Sim-to-Real Transfer in Multi-agent Reinforcement Networking for Federated Edge Computing

Pinyarash Pinyoanuntapong*
ppinyoan@uncc.edu
University of North Carolina Charlotte
Charlotte, NC, USA

Tagore Pothuneedi*
tpothune@uncc.edu
University of North Carolina Charlotte
Charlotte, NC, USA

Ravikumar Balakrishnan
ravikumar.balakrishnan@intel.com
Intel Labs
USA

Minwoo Lee
minwoo.lee@uncc.edu
University of North Carolina Charlotte
Charlotte, USA

Chen Chen
chen.chen@ucf.edu
University of Central Florida
Florida, USA

Pu Wang
pu.wang@uncc.edu
University of North Carolina Charlotte
Charlotte, NC, USA

Abstract

Federated Learning (FL) over wireless multi-hop edge computing networks, i.e., multi-hop FL, is a cost-effective distributed on-device deep learning paradigm. This paper presents FedEdge simulator, a high-fidelity Linux-based simulator, which enables fast prototyping, sim-to-real code, and knowledge transfer for multi-hop FL systems. FedEdge simulator is built on top of the hardware-oriented FedEdge experimental framework with a new extension of the realistic physical layer emulator. This emulator exploits trace-based channel modeling and dynamic link scheduling to minimize the reality gap between the simulator and the physical testbed. Our initial experiments demonstrate the high fidelity of the FedEdge simulator and its superior performance on sim-to-real knowledge transfer in reinforcement learning-optimized multi-hop FL.

Keywords: Federated learning, Reinforcement Learning, Transfer Learning, Edge Computing, Network Simulation

ACM Reference Format:
Pinyarash Pinyoanuntapong, Tagore Pothuneedi, Ravikumar Balakrishnan, Minwoo Lee, Chen Chen, and Pu Wang. 2021. Sim-to-Real Transfer in Multi-agent Reinforcement Networking for Federated Edge Computing. In *The Sixth ACM/IEEE Symposium on Edge Computing (SEC '21), December 14–17, 2021, San Jose, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3453142.3491419

Both authors contributed equally to this research*.
This work is funded by Intel/NSF joint grant 2003198 and NSF 2008447.

SEC '21, December 14–17, 2021, San Jose, CA, USA

ACM ISBN 978-1-4503-8390-5/21/12…$15.00
https://doi.org/10.1145/3453142.3491419

1 Introduction

In distributed machine learning, federated learning (FL)[1] is envisioned as a breakthrough technology, enabling machine learning to work in a distributed manner. In FL, worker nodes compute the model updates locally and send them to the server to update the shared global model. This prohibits raw data exchange and reduces potential data privacy risks. The edge computing devices interconnected by the wireless multi-hop network constitute the multi-hop wireless edge computing network. Enabling FL over multi-hop wireless edge computing networks (i.e., multi-hop FL) not only can augment AI experiences for urban mobile users, but also can democratize AI and make it accessible in a low-cost manner to everyone, including the large population of people in low-income communities, under-developed regions, and disaster areas. Despite its great advantages, the convergence of multi-hop FL can be greatly slowed down by the noisy and bandwidth-limited multi-hop wireless links. To address this fundamental challenge, we exploited multi-agent reinforcement learning (MA-RL) for FL optimization, which minimizes the networked induced latency by learning the forwarding paths with the least delay for FL traffic flows [2, 3]. To demonstrate the practical impact of our proposed solution, we developed the FedEdge [3], which is the first experimental framework in the literature for FL over multi-hop wireless edge computing networks. FedEdge thus enables fast prototyping, deployment, and evaluation of novel FL algorithms along with machine learning based FL system optimization methods in real-life wireless devices.

Although FedEdge can provide valuable and broader insights into the practical performance of FL in the field, FedEdge can only run on physical wireless devices. It demands much higher training time when testing a number of different networking and computing configurations. Scaling the testbed with a more extensive setup makes the computational demands even worse. Simulations provide a cost-efficient and scalable solution to this problem. Running multiple physical layer simulations simultaneously significantly reduces the training time for faster convergence. In addition, high fidelity simulation environment can facilitate the training of

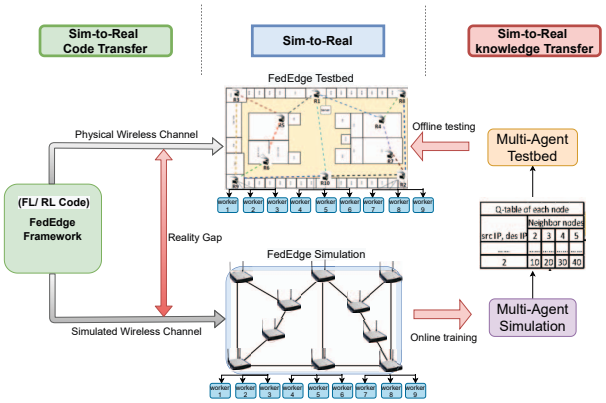


Figure 1. Overview of Sim-to-Real Transfer

the effective reinforcement learning (RL) policies during the exploration stage and enable sim-to-real knowledge transfer [4], where the knowledge, e.g., Q networks, learnt from simulations, can be transferred to the real testbed to enhance the efficiency of RL-based approaches. A major impediment is that the only existing network environment simulator to support such requirements is ns3-gym[5]. ns3-gym is a simulator that integrates the ns-3 and an RL simulation interface, OpenAI gym [6]. It has a wide range of ML and RL applications to simulate various wired and wireless environments. However, the usage of a real network stack with recent implementations of ns3 using direct execution code (DCE) [7] is quite limited due to the lack of significant community effort and limited flexibility/programmability in SDN/open-flow environments. ns-3's implementation of open-flow switch is a class 'OpenFlowSwitchNetDevice' and does not have access to the programmable flow tables.

1.1 Challenges

Overall, we believe the following challenges in the wireless networking domain limit the progress of federated learning in wireless multi-hop networks.

- *Existence of reality gap*: Even with the overwhelming success of reinforcement learning and federated learning for SD-WAN [3], training the models on simulations and directly executing them on real networks yields poor performance due to the differences between the physical layer simulators and real networks.
- *Lack of reliable training environment*: The performance of machine learning techniques like FL and RL depends on the training data and a reliable training environment. For many reasons, such as lack of knowledge, time, or resources to build the wireless environment, researchers usually tend to work directly on the testbed setup and spend a large amount of training time to develop models.
- *Need for wireless multi-hop simulator*: The existing wireless simulators mainly support infrastructure-based wireless environments with single-radio single-channel per link which seriously limits the ability to simulate realistic usage settings. They do not account for multi-channel multi-radio based multi-hop network operations which are increasingly becoming realistic. Specifically, using multi-channel multi-radios for wireless mesh backbone networks will tremendously reduce the interference from other nodes and increase the overall performance.

Therefore, we build a physical simulator for wireless multi-hop networks called FedEdge simulator to address the current problems of emulating a wireless multi-hop network and integrate it with our FedEdge framework. This approach will give FedEdge the advantage of being the first framework to incorporate a fully functional multi-agent reinforcement wireless network for federated edge computing, which enable to develop and test wireless networks involving federated learning with high fidelity simulations.

1.2 Contributions

To the best of our knowledge, our implementation is the first attempt in the wireless networking domain to test and validate knowledge transfer from simulation to reality in federated computing with multi-agent reinforcement networking. Our objective in this work is to build a custom realistic physical layer simulator called FedEdge simulator to integrate with our existing FedEdge architecture as shown in Figure 2. Following are our contributions in this paper:

- *Simulation to real code transfer*: Using real-time production environments such as OpenFlow software switches and Linux software tools such as mac80211_hwsim, hostapd, wpa_supplicant, Linux Namespace Containers (LXC), Traffic Control (TC), and Netlink protocol in the simulator enables rapid development and testing of the code and porting the same codebase to real testbed environments. This reduces the development and debugging times and promotes productivity.
- *Simulation to real knowledge transfer*: Our implementation of the FedEdge simulator using the dynamic link scheduling method reduces the reality gap between the simulations and actual physical networks. Integrated with our existing FedEdge framework, the simulator can help develop accurate models and yield better simulation to real transfer performance.
- *Accelerated prototyping for training*: Multiple simultaneous runs enable FL and RL to quickly discover hyper-parameters such as the number of local training round, regularization parameters for model updates, the number of stragglers, model tuning, learning rate, and exploration parameters. FedEdge simulator allows us to evaluate several configurations in parallel thereby speeding up the model selection for the development cycle.

2 Preliminaries

In this section we briefly provide a background on the fundamental concepts and components considered for our integrated simulation and testbed framework.

2.1 Accelerating Multi-hop FL via Multi-Agent RL

A multi-hop FL[2, 3] system consists of a central server that serves as an aggregator and a multi-hop wireless link to edge servers, referred as workers. The architecture of FL across a multi-hop wireless network is shown in Figure 2. FL methods are developed to manage distributed neural network training over numerous devices, where each device has its own training data and the goal is to train a common model with an objective to minimize the training loss.

FL methods use a standard optimization technique called local stochastic gradient descent (SGD) to unravel the above optimization problem, which alternates between local SGD iteration and global model averaging for multiple rounds. The worker seeks to decrease its local loss $F^k w$ throughout each round by conducting H mini-batch SGD iterations. After iterating through H local SGD iterations, the worker nodes send their updates to the server, which averages the collected local models and aggregates them into a global model. The updated global model is distributed to the workers, and this process continues again. Due to the dynamic nature of wireless multi-hop networks, high and nomadic end-to-end delays become the governing factor in determining the convergence times in multi-hop FL.

To address such problems, MA-RL was proposed to minimize the wall-clock convergence time to achieve the desired FL accuracy. The problem was formulated as a multi-agent Markov decision Process (MA-MDP) and solved by model-free multi-agent reinforcement learning. In this case, each router is an agent, which observes the network states (e.g., the source IP and destination IP of the incoming FL packet) and learns the optimal networking policy (e.g., the forwarding action at each router) based on the reward signals (i.e., negative per-hop delay) with an objective to maximize the expected total return (i.e., end-to-end delay) from the initial state (i.e., when the FL packet entering the network) to the terminal state (i.e., when the FL packet leaving the network).

2.2 FedEdge Experimental Framework

FedEdge (Figure 2) is the experimental prototype framework developed to support wireless multi-hop federated learning. To fast prototype multi-hop FL in the physical wireless devices, we developed FedEdge framework that includes federated computing and federated networking [3]. Each component is built in a layered approach where each layer has bidirectional communications to upper or lower layers. In general, federated computing involves customizing and configuring FL functionality, and federated networking is an AI-oriented wireless network operating system that is mainly responsible for fast and reliable wireless network links between the aggregator node and worker nodes. The main goal of this component is to optimize the wireless network through AI-enabled algorithms to perform route optimizations and provide in-band telemetry data. Federated computing contains three layers: (1) Datasets layer that stores the training datasets for FL, (2) Compute layer that provides core functionalities, for instance, to train a model and store the trained model, and finally (3) Communication layer that uses FedEdge COMM protocol to establish and maintain connections with aggregator and worker nodes. Similar to the federated computing component, the federated networking component has three layers that include (1) Dataplane Layer that integrates software switch to allow programmable packet switching and the in-band telemetry, (2) The Network Core Layer that handles the traditional network functions such as node discovery, maintaining network links, counters and status database, and (3) RL App layer that contains the actor-critic RL agent for learning delay-optimized routes at the edge nodes. The integration of the simulator to FedEdge architecture is simple because of the modularity of FedEdge, which can use the topology built by the FedEdge simulator and train the RL agent on the wireless multi-hop backbone network.

2.3 FedEdge Physical Testbed

A software-defined wireless mesh network testbed was deployed in Woodward Hall at the University of North Carolina at Charlotte (UNCC). This testbed consists of 10 Gateworks routers connected with three of WLE900VX wireless interface cards to enable the multi-radio wireless node and run with Ubuntu 20.04 as the operating system. Each mesh router was configured to operate in Mesh Point (MP) mode, with fixed 2.4 and 5 GHz channel, 20 Mhz channel width in 802.11ac operating mode, and 15 dBm transmit power. Then, three wireless interfaces were bridged to a data plane with a programmable packet handling routine (i.e., OpenFlow flow table). Besides, we deployed 10 of Nvidia Jetson as edge computing nodes.

3 FedEdge Simulator Design and Prototyping

3.1 FedEdge simulator overview

FedEdge simulator is able to mimic electronic hardware wireless radio and its operations to model an environment inside a computer system. It uses virtualized hardware and Linux-based software tools to build and maintain wireless networks. Most of the tools used are Linux in-built tools namely IP namespaces for containerization, mac80211_hwsim[8] to create virtual radio interfaces, iw tool to manage the radio interfaces, and batman_adv[9] for shortest path routing. The simulator is completely built using the python language and allows the use of open-flow softswitch [10] for software-defined networking. In this section, we describe the framework of the

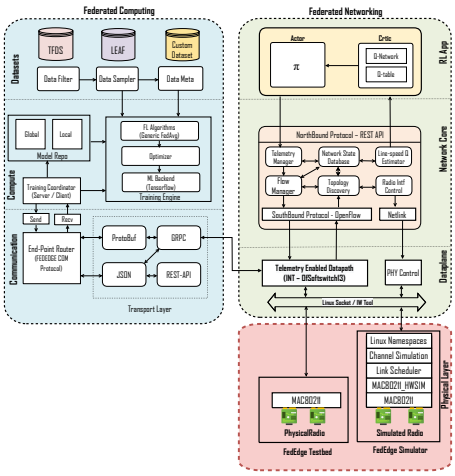


Figure 2. FedEdge Framework Architecture with FedEdge Simulator

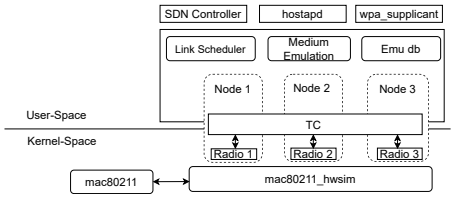


Figure 3. FedEdge Simulator in TClink

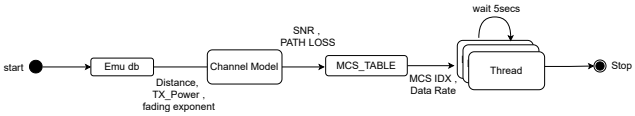


Figure 4. Dynamic Link Scheduling

FedEdge simulator, the modes of operation, workflow, and channel modeling.

As shown in Figure 3, our simulator is built on top of traffic control (tc) link [11] to shape the traffic on the egress interface of each node based on the signal-to-noise ratio (SNR) obtained from the channel models and interference models. There is no switching of frames from kernel to user space, ensuring zero copies thereby reducing the overhead. Link scheduler comes into the play in tclink mode to dynamically schedule the links between nodes and runs for every 5 seconds and interacts with the medium simulator to introduce signal fading and interference between the nodes. Figure 4 describes the flow of the link scheduler. Once the simulator starts and builds the necessary wireless medium parameters such as signal fading, interference, and propagation loss, several threads are created for each node in the topology building stage based on the number of the interfaces each node contains.

3.2 Simulator Workflow

3.2.1 Stages. This section describes the general workflow for the FedEdge simulator and its input and output. The input to the simulator is a JSON file that contains context related to the topology. Parsing the input configuration file, the simulator builds the wireless medium parameters and constructs the topology. The topology is constructed using the mac80211_hwsim driver, and the links between the nodes are created using the iw tool if the topology is a complete mesh. In addition, the simulator uses wpa_supplicant and hostapd for client-based connections. If the topology contains a SDN controller setup, the simulator configures the OpenFlow datapath connections on the nodes and connects the nodes to the controller. Next, the simulator uses a link scheduler to update link parameters of each interface.

3.2.2 Channel Modelling. FedEdge simulator supports multi-hop wireless mesh networks and standard UAV networks. Propagation models available in the simulator are traditional static channel models such as Log Distance, Log-Normal Shadowing, etc. In addition, custom models can be integrated into the simulator that also includes GAN-Based Channel Model [12] and Trace-Based Channel Model. In stage 1, the channel model name is parsed from the input configuration file. The propagation loss is calculated between the nodes and stored in the simulator database based on the model. In stage 2, the simulator uses the same information to calculate the wireless parameters.

3.2.3 Trace-Based Channel Model. Verifying or evaluating a wireless network is the most challenging task given the random, unpredictable nature of the wireless medium. Trace-based channel modeling is an essential technique to replay or reproduce the wireless scenarios from testbed to simulator or vice versa. This approach will provide the required data to model complex wireless systems. Trace-based modeling in the FedEdge simulator works in two modes: replay an existing trace and trace generation for a given input topology.

Trace Replay: FedEdge simulator can replay the trace for each interface independent of a node. For replaying trace, the simulator checks the topology file for trace file locations and processes the files of each interface. Threads are generated based on the number of replay files and passed to the link scheduler. Based on the signal level on each trace file, the link scheduler runs the traffic control (tc) and sets the bandwidth on each interface of the replay nodes periodically until the trace files are complete. The node's interfaces excluded from the trace replay are manually updated by the simulator and link scheduler based on the input channel model and interference model.

Trace Generator: The second mode of the trace model works to generate the trace logs, which can be used to replay on the testbed, visualize or train data for machine learning. When the trace functions are executed, they take the input

from the parsed input configuration and learn the replay interfaces. The program works in reverse order compared to replay, where it takes in the interface names and produces the trace files in a csv file format, which contains time, MCS index, RSSI, loss, and traffic rate. The link scheduler captures the data every 5 seconds until an interrupt occurs. Based on user-selected channel models, interference models, and the topology of the network, the simulator calculates and records all the parameters to the trace files.

4 Experimental Evaluation

In this study, we present our experiments in the following scenarios. First, we evaluate the physical layer of FedEdge simulator by comparing the FL experiments alone without MA-RL on both simulator and physical testbed. Next, we compare the reality gap difference by training MA-RL agents in on online fashion in both environments. Last, we study the effect of knowledge transfer of the MA-RL by evaluating the pre-trained model's performance in the physical testbed.

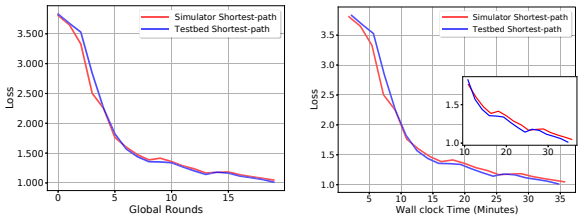
4.1 Experiment Setup

Network Environment Setup: To evaluate the gap between the simulated network and the real testbed environment, we identically set up the topology with the same wireless conditions with 10 routers, 9 workers, and a server as shown in Figure 1. We set up our FedEdge framework to work on top of the physical layer of the simulator by using shortest path routing and MA-RL to route packets. A replay simulation runs by inputting the RSSI signal logs from the testbed backbone nodes in WiNSLAB at UNCC. The simulator uses the link scheduler and updates the bandwidth on the link interfaces based on the signal seen at the point of the replay time. The link scheduler converts the signal to traffic rate based on MCS index table of 802.11ac 20MHz channels specified at [13].

Model and Dataset: The CNN model has two convolution layers, with 32 and 64 filters respectively. Each convolutional layer is attached with a 2x2 max pooling layer. The convolutions are followed by a fully connected layer with 128 units with ReLU activation. The final output layer consists of a fully connected layer with sofmax activation. The learning rate of 0.1 is used for local SGD in all workers and the model size is 5.8 Mbytes. We test the model with the well-known FEMNIST benchmark dataset, the extended federated version of MNIST [14] from the LEAF[15], which consists of 62 classes.

4.2 Main Results

Simulator Fidelity Evaluation: Figure 5 examines the closeness of the realistic physical layer in the proposed simulation to the physical testbed with the experimental results of shortest-path routing in both environments. Both FL experiments lead to the same iteration convergence performance at

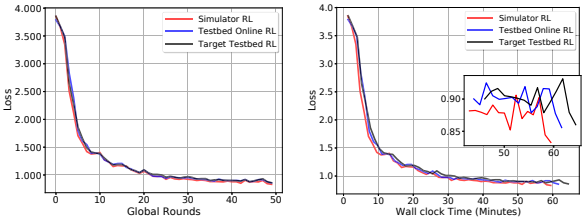


(a) Iteration loss convergence (b) Wall-clock time loss convergence

Figure 5. Loss convergence of FL after 20 epochs of shortest-path routing in FedEdge simulator (red) and on testbed (blue)

which they achieve the same loss after running the same number of epochs (Figure 5(a)) and slightly different wall-clock time (Figure 5(b)). Therefore, we verify that both environments are almost identical.

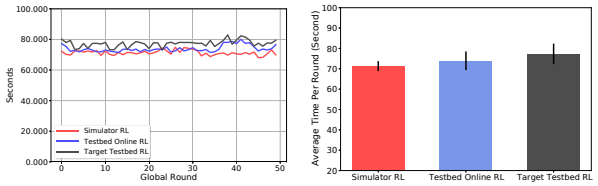
In the FL experiment with RL-based routing, where each agent is trained in an online fashion in simulator and physical testbed, the iteration loss convergence in terms of global rounds are identical as shown in Figure 6. However, an online RL routing in the simulator achieves slightly better wall-clock time than in the testbed after 50 epochs, which is only 2 minutes difference. Given the identical protocol stacks for both, the results confirm that the FedEdge simulator narrows the reality gap effectively.



(a) Iteration loss convergence (b) Wall-clock time loss convergence

Figure 6. Loss convergence comparison results after 50 epochs of On-policy softmax in FedEdge simulator (red), On-policy softmax on Testbed Online learning (blue), On-policy softmax on Testbed Target testing (black)

MA-RL Sim-to-real Knowledge Transfer: Now, we evaluate the initial sim-to-real transfer performance of the online MA-RL routing given the verified close reality gap. We pre-train MA-RL agents in a simulated environment and transfer the learned knowledge (Q-table of each agent) to the testbed for the target testing, where we freeze the Q-table and allow an agent to only exploit the pre-trained softmax policy. In Figure 6, we observe that online training both on the simulator and testbed achieve a better performance in terms of wall-clock time compared to the target testing in the testbed. This is because the frozen target testing lost adaptability to re-learn the dynamic FL traffic patterns in the wireless environment.



(a) Time per round (b) Average time per round

Figure 7. Time and average time per round of On-policy softmax in FedEdge simulator (red), On-policy softmax on Testbed Online learning (blue), On-policy softmax on Testbed Target testing (black) across 50 rounds of FL communications

However, the gap between the target testing in the testbed and both online trainings is marginal. Figure 7(a) shows that training time per epoch for both testbed experiments follow the same trend, which are around 72-80 seconds per epoch and slightly slower than the online simulator. Figure 7(b) shows that the average time and variance per epoch of all the approaches across 50 rounds of FL communications.

5 CONCLUSIONS

MA-RL knowledge transfer in FL over wireless multi-hop network is challenging due to the reality gap between the code base and entire stack from the physical layer to the application layer. In this paper, we presented a fully functional approach for transfer learning from simulation to reality in wireless federated edge computing networks. Moreover, we implemented a real physical testbed to validate the proposed FedEdge simulator for effective sim-to-real transfer. Our results confirm the high fidelity of the proposed FedEdge simulator that significantly minimizes the gap from real testbed.

References

[1] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016. [Online]. Available: http://arxiv.org/abs/1610.05492

[2] P. Pinyoanuntapong, P. Janakaraj, P. Wang, M. Lee, and C. Chen, "Fedair: Towards multi-hop federated learning over-the-air," in *Proceedings of IEEE SPAWC*, 2020.

[3] P. Pinyoanuntapong, P. Janakaraj, R. Balakrishnan, M. Lee, C. Chen, and P. Wang, "Edgeml:towards network-accelerated federated learning over wireless edge," in *arXiv pre-print*, 2021.

[4] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai, "Retinagan: An object-aware approach to sim-to-real transfer," 2021.

[5] P. Gawłowicz and A. Zubow, "ns-3 meets OpenAI Gym," in *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, November 2019. [Online]. Available: http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/2019/gawlowicz19_mswim.pdf

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[7] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, "Direct code execution: Revisiting library os architecture for reproducible network experiments," ser. CoNEXT '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 217–228. [Online]. Available: https://doi.org/10.1145/2535372.2535374

[8] J. Malinen. (2008) mac80211_hwsim. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/mac80211_hwsim.html

[9] S. W. Marek Lindner. (2011) batman_adv. [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki

[10] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to bofuss: The basic openflow userspace software switch," *Journal of Network and Computer Applications*, p. 102685, 2020.

[11] "Linux traffic control," 2001. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[12] W. Xia, S. Rangan, M. Mezzavillla, A. Lozano, G. Geraci, V. Semkin, and G. Loianno, "Generative neural network channel modeling for millimeter-wave uav communication," 12 2020.

[13] wlanprofessionals mcs snr rssi chart. [Online]. Available: https://wlanprofessionals.com/mcs-snr-rssi-chart/

[14] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[15] S. Caldas, S. Meher Karthik Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A Benchmark for Federated Settings," *arXiv e-prints*, p. arXiv:1812.01097, Dec. 2018.