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

Conference Paper · January 2016

DOI: 10.1145/2934872.2959070

2 authors:

Ramon Fontes
University of Campinas
7 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Christian Esteve Rothenberg
University of Campinas
78 PUBLICATIONS **1,292** CITATIONS
SEE PROFILE

Mininet-WiFi: Emulating Software-Defined Wireless Networks

Ramon R. Fontes, Samira Afzal, Samuel H. B. Brito, Mateus A. S. Santos, Christian Esteve Rothenberg
School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, Sao Paulo, Brazil
Email:{ramonrf,samira,shbbrito,msantos,chesteve}@dca.fee.unicamp.br

***Abstract*—As the density of wireless networks continues to grow with more clients, more base stations, and more traffic, designing cost-effective wireless solutions with efficient resource usage and ease to manage is an increasing challenging task due to the overall system complexity. A number of vendors offer scalable and high-performance wireless networks but at a high cost and commonly as a single-vendor solution, limiting the ability to innovate after roll-out. Recent Software-Defined Networking (SDN) approaches propose new means for network virtualization and programmability advancing the way networks can be designed and operated, including user-defined features and customized behaviour even at run-time. However, means for rapid prototyping and experimental evaluation of SDN for wireless environments are not yet available. This paper introduces Mininet-WiFi as a tool to emulate wireless OpenFlow/SDN scenarios allowing high-fidelity experiments that replicate real networking environments. Mininet-WiFi augments the well-known Mininet emulator with virtual wireless stations and access points while keeping the original SDN capabilities and the lightweight virtualization software architecture. We elaborate on the potential applications of Mininet-Wifi and discuss the benefits and current limitations. Two use cases based on IEEE 802.11 demonstrate available functionality in our open source developments.**

***Keywords*—Wireless networks, Emulation, SDN, OpenFlow.**

I. Introduction

Network emulation [1] is a widely used technique to evaluate performance, test and debug protocols as well as support multiple network-related research issues. At a fraction of the cost of real testbeds and different than simulations, emulation allows running real code in realistic networking and computing conditions. In support of research on Software-Defined Networking (SDN) [2] architectures, the Mininet emulator [3] allows rich experiments and fast prototyping cycles, which are especially interesting for teaching, research and reproducibility purposes in academia as well as pre-deployment evaluation of "exactly" the same SDN control software to be later used in production (e.g., BSN Labs,[1] Google B4 [4]).

When it comes to the experimentation of SDN and the OpenFlow protocol [5] in wireless networks, only a few alternatives are available based on popular simulators (e.g., NS-3[2] and OMNet++ [6] engines). In order to validate the concepts and experimentally evaluate the benefits of Software-Defined Wireless Networks (SDWN) [7], a suitable emulation tool is needed. Likewise to the wired SDN experiences, a wireless SDN emulator tool becomes fundamental to prototype, test and benchmark new protocols, end-to-end network architectures and applications.

In order to fill this gap, this paper presents Mininet-WiFi, a fork of Mininet extended to support WiFi by adding virtualized WiFi Stations (STAs) and Access Points (APs) based on the most common Linux wireless device driver, namely mac80211/SoftMac[3]. The majority of Linux wireless drivers today use mac80211/SoftMac, which supports most of the features provided by wireless NICs and allows Mininet-WiFi to exercise fine-grained control over wireless network packets at a low layer. Mininet-Wifi is being developed as a clean extension of the high-fidelity Mininet emulator by adding the new abstractions and classes to support wireless NICs and emulated links while conserving all native lightweight virtualization and OpenFlow/SDN features. As a proof of concept of our initial round of SDWN experiments with Mininet-Wifi, this paper describes (1) a use case on OpenFlow-based bicasting over two 802.11 APs, (2) a use case featuring the integration with physical wireless NICs, and (3) Mobility.

The remainder of this paper is organized as follows. Section 2 provides background on SDWN and introduces relevant concepts to understand the applicability and relevance of our work. Section 3 describes the system architecture of Mininet-WiFi. Section 4 describes some case studies. Section 5 discusses current limitations and the ongoing/future work. Section 6 describes related work, and, finally, Section 7 concludes the paper with some final thoughts and motivates the reader to follow the developments in the open source code repository.

II. Software-Defined Wireless Networking

Software-Defined Networking (SDN) [2] is a recent networking paradigm based on a programmatic separation of the control plane (aka. Network OS) from the data plane (aka. forwarding plane). SDN allows network administrators to specify the behavior of the network in a logically centralized manner via the controller platform that leverages southbound interfaces to the forwarding devices –the OpenFlow protocol being the most popular one. In the same spirit, Software-Defined Wireless Networks (SDWN) [7] aims at providing programmatic centralized control of the network outside the wireless boxes (APs) which enforce the received instructions

[1] http://www.bigswitch.com/press-releases/2015/02/19/big-switchs-bsn-labs-now-available-for-free-hands-on-experience-with-sdn

[2] https://www.nsnam.org/docs/models/html/wifi.html

[3] http://linuxwireless.org/en/developers/Documentation/mac80211

978-3-901882-77-7

(policy decisions) and remain responsible for the transmission and reception of the traffic over the wireless link.

Separation of both control and data plane has existed in the wireless domain prior to SDN and OpenFlow. IETF standardized the Control and Provisioning of Wireless Access Points (CAPWAP) protocol [8] several years ago, which centralizes the control in wireless networks, allowing ACs (access controllers) to manage WTPs (wireless termination points) over a wireless network. Besides, there are multiple proprietary solutions (e.g., Aerohive, Aruba, Cisco HDX, Meraki, Ruckus) based on external controllers responsible for the management of the APs. These commercial solutions introduce a number of extensions to standardized protocols or define their own APIs between the controller and APs, and present differences in the refactoring of control and data plane functions in addition to a series of proprietary radio resource enhancements. While all these solutions have proven to work well at scale, their cost is often prohibitive for many deployments and raise concerns due to their closely integrated nature and the consequent vendor lock-in and inability for in-house or third-party innovations.

SDWN has become an emerging and significant research branch of SDN [9]–[11], including increased attention of mobile network operators [12], [13] and the identified synergies to Network Function Virtualisation (NFV) [14].

In its current form, OpenFlow does not address specific needs of WiFi protocols and networks, which include interference mitigation, mobility management, and channel selection techniques [15]. Ongoing work at the Open Networking Foundation (ONF) Wireless & Mobile Working Group (WMWG)[4] may change the picture in the near future.

As today, the only realistic way to try out WiFi and OpenFlow together is using open source firmware and OS solutions like OpenWRT that allow turning commodity wireless routers into OpenFlow-enabled switches. However, even OpenWRT is constrained to a limited set of resources, and, like any real testbed, this approach is subject to challenges on the scale of the experiments, the control and reproducibility options as well as the high setup times.

Wireless SDN emulators, on the other hand, would be an interesting option to work with multiple devices (both APs and STAs) at reasonable scale on experimenter-defined environments, allowing research on new SDWN features in addition to the fast prototyping and rich experimentation benefits known by the large Mininet user community.

We envision that the Mininet-WiFi emulator has the potential to become an important tool for wireless SDN research by enabling real-world wireless network systems and (Linux-based) end-user device software in a fully controlled environment yielding high fidelity results in support of SDWN research.

III. Mininet-WiFi Design and Workflow

Delivering software-based wireless network emulation usually means to completely abandon wireless network hardware and performing the emulation on a per-packet level. The common ground across wireless network emulation approaches is that the wireless channel is replaced by an artificial model providing a convenient way to evaluate the wireless channel effects.

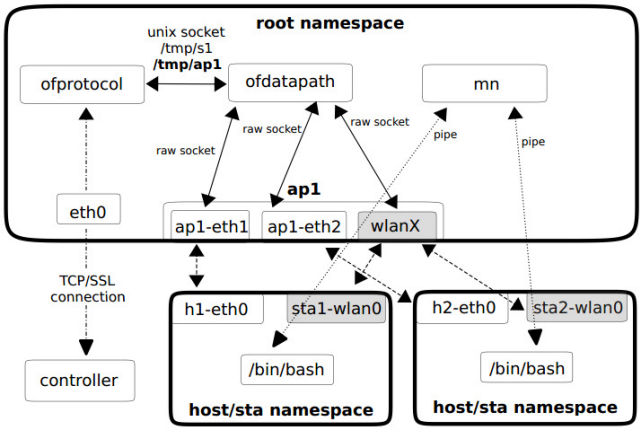


Fig. 1. Components and connections in a two-host network created with Mininet-WiFi.

Currently Mininet-WiFi emulator uses basic Linux TC tools to emulate the wireless channel by setting link parameters such as packet loss, delay, and channel bandwidth. A more elaborated wireless model support is under ongoing work and left out of scope of this paper, which focuses on the underpinning software design and a set of strawman applications.

A. Software Architecture and Implementation

Figure 1 depicts the components and connections in a simple topology with two hosts created with Mininet-WiFi, where the newly implemented components (highlighted in gray) are presented along the original Mininet building blocks.

More specifically, we added WiFi interfaces on STAs that now are able to connect to an AP through its (`wlanX`) interface that is bridged to an OpenFlow switch with AP capabilities represented by (`ap1`). Similar to Mininet, the virtual network is created by placing host processes in Linux OS network namespaces interconnected through virtual Ethernet (veth) pairs. The wireless interfaces to virtualize WiFi devices work on *master* mode for APs and *managed* mode for STA.

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

Access Points: Are devices that manage associated stations. Virtualized through `hostapd`[5] daemon and use virtual wireless interfaces for access point and authentication servers. While virtualized APs do not have (yet) APIs allowing users to configure several parameters in the same fashion of a real one, the current implementation covers the most important features, for example ssid, channel, mode, password, cryptography, etc.

[4]https://www.opennetworking.org/images/stories/downloads/working-groups/charter-wireless-mobile.pdf

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

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

B. Creating a Network.

To start Mininet-WiFi a simple command suffices (*sudo mn --wifi*) to bring up two Stations (`sta1` and `sta2`) connected to an Access Point (`ap1`) configured with *"my-ssid"* as the default SSID of the wireless network. Additional parameters can be provided, including *--ssid*, *--channel* and *--mode* for customized setups. In addition to the CLI options, it is possible to build topologies based on some sample files that are in the examples directory, such as: (*i*) `adhoc` for experiments with adhoc mode, (*ii*) `simplewifitopology` to create exactly the same topology mentioned above with 2 STAs and 1 AP; (*iii*) `wifiStationsAndHosts` creates a topology with stations and hosts enabling communication between them; and (*iv*) `2AccessPoints` to create a topology with STAs associated to different APs and allowing communication among all STAs. In addition, (*v*) `wifiMobility` and (*vi*) `wifiMobilityModel` provide sample mobility scenarios based on different well-known models (e.g., *GaussMarkov*, *RandomDirection*, *RandomWalk*, *RandomWaypoint* and *TruncatedLevyWalk*). The aforementioned examples (and further provided by the community) are important facilitators to test and share new topologies, that can be reused between different experiments and among research groups.

GUI. In addition to the CLI (Fig. 2(a)), another option to work with Mininet-WiFi is using the complementary GUI tool (Fig. 2(b)) called Visual Network Description [16].

C. Customizing the network & User interaction.

Besides all OpenvSwitch (OVS) related commands, Mininet-WiFi supports commands related to `iw`[6]: ***sta1 iw dev sta1-wlan0 scan*** to scan for available access points, and ***sta1 iw dev sta1-wlan0 connect my-ssid*** to connect to the access point which has the same SSID. Being a runtime emulator Mininet-WiFi allows the user to add new Stations to the topology on demand, however, there is one important factor to be considered: the command *--radios* needs to be used besides the *--wifi* command (e.g., ***mn --wifi --radios=5***), since the system needs to know how many radios (stations and access points) should be handled before starting the topology. These commands are relevant in case of experiments with dynamic topologies that add/remove stations to the network topology.

To verify the connectivity between stations and access points on Mininet-WiFi the user can type the following CLI command: ***sta1 ping sta2***. Common Linux commands can be executed by the user at runtime, for instance to verify the available bandwidth between two stations between stations *sta1* and *sta2*: ***sta1 iperf -s*** & ***sta2 iperf -c 10.0.0.1***.

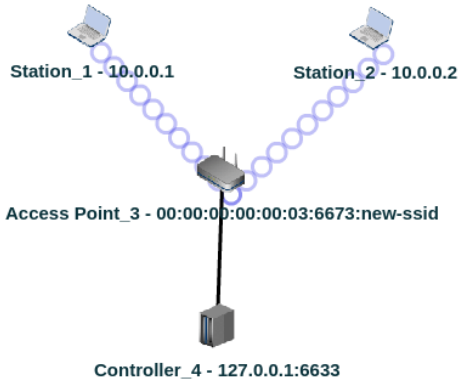
In addition to the default Mininet `addHost` and `addSwitch` classes, Mininet-Wifi provides `addStation` and `addBaseStation` classes, and a modified `addLink` class to define the wireless environment:

addStation("sta1")
addBaseStation("ap1", ssid="new_ssid", channel="10", mode="g")
addLink(ap1,sta1)

The `addStation` class allows the user to add a new station to the topology and the `addBaseStation` class allows to customize the device with SSID, channel and mode configuration. The class name `addBaseStation` has chosen on purpose due to the intention of extending Mininet-WiFi to further mobile wireless technologies in the future. Finally, the `addLink` class instantiates the association between STAs and APs, which in the current implementation is based on user-defined static configuration of Linux TC to control the wireless channel properties (e.g., bw='54Mbps', loss='0.1%', delay='15ms').

```
alpha@alpha-Inspiron-5547:~$ sudo mn --wifi --ssid=new-ssid
*** Enabling Wireless Module
*** Creating network
*** Adding controller
*** Adding Station(s):
sta1 sta2
*** Adding Access Point(s):
ap1
*** Associating Station(s):
(sta1, ap1) (sta2, ap1)
*** Starting controller(s)
c0
*** Starting 1 Access Point(s)
ap1 ...
*** Starting CLI:
mininet-wifi>
```

(a) Mininet-WiFi CLI.



(b) Topology on Visual Network Description.

Fig. 2. Alternatives to work with Mininet-WiFi.

D. System Profiling

An important aspect of any emulation tool is its performance and ability to scale. In general, Mininet-WiFi inherits the same performance and scalability properties of Mininet, but it also adds new features and there are other metrics that need to be considered for system profiling purposes. We measured the time required for some management operations of virtual WiFi interfaces in the Linux kernel (3.19.10-18), `iw` utility, and `hostapd` startup/shutdown operations. Table 1 shows the time consumed by individual operations when building a simple topology with two STAs and one AP.

[6]A tool for managing and configuring wireless devices and an alternative for iwconfig

TABLE I. TIME FOR BASIC MININET-WIFI OPERATIONS IN A BUILDING A SIMPLE TOPOLOGY WITH TWO STAS AND ONE AP.

Operation	Time(ms)
Create an AP	17
Create a STA	63
Association between two nodes	10
Starting mac80211 module	5
Stopping STAs and APs	350

IV. CASE STUDIES

This section illustrates functionalities of Mininet-WiFi by means of three use cases, firstly on bicasting over wireless networks, secondly on integrating physical wireless interfaces, and the last one on mobility. In the context of SDWN, we expect Mininet-WiFi delivering significant and complementary advantages compared to simulation or testbed-based experimental approaches. The project repository includes links to a VM Disk and a Docker Image along user instructions to reproduce the use cases.

A. Use Case 1: Wireless Bicasting

In order to illustrate the potential of Mininet-WiFi to carry and reproduce experiments from the literature, this use cases demonstrates a bicasting over wireless scenario inspired by the work in [17]. The real experiment consisted of a video streaming application running in a mobile station and receiving the packet flow over multiple radios simultaneously (n-casting) – the mobile station, equipped with two WiFi and one WiMax interface, was attached to multiple APs using the same SSID. OpenFlow was used to duplicate packets in the wired network and re-write the L2/L3 headers at the radio access points. As a result, the quality of experience was improved, since the packet loss over a radio link was compensated by the duplicated stream received over the alternative radio(s).

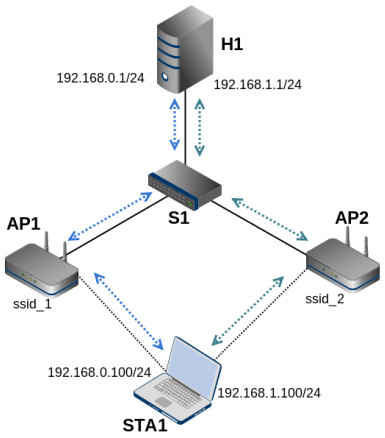
In the scenario shown in Fig 3(a), we use Iperf to measure the bandwidth between STA1 and H1 during 60s. STA1 has two wireless interfaces and both interfaces are connected to different APs (AP1 and AP2), which are connected to an OpenFlow controller as well as the switch S1. OpenFlow rules ensure that packets are copied and sent through the different paths to the stations.

During the first 20s the measured bandwidth is about 90Mbps, which seems coherent with the 54Mbps limit of each interface in mode g. When the time reaches 20s one wireless interface is disconnected (from AP2) causing a traffic decrease, but increase again when the same wireless interface is connected to AP2 at 40s (Fig. 3(b)).

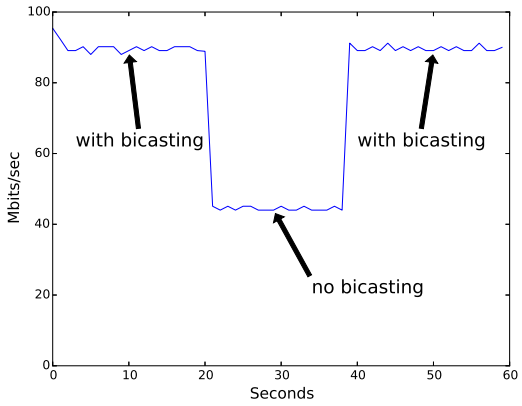
While the experiment is a strawman effort that oversimplifies properties from the wireless channels, it serves as a proof of concept of the feasibility and flexibility of Mininet-WiFi in enabling multiple simultaneous WiFi connections and traffic control through OpenFlow.

B. Use Case 2: Integration with Physical Wireless Interface

In this use case, we leverage a USB wireless card connected to a PC running Mininet-WiFi (Fig. 4) to illustrate how physical wireless interfaces can be integrated into Mininet-WiFi.



(a) Sample Topology.



(b) Bandwidth Measurement between STA1 and H1.

Fig. 3. Bicasting over WiFi.

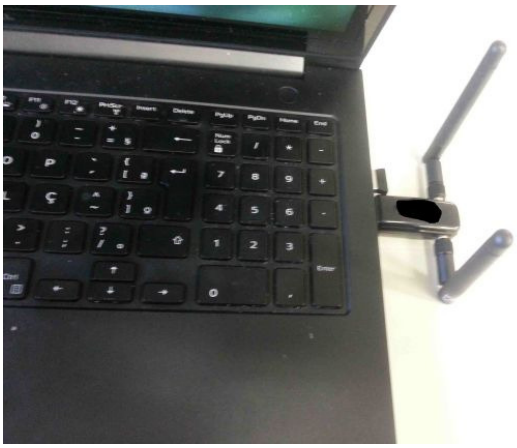


Fig. 4. Physical wireless NIC integrated into Mininet-WiFi.

The process of turning a physical wireless interface to behave like an AP can be done by different means and this experiment uses `hostapd`. The following lines are used to change the wireless interface to behave like an AP:

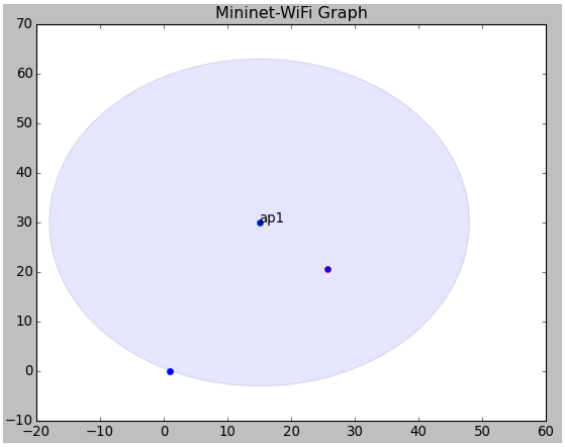
interface=wlan1

ssid=Mininet-WiFi-AP
hw_mode=g
channel=7

This code snippet is stored in a simple *txt* file and executed in a terminal. After running the code, a mobile device can be connected to this AP and a single Mininet-Wifi topology is instantiated with the command *mn –wifi*. Then, a bridge is created (*ovs-vsctl add-port ap1 wlan1*) to allow the interface `wlan1` (usb wireless physical interface) communicate with the virtualized Mininet-WiFi environment. Finally, virtual STAs (running inside Mininet-WiFi environment) and the physical mobile device can communicate with each other, as can be seen in the companion video material.[7]

C. Use Case 3: Mobility

Basic mobility support allows users to configure parameters such as time and also initial and final positions to provide mobility. Figure 5(a) illustrates an AP1 and two moving STAs. The signal range of AP1 depends of the mode (e.g., b, g, n), and when STA1 exceeds this limit the association from AP1 is broken. Figure 5(b) shows the increase in response time response as the distance between the STAs and AP1 increases. Please refer to `wifiMobility.py` in order to reproduce this sample use case.



(a) Mininet-WiFi Graph (optional).

```
mininet-wifi> sta1 ping sta2
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=38.0 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=18.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=22.9 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=25.8 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=29.0 ms
From 10.0.0.2 icmp_seq=37 Destination Host Unreachable
```

(b) Connectivity test.

Fig. 5. Mobility within Mininet-WiFi.

V. Limitations and Future Work

Mininet-WiFi inherits any limitations of the lightweight virtualization Mininet architecture and certainly adds a couple of more related to our current approach to emulate the wireless medium with high-fidelity. We are on the process of collecting information to find and evaluate all limitations to be documented in the github repository.

[7]https://www.youtube.com/watch?v=WH6bSOKC7Lk

Our ongoing efforts are devoted to relevant missing features pointed by early users of Mininet-WiFi. Probably the most challenging research and development effort upfront is an abstraction for the wireless broadcast links with good enough fidelity, supporting dynamic reconfiguration of link parameters to consider the effects of interference, signal strength depending on node distances, among others. Also on our roadmap is advancing the mobility support and validate the implemented models. On all these fronts, we intend to leverage as much as possible from related open source work (e.g. NS-3 models) and lessons from the vast literature on wireless research.

A future experimenter-friendly feature we foresee is allowing emulated experiments be based on trace-based approaches [18] towards reproducing the behavior of a real network inside of Mininet-WiFi according to previously captured traces. Challenges upfront include the ability to capture, interpret, and process traces to produce reliable (high-fidelity) results.

On the software-defined front, we intend to move beyond traditional (packets- and flow-based) traffic manipulation and allow the SDWN controller (via OpenFlow extensions or alternative southbound interfaces) to dynamically re-configure Access Points (e.g., Ethanol [19]), exploring alternative offerings to the closed solutions commercialized by the industry.

VI. Related Work

Experimental tools to evaluate new designs are key to support academic and industrial research one new networking technologies, troubleshooting, change planning of deployed networks, and so on. Hence, wireless simulators, emulators, and new testbeds have received a lot of attention. Related work on wireless experimentation environments is presented in Fig. 6 which tries to illustrate the fundamental differences between each type of platform, pointing to emulators as a sweet spot when compared to hardware-based testbeds (fast, accurate, shared, expensive) and simulators (cheap, detailed, often slow and requiring code changes) [20].

When looking into existing emulators such as Estinet [21], Emane[8] and Core [22], the first one is capable to simulate and

[8]http://www.nrl.navy.mil/itd/ncs/products/emane

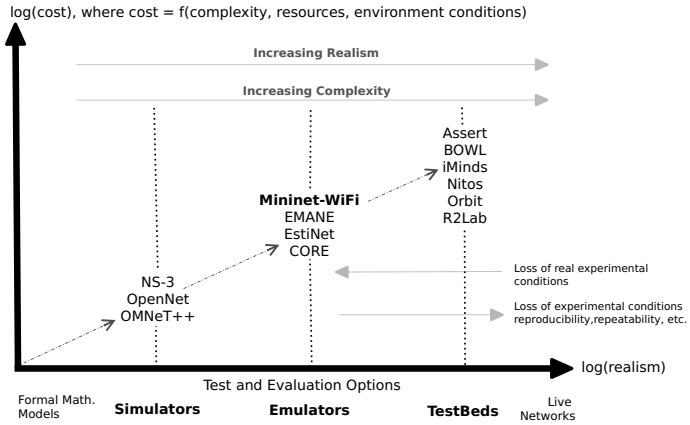


Fig. 6. Overview of related work and trade-offs of different wireless experimental platforms.

emulate wireless networks, the last two focus only on MANET networks, and the first one is a commercial solution and thus does not provide the flexibility of source-code level modification and extensibility. Mininet-WiFi, in turn, is open source code and aims at supporting any type of WiFi networking.

In the popular NS-3 simulator, support of 802.11 includes both MAC and PHY layers simulation models for 802.11a, 802.11b, 802.11g and 802.11n (both 2.4 and 5 GHz bands) networks as well as infrastructure and adhoc scenarios. Closest to our emulation efforts are the NS-3 real-world integration features like real-time/emulation mode and DCE [23]. However, seamless support of unmodified POSIX code is not broadly possible and very specific to the target application and libraries, in addition to real-time constraints along performance and scalability concerns. A related Mininet effort[9] adds wireless link modeling by gluing together NS-3 processes and Linux tap devices. Unfortunately, as far as we can infer from the public repositories, the project seems to be inactive since 2013. OpenNet [24] is another recent attempt to partner Mininet with NS-3 by implementing WiFi scan mechanisms that enable experiments with layer-2 handover of mobile nodes between two OpenFlow-enabled APs operating on two different channels.

VII. Conclusion

The widespread and popularity of wireless networks calls for novel control and management approaches that need to be prototyped and evaluated under realistic topology and networking conditions. Ideally, these controlled and experimenter-friendly environments should be affordable and yield high fidelity results. We believe that Mininet-WiFi is a promising step towards adding wireless research capabilities to the SDWN research toolbox. The results of our first release of Mininet-WiFi are quite encouraging. We believe that the current limitations are solvable in the near future and we will overcome the challenges towards a scalable and high-fidelity wireless emulator. Mininet-WiFi is ready to be tested and allows for OpenFlow and WiFi experiments. There is a growing user community that will hopefully contribute to the developments efforts through the publicly available Mininet-WiFi code repository at: *https://github.com/intrig-unicamp/mininet-wifi*.

Acknowledgements

This project was partially supported by FAPESP grant # 14/18482-4. Mateus A. S. Santos is currently with Ericsson Research.

References

[1] Fall, K. R. Network emulation in the Vint/NS simulator. In 4th IEEE Symposium on Computers and Communication (1999).

[2] D. Kreutz, F. M. V. Ramos, P. E. P. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig "Software-defined networking: A comprehensive survey", Proc. IEEE, vol. 103, no. 1, 2015.

[3] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ser. HotnetsIX. New York, NY, USA: ACM, p 19:1-19:6, 2010.

[9] https://github.com/mininet/mininet/wiki/Link-modeling-using-ns-3

[4] Sushant Jain et al. "B4: experience with a globally-deployed software defined wan". Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, August 12-16, 2013, Hong Kong, China.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2):6974, April 2008.

[6] Varga, A. The OMNeT++ Discrete Event Simulation System. In the Proceedings of the European Simulation Multiconference (ESM2001. June 6-9, 2001. Prague, Czech Republic).

[7] S.Costanzo, L.Galluccio, G.Morabito, and S.Palazzo. Software Defined Wireless Networks: Unbridling SDNs. Proc. of EWSDN 2012. Darmstadt, Germany, October 2012.

[8] P. Calhoun, M. Montemurro, D. Stanley "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, March 2009.

[9] Nachikethas A. Jagadeesan, Bhaskar Krishnamachari, Software-Defined Networking Paradigms in Wireless Networks: A Survey, ACM Computing Surveys (CSUR), v.47 n.2, p.1-11, January 2015.

[10] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos. Software-defined and virtualized future mobile and wireless networks: A survey. CoRR, abs/1409.0079, 2014.

[11] S. Shahila, S. Fizza, M. Tahira. "A Survey on Wireless Software Defined Networks", International Journal of Computer and Communication System Engineering (IJCCSE), Vol. 2 (1), 155-159, 2015.

[12] C. Bernardos "An architecture for software defined wireless networking", IEEE Wireless Commun. Mag., vol. 21, no. 3, p 52-61, 2014.

[13] Sama, M.R., Contreras, L.M., Kaippallimalil, J., Akiyoshi, I. et al. "Software-defined control of the virtualized mobile packet core", Communications Magazine, IEEE. vol. 53, no. 2, p 107-115, 2015

[14] Bo Han, Gopalakrishnan, V., Lusheng Ji, Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations., Communications Magazine, IEEE. vol. 53, no. 2, p 90-97, 2015.

[15] Schulz-Zander, J. Suresh, L. Sarrar, N. Feldmann, A. Hhn, T. & Merz, R. Programmatic Orchestration of WiFi Networks, in '2014 USENIX Annual Technical Conference (USENIX ATC 14)' , USENIX Association, Philadelphia, PA, pp. 347-358, 2014.

[16] Fontes, R. R. and Sampaio, P. N. M. Visual Network Description: A Customizable GUI for the Creation of Software Defined Network Simulations. In: EUROSIS - The European Multidisciplinary Society for Modelling and Simulation Technology. Lancaster, UK, pp 149-153, 2013.

[17] K.-K. Yap, T.-Y. Huang, M. Kobayashi, M. Chan, R. Sherwood, G. Parulkar, and N. McKeown. Lossless Handover with n-casting between WiFiWiMAX on OpenRoads. In ACM Mobicom (Demo), 2009.

[18] Brian D. Noble and M. Satyanarayanan and Giao T. Nguyen and Randy H. Katz. "Trace-Based Mobile Network Emulation." In Proceedings of ACM SIGCOMM 97, pp 51-61, September 1997.

[19] M. Henrique, B. Gabriel, V. Marcos, M. Daniel. "Ethanol: Software defined networking for 802.11 Wireless Networks." Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. Ottawa, ON, Canada, pp 388-396, 2015.

[20] M. Imran, A. Said, and H. Hasbullah, "A survey of simulators, emulators and testbeds for wireless sensor networks", in Information Technology (ITSim), International Symposium in, vol. 2, pp 897-902, June 2010.

[21] W. Shie-Yuan, C. Chih-Liang, and Y. Chun-Ming. EstiNet OpenFlow network simulator and emulator. IEEE Communication Magazine, Vol. 51, Issue 9, 2013.

[22] J. Ahrenholz, Comparison of CORE Network Emulation Platforms, Proceedings of IEEE Military Communications Conference Conference 2010 (MILCOM), pp 864-869, November 2010.

[23] Q. Alina, S. Damien, T. Thierry, D. Walid. "Automating ns-3 Experimentation in Multi-Host Scenarios." Wireless and Mobile Computing, Proceedings of the 2015 Workshop on ns-3, New York, NY, USA, 2015.

[24] C., Min-Cheng, C. Chien, H. Jun-Xian, T. Kuo, Y. Li-Hsing and T. Chien-Chao. OpenNet: A simulator for software-defined wireless local area network. In the Proceedings of Wireless Communications and Networking Conference (WCNC), pp 3332-3336, Istanbul, 2014.