Ramon Fontes
Christian Rothenberg

Wireless Network Emulation with Mininet-WiFi

*Featuring new **OpenFlow/SDN** and **P4** use cases and **advanced topics***

1st edition
2019

Ramon dos Reis Fontes
Christian Rodolfo Esteve Rothenberg

Wireless Network Emulation with Mininet-WiFi

1st edition

Campinas
Christian Rodolfo Esteve Rothenberg
2019

Credits

Authors
Ramon dos Reis Fontes
Christian Rodolfo Esteve Rothenberg

Reviewers
Michel Daoud Yacoub

About the authors

Ramon dos Reis Fontes received the degree in Information Systems from the Faculty of Technology and Sciences (FTC), in 2009, the Master degree in Systems and Computing from Salvador University (UNIFACS), in 2013, and the Ph.D. degree in Electrical Engineering in the area of Computer Engineering from the University of Campinas (UNICAMP), in 2018. His research interests includes Software-Defined Networking (SDN), wireless networks, distributed systems, cloud and fog computing, Network Functions Virtualization (NFV), and security. Ramon has published several papers on conferences and journals, and has continuously contributed to the development of free and open source software through his Github account (`https://github.com/ramonfontes`). A variety of codes and instructions on how to reproduce his research works can be found at `https://github.com/ramonfontes/reproducible-research/`.

I would like to thank the readers for their interest in this book. Please do not hesitate to contact us if you need any help on any subject covered in this book. I would like to thank my friends, colleagues and teachers who have helped me in many ways. I would also like to thank Prof. Dr. Christian Rothenberg, counselor, advisor and co-author of this book, for sharing his wisdom and encouragement throughout my career at the Faculty of Electrical Engineering and Computing at UNICAMP.

My special thanks go to my beloved wife, Suian. This book would not be a reality without your continued support. I would also like to thank my parents, Helio and Conceição, for their incentives and support.

To my daughter, Pietra, I dedicate this book.

Christian Rodolfo Esteve Rothenberg is Professor at the Department of Computer Engineering and Industrial Automation (DCA) of the Faculty of Electrical Engineering and Computation (FEEC) at the University of Campinas (UNICAMP) since 2013. He holds the Telecommunication Engineering

degree from the Technical University of Madrid (ETSIT - UPM), Spain, the M.Sc. (Dipl. Ing.) degree in Electrical Engineering and Information Technology from Darmstadt University of Technology (TUD), Germany, 2006, and the Ph.D. in Electrical Engineering from UNICAMP (2010). From 2010 to 2013, he worked as a senior researcher at CPqD R&D center in telecommunication on P&D projects in the area of IP platforms. He is the Principal Investigator of the Information & Networking Technologies Research & Innovation Group (INTRIG – `https://intrig.dca.fee.unicamp.br/`), CNPq Research Productivity Fellow level 2 (2017-2020), and CNPq Technological Development and Innovative Extension Fellow level 2 (2014-2016). His research interests include computer network architectures, virtualization, cloud computing, SDN, NFV, among others. He has 2 international patents and more than 120 magazine and conference publications, accumulating more than 6000 citations (h-index: 28, i10-index: 50+ – `https://scholar.google.com.br/citations?user=8PxuHPkAAAAJ&hl=en`).

The completion of a book is a great opportunity to reflect and express gratitude. Starting with our ancestors, in my case, my parents José Luis and Ana. In my academic life, I I thank all the teachers who influenced me with a special highlight to Professor Mauricio, PhD advisor, academic father, and friend, a key figure since I landed on this beloved Brazil. I am grateful for all the opportunities I have received from this country, its people, and institutions, including CPqD, FEEC, UNICAMP, the national funding agencies CNPq and FAPESP, Ericsson, among others. I thank students, from undergraduate to postgraduate, professional colleagues, and friends of everyday life. To our INTRIG group, and of course, to Prof. Dr. Ramon Fontes, the first PhD made in INTRIG, an example of graduate student and human being, father and co-author of this book that I am sure will contribute to the formation of more professionals. Finally, the most important vector in life, the family, my wife Marcela, my children Gabriel and Marina, my sources of energy and happiness. Thank you!

About the Reviewers

Daniel Senna

We would like to thank Daniel Senna, an editor at Textual Assessoria, for providing proofreading assistance.

Acknowledgements

The success of this project and the writing of this book were only possible thanks to the support, collaboration and trust of many people and institutions that helped to make them come true. Therefore, we would like to record our thanks.

We thank Katia Obraczka, professor at the University of California, Santa Cruz, California, for his valuable input and suggestions on Mininet-WiFi development steps. We also thank the *Institut National de Recherche en Informatique et en Automatique* (INRIA), especially to Thierry Turletti and Walid Dabbous, for receiving us at INRIA for six (6) months and for contributing in various aspects related to the development of Mininet-WiFi. For the financial support, we thank the *Fundação de Amparo à Pesquisa do Estado de São Paulo* (FAPESP), process 2014/18482-4 and *Conselho Nacional de Pesquisas* (CNPq), process 310930/2016-2. The INTRIG research group thanks Ericsson for the research funding received, without which the group would not have achieved its results, many of them leveraged by and contributing to Mininet-WiFi.

Thanks also to users, researchers, and/or developers who contributed to making this book a reality, more specifically: Prof. Dr. Chih-Heng Ke, from the National Quemoy University/Taiwan, for tips and shared experience on how wireless networks work; Brian Linkletter, for writing a tutorial, and helping spread Mininet-WiFi; Patrick Große, for developing a *Wmediumd* extension for Mininet-WiFi; and, of course, to the Mininet-WiFi community for all discussions that result in the development of an increasingly stable and complete emulator in terms of features supported.

Wireless Network Emulation with Mininet-WiFi
ISBN: 978-65-900571-5-0 (E-book)

Table of Contents

Preface

We are witnessing an impressive revolution in the field of Computer Networks. Advances in wireless communications such as the imminent deployment of 5G networks worldwide, the ability to virtualize network infrastructures (e.g. Network Slicing and Network Function Virtualization) and to program their behavior (e.g. Software-Defined Networking) are concrete examples of this new era.

These advances enable the design, development and deployment of innovative mechanisms aimed at, for instance, higher resilience, performance, energy efficiency, and security of the network and service ecosystem. A key element to explore these new opportunities is the use of tools that enable prototyping and testing novel ideas at an early stage, without the constraints and complexities associated with employing a real infrastructure. This is precisely the role occupied by Mininet-WiFi, an environment that allows one to create, explore and experiment with software-defined wireless networks from a personal computer.

As an instructor of the Undergraduate and Graduate Networking courses at INF-UFRGS, I often assign students to develop new mechanisms on software-defined network infrastructures. It was a pleasant surprise when, in 2016, I began using Mininet-WiFi. This environment has dramatically expanded the scope of possible work we could do, enabling us to design a whole new set of proposals, such as routing algorithms for efficient mobile video streaming, handoff strategies with traffic fluctuation awareness and load balancing mechanisms. This type of work was challenging, if not impossible, to carry out in the classic Mininet environment, as it was beyond its scope to provide primitives for dealing specifically with wireless communication.

This book closes a cycle of development, innovation and transfer of new knowledge to society. By very didactically documenting and explaining how to use Mininet-WiFi through scripts, example codes, illustrations and other resources, the book will foster many new experiences like the ones just mentioned. There is no doubt that it will endure as an important work in the academic, industry and government sectors. I congratulate the authors for their praiseworthy initiative of advancing knowledge in this fascinating and

fundamental field. There is no time to waste: the time has come to "roll up our sleeves" and begin our immersion in Mininet-WiFi. Good reading and enjoy!

Prof. Luciano Paschoal Gaspary
Institute of Informatics - UFRGS

Chapter Organization

This book is organized as follows:

Chapter I introduces theoretical fundamentals of wireless networks, software-defined wireless networks and also Mininet-WiFi. This chapter goes in-depth into concepts relevant to the learning objectives of this book. For a deeper understanding of the different topics explored, the reader will be provided references to relevant literature in the field;

Chapter II introduces the beginner level of Mininet-WiFi proficiency and is devoted solely to providing the working details of Mininet-WiFi, where its key functional aspects are described. If you are already proficient with Mininet-WiFi, you can focus on chapters III and IV instead. You do not need to be familiar with Mininet to use Mininet-WiFi, but if you are, you will certainly have a smoother feel as to how Mininet-WiFi works. The tutorials included in this chapter can be used as complementary activities in theory classes at the undergraduate level (e.g. EA074 at FEEC/UNICAMP), as well as in practical laboratory courses (e.g. EA080 at FEEC/UNICAMP);

Chapter III introduces the intermediate level of Mininet-WiFi proficiency, covers tutorials that employ wireless networking, software-defined wireless networking, as well as a number of concepts related to computer networking. This chapter also describes the use of some network applications, such as *tcpdump* and *Wireshark*. In addition to meeting the pedagogical goals of more advanced computer network classes such as those involving laboratory activities, the tutorials in this chapter are also suitable for graduate classes (e.g. IA369, IA376 at FEEC/UNICAMP) and specialization courses (e.g. INF-556 at IC/UNICAMP), as they allow experimental research to be carried in more complex scenarios, such as the development of SDN solutions using the OpenFlow protocol;

Finally, **Chapter IV** introduces the expert level of Mininet-WiFi proficiency, has tutorials about kernel manipulation, containers, security, IoT, vehicular networks, etc., with valuable information on adapting the OpenFlow protocol to wireless networks. This chapter is labeled as advanced because it requires

more in-depth knowledge and the use of third-party applications. Therefore, the tutorials in this chapter are best suited for specialization and graduate courses, not only in classes but also as technical training for master's and doctoral students, thus helping the development of experimental research aimed at advancing the state of the art. However, nothing prevents curious readers from reproducing these tutorials, since they have similar walkthroughs, as well as the support provided by the codes from the previous chapters.

Conventions used in this book

To facilitate the reading of this book, the following conventions have been adopted:

italic: indicates foreign language words or program/tool names.

<*file*>: indicates files or scripts.

The symbols below represent:

Complementary information to previously exposed content.

Relevant alert or remark.

Question regarding the topic being explored.

Citations and other complementary sources.

Demonstration videos.

Requirement(s) For each existing experiment there will be an indication of the prerequisites to conduce it. For convenience, we have assigned the "script(s) only" label to the prerequisites requiring only the use of scripts that are already available for use. Since all scripts were coded with Mininet-WiFi in mind, we do not include Mininet-WiFi in the Prerequisites tab. The same goes for all packages that are installed during the Mininet-WiFi installation process, such as OVS, the OpenFlow protocol, etc.

Other conventions

As there is a tendency to replace tools from the *net-tools* package by those of the *iproute2* package in Linux operating systems, network tools like *iw* and *ip* are preferred for the tutorials. Nonetheless, programs from the *net-tools* package can also be used.

Finally, due to code update issues all scripts are available in a repository on Github (`https://github.com/ramonfontes/mn-wifi-book-en`).

Precautions

It is recommended that all tutorials available in this book be completed using the latest version of Mininet-WiFi available on Github. Should you, the reader, find any inconsistencies in the tutorials, you may contact the authors of this book at any time for clarification.

Although this book brings hands-on experience at all times, we recommend that you review each command or configuration beforehand so that the entire process can be understood.
Do not try to complete the tutorials without clearly understanding what is being done!

I must use Linux. But why?

Because the code base of Mininet-WiFi, Mininet, was developed for Linux systems. The development of Mininet-WiFi has maintained the same operating structure as that of Mininet. The Ubuntu operating system should be preferred, especially its Long Term Support (LTS) versions, as they are the most stable Ubuntu distributions.

Why open source code?

We will answer this question with a simple answer: because most of the time we (you, I and everyone) have the freedom to use the tool/program we want. Whether it is because we need to do work or academic research, or because we want to know more about Mininet-WiFi, or even because we need to modify it to suit our needs, we can choose. However, this seems like a ready answer, which we often receive as an answer by others when we wonder about the advantages of opening the source code of a particular program.

Arguing chronologically, we could say that without Mininet-WiFi, I, Ramon, would not have obtained a doctoral degree; many researchers would not have done their research; Mininet, the emulator Mininet-WiFi was based on, would probably not exist either, and so on. What we mean is that without open source philosophy, we would not have access to Mininet and would not have developed Mininet-WiFi. Just as Mininet would not have been developed with-

out the previous development of its backbone and its subsequent availability for anyone to use. Most likely you would not even be reading this book now and many fewer persons would be interested in the subjects we covered in it.

Can you imagine how much research on other topics would be undermined by only focusing on the field of research covered in this book and ignoring other possibilities? Perhaps you will have a better idea as you complete the tutorials proposed throughout this book.

There is a whole chain that would be seriously impacted if the codes were not free to use. The main paper [14] about Mininet alone has had about 1500 direct citations, not to mention countless indirect citations by blogs and even the media.

Experiences: Impact, Reproducibility and Quality

If you are wondering whether it is worth researching and exposing your code to the public, even though it is still in the early stages of development, here are some reports of experiences we have gained throughout Mininet-WiFi development.

Impact. Making code, data, documentation and demonstration videos public has certainly contributed and still greatly contributes to increasing Mininet-WiFi's visibility. Although there has been no systematic and qualitative assessment of the Mininet-WiFi community, users have contributed to the project several times, whether by discussing or even suggesting code improvements and new implementations.

Reproducibility. Making data public and reproducible is not always a simple task. By the way, writing a book whose tutorials users can complete without any setbacks is not easy. Most likely there will be one or another tutorial that will not flow as expected. This is due to several factors, such as differences in tool versions, issues that may be related to the operating system, hardware, etc. Throughout the development of Mininet-WiFi, we had a hard time reproducing experiments by other researchers, as there was often not enough information to do so. For this reason, we have chosen not only to make our experiments

public, but also to describe how they can be reproduced. Making research reproducible generally adds credibility to its respective work and obtained results.

Quality. In a broader sense of research quality, all the experiences gained throughout the development of Mininet-WiFi allowed us to learn and refine our research. Although ensuring the reproducibility of a project increases workload, reproducible work has a number of significant advantages, such as: (i) synergy with open source functionality, as the latter increases the chances of direct and indirect reproducibility and, consequently, (ii) greater impact, since the chances of researchers using the solutions proposed by reproducible works increase; (iii) improvement of programming habits, wherein special attention is given to code quality; (iv) encouragement of the use of scientific workflows, as researchers are carefully concerned with providing reliable results so that anyone can produce results similar to those they have obtained.

So if you have the opportunity, and if your code is not a license, patent, or any other copyrighted content, try making your code public!

List of Figures

List of Tables

List of Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AP	Access Point
BER	Bit Error Rate
BOFUSS	Basic OpenFlow User-space Software Switch
BSS	Basic Service Set
CAPWAP	Control and Provisioning of Wireless Access Points
CD	Collision Detection
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
Hostapd	Host Access Point Daemon
HTTPS	Hyper Text Transfer Protocol Secure
IBSS	Independent Basic Service Set
IDS	Intrusion Detection System
IFB	Intermediate Functional Block

IoT	Internet of Things
LTE	Long-Term Evolution
LWAPP	Lightweight Access Point Protocol
MCS	Modulation and Coding Scheme
MIMO	Multiple Input, Multiple Output
MLME	Media Access Control Sublayer Management Entity
MPTCP	MultiPath TCP
MQTT	Message Queuing Telemetry Transport
NFV	Network Function Virtualization
OF	OpenFlow
ONF	Open Networking Foundation
OVSAP	OpenvSwitch Access Point
RSSI	Received Signal Strength Indicator
RTC	Request To Receive
RTS	Request To Send
SDN	Software Defined Networking
SDWN	Software Defined Wireless Networking
SNR	Signal to Noise Ratio
SSID	Service Set Identifier
STA	Station
SUMO	Simulation of Urban MObility
TC	Traffic Control
UserAP	User level Access Point
VANET	Vehicular Ad hoc NETwork
WLAN	Wireless Local Area Network

Introduction

1. Background

1.1 Wireless communications

Wireless communications continues to be one of the most vibrant fields in the telecommunications sector. Although they began in the late nineteenth and early twentieth centuries, wireless communication research and development activities intensified between the 1970s to 1990s, fueled by a growing demand for increasingly better connectivity. Initially driven by the development of cell phones for voice services and then data applications, wireless technologies keep evolving, fostered by new forms of content creation and consumption and interaction between humans, machines and everyday objects, a trend commonly known as the Internet of Things (IoT).

Conventional wireless communication networks encompass several elements, the most basic of which are listed below: (i) the wireless terminals - such as laptops, smartphones, which are the interface between the user and the network; (ii) radio links, which connect the terminals to an agent providing the network coverage service; (iii) base stations, which function as the coverage agents; (iv) switching and control centers, which concentrate the base stations

and connect them to other communication services.

There are numerous technologies that provide wireless services, such as Bluetooth, LTE, Zigbee, WiFi, among other means. Wireless communications have unique features that make them distinct from other technologies. One of them, and certainly the most important one, is the propagation of radio waves. A signal propagating from one point to another undergoes three types of phenomena, namely: attenuation, long-term fading and short-term fading. Attenuation refers to loss of transmission when the receiver moves away from the source. Long-term fading refers to conditions when the average signal changes slowly over time due to obstructions to the signal path, such as buildings, trees, etc. Short-term fading refers to quick fluctuations of the signal due to reflection, scattering and diffraction. There is also the problem of interference by services using the same frequency or even approximate frequencies.

Due to the increasing worldwide demand for wireless communications, new technologies are emerging so that systems can meet this demand. In any case, the development of any system, and, more specifically, wireless systems, requires a deep knowledge of the phenomena involved. Figure 1.1 exem-

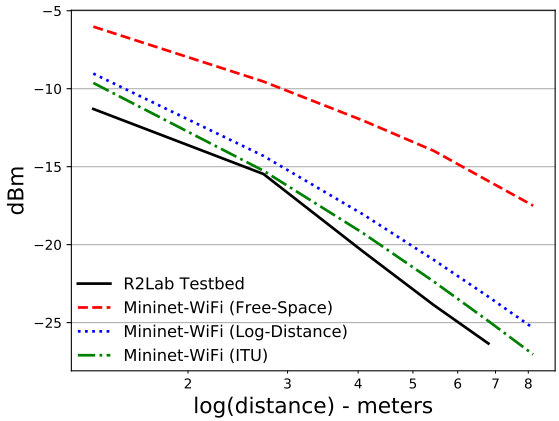

Figure 1.1: RSSI. Source: [18]

plifies the phenomenon of path loss by showing how the Received Signal Strength Indicator (RSSI), in dBm, oscillates in relation to the physical distance between a base station and a wireless station. The figure compares the estimations of different propagation models described in the literature (*Free-Space, Log-Distance, ITU*) - which are available in the Mininet-WiFi emulator - compared to measurements taken in a laboratory environment, the R2Lab[1] testbed. Figure 1.2, in turn, illustrates the phenomena of long-term and short-term fading.

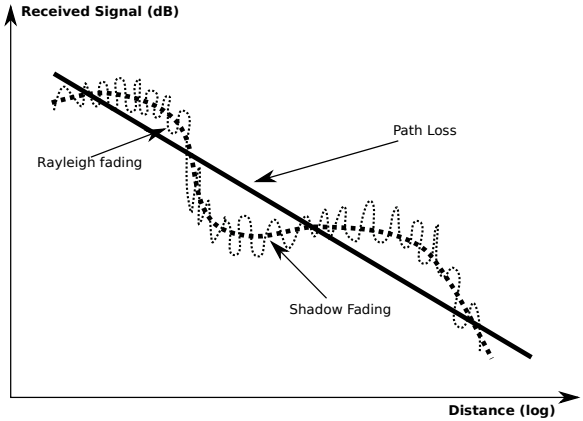

Figure 1.2: Effect of path loss.

- M. D. Yacoub, *Foundations Of Mobile Radio Engineering*. CRC Press, 1993. ISBN: 978-0849386770
- T. Rappaport, *Wireless Communications: Principles and Practice*, Pearson Education India, 2010. ISBN: 978-0130422323
- A. K. Jagannatham, *Principles of Modern Wireless Communication Systems Theory and Practice*. McGraw Hill Education, 2017. ISBN: 978-1259029578

1.2 WiFi: IEEE 802.11-based wireless local area networks

Established by the Institute of Electrical and Electronics Engineers (IEEE), IEEE 802.11 is the most accepted wireless communications standard in the

[1] `https://r2lab.inria.fr`

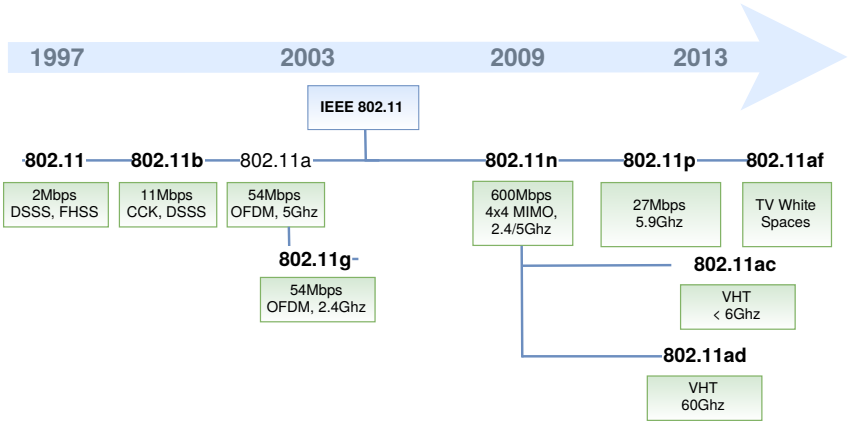

Figure 1.3: IEEE 802.11 modes.

world. WiFi technology, as it is most commonly known, is the Wireless Local Area Network (WLAN) technology based on IEEE 802.11, and it is a trademark of the Wi-Fi Alliance. The reasons for the wide acceptance of this pattern are diverse, but the main justification is cost-performance ratio.

As illustrated in Figure 1.3, there are several 802.11 standards, such as the older 802.11b, 802.11a, and 802.11g versions, and other versions that may be considered as newer, such as 802.11n, 802.11ac, 802.11p, and so on. In general, the standards defined for 802.11 operate on two main frequencies: 2.4 GHz or 5 GHz. In the example given by Figure 1.4, it can be seen how the 802.11b standard defines 13 channels on the 2.4 GHz band at 2.4835 Ghz, allocating 22 MHz for each channel, with a spacing of 5 MHz among them. With this arrangement, only channels 1, 6 and 11 can operate without band overlap.

The Bit Error Rate (BER), which is a requirement to be fulfilled in the system design, can be determined by knowing the modulation scheme, the type of encoding and the signal-to-noise ratio (SNR). It is known that an increase in transmitter power results in a higher SNR and a consequent decrease in BER. Obviously, power cannot be increased indefinitely, due to interference and to power limitations in the transmitter itself.

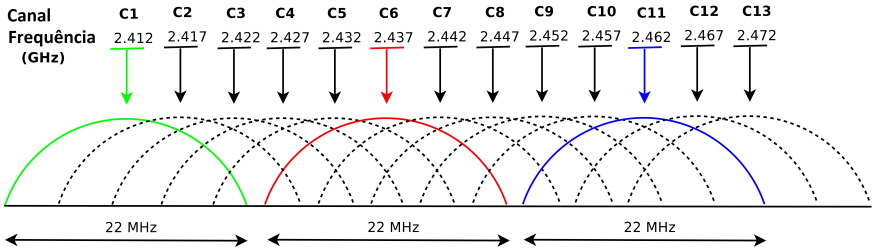

Figure 1.4: IEEE 802.11b channels. Source: adapted from [3] (CC BY 2.0)

Table 1.1: Comparing IEEE 802.11 modes.

Protocol	Freq. (GHz)	Bandwidth (MHz)	Internal Signal Range	External Signal Range
802.11	2.4	20	20 m / 66 ft	100 m / 330 ft
802.11a	3.7/ 5	20	35 m / 115 ft	120 m / 390 ft
802.11b	2.4	20	35 m / 115 ft	140 m / 460 ft
802.11g	2.4	20	38 m / 125 ft	140 m / 460 ft
802.11n	2.4/5	20 - 40	70 m / 230 ft	250 m / 820 ft
802.11ac	5	20/40/80/160	35 m / 115 ft	n/d
802.11ad	60	2,160	60 m / 200 ft	100 m / 300 ft
802.11ay	60	8000	60 m / 200 ft	1000 m / 3000 ft

Table 1.1 compares different 802.11 standards in terms of operating frequency, channel bandwidth and coverage radius estimates in indoor and outdoor environments. Table 1.2 compares 802.11n and 802.11ac, two of the newer standards that incorporate recent advances in wireless communications, such as spatial flows based on MIMO (Multiple Input Multiple Output).

The 802.11 architecture consists primarily of an access point and a number of wireless stations (clients). In this case, the architecture is defined as Basic Service Set (BSS), or infrastructure mode. In contrast, 802.11 networks composed of only wireless stations (clients) are referred to as Independent Basic Service Set (IBSS) or ad-hoc mode. The graphical representations of these two architectures (or modes of operation) are illustrated in Figures 1.5 e 1.6.

Table 1.2: IEEE 802.11n and IEEE 802.11ac comparison.

	IEEE 802.11n	**IEEE 802.11ac**
Frequency	2.4 GHz & 5 GHz	5 GHz
MIMO	Single User (SU)	Multi User (MU)
Spatial Flows	4	8
Taxa PHY	600 Mbps	6.9 Gbps
Channel Width	20 or 40 MHz	20, 40, 80, 80-80, 160 MHz
Modulation	64 QAM	256 QAM
Flow rate MAC*	390 Mbps	4.49 Gbps

*Assuming 65 % MAC efficiency and the highest rate of MCS (*Modulation and Coding Scheme*)

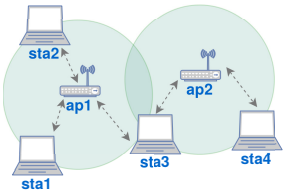

Figure 1.5: infrastructure.

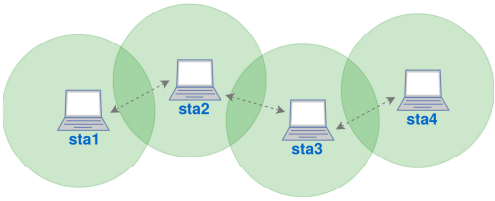

Figure 1.6: ad hoc.

As is true with Ethernet devices, each 802.11 wireless device has a 6-byte MAC address stored on the network interface card. It is through the wireless network interface that stations can associate with an access point or even other client stations before receiving or sending 802.11 frames.

Because wireless networks do not have the physical means to prevent collisions, these do happen even with the most advanced wireless technologies. While the IEEE 802 standards for Ethernet family cabling enable Collision Detection (CD), wireless networks have no means to detect a collision. The strategy adopted by the 802.11 standards to handle wireless access control is known as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA).

When the CSMA/CA method is used, each station informs about its own transmission intent and the associated time for Collision Avoidance (CA). The stations, which are equipped with wireless interfaces, listen to the medium using wireless interfaces to verify the presence of signals (signal level at

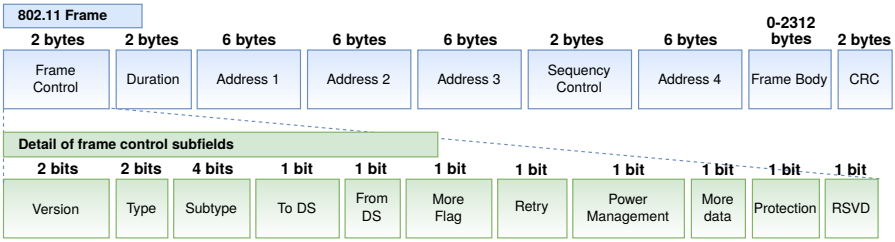

Figure 1.7: IEEE 802.11 frame header.

the carrier frequency) and wait until the medium is clear before transmitting. These mechanisms are known as Request to Send (RTS) and Clear to Send (CTS).

Despite the similarities between Ethernet frames and 802.11 frames, there are several fields that are specific to wireless links. The fields in the 802.11 table are shown in Figure 1.7. The numbers above each field in the frame represent their lengths in bytes, while the numbers above each of the sub-fields in the frame control field represent the lengths of the sub-fields in bits.

Although we do not go into detail about the function of each of the fields and sub-fields belonging to frame 802.11, it is advisable to know about them even if superficially. These fields may be useful for further exploration of some of the tutorials that will be presented throughout this book.

The future of WiFi

Although wireless networks are very important, there are still structural barriers that prevent their innovation, even with regard to WiFi itself. Furthermore, large wireless infrastructure is not completely accessible because there are restrictions on its use or authentication requirements. Namely, the issue here is not to open access to wireless networks completely and freely, but to allow users to connect to multiple networks (preserving security and quality standards), thus opening up a huge capacity for coverage and enabling continuous innovation, as proposed by [20].

Nevertheless, there are already several studies on vehicular networks and

also the Internet of Things that use WiFi in their methods. Many of them, of course, provide only suggestions for improvements that may advance 802.11 in the future. Yet it is not for nothing that researchers already speak of 802.11ax, an evolution of 802.11ac that promises to connect more devices with higher baud rates than its predecessor.

Among the proposals for improvements and advancements in wireless networks and especially WiFi, is the concept of software-defined wireless networks, which also promises significant progress by constructing a new idea of connectivity. Therefore, along with the concept of software-defined wireless networks, this book will present a series of tutorials that will explore various cases involving Mininet-WiFi. Mininet-WiFi is the wireless emulator that we will use extensively throughout this book. It was developed with the aim of providing an environment capable of supporting research on wireless networks and software-defined wireless networks, enabling innovations to be developed for the most diverse wireless technologies.

- Matthew S. Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, 2005. ISBN-13: 978-0596100520
- Matthew S. Gast. *802.11ac: A Survival Guide: Wi-Fi at Gigabit and Beyond*. O'Reilly Media (Edição: 2), 2013. ISBN-13: 978-1449343149
- Jim Geier, *Designing and Deploying 802.11 Wireless Networks: A Practical Guide to Implementing 802.11n and 802.11ac Wireless Networks For Enterprise-Based Applications*. Cisco Press, 2015. ISBN-13: 978-1587144301
- IEEE 802.11 Wireless Local Area Networks. The Working Group for WLAN Standards. Available at: `http://www.ieee802.org/11/`

1.3 Software-defined wireless networking

Software-defined wireless networking (SDWN) [5, 11] is an approach that allows centralized control of the network through the use of programs that do not necessarily have to be located in access points. Thus, rules defined by these programs (commonly known as controllers) dictate the behavior of the network. The principles of SDWN, which separate the control plane from the data plane, are very similar to those of software-defined networks (SDN) [12].

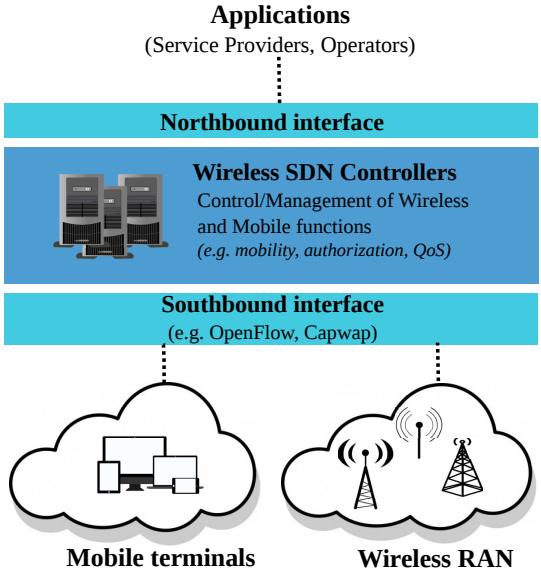

Figure 1.8: High-level and generic architecture for SDWN. Source: Adapted from [18]

The software-defined approach allows network administrators to specify network behavior in a logical and centralized way. To do so, they use programs provided by control platforms that implement southbound interfaces on network devices such as switches. In this context, the OpenFlow protocol [15] is the most popular southbound interface. However, there are other viable interfaces, such as CAPWAP [22], FORCES [7], NETCONF [8], etc.

Due to the increased interest of mobile operators [2, 19], mainly in Network Function Virtualization (NFV) [10], SDWN has become a branch of software-defined networks of considerable interest to the scientific community. The separation between the control plane and the data plane is not new in the history of wireless networks. The IETF standardized both the LWAPP (Lightweight Access Point Protocol) and the CAPWAP (Control and Provisioning of Wireless Access Points) many years ago by issuing RFC5412 [4] and RFC4564 [22], respectively - even before the development of software-defined networks and the OpenFlow protocol.

Many companies use wireless network management systems by means of protocols such as LWAPP and CAPWAP. LWAPP defines message control for configuration, authentication and other operations, while CAPWAP is based on LWAPP and allows a controller to manage different access points.

The number of studies on software-defined wireless networks has grown significantly in recent years. It is worth reading [11] for a more comprehensive survey, in addition to some software projects, such as: OpenRoads [23], Odin [21], OpenRF [13], Ethanol [17]. Architectures such as CloudMac [6] and Chandelle [16] use CAPWAP in their code. CloudMac describes wireless network management protocols, such as CAPWAP, as difficult to be configured with new features, since access point controllers that use CAPWAP are mostly proprietary systems. Chandelle, on the other hand, proposes a migration between smooth and fast access points using SDN/OpenFlow, but faces integration issues with regard to traditional switches and CAPWAP.

It is important to mention that there is an open source implementation of the CAPWAP protocol that is compatible with RFC 4515 and RFC 4516, called OpenCAPWAP [1], whose development started in 2015 (`https://github.com/vollero/openCAPWAP`).

The benefits of integrating wireless networks with OpenFlow generally involve centralized management and monitoring, unified policies, greater scheduling, and better control of wireless functions.

Taking into account these benefits and the limitations associated with CAPWAP, which is likely to be a more robust but closed-source solution, some questions are unavoidable: *"Is CAPWAP compatible with SDWN?"*, *"How to improve the OpenFlow specification, so that it supports centralized management of wireless networks? Or even, could you extend it to wireless networks?"*, *"Are new approaches needed?"* or *"How much could be recycled from the existing infrastructure?"*.

- L. E. Li, Z. M. Mao and J. Rexford, *Toward Software-Defined Cellular Networks*. European Workshop on Software Defined Networking (EWSDN), 2012.
- A. Gudipati et al., *SoftRAN: software defined radio access network*. Proceedings of Hot topics in software defined networking (HotSDN). 2013.
- C. J. Bernardos et al., *An architecture for software defined wireless networking*. IEEE Wireless Communications. 2014.
- T. Chen et al., *Software defined mobile networks: concept, survey, and research directions*, IEEE Communications Magazine. 2015.
- Mao Yang et al., *Software-Defined and Virtualized Future Mobile and Wireless Networks: A Survey*. Mob. Netw. Appl. 2015.
- I. T. Haque and N. Abu-Ghazaleh, *Wireless Software Defined Networking: A Survey and Taxonomy*, in IEEE Communications Surveys & Tutorials. 2016.
- A. Abdelaziz et al. *On Software-Defined Wireless Network (SDWN) Network Virtualization: Challenges and Open Issues*. Computer Journal. 2017.
- Linux Foundation's Open Networking Foundation (ONF) SDN Wireless Transport. Available at: `https://www.opennetworking.org/tag/wireless-transport/`

1.4 Mininet-WiFi

Network emulation has been widely used in performance evaluation, protocol testing and debugging, as well as in a variety of research on computer network architectures. A researcher typically has several possible methods to evaluate and validate research data and network protocols, as well as perform analyses, among other operations.

Simulators, emulators and testbeds are the main evaluation tools that help researchers in their tasks. Still, regarding their practical applications, all these evaluation tools are very different in their degree of abstraction. Some of the experimental platforms that can be used for experimentation with wireless networks are shown in Figure 1.9. In this research field, the emulation of wireless networks - which has peculiar characteristics, especially compared with emulators for wired networks - has to implement node mobility, signal propagation, among other features, to allow experiments with environments that have interference, signal attenuation, etc.

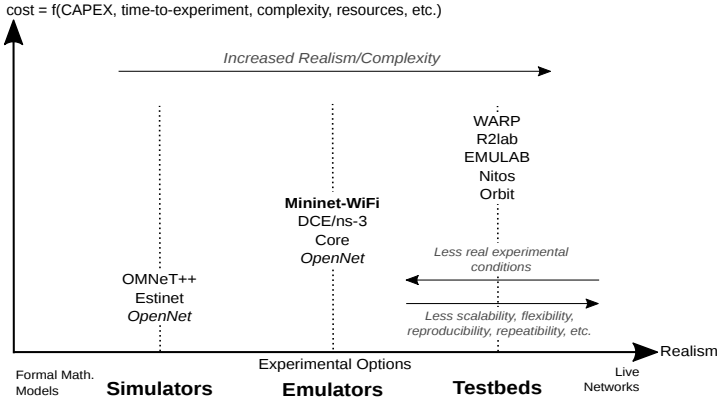

Figure 1.9: Experimental platforms for wireless networks. Source: Adapted from [9].

We will not go into detail about the differences between experimental platforms, but we can highlight two important features of Mininet-WiFi: (i) it allows the use of third-party tools without modifications to the source code of these tools, and (ii) it uses the actual network protocol stack.

Mininet-WiFi is an emulator for wireless networks that was extended from Mininet, a well-known emulator to researchers working in the field of software-defined networks. Mininet-WiFi has native WiFi support, but other wireless networking technologies can also be simulated in experiments using it. With Mininet-WiFi, the user can virtualize stations and access points and also use existing Mininet nodes such as hosts, switches and OpenFlow controllers. Consequently, Mininet-WiFi also enables the processing of packages using the OpenFlow protocol, an important solution for SDN.

SoftMAC is a term used to describe a type of wireless network interface in which the MAC Layer Management Entity (MLME), for example, is expected to be managed using software. Mac80211 is a driver API for SoftMAC.

Mininet-WiFi is developed based on the Mininet code and the most used WiFi *driver* for Linux systems, *SoftMac*. With Mininet-WiFi, the user can choose to

use the old Mininet features independently or use the extensions implemented for Mininet-WiFi.

1.4.1 Architecture

The entire virtualization process of Mininet-WiFi works similarly to Mininet, i.e. it is based on processes that run on Linux network namespaces and virtual network interfaces (see Figure 1.10). Linux network namespaces are, in a logical sense, copies of the Linux operating system's network stack, which includes its own routes, firewall rules and network devices. They act as if they were real computers, with the same network properties that a physical computer can have.

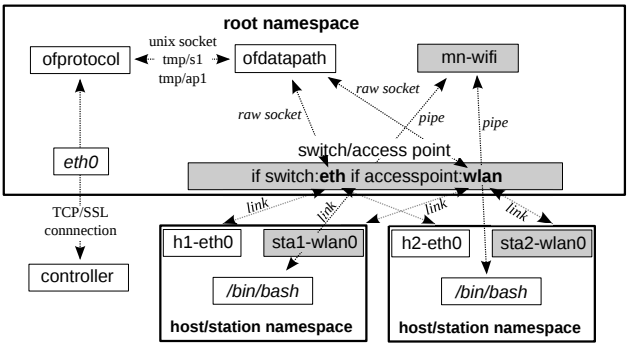

Figure 1.10: Mininet-WiFi architecture. Source: [9]

The behavior of wireless interfaces basically depends on the function they perform, such as, for instance, the case of stations and access points, whose interfaces operate in the managed or master modes, respectively. Just as with a real environment, the stations communicate with access points by a process called authentication and association. By default, each station has only one wireless interface, and more can be added if needed. Once connected to an access point, stations can communicate with traditional Mininet hosts, if they are also connected to the access point. Access points, on the other hand, are responsible for managing stations that are associated with them.

Conceptually, access points are the same entities as the Mininet switches,

but equipped with WiFi network cards operating in master mode. Access points are virtualized in the *hostapd*[2] daemon, which basically uses virtual WiFi interfaces to provide access point capabilities. Details on the running environment of Mininet-WiFi are discussed below.

1.4.2 Components

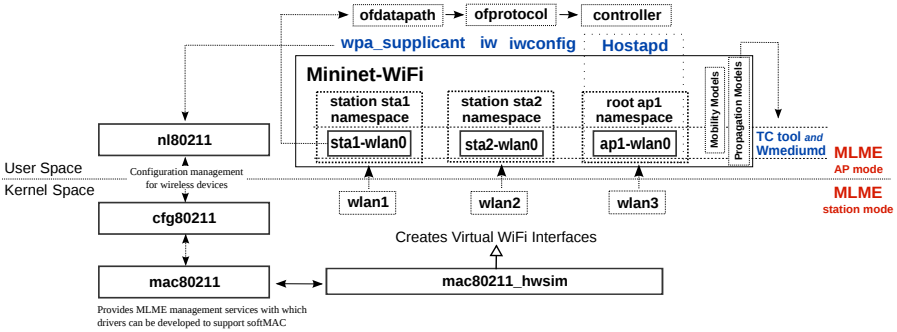

Figure 1.11: Main components of Mininet-WiFi. Source: [9]

The components comprising the Mininet-WiFi architecture are shown in Figure 1.11. Communication among them occurs as follows: during its initialization, the module called *mac80211_hwsim*, responsible for the virtualization of WiFi network cards, is loaded with the number of virtual wireless interfaces required for all nodes previously defined by the user. Located in the kernel space of the Linux operating system, all features supported by *mac80211_hwsim* come from mac80211, a framework based on *SoftMAC* that developers use to write drivers for wireless devices.

Also in the kernel space is *cfg80211*, which is an 802.11 heap configuration API for Linux systems. Its configuration is done by running *nl80211*, which also performs the interaction between kernel and user spaces.

The main network applications used by Mininet-WiFi are in the user space. Among them is *hostapd*, whose function is to provide access point services;

[2]Hostapd (**H**ost **A**ccess **P**oint **D**aemon) is a user-level software capable of launching a wireless network interface on access points and authentication servers.

the TC and Wmediumd programs, which will be described below; *iw*, *iwconfig* and *wpa_supplicant*. The latter is used for, among other tasks, WPA/WPA2 authentication.

Interacting with the emulation environment

Mininet-WiFi also maintains the same interaction structure as Mininet. E.g., commands such as those shown below can be used, respectively, for connectivity tests or to measure th bandwidth between two nodes. If you are already familiar with Mininet, this is certainly nothing new.

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

In addition to these, other commands exclusive to Mininet-WiFi can be used for a better experience with the WiFi environment, such as the ones described below:

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan
mininet-wifi> sta1 iw dev sta1-wlan0 connect ssid-ap1
```

These commands allow you to scan WiFi networks and connect to one of them, respectively. Scripts such as *iw*, the command used above, are natively supported by most Linux operating systems and have not been ported or modified to work on Mininet-WiFi. Mininet-WiFi can execute any command and/or program that runs on Linux distributions, such as Ubuntu.

- Ramon dos Reis Fontes, Samira Afzal, Samuel Brito, Mateus Santos, Christian Esteve Rothenberg. *Mininet-WiFi: Emulating Software-Defined Wireless Networks.* In 2nd International Workshop on Management of SDN and NFV Systems 2015. Barcelona, Spain, Nov. 2015. [9]

Level: beginner

2. Beginner

In this chapter we introduce Mininet-WiFi and all features supported by this emulator, highlighting critical information necessary to understand the tutorials explored throughout this book. We begin by discussing all steps needed to get Mininet-WiFi up and running on your computer.

2.1 Downloading and installing Mininet-WiFi

The Mininet-WiFi source code is a *Git repository* publicly available on *Github*. Git is an amazing open source system, capable of handling the distributed version control of any given project, in this case the Mininet-WiFi project. It was devised by Linus Torvalds[1] himself as a means of helping the development, at the time, of a tiny project called Linux.

To ease the searching and following-up of projects managed by Git, developers usually share their Git repository on Github, a platform for creating, managing, further distributing and interacting with open-source projects. This means

[1] `github.com/torvalds`

that the life cycle of a project can be easily analyzed by contributors through Github, which keeps any file's modification history since its origin. In this book we only use the basic concepts of the Git system. For more information about Git and Github, please refer to <*git-scm.com*> and <*github.com/*>.

To obtain Mininet-WiFi's source code and install it, you will need to perform a process called cloning, in which all the information pertaining to a project is downloaded to your computer. Since Mininet-WiFi is a Git repository on Github, its cloning is carried out using the Git system.

After this brief introduction to Git and Github, you can clone the Mininet-WiFi source code by using the following command line, which consists of the instruction git clone followed by a link to Mininet-Wifi's Github repository.

```
~$ git clone https://github.com/intrig-unicamp/mininet-wifi
```

Mininet-WiFi relies on Linux Kernel components to function properly. Of the different Linux distributions that can be used to this end, we recommend Ubuntu, since Mininet-WiFi was extensively tested on it.

In the link to Mininet-WiFi's repository, *intrig-unicamp* refers to the profile or organization where the repository is located on Github. *mininet-wifi*, in turn, is the name of the repository where the source code is deposited.

If you do not have *git*, you can install it using the `sudo apt install git` command.

Once the clone is complete, a directory named <*mininet-wifi*> should be created. Since the cloning was done from the user's directory, the Mininet-WiFi source code should be located at <*/home/your_username/mininet-wifi*>, or simply <*∼/mininet-wifi*>.

Now you need to install Mininet-WiFi. To do so, you will need to access the created directory and execute the `sudo util/install.sh` command, as follows.

```
~$ cd mininet-wifi
~/mininet-wifi$ sudo util/install.sh -Wlnfv6
```

Further information on the *Wlnfv6* parameters can be found on the Mininet-WiFi source code page on Github.

Alternatively, you can also use the virtual machine available on the source page. To ensure that the virtual machine has the latest version of Mininet-WiFi, you must use the commands below.

```
~/mininet-wifi$ git pull
~/mininet-wifi$ sudo make install
```

Capturing the code through the `git clone` command ensures that the source code will always contain the latest updates implemented for Mininet-WiFi.

Even if you already have Mininet-WiFi and/or the virtual machine installed, the `git pull` command can be issued from the Mininet-WiFi directory at any time. This command will synchronize the code that is on your computer with the source code available in the Mininet-WiFi source code repository. By doing this, you will always have the latest version of Mininet-WiFi installed.

2.2 First steps to use Mininet-WiFi

In the following paragraphs, we will begin to understand how to use Mininet-WiFi.

First, we need to be aware of three commands: `sudo mn --version`, which prints the Mininet-WiFi version in use; `sudo mn --help`, which prints a help menu; and `sudo mn -c`, which is responsible for cleaning up poorly-made Mininet-WiFi executions. Remember this last command, because it will be very useful later on.

Mininet-WiFi can be started by running a very simple command, `sudo mn --wifi`. In addition to opening the Command Line Interface (CLI), this command will create a topology consisted of two stations connected to an access

point via a wireless medium, as well as an SDN controller that is connected to the access point, as shown in Figure 2.1.

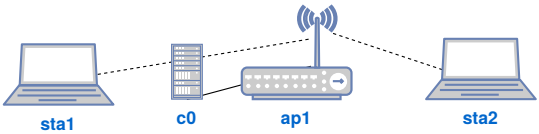

Figure 2.1: Simple topology.

```
~/mininet-wifi$ sudo mn --wifi
*** Creating network
*** Adding controller
*** Adding stations:
sta1 sta2
*** Adding access points:
ap1
*** Configuring wifi nodes...
*** Adding link(s):
(sta1, ap1) (sta2, ap1)
*** Configuring nodes
*** Starting controller(s)
c0
*** Starting switches and/or access points
ap1 ...
*** Starting CLI:
mininet-wifi>
```

If you already know Mininet, you have probably already used the `sudo mn` command, which creates a simple topology with two hosts, one switch and one OpenFlow controller, connected by a wired medium.

If you notice an error similar to the one below, it means that there is a controller or process already running on port 6653, the default port used by the most recent OpenFlow controllers. This problem can be solved using the `sudo fuser -k 6653/tcp` command, which will kill the process that is using port 6653. If the controller is running on port 6633, the same must be done with this port number.

```
Exception: Please shut down the controller which is running on port 6653:
Active Internet connections (servers and established)
tcp  0  0 0.0.0.0:6653   0.0.0.0:*  LISTEN   2449/ovs-testcontro
tcp  0  0 127.0.0.1:55118   127.0.0.1:6653 TIME_WAIT   -
```

To identify the Mininet-WiFi CLI, just search for the text below:

```
mininet-wifi>
```

Within the CLI you can essentially use any network commands or programs. Additionally, it is also possible to list and execute a number of commands that have been implemented exclusively for Mininet-WiFi. The `help` command allows you to list available commands, as follows.

```
  mininet-wifi> help
Documented commands (type help <topic>):
========================================
EOF      exit  iperf    nodes       pingpair     py     start  x
distance gterm iperfudp noecho      pingpairfull quit   stop   xterm
dpctl    help  links    pingall     ports        sh     switch
dump     intfs net      pingallfull px           source time
```

Most of these commands already existed in Mininet and were kept for Mininet-WiFi. Only three new commands have been added to Mininet-WiFi: `distance`, `start` and `stop`. `distance` allows you to check the distance between two nodes, while `start` and `stop` allow you to pause and continue experiments that implement node mobility.

This book will demonstrate the commands implemented for Mininet-WiFi, in addition to some others already implemented on Mininet.

Try using the `nodes` command to identify nodes that are part of the topology. Note that the nodes described by the `nodes` command are the same as those shown previously in Figure 2.1.
Note: Node `c0` will be discussed later.

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

As previously mentioned, the `sudo mn --wifi` command creates a topology with stations that are connected through a wireless medium to an access point. This can be easily verified using wireless networking tools.

Although the `sudo mn --wifi` command creates an AP with an SSID called "my-ssid" operating on channel 1 (2412MHz), these values can also be customized. For instance, we will exit the Mininet-WiFi CLI with the `exit` command and then set up a new SSID and a new channel, as follows:

```
mininet-wifi> exit
~/mininet-wifi$ sudo mn --wifi --ssid=new-ssid --channel=10
```

Then try the following command.

```
mininet-wifi> sta1 iw dev sta1-wlan0 info
Interface sta1-wlan0
        ifindex 33
        wdev 0x1000000001
        addr 02:00:00:00:00:00
        ssid new-ssid
        type managed
        wiphy 16
        channel 10 (2457 MHz), width: 20 MHz (no HT), center1: 2457 MHz
        txpower 14.00 dBm
```

If you are new to wireless networking, especially on Linux operating systems, you might not have noticed, but you have just used a very common program in wireless networking environments, the *iw* tool. *iw* is a utility for wireless networks that is gradually replacing *iwconfig*. We will use it extensively throughout this book.

iwconfig is certainly already installed on your system and you can also use it. For example, *sta1 iwconfig* will produce a similar result to the one shown previously by *iw*. Try running iwconfig --help for more information on how to use it.

With respect to the command that we have just used, the *info* parameter brings up information about the association (or no association) between nodes. It is noticeable that `sta1` is associated with an access point with a SSID *new-ssid*

that also operates on channel 10, exactly as defined by the command.

Additionally, using the *link* parameter instead of *info* allows the user to obtain the signal level perceived by the node and the *bitrate*, in addition to transmitted and received packets, among other data.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:02:00 (on sta1-wlan0)
    SSID: new-ssid
    freq: 2457
    RX: 1241 bytes (22 packets)
    TX: 93 bytes (2 packets)
    signal: -36 dBm
    tx bitrate: 1.0 MBit/s

    bss flags:        short-slot-time
    dtim period:        2rendering this PDF.

    beacon int:        100
```

Now, let us use the *ping* command to verify the connectivity between `sta1` and `sta2`.

```
mininet-wifi> sta1 ping -c1 sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.380 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.380/0.380/0.380/0.000 ms
```

The command shows that there is communication between the two nodes in question, since it also displays a response time in milliseconds belonging to `sta2`(*ms*). It is important to note that because Mininet-WiFi is an emulation platform capable of emulating several nodes, it is necessary to define in the CLI the source node that will be responsible, in practice, for issuing a given command.

The `-c1` parameter used with the *ping* command means that only one ICMP packet will be sent. Otherwise, `sta1` will send endless ICMP packets.

Thus, as the *ping* command needs a target node - which can be either a name or an IP address -, sta2's destination can also be replaced by its IP address. As can be seen below, the IP address that identifies sta2 is 10.0.0.2/8.

```
mininet-wifi> sta2 ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
↪  default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
34: sta2-wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc htb state
↪  UP group default qlen 1000
    link/ether 02:00:00:00:01:00 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/8 scope global sta2-wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::ff:fe00:100/64 scope link
       valid_lft forever preferred_lft forever
```

Alternatively, you can also open different terminals for each node and issue commands as if they were being sent directly to a computer, exactly as it happens in the real world (see Figure 2.2). For example, the following command will open two terminals, one for sta1 and another for sta2. Once there is a terminal for each node, it will no longer be necessary to indicate which one is the origin, as explained in the previous paragraph.

```
mininet-wifi> xterm sta1 sta2
```

Xterm may not work as expected if there is no GUI enabled on your operating system.

Now, we will perform a few routines and exclusive actions of the wireless environment. To begin, we will disconnect sta1 from ap1 and confirm the disassociation by issuing the following command:

```
mininet-wifi> sta1 iw dev sta1-wlan0 disconnect
mininet-wifi> sta1 iw dev sta1-wlan0 link
    Not connected.
```

So let us try a new *ping* between sta1 and sta2.

```
mininet-wifi> sta1 ping -c1 sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

As you can see, station `sta1` is no longer associated with access point `ap1`, so it would be logically impossible to perform any kind of communication with `sta2`.

Now, we will connect `sta1` again to the `ap1` access point and confirm the association.

```
mininet-wifi> sta1 iw dev sta1-wlan0 connect new-ssid
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:02:00 (on sta1-wlan0)
        SSID: new-ssid
        freq: 2457
        RX: 370 bytes (9 packets)
        TX: 202 bytes (3 packets)
        signal: -36 dBm
        tx bitrate: 6.0 MBit/s

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

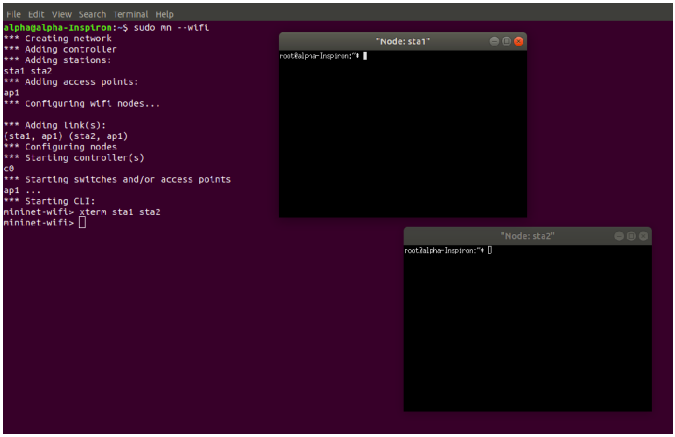

Figure 2.2: Executing xterm.

And then we will try a new *ping* between `sta1` and `sta2`. The *ping* command should run successfully, as follows.

```
mininet-wifi> sta1 ping -c1 sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1011 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1011.206/1011.206/1011.206/0.000 ms
```

Another very useful operation for WiFi networks is scanning, which allows you to check which access points a certain station can see. For example, let us assume that the SSID of access point `ap1` is unknown. In this case, the following command can be used to display `ap1`'s SSID.

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan
BSS 02:00:00:00:02:00(on sta1-wlan0) -- associated
    TSF: 1534710096681871 usec (17762d, 20:21:36)
    freq: 2457
    beacon interval: 100 TUs
    capability: ESS ShortSlotTime (0x0401)
    signal: -36.00 dBm
    last seen: 0 ms ago
    Information elements from Probe Response frame:
    SSID: new-ssid
    Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
    DS Parameter set: channel 1
    ERP: Barker_Preamble_Mode
    Extended supported rates: 24.0 36.0 48.0 54.0
    Extended capabilities:
             * Extended Channel Switching
             * Operating Mode Notification
```

2.3 Customizing topologies

Different topologies can be created in Mininet-WiFi, through simple commands or even by using scripts written in *Python*.

The topologies that can be created through commands are *single* and *linear*. To generate these two kinds of topologies, we will need to close Mininet-WiFi.

```
mininet-wifi> exit
```

So let us start with the *single* topology, which consists of one access point, `ap1`, and n stations associated with it. For example, the following command creates four stations, one access point and one SDN controller, as shown in Figure 2.3.

```
~/mininet-wifi$ sudo mn --wifi --topo single,4
```

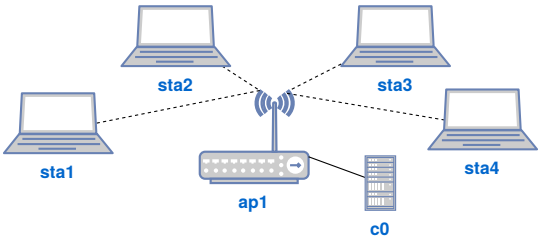

Figure 2.3: Single topology.

At this point, we can test the connectivity between all the nodes by issuing the `pingall` command, as follows.

```
mininet-wifi> pingall
*** Ping: testing ping reachability
sta1 -> *** sta1 : ('ping -c1  10.0.0.2',)
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.170 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.170/0.170/0.170/0.000 ms
sta2 *** sta1 : ('ping -c1  10.0.0.3',)
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.121 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.121/0.121/0.121/0.000 ms
sta3 *** sta1 : ('ping -c1  10.0.0.4',)
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.129 ms

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.129/0.129/0.129/0.000 ms
```

The other topology that can be created using commands is *linear*, which consists of n access points and n stations, in which each station is associated with one access point and all the access points are connected in a linear way. For example, the following command creates four access points, four stations, and one SDN controller, as shown in Figure 2.4.

```
~/mininet-wifi$ sudo mn --wifi --topo linear,4
```

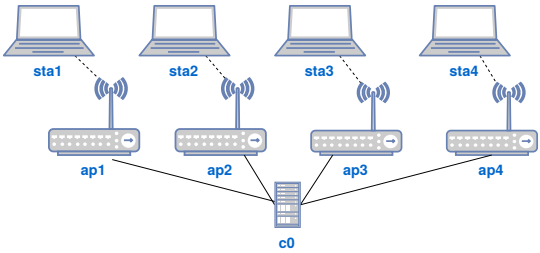

Figure 2.4: Linear topology.

The customization of topologies, on the other hand, is done by means of scripts that contain all the information about the topology as well as the configuration of its nodes. In the *</mininet-wifi/examples>* directory there is a wide variety of scripts that can be used as a basis for creating custom topologies.

It is always recommended that you check whether there is a script already developed for the scenario you want to work on. This helps you to create your own. Throughout this book we will use various scripts, which will certainly help in understanding how they can be customized.

2.4 Accessing node information

Now, let us learn how to get information from the nodes that make up a topology. To do so, we will create the simplest topology and add two new parameters: *position* and *wmediumd*. The *position* parameter will define initial positions for the nodes, while the *wmediumd* parameter will enable *wmediumd*, a wireless simulator that will be shown in 2.7.2.

```
~/mininet-wifi$ sudo mn --wifi --link=wmediumd --position
```

Then try issuing the `distance` command, as follows:

```
mininet-wifi> distance sta1 sta2
The distance between sta1 and sta2 is 100.00 meters
```

Now, check the position of `sta1` and `sta2`. Note that the x, y, and z axes are separated by commas.

```
mininet-wifi> py sta1.position
[1.0, 0.0, 0.0]

mininet-wifi> py sta2.position
[101.0, 0.0, 0.0]
```

As you can see, the initial positions were defined, and the `distance` command can be used to verify the distance between two nodes.

At this point a question surely may arise: what if a specific position for a node must be defined? In this case, there are two possible solutions: either through the Mininet-WiFi CLI or scripts. The example below shows the `setPosition()` method, which can be used with the CLI and scripts.

```
mininet-wifi> py sta1.setPosition('10,0,0')
```

Note that when a method implemented on the Mininet-WiFi source code is evoked by the CLI, the prefix *py* must always be used. In addition to `setPosition()`, other methods will be demonstrated throughout this book.

Now, let us check the newly defined position.

```
mininet-wifi> py sta1.position
[10.0, 0.0, 0.0]
```

In this case, the position is defined as: x=10, y=0 and z=0.

Various other data about a particular node can be obtained using the generic form *node.params* or *node.wintfs*, as shown below.

```
mininet-wifi> py sta1.params
{'wlan': ['sta1-wlan0'], 'ip': '10.0.0.1/8', 'ip6':
↪  '2001:0:0:0:0:0:0:1/64', 'channel': 1, 'mode': 'g'}
mininet-wifi> py sta1.wintfs
{0: <managed sta1-wlan0>}
```

Now, you can filter the desired information as follows.

```
mininet-wifi> py sta1.wintfs[0].freq
2.412
mininet-wifi> py sta1.wintfs[0].mode
g
mininet-wifi> py sta1.wintfs[0].txpower
14
mininet-wifi> py sta1.wintfs[0].range
62
mininet-wifi> py sta1.wintfs[0].antennaGain
5
```

wintfs[0] means that the information to be obtained comes from the first wireless interface. If the node has multiple interfaces, *wintfs[n]* - e.g. *wintfs[1]* to indicate the second interface and so on - can also be used.

2.5 OVSAP *versus* UserAP

Mininet-WiFi supports two types of access points that differ basically in the location where they are run. *OVSAP* or *OVSKernelAP* runs in the kernel space of the operating system, whereas the *UserAP* is executed in the user space. Additionally, you may prefer one over the other due to possible advantages, such as supported features and performance.

For example, some features may be supported by one and not by another. Until recently, *OVSAP* did not support *meter tables*, a type of table belonging to the OpenFlow protocol that is responsible for Quality of Service (QoS)-related operations, which was included in version 1.3 of this protocol. On the other hand, *UserAP* already supported it by then.

Another important issue is the possibility of running switches or access points in particular *network namespaces*. In this case, *OVS* does not support this feature natively yet, unlike *UserAP*, which supports it. What does that mean? Try using the following command.

```
~/mininet-wifi$ sudo mn --wifi
```

It allows you to view the interfaces of the `ap1` access point.

```
mininet-wifi> ap1 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
↪  DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp2s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel
↪  state DOWN mode DEFAULT group default qlen 1000
    link/ether 84:7b:eb:fc:63:1a brd ff:ff:ff:ff:ff:ff
3: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
↪  UP mode DORMANT group default qlen 1000
    link/ether f8:da:0c:95:12:d3 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
↪  state DOWN mode DEFAULT group default
    link/ether 02:42:04:ed:bc:24 brd ff:ff:ff:ff:ff:ff
5: br-7e51375c6c71: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
↪  noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:6f:43:07:ee brd ff:ff:ff:ff:ff:ff
6: hwsim0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode
↪  DEFAULT group default qlen 1000
    link/ieee802.11/radiotap 12:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
9: ap1-wlan1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc tbf master
↪  ovs-system state UP mode DEFAULT group default qlen 1000
    link/ether 02:00:00:00:02:00 brd ff:ff:ff:ff:ff:ff
10: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode
↪  DEFAULT group default qlen 1000
    link/ether ee:99:70:bb:39:89 brd ff:ff:ff:ff:ff:ff
11: ap1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
↪  group default qlen 1000
    link/ether 0a:94:5d:2c:8b:40 brd ff:ff:ff:ff:ff:ff
```

Note that a large number of network interfaces can be viewed, including those that, in practice, do not integrate the `ap1` access point, such as the wireless and wired interfaces of the computer running Mininet-WiFi. This is the behavior observed when *OVS*, the default type of switch or access point for Mininet-WiFi, is being used.

Now, let us look at how *UserAP* behaves. To do so, run the following command.

```
~/mininet-wifi$ sudo mn --wifi --ap user --innamespace
```

The *–innamespace* parameter was not used with OVS because it does not support this command yet. *–innamespace* is responsible for making the node run in its own *network namespace*, instead of the root network namespace.

After that, check the interfaces of the ap1 access point.

```
mininet-wifi> ap1 ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group
↪  default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ap1-eth0@if46: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
↪  state UP mode DEFAULT group default qlen 1000
    link/ether de:93:af:2c:68:a0 brd ff:ff:ff:ff:ff:ff link-netnsid 0
45: ap1-wlan1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc tbf state
↪  UP mode DEFAULT group default qlen 1000
    link/ether 02:00:00:00:02:00 brd ff:ff:ff:ff:ff:ff
```

As you can realize, the number of network interfaces has dropped considerably. The loopback interface was expected to appear among the results, in addition to the ap1-wlan1 interface, which is the wireless interface of the ap1 access point. The only interface that could be considered as unexpected would be the ap1-eth0 interface, which is the interface used to connect to the SDN controller.

Another relevant issue between *OVSAP* and *UserAP* is the performance. *UserAP*'s performance has significantly worsened since the releases of the newer versions of the Linux kernel. The reason? We confess not to have an answer to this question. However, we invite you to check this issue in practice.

The following command will run Mininet with *OVS* and test the throughput between nodes h1 and h2.

```
~/mininet-wifi$ sudo mn --test iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['40.1 Gbits/sec', '40.0 Gbits/sec']
```

The following command runs Mininet with *UserSwitch* and measures the throughput between the same nodes, h1 and h2.

```
~/mininet-wifi$ sudo mn --switch=user --test iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['171 Mbits/sec', '172 Mbits/sec']
```

If you are not able to execute the previous command, you need to start an SDN controller on another terminal and replace `--test iperf` by `--controller=remote`. Then, after starting the controller, run *iperf* on the Mininet-WiFi CLI as follows: `iperf h1 h2`.

You may be wondering: why *UserSwitch*? *UserAP* has been extended from Mininet's *UserSwitch*. So in practice, they are the same switch or access point. But back to the result, did you notice the difference between them? *UserSwitch* has much lower performance compared to OVS.

With respect to *UserAP* still, an example of its implementation is *Basic OpenFlow Software Switch (BOFUSS)*, a successor of *ofsoftswitch13*[2]. It has been employed in several studies and you may definitely want to use it at some point. *BOFUSS* promises to eliminate most performance-related issues.

To install *BOFUSS*, just run `sudo util/install.sh -3f` from Mininet-WiFi's root directory.

2.6 Graphical User Interface (GUI)

For those who do not know *Python* or are new to Mininet-WiFi, currently there are two options for creating scripts in *Python* with the support of graphic interfaces: by using *Visual Network Descriptor (VND)* or *MiniEdit*.

2.6.1 Visual Network Descriptor

Requirement(s): web server, php, flash player, visual network descriptor

Visual Network Descriptor, or simply VND, is a tool created for a master's work that is able to generate *Python* scripts for Mininet-WiFi via a web browser. Written predominantly in the *Flex* programming language, VND also includes

[2] `https://github.com/CPqD/ofsoftswitch13`

some instructions in PHP and XML.

Using VND is relatively simple. First you need to make a clone of its source code, which can be downloaded at `https://github.com/ramonfontes/vnd`, and follow the installation steps available on the source code page. In general terms, you must have a web server, PHP and Flash Player installed. Then just access it through your preferred web browser. If all goes well, a screen similar to the one shown in Figure 2.5 should appear.

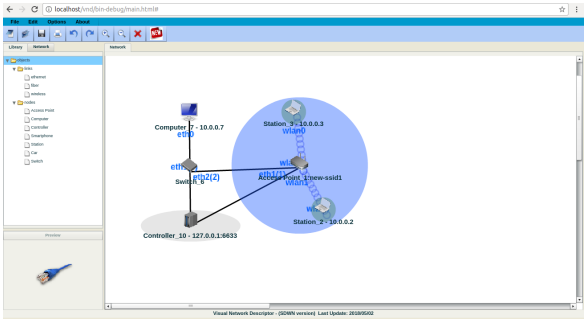

Figure 2.5: Visual Network Descriptor.

With VND open, you can use the mouse cursor to select the nodes that you want to include in the topology and their respective connections. You can also create settings for nodes and save the topology for later use. To generate scripts for Mininet-WiFi, just follow the *File->Export->Export to Mininet-WiFi* menu. A standard file with a .sh extension will be created; it consists of *Python* statements and can be executed as if it were a *Python* file.

For example, a script named <*mytopology.sh*> can be executed as follows.

```
~/mininet-wifi$ sudo python mytopology.sh
```

Visual Network Descriptor:
`https://youtu.be/KsoRMnDP_PA`

2.6.2 MiniEdit

Requirement(s): scripts only

Another alternative for creating topologies with graphics support is *MiniEdit*. Written in *Python*, *MiniEdit* was initially developed for Mininet and has been constantly upgraded to work with Mininet-WiFi. The goal of *MiniEdit*'s developers is to make all the features supported by Mininet-WiFi available on *MiniEdit*.

MiniEdit has a fairly simple user interface that features a screen with a line of tool icons on the left side of the window and a menu bar at the top. It comes already included in the Mininet-WiFi source code.

To use it, just run <*examples/miniedit.py*>.

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

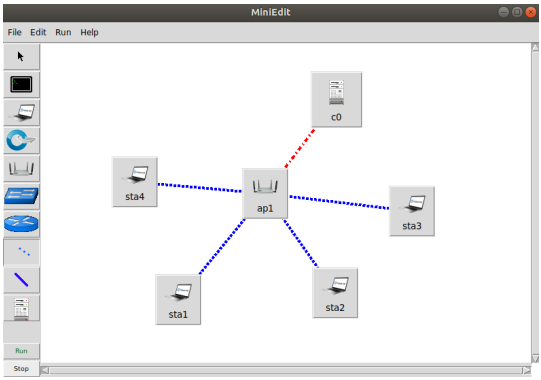

Figure 2.6: Miniedit.

After you run it, a screen similar to the one shown in Figure 2.6 should appear. With it you can add nodes supported by Mininet-WiFi and their respective settings, in addition to their links, of course. In the current version of *MiniEdit*, different types of scenarios are already supported, for example: *adhoc* and *mesh* networks, *WiFi-Direct*, Radius protocol, WPA, among other environments.

You might ask: what is the best alternative to work with GUI? *MiniEdit* or VND? You may want to use *MiniEdit*, since it is a part of Mininet-WiFi. There is also a tendency for VND to be gradually discontinued.

MiniEdit and Mininet-WiFi:
`https://youtu.be/j4JS4xxCrCA`

2.6.3 Viewing 2D and 3D graphics

Viewing topologies by means of graphics is another feature that can be used in Mininet-WiFi. You can generate both 2D and 3D graphics. Nevertheless, there are situations where 3D graphics are very useful, such as surveys involving drones and satellites, since the representation of different levels of altitudes may be necessary.

Thus, given its importance, we will then understand how it is possible to generate 2D and 3D graphics on Mininet-WiFi. Initially we will learn to create the two types of graphics (2D and 3D) using the CLI.

The command below will generate a 2D topology.

```
~/mininet-wifi$ sudo mn --wifi --plot --position
```

While the following command will generate a 3D topology.

```
~/mininet-wifi$ sudo mn --wifi --plot3d --position
```

All scripts available in the *</examples>* directory - a Mininet-WiFi directory where you can find a wide variety of ready-to-run scripts - generate 2D graphics. If you choose to generate 3D graphics, simply make a small change in the code.

For example, let us take as an example *<position.py>*, available in the *</examples>* directory. This file contains the following content:

```
net.plotGraph(max_x=100, max_y=100)
```

As can be seen, only the *x* and *y* axes were defined and the resulting graph will be somewhat similar to the one shown in Figure 2.7. Therefore, to generate 3D graphics, simply add the *z* axis, which will produce something similar to what was shown in Figure 2.8.

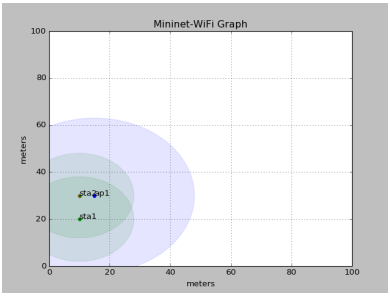

Figure 2.7: 2D Graphic.

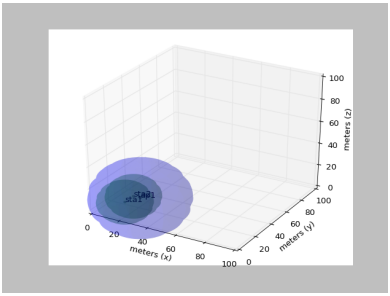

Figure 2.8: 3D Graphic.

```
net.plotGraph(max_x=100, max_y=100, max_z=100)
```

Optionally, minimum values for *x*, *y* and *z* axes can also be defined.

```
net.plotGraph(min_x=10, min_y=10, min_z=10, max_x=100, max_y=100,
↪  max_z=100)
```

Graphics generated by Mininet-WiFi are supported by *matplotlib*, a data visualization library available in the *Python* programming language.

Building 3D Graphic:
`https://youtu.be/lMkIV0YBTss`

2.7 Wireless network emulation

Wireless media emulation in Mininet-WiFi can be done in two ways: with TC[3] or Wmediumd[4]. Let us then understand how to use them and what are the differences between them.

2.7.1 TC (*Traffic Control*)

If Mininet-WiFi is running, quit it. Then start it again with the following command:

```
~/mininet-wifi$ sudo mn --wifi --position
```

Now, view the TC information on the Mininet-WiFi CLI:

```
mininet-wifi> ap1 tc qdisc
qdisc noqueue 0: dev lo root refcnt 2
qdisc pfifo_fast 0: dev enp2s0 root refcnt 2 bands 3
priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc noqueue 0: dev wlp1s0 root refcnt 2
qdisc noqueue 0: dev docker0 root refcnt 2
qdisc tbf 2: dev ap1-wlan1 root refcnt 5 rate 54Mbit burst 14998b lat 1.0ms
qdisc pfifo 10: dev ap1-wlan1 parent 2:1 limit 1000p
```

The output that interests us is highlighted below:

```
qdisc tbf 2: dev ap1-wlan1 root refcnt 5
rate 54Mbit burst 14998b lat 1.0ms
```

In general terms, this output instructs `ap1` to limit the bandwidth to up to 54 Mbits/s, which represents the maximum value that the IEEE 802.11g standard can nominally support. With this information it is easier to understand how TC works.

Considering that TC values are applied to stations and also that propagation models are supported by Mininet-WiFi, for a distance d between access point and station, there is always a received signal value, which can vary with the chosen propagation model. Then, as the node is moving to a new position, a new value for d is calculated and from this we obtain the received signal and

[3] `https://en.wikipedia.org/wiki/Tc_(Linux)`

[4] `https://github.com/ramonfontes/wmediumd/`

the bandwidth value that will be applied by TC to the wireless interface of the node.

In practice, the value applied by TC to `ap1` does not change; instead, the values applied to the interfaces of the stations change. Thus, in an infrastructure environment we will always have two reference points for the calculation of *d*: the station and the access point. Two stations may be associated with the same access point, and different bandwidth values can be assigned to their interfaces: one for `sta1` and another for `sta2`.

Now let us imagine a wireless ad hoc network with three stations. As a wireless network, we can say that this network consists of a non-infrastructure environment, which means that this topology does not have a central node, i.e. the access point. In this type of network the three stations can associate with each other, where, for example, `sta1` can maintain association with `sta2` and `sta3`. However, they only have one wireless interface, and thus are particularly difficult to control using TC.

What would be the reference point to calculate *d* for `sta1`, `sta2` or `sta3`? Differently from an infrastructure network, where we have two reference points for the calculation of *d* - station and access point -, in a non-infrastructure network this does not occur. Thus, wireless *adhoc* and *mesh*, require the use of *Wmediumd*, which implements an ideal wireless medium simulator for these types of networks.

2.7.2 Wmediumd

The module responsible for virtualizing WiFi network cards on Mininet-WiFi, mac80211_hwsim, uses the same virtual medium for all its wireless nodes. This means that all nodes are internally within reach of each other and can be discovered by scanning, as we have done with *iw* previously. If the wireless interfaces need to be isolated from each other, the use of *Wmediumd* is recommended.

Wmediumd had been developed since 2011[5], but only in 2017 it was inte-

[5] `https://github.com/jlopex/mac80211_hwsim`

grated into Mininet-WiFi thanks to Patrick Große[6], the developer responsible for creating the first *Wmediumd* extension for Mininet-WiFi.

Unlike TC, which limits the bandwidth available to the interface, *Wmediumd* relies on a signal table[7] and manages the isolation of the interfaces in real time as data travels across the network.

2.7.3 TC *versus* Wmediumd in practice

Start Mininet-WiFi with the following command.

```
~/mininet-wifi$ sudo mn --wifi --topo single,3 --position --plot
```

This command will create three stations that will associate themselves with access point `ap1`. The --plot parameter will open a topology graph. More details on this parameter will be seen later.

Now, check the signal strength perceived by `sta1` in relation to the `ap1` access point by running the scan command.

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan
BSS 02:00:00:00:03:00(on sta1-wlan0) -- associated
        TSF: 1536705774286475 usec (17785d, 22:42:54)
        freq: 2412
        beacon interval: 100 TUs
        capability: ESS ShortSlotTime (0x0401)
        signal: -36.00 dBm
        last seen: 0 ms ago
        Information elements from Probe Response frame:
        SSID: my-ssid
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 1
        ERP: Barker_Preamble_Mode
        Extended supported rates: 24.0 36.0 48.0 54.0
        Extended capabilities:
                * Extended Channel Switching
                * Operating Mode Notification
```

Also check the RSSI using the *wintfs* option:

[6] https://github.com/patgrosse

[7] https://github.com/ramonfontes/wmediumd/blob/mininet-wifi/tests/signal_table_ieee80211ax

```
mininet-wifi> py sta1.wintfs[0].rssi
-66.0
```

As you can see, there is a difference in the signals received by the `iw scan` and *wintfs* options. With `iw` the received signal was -36 dBm, whereas using `wintfs` it was -66 dBm. While TC is in use, it will not be possible to get any updated information about the signal strength, or any other data that depends on it, by using network commands, such as *iw*. Perhaps one of the few useful data to verify is whether the station `sta1` is in fact associated with the access point `ap1`.

The *iw link* command can also be used for this, as follows.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
SSID: my-ssid
freq: 2412
RX: 12486 bytes (236 packets)
TX: 805 bytes (9 packets)
signal: -36 dBm
tx bitrate: 18.0 MBit/s

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

Now let us move `sta1` and test the received signal again by performing a new scan.

```
mininet-wifi> py sta1.setPosition('250,250,0')
mininet-wifi> sta1 iw dev sta1-wlan0 scan
BSS 02:00:00:00:03:00(on sta1-wlan0)
        TSF: 1536706071142532 usec (17785d, 22:47:51)
        freq: 2412
        beacon interval: 100 TUs
        capability: ESS ShortSlotTime (0x0401)
        signal: -36.00 dBm
        last seen: 0 ms ago
        Information elements from Probe Response frame:
        SSID: my-ssid
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 1
        ERP: Barker_Preamble_Mode
        Extended supported rates: 24.0 36.0 48.0 54.0
        Extended capabilities:
```

```
                * Extended Channel Switching
                * Operating Mode Notification
```

Surprisingly, the signal remained the same as before, even when changing the position of `sta1`. In fact, `ap1` should not even appear in the scan, as `sta1` is no longer under the signal coverage of access point `ap1`. This shows that the TC really does not perceive the wireless medium, since even when the station is no longer within the signal coverage of the `ap1` access point, it is still capable of seeing it.

Now, let us repeat what we did earlier by using *Wmediumd*.

```
~/mininet-wifi$ sudo mn --wifi --topo single,3 --link=wmediumd --position
↪ --plot
```

Then we perform the scan from `sta1`, change its position and repeat the scan.

```
mininet-wifi> sta1 iw dev sta1-wlan0 scan
BSS 02:00:00:00:03:00(on sta1-wlan0) -- associated
        TSF: 1536709235310507 usec (17785d, 23:40:35)
        freq: 2412
        beacon interval: 100 TUs
        capability: ESS ShortSlotTime (0x0401)
        signal: -67.00 dBm
        last seen: 0 ms ago
        Information elements from Probe Response frame:
        SSID: my-ssid
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 1
        ERP: Barker_Preamble_Mode
        Extended supported rates: 24.0 36.0 48.0 54.0
        Extended capabilities:
                * Extended Channel Switching
                * Operating Mode Notification
mininet-wifi> py sta1.setPosition('250,250,0')
mininet-wifi> sta1 iw dev sta1-wlan0 scan
```

As you can see, the signal strength perceived by `sta1` was initially -67 dBm. However, when it went out of the signal range of access point `ap1`, there was a predicted change in the result. In addition to returning an expected signal value at the first moment, in the second the `ap1` access point could not be reached, since `ap1` was not able to reach `sta1` anymore.

Using *wintfs* with *Wmediumd* is not recommended since some implementations of the latter for the calculation of the received signal were not transferred to Mininet-WiFi. In this case, it is always preferable to use *iw* or *iwconfig*.

2.8 Propagation model

Propagation models are mathematical models typically used by simulators and wireless network emulators to try to mimic the behavior of a wireless medium. In the literature, several propagation models have been proposed in order to support the different features of wireless media, such as varied environment types (indoor and outdoor), signal attenuation, interference, etc.

Mininet-WiFi currently supports the following propagation models: *Friis Propagation Loss Model*, *Log-Distance Propagation Loss Model* (default), *Log-Normal Shadowing Propagation Loss Model*, *International Telecommunication Union (ITU) Propagation Loss Model* and *Two-Ray Ground Propagation Loss Model*.

The correct choice of propagation model makes a big difference. For example, one of the variables used in propagation models is the exponent. The exponent is variable that will instruct the propagation model as to the testing environment, i.e. whether it is an indoor or outdoor environment, or whether it is an interference-free environment or not.

Specifying a propagation model is a simple task. The various Mininet-WiFi sample scripts will certainly support this task, especially <*propagationModel.py*>. In it you can find the function responsible for defining the propagation model and its parameters.

To demonstrate how the propagation model can affect the configuration of the nodes that make up the network, we will execute the following script.

```
~/mininet-wifi$ sudo python examples/propagationModel.py
```

Using the pre-defined propagation model, we can observe that the signal

strength perceived by `sta1` was around -79 dBm.

```
mininet-wifi> py sta1.wintfs[0].rssi
-79.0
```

On the other hand, after configuring the *free space* propagation model, the signal strength increased to approximately -47 dBm. If you did not find the -79 dBm and -47 dBm values, do not worry. What matters is the value obtained after setting up the propagation model. This should be higher than the previously noted values.

The propagation model can be modified as follows:

from:

```
net.setPropagationModel(model="logDistance", exp=4)
```

to:

```
net.setPropagationModel(model="friis")
```

Then, check the RSSI after running the modified script.

```
mininet-wifi> py sta1.wintfs[0].rssi
-47.0
```

Another relevant change you can see is related to the range of the access point. Certainly the new range of access point `ap1` is now much larger than the previously observed one.

<propagationModel.py> does not use *Wmediumd*, so if it is necessary to obtain the signal strength it should always be obtained using the *wintfs* command.

The new signal range value evidences the importance of choosing the correct propagation model for the scenario on which it is necessary to work. The new signal strength, which is higher than the previous one, also shows the behavior of the *free space* propagation model, since *free space* does not take into account any kind of interference or barrier that could attenuate the signal.

It is important to note that in addition to the exponent discussed above, there are other parameters that may be unique or not in relation to each model. You can find more information about the supported models and their parameters on Mininet-WiFi's web page[8].

2.8.1 Providing more realism

Some propagation models have no signal variation over time. This means that if we check the signal strength of a particular node, the perceived signal strength will always be the same. However, as we all know, the wireless medium is not constant and many factors can affect the perceived signal strength.

Therefore, in cases where the variation in signal strength is important and you need to represent what happens in the real world with greater fidelity, it is necessary to set up the *fading_coefficient*, which produces signal attenuation over time.

To check the effect caused by *fading* in practice, let us run the following code.

```
~/mininet-wifi$ sudo python examples/wmediumd_interference.py
```

Now we are able to verify the signal strength variation perceived by a given station through *iw*, as below. Notice that as <*wmediumd_interference.py*> uses *Wmediumd*, the signal strength can be obtained by running either *iw* or *iwconfig*.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
SSID: new-ssid
freq: 5180
RX: 6901 bytes (124 packets)
TX: 712 bytes (8 packets)
signal: -65 dBm
tx bitrate: 12.0 MBit/s

bss flags:        short-slot-time
dtim period:        2
```

[8] http://mininet-wifi.github.io/

```
beacon int:        100

mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
SSID: new-ssid
freq: 5180
RX: 8827 bytes (165 packets)
TX: 800 bytes (9 packets)
signal: -62 dBm
tx bitrate: 12.0 MBit/s

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

As you can see, the signal strength perceived by `sta1` was initially -65 dBm and then switched to -62 dBm later on. This variation is expected to happen whenever the received signal level is checked. This is a variation that occurs in a random fashion while also respecting the interval defined by the *fading* parameter.

Try changing the *fading* value and check the result. The higher the *fading* value, the greater the signal variation.

All propagation models supported by Mininet-WiFi can be found in <*mn_wifi/propagationModels.py*>. Should you want to implement new propagation models, you will need to include them in this file.

2.9 Distance *versus* received signal

In addition to the throughput, the distance variation will also impact the signal strength received from the nodes. Obviously, the more distant the source and destination are, the worse the perceived signal should be. This is due to signal attenuation.

We have already seen in 2.2 how we can visualize the signal strength perceived by a node. Let us use, then, the same command to observe the perceived signal strength from different positions. To do so, run <*wmediumd_interference.py*>.

```
~/mininet-wifi$ sudo python examples/wmediumd_interference.py
```

Then check the received signal.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
    SSID: new-ssid
    freq: 5180
    RX: 9142 bytes (184 packets)
    TX: 88 bytes (2 packets)
    signal: -64 dBm
    tx bitrate: 6.0 MBit/s

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

Now, let us use the `distance` command to view the distance between `sta1` and `ap1`.

```
mininet-wifi> distance sta1 ap1
The distance between sta1 and ap1 is 11.18 meters
```

As you can see, the distance between them is just over 11 meters, and the signal level perceived by `sta1` was -64 dBm. So let us change the position of `sta1` in order to reduce the distance from access point `ap1` and check again the signal strength received by `sta1`.

```
mininet-wifi> py sta1.setPosition('40,40,0')
mininet-wifi> distance sta1 ap1
The distance between sta1 and ap1 is 26.93 meters
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
    SSID: new-ssid
    freq: 5180
    RX: 176746 bytes (4379 packets)
    TX: 1668 bytes (19 packets)
    signal: -79 dBm
    tx bitrate: 18.0 MBit/s

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

We can see that after changing the position, the distance increased and consequently the signal level decreased from -64 dBm to -79 dBm. This may be

a simple and obvious conclusion; however, the steps we have just taken aid greatly in the teaching and learning process.

- Ramon dos Reis Fontes, Mohamed Mahfoudi, Walid Dabbous, Thierry Turletti, Christian Esteve Rothenberg. *How far can we go? Towards Realistic Software-Defined Wireless Networking Experiments*. In The Computer Journal (Special Issue on Software Defined Wireless Networks), 2017.

2.10 Modifying *bitrate*

Bitrate refers to the rate of data transmission supported for a given moment. Wi-Fi devices are able to adjust their Modulation and Coding Scheme according to the received signal level. In practice, the more complex the modulation scheme is, the more bits can be transmitted. In contrast, they also become more sensitive to interference and noise, and hence require a cleaner channel.

We will use *iw* to modify bit rates. To do so, consider using <*wmediumd_interference.py*> one more time.

```
~/mininet-wifi$ sudo python examples/wmediumd_interference.py
```

Next, let us do some simple tests and note the difference in the bandwidth values obtained for different bitrates.

First, run *iperf* without changing the bitrate values.

```
mininet-wifi> iperf sta1 sta2
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['14.3 Mbits/sec', '14.4 Mbits/sec']
```

Then look at the current bitrate. As you can see below, the bitrate value was 54 Mbits/s.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
        SSID: new-ssid
        freq: 5180
        RX: 581186 bytes (7475 packets)
```

```
        TX: 19278284 bytes (12610 packets)
        signal: -64 dBm
        tx bitrate: 54.0 MBit/s

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

Now, change the bitrate and re-measure the bandwidth.

```
mininet-wifi> sta1 iw dev sta1-wlan0 set bitrates legacy-5 6 9
mininet-wifi> iperf sta1 sta2
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['5.87 Mbits/sec', '5.93 Mbits/sec']
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
        SSID: new-ssid
        freq: 5180
        RX: 840551 bytes (12506 packets)
        TX: 23301226 bytes (15251 packets)
        signal: -64 dBm
        tx bitrate: 9.0 MBit/s

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

Note that the bitrate was limited to 9 Mbits/s and the measurement from *iperf* dropped to less than 6 Mbits/s.

Finally, let us make another bitrate change and measure the bandwidth once more.

```
mininet-wifi> sta1 iw dev sta1-wlan0 set bitrates legacy-5 6
mininet-wifi> iperf sta1 sta2
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['4.37 Mbits/sec', '4.44 Mbits/sec']
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:03:00 (on sta1-wlan0)
        SSID: new-ssid
        freq: 5180
        RX: 1044693 bytes (16503 packets)
        TX: 26353234 bytes (17256 packets)
        signal: -64 dBm
        tx bitrate: 6.0 MBit/s

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

Once again the available bandwidth dropped and the *bitrate* was limited to 6 Mbits/s, as defined by the command.

Another interesting test is to verify the difference in the transfer rate supported by different Wi-Fi standards. For example, since the script is configured to operate on the IEEE 802.11a standard, which supports up to 54 Mbits/s, it was possible to get the 14 Mbits/s acquired in the previous test. On the other hand, another standard, IEEE 802.11b, would support only up to 11 Mbits/s.

Let us carry out a simple test: change the operating mode of the script from *mode='a'* to *mode='b'*, and change the channel from 36 to 1. Then run *iperf* one more time and observe the result.

Due to a lack of knowledge, many users end up making mistakes when they configure the channel on an access point. What happens is that, for example, channel one does not work at 5 GHz, the frequency used in the IEEE 802.11a standard. You cannot, thus, use channel strips that are not compatible with certain 802.11 standards. The document available on *hostapd*[9] can serve as a good reference point to identify the correct channels for certain operating standards.

```
mininet-wifi> iperf sta1 sta2
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['4.50 Mbits/sec', '4.57 Mbits/sec']
```

As you can see, the measured bandwidth was 4.5 Mbits/s, which is limited to 11 Mbits/s, exactly as defined by the IEEE 802.11b standard.

2.11 Distance *versus* throughput

Throughput is the ability to transmit data from one network point to another over a period of time, determining the speed at which data travels through a link. In wireless networks, throughput is, in theory, highly impacted by the distance between two nodes.

[9] `https://w1.fi/cgit/hostap/plain/hostapd/hostapd.conf`

Among the tools for measuring throughput, *iperf* is certainly the one that stands out the most, as it is the preferred tool in most cases. Throughput measurement using *iperf* is relatively simple, since two nodes executing it are enough for it to function, with one of the nodes operating as client and the other as server.

In order to run *iperf* to verify the relation between distance and bandwidth, we will use the <*position.py*> file as a basis.

```
~/mininet-wifi$ sudo python examples/position.py
```

After running it, you will see a topology with two stations and one access point.

Since the script file has been successfully executed, let us measure the throughput between `sta1` and `sta2` according to their initial arrangement.

```
mininet-wifi> iperf sta1 sta2
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['8.42 Mbits/sec', '9.04 Mbits/sec']
```

Keep a record of the result observed in this test round. *The result may suffer slight variations*.

Now, we will change the positions of `sta1` and `sta2` so that they are further away from access point `ap1`. Then we will measure the throughput between `sta1` and `sta2` again.

```
mininet-wifi> py sta1.setPosition('40,90,0')
mininet-wifi> py sta2.setPosition('60,10,0')
mininet-wifi> iperf sta1 sta2
*** Iperf: testing TCP bandwidth between sta1 and sta2
*** Results: ['6.98 Mbits/sec', '7.14 Mbits/sec']
```

Comparing this new result with the previous one, it is clear that the more distant the stations are from the access point, the smaller the throughput tends to be.

In Mininet-WiFi, the `iperf sta1 sta2` command automatically defines `sta1` as a server and `sta2` as a client. Later on we will see examples of the most common way of using *iperf*.

Publications that have already used Mininet-WiFi for performance research:

- Gilani S.M.M., Heang H.M., Hong T., Zhao G., Xu C. *OpenFlow-Based Load Balancing in WLAN: Throughput Analysis*. Communications, Signal Processing, and Systems (CSPS), 2016.
- Krishna Vijay Singh, Sakshi Gupta, Saurabh Verma, Mayank Pandey. *Improving performance of TCP for wireless network using SDN*. Proceedings of ICDCN, 2019.

2.12 Mobility models

Mobility models are also very important because they try to mimic the mobility of persons, vehicles or any other object that is mobile, i.e. able to move from one point to another. There are several studies that try to identify the patterns of human mobility during natural disasters, such as major storms, floods, etc.

As for the propagation models, there are several mobility models that are accepted by the scientific community worldwide, and some of them are supported by Mininet-WiFi, such as *Random Direction*, *Random Walk*, *Gauss Markov*, among other models. Mobility can be observed either using CLI or a graph, as illustrated in Figure 2.9.

Because of their importance, we will now check how mobility models can be configured on Mininet-WiFi. To do so, let us use the <*mobilityModel.py*> script, which contains the *Random Direction* mobility model in its code. It is important to note that for each mobility model there may be unique parameters such as minimum and maximum speed limits, areas where nodes can move, etc. All the information you need about mobility model settings can be found on Mininet-WiFi's web page[10].

Seed is one of the most important mobility model settings. It modifies mobility significantly. For instance, if a *seed* number one causes the nodes to move from certain x and y values, a seed number two will change the initial values of x and y. It will not be able to change the mobility behavior, but it may

[10] `http://mininet-wifi.github.io/`

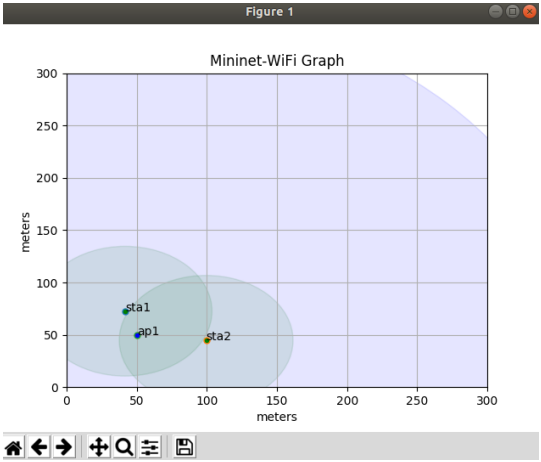

Figure 2.9: Working with mobility.

change its own initial positions. This is an important feature because the initial arrangement of the nodes may not always meet the requirements defined for a particular experiment.

Now that we know a little more about the theory of mobility models, let us run the following script and observe how the nodes behave when configured with the *Random Direction* mobility model.

```
~/mininet-wifi$ sudo python examples/mobilityModel.py
```

Record the behavior of the nodes and try to change the mobility model at a later time for comparison purposes.

Then we will test two of the three implemented Mininet-WiFi commands that were presented in the 2.2 section: `stop` and `start`.

Let us first try the `stop` command.

```
mininet-wifi> stop
```

Should everything go as expected, the `stop` command will cease mobility, causing the nodes to stop moving. This feature is useful in cases where the

user wishes to observe information such as signal strength or even available bandwidth connected to an arrangement of nodes.

Now, we can issue the `start` command to resume mobility.

```
mininet-wifi> start
```

All mobility models supported by Mininet-WiFi can be found in *<mn_wifi/mobility.py>*. Should you want to implement new mobility models, you must include them in this file.

Studies that previously used Mininet-WiFi for research on mobility:

- K. V. K. Singh, M. Pandey. *Software-defined mobility in IP based Wi-Fi networks: Design proposal and future directions*. IEEE ANTS, 2016
- D. Tu, Z. Zhao and H. Zhang. *ISD-WiFi: An intelligent SDN based solution for enterprise WLANs*. WCSP, 2016,
- A. Kaul, L. Xue, K. Obraczka, M. Santos, T. Turletti. WiMobtitHandover and Load Balancing for Distributed Network Control: Applications in ITS Message Dissemination. ICCCN, 2018.
- Z. Han, T. Lei, Z. Lu, X. Wen, W. Zheng, L. Guo. *Artificial Intelligence Based Handoff Management for Dense WLANs: A Deep Reinforcement Learning Approach*. IEEE Access, 2019.