A NS-3-based Cloud Wireless Network's Emulator for Undergraduate Teaching

Alejandro Huamantuma, Cesar Santivanez
Advanced Networks Research Lab (GIRA-PUCP)
Pontificia Universidad Católica del Perú
Lima, Perú
{alejandro.huamantuma, csantivanez}@pucp.edu.pe

***Abstract*—The COVID-19 pandemic limited the availability of in-classroom laboratory experiences with physical wireless equipment. Existing simulation tools, while adequate for performance analysis as required for research, fail to provide the real-world feeling and user interface of a physical equipment. Due to this lack of realism, they do not fulfill the educational objectives of undergraduate courses. Indeed, under a competencies-based curricula, it is expected that a student interacts with the equipment, configures it, run her experiments, evaluates the results, decides in any changes, and then implements and validates them. This paper reports our development of a cloud-based wireless network emulator that intends to give the students the same user experience as the one obtained with our testbed of (physical) Wireless Access Points running OpenWRT. We also discuss students' experiences using our emulator on a Wireless Networking undergrad course in the second half of 2020.**

***Index Terms*—Wireless Networks, testbeds, emulation, virtualization, undergraduate teaching.**

I. Introduction

Due to many factors that affect the performance of a networking technology, experimentation (either real or based on simulations) have become a valuable tool for the network engineer. This is even more so in wireless networking, where developing theoretical models that properly tracks the behavior induced by the wireless channel's variability has remained an elusive task. Because of this, significant effort has been put in the development of testbeds researchers and engineers can analyze the performance of the new networking technologies under different environmental conditions.

In general, testbeds can be classified into real, emulated and simulated. Each one with different levels in costs, complexity, realism, flexibility and repeatability among others. Real testbeds are typically more expensive, complex and have higher fidelity than the others but usually they lack flexibility and repeatability (see Table I). Simulators, on the other hand, often lack realism or fidelity. Emulators present better realism, since they make it possible to test the same code/application the end-system will use.

Physical testbeds are expensive due to the high costs for hardware. To alleviate this, institutions have been pooling resources in collaborative testbeds. The NSF's sponsored Global Environment for Network Innovations (GENI) testbed [7] is arguably the better known testbed providing a virtual laboratory extended all across the United States connecting many local testbeds and sharing a lot of compute resources. But in the case of wireless networks, the dynamic nature of the transmission medium complicate the development of wireless testbeds. Some projects associated to GENI also contain wireless nodes in different topologies allowed to be used. For example, Emulab [1] has nodes with 802.11 a/b/g interfaces distributed at various locations in a large building and provide a physical diagram so you can reserve the ones which best fits your necessities. It is easy to infer that one main drawback of this physical testbeds is the absence of control in the channel characteristics and the typologies available limiting its flexibility.

The Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT) [3] is another example of a physical wireless network testbed. It includes 400+ nodes (20 x 20 grid) ready to be used remotely for different researchers through an automated software platform. But in this case, the testbed uses four multi-band antennas located in the four corners of the grid as a noise injection subsystem trying to improve the flexibility of this platform. Even though this testbed may improve the accuracy for specific scenarios, it again lacks flexibility, is relatively complex to maintain and needs high investments for hardware.

A different approach was needed to overcome physical testbeds' drawbacks. A promising one is the emulation of the wireless nodes while simulating the wireless channel. This

TABLE I
Comparison of testbeds

Testbed	Pros	Cons
Emulab [1]	Real wireless devices	Limited flexibility and no control on transmission channel
WiTest [2]	Real wireless devices	Limited flexibility and no control on transmission channel
Orbit [3]	Real wireless devices	Limited flexibility and control on transmission channel
Emane [4]	Complete control of transmission channel and real application	Relatively few models limiting the simulated scenarios
QOMB [5]	Emulated wireless channel over a wired channel	Each node correspond to a real computer
Virtual-mesh [6]	Complete control of transmission channel, topology scalability using VMs	Large topologies require enough hardware resources for virtualization

configuration add flexibility to the system and avoid the costs of implementing real hardware but do not compromise its reliability.

An example of this approach is the QOMB testbed [5] which is capable of recreating a multi-hop wireless network on a wired cluster of around 1000 PCs where each node has a set of libraries that configure a new kernel module.

As dedicated physical nodes are expensive and complex to maintain, virtualization becomes the next natural choice. On [6] the authors proposed a testbed based on OMNet++ Discrete Event Simulator. Their *Virtualmesh* framework provides new modules that allow the communication between the simulator and Virtual Machines (VMs) representing nodes in the network. Even though the flexibility gained by the usage of VMs, this testbed seems to be not easily scalable for large amounts of nodes due to the use of VMs which tent not to be efficient considering the testbed target and the resources' consumption.

The Extendable Mobile Ad-hoc Network Emulator (EMANE) [4] is an open source framework written in C++ that provides a flexible modular environment for simulated wireless networks and connecting them with external real applications with a TUN/TAP interface. Unfortunately, This project has few models that are not enough to simulate a wide range of environments compared to other popular simulators available.

Most of the testbed discussed so far have been developed for research purposes, with an emphasis in performance evaluation. Undergraduate education, however, have some additional requirements, such as higher realism, repeatability, and user experience. As such, most of the tools described so far are not the most adequate for classroom instruction. Thus, the main objective of our work was to develop an emulation tool that

- Provide the same user experience/interface as the one provided by physical devices running UNIX-based systems (e.g., AP running OpenWRT [8]).
- Provide the ability to add open-source packages such as OLSR, Quagga, etc.
- Allow to use typical networking diagnostic tools (e.g. tcpdump, wireshark)
- Provide high-fidelity, repeatable results.
- Scale to the same size as physical testbeds used in undergrad courses.
- Runs in a VM in our private cloud, so we can accommodate a large number of (remote) students simultaneously.
- Include a fidelity self-check where the system report latency when inserting a physical packet into the simulator's event queue, and validate that it doesn't exceed the packet time budget

The remainder of the paper is organized as follows: Section II reviews the related work, Section III describes our solution, Section IV we share our experiences implementing our testbed inside the university's private cloud and the benefits of using it on Undergraduate Teaching, and Section V presents our conclusions.

II. Related Work

In [9], Dorathy et. al. compare different simulation tools for wireless networks. They conclude that the open source NS-3 [10] simulator is to be preferred due to its long and growing list of modules that can recreate many desired scenarios, their periodic releases also fixes different bugs reported by their community of researchers. Furthermore, NS-3's emulation capabilities allow real-time simulations to forward packets outside the simulator through a TUN/TAP interface (see Figure 1), allowing real or emulated devices to interact with the simulation. Thus, NS-3 emulation framework is a good starting point for creating a high-realism testbed for undergrad classroom use.

Several attempts have been made to close the gap between simulations and real experiments using ns-3' emulation. On [11] the authors tried to reproduce a real experiment with NS-3 by feeding the simulator with real traces (nodes positions, link quality, etc). After comparing the real, trace-based simulation and plain simulation results, researchers concluded that the trace-based approach is much more accurate as it was able to reproduce the link instability encountered in the real experiment. Similarly, the authors of [12] proposed methodologies to pass from simulation to experimentation (allowing to run emulated resources in real testbeds) and from experimentation to simulation (repeating a real test with a trace-based simulation). These works, however, have focused on performance and not in providing a real user experience (as required for classroom education).

The Scala Compute Platform for Network Simulations (SCP for ns-3) [13] provides a user-friendly commercial solution to run (MPI-based) distributed ns-3 simulations in a cloud environment, with optimizations allowing it to scale to 100s and 1000s of nodes. However, this solution targets wireline networks where the topology can be easily split in clusters. In a wireless scenario, however, the physical channel induces dependencies that make such a division non trivial.

This paper differs from past works in that the emphasis is put on education as opposed to research. This, it aims to provide the end user (student) with the same interface as the actual equipment using common unix-based utilities such as *iwconfig-ns3*.

III. Testbed Development

We leveraged the NS-3's Emulation Mode shown in Figure 1 to built a testbed that is flexible, repeatable, accurate and realistic enough for use by undergraduate students, but requiring low cost resources.

With Emulation Mode, it is possible to use containers in lieu of physical devices, and use the NS-3 process to simulate the wireless channel. The devices (containers) can then run UNIX-based systems (OpenWRT [8], Ubuntu, etc.), popular networking tools (Python, Wireshark, etc.), and protocol implementations (OLSR, Quagga, etc.). However, manipulation of the simulated wireless interface (for example, to change its transmit power) was still only possible from ns-3 code inside the simulation, which breaks the realism required for

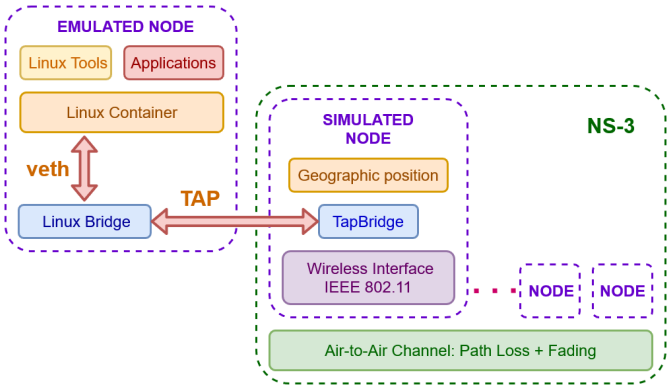

Fig. 1. NS-3's Emulation Mode with Linux Containers

undergraduate class use. Instead, a user in a real device would use the well-known "iwconfig" tool, or similar.

Thus, a new tool was developed (iwconfig-ns3) capable of managing NS-3 radio parameters from the devices (containers) imitating the behavior of iwconfig. To this end, extensions were added to the WifiNetDevice module of ns-3 so that it can process special control messages sent by our iwconfig-ns3 tool running in the devices (Linux Containers). These messages include commands to query (GET) information from the simulated wireless interface, and messages to change (SET) the value of one parameter from the simulated wireless interface. The end-user experience is the same as if the iwconfig tool was used on a physical device. The following sections explain deeper our testbed development with an special interest in our "iwconfig-ns3" tool.

A. NS-3 Emulation Mode with Linux Containers

As part of it's emulation mode, ns-3 has a module named *ns3::TapBridge* that creates a bridge between a simulated interface and a virtual TAP interface outside the simulator environment. Even tough this feature is presented on ns-3's documentation [10] as a way to connect Virtual Machines through a simulated communication channel, we saw an opportunity to our target of adding realism but still requiring low resources by the usage of Linux Containers (LXC) instead. We preferred LXC over other high-level tools such as Docker, Kubernetes, etc., since it provided the visibility/flexibility needed during our development.

A node in our testbed is made by the combination of the emulated (LXC) and simulated (ns-3) node. LXC containers let us use well know networking tools and test linux-based applications using few resources compared to a Virtual Machine. Traffic generated on the LXC travels over a virtual ethernet, a linux bridge and a TAP interface which its other side is attached to a socket inside the simulator. NS-3 associates each TAP interface with a *ns3::TapBridge* and, in our case, each of them with an object *ns3::WifiNetDevice* wich is the representation of the wireless interface (IEEE 802.11) inside the simulator.

After real packets are sent into the simulated environment, ns-3 take out its Ethernet headers, the rest is copied into an attribute called *buffer* of an object from class *ns3::Packet* that will be used inside the simulation. The simulated wireless interface adds layer 2 and layer 1 headers before calling the corresponding methods for transmission into the wireless channel. Similarly after a packet is received by a simulated node, if that node is the destination, the buffer is transformed in an Ethernet packet and forwarded to the emulated node through the TAP interface. Then the emulated node performs the expected processes with that packet.

B. New API for simulated radio interface live management

As this testbed is going to be used for education purposes, we needed to emulate the real experiences as close as possible. On physical laboratories, students can manipulate radio parameters inside each wireless router and analyze changes in the wireless communication performance thanks to the useful "iwconfig" linux tool. In contrast, in ns-3 simulations the wireless interface is a C++ object and its attributes can only be predefined before each simulation run. The functionality of manipulating radio parameters on-the-fly during a simulation is not a feature of ns-3. Although it will not be difficult for students to learn how to modify those attributes on the simulator, we wanted to provide them with a user experience as close to the real system as possible and mainly focus on the educational objectives of the lab experience.

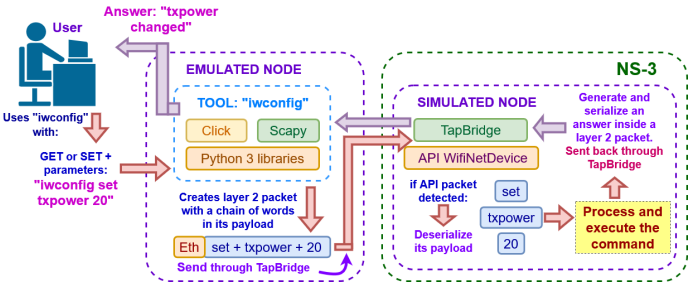

Fig. 2. Our iwconfig tool workflow

To this end, a "iwconfig-ns3" tool was developed that mimics the behavior of the well-known iwconfig tool (up to the help menu), so that the end user was not aware that he wasn't using the real iwconfig tool. The tool consists of two parts: The script "iwconfig" inside the Linux Container (emulation) and the new module *ns3::TapBridgeApiWifiNetDevice* added to the source code of ns-3 (simulation). The workflow of our tool is as follows (see Figure 2):

1) The python script iwconfig inside the container get the parameters that the user wants to query or change (GET or SET).
2) Then the script, with help from the packet-generating library Scapy, creates (serialize) a layer two packet with the special EtherType "0x0810", that contains a predefined chain of words with the parameters inside its payload, using the special character "@" as field separator.

3) This packet is forwarded through TAP interface to the simulation environment.
4) Once it is inside ns-3, a new method in *ns3::TapBridge* analyze the EtherType. If it coincides with the one specific for "iwconfig-ns3" then the packet is deserialized by the new method (*ns3::Buffer*) to obtain the parameters inside its payload.
5) Obtained parameters are forwarded to a new module (*ns3::TapBridgeApiWifiNetDevice*) on ns-3 that process them and determine the user request.
6) The algorithm then execute the change (if requested) or query the information requested.
7) Immediately, an answer is serialized inside a new packet and sent back to the container through TAP interface.
8) Finally, the script inside the container receives the packet, deserialize it and shows the output to the user.

An example of the usage of "iwconfig-ns3" is shown in Figure 3, where the TxPower is set to 30dBm, and then the values of all parameters are listed. This is the same behavior as expected with the actual *iwconfig* tool. The parameters listed in Figure 3 are all the iwconfig parameters that the tool currently supports (i.e., can list or modify).

```
root@node-1:~# iwconfig eth0 set TxPower 30
*TxPower changed to 30
root@node-1:~# iwconfig eth0
Wireless interface eth0:
*IEEE 802.11n 2.4 GHz
Mtu: 2296
CcaEdThreshold: -62
ChannelNumber: 36
RxSensitivity: -101
TxPower: 30
ControlMode: HtMcs7
DataMode: HtMcs7
FragmentationThreshold: 65534
MaxSlrc: 0
MaxSsrc: 0
RtsCtsThreshold: 65500
```

Fig. 3. "iwconfig-ns3" tool example for SET and then GET

C. Implementation at university's private cloud

Our school's Telecommunications Engineering department has a private cloud that provides support to the department including researchers' compute slices, virtual laboratories, and research testbeds. The cloud is orchestrated by our own Virtual Network Research Testbed (VNRT), which is based on OpenStack. with some modifications to run natively at layer 2 and to reduce the networking overhead and latencies introduced by the hypervisor. Its features include the virtualization of switches, routers and servers in user-defined topologies with a friendly graphic interface. In addition, VNRT has a hierarchical system of roles allowing, for example, that lab assistant can visualize the virtual console of their students, which is very useful for education purposes.

Each testbed user was provided a Virtual Machine (VM) inside our private cloud. In that VM a script was provided to setup the required number of containers (emulated devices) as well as to launch the ns-3 process. The VM was to be properly dimensioned to the anticipated load (see calibration, below).

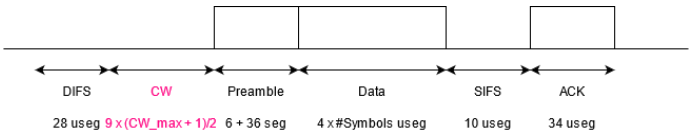

Fig. 4. Theoretic Model used to validate the calibration experiments' results

In order to maximize the performance of the ns-3 based system in a multicore *and* virtual environment, the priority and affinity of the ns-3 process needed to be modified. Indeed, as it was reported in [14], the NS-3 process, being single threaded cannot take advantage of the existence of multiple cores, but instead it suffers from performance degradation each time it moved to a different physical core, invalidating the previous cache/TLB buffer. Thus, the ns-3 process encounters a "cold" cache/TLB. In our system, the NS-3 process was pinned to the last (virtual) core of VM changing its affinity via the *taskset* command) and its dynamic priority was increased using the *nice* command. This process is then repeated in the physical server by setting the affinity and priority of the the actual process associated the last virtual cpu of the VM. The intention is to guarantee the ns-3 process ("the channel") of each student an get as close to 100% of CPU time as required.

The first version of the tool (used on the second semester of 2020) fulfilled the initial expectations but needed careful calibration to ensure the correctness of packet's latencies between each simulation. This required that the laboratory instructors have a good understanding of the wireless medium as well as ns-3 to debug it and check its functionality.

For the second version, completed in 2021, a fidelity self-check tool was included where the system report latency when inserting a physical packet into the simulator's event queue, and validate that it doesn't exceed the packet time budget. At the end of a simulation, the tool report the number of such incidents, if any, allowing the instructor to verify its correctness without having to understand the details of the simulator or the dynamic of the wireless system.

D. Testbed Calibration

In order to validate the proper functioning of the tool and its ability to handle the required load without introducing artificial packet delays or losses, stress and calibration tests were performed using a simple two-node topology with the channel pathloss set to 40dB (fixed, no fading). The 802.11n wireless interface of each node is set to MCS Index 7 (65Mbps) and the rest of parameters are set so that no RTS/CTS are sent. Using the Iperf tool, one node sends more traffic than the channel's capacity (saturation), and the received throughput is compared against the expected (theoretical) value obtained using the model shown in Figure 4.

Custom logs were added to our tool to record information that allows for validation without slowing down the simulation, which would result in the system's failure to respond to the packets arrival in real-time. The content of the logs are minimal: MAC packets transmission and reception times, if

```
root@node-2:~# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 46691
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  3.68 MBytes  30.9 Mbits/sec   0.046 ms    0/ 2624 (0%)
[  3]  1.0- 2.0 sec  3.66 MBytes  30.7 Mbits/sec   0.108 ms  146/ 2759 (5.3%)
[  3]  2.0- 3.0 sec  3.66 MBytes  30.7 Mbits/sec   0.090 ms  364/ 2977 (12%)
[  3]  3.0- 4.0 sec  3.67 MBytes  30.8 Mbits/sec   0.096 ms  358/ 2977 (12%)
[  3]  4.0- 5.0 sec  3.67 MBytes  30.8 Mbits/sec   0.093 ms  354/ 2974 (12%)
```

Fig. 5. Iperf test results

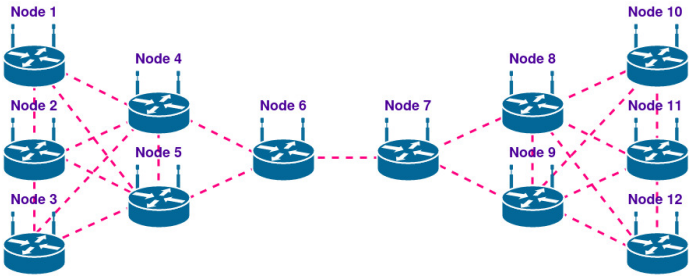

Fig. 6. Topology used in Lab. 4: A 12-node MANET with Hidden Terminals

the packet is being transmitted or received, node ID, type of packet, the MPDU, and the MAC address. Still, even with these lightweight logs, the simulation has to be compiled with the optimization flag enabled for g++ compilation (‘ -O -g -fstrict-overflow -march=native ‘) in order to prevent the logs from slowing down the simulation.

The calibration begins with a simple test of sending 1 UDP datagram of 5000 bytes (5000 Payload + 8 UDP Header). The MTU was set to 1500 bytes, consistent with a TUN/TAP interface that mimics a Ethernet interface, it is expected that 4 packets are forwarded through the container TAP interface to the ns3 simulation. The tool’s custom logs were reviewed to verify the size and number of fragments sent, to validate that our theoretical model was properly accounting for all the headers at different layers, and that throughput-optimizing features of 802.11n, such as fragment aggregation, were disabled. The timing of packets transmissions and receptions were also validated to make sure all the physical layer parameters were set as expected in the theoretical model shown in Figure 4.

Figure 5 shows the results of the stress test reported by the receiver (iperf server). It can be seen that the achieved throughput is in the range of [30.7, 30.9] Mbps, which closely resembles the expected theoretical value of 30.7Mbps. This proofs both that the system is well calibrated and that it is able to cope with the traffic load without introducing additional delays or packet losses.

IV. Experience at Undergrad Education

The development of this testbed began in the middle of the COVID-19 pandemic with the goal of using it in the upcoming semester. A first version was completed before the beginning of our Wireless Networking course on the second semester of 2020.

This course introduces the undergrad student to the analysis of the throughput of the wireless channel as an stochastic process. The first half of the course deals with point-to-point communication under a variable channel (fading) ignoring contention. The second half of the course studies the effect of contention under the CSMA-family of protocols, with emphasis 802.11n (WiFi). In the accompanying laboratories before pandemic, the students used a physical testbed made of TP-LINK TL-WR3600 Access Points (APs) to run experiments where they are required to choose the best configuration for a given scenario. The firmware of the APs has been reflashed with an image of the OpenWRT [8] operating system.

For the 2020-2 semester, 17 students enrolled in the course, and each was assigned a Virtual Machine (VM) with 4 cores and 4GB of RAM in our private cloud running the VNRT orchestrator. Each VM included scripts to deploy the containers and our modified ns-3 simulator required for each experience. Each container emulated one AP running OpenWRT with the ”iwconfig-ns3” tool to interact with simulated radio.

The laboratory guides were slightly modified to account for the fact that the access to the ”equipment” is through our private cloud and that now each so-called equipment is actually a container inside the student’s VM. Besides that, the students used the same commands and tools that were used in the pre-pandemic experiences. A total of 4 remote laboratory experiences were possible thanks to this testbed:

1) Bit-rate selection considering multi-path fading using UDP protocol. Ad-Hoc mode.
2) Bit-rate selection considering multi-path fading using TCP protocol. Ad-Hoc mode.
3) Analysis of CSMA/CA protocol with and without RTS/CTS considering multi-path fading and hidden terminals. Infrastructure mode.
4) Analysis of CSMA/CA protocol with and without RTS/CTS considering multi-path fading and hidden terminals. Ad hoc mode.

Laboratory 4 was the most challenging due to the usage of 12 nodes forming a multi-hop (mesh) network (Figure 6) and the routing protocol OLSR for its end-to-end communication. The main objective of this laboratory was to analyze the achievable MAC throughput under the ”loss-of-state” induced by RTS/DATA packets collisions on a mesh networks [15].

Students were asked to measure end-to-end throughput under different load levels for UDP traffic, comparing the case where RTS/CTS packets were used against the case where they were not. Traffic flowed from left to right and vice-versa, causing collisions in the link between nodes 6 and 7. The experiment were performed with 1470-bytes datagram as well as with 300-bytes datagrams. The students were able to identify that RTS/CTS mechanism can not stabilize the throughput for multi-hop ad-hoc networks. In addition, they could see the difference on the impact of the usage of RTS/CTS for for different packet sizes.

Before pandemic, students used to mention that only the physical testbed arrangement and the configuration of APs required too much time. In consequence, the percentage of students that did not finish the actual graded activities was high (around 50%). In addition, their different tests used to show different results which makes even more difficult to compare and analyze with other students in group as math calculus were not consisted with what they were experimenting.

On semester 2020-2 we ask students to tell us their experiences during these new format of laboratories.

- Most of them agree that thanks to the simulated channel, they were able to identify clearly the effects that were the goal of each laboratory.
- Their calculus were consistent with their results and encourage them to continue studying.
- Another comment was related to the cloud environment. Students mentioned that having a personal VM let them practice on-demand and be more prepared for the next evaluation.

At the end of the semester our first impressions, as teachers, were related to the accuracy of the laboratory experiences. The target is that students analyze specific phenomenons which are part of the course topics but that is difficult when many other factors are involved (like in real testbeds). Experiences inside the simulated-emulated testbed are completely controllable. Results are consistent with the expectations and students can analyze without distractions. In sum, the usage of our testbed leads to a better understanding of the course topics.

Based on the good results obtained when using the testbed so far, we have decided to continue its use in the course once the school reopens after the pandemia, combining in-person shorter-duration lab sessions with physical equipment with longer-duration homework assignments.

V. Conclusions

This paper presented the development of an NS3-based wireless network's emulation testbed enhanced with an API adding realism and its usage on undergraduate courses at this University.

A brief discussion of the state of the art highlighting the limitations of existing testbeds for undergrad education was presented. That information was used to determine the objectives of our own design: flexibility and repeatability without compromising reliability, as well as rational use of resources in a cloud environment.

Laboratory experiences exemplifies the advantages of our developed wireless testbed. On one hand, we were able to use real implementations like OLSR for linux and test its performance with linux-based tools like Iperf. On the other hand, the tool "iwconfig-ns3" did help to keep the complex simulated-emulated environment behind the scenes and let students focus on the main important topics of each lab.

Discussion of our classroom experiencies is presented showing that the goal of providing an adequate user experience for our students – at a time when real laboratory experiments were not possible – was achieved.

VI. Future Work

While extra care was taken to calibrate our experiences to guarantee fidelity of the results, we are currently working on a next version of the tool to overcome the one-core barrier by developing a multi-threaded channel simulator.

References

[1] *EMULAB*. Accesed on 08-23-2021. URL: https://www.emulab.net/.

[2] *WiTest*. Accesed on 08-23-2021. URL: https://witestlab.poly.edu/site/.

[3] D. Raychaudhuri et al. "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols". In: *IEEE Wireless Communications and Networking Conf.* Vol. 3. 2005, pp. 1664–1669.

[4] *Extendable Mobile Ad-hoc Network Emulator (EMANE)*. Accesed on 04-27-2020. URL: https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/EMANE/.

[5] R. Beuran et al. "QOMB: A Wireless Network Emulation Testbed". In: *IEEE GLOBECOM* (Mar. 2009).

[6] Maxa et al. "Emulation-Based Performance Evaluation of Routing Protocols for Uaanets". In: *LNCS* (2015), pp. 227–240.

[7] Rick McGeer et al. *The GENI Book*. 2016.

[8] A. Holt et al. "OpenWRT". In: *Embedded Operating Systems: A Practical Approach*. London: Springer, 2014, pp. 161–181.

[9] I. Dorathy et al. "Simulation tools for mobile ad hoc networks: a survey". In: *Journal of Applied Research and Technology* 16.5 (June 2019).

[10] *ns-3 Network Simulator*. Accesed on 08-23-2021. URL: https://www.nsnam.org/.

[11] H. Fontes et al.l. "A Trace-based ns-3 Simulation Approach for Perpetuating Real-World Experiments". In: *Proceedings of the Workshop on ns-3* (June 2017).

[12] H. Fontes et al. "ns-3 NEXT: Towards a Reference Platform for Offline and Augmented Wireless Networking Experimentation". In: *Proceedings of the Workshop on ns-3* (June 2019).

[13] *Scala Compute Platform for Network Simulations (SCP for ns-3)*. Accesed on 11-1-2020. URL: https://scalacomputing.com/network-simulation.

[14] David P. Wiggings et al. *Scaling NS-3 DCE Experiments on Multi-Core Servers*. MIT Lincoln Labs Tech. Report, Accession Number AD1033806. Lexington, MA, USA, June 2016.

[15] Cesar A. Santivanez. "Exploiting Multi-User Detection (MUD) Radio Capabilities to Improve Stability of CSMA/CA for MANETs". In: *IEEE LATINCOM*. 2018, pp. 1–6.