**Mininet-WiFi exercises (Part 1)**

February 15, 2024

**General tips:**

* Resetting Mininet-WiFi (good practice if emulator aborts):
  + Linux CLI: **sudo mn –c**
* Disable Linux’s built-in network-manager (should be done every time the VM is started for the Mininet-WiFi exercise):
  + Linux CLI: **sudo systemctl stop NetworkManager.service**
* Saving Mininet-WiFi output (but unfortunately not your input) in a file (here **outputfile.txt**):
  + Linux CLI: **sudo python *myscript*.py 2>&1 | tee outputfile.txt**
* You can use the up and down arrow keys on your keyboard to quickly scroll up and down in your command history. This works for both the Linux CLI and the Mininet-WiFi CLI.

**Step 1:**

Download the file **skeleton.py** from DTU Learn and make[[1]](#footnote-1) a copy named **network1.py**

Modify **network1.py** so that you have a small WiFi network with an access point (called ap1 using 802.11g) and two wireless stations (called sta1 and sta2). Ensure that all devices have unique MAC addresses and that the stations have unique IP addresses in the same IP subnet. Position the two stations so they are no more than 5 meters from the access point.



Figure 1 – Network for step 1.

Run the emulator with: **sudo ./network1.py** and document that both stations are associated with the access point. To verify that, e.g., sta1 is associated with the access point, you can use the command: **sta1 iwconfig** and similar for sta2. Document that the two station can communicate by ping’ing sta2 from sta1.

**Step 2:**

Test the throughput between the two stations using the iperf program like last week. Reposition the two stations so they are near the edge of the access point’s coverage area. Repeat the throughput measurements. Compare and discuss your results (remember that the maximum physical bit rate for an 802.11g network is 54 Mbit/s).

**Step 3:**

Create a copy of **network1.py** named **network2.py**. Modify **network2.py** to add a wired host (called h1[[2]](#footnote-2)) to the network and connect it to the access point with a (wired) link. Document that the wireless stations and the wired host can communicate using the Mininet-WiFi command: **pingall**



Figure 2 – Network for step 3.

Measure the throughput between one of the WiFi stations, e.g., sta1, and h1 – are the values the same as between the two WiFi stations? (Why / why not)? You must test both “directions”, i.e., both when the iperf server (**iperf –s**) is running on h1 and the iperf client (**iperf –c *server***) is running on sta1 and the opposite configuration, and also (in both “directions” compare the throughput values reported by both the client and the server). Create a table like the one shown below and comment on the results.

|  |  |  |
| --- | --- | --- |
|  | **Throughput reported by server** | **Throughput reported by client** |
| **Iperf server on h1** |  |  |
| **Iperf server on sta1** |  |  |

**Step 4:**

Modify network2.py again to add an additional wired host. You will need to add a switch[[3]](#footnote-3) to the network that connects to the access point and to the two wired hosts.



Figure 3 – Network for step 4.

Measure the throughput between the two wired hosts and comment on the result.

**Step 5:**

Start **network1.py** again. Enable the hwsim0 interface with **ap1 ifconfig hwsim0 up**

Start a Wireshark capture on the hwsim0 interface and stop it again when you have captured 5-10 beacon frames. How often are beacon frames broadcasted? What bit rates are supported[[4]](#footnote-4) by the access point? What bit rate[[5]](#footnote-5) is the beacon frame transmitted with and why?

Start a new capture with Wireshark on the hwsim0 interface. Execute the following two commands in the Mininet-WiFi CLI:

* **link ap1 sta1 down**
* **link ap1 sta1 up**

Stop wireshark and examine the packets captured and look for packets related to the two “link ap1 sta1 ...” commands. Explain what happens when the commands are executed.

Note: You can use a display filter in Wireshark to hide packets you’re not interested in. E.g., the display filter **wlan.fc.type == 0** makes sure that you only see control packets. If you want to see control packets *except* for beacons, you can use:  
**(wlan.fc.type == 0) and (wlan.fc.subtype != 8)**

Some important methods of the Mininet\_wifi class:

The following notation is used:

* methodName(arg1, arg2, ..., argn, ...)
  + arg1, arg2, ..., argn are *normal* (positional) arguments, i.e., values for all has to be provided
  + The remaining arguments (indicated by '...') are *named* arguments so they have to be provided in the form of 'name1 = <value\_of\_name1>' (i.e. 'argumentname = argumentvalue') and they must come after the normal arguments.
* Example: if a method is given as methodName(name, address, ...) it can be called as
  + methodName('John Doe', 'Main Street 1', zipcode = '1234', country = 'US')  
    where two named arguments (zipcode and country) are provided in addition to the two normal arguments.

WiFi nodes

* addAccessPoint(name, ...)
  + name: internal name of the access point, used to distinguish between different APs
  + <ssid> : (Mandatory) SSID of the WiFi network, e.g., ssid = 'my-ssid'
  + <mode> : (Mandatory) WiFi mode, single character such as 'a', 'b', 'g', 'n', ...
  + <channel> : (Mandatory) WiFi channel as a string, e.g., channel = '1'
  + <mac> : (Optional) MAC address of the AP in the form 'xx:xx:xx:xx:xx:xx' where the x’es should be replaced by valid hexadecimal digits. A random (unique) MAC address is assigned if this argument is not provided
  + <position> : (Mandatory) Position in 3D-space as a string 'xpos,ypos,zpos', e.g., '70,30,0' means an x-position of 70 m, y-position of 30 m, and z-position (height) of 0 m.
  + <freq> : Which frequency band to use, either '2' (for 2.4 GHz) or '5' (for 5 GHz) – only needed if the <mode> argument specifies a technology that can use both bands.
* addStation(name, ...)
  + name
  + <mac>: (Optional), as for addAccessPoint()
  + <ip>: (Mandatory). IP address of the station in CIDR-notation[[6]](#footnote-6)
  + <position>: (Mandatory), as for addAccessPoint()

Wired node:

* addHost(name, ...)
  + name
  + <ip>: (Mandatory)
  + <mac>: (Optional)
  + <position>: (Optional)
* addSwitch(name, ...)
  + name
  + <position>: (Optional)

1. In the Linux CLI, you can use the command: **cp skeleton.py network1.py** [↑](#footnote-ref-1)
2. Make sure the host h1 has a unique IP addresses in the same IP-subnet as the wireless stations. [↑](#footnote-ref-2)
3. Remember to start the switch just like you would start an access point. [↑](#footnote-ref-3)
4. Note: You’ll have to combine information from two parts of the beacon frame to answer this. [↑](#footnote-ref-4)
5. Hint: Look in the “802.11 radio information” section of the wireshark capture. [↑](#footnote-ref-5)
6. Remember that CIDR-notation includes the size of the network-part of the address, e.g., “192.168.0.24/16” [↑](#footnote-ref-6)