Mininet-WiFi

Emulator for Software-Defined Wireless Networks

https://github.com/intrig-unicamp/mininet-wifi

Software-Defined Wireless Networking (SDWN): From Theory to Practice with Mininet-WiFi

International Summer School on Latency Control for Internet of Services

Karlstad, Sweden

2017

INFORMATION & NETWORKING TECHNOLOGIES RESEARCH & INNOVATION GROUP (INTRIG)
DEPARTMENT OF COMPUTER ENGINEERING AND INDUSTRIAL AUTOMATION (DCA)
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.

Mininet-WiFi

Emulator for Software-Defined Wireless Networks

https://github.com/intrig-unicamp/mininet-wifi

Contents

1. Background: SDWN & Mininet-WiFi

In this chapter, you will learn the fundamentals of Software-Defined Wireless Networking along the Mininet-WiFi emulation platform.

1.1 Software-Defined Wireless Networking

Software-Defined Wireless Networking (SDWN) [14, 6] is based on providing programmatic centralized control of the network outside wireless-enabled devices (Access Points - APs) which enforce the data plane instructions (i.e.. policy decisions) received from the controllers. The principles of SDWN are similar to those of Software-Defined Networking (SDN) [16], i.e., a networking approach based on a programmatic separation of the control plane (aka. Network OS) from the data plane (aka. forwarding- or data-plane). The software-defined approach allows administrators to specify the behavior of the network in a logically centralized manner and at a higher-level through APIs provided by the controller platform that implements southbound interfaces to the forwarding devices –the OpenFlow protocol [20] being the most popular southbound interface (as illustrated in Figure 1.1) but not the only one, CAPWAP [32], FORCES [8], or NETCONF [9] are also candidate protocols in scope.

SDWN has become an emerging and significant research branch of SDN, mainly driven by the increased interest of mobile network operators [2, 25] and the synergies with Network Function Virtualisation (NFV) [13]. The separation between control and data planes has existed in the wireless domain prior to SDN and OpenFlow. Indeed, IETF standardized both LWAPP (*Lightweight Access Point Protocol*) RFC5412 [3] and CAPWAP (*Control And Provisioning of Wireless Access Points*) RFC4564 [32]. several years ago. Many enterprise WLAN management systems use protocols such as LWAPP and CAPWAP to manage their wireless network systems. LWAPP defines the control messaging for setup and path authentication and run-time operations whereas CAPWAP is based on LWAPP and enables a controller to manage a collection of wireless access points. Within the Open Networking Foundation (ONF), the Wireless & Mobile Working Group (WMWG) is defining a common ground architectural framework along the necessary OpenFlow protocol extensions or enhancements to realize the identified use cases while leveraging related work in other Standards

Developing Organizations (SDOs) such as 3GPP, IEEE, NGMN, ITU, ETSI, IETF, etc. As per [11], over 15 use cases have been identified, ranging from flexible and scalable packet core to unified access networks, encompassing different elements of OpenFlow-based or OpenFlow-oriented wireless and mobile network domains.

SDWN research in academia has bloomed over the last years (refer to [14] for a comprehensive survey), including proposals such as OpenRoads [33], Odin [27], OpenRF [17], Ethanol [22], among others. Architectures such as CloudMac [7] and Chandelle [21] use CAPWAP in their proposals. CloudMac describes current WLAN management protocols such as CAPWAP, as a protocol hard to extend with new functionalities since CAPWAP AP controllers are mostly proprietary systems. Chandelle, instead, proposes a smooth and fast Wi-Fi roaming with SDN/OpenFlow but suffers from integration challenges with traditional switches and CAPWAP. One issue with CAPWAP is that it tries to solve both control and provisioning/management at once, opening the door for conflicts due to the split of roles, e.g., consider the management layer hazards of an AP receiving a CAPWAP firmware update command.

Identified benefits of integrating WLAN and OpenFlow [11] are commonly related to centralized management and monitoring, unified policies, increased programmability and fine-grained control of WLAN functions. Taking into account these benefits and the limitations associated to CAPWAP –arguably the most advanced (closed) solution today for centralizing wireless networks management

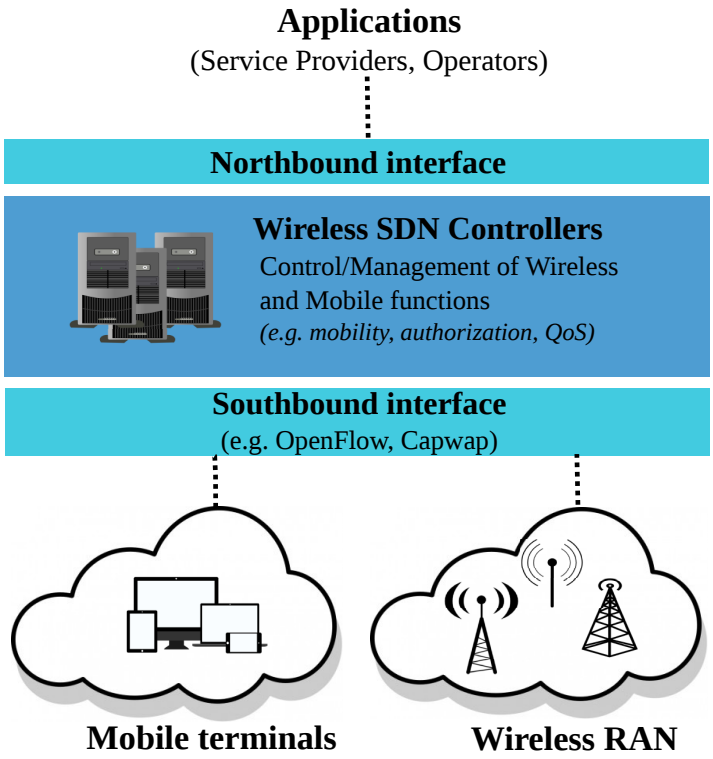

Figure 1.1: Generalized and Simplified Software-Defined Wireless Networking Architecture.

prior to SDN– some questions are inevitable: *"Is CAPWAP in scope of SDWN?"*, *"How to improve the OpenFlow specification to support centralized management of wireless networks?"*, *"Are radical new designs required?"* or *"How much can be leveraged from currently deployed infrastructures?"*. Although these questions are still majorly open, some noteworthy initials steps are undergoing. There is IETF work [4] on extending the CAPWAP protocol to support separate termination points for management, control and data plane tunnels, and the definition of the role of an AP and its controller(s) in RFC7494 [26]. In spirit of the longer-term mission and deliverables of the ONF WMWG, the OpenFlow protocol specification version 1.5 (section B.6.3 [12]) includes a revised behaviour when sending packets out to incoming ports, which was a longstanding issue when mapping wireless interfaces to switch ports.

In addition to CAPWAP-based products, there are multiple proprietary solutions (e.g., Aerohive, Aruba, Cisco HDX, Meraki, Ruckus) based on external controllers to manage a collection of APs. These commercial solutions introduce a number of extensions to standardized protocols or define their own APIs between the controller and APs, presenting differences in the refactoring of control and data plane functions in addition to a series of proprietary radio resource enhancements. While arguably all these solutions have proven to work well at scale, their cost is often prohibitive for many deployments and raise concerns due to their closely integrated nature, the consequent vendor lock-in, and the inability for in-house or third-party innovations.

SDWN Experimental Platforms

As today, the most realistic way to experiment with WiFi and OpenFlow together is using open source firmware and OS solutions like OpenWRT that allows turning commodity wireless routers into OpenFlow-enabled switches. However, like any real testbed, such approach is subject to challenges related to the costs and scale of the experiments, in addition to reproducibility constraints as well as high setup times. Wireless SDN simulators and emulators, on the other hand, are interesting alternatives allowing to work with multiple devices (e.g. APs and STAs) at reasonable scale in experimenter-defined environments. As is well-known but commonly underestimated or misjudged when choosing an experimental platform to support research efforts, each environment excels in some aspects but is subject to certain limitations and/or constraints, as depicted in Figure 1.2.

While the exact quantification of each characteristic and the degree of realism ultimately depend on the accuracy of the model implemented in each specific tool among other platform aspects that may affect each feature, Table 1.1 (adapted from [34]) aims at illustrating the main strengths and shortcomings typically common to each type of experimentation approach as a first guide to choose the best type tool for a given set of research goals and constraints.

Turning now the attention to specific wireless simulators and emulators, Table 1.2 compares a number of features including *Type* (e.g., Simulator/Emulator, Open/Close source), *Programming Language* (which language the solution is written) and finally *Supported Protocols* (relevant protocols available by default), *Last Activity* (i.e., recent updates). Similarly, Table 1.3 attempts to categorize and

Table 1.1: Ranking of Simulators, Emulators and Testbeds (adapted from [34])

Characteristic	Simulators	Emulators	Testbeds
Total Cost	●○○	●○○	●●●
Overall Fidelity	●○○	●●○	●●●
Replay Real Traces	●●○	●●○	●●●
Real Applications	●○○	●●●	●●●
Traffic Realism	●○○	●●●	●●●
Timing Realism	●●●	●●○	●●●
Scalability	●●●	●●○	●○○
Maintainability	●●●	●●●	●○○
Flexibility	●●●	●●●	●○○
Replication	●●●	●●●	●○○
Isolation	●●●	●●○	●●●

Table 1.2: Comparison between Mininet-WiFi and Wireless Simulators & Emulators

Software	Type	Source Type	Programming Language	Supported Protocols	Last Activity
Mininet-WiFi [10]	Emulator	Open	Python	Any (L3 - L7) IEEE 802.11, 802.3, OpenFlow	2016
DCE/ns-3 [19]	Emulator	Open	C++, Python	IEEE 802.11, LTE, OpenFlow	2016
Core[1]	Emulator	Open	several different languages	IEEE 802.2, 802.11	2015
OpenNet [5]	Simulator/ Emulator	Open	C++, Python	IEEE 802.11, LTE, OpenFlow	2016
OMNeT++ [29]	Simulator	Open	C++	IEEE 802.11, OpenFlow	2016
Estinet [30]	Simulator	Proprietary	?	IEEE 802.3, 802.11, OpenFlow	2015
ns-2 [15]	Simulator	Open	C++, TLC	IEEE 802.11, LTE	2012

Table 1.3: Comparative table between Mininet-WiFi and testbeds

Software	Availability	Mobility Support	Supported Protocols	Last activity
Mininet-WiFi [10]	Public	Yes	Any (L3 - L7) IEEE 802.11, 802.3, OpenFlow	2016
Nitos[23]	Public	No	IEEE 802.11, WiMAX, LTE, OpenFlow	2016
R2lab[28]	Public	No	IEEE 802.11, LTE	2016
EMULAB [31]	Public	No	IEEE 802.11	2016
Orbit[24]	Public	No	IEEE 802.11, WiMAX, LTE Bluetooth, ZigBee	2016

compare relevant wireless testbeds considering different criteria.

As we can see, there are a couple of alternatives for OpenFlow-based SDWN experimentation such as DCE/ns-3, OpenNet, OMNeT++, and the Estinet and Nitos testbeds. Under the emulation/simulation space, OpenNet [5] highlights as recent work (arguably closest to Mininet-WiFi) on combining DCE/ns-3 and the Mininet SDN emulator [18] to provide rich SDWN experimentation features by allowing the execution of external controllers and real applications at the cost of a strongly coupled solution. In addition, OpenNet does not provide high-level abstraction APIs for wireless links nor emulation of wireless nodes (e.g., access points and stations are not equipped with wireless network interfaces), and neither mechanisms to select new access points before disconnection of current link to further shorten handover latency.

As today, there is few information regarding Estinet, which according to the developers can be used for many different scenarios, including SDN. Being a proprietary solution and due to its testbed nature, the availability to the wider research community is limited. Nitos, in turn, supports four OpenFlow switches and allows users the possibility to conduct experiments in indoor and outdoor environments.

Table 1.4 compares Mininet-WiFi and DCE/ns-3.[1] In general, in contrast to Mininet-WiFi, DCE/ns-3 does not incorporate real-world network stacks yet and might not support execution of unmodified applications and/or without kernel modification. Another important weakness of DCE/ns-3 is about the development and/or improvements for IEEE 802.11. DCE/ns-3 depends on the development of new models while Mininet-WiFi relies on the mac80211 wireless stack of the Linux kernel. On the other hand, DCE/ns-3 supports greater variety of mobility and propagation models and also Long-Term Evolution (LTE) and because of this, DCE/ns-3 has been very useful during the development of

[1] Information about DCE/ns-3 relesase 1.8 were obtained from the online manual: https://www.nsnam.org/docs/dce/manual/ns-3-dce-manual.pdf

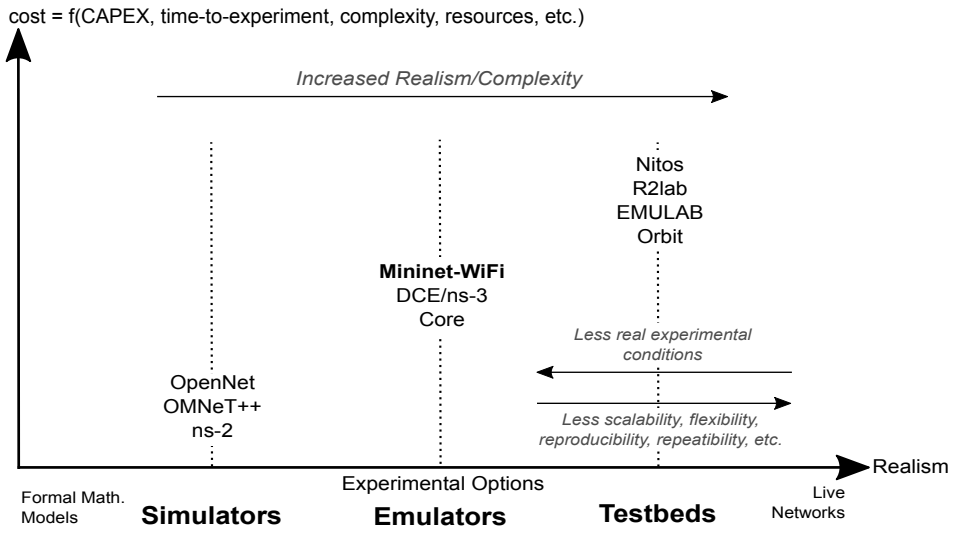

Figure 1.2: Overview of related work and trade-offs of different wireless experimental platforms.

	Mininet-WiFi (v1.9)	**DCE/ns-3** (v1.8)
sysctl, *ifconfig*, *route*	✓	✗
IPv6 address config.	✓	✗
full *POSIX*	✓	✗
poll implementation	✓	✗
Quagga routing stack	✓	✓
extensive test	✓	✗
real time scheduler	✗	✓
mobility models	●●○	●●●
propagation models	●○○	●●●
supported technologies	WiFi	LTE/WiFi

Table 1.4: Comparison between Mininet-WiFi and DCE/ns-3

Mininet-WiFi, serving as a basis for reference implementation.

1.2 Mininet-WiFi

Mininet-WiFi is a fork of Mininet [18] extended with the required classes to add wireless channel emulation, node mobility, and support of 802.11 through *SoftMac*, a MAC layer that provides a flexible environment for experimenting with MAC protocols. Figure 1.4 depicts the architecture of Mininet-WiFi connections in a simple topology with two hosts, where the newly implemented components are presented along the original Mininet building blocks. More specifically, we added WiFi interfaces on STAs that now are able to connect to an AP 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 Linux OS network namespaces interconnected through virtual Ethernet (veth) pairs. The wireless interfaces to virtualize WiFi devices work on master mode for APs and managed mode for STA.

Stations: are devices that connect to an AP 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 STA). Since the traditional Mininet hosts are connected to an AP, STAs 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 APs 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.

[2]Hostapd (Host Access Point Daemon) user space software capable of turning normal wireless network interface cards into access points and authentication servers

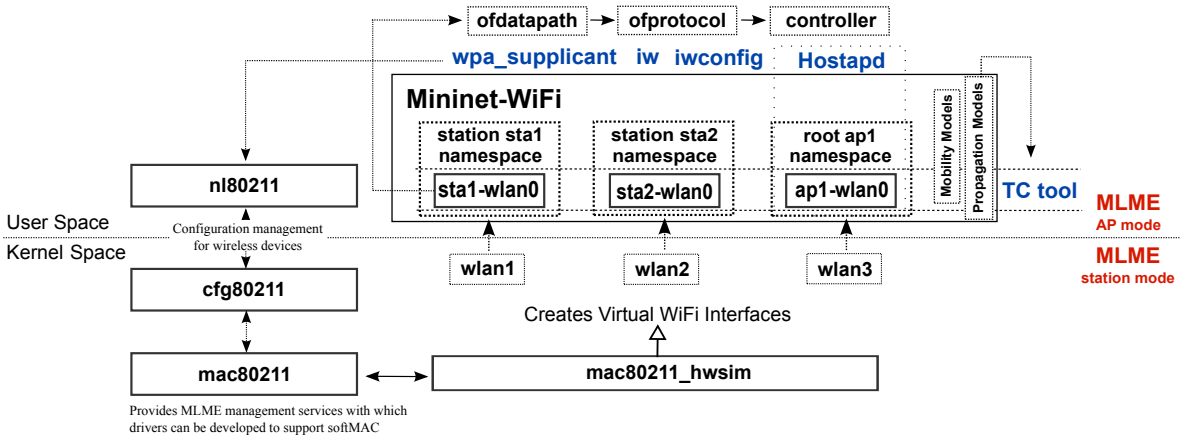

Figure 1.3: Main components of the Mininet-WiFi architecture.

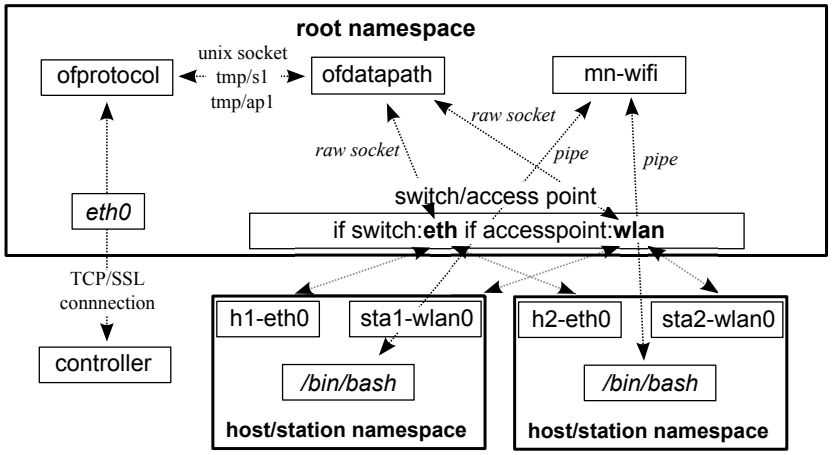

Figure 1.4: System architecture of Mininet-WiFi in a two-host environment.

The main components of the Mininet-WiFi architecture are illustrated in Figure 1.3. In the kernel-space, the module *mac80211_hwsim* is responsible for creating virtual Wi-Fi interfaces, enabling the creation of stations and access points. Still in the kernel-space, MLME (*Media Access Control Sublayer Management Entity*)[3] is realized by stations, while *hostapd* is responsible for the counterpart task at the user-space side in APs.

Mininet-WiFi relies on a couple of standard Linux utilities such as *iw*, *iwconfig*, and *wpa_supplicant*. The first two tools are used for interface configuration and for getting information from wireless interfaces while the latter is used with *Hostapd* in order to support WPA (*Wi-Fi Protected Access*), among other tasks. Both *infrastructure* and *ad-hoc* networks are supported. Another fundamental utility to realize the emulation of the wireless channel is *TC* (*Traffic Control*),[4] a user-space utility program used to configure the Linux kernel packet scheduler to control packet rate, delay, latency, and

[3] Some of the functions performed by MLME are authentication, association, sending and receiving *beacons*, etc.
[4] http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html

loss. TC applies these attributes in virtual interfaces of stations and APs, allowing Mininet-WiFi to replicate with high fidelity the actual packet behavior as observed in the real world. The *mobility* and *propagation models* do not require any kernel modification or changes in the applications.

Further Info & Resources

- Github repository and Wiki: `https://github.com/intrig-unicamp/mininet-wifi`
- User Manual: `https://www.dropbox.com/s/jgkhsdrz736e4q2/mininet-wifi-draft-manual.pdf?dl=1`
- Demos and Videos: `https://github.com/intrig-unicamp/mininet-wifi/wiki/Demos`
- Reproducible Research: `https://github.com/ramonfontes/reproducible-research`

Question 1.1: Which is the main scientific publication related to Mininet-WiFi?

2. Guided exercises/demo

In this chapter, you will learn the basics on how the Mininet-WiFi wireless network emulator works. Guided exercises will be explored and pointers to source code for those interested in delving deeper into the system architecture will be also provided. The pointers include stretches of the code where the link (including latency properties) can be customized.

2.1 Activity 1: Warming up

First of all, you have to stop the `network-manager`:

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

Then, create a simple topology:

```
sudo mn --wifi
```

The command above will start Mininet-WiFi and configure a small network with two stations, and one access point. Use the option --topo of the `mn` command to discover further topology options.
In Mininet-WiFi terminal (*mininet-wifi*>), execute the command nodes to observe the created network. Then, execute `iwconfig` to verify the association between the stations and ap1:

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

Then, disconnect sta1 and confirm the disconnection:

```
mininet-wifi>sta1 iw dev sta1-wlan0 disconnect
mininet-wifi>sta1 iwconfig
mininet-wifi>sta1 ping sta2
```

Now, connect sta1 again:

```
mininet-wifi>sta1 iw dev sta1-wlan0 connect my-ssid
mininet-wifi>sta1 iwconfig
mininet-wifi>sta1 ping sta2
```

2.2 Activity 2: Loading your first Wireless 3D Topology

Run the `wifiPosition.py` wireless-enabled topology (featuring 3D coordinates) example:

```
sudo python examples/wifiPosition.py
```

Now, observe the position of sta1, sta2 and ap1:

```
mininet-wifi>py sta1.params['position']
mininet-wifi>py sta2.params['position']
mininet-wifi>py ap1.params['position']
```

Observe the signal power as well:

```
mininet-wifi>py sta1.params['rssi']
mininet-wifi>py sta2.params['rssi']
```

You can also verify the distance between any two nodes:

```
mininet-wifi>distance sta1 ap1
mininet-wifi>distance sta1 sta2
```

Question 2.1: What is the observed bandwidth between sta1 and sta2?
***Tip**: try iperf sta1 sta2*

Now, move sta1 to a new position:

```
mininet-wifi>py sta1.setPosition('70,40,0')
```

Question 2.2: What happened with the association between sta1 and ap1?
***Tip**: try sta1 iwconfig*

Finally, increase the signal range of ap1:

```
mininet-wifi>py ap1.setRange(60)
```

Question 2.3: What happened with the association between sta1 and ap1 now?
Question 2.4: Now, observe the bandwidth between sta1 and sta2 again. What can we conclude?

2.3 Activity 3: Customizing the Wireless Channel

Mininet-WiFi relies on the configuration of Linux TC (by default) to control the wireless channel properties, such as *bandwidth*, *packet loss*, *delay* and latency. Please see the equations in `mininet/wifiChannel.py` (cf. `equationBW`, `equationLoss`, equationDelay and `equationLatency`). Those equations can be customized by calling `setChannelEquation()`, for example:

```
net.setChannelEquation(bw='value.rate * (1.1 ** -dist)', loss='(dist *
↪ 2)/100', delay='(dist / 10) + 1', latency='2 + dist')
```

Alternatively, you can use a recently implemented approach called `wmediumd`, a wireless medium simulation tool for Linux based on the *netlink* API implemented in the `mac80211_hwsim` kernel driver. A couple of sample files that use `wmediumd` are available at `/examples`.

Question 3.1: Run `wifiPosition.py` again and try `sta1 tc qdisc` before and after moving sta1 to the new position. What can you conclude about the configuration applied by `tc`?

3. Deeper hands-on: Mobility & RSS

3.1 Activity 4: Mobility

Open `examples/wifiMobilityModel.py` with any text editor and change the speed of stations:

```
from: net.startMobility(startTime=0, model='RandomDirection', max_x=100,
↪  max_y=100, min_v=0.5, max_v=0.8)
to: net.startMobility(startTime=0, model='RandomDirection', max_x=100,
↪  max_y=100, min_v=0.1, max_v=0.1)
```

Run `examples/wifiMobilityModel.py` and change the signal range of ap1:

```
python examples/wifiMobilityModel.py
mininet-wifi>py ap1.setRange(60)
```

Then, ping sta1 and sta2:

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

Question 4.1: What can you conclude about the observed latency?

Tip: *you can issue sta1 tc qdisc, repeatedly, to see the values applied by tc.*

3.2 Activity 5: Received Signal Strength

Open *examples/wifiPosition.py* and add sta3 at `position='10,10,10'` and set `max_z=100` in order to plot a 3D graph. Then, run `examples/wifiPosition.py`.

Question 5.1: What is the received signal strength indicator (RSSI) observed from sta3?

Question 5.2: What is the average ping response time between sta2 and sta1? And between sta3 and sta1? ***Note: set the number of packets to 10 (`ping -c10`).***

3.3 Activity 6: TC *versus* Wmediumd

First of all you have to get the code from `https://github.com/ramonfontes/reproducible-research/blob/master/mininet-wifi/ACROSS-Sweden-2017/tc_wmediumd.py` and run it with the command below (the content of `tc_wmediumd.py` is also available in the Appendix 5.1):

```
sudo python tc_wmediumd.py
```

Then, run (*observe the signal level from iwconfig*):

```
mininet-wifi>sta1 iw dev sta1-wlan0 scan
mininet-wifi>sta1 tc qdisc
mininet-wifi>sta2 iwconfig
mininet-wifi>sta3 iwconfig
mininet-wifi>exit
```

Afterwards, run tc_wmediumd again with the parameter **-w**:

```
sudo python tc_wmediumd.py -w
```

Then, run (*observe the signal level from iwconfig*):

```
mininet-wifi>sta1 iw dev sta1-wlan0 scan
mininet-wifi>sta1 tc qdisc
mininet-wifi>sta2 iwconfig
mininet-wifi>sta3 iwconfig
mininet-wifi>exit
```

Question 6.1: What can you conclude about the behavior of Mininet-WiFi by using tc and wmediumd? In your opinion, what is the best solution (tc or wmediumd)?

4. OpenFlow

You now will learn and put into practice basic concepts of the OpenFlow protocol, such as idle/hard timeouts, and apply them into a wireless scenario to understand the impact in mobile communications.

4.1 Activity 7: Mobility and OpenFlow

First of all, get the code from `https://github.com/ramonfontes/reproducible-research/blob/master/mininet-wifi/ACROSS-Sweden-2017/handover.py` and run it with the command below (the content of `handover.py` is also available in the Appendix 5.2):
Curiosity: *Mininet-WiFi already supports bgscan and 802.11r, alternatives that provide faster handover.*

```
sudo python handover.py
```

Now, let h1 keep pinging sta1:

```
mininet-wifi>h1 ping sta1
```

Question 7.1: As you can see, h1 cannot reach sta1 when sta1 goes to ap2. Why? Two important commands should help you to answer this question:

```
mininet-wifi>links
mininet-wifi>sh ovs-ofctl dump-flows s3
```

Tip: *Observe both `idle_timeout` and `idle_age`.*

Question 7.2: Now you know the answer, how could sta1 be reached by h1? Suggest (and prototype) some possible solutions.

5. Appendix

5.1 tc_wmediumd.py

```
#!/usr/bin/python

'Starting a topology with and without wmediumd'

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

def topology(wmediumd):

    "Create a network."
    if wmediumd:
        net = Mininet(controller=Controller, link=TCLink, accessPoint=OVSKernelAP,
    useWmediumd=True, enable_interference=True)
    else:
        net = Mininet(controller=Controller, link=TCLink, accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1', mac='00:00:00:00:00:01', ip='10.0.0.1/8', position='
    120,140,0')
    sta2 = net.addStation('sta2', mac='00:00:00:00:00:02', ip='10.0.0.2/8', position='
    10,30,0')
    sta3 = net.addStation('sta3', mac='00:00:00:00:00:03', ip='10.0.0.3/8', position='
    10,50,0')
    ap1 = net.addAccessPoint('ap1', ssid='new-ssid', mode='g', channel='1', position='
    15,30,0', ieee80211r='yes', mobility_domain='a2b2')
    c1 = net.addController('c1', controller=Controller)

    print "*** Configuring wifi nodes"
    net.configureWifiNodes()

    print "*** Starting network"
    net.build()
```

```
    c1.start()
    ap1.start([c1])

    """uncomment to plot graph"""
    net.plotGraph(max_x=150, max_y=150)

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

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

if __name__ == '__main__':
    setLogLevel('info')
    wmediumd = True if '-w' in sys.argv else False
    topology(wmediumd)
```

Code 5.1: tc_wmediumd.py

5.2 handover.py

```
#!/usr/bin/python

'Example for handover'

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

def topology():

    "Create a network."
    net = Mininet(controller=Controller, link=TCLink, switch=OVSKernelSwitch,
    accessPoint=OVSKernelAP)

    print "*** Creating nodes"
    sta1 = net.addStation('sta1', mac='00:00:00:00:00:01', ip='10.0.0.1/8')
    ap1 = net.addAccessPoint('ap1', ssid='new-ssid1', mode='g', channel='1', position=
    '15,30,0')
    ap2 = net.addAccessPoint('ap2', ssid='new-ssid1', mode='g', channel='6', position=
    '55,30,0')
    s3 = net.addSwitch('s3')
    h1 = net.addHost('h1', mac='00:00:00:00:00:02', ip='10.0.0.2/8')
    c1 = net.addController('c1', controller=Controller, port=6653)

    print "*** Configuring WiFi Nodes"
    net.configureWifiNodes()

    net.plotNode(h1, position='35,90,0')
    net.plotNode(s3, position='35,80,0')

```

```
30      print "*** Creating links"
31      net.addLink(ap1, s3)
32      net.addLink(ap2, s3)
33      net.addLink(h1, s3)
34
35      print "*** Starting network"
36      net.build()
37      c1.start()
38      ap1.start([c1])
39      ap2.start([c1])
40      s3.start([c1])
41
42      """uncomment to plot graph"""
43      net.plotGraph(max_x=100, max_y=100)
44
45      net.startMobility(startTime=0)
46      net.mobility(sta1, 'start', time=1, position='10,30,0')
47      net.mobility(sta1, 'stop', time=80, position='60,30,0')
48      net.stopMobility(stopTime=80)
49
50      print "*** Running CLI"
51      CLI(net)
52
53      print "*** Stopping network"
54      net.stop()
55
56  if __name__ == '__main__':
57      setLogLevel('info')
58      topology()
```

Code 5.2: handover.py

Bibliography

Ahrenholz, Jeff et al. (2008). "CORE: A real-time network emulator". In: *MILCOM 2008 - 2008 IEEE Military Communications Conference*. (IEEE). DOI: 10.1109/milcom.2008.4753614. URL: http://dx.doi.org/10.1109/MILCOM.2008.4753614.

Bernardos, Carlos et al. (2014). "An architecture for software defined wireless networking". In: *Wirel. Commun., IEEE* 21.3, pp. 52–61. ISSN: 1536-1284. DOI: 10.1109/mwc.2014.6845049. URL: http://dx.doi.org/10.1109/mwc.2014.6845049.

Calhoun, P. et al. (2010). *Lightweight Access Point Protocol - RFC 5412*. Tech. rep. DOI: 10.17487/rfc5412.

Cao, Zhen et al. (2016). *Alternate Tunnel Encapsulation for Data Frames in CAPWAP*. Internet-Draft draft-ietf-opsawg-capwap-alt-tunnel-08. Work in Progress. Internet Engineering Task Force. 24 pp. URL: https://tools.ietf.org/html/draft-ietf-opsawg-capwap-alt-tunnel-08.

Chan, Min-Cheng et al. (2014). "OpenNet: A simulator for software-defined wireless local area network". In: *IEEE Wireless Communications and Networking Conference, WCNC 2014, Istanbul, Turkey, April 6-9, 2014*. http://dx.doi.org/10.1109/WCNC.2014.6953088: IEEE, pp. 3332–3336. DOI: 10.1109/WCNC.2014.6953088.

Costanzo, Salvatore et al. (2012). "Software Defined Wireless Networks: Unbridling SDNs". In: *Software Defined Networking (EWSDN), 2012 European Workshop on*. http://dx.doi.org/10.1109/ewsdn.2012.12: IEEE, pp. 1–6. ISBN: 978-1-4673-4554-5. DOI: 10.1109/ewsdn.2012.12.

Dely, Peter et al. (2012). "CloudMAC - An OpenFlow based architecture for 802.11 MAC layer processing in the cloud". In: *Globecom Workshops (GC Wkshps), 2012 IEEE*. http://dx.doi.

org/10.1109/glocomw.2012.6477567: IEEE, pp. 186–191. ISBN: 978-1-4673-4942-0. DOI: 10.1109/glocomw.2012.6477567.

Doria, Avri et al. (2015). *Forwarding and Control Element Separation (ForCES) Protocol Specification*. RFC 5810. DOI: 10.17487/rfc5810. URL: https://rfc-editor.org/rfc/rfc5810.txt.

Enns, Rob et al. (2015). *Network Configuration Protocol (NETCONF)*. RFC 6241. DOI: 10.17487/rfc6241. URL: https://rfc-editor.org/rfc/rfc6241.txt.

Fontes, Ramon R. et al. (2015). "Mininet-WiFi: Emulating Software-defined Wireless Networks". In: *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*. CNSM '15. Washington, DC, USA: IEEE Computer Society, pp. 384–389. ISBN: 978-3-9018-8277-7. DOI: 10.1109/CNSM.2015.7367387. URL: http://dx.doi.org/10.1109/CNSM.2015.7367387.

Foundation, Open Networking. *Wireless & Mobile*. (accessed 07 March 2017).

— (2014). *OpenFlow Switch Specification - Version 1.5.0*. (accessed 07 March 2017).

Han, Bo et al. (2015). "Network function virtualization: Challenges and opportunities for innovations". In: *Commun. Magaz., IEEE* 53.2, pp. 90–97. DOI: 10.1109/MCOM.2015.7045396. URL: http://dx.doi.org/10.1109/MCOM.2015.7045396.

Jagadeesan, Nachikethas A. and Bhaskar Krishnamachari (2014). "Software-Defined Networking Paradigms in Wireless Networks: A Survey". In: *ACM Comput. Surv.* 47.2, 27:1–27:11. ISSN: 0360-0300. DOI: 10.1145/2655690. URL: http://doi.acm.org/10.1145/2655690.

Kargl, Frank and Elmar Schoch (2007). "Simulation of MANETs: A Qualitative Comparison Between JiST/SWANS and Ns-2". In: *Proceedings of the 1st Int. Workshop on System Evaluation for Mobile Platforms*. MobiEval '07. San Juan, Puerto Rico: ACM, pp. 41–46. ISBN: 978-1-59593-762-9. DOI: 10.1145/1247721.1247730. URL: http://doi.acm.org/10.1145/1247721.1247730.

Kreutz, Diego et al. (2015). "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1, pp. 14–76. DOI: 10.1109/jproc.2014.2371999. URL: https://doi.org/10.1109%2Fjproc.2014.2371999.

Kumar, Swarun et al. (2013). "Bringing Cross-layer MIMO to Today's Wireless LANs". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. Hong Kong, China: ACM, pp. 387–398. ISBN: 978-1-4503-2056-6. DOI: 10.1145/2486001.2486034. URL: http://doi.acm.org/10.1145/2486001.2486034.

Lantz, Bob, Brandon Heller, and Nick McKeown (2010). "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 19:1–19:6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868466. URL: http://doi.acm.org/10.1145/1868447.1868466.

Mancini, Emilio et al. (2014). "Demo abstract: Realistic Evaluation of Kernel protocols and Software Defined Wireless Networks with DCE/ns-3". In: Demo Abstract in Proceedings of ACM MSWiM, Montreal, Canada, September 21-26 2014, pp. 335–337. DOI: 10.1145/2641798.2655182. URL: https://hal.inria.fr/hal-01111026.

McKeown, Nick et al. (2008). “OpenFlow: Enabling Innovation in Campus Networks”. In: *SIGCOMM Comput. Commun. Rev.* 38.2, pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: http://doi.acm.org/10.1145/1355734.1355746.

Monin, Sergey, Alexander Shalimov, and Ruslan Smeliansky (2014). *Chandelle: Smooth and Fast WiFi Roaming with SDNOpenFlow*. http://www.usenix.org/sites/default/files/ons2014-poster-monin.pdf. (accessed 07 March 2017).

Moura, Henrique et al. (2015). “Ethanol: Software defined networking for 802.11 Wireless Networks”. In: *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*. http://dx.doi.org/10.1109/INM.2015.7140315: IEEE, pp. 388–396. DOI: 10.1109/INM.2015.7140315. URL: http://dx.doi.org/10.1109/INM.2015.7140315.

Pechlivanidou, Katerina et al. (2014). “NITOS testbed: A cloud based wireless experimentation facility”. In: *2014 26th International Teletraffic Congress (ITC)*. Blekinge Institute of Technology, Karlskrona, Sweden: (IEEE), pp. 1–6. DOI: 10.1109/itc.2014.6932976. URL: http://dx.doi.org/10.1109/ITC.2014.6932976.

Raychaudhuri, D (2003). “Orbit: Open-access research testbed for next-generation wireless networks”. In: *NSF award# ANI-0335244* 7.

Sama, Malla Reddy et al. (2015). “Software-defined control of the virtualized mobile packet core”. In: *IEEE Commun. Magaz.* 53.2, pp. 107–115. DOI: 10.1109/MCOM.2015.7045398. URL: http://dx.doi.org/10.1109/MCOM.2015.7045398.

Shao, Chunju et al. (2015). *IEEE 802.11 Medium Access Control (MAC) Profile for Control and Provisioning of Wireless Access Points (CAPWAP)*. RFC 7494. DOI: 10.17487/rfc7494. URL: https://rfc-editor.org/rfc/rfc7494.txt.

Suresh, Lalith et al. (2012). “Towards Programmable Enterprise WLANS with Odin”. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN ’12. Helsinki, Finland: ACM, pp. 115–120. ISBN: 978-1-4503-1477-0. DOI: 10.1145/2342441.2342465. URL: http://doi.acm.org/10.1145/2342441.2342465.

Testbeds, Onelab Future Internet (2016). *R2lab Testbed*. http://r2lab.inria.fr/. (accessed 27 September 2016).

Varga, András and Rudolf Hornig (2008). “An Overview of the OMNeT++ Simulation Environment”. In: *Proceedings of the 1st Int. Conf. Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Simutools ’08. Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 60:1–60:10. ISBN: 978-963-9799-20-2. URL: http://dl.acm.org/citation.cfm?id=1416222.1416290.

Wang, Shie-Yuan, Chih-Liang Chou, and Chun-Ming Yang (2013). “EstiNet openflow network simulator and emulator”. In: *IEEE Communications Magazine* 51.9, pp. 110–117. DOI: 10.1109/mcom.2013.6588659. URL: https://doi.org/10.1109%2Fmcom.2013.6588659.

White, Brian et al. (2002). “An Integrated Experimental Environment for Distributed Systems and Networks”. In: *SIGOPS Oper. Syst. Rev.* 36.SI, pp. 255–270. ISSN: 0163-5980. DOI: 10.1145/844128.844152. URL: http://doi.acm.org/10.1145/844128.844152.

Yang, Lily, Saravanan Govindan, and Hong Cheng (2015). *Objectives for Control and Provisioning of Wireless Access Points (CAPWAP)*. RFC 4564. DOI: 10.17487/rfc4564. URL: https://rfc-editor.org/rfc/rfc4564.txt.

Yap, Kok-Kiong et al. (2010). “Blueprint for Introducing Innovation into Wireless Mobile Networks”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*. VISA ’10. New Delhi, India: ACM, pp. 25–32. ISBN: 978-1-4503-0199-2. DOI: 10.1145/1851399.1851404. URL: http://doi.acm.org/10.1145/1851399.1851404.

Zimmermann, Alexander et al. (2006). “Architecture of the Hybrid MCG-mesh Testbed”. In: *Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*. WiNTECH ’06. Los Angeles, CA, USA: ACM, pp. 88–89. ISBN: 1-59593-539-8. DOI: 10.1145/1160987.1161004. URL: http://doi.acm.org/10.1145/1160987.1161004.