The User Manual

Mininet-WiFi
Emulator for Software-Defined Wireless Networks

Draft Release

Ramon dos Reis Fontes and Christian Esteve Rothenberg

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING (FEEC), UNIVERSITY OF CAMPINAS - UNICAMP - BRAZIL

GITHUB.COM/INTRIG-UNICAMP/MININET-WIFI

Mininet-Wifi is being developed as a clean extension of the high-fidelity Mininet emulator by adding the new abstractions and classes to support wireless NICs and emulated links while conserving all native lightweight virtualization and OpenFlow/SDN features.

draft release, June 2016

Mininet-WiFi

Emulator for Software-Defined Wireless Networks

Contents

Mininet-WiFi

Emulator for Software-Defined Wireless Networks

1. About Mininet-WiFi

Mininet-WiFi is a fork of the Mininet SDN network emulator and extended the functionality of Mininet by adding virtualized WiFi stations and access points based on the standard Linux wireless drivers and the 80211_hwsim wireless simulation driver. We 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.

1.1 Requirements

Mininet-WiFi should work fine in any Ubuntu Distribution from 14.04, though we have been perceived some difficulties in Ubuntu 16.04.

1.1.1 known issues

If you are using a Virtual Machine probably you have to stop network-manager. You may try these commands in order to stop it:

```
sudo stop network-manager
```

or

```
sudo systemctl stop network-manager
```

or

```
sudo service network-manager stop
```

1.2 Installing Mininet-WiFi

1.2.1 GitHub

You have to follow only for 4 steps to install Mininet-WiFi:

- sudo apt-get install git
- git clone https://github.com/intrig-unicamp/mininet-wifi
- cd mininet-wifi
- sudo util/install.sh -Wnfv

Mininet WiFi is installed by a script. Run the script with the `-h help` option to see all the options available.

```
wifi:~$ util/install.sh -h
```

1.2.2 Docker

Mininet-WiFi is available on Docker.

1.3 Limitations

Mininet-WiFi inherits all limitations of Mininet, including:

- You cannot handle with packets that go out to the incoming port with the OpenFlow protocol.
- There is a limit of 100 wifi nodes. If you want to use more than 100 wifi nodes you have to recompile the module mac80211_hwsim - see function init_mac80211_hwsim.
- WDS -v Wireless Distribution System is not supported.

1.4 Architecture and Components

The main components that make part of the development of Mininet-WiFi are illustrated in Figure 1.1. In the kernel-space the module *mac80211_hwsim* is responsible for creating virtual Wi-Fi interfaces, important for stations and access points. Continuing in the kernel-space, MLME (*Media Access Control Sublayer Management Entity*)[1] is realized in the stations side, while in the user-space the *hostapd* is responsible for this task in the AP side.

Mininet-WiFi also uses a couple utilities such as *iw*, *iwconfig* e o *wpa_supplicant*. The first two are used for interface configuration and for getting information from wireless interfaces and the last one is used with *Hostapd*, in order to support WPA (*Wi-Fi Protected Access*), among other things. Besides them, another fundamental utility is *TC* (*Traffic Control*). The *TC* is an user-space utility program used to configure the Linux kernel packet scheduler, responsible for controlling the rate, delay, latency and loss, applying these attributes in virtual interfaces of stations and APs, representing with higher fidelity the behavior of the real world.

Figure 1.2 depicts the components and connections in a simple topology with two hosts created with Mininet-WiFi, where the newly implemented components (highlighted in gray) are presented along the original Mininet building blocks.

More specifically, we added WiFi interfaces on stations that now are able to connect to an access point through its (`wlanX`) interface that is bridged to an OpenFlow switch with AP capabilities represented by (`ap1`). Similar to Mininet, the virtual network is created by placing host processes in

[1] some of the functions performed by MLME are authentication, association, sending and receiving *beacons*, etc.

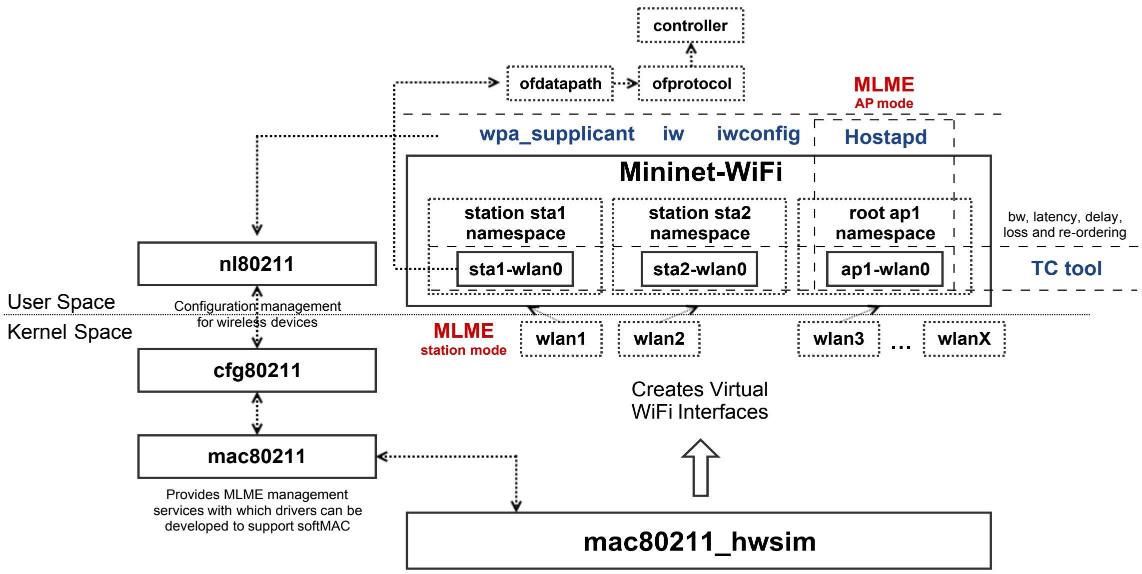

Figure 1.1: Mininet-WiFi Components.

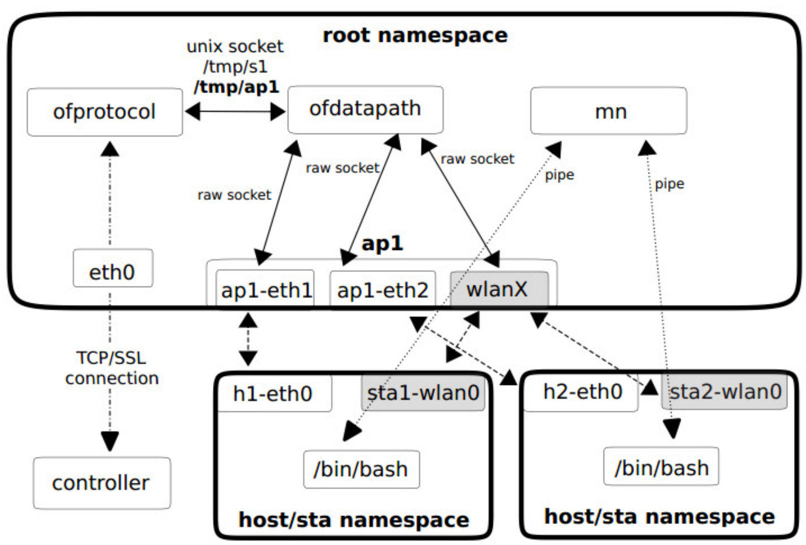

Figure 1.2: Components and connections in a two-host network created with Mininet-WiFi.

Linux OS network namespaces interconnected through virtual Ethernet (veth) pairs. The wireless interfaces to virtualize WiFi devices work on *master* mode for access points and *managed* mode for stations.

Stations: Are devices that connect to an access point through authentication and association. In our implementation, each station has one wireless card (`staX-wlan0` - where X shall be replaced by the number of each station). Since the traditional Mininet hosts are connected to an access point, stations are able to communicate with those hosts.

Access Points: Are devices that manage associated stations. Virtualized through `hostapd`[2] daemon and use virtual wireless interfaces for access point and authentication servers. While virtualized access points do not have (yet) APIs allowing users to configure several parameters in the same fashion of a real one, the current implementation covers the most important features, for example ssid, channel, mode, password, cryptography, etc.

Both stations and access points use `cfg80211` to communicate with the wireless device driver, a Linux 802.11 configuration API that provides communication between stations and `mac80211`. This framework in turn communicates directly with the WiFi device driver through a `netlink` socket (or more specifically `nl80211`) that is used to configure the `cfg80211` device and for kernel-user-space communication as well.

1.4.1 Files

- mininet/wifiAccessPoint.py - all implementation related to AP
- mininet/wifiAssociationControl.py - association Control techniques
- mininet/wifiChannel.py - channel details (including interference)
- mininet/wifiDevices.py - specification of real devices
- mininet/wifiMeshRouting.py - wireless mesh routing
- mininet/wifiMobility.py - mobility parameters
- mininet/wifiMobilityModels.py - mobility models
- mininet/wifiModule.py - module
- mininet/wifiPlot.py - graphs
- mininet/wifiPropagationModels.py - propagation models
- mininet/vanet.py - VANET networks

1.4.2 Important classes

- *addHost():* adds a host to a topology and returns the host name
- *addStation():* adds a station to a topology and returns the station name
- *addBaseStation():* adds a base station to a topology and returns the base station name
- *addPhysicalBaseStation():* attach a physical usb interface to a virtual base station to a topology and returns the physical base station name
- *addSwitch():* adds a switch to a topology and returns the switch name
- *addLink():* adds a bidirectional link to a topology (and returns a link key, but this is not important). Links in Mininet-WiFi are bidirectional unless noted otherwise.
- *plotGraph():* use this class if you want to plot graph.
- *startMobility:* useful when you want to use a mobility model.

1.5 Creating Network

You can create a simple network (2 stations and 1 access point) with the following command:

```
sudo mn --wifi
```

The command above can be utilized with other parameters, like in:

```
sudo mn --wifi --ssid=new_ssid --mode=g --channel=1
```

[2]Hostapd (**H**ost **A**ccess **P**oint **D**aemon) user space software capable of turning normal wireless network interface cards into access points and authentication servers

You can also use just mn if you want to work only with Mininet instead of Mininet-WiFi.

1.5.1 Changing Topology Size and Type

The default topology is a single Access Point connected to two Stations. You could change this to a different topo with –topo and pass parameters for that topology's creation. For example, to verify all-pairs ping connectivity with one Access Point and five Stations:

```
sudo mn --wifi --test pingall --topo single,5
```

Another example, with a linear topology (where each Access Point has one host, and all Access Points connect in a line via wired media):

```
sudo mn --wifi --test pingall --topo linear,5
```

1.5.2 Examples

If you are just beginning to write scripts for Mininet-WiFi, you can use the example scripts as a starting point. We created example scripts in the `/mininet-wifi/examples` directory that show how to use most of the features in Mininet-WiFi.

1.5.3 Getting information from nodes

getting basic information

```
mininet-wifi>info sta1
```

getting the position:

```
mininet-wifi>py sta1.params['position']
```

getting AP that a specific station is associated:

```
mininet-wifi>py sta1.params['associatedTo']
```

getting APs in range:

```
mininet-wifi>py sta1.params['apsInRange']
```

getting the channel:

```
mininet-wifi>py sta1.params['channel']
```

getting the frequency:

```
mininet-wifi>py sta1.params['frequency']
```

getting the mode:

```
mininet-wifi>py sta1.params['mode']
```

getting the rssi:

```
mininet-wifi>py sta1.params['rssi']
```

getting the Tx Power:

```
mininet-wifi>py sta1.params['txpower']
```

changing the associated AP:

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

changing the position:

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

changing the signal range

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

changing the signal range

```
mininet-wifi>py sta1.ssid
```

getting the stations associated to AP1:

```
mininet-wifi>py ap1.associatedStations
```

1.5.4 Videos

You can find many videos about Mininet-WiFi in a channel on youtube.

1.6 Contact Us

You are invited to participate of our mailing list.

Mininet-WiFi

Emulator for Software-Defined Wireless Networks

2. Tutorial

2.1 Introduction

This tutorial has been developed by Brian Linkletter (http://www.brianlinkletter.com/mininet-wifi-software-defined-network-emulator-supports-wifi-networks). We thank Brian for his time and this helpful tutorial.

2.2 First ideas

In this post, I describe the unique functions available in the Mininet-WiFi network emulator and work through a few tutorials exploring its features.

2.2.1 How to read this post

In this post, I present the basic functionality of Mininet-WiFi by working through a series of tutorials, each of which works through Mininet-WiFi features, while building on the knowledge presented in the previous tutorial. I suggest new users work through each tutorial in order.

I do not attempt to cover every feature in Mininet-WiFi. Once you work through the tutorials in this post, you will be well equipped to discover all the features in Mininet-WiFi by working through the `Mininet-WiFi example scripts-` https://github.com/intrig-unicamp/mininet-wifi/tree/master/examples, and reading the `Mininet-WiFi wiki`[1] and `mailing list`[2].

I assume the reader is already familiar with the [Mininet network emulator](http://mininet.org/) so I cover only the new WiFi features added by Mininet-WiFi. If you are not familiar with Mininet, please read my `Mininet network simulator review`[3] before proceeding. I have also written `many other posts about Mininet`[4].

---

[1] https://github.com/intrig-unicamp/mininet-wifi/wiki
[2] https://groups.google.com/forum/#!forum/mininet-wifi-discuss
[3] http://www.brianlinkletter.com/mininet-test-drive/
[4] http://www.brianlinkletter.com/tag/mininet/

I start by discussing the functionality that Mininet-WiFi adds to Mininet: Mobility functions and WiFi interfaces. Then I show how to install Mininet-WiFi and work through the tutorials listed below:

Tutorial #1: One access point shows how to run the simplest Mininet-WiFi scenario, shows how to capture wireless traffic in a Mininet-Wifi network, and discusses the issues with OpenFlow and wireless LANs.

Tutorial #2: Multiple access points shows how to create a more complex network topology so we can experiment with a very basic mobility scenario. It discusses more about OpenFlow and shows how the Mininet reference controller works in Mininet-WiFi.

Tutorial #3: Python API and scripts shows how to create more complex network topologies using the Mininet-WiFi Python API to define node positions in space and other node attributes. It also discusses how to interact with nodes running in a scenario with the Mininet-WiFi CLI, the Mininet-WiFi Python interpreter, and by running commands in a node's shell.

Tutorial #4: Mobility shows how to create a network mobility scenario in which stations move through space and may move in and out of range of access points. It also discusses the available functions that may be used to implement different mobility models using the Mininet-WiFi Python API.

2.2.2 Mininet-WiFi compared to Mininet

Mininet-WiFi is an extension of the Mininet software defined network emulator. The Mininet-WiFi developer did not modify any existing Mininet functionality, but added new functionality.

2.2.3 Mininet-WiFi and Mobility

Broadly defined, mobility in the context of data networking refers to the ability of a network to accommodate hosts moving from one part of the network to another. For example: a cell phone user may switch to a wifi access point when she walks into a coffee shop; or a laptop user may walk from her office in one part of a building to a meeting room in another part of the building and still being able to connect to the network via the nearest WiFi access point.

While the standard Mininet network emulator may be used to test mobility (In the Mininet examples folder, we find a `mobility.py` script that demonstrates methods that may be used to create a scenario where a host connected to one switch moves its connection to another switch), Mininet-WiFi offers more options to emulate complex scenarios where many hosts will be changing the switches to which they are connected. Mininet-WiFi adds new classes that simplify the programming work required by researchers to create Mobility scenarios.

Mininet-WiFi does not modify the reference SDN controller provided by standard Mininet so the reference controller cannot manage the mobility of users in the wireless network. Researchers must use a remote controller that supports the CAPWAP protocol (NOTE: I've not tried this and I do not know if it will work without modifications or additional programming), or manually add and delete flows in the access points and switches.

2.2.4 802.11 Wireless LAN Emulation

Mininet-wifi incorporates the Linux 802.11 SoftMAC[5] wireless drivers, the cfg80211[6] wireless configuration interface and the mac80211_hwsim[7] wireless simulation drivers in its access points.

The `mac80211_hwsim` driver is a software simulator for Wi-Fi radios. It can be used to `create virtual wi-fi interfaces`[8] that use the `802.11 SoftMAC wireless LAN driver`[9]. Using this tool, researchers may `emulate a Wi-Fi link between virtual machines`[10] - some mac80211_hwsim practical examples and supporting information are at the following links: `lab`[11], `thesis`[12], `hostapd`[13], `wpa-supplicant`[14], `docs-1`[15], and `docs-2`[16]. The 80211_hwsim driver enables researchers to emulate the wifi protocol control messages passing between virtual wireless access points and virtual mobile stations in a network emulation scenario. By default, 80211_hwsim simulates perfect conditions, which means there is no packet loss or corruption.

You can use Wireshark to `monitor wireless traffic`[17] 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`[18] before making traffic on these interfaces available to user processes like Wireshark. You will have to install additional tools and follow a complex procedure to `enable monitoring of WiFi traffic on the ap1-wlan0 interface`[19]. An easier method is available: look for the `hwsim0` interface on an access point, enable it, and monitor traffic on it. The `hwsim0` 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 traffic[20]. We'll see this in the examples we work through, below.

2.2.5 Mininet-WiFi display graph

Since locations of nodes in space is an important aspect of WiFi networks, Mininet WiFi provides a graphical display (figure 2.1) showing locations of WiFi nodes in a graph. The graph may be created by calling its method in the Mininet-WiFi Python API (see examples in the tutorials below).

The graph will show wireless access points and stations, their positions in space and will display the affects of the range parameter for each node. The graph will not show any "wired" network elements such as standard Mininet hosts or switches, Ethernet connections between access points, hosts, or switches.

---

[5] https://wireless.wiki.kernel.org/en/developers/documentation/glossary#softmac
[6] http://www.linuxwireless.org/en/developers/Documentation/cfg80211
[7] https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim
[8] http://stackoverflow.com/questions/33091895/virtual-wifi-802-11-interface-similar-to-veth-on-linux
[9] http://linuxwireless.org/en/developers/Documentation/mac80211/__v49.html
[10] https://w1.fi/cgit/hostap/plain/tests/hwsim/example-setup.txt
[11] http://www2.cs.siu.edu/s̃harvey/code/cs441/cs441_lab.pdf
[12] http://upcommons.upc.edu/bitstream/handle/2099.1/19202/memoria.pdf?sequence=4
[13] https://nims11.wordpress.com/2012/04/27/hostapd-the-linux-way-to-create-virtual-wifi-access-point/
[14] https://wiki.debian.org/WiFi/HowToUse#wpa_supplicant
[15] https://www.kernel.org/doc/readme/Documentation-networking-mac80211_hwsim-README
[16] https://github.com/penberg/linux-kvm/tree/master/Documentation/networking/mac80211_hwsim))
[17] http://sandilands.info/sgordon/capturing-wireless-lan-with-ubuntu-tcpdump-kismet
[18] https://wiki.wireshark.org/Wi-Fi
[19] https://wiki.wireshark.org/CaptureSetup/WLAN#Linux
[20] from http://teampal.mc2lab.com/attachments/685/C2012-12.pdf

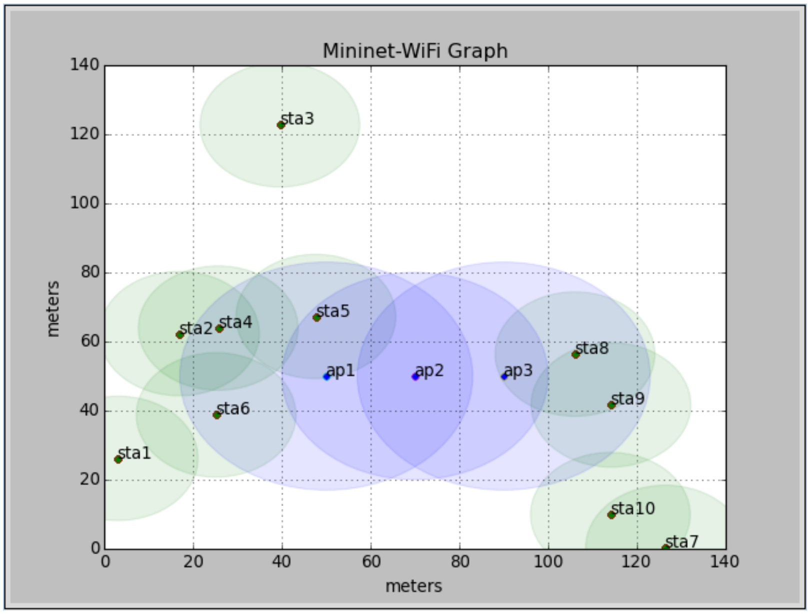

Figure 2.1: Mininet-WiFi Graph

2.2.6 Install Mininet-WiFi on a Virtual Machine

First, we need to create a virtual machine that will run the Mininet-WiFi network emulator. In the example below, we will use the VirtualBox virtual machine manager because it is open-source and runs on Windows, Mac OS, and Linux.

2.2.7 Set up a new Ubuntu Server VM

Install Ubuntu Server in a new VM. Download an Ubuntu Server ISO image from the Ubuntu web site. See my post about `installing Debian Linux in a VM`[21]. Follow the same steps to install Ubuntu.

In this example, we will name the VM `Mininet-WiFi`.

2.2.8 Set up the Mininet-WiFi VM

To ensure that the VM can display X applications such as Wireshark on your host computer's desktop, read through my post about `setting up the standard Mininet VM`[22] and set up the host-only network adapter, the X windows server, and your SSH software.

Now you can connect to the VM via SSH with X Forwarding enabled. In the example below, my host computer is `t420` and the Mininet WiFi VM is named `wifi`.

```
t420:~$ ssh -X 192.168.52.101
wifi:~$
```

[21] http://www.brianlinkletter.com/installing-debian-linux-in-a-virtualbox-virtual-machine/
[22] http://www.brianlinkletter.com/set-up-mininet/

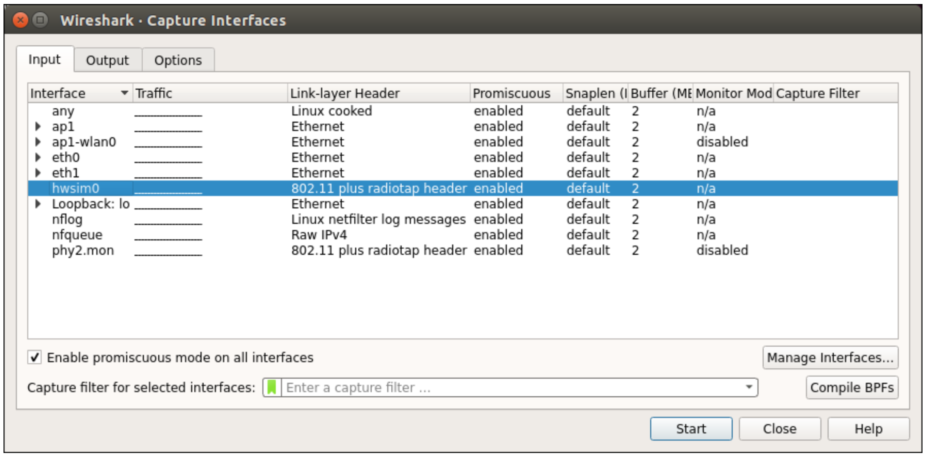

Figure 2.2: Start capture on hwsim0 interface

2.3 Mininet-WiFi Tutorial #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.

2.3.1 Capturing Wireless control traffic in Mininet-WiFi

To view wireless control traffic we must first start Wireshark:

```
wifi:~$ wireshark &
```

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

```
wifi:~$ sudo mn --wifi
```

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.

You should see wireless control traffic. Next, tun 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.

Stop the ping command by pressing `Ctrl-C`. In this default setup, any flows created in the access point (that's if they're created – see below for more on this issue) will expire in 60 seconds.

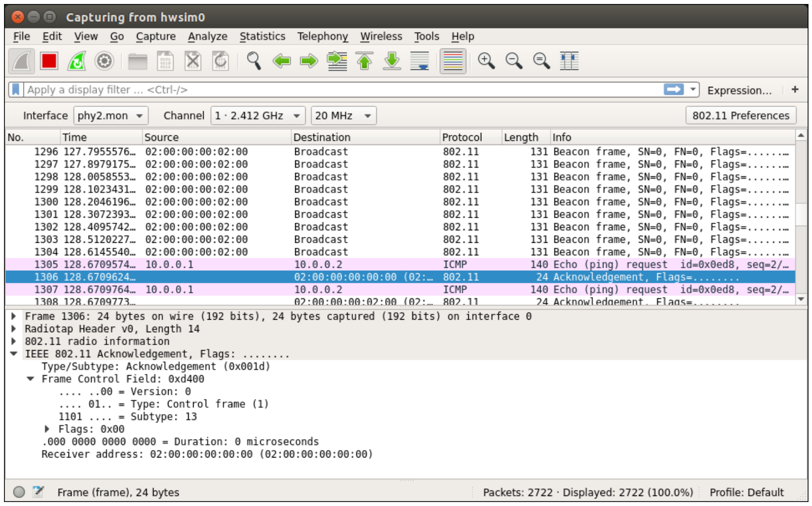

Figure 2.3: Wireshark capturing WiFi control traffic

2.3.2 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_1.0` 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
```

I was expecting that the first ICMP packet generated by the `ping` command should be flooded to the controller, and the controller would set up a flows on the access point so the two stations could exchange packets. Instead, I found that the two stations were able to exchange packets immediately and the access point did not flood the ICMP packets to the controller. Only an ARP packet, which is in a broadcast frame, gets flooded to the controller and is ignored.

Check to see if flows have been created in the access point:

```
mininet-wifi> dpctl dump-flows
*** ap1 -----------------------------------------
NXST_FLOW reply (xid=0x4):
```

We see that no flows have been created on the access point. How do the two access points communicate with each other?

I do not know the answer but I have an idea. My research indicates that OpenFlow-enabled switches (using OpenFlow 1.0 or 1.3) will reject "hairpin connections", which are flows that cause

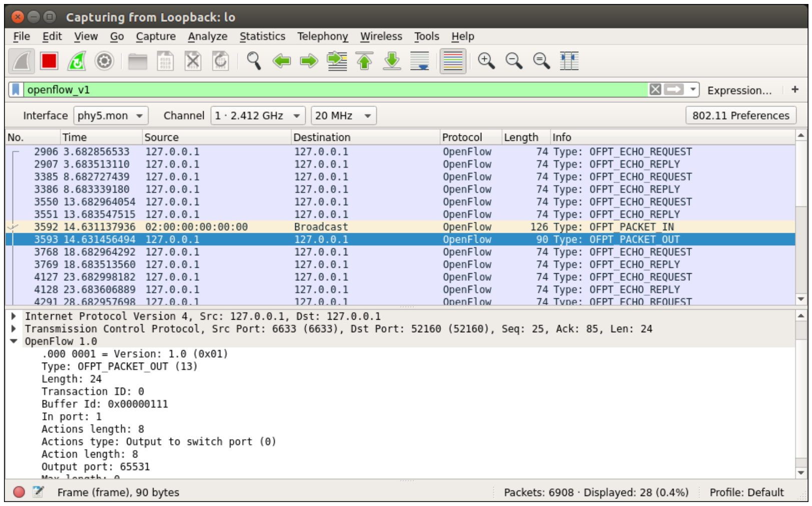

Figure 2.4: No OpenFlow messages passing to the controller

traffic to be sent out the same port in which it was received. A wireless access point, by design, receives and sends packets on the same wireless interface. Stations connected to the same wireless access point would require a "hairpin connection" on the access point to communicate with each other. I surmise that, to handle this issue, Linux treats the WLAN interface in each access point like the radio network sta1-ap1-sta2 as if it is a "hub", where `ap1-wlan0` provides the "hub" functionality for data passing between sta1 and sta2. `ap1-wlan0` switches packets in the wireless domain and will not bring a packet into the "Ethernet switch" part of access point `ap1` unless it must be switched to another interface on `ap1` other than back out `ap1-wlan0`.

2.3.3 Stop the tutorial

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
```

2.4 Mininet-WiFi Tutorial #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 tutorial, we will create a linear topology with three access points, where one station is connected to each access point. Remember, you need to already know `basic Mininet commands`[23] to appreciate how we create topologies using the Mininet command line.

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

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

[23] http://mininet.org/walkthrough/

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`. For example, we can run the `net` command to see the connections between nodes:

```
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 `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>
```

2.4.1 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.

Note: The `iw` commands may be used in static scenarios like this but should not be used when Mininet-WiFi automatically assigns associations in more realistic mobility scenarios. We'll discuss how Mininet-WiFi handles real mobility and how to use `iw` commands with Mininet-WiFi later in this post.

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
```

Verify the change with the `iw link` command:

```
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: 112 bytes (4 packets)
            TX: 103 bytes (2 packets)
            signal: -30 dBm
            tx bitrate: 1.0 MBit/s

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

We see that `sta1` is now associated with `ap2`. So we've demonstrated a basic way to make stations mobile, where they switch their association from one access point to another.

2.4.2 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`.

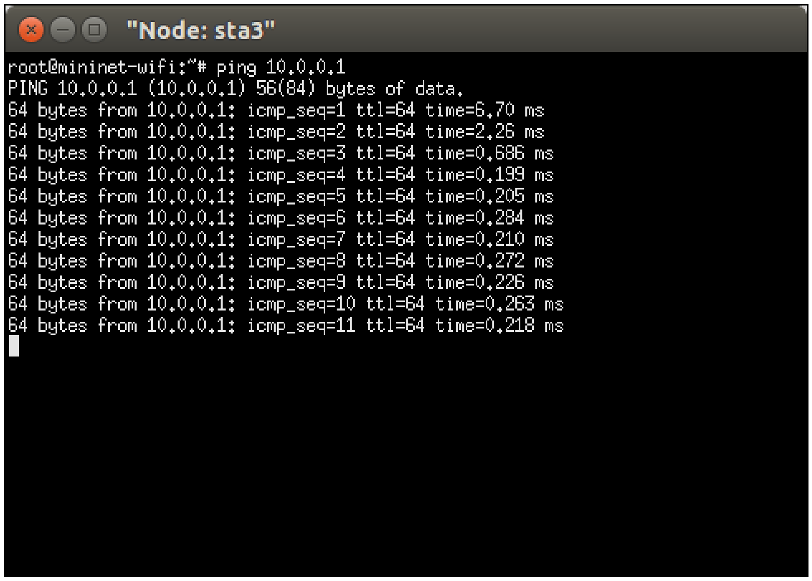

Figure 2.5: xterm window on sta3

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
```

This opens an xterm window from `sta3`.

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

```
root@mininet-wifi:~# ping 10.0.0.1
```

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`.

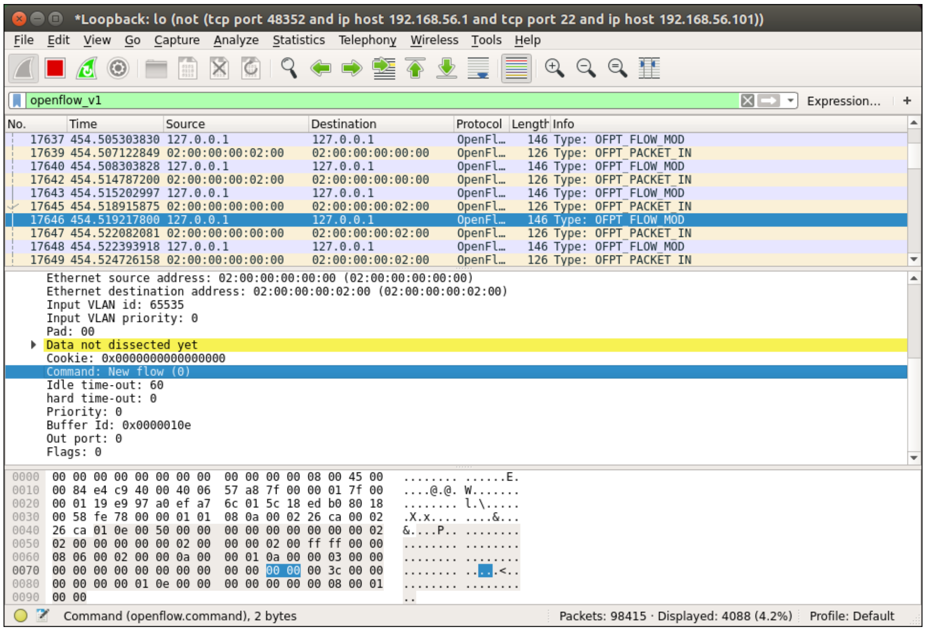

Figure 2.6: Wireshark capturing OpenFlow messages

If we run Wireshark and enable packet capture on the `Loopback` interface, then filter using with `of` (for Ubuntu 14.04) or `openflow_v1` (for Ubuntu 15.10 and later), we will see OpenFlow messages passing to and from the controller. 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=02:00:00:00:02:00,
     ↪ 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:3
     cookie=0x0, duration=1068.17s, table=0, n_packets=35, n_bytes=1470,
      ↪ idle_timeout=60, idle_age=0,
      ↪ priority=65535,arp,in_port=3,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪ dl_dst=02:00:00:00:02:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1
      ↪ actions=output:2
     cookie=0x0, duration=1073.174s, table=0, n_packets=1073,
      ↪ n_bytes=105154, idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=3,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪ dl_dst=02:00:00:00:02:00,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=1073.175s, table=0, n_packets=1073,
      ↪ n_bytes=105154, idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪ 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:3
    *** ap3 -----------------------------------------------
    NXST_FLOW reply (xid=0x4):
     cookie=0x0, duration=1068.176s, table=0, n_packets=35, n_bytes=1470,
      ↪ idle_timeout=60, idle_age=0,
      ↪ priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪ 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
    idle_timeout=60, idle_age=0,
     ↪ priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
     ↪ dl_dst=02:00:00:00:02:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=1
     ↪ actions=output:2
     cookie=0x0, duration=1073.182s, table=0, n_packets=1073,
      ↪ n_bytes=105154, idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪ dl_dst=02:00:00:00:02:00,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=1073.185s, table=0, n_packets=1073,
      ↪ n_bytes=105154, idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪ 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
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? Move sta1 back to access point ap1 with the following commands:

```
mininet-wifi> sta1 iw dev sta1-wlan0 disconnect
mininet-wifi> sta1 iw dev sta1-wlan0 connect ssid_ap1
```

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_age=40,
      ↪  priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪  dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=1
      ↪  actions=output:2
     cookie=0x0, duration=40.958s, table=0, n_packets=1, n_bytes=42,
      ↪  idle_timeout=60, idle_age=40,
      ↪  priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪  dl_dst=02:00:00:00:02:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=2
      ↪  actions=output:1
    *** ap2 ------------------------------------------------
    NXST_FLOW reply (xid=0x4):
     cookie=0x0, duration=40.968s, table=0, n_packets=1, n_bytes=42,
      ↪  idle_timeout=60, idle_age=40,
      ↪  priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪  dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=1
      ↪  actions=output:1
     cookie=0x0, duration=40.964s, table=0, n_packets=1, n_bytes=42,
      ↪  idle_timeout=60, idle_age=40,
      ↪  priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪  dl_dst=02:00:00:00:02:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=2
      ↪  actions=output:2
     cookie=0x0, duration=1214.279s, table=0, n_packets=1214,
      ↪  n_bytes=118972, idle_timeout=60, idle_age=0,
      ↪  priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪  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:3
    *** ap3 ------------------------------------------------
    NXST_FLOW reply (xid=0x4):
     cookie=0x0, duration=40.978s, table=0, n_packets=1, n_bytes=42,
      ↪  idle_timeout=60, idle_age=40,
      ↪  priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪  dl_dst=02:00:00:00:00:00,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=1
      ↪  actions=output:1
     cookie=0x0, duration=40.971s, table=0, n_packets=1, n_bytes=42,
      ↪  idle_timeout=60, idle_age=40,
      ↪  priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪  dl_dst=02:00:00:00:02:00,arp_spa=10.0.0.1,arp_tpa=10.0.0.3,arp_op=2
      ↪  actions=output:2
     cookie=0x0, duration=1214.288s, table=0, n_packets=1214,
      ↪  n_bytes=118972, idle_timeout=60, idle_age=0,
      ↪  priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪  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
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, idle_age=0,
      ↪ priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪ dl_dst=02:00:00:00:02:00,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=9.41s, table=0, n_packets=10, n_bytes=980,
      ↪ idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪ 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
    *** ap2 ----------------------------------------------
    NXST_FLOW reply (xid=0x4):
     cookie=0x0, duration=10.414s, table=0, n_packets=11, n_bytes=1078,
      ↪ idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
      ↪ dl_dst=02:00:00:00:02:00,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=9.417s, table=0, n_packets=10, n_bytes=980,
      ↪ idle_timeout=60, idle_age=0,
      ↪ priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
      ↪ 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
    *** ap3 ----------------------------------------------
    NXST_FLOW reply (xid=0x4):
```

```
    cookie=0x0, duration=10.421s, table=0, n_packets=11, n_bytes=1078,
     ↪ idle_timeout=60, idle_age=0,
     ↪ priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=02:00:00:00:00:00,
     ↪ dl_dst=02:00:00:00:02:00,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=9.427s, table=0, n_packets=10, n_bytes=980,
     ↪ idle_timeout=60, idl_age=0,
     ↪ priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=02:00:00:00:02:00,
     ↪ 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
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 post.

2.4.3 Stop the tutorial

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
```

2.5 Mininet-WiFi Tutorial #3: Python API and scripts

Mininet provides a Python API so users can create simple Python scripts that will set up custom topologies. Mininet-WiFi extends this API to support a wireless environment.

When you use the normal Mininet mn command with the -wifi option to create Mininet-WiFi topologies, you do not have access to most of the extended functionality provided in Mininet-WiFi. To access features that allow you to emulate the behavior of nodes in a wireless LAN, you need to use the Mininet-Wifi extensions to the Mininet Python API.

2.5.1 The Mininet-WiFi Python API

The Mininet-WiFi developers added new classes to Mininet to support emulation of nodes in a wireless environment. Mininet-WiFi adds addStation and addBaseStation methods, and a modified addLink method to define the wireless environment.

If you are just beginning to write scripts for Mininet-WiFi, you can use the example scripts as a starting point. The Mininet-WiFi developers created example scripts that show how to use most of the features in Mininet-WiFi. In all of the tutorials I show below, I started with an example script and modified it.

Mininet-Wifi example scripts are in the ~/mininet-wifi/examples directory.

2.5.2 Basic station and access point methods

In a simple scenario, you may add a station and an access point with the following methods in a Mininet-WiFi Python script:

Add a new station named `sta1`, with all parameters set to default values:

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

Add a new access point named `ap1`, with SSID `ap1-ssid`, and all other parameters set to default values:

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

Add a wireless association between station and access point, with default values for link attributes:

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

For more complex scenarios, more parameters are available for each method. You may specify the MAC address, IP address, location in three dimensional space, radio range, and more. For example, the following code defines an access point and a station, and creates an association (a wireless connection) between the two nodes and applies some traffic control parameters to the connection to make it more like a realistic radio environment, adding badwidth restrictions, an error rate, and a propagation delay:

Add a station and specify the wireless encryption method, the station MAC address, IP address, and position in virtual space:

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

Add an access point and specify the wireless encryption method, SSID, wireless mode, channel, position, and radio range:

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

Add a wireless association between a station and an access point and specifiy link properties of maximum bandwidth, error rate, and delay:

```
net.addLink( ap1, sta1, bw='11Mbps', loss='0.1%', delay='15ms' )
```

To activate association control in a static network, you may use the *associationControl* method, which makes Mininet-WiFi automatically choose which access point a base station will connect to based on the range between stations and access points. For example, use the following method to use the *strongest signal first* when determining connections between station and access points:

```
net.associationControl( 'ssf' )
```

2.5.3 Classic Mininet API

The Mininet WiFi Python API still supports the standard Mininet node types – switches, hosts, and controllers. For example:

Add a host. Note that the station discussed above is a type of host nodem with a wireless interface instead of an Ehternet interface.

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

Add a switch. Note that the access point discussed above is a type of switch that has one wireless interface (*wlan0*) and any number of Ethernet interfaces (up to the maximum supported by your installed version of Open vSwitch).

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

Add an Ethernet link between two nodes. Note that if you use *addLink* to connect two access points together (and are using the default Infrastructure mode), Mininet-WiFi creates an Ethernet link between them.

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

Add a controller:

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

Using the Python API, you may build a topology that includes hosts, switches, stations, access points, and multiple controllers.

2.5.4 Example

In the example below, I created a Python program that will set up two stations connected to two access points, and set node positions and radio range so that we can see how these properties affect the emulated network. I used the Mininet-WiFi example script `2AccessPoints.py` as the base for the script shown below, then I added the position information to each node and enabled association control.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller,OVSKernelSwitch
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel

def topology():

    net = Mininet( controller=Controller, link=TCLink, switch=OVSKernelSwitch )

    print "*** Creating nodes"
    ap1 = net.addBaseStation( 'ap1', ssid= 'ssid-ap1', mode= 'g', channel= '1',
    position='10,30,0', range='20' )
    ap2 = net.addBaseStation( 'ap2', ssid= 'ssid-ap2', mode= 'g', channel= '6',
    position='50,30,0', range='20' )
```

```
    sta1 = net.addStation( 'sta1', mac='00:00:00:00:00:01', ip='10.0.0.1/8',
    position='10,20,0' )
    sta2 = net.addStation( 'sta2', mac='00:00:00:00:00:02', ip='10.0.0.2/8',
    position='50,20,0' )
    c1 = net.addController( 'c1', controller=Controller )

    """plot graph"""
    net.plotGraph(max_x=60, max_y=60)

    # Comment out the following two lines to disable AP
    print "*** Enabling association control (AP)"
    net.associationControl( 'ssf' )

    print "*** Creating links and associations"
    net.addLink( ap1, ap2 )
    net.addLink( ap1, sta1 )
    net.addLink( ap2, sta2 )

    print "*** Starting network"
    net.build()
    c1.start()
    ap1.start( [c1] )
    ap2.start( [c1] )

    print "*** Running CLI"
    CLI( net )

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    topology()
```

codes/position–test.py

I saved the file with the name `position-test.py` and made it executable.

2.5.5 Working at runtime

Mininet-WiFi python scripts may be run from the command line by running the script directly, or by calling it as part of a Python command. The only difference is how the path is stated. For example:

```
wifi:~/scripts $ sudo ./position-test.py
```

or,

```
wifi:~$ sudo python position-test.py
```

The `position-test.py` script will set open the Mininet-WiFi graph window and show the locations of each wireless node in space, and the range attribute of each node.

While the scenario is running, we can query information about the network from either the Mininet-WiFi command line or from the Python interpreter and we can log into running nodes to gather information or make configuration changes.

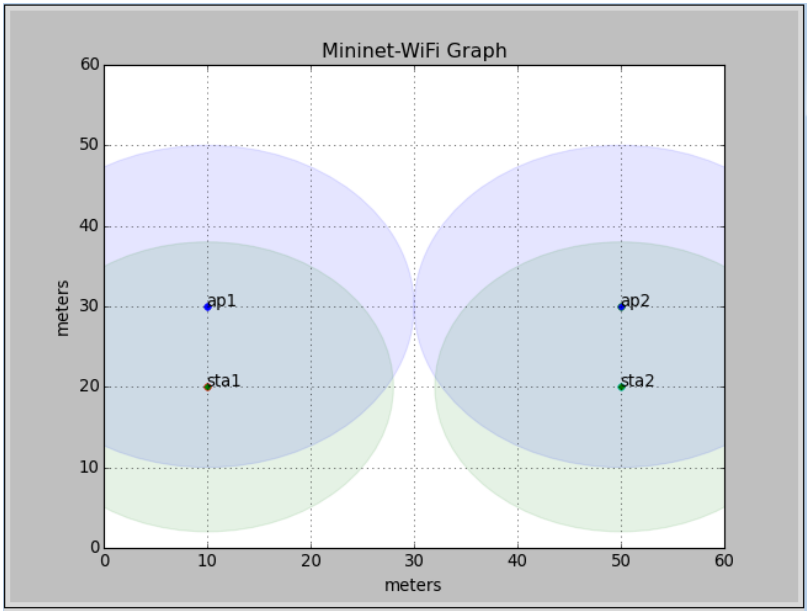

Figure 2.7: The position-test.py script running

2.5.6 Mininet-WiFi CLI

The Python script `position-test.py` places nodes in specific positions. When the scenario is running, we can use the Mininet-WiFi command line interface (CLI) commands to can check the geometric relationship between nodes in space, and information about each node.

2.5.7 Position

The `position` CLI command outputs the location of a node in virtual space as measured by three values, one for each of the vertices X, Y, and Z.

Suppose we want to know the position of the access point `ap1` in the network scenario's virtual space. We may use the `position` CLI command to view a node's position:

```
mininet-wifi> py ap1.params['position']
```

We may also check the position of the station `sta2`:

```
mininet-wifi> py sta.params['position']
```

2.5.8 Distance

The `distance` CLI command tells us the distance between two nodes.

For example, we may check how far apart access point `ap1` and station `sta2` are from each other using the `distance` CLI command:

```
mininet-wifi> distance ap1 sta2
The distance between ap1 and sta2 is 41.23 meters
```

2.5.9 Info

The `info` CLI command prints information about each node running in the scenario.

For example, to see information about access point `ap1`, enter the CLI command:

```
mininet-wifi> info ap1
Tx-Power: [20] dBm
SSID: ssid-ap1
Number of Associated Stations: 1
```

To see information about station `sta1`, enter the CLI command:

```
mininet-wifi> info sta1
--------------------------------
Interface: sta1-wlan0
Associated To: ap1
Frequency: 2.412 GHz
Signal level: -40.10 dbm
Tx-Power: 20 dBm
```

2.5.10 Mininet-WiFi Python runtime interpreter

In addition to the CLI, Mininet-WiFi supports running Python code directly at the command line using the py command. Simple, Python functions may be called to get additional information about the network, or to make simple changes while the scenario is running.

2.5.11 Getting network information

The examples I show below are useful for gathering information about stations and access points.

To see the range of an access point or station, call the `range` function. Call it using the name of the node followed by the function as shown below for access point `ap1`:

```
mininet-wifi> py ap1.range
20
```

To see which station is associated with an access point (in this example `ap1`) call the `associatedStations` function:

```
 mininet-wifi> py ap1.associatedStations
[<Host sta1: sta1-wlan0:10.0.0.1 pid=3845> ]
```

To see which access point is associated with a station (in this example `sta1`) call the `associatedTo` key:

```
mininet-wifi>py sta1.params['associatedTo']
[<OVSSwitch ap1: lo:127.0.0.1,ap1-eth1:None pid=3862>]
```

You may also query the received signal strength indicator (`rssi`), transmitted power (`txpower`), service set indicator (`ssid`), `channel`, and `frequency` of each wireless node using the Python interpreter.

As we can see, the output of Python functions is formatted as strings and numbers that may sometimes be hard to read. This is because these functions are built to support the program, not to be read by humans. However, if you which functions are available to be called at the Mininet-WiFi command line you will be able to get information you cannot get through the standard Mininet-WiFi CLI.

2.5.12 Changing the network during runtime

Mininet-WiFi provides Python functions that can be used during runtime to make changes to node positions and associations. These functions are useful when we have a static setup and want to make arbitrary changes on demand. This makes it possible to do testing or demonstrations with carefully controlled scenarios.

To change the access point to which a station is associated (provided the access point is within range):

```
sta1.moveAssociationTo('sta1-wlan0', 'ap1')
```

To move a station or access point in space to another coordinate position:

```
sta1.moveStationTo('40,20,40')
```

To change the range of a station or access point:

```
sta1.setRange(100)
```

The commands above will all impact which access points and which stations associate with each other. The behavior of the network will be different depending on whether association control is enabled or disabled in the `position-test.py` script.

2.5.13 Running commands in nodes

When running a scenario, users may make configuration changes on nodes to implement some additional functionality. This can be done from the Mininet-WiFi command line by sending commands to the node's command shell. Start the command with the name of the node followed by a space, then enter the command to run on that node.

For example, to see information about the WLAN interface on a station named `sta1`, run the command:

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
```

Another way to run commands on nodes is to open an `xterm` window on that node and enter commands in the xterm window. For example, to open an xterm window on station *sta1*, run the command:

```
mininet-wifi> xterm sta1
```

Running commands on nodes is standard Mininet feature but it is also an advanced topic. See the `Mininet documentation`[24] for more details. You can run simple commands such as `ping` or `iwconfig` but more advance commands may require you to mount `private directories`[25] for configuration or log files.

[24] https://github.com/mininet/mininet/wiki/Documentation
[25] https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#important-shared-filesystem

2.5.14 Mininet-WiFi and shell commands

Mininet-WiFi manages the affect of range using code that calculates the ability of each node to connect with other nodes. However, Mininet-WiFi does not change the way networking works at the operating system level. So `iw` commands executed on nodes will override Mininet-WiFi and do not gather information generated by Mininet-WiFi about the network.

I suggest you do not rely on `iw` commands. For example, the `iw scan` command will still show that `sta1` can detect the SSIDs of all access points, even the access point `ap2` which should be out of range. The `iw link` command will show the same signal strength regardless of how far the station is from the access point, while the Mininet-WiFi *info* command will show the calculated signal strength based on the propagation model and distance between nodes.

For example, the `iw` command run on `sta1` shows received signal strength is -30 dBm. This never changes no matter how far the station is from the access point.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:00:00 (on sta1-wlan0)
        SSID: ssid-ap1
        freq: 2412
        RX: 164628 bytes (2993 packets)
        TX: 775 bytes (10 packets)
        signal: -30 dBm
        tx bitrate: 6.0 MBit/s

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

The `info` command shows Mininet-WiFi's calculated signal strength received by the station is -43.11 dBm. This value will change if you reposition the station.

```
mininet-wifi> info sta1
--------------------------------
Interface: sta1-wlan0
Associated To: ap1
Frequency: 2.412 GHz
Signal level: -43.11 dbm
Tx-Power: 20 dBm
```

When working with Mininet-WiFi during runtime, use the built-in Mininet-WiFi commands or use the Python functions to check the wireless attributes of nodes.

2.5.15 Stop the tutorial

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

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

2.6 Mininet-WiFi Tutorial #4: Mobility

The more interesting features provided by Mininet-WiFi support mobile stations moving around in virtual space. Mininet-Wifi provides new methods in its Python API, such as `startMobility` and `Mobility`, with which we may specify a wide variety of wireless LAN scenarios by controlling station movement, access point range, radio propagation models, and more.

In this tutorial, we will create a scenario where one station moves about in space, and where it changes which access point it connects to, based on which access point is the closest.

2.6.1 Python API and mobility

The Mininet-WiFi Python API adds new methods that allow the user to create stations that move around in virtual space when an emulation scenario is running.

To move a station in a straight line, use the `net.StartMobility` and `net.mobility` methods. See the example script `wifiMobilty.py`. For example, to move a station from one position to another over a period of 60 seconds, add the following lines to your script:

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

Mininet-WiFi can also automatically move stations around based on predefined mobility models. See the example script `wifiMobilityModel.py`. Available mobility models are: `RandomWalk`, `TruncatedLevyWalk`, `RandomDirection`, `RandomWayPoint`, `GaussMarkov`, `ReferencePoint`, and `TimeVariantCommunity`. For example, to move a station around in an area 60 meters by 60 meters with a minimum velocity of 0.1 meters per second and a maximum velocity of 0.2 meters per second, add the following line to your script:

```
net.startMobility(startTime=0, model='RandomDirection', max_x=60, max_y=60,
↪  min_v=0.1, max_v=0.2)
```

Mininet-WiFi will automatically connect and disconnect stations to and from access points based on either calculated signal strength or load level. See the example script `wifiAssociationControl.py`. To use association control, add the `AC` parameter to the `net.startMobility` call. For example, to switch access points based on the "least loaded first" criteria, add the following line to your script:

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

The valid values for the `AC` parameter are:

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

When creating a scenario where stations will be mobile, we may set the range of the access points. In an example where we use "strongest signal first" as the Association Control method, the range of each access point will determine where handoffs occur between access points and which stations may connect to which access points. If you do not define the range, Mininet-WiFi assigns a default value.

Mininet-WiFi supports more methods than mentioned above. See the example scripts (mentioned further below) for examples of using other methods.

2.6.2 Moving a station in virtual space

A simple way to demonstrate how Mininet-WiFi implements scenarios with mobile stations that hand off between access points is to create a script that moves one station across a path that passes by three access points.

The example below will create three access points – `ap1`, `ap2`, and `ap3` – arranged in a line at differing distances from each other. It also creates a host `h1` to serve as a test server and a mobile station `sta1` and moves `sta1` across space past all three access points.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller,OVSKernelSwitch
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel

def topology():

    net = Mininet( controller=Controller, link=TCLink, switch=OVSKernelSwitch )

    print "*** Creating nodes"
    h1 = net.addHost( 'h1', mac='00:00:00:00:00:01', ip='10.0.0.1/8' )
    sta1 = net.addStation( 'sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8', range='20' )
    ap1 = net.addBaseStation( 'ap1', ssid= 'ap1-ssid', mode= 'g', channel= '1', position='30,50,0', range='30' )
    ap2 = net.addBaseStation( 'ap2', ssid= 'ap2-ssid', mode= 'g', channel= '1', position='90,50,0', range='30' )
    ap3 = net.addBaseStation( 'ap3', ssid= 'ap3-ssid', mode= 'g', channel= '1', position='130,50,0', range='30' )
    c1 = net.addController( 'c1', controller=Controller )


    print "*** Associating and Creating links"
    net.addLink(ap1, h1)
    net.addLink(ap1, ap2)
    net.addLink(ap2, ap3)

    print "*** Starting network"
    net.build()
    c1.start()
    ap1.start( [c1] )
    ap2.start( [c1] )
    ap3.start( [c1] )

    net.plotGraph(max_x=160, max_y=160)

    net.startMobility(startTime=0, AC='ssf')
    net.mobility('sta1', 'start', time=20, position='1,50,0')
    net.mobility('sta1', 'stop', time=79, position='159,50,0')
    net.stopMobility(stopTime=80)

    print "*** Running CLI"
    CLI( net )

```

```
    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    topology()
```

codes/line.py

Save the script and call in `line.py`. Make it executable, then run the command:

```
wifi:~$ sudo ./line.py
```

The Mininet-Wifi graph will appear, showing the station and the access points.

The station `sta1` will sit still for 20 seconds, and then start to move across the graph from left to right for 60 seconds until it gets to the far side of the graph. The host `h1` and the virtual Ethernet connections between `h1`, `ap1` and between the three access points are not visible.

2.6.3 Re-starting the scenario

This simple scenario has a discreet start and stop time so, if you wish to run it again, you need to quit Mininet-WiFi, and start the script again.

For example, suppose the scenario is at its end, where the station is now at the far right of the graph window. To stop and start it again, enter the following commands:

```
mininet-wifi> exit
wifi:~$ sudo mn -c
wifi:~$ sudo ./line.py
```

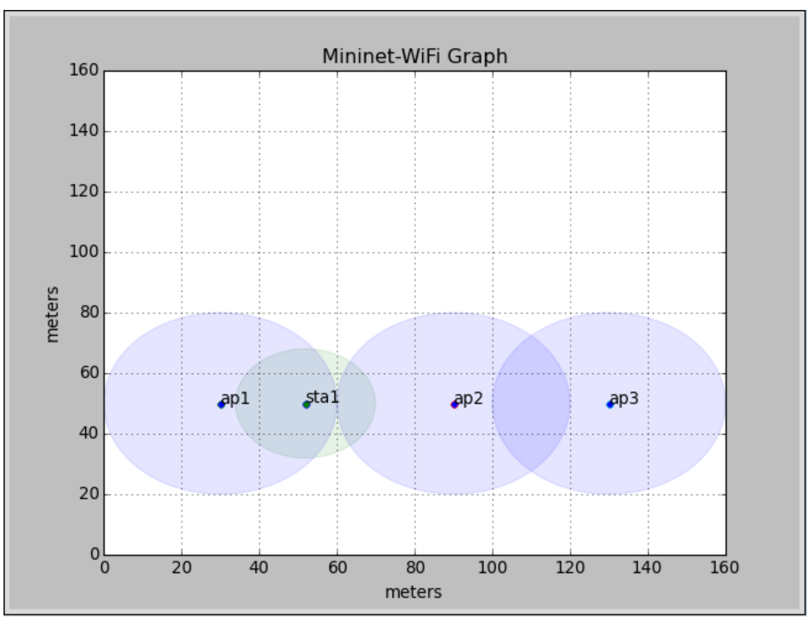

Figure 2.8: The line.py script running

2.6.4 More Python functions

When running a scenario with the mobility methods in the Python API, we have access to more information from Mininet-WiFi's Python functions. To see all access points that are within range of a station such as `sta1` at any time while the scenario is running, call the `apsInRange` function:

```
mininet-wifi> py sta1.params['apsInRange']
[<OVSSwitch ap1: lo:127.0.0.1,ap1-eth1:None pid=3862>]
```

2.6.5 Test with iperf

To see how the system responds to traffic, run some data between host `h1` and station `sta1` when the scenario is started.

We've seen in previous examples how to use the `ping` program to create traffic. In this example, we will use the `iperf` program. First, start the `line.py` script again. Then start an iperf server on the station

```
mininet-wifi> sta1 iperf --server
```

Then open an xterm window on the host `h1`.

```
mininet-wifi> xterm h1
```

From the xterm window, we will start the iperf client command and create a stream of data between `h1` and `sta1`. On the `h1` xterm, run the command:

```
iperf --client 10.0.0.2 --time 60 --interval 2
```

Watch the iperf output as the station moves through the graph. When it passes from one access point to the next, the traffic will stop. To get the traffic running again, clear the flow tables in the access points. In the Mininet-WiFi CLI, run the command shown below:

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

Traffic should start running again. As stated in Tutorial #2 above, we must clear flows after a hand off because the Mininet reference controller cannot respond correctly in a mobility scenario. The topic of configuring a remote controller to support a mobility scenario is outside the scope of this post. Clear the flows every time the station switches to the next access point.

2.6.6 Stop the tutorial

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

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

2.7 Mininet-WiFi example scripts

The Mininet-WiFi developers created many example scripts that show examples of most of the API extensions they added to Mininet. They placed these example scripts in the folder `mininet-wifi/examples/`. Try running these scripts to see what they do and look at the code to understand how each feature is implemented using the Python API. Some interesting Mininet-WiFi example scripts are:

`adhoc` shows how to set up experiments with adhoc mode, where stations connect to each other without passing through an access point. `simplewifitopology` show the Python code that create the same topology as the default topology created my the `mn -wifi` command (two stations and one access point). `wifiStationsAndHosts` creates a topology with stations and hosts `2AccessPoints` to create a topology with two access points connected to each other via an Ethernet link and two stations associated with each access point. `wifiPosition.py` shows how to create a network where stations and access points are places in specific locations in virtual space. `wifiMobility` and `wifiMobilityModel` show how to move stations and how mobility models can be incorporated into scripts. `wifiAssociationControl` shows how the different values of the AC parameter affect station handoffs to access points. `wifimesh.py` shows how to set up a mesh network of stations. `handover.py` shows how to create a simple mobility scenario where a station moves past two access points, causing the station to hand off from one to the other. `multipleWlan.py` shows how to create a station with more than one wireless LAN interface. `wifiPropagationModel.py` shows how to use propagation models that impact how stations and access points can communicate with each other over distance. `wifiAuthentication.py` shows how to set up WiFi encryption and passwords on access points and stations.

2.8 Conclusion

The tutorials presented above work demonstrate many of the unique functions offered by Mininet-Wifi. Each tutorial revealed more functionality and we stopped at the point where we were able to emulate mobility scenario featuring a WiFi station moving in a straight line past several wireless access points.

To learn more about Mininet-WiFi, go to the `Mininet-WiFi wiki`[26] page. Also, read through posts on the `Mininet-WiFi mailing list`[27], which is very active and is a useful source of more information about Mininet-WiFi. I am looking for an OpenFlow controller that will support WiFi switches using OpenFlow 1.3, which is the version of OpenFlow supported by Mininet and Mininet-WiFi. If you know of any, please add a comment to this post.

R Thank you Brian Linkletter for this helpful tutorial.

[26] https://github.com/intrig-unicamp/mininet-wifi/wiki
[27] https://groups.google.com/forum/#!forum/mininet-wifi-discuss

Mininet-WiFi
Emulator for Software-Defined Wireless Networks

3. Reproducing Paper

3.1 SwitchOn 2015

Extended abstract - Towards an Emulator for Software Defined Wireless Networks

Get the code in https://github.com/intrig-unicamp/mininet-wifi/blob/master/demos/allWirelessNetworksAroundUs.py, then execute it:

```
sudo python allWirelessNetworksAroundUs.py
```

```
mininet-wifi> xterm sta1 h1
```

On sta1:

```
cvlc -vvv v4l2:///dev/video0 --input-slave=alsa://hw:1,0 --mtu 1000
 --sout'#transcode{vcodec=mp4v,vb=800,scale=1,acodec=mpga,ab=128,channels=1}:
 duplicate{dst=display,dst=rtp{sdp=rtsp://10.0.0.10:8080/helmet.sdp}'
```

On h1:

```
cvlc rtsp://10.0.0.10:8080/helmet.sdp
```

3.2 SBRC 2016

DEMO - Mininet-WiFi: Emulação de Redes Sem Fio Definidas por Software com suporte a Mobilidade

3.2.1 Case 1 - Simple test

```
sudo mn --wifi
mininet-wifi>sta1 ping sta2
mininet-wifi>sta1 iwconfig
mininet-wifi>sta2 iwconfig
```

3.2.2 Case 2 (1/2) - Communication among stations and hosts/Verifying flow table

```
sudo python examples/wifiStationsAndHosts.py
mininet-wifi>nodes
mininet-wifi>sh ovs-ofctl dump-flows ap1
mininet-wifi>sta1 ping h3
mininet-wifi>sh ovs-ofctl dump-flows ap1
```

3.2.3 Case 2 (2/2) - Changing the controller (from reference to external controller)

Open the code examples/wifiStationsAndHosts.py and make the following changes:

```
from: net = Mininet( controller=Controller, link=TCLink,
 ↪  switch=OVSKernelSwitch )
to: net = Mininet( controller=RemoteController, link=TCLink,
 ↪  switch=OVSKernelSwitch )
from: c0 = net.addController('c0', controller=Controller, ip='127.0.0.1' )
to: c0 = net.addController('c0', controller=RemoteController, ip='127.0.0.1'
 ↪  )
sudo python examples/wifiStationsAndHosts.py
mininet-wifi>sta1 ping h3  #Why there is no communication?
```

3.2.4 Case 3 - Handover

```
sudo python examples/handover.py
 mininet-wifi>sta1 iwconfig
 mininet-wifi>sta1 ping sta2  #here I suggest you wait sta1 reaches ap2
  ↪  before going to the next step
 mininet-wifi>sta1 iwconfig
```

3.2.5 Case 4 - Changes at runtime

```
sudo python examples/wifiPosition.py
mininet-wifi>sta1 iwconfig
mininet-wifi>sta1 ping sta2
mininet-wifi>py sta1.moveStationTo('70,40,0')
mininet-wifi>sta1 iwconfig
mininet-wifi>sta1 ping sta2
mininet-wifi>py ap1.setRange(60)
mininet-wifi>sta1 iwconfig
mininet-wifi>sta1 ping sta2
```

Case 5 - Bridging physical and virtual emulated environments

```
sudo systemctl stop network-manager
Open demos/sbrc.py and change:
from: phyap1 = net.addPhysicalBaseStation( 'phyap1', ssid=
 ↪  'SBRC16-MininetWiFi', mode= 'g', channel= '1', position='50,115,0',
 ↪  wlan='wlan11' )
```

```
to: wlan11 to your usb wlan interface.
sudo python demos/sbrc.py
```

At this moment users attending the conference will be invited to connect their mobile devices into the physical/emulated environment.

3.3 SIGCOMM 2016

Demo: Mininet-WiFi: A Platform for Hybrid Physical-Virtual Software-Defined Wireless Networking Research

Requirements to reproduce:

- USB WiFi dongle
- Floodlight OpenFlow controller
- ofsoftswitch13 - https://github.com/CPqD/ofsoftswitch13
- Speedtest-cli

Important (changes in the code - You have to set the wlan interface created by your USB WiFi dongle):

```
e.g. phyap1 = net.addPhysicalBaseStation( 'phyap1', ssid= 'ap-ssid1', mode=
 ↪ 'g', channel= '6', position='170,185,0', wlan='wlan1' )
You have to set the wlan that is connected to the Internet:
e.g. net.addOfDataPath('ap3', 'wlan0')
```

Getting the code in https://github.com/intrig-unicamp/mininet-wifi/blob/master/demos/hybridVirtualPhysical.py

Next, executing the Floodlight OpenFlow controller:

```
sudo java -jar target/floodlight.jar
```

Then,

```
sudo py hybridVirtualPhysical.py
```

Now, stations should be able to communicate with the Internet. You may use any station connected to any Access Point and try it out:

```
mininet-wifi>xterm sta11
mininet-wifi>speedtest-cli
```

Using speedtest-cli you test Download and Upload speed of your internet connection. The available bandwidth is controlled by OpenFlow meter entries. *You can use any mobile device to communicate with stations in Mininet-WiFi and also the Internet if the device is associated to the Access Point.*

4. Publications

4.1 SDN For Wireless 2015

Exhibit
Fontes, R. R., Afzal, S., Brito, S. H. B., Santos, M., Rothenberg, C. E. “Towards an Emulator for Software Defined Wireless Networks“. In EAI International Conference on Software Defined Wireless Networks and Cognitive Technologies for IoT. Rome, Italy, Oct 2015.

4.2 CNSM 2015

This paper explains the basic Mininet-WiFi design and useful Case Studies.
Fontes, R. R., Afzal, S., Brito, S. H. B., Santos, M., Rothenberg, C. E. “Mininet-WiFi: Emulating Software-Defined Wireless Networks“. In 2nd International Workshop on Management of SDN and NFV Systems 2015. Barcelona, Spain, Nov 2015.

4.3 SwitchOn 2015

This paper presents a demo use case in a mobile video streaming scenario to showcase the ability of Mininet-WiFi to emulate the wireless channel in terms of bandwidth, packet loss, and delay variations as a function of the distance between the communicating parties.
Fontes, R. R., Rothenberg, C. E. Towards an Emulator for Software-Defined Wireless Networks. In: SwitchOn 2015, São Paulo – SP – Brazil.

4.4 SBRC 2016

DEMO
Ramon dos Reis Fontes and Christian Esteve Rothenberg. Mininet-WiFi: Emulação de Redes Sem Fio Definidas por Software com suporte a Mobilidade. In Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2016) - Salão de Ferramentas, 2016, Salvador - BA - Brazil.

4.5 SIGCOMM 2016

DEMO

Ramon dos Reis Fontes and Christian Esteve Rothenberg. Mininet-WiFi: A Platform for Hybrid Physical-Virtual Software-Defined Wireless Networking Research (SIGCOMM 2016) - 2016, Florianopolis - ES - Brazil.

Mininet-WiFi

Emulator for Software-Defined Wireless Networks

5. Citations & Users of Mininet WiFi

5.1 Research papers using Mininet-WiFi

- *Felipe S. Dantas Silva, Augusto Neto, Douglas Maciel, José Castillo-Lema, Flávio Silva, Pedro Frosi and Eduardo Cerqueira*. **An Innovative Software-Defined WiNeMO Architecture for Advanced QoS-Guaranteed Mobile Service Transport**. Journal - Computer Networks, 2016. ISSN: 1389-1286, DOI: http://dx.doi.org/10.1016/j.comnet.2016.04.019.
- *Luciano Jerez Chaves, Islene Calciolari Garcia and Edmundo Roberto Mauro Madeira*. **OFSwitch13: Enhancing ns-3 with OpenFlow 1.3 Support.** Proceeding WNS3 '16 Proceedings of the Workshop on ns-3 Pages 33-40 ACM New York, NY, USA ©2016 table of contents ISBN: 978-1-4503-4216-2 doi>10.1145/2915371.2915381.

5.2 Users

Who uses our tool? Please, let us know

- Dr. Chih-Heng Ke Department of Computer Science and Information Engineering, National Quemoy University, Kinmen, Taiwan. http://csie.nqu.edu.tw/smallko/sdn/sdn.htm
- Brian Linkletter: http://www.brianlinkletter.com/mininet-wifi-software-defined-network-emulator-supports-wifi-networks/
- Code Project: http://www.codeproject.com/Tips/1064353/Using-Mininet-wifi-to-Simulate-Software-Defined-Wi

5.3 Citing Mininet-WiFi

Please use:
R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," Network and Service Management (CNSM), 2015 11th International Conference on, Barcelona, 2015, pp. 384-389. doi: 10.1109/CNSM.2015.7367387 URL:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7367387&isnumber=7367318 .bib(http://www.bibsonomy.org/bib/bibtex/2b4f6ac6538a6228f2eb78e24db7cdc2c/chesteve)

@INPROCEEDINGS7367387, author=R. R. Fontes and S. Afzal and S. H. B. Brito and M. A. S. Santos and C. E. Rothenberg, booktitle=Network and Service Management (CNSM), 2015 11th International Conference on, title=Mininet-WiFi: Emulating software-defined wireless networks, year=2015, pages=384-389, keywords=software defined networking;virtualisation;wireless LAN;IEEE 802.11;Mininet-WiFi;network virtualization;software defined network;software-defined wireless network emulation;wireless OpenFlow-SDN scenarios;Emulation;IEEE 802.11 Standard; Linux;Protocols;Topology;Wireless networks;Emulation;OpenFlow;SDN;Wireless networks, doi=10.1109/CNSM.2015.7367387, month=Nov,

6. Acknowledgment