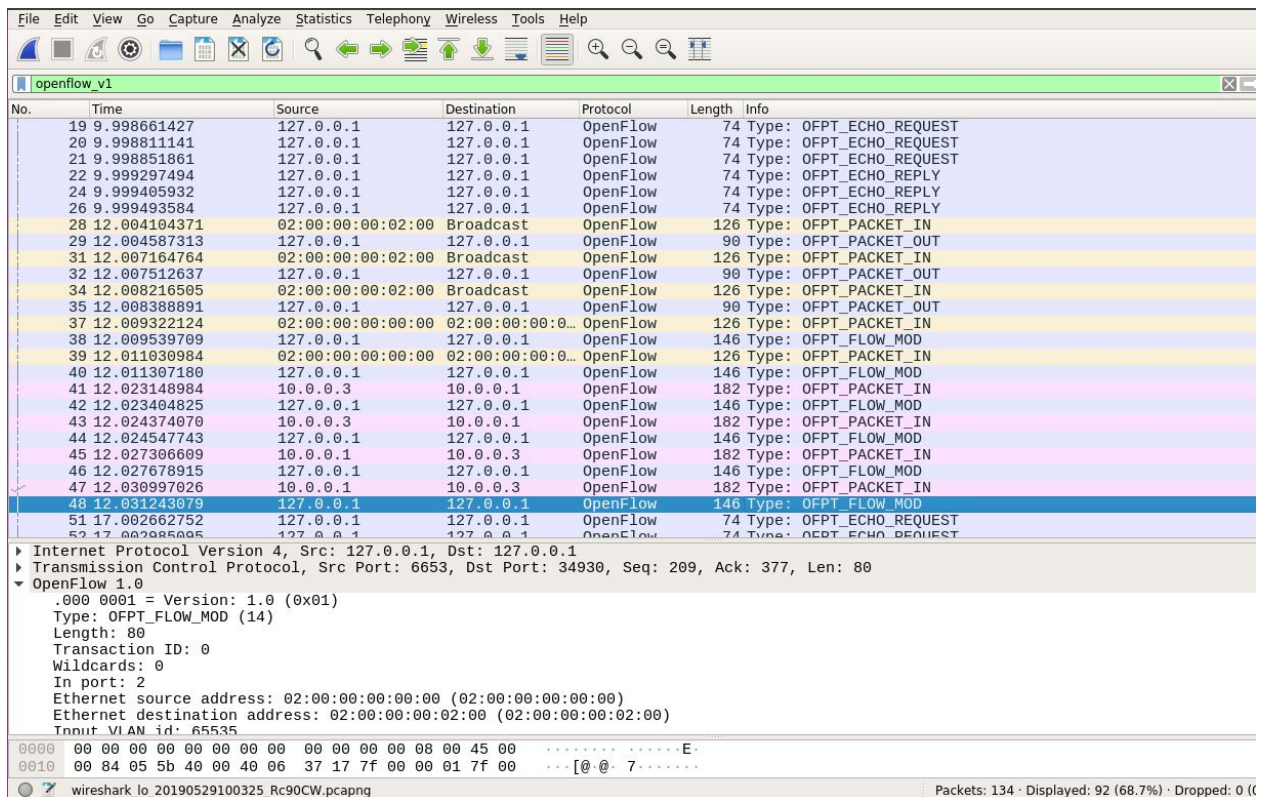
Figura 1.27: Ping desde *Sta3* a *Sta1*.

Figura 1.28: Captura de Wireshark del ping.



Podemos apreciar como el punto de acceso ap3 empieza mandando el paquete de ARP request para que Sta3 sepa la *MAC* de Sta1 hacia arriba para que decida que debe hacer con el controlador. El controlador hace un *PACKET\_OUT*. Para el ARP reply el controlador hace un *PACKET\_IN*. Una vez que Sta3 sabe la dirección *MAC* del destino se dispone a enviar el ping. El controlador instanciará un *flow\_mod* por cada punto de acceso, y otro para el ECHO reply del ping.

```
mininet-wifi> dpctl dump-flows
*** ap1 ***
NXST_FLOW reply (xid=0x4):
*** ap2 ***
NXST_FLOW reply (xid=0x4):
*** ap3 ***
NXST_FLOW reply (xid=0x4):
mininet-wifi> dpctl dump-flows
*** ap1 ***
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=12.644s, table=0, n_packets=9, n_bytes=882, idle_timeout=60, idle_age=4, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:3
cookie=0x0, duration=11.640s, table=0, n_packets=8, n_bytes=784, idle_timeout=60, idle_age=4, priority=65535,icmp,in_port=3,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
cookie=0x0, duration=7.496s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=7, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1 actions=output:3
cookie=0x0, duration=7.489s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=7, priority=65535,arp,in_port=3,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=2 actions=output:1
*** ap2 ***
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=12.666s, table=0, n_packets=9, n_bytes=882, idle_timeout=60, idle_age=4, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
cookie=0x0, duration=12.665s, table=0, n_packets=9, n_bytes=882, idle_timeout=60, idle_age=4, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
cookie=0x0, duration=7.518s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=7, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1 actions=output:1
cookie=0x0, duration=7.515s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=7, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=2 actions=output:2
mininet-wifi>
```

Figura 1.29: Flow tables de los distintos puntos de acceso.

Como se puede apreciar en la figura, al principio de la comprobación de las tablas de cada punto de acceso estaban vacías. Esto es ya que estas entradas en la tabla tienen un tiempo de vida, agotado se eliminan de la tabla. Por lo que al repetir el ping se vuelve a repetir el proceso de instanciar los *Flow\_Mod* en cada tabla de cada punto de acceso para poder llevarse así a cabo el ping.

Pero resulta curioso el hecho de que en el punto de acceso 1 (ap1) no haya ninguna entrada en su tabla de Flows. Esto es ya que este punto de acceso no interviene ya que anteriormente hicimos un Handover de Sta1, entre ap1 y el ap2. Podemos comprobar que si volvemos a enlazar Sta1 con ap1, ap1 deberá tener también las entradas suficientes en su Flow table para poder llevar a cabo la conexión entre Sta1 y Sta3.

- `sta1 iw dev sta1-wlan0 disconnect`
- `sta1 iw dev sta1-wlan0 connect ssid_ap1`

```
mininet-wifi> dpctl dump-flows
*** ap1 ***
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=17.230s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, idle_age=13, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
cookie=0x0, duration=17.223s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, idle_age=13, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:2
cookie=0x0, duration=12.122s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=12, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1 actions=output:3
cookie=0x0, duration=12.098s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=12, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=2 actions=output:1
*** ap2 ***
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=17.247s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, idle_age=13, priority=65535,icmp,in_port=3,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
cookie=0x0, duration=17.230s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, idle_age=13, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:3
cookie=0x0, duration=12.131s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=12, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1 actions=output:3
cookie=0x0, duration=12.115s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=12, priority=65535,arp,in_port=3,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=2 actions=output:2
*** ap3 ***
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=17.266s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, idle_age=13, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
cookie=0x0, duration=17.245s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, idle_age=13, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
cookie=0x0, duration=12.143s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=12, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1 actions=output:1
cookie=0x0, duration=12.136s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=12, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=2 actions=output:2
mininet-wifi>
```

Figura 1.30: Flow tables de los distintos puntos de acceso.

Si este proceso lo hacemos lo suficientemente rápido nos daremos cuenta que al mover Sta1 el ping no funciona. Esto se debe a que las entradas de la tabla de Flows siguen configuradas para llevar un paquete desde ap3 hasta ap2, pero no a ap1. Si pasa el tiempo suficiente estas entradas serán desechadas o bien, podemos quitarlas nosotros para que se creen unas entradas en las tablas de Flows nuevas y que así se pueda alcanzar Sta1 desde su nueva posición.

- **dpctl del-flows**

Llegados a este punto, ya habríamos acabado el test 2, por lo que solo nos quedaría salir y limpiar Mininet-Wifi. Como ya se explicó:

- (Desde la CLI de Mininet-Wifi) `exit`
- `sudo mn -c`



## 1.7. Test 3: Python API y scripts

Mininet proporciona una API de Python para que los usuarios puedan crear secuencias de comandos de Python simples que configurarán topologías personalizadas en Mininet-WiFi. Mininet-WiFi extiende esta API para soportar un entorno inalámbrico a la ya existente API de Mininet para Python.



Cuando se usa el comando normal Mininet **mn** con la opción `—wifi` para crear Mininet-WiFi, no se tiene acceso a la mayoría de las funciones ampliadas que se proporcionan en Mininet-WiFi para crear topologías. Para acceder a las funciones que le permiten emular el comportamiento de los nodos en una LAN inalámbrica, se debe usar las extensiones de Mininet-Wifi para la API de Mininet de Python.

Las grandes diferencias que tiene la nueva API de Mininet-Wifi respecto a la anterior de Mininet, es que han añadido nuevos métodos al objeto de la topología, llamados *addStation* y *addAccessPoint*. Además han añadido código y han modificado gran parte de él para añadir el factor wireless a los enlaces, en el método *addLink*. La guía nos aconseja empezando a mirar los ejemplos de scripts ya hechos en Python.

### 1.7.1. Métodos básicos API Mininet-Wifi

A continuación, se va a exponer los métodos más básicos para trabajar con Mininet-Wifi. En los escenarios anteriores al ser escenarios por defecto se pudo apreciar que al no suministrar valores a los parámetros de los elementos de la red, se aplicaron los valores por defecto. Ahora desde los scripts creados se podrá especificar con exactitud los valores que queremos.

*#De esta manera añadiríamos una estación Wifi por defecto*

```
net.addStation( 'sta1' )
```

*#Con esta sentencia añadiríamos un nuevo punto de acceso,  
# pero a diferencia de los punto de acceso usados,  
# este no tendrá un SSID por defecto apX-ssid, si no, el suministrado.*

```
net.addAccessPoint( 'ap1', ssid='new_ssid' )
```

*#Para añadir un enlace Wireless entre ambos se hace con la siguiente  
# sentencia, pero el enlace tendrá los valores por defecto.*

```
net.addLink( ap1, sta1 )
```

Para crear escenarios más complicados y complejos se puede explotar todos los posibles parámetros que recogen estos métodos. Entre los cuales, se puede indicar, MAC, IP, localización (x,y,z), radio de alcance, y mucho más. Por ejemplo, el siguiente código define un punto de acceso y una estación, y crea una asociación

(una conexión inalámbrica) entre los dos nodos y aplica algunos parámetros de control de tráfico a la conexión a de radio realista, agregando restricciones de ancho de banda, una tasa de error de bit y un retardo de propagación.

```
# Añadimos una estación Wifi con un método de encriptado,
# MAC, Ip, posición, contraseña y nombre.

net.addStation( 'sta1', passwd='123456789a', encrypt='wpa2',
               mac='00:00:00:00:00:02', ip='10.0.0.2/8', position='50,30,0' )

# Añadimos un punto de acceso indicándole, método de encriptado,
# MAC, SSID, canal, modo de wifi (Entiendo que se refieren a que tipo de estándar IEEE
# de Wifi), posición, radio de cobertura.

net.addAccessPoint( 'ap1', passwd='123456789a', encrypt='wpa2', ssid=
                  'ap1-ssid', mode= 'g', channel= '1', position='30,30,0', range=30 )
```

Para activar la asociación de control en una red estática se debe usar el método que nos proporciona la api de Python de Mininet-Wifi que automáticamente elige el punto de acceso al cual una estación wifi debe ser conectado(Basado en la distancia entre estaciones Wifi y puntos de acceso). Por ejemplo podemos utilizar este criterio de elección **ssf** (Strongest signal first) para decidir desde una estación wifi a que punto de acceso debe ser conectada.

```
net.setAssociationCtrl( 'ssf' )
```

### 1.7.2. Métodos básicos API Mininet

La API de Python de Mininet sigue siendo compatible en la API de Mininet-Wifi. Añade switches host y controladores.

- Añadir un **Host**, la diferencia entre los host usados en Mininet-Wifi y los host convencionales de Mininet es que unos tienen una interfaz virtual de Ethernet y los de Mininet-Wifi tienen una interfaz virtual wireless.

```
net.addHost( 'h1' )
```

- Añadir un **Switch**, hay que tener en cuenta que los puntos de accesos usados en Mininet-Wifi operan como switches pero con interfaz virtual wireless.

```
net.addSwitch( 's1' )
```

- Añadir un **Link**, de tipo Ethernet entre dos nodos de la red. Por lo visto según la gui al enlazar dos Puntos de acceso por defecto se emplea en enlace de Ethernet.

```
net.addLink( s1, h1 )
```

- Añadir un controlador.

```
net.addController( 'c0' )
```

Podemos contemplar que con esta combinación de la API anterior de Mininet más los añadidos con Mininet-Wifi nos da la posibilidad de construir una topología que incluya host, switches, estaciones wifi, puntos de acceso y múltiples controladores.

### 1.7.3. Ejemplo script Mininet-Wifi

```

1  #!/usr/bin/python
2
3  from mininet.node import Controller
4  from mininet.log import setLogLevel, info
5  from mn_wifi.node import OVSKernelAP
6  from mn_wifi.cli import CLI_wifi
7  from mn_wifi.net import Mininet_wifi
8
9
10 def topology():
11
12     net = Mininet_wifi(controller=Controller, accessPoint=OVSKernelAP)
13
14     info("*** Creamos los nodos de la topologia\n")
15     sta1 = net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8',
16                          range=20, position='10,20,0')
17     sta2 = net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8',
18                          range=20, position='50,20,0')
19     ap1 = net.addAccessPoint('ap1', ssid='ssid-ap1', mode='g', channel='1',
20                             position='15,30,0', range=30)
21     ap2 = net.addAccessPoint('ap2', ssid='ssid-ap2', mode='g', channel='6',
22                             position='55,30,0', range=30)
23     c1 = net.addController('c1', controller=Controller)
24
25     #Config nodos
26     info("*** Configuramos los nodos de la topologia\n")
27     net.configureWifiNodes()
28     net.setAssociationCtrl('ssf')
29
30     #Matplotlib dibujo
31     net.plotGraph(max_x=100, max_y=100)
32
33     #Enlaces
34     info("*** Creamos los links\n")
35     net.addLink(ap1, ap2)
36     net.addLink(ap1, sta1)
37     net.addLink(ap2, sta2)
38
39     info("*** Arrancar la red\n")
40     net.build()
41     c1.start()
42     ap1.start([c1])
43     ap2.start([c1])
44
45     info("*** MininetWifi CLI ... \n")
46     CLI_wifi(net)
47
48     info("*** Parar la red\n")
49     net.stop()
50
51
52 if __name__ == '__main__':
53     setLogLevel('info')
54     topology()

```

En este ejemplo hemos tenido una dificultad, y es el método de la topología que establece el criterio de asociación entre estaciones Wifi y puntos de acceso. En la documentación lo expone según `associationControl()` y en la API se llama `setAssociationCtrl()` .

Se ha abierto un Issue indicándole a los creadores y a los desarrolladores que mantienen el proyecto si pueden cambiarlo o poner una nota para que la gente que inicia leyendo la documentación no tenga dificultades añadidas.

<https://github.com/intrig-unicamp/mininet-wifi/issues/226>

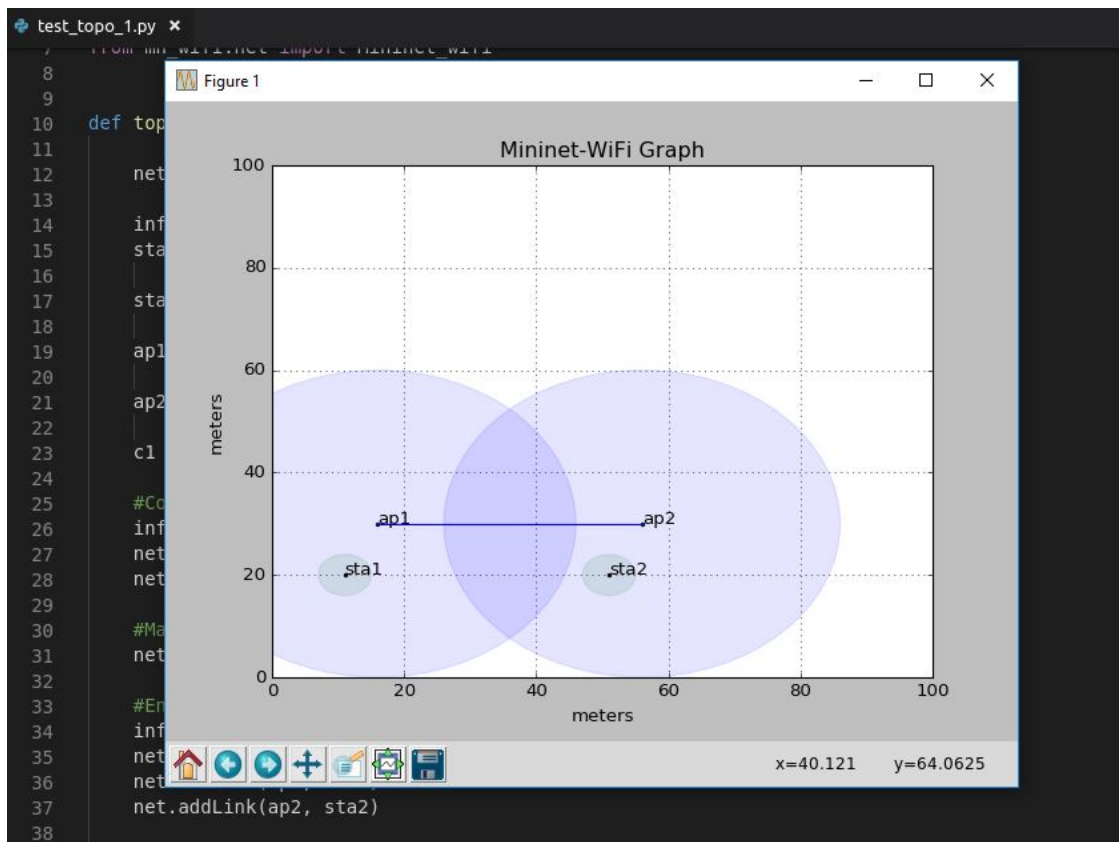


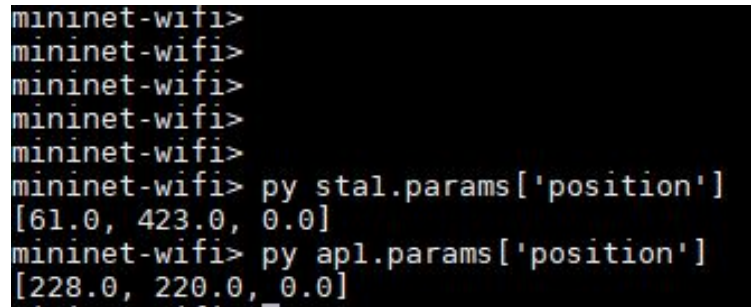
Figura 1.31: Resultado de ejecución del ejemplo.

El script lo hemos ejecutado desde la terminal haciendo uso del interprete de comandos. Lo hemos invocado según el siguiente comando:

- `sudo python test_topo.1.py`

También se podría haber ejecutado directamente siempre y cuando el Path del interprete estuviera bien puesto. Una vez ejecutado el script, tendremos una topología cargada en Mininet-Wifi de dos puntos de acceso y dos estaciones Wifi emparejadas a cada uno de estos puntos de acceso.

Como hemos indicado en el script que queríamos que cargase la CLI de Mininet-Wifi y que dibujara la topología, al completar la ejecución del script entrará en la CLI de Mininet-Wifi y con ayuda de matplotlib graficará la topología. Una vez dentro de la CLI de Mininet-Wifi podremos consultar los parámetros tales como la posición de un nodo de la topología. Esto lo podemos hacer haciendo del comando **py** que incorpora la CLI. Además, por lo que hemos podido ver, la CLI soporta código de python directamente haciendo uso del comando **py**. Esto será muy útil para consultar información o para hacer breves cambios en la red cuando el escenario ya está corriendo.



```
mininet-wifi>
mininet-wifi>
mininet-wifi>
mininet-wifi>
mininet-wifi> py sta1.params['position']
[61.0, 423.0, 0.0]
mininet-wifi> py ap1.params['position']
[228.0, 220.0, 0.0]
```

Figura 1.32: Consultar parámetros desde la CLI.

#### 1.7.4. Cambiando la red durante el tiempo de ejecución

Mininet-WiFi proporciona funciones de Python que se pueden usar durante el tiempo de ejecución para realizar cambios en el nodo en lo relativo a posiciones y asociaciones. Estas funciones son útiles cuando tenemos una configuración estática y queremos hacer cambios puntuales.

Para cambiar el punto de acceso al que está asociada una estación wifi (siempre que la estación Wifi esté en el rango del punto de acceso a cambiar):

```
mininet-wifi> py sta1.setAssociation('sta1-wlan0', ap1)
```

Para mover una estación Wifi o un punto de acceso dado unas coordenadas en el espacio. Formato (X,Y,Z).

```
mininet-wifi> py sta1.setPosition('40,20,40')
```

También podemos modificar a tiempo real el rango de un punto de acceso.

```
mininet-wifi> py sta1.setRange(100)
```

Las modificaciones que hagamos vía CLI repercutirán directamente en el funcionamiento de la red. Un cambio de posición de asociación puede ser clave en el desempeño en la operativa de funcionamiento de la red.

Llegados a este punto, ya habríamos acabado el test 3, por lo que solo nos quedaría salir y limpiar Mininet-Wifi. Como ya se explicó:

- (Desde la CLI de Mininet-Wifi) exit
- sudo mn -c

## 1.8. Test 4

En este test se va a explorar una de las funcionalidades más importantes de Mininet-Wifi, la movilidad. Mininet-Wifi proporciona nuevos métodos en su API de Python, como *startMobility* y *Mobility*, con lo que podemos especificar una amplia variedad de escenarios de LAN inalámbrica mediante el control movimiento de la estación Wifi.

### 1.8.1. Python API y métodos de movilidad

La API de Python de Mininet-WiFi agrega nuevos métodos que permiten al usuario crear estaciones que se mueven alrededor en el espacio virtual cuando se ejecuta un escenario de emulación.

Para mover una estación en línea recta, use los métodos **net.startMobility**, **net.mobility** y **net.stopMobility**. Por ejemplo, para mover una estación Wifi de una posición a otra durante un período de 60 segundos.

```
net.startMobility( time=0 )
net.mobility( sta1, 'start', time=1, position='10,20,0' )
net.mobility( sta1, 'stop', time=59, position='30,50,0' )
net.stopMobility( time=60 )
```

Mininet-WiFi también puede mover estaciones wifi de forma automática según los modelos de movilidad predefinidos. Los modelos de movilidad disponibles son:

- RandomWalk
- TruncatedLevyWalk
- RandomDirection
- RandomWayPoint
- GaussMarkov
- ReferencePoint
- TimeVariantCommunity

Por ejemplo, para mover una estación en un área de 60 metros por 60 metros, con una velocidad mínima de 0.1 metros por segundo y una velocidad máxima de 0.2 metros por segundo.

```
net.setMobilityModel(time=0, model='RandomDirection', max_x=60,
max_y=60, min_v=0.1, max_v=0.2)
```

Mininet-WiFi conectará y desconectará automáticamente las estaciones desde y hacia los puntos de acceso en función de la intensidad de señal calculada o el nivel de carga en ese punto de acceso.

Para utilizar un método de control de asociación, necesita agregue el parámetro AC en `net.setMobilityModel()`. Por ejemplo, para cambiar los puntos de acceso basados en los criterios de "menos cargado primero":

```
net.setMobilityModel(time=0, model='RandomWayPoint', max_x=140,
max_y=140, min_v=0.7, max_v=0.9, AC='llf')
```

Por lo que tendríamos dos criterios de asociación a un punto de acceso.

- llf (Least-Loaded-First)
- ssf (Strongest-Signal-First)

### 1.8.2. Ejemplo script movilidad

```

1  #!/usr/bin/python
2
3
4
5  from mininet.node import Controller
6  from mininet.log import setLogLevel, info
7  from mn_wifi.net import Mininet_wifi
8  from mn_wifi.node import OVSKernelAP
9  from mn_wifi.cli import CLI_wifi
10
11
12
13  def topo():
14
15
16      net = Mininet_wifi(controller= Controller, accessPoint = OVSKernelAP)
17
18      info("*** Creando nodos\n")
19      h1 = net.addHost( 'h1', mac='00:00:00:00:00:01',ip='10.0.0.1/8')
20
21      sta1 = net.addStation( 'sta1', mac='00:00:00:00:00:02',ip='10.0.0.2/8', range = '20')
22
23      ap1 = net.addAccessPoint('ap1', ssid = 'ap1-ssid', mode='g', channel= '1',
24      position='30,50,0', range='30')
25
26      ap2 = net.addAccessPoint('ap2', ssid = 'ap2-ssid', mode='g', channel= '1',
27      position='90,50,0', range='30')
28
29      ap3 = net.addAccessPoint('ap3', ssid = 'ap3-ssid', mode='g', channel= '1',
30      position='150,50,0', range='30')
31
32      c1 = net.addController('c1' , controller = Controller)
33
34      info("*** Configurando nodos\n")
35      net.configureWifiNodes()
36
37
38      info("*** Creando links\n")
39      net.addLink(ap1, h1)
40      net.addLink(ap1, ap2)
41      net.addLink(ap2,ap3)
42
43      info("*** Dibujando Topo con matplotlib\n")
44      net.plotGraph(max_x=200, max_y = 200)
45
46      info("*** Configurando Movilidad (sta1)\n")
47      net.startMobility(time = 0, AC='ssf')
48      net.mobility(sta1, 'start', time = 20, position='1,50,0')
49      net.mobility(sta1, 'stop', time = 79, position='159,50,0')
50      net.stopMobility(time=200)
51
52      info("*** Network UP!\n")
53      net.build()
54      c1.start()
55      ap1.start([c1])
56      ap2.start([c1])
57      ap3.start([c1])
58
59

```

```

60
61     info("*** Cargamos Mininet-Wifi CLI\n")
62     CLI_wifi(net)
63
64     info("*** Parando network\n")
65     net.stop()
66
67
68 if __name__ == '__main__':
69     setLogLevel('info')
70     topo()

```

Nos hemos dado cuenta que para que complete su recorrido desde  $x=1$  a  $x=159$  debemos darle más tiempo de movilidad por lo que hemos aumentado el periodo de movilidad hasta 200 segundos, ya que con los 79 segundos apenas llegaba a la mitad del recorrido.

[https://github.com/davidcawork/testMininet-Wifi/blob/master/doc/Topologies\\_img/test\\_movilidad\\_1.gif](https://github.com/davidcawork/testMininet-Wifi/blob/master/doc/Topologies_img/test_movilidad_1.gif)

Podemos hacer uso del comando **py** de la CLI de Mininet-Wifi para obtener información relativa a que punto de acceso estamos en rango, esto es muy útil para ver en que momento la estación Wifi está en un conflicto y debe aplicar el criterio de selección para decidir a que punto de acceso conectarse.

```

mininet-wifi> py sta1.params['apsInRange']
[<OVSAP ap1: 10:127.0.0.1,ap1-wlan1:None,ap1-eth2:None,ap1-eth3:None pid=28721> ]
mininet-wifi> py sta1.params['apsInRange']
[<OVSAP ap2: 10:127.0.0.1,ap2-wlan1:None,ap2-eth2:None,ap2-eth3:None pid=28727> ]

```

*# Handover entre dos células de Ap1 a Ap2*

### 1.8.3. Prueba con la herramienta Iperf

En esta prueba se va a utilizar la topología expuesta en el punto anterior(test\_movilidad\_1.py). La operativa va a ser de generar tráfico TCP entre una estación Wifi y un host estático con Iperf. Nuestro servidor será la estación wifi dotada de movilidad, que irá recorriendo los tres puntos de acceso ap1, ap2 y ap3.

El objetivo de la prueba es ver en el momento del handover entre dos puntos de acceso, como el tráfico deja de fluctuar entre servidor y cliente. Esto se debe a que las reglas instauradas por el controlador en el ap1 dejan de ser útiles ya que este se encuentra en el dominio del ap2 por lo que el trafico deja de fluctuar. Para resolverlo podemos eliminar las reglas instauradas con **dpctl del-flows**. Pasos a seguir:

- sudo python test\_movilidad\_1.py
- xterm sta1
- xterm h1
- (En la consola de sta1) iperf -server
- (En la consola de h1) iperf -client 10.0.0.2 -time 60 -interval 2
- Observar como deja de fluctuar el tráfico tras producirse el handover.
- dpctl del-flows
- Se recupera el tráfico.



```
n0obie@n0obie-VirtualBox:~/repo_mininet_wifi/testMininet-Wifi/scripts$ sudo python test_movilidad_1.py
*** Creando nodos
*** Configurando nodos
*** Creando links
*** Dibujando Topo con matplotlib
*** Configurando Movilidad (stal)
*** Network UP!
*** Configuring nodes
*** Cargamos Mininet-Wifi CLI
*** Starting CLI:
mininet-wifi>
```

Figura 1.33: Levantamos la topología.

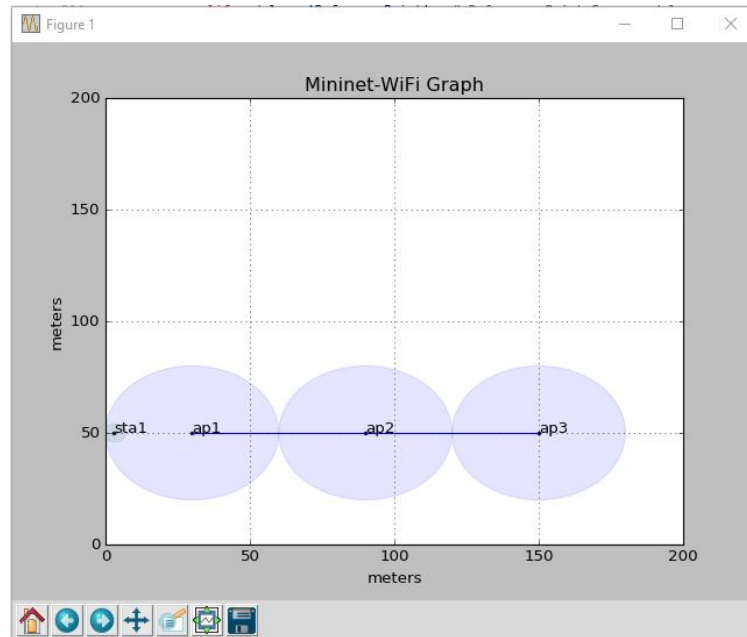


Figura 1.34: Topología Inicial.

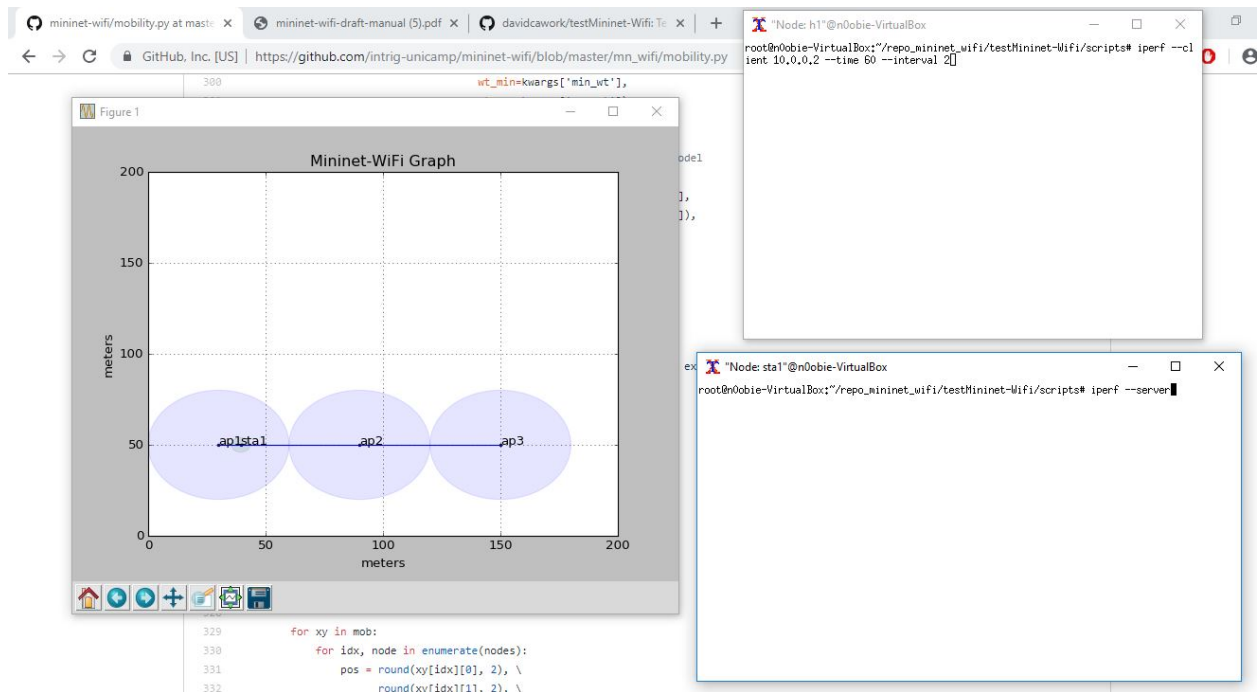


Figura 1.35: Lanzamos las consolas en Sta1 y h1.

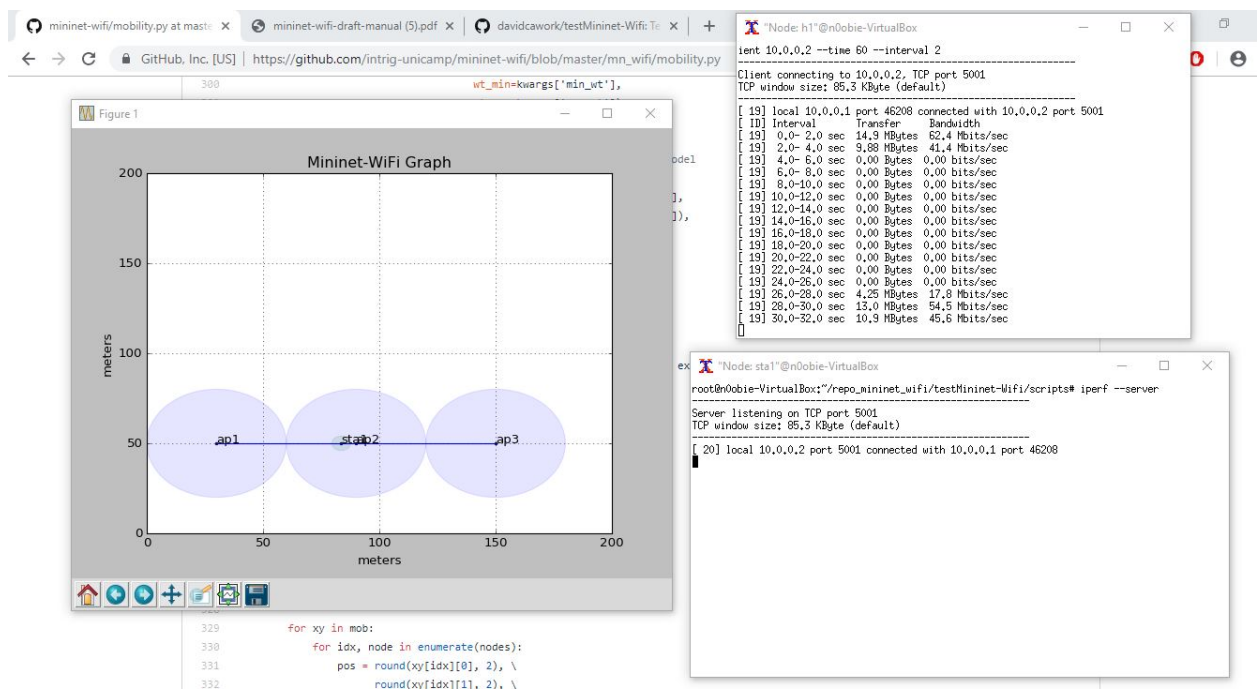


Figura 1.36: Creamos tráfico entre Sta1 y h1.

```

mininet-wifi> dpctl dump-flows
*** ap1 ***
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=73.933s, table=0, n_packets=32438, n_bytes=206463580, idle_timeout=60, idle_age=39, priority=65535,tcp,in_port=2,vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=46208,tp_dst=5001 actions=output:3
 cookie=0x0, duration=73.920s, table=0, n_packets=41220, n_bytes=2729928, idle_timeout=60, idle_age=39, priority=65535,tcp,in_port=3,vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=46208 actions=output:2
 cookie=0x0, duration=32.923s, table=0, n_packets=32216, n_bytes=214809976, idle_timeout=60, idle_age=0, priority=65535,tcp,in_port=2,vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=46210,tp_dst=5001 actions=output:3
 cookie=0x0, duration=32.914s, table=0, n_packets=64674, n_bytes=4268492, idle_timeout=60, idle_age=0, priority=65535,tcp,in_port=3,vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=46210 actions=output:2
 cookie=0x0, duration=59.589s, table=0, n_packets=2, n_bytes=84, idle_timeout=60, idle_age=0, priority=65535,arp,in_port=3,vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=1 actions=output:2
 cookie=0x0, duration=59.571s, table=0, n_packets=2, n_bytes=84, idle_timeout=60, idle_age=0, priority=65535,arp,in_port=2,vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op=2 actions=output:3
*** ap2 ***
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=73.971s, table=0, n_packets=32438, n_bytes=206463580, idle_timeout=60, idle_age=39, priority=65535,tcp,in_port=2,vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=5001,tp_dst=46208 actions=output:1
 cookie=0x0, duration=73.963s, table=0, n_packets=41220, n_bytes=2729928, idle_timeout=60, idle_age=39, priority=65535,tcp,in_port=1,vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=46208 actions=output:2
 cookie=0x0, duration=32.962s, table=0, n_packets=32216, n_bytes=214809976, idle_timeout=60, idle_age=0, priority=65535,tcp,in_port=2,vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=46210,tp_dst=5001 actions=output:1
 cookie=0x0, duration=32.958s, table=0, n_packets=64674, n_bytes=4268492, idle_timeout=60, idle_age=0, priority=65535,tcp,in_port=1,vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=46210 actions=output:2
 cookie=0x0, duration=59.635s, table=0, n_packets=2, n_bytes=84, idle_timeout=60, idle_age=0, priority=65535,arp,in_port=1,vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=1 actions=output:2
 cookie=0x0, duration=59.603s, table=0, n_packets=2, n_bytes=84, idle_timeout=60, idle_age=0, priority=65535,arp,in_port=2,vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op=2 actions=output:1
*** ap3 ***
NXST_FLOW reply (xid=0x4):
mininet-wifi>

```

Figura 1.37: Tablas antes del Handover.

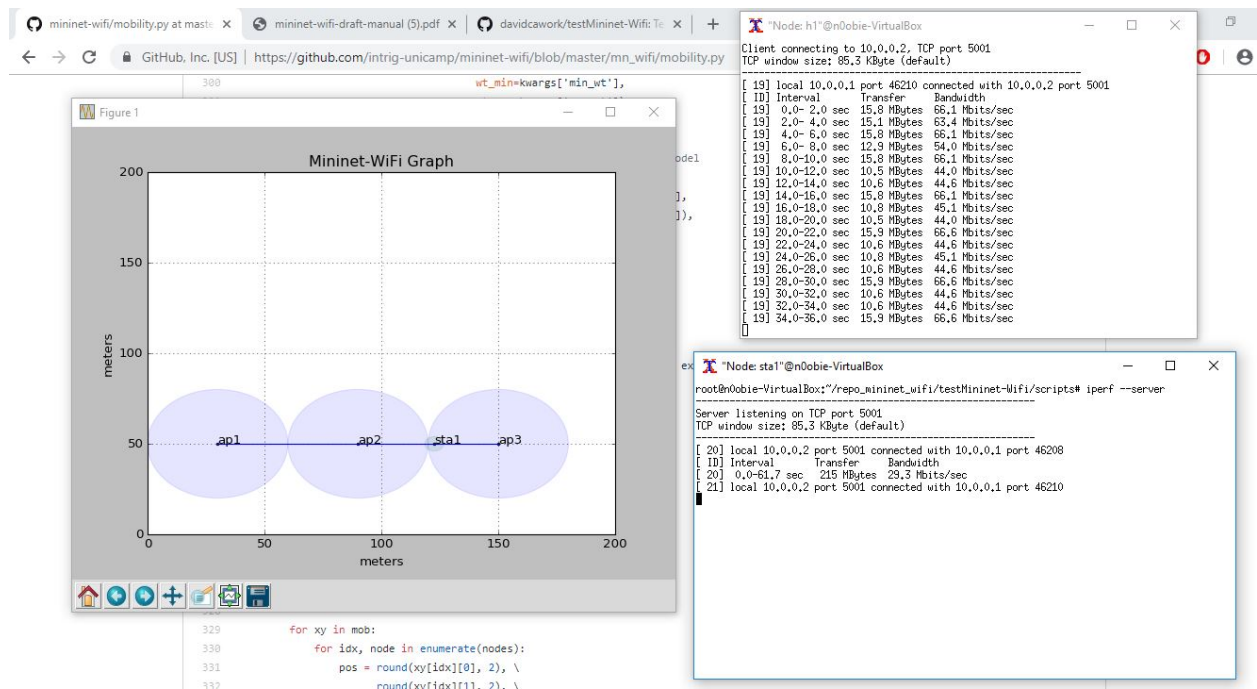


Figura 1.38: Handover realizado el tráfico se detiene.

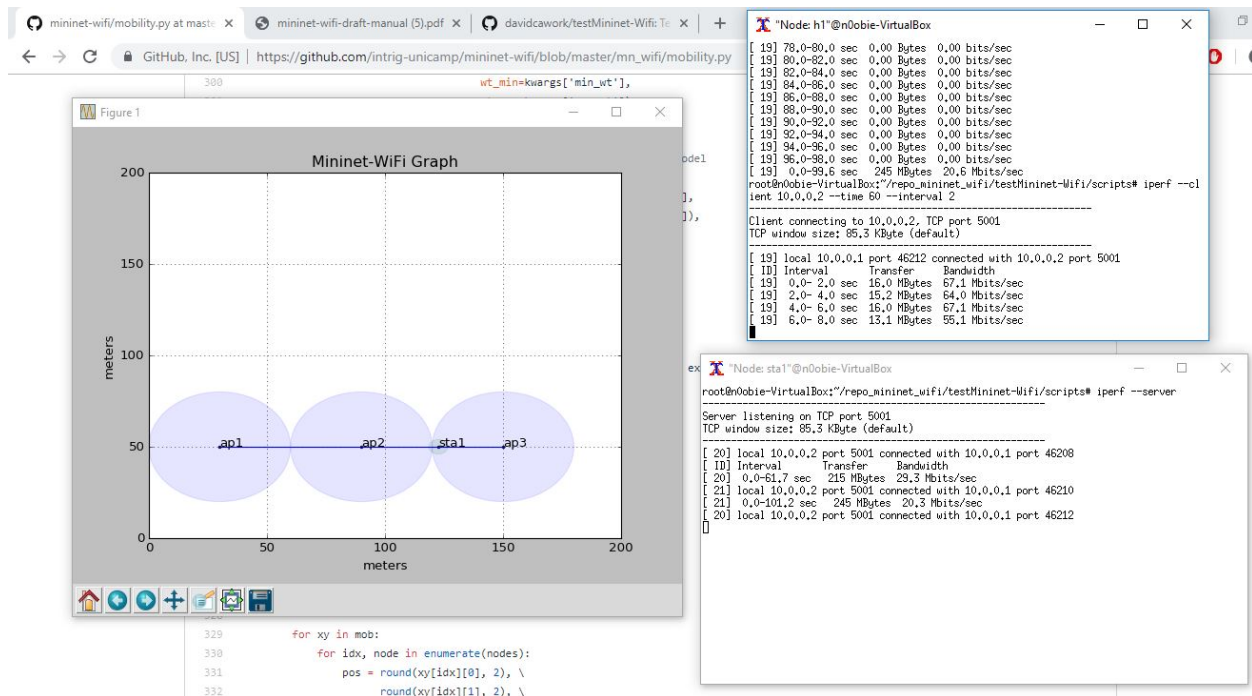


Figura 1.39: Borrarnos las tablas, el tráfico se recupera pasados unos segundos.

Llegados a este punto, ya habríamos acabado el test 4, por lo que solo nos quedaría salir y limpiar Mininet-Wifi. Como ya se explicó:

- (Desde la CLI de Mininet-Wifi) exit
- sudo mn -c

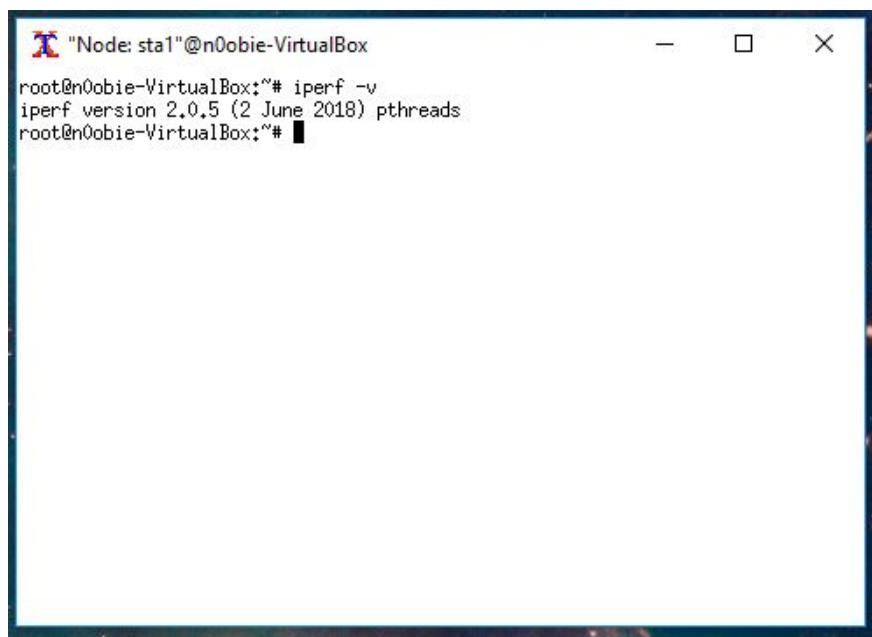
# Anexos



## Apéndice A

# Herramienta IPerf

IPerf es una herramienta de evaluación del rendimiento en las comunicaciones en una red. Esta evaluación la llevará a cabo por medio de del trafico generado y el tiempo transcurrido entre cliente servidor. Hay distintas posibilidades a la hora de generar tráfico, tráfico TCP por defecto o UDP indicándolo debidamente en sus parámetros opcionales. La arquitectura de la aplicación se estructura en un nodo cliente y otro servidor. En función de la parte de la herramienta tendrá una serie de parámetros y opciones con respecto a la otra parte en cuestión.

A screenshot of a terminal window titled '"Node: sta1"@n0obie-VirtualBox'. The terminal shows the command 'root@n0obie-VirtualBox:~# iperf -v' being executed, which returns the output 'iperf version 2.0.5 (2 June 2018) pthreads'. The prompt 'root@n0obie-VirtualBox:~#' is visible again on the next line.

```
"Node: sta1"@n0obie-VirtualBox
root@n0obie-VirtualBox:~# iperf -v
iperf version 2.0.5 (2 June 2018) pthreads
root@n0obie-VirtualBox:~#
```

Figura A.1: Versión de IPerf a utilizar.

### A.1. Opciones

Parámetros parte **servidor**, para especificar que se trata de esta parte de la herramienta hay que indicarlo con la opción **-s** ó **-server**. Se van a indicar los más importantes, si se requiere más información consulte la pagina del desarrollador.

<https://iperf.fr/iperf-doc.php>

Opciones en el servidor:

- -D, como servicio (En Unix como un daemon).
- -R, remover el servicio.
- -u, recibir datagramas UDP en vez de TCP por defecto.
- -P, indicando el número de conexiones simultáneas.
- -m, muestra MTU.
- -w, especifica el tamaño de Ventana (TCP window size).
- -f, [bkmBKB] Cambiar unidades los resultados en **bits/s**, **kilobits/s**, **megabytes/s**, **Bytes/s**, **KiloBytes/s**, **MegaBytes/s**.

Parámetros parte **cliente**, para especificar que se trata de esta parte de la herramienta hay que indicarlo con la opción **-c** ó **-client**. Se van a indicar los más importantes, si se requiere más información, como se ha indicado anteriormente, consulte la pagina del desarrollador.

Opciones en el cliente:

- -t, segundos tiempo duración transmisión. Hace más fiable la medida.
- -i, segundos especifica un intervalo, medido en segundos, en el cual se volverá a realizar la medición.
- -T, ttl especifica valor TTL.
- -m, muestra MTU.
- -w, especifica el tamaño de Ventana (TCP window size).
- -f, [bkmBKB] Cambiar unidades los resultados en **bits/s**, **kilobits/s**, **megabytes/s**, **Bytes/s**, **KiloBytes/s**, **MegaBytes/s**.

## A.2. Ejemplo de uso

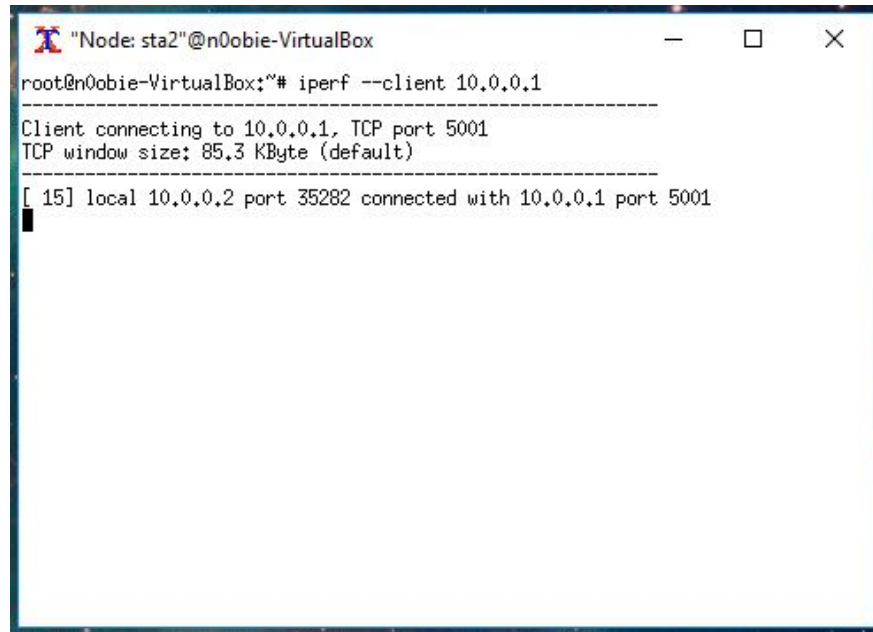
Lo primero que debemos hacer es escoger dos nodos de la red y decidir quien va a ser el servidor y quien será el cliente. Una vez tomada esa decisión debemos poner a escuchar a la parte servidor. Esto lo haremos según hemos indicado antes. Podemos ver como el servidor está a la escucha en el puerto 5001.



Figura A.2: Levantar parte servidor.



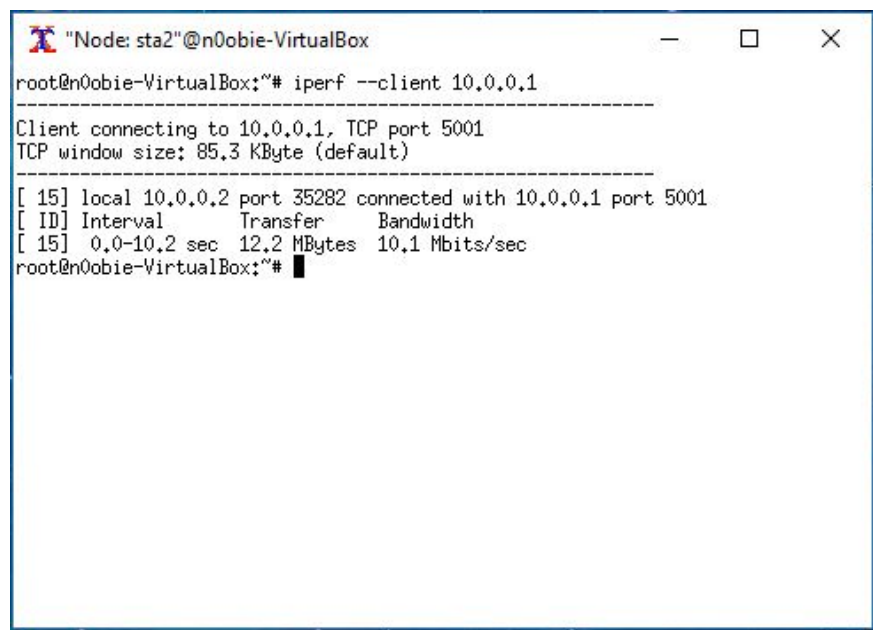
Acto seguido vamos al otro nodo e indicamos su condición de cliente, a quien debe conectarse, y en el caso que lo queramos las configuraciones extras que consideremos oportunas.



```
"Node: sta2"@n0obie-VirtualBox
root@n0obie-VirtualBox:~# iperf --client 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 15] local 10.0.0.2 port 35282 connected with 10.0.0.1 port 5001
█
```

Figura A.3: Levantar parte cliente y conectarla con el servidor.

Una vez completado el test de performance podremos ver los resultados de ancho de banda obtenidos. En este caso nos lo indica en las unidades por defecto pero haciendo uso del flag `-f` podemos cambiar el formato de las unidades de los resultados obtenidos.



```
"Node: sta2"@n0obie-VirtualBox
root@n0obie-VirtualBox:~# iperf --client 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 15] local 10.0.0.2 port 35282 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 15]  0.0-10.2 sec  12.2 MBytes  10.1 Mbits/sec
root@n0obie-VirtualBox:~# █
```

Figura A.4: Resultados en el cliente.