NeXT: Architecture, Prototyping and Measurement of a Software-Defined Testing Framework for Integrated RF Network Simulation, Experimentation and Optimization

Jiangqi Hu[1], Zhiyuan Zhao[1], Maxwell McManus[1],
Sabarish Krishna Moorthy[1], Yuqing Cui[1], Nicholas Mastronarde[1],
Elizabeth Serena Bentley[2], Michael Medley[2], Zhangyu Guan[1*]
[1]Department of Electrical Engineering, University at Buffalo, USA
[2]Air Force Research Laboratory (AFRL), Rome, NY 13440, USA
Email: {jiangqih, zzhao24, memcmanu, sk382, yuqingcu, nmastron, guan}@buffalo.edu, {elizabeth.bentley.3, michael.medley}@us.af.mil

Abstract

To support rigorous and repeatable experimental evaluation of wireless networked systems, the community has made significant efforts to develop experimentation platforms. However, existing platforms primarily focus on the data plane, i.e., the forwarding infrastructure, without explicitly considering the control plane. To fill this gap, in this work we develop *NeXT*, a software-defined playground with integrated wireless network simulation, experimentation and optimization capabilities. We first design the data plane, which integrates an event-driven broadband wireless network simulator called *UBSim* and a software-defined wireless network testing facility called *RoboNet*.

*Corresponding author

[1]A preliminary shorter version of this paper appeared in the Proceedings of *IEEE FNWF Workshop on Federated Testbed as a Service for Future Networks: Challenges the State of the Art*.

[2]ACKNOWLEDGMENT OF SUPPORT AND DISCLAIMER: (a) Contractor acknowledges Government's support in the publication of this paper. This material is based upon work funded by AFRL, under AFRL Contract FA8750-20-C-1021 and FA8750-21-F-1012, and in part by the NSF under Grant SWIFT-2229563. (b) Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL.

[3]Distribution A. Approved for public release: Distribution unlimited: AFRL-2023-2490 on 22 May 2023.

We then design NeXT's control plane, where a software toolchain is developed and deployed to support both traditional model-based optimization and new data-driven control techniques. We showcase the experimentation capability of NeXT considering a series of optimization and control problems in different wireless networks.

Keywords: Software-defined testbed, Wireless networks, AI/ML

1. Introduction

In the past decades, the evolution of wireless network systems has significantly changed and will continue to change the way we live and work, our commercial activities as well as national security. However, as of today the wireless research community is still lacking a mature ecosystem to support rigorous and repeatable experimental evaluation of wireless networked systems. To fill this gap, significant efforts have been made by the community. A recent milestone is the NSF Platforms for Advanced Wireless Research (PAWR) program, which attempts to develop four large-scale outdoor experimentation platforms for advanced wireless research [1]. As of today, three of them have already been developed and are available to the wireless community. These are POWDER-RENEW for experiments in the sub-6 GHz frequency bands [2], COSMOS for experiments in both sub-6 GHz and mmWave frequency bands as well as edge computing [3], and AERPAW for experiments with wireless unmanned aerial vehicles (UAVs) [4].

While existing community shared facilities have significantly advanced experimental research for new wireless systems, it is still challenging to fully meet the needs of experimental wireless research in the era of data-driven networking. First, to simplify the modeling, control and optimization of heterogeneous NextG networks, data-driven control based on Artificial Intelligence (AI) and Machine Learning (ML) has attracted significant research attention [5, 6]. However, the effectiveness of AI/ML algorithms largely relies on sufficient well-labeled data for policy training [7, 8]. It is typically time consuming and sometimes unsafe to collect training data in real-world environments [9, 10]. Second, the design, prototyping and verification of new network control algorithms require engineers to grapple simultaneously with mathematical modeling, distributed control, protocol design across different layers of the protocol stack, as well as their implementation and deployment. This process is typically complex, tedious and error-prone.

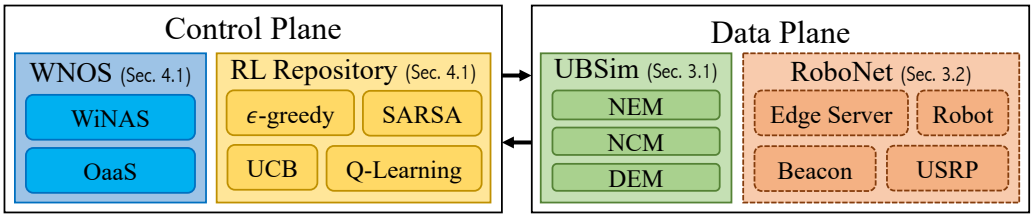


Figure 1: NeXT testbed architecture and paper organization.

To address these challenges, *in this paper we present NeXT, a software-defined wireless Network X-Control Testbed, where "X" refers to optimization, simulation and experimentation.* In a nutshell, NeXT provides an integrated testing framework, in which researchers are allowed to generate in an automated manner distributed cross-layer network optimization algorithms, simulate the generated algorithms in software, and then validate the simulation results based on testbed experiments. The overall architecture of NeXT is illustrated in Figure 1, where there are two planes, *Data Plane* and *Control Plane*. The former provides simulation and experimentation capabilities, and the latter implements network optimization and control functionalities.

The main contributions of this work are as follows:

- We first design the data plane for the NeXT testbed. In this plane, we first integrate UBSim with NeXT for software-based network simulation. UBSim is an event-driven simulator that has been developed at the University at Buffalo for broadband (microwave, mmWave and terahertz bands) aerial and ground wireless networking. We also develop a testing facility for mobile networks based on software defined radios (SDRs).

- We then design NeXT's control plane, which supports traditional model-based control and new data-driven control techniques. For the former, Wireless Network Operating System (WNOS) [11] has been deployed to enable automated generation of distributed cross-layer control algorithms. For the latter, a reinforcement learning (RL) repository is developed supporting various RL algorithms. A scheme to automatically adjust robots' posture and positions is proposed to mitigate the error introduced by the mobile hotspots.

- We showcase the optimization, simulation and experimentation capa-

bilities of the NeXT testbed considering a series of wireless network control problems. These include narrow-band multi-hop communications, srsRAN-based cellular networks and millimeter wave (mmWave)-band communications. A set of application programming interfaces (APIs) have been designed to simplify access to NeXT's data and control planes.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. We present the testbed's data plane in Section 3 and its control plane design in Section 4. Seven example experiments and results are given in Section 5. In Section 6, we discuss the new research topics that can be studied by using our testbed. Finally, we conclude in Section 7.

2. Related Work

With the development of wireless technology, researchers from both industry and academia are no longer satisfied testing their algorithms in a single simulated environment. Thus a lot of testbeds have been proposed and established to meet the needs of experimentation and verifying algorithms in the real world. The NSF PAWR program aims to enable experimental wireless communications research across devices, communication techniques, networks, systems, and services conceived by the US academic and industrial wireless research community and deployed in partnership with local communities [12]. POWDER is a facility for testing future wireless communications and networking technologies in a city-scale "living laboratory" [2]. COSMOS aims at design, development, and deployment of a city-scale advanced wireless testbed to support real-world experimentation on next-generation wireless technologies and applications [3]. Colosseum is the world's largest network emulator providing researchers with testing at scale, offsetting the site specificity of a physical testbed [13]. AERPAW is the first aerial wireless experimentation platform spanning 5G technologies and beyond and with the potential to create transformative wireless advances for aerial systems [4]. In [14], the world's first fully programmable and open-source massive-multiple input multiple output (MIMO) platform named RENEW is introduced. However, these platforms either do not consider mobile nodes or do not provide data-driven tools that simplify the experimentation process.

A unique national research infrastructure called FABRIC is proposed in [15] to enable cutting-edge and exploratory research at-scale in networking,

cybersecurity, distributed computing and storage systems, machine learning, and science applications. *DeterLab* is a shared testbed providing a platform for research in cybersecurity and serving a broad user community [16]. An open-source platform called M^3 is designed in [17, 18] to facilitate research in 5G vehicular networking and automotive sensing. In [19], a community-shared, open-source, open-architecture infrastructure for mobile underwater wireless networks called *mu-Net* is proposed. Readers are referred to [20, 21] for more information about the aforementioned testbeds. Arena is an open-access wireless testing platform [22] that can be used to test key wireless technologies, such as synchronized MIMO transmission schemes, multi-hop ad hoc networking, multi-cell long term evolution (LTE) networks, and spectrum sensing for cognitive radio. The authors in [23] propose the SkyHaul platform for channel modeling in mobile scenarios. An integrated testbed TeraNova for ultra-broadband wireless communications is developed in [24], which supports the testing and validation of new terahertz (THz) channel models and physical layer solutions. A testbed based on FlockLab [25] deployed in a campus-scale is designed in [26] to better support testing of long-range communications. However, these primarily focus on the physical platform's development, while neglecting the potential benefits of pairing the physical testbed with a simulator (e.g., using the simulator to accelerate the training of AI/ML algorithms).

Different from the above discussed testing facilities that primarily focus on the development of the forwarding infrastructure, i.e., the data plane, in this work, we focus on both data and control planes and aim to design a software-defined testbed with integrated simulation, experimentation and optimization capabilities for mobile wireless networks.

3. Data Plane Design

The data plane provides the forwarding infrastructure for the NeXT testbed. As illustrated in Figure 1, two forwarding infrastructures have been designed: *UBSim* for software-based network simulations and *RoboNet* for experiments based on SDRs.

3.1. Software Simulations Based on UBSim

UBSim, evolved from simulators in [6, 27], is a new wireless network simulator written in Python and based on the SimPy discrete-event simulation framework [28]. The simulator provides a configurable network-layer

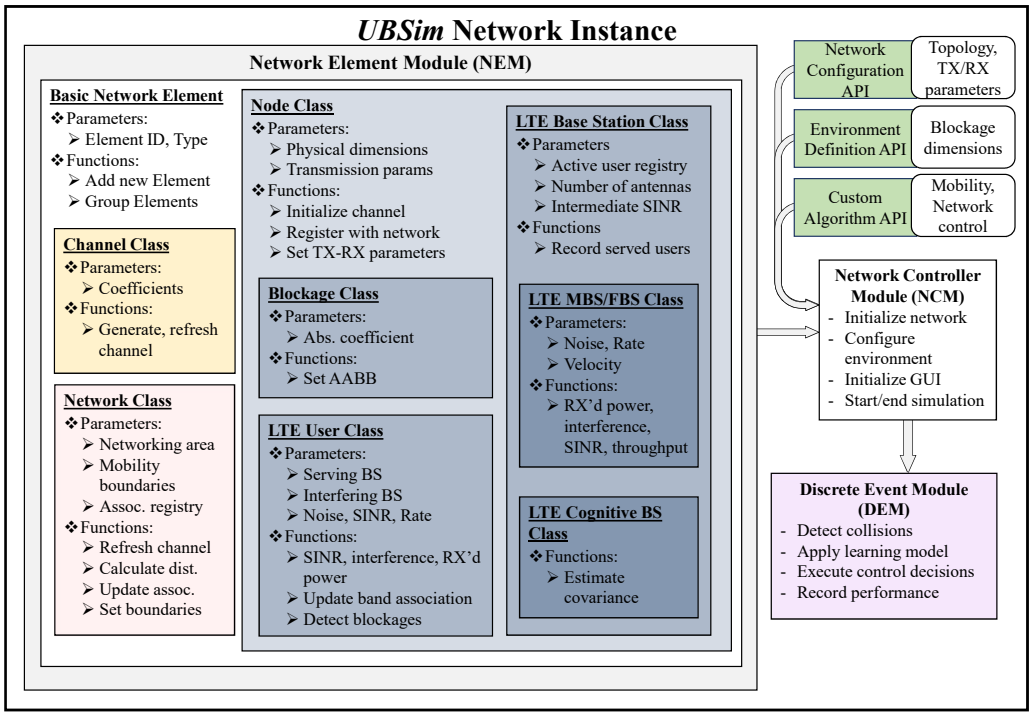


Figure 2: Architectural overview of UBSim network simulator.

simulation supported by analytical models for various PHY- and MAC-layer protocols. This lightweight computational design enables faster-than-real-time iteration as well as on-the-fly adjustments to the protocol stack of each simulated node, which sets UBSim apart as a highly effective simulator for experiments focusing on protocol stack and topology self-configuration. The various node mobility types supported by UBSim enables investigation into aerial networking, including both UAV swarm and hybrid aerial-ground network control problems. UBSim supports general AI/ML algorithm deployments, and has been demonstrated for reinforcement learning (RL), deep RL, and multi-agent RL experiments.

As depicted in Figure 2, UBSim comprises three primary modules to handle the behavior definition of various network elements, as well as three APIs to support a wide range of custom networking scenarios. Specifically, the network element module (NEM) defines the behaviors of all types of communication nodes, environmental blockages, channels, and the network as

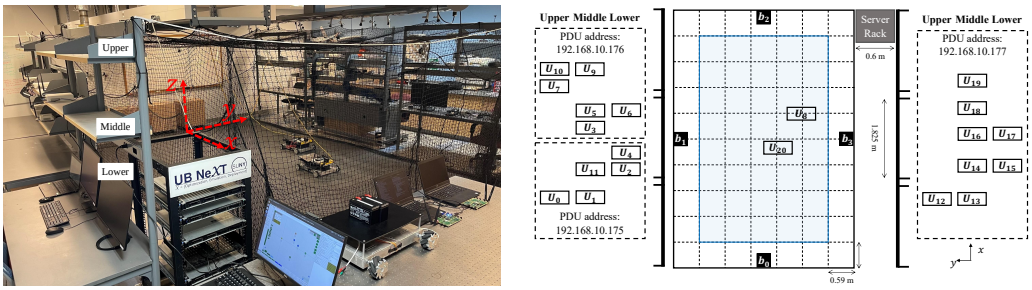
a whole. The network controller module (NCM) organizes the information from the NEM and each user API to define the network topology, environment, and control objective. The discrete event module (DEM) then takes the resulting full scenario definition and starts the discrete event-driven simulation process.

The simulator APIs offer full configuration over network behaviors, environment specification, and control specification. Specifically, the network configuration API provides control over parameters such as frequency, bandwidth, mobility, and location of nodes, as well as networking area and propagation characteristics. The environmental definition API provides control over the locations and sizes of blockages as well as their RF absorption coefficients over different frequency bands. In general, all physical environmental features, including lab benches, server rack, and UAV enclosure as shown in Figure 3(a), are modeled as blockages within the networking area. Finally, the custom algorithm API provides access to the run time behavior of all the nodes, such as mobility, transmission patterns, band association, among others. Particularly, this API module provides direct support for experimental applications of AI/ML for tasks such as network automation and self-configuration.

The parallel deployment of UBSim alongside the NeXT testbed provides several advantages. The highly configurable nature of UBSim provides a virtual sandbox in which experiments can be designed and evaluated for deployment on the NeXT testbed much faster than using SDR hardware alone. Additionally, the speed of simulation design and execution in UBSim enables pre-training or parallel training of AI models prior to deployment on hardware. This is particularly important for models in which significant amounts of environmental data must be available to generate an optimal solution, such as those used for deep learning and reinforcement learning. Furthermore, over-the-air data collected from the NeXT testbed can be used to improve the accuracy of data generated by UBSim by means of system identification [29], addressing challenges associated with high-quality data collection for AI/ML algorithms mentioned in Section 1.

3.2. Software-Defined Forwarding Infrastructure: RoboNet

The design objective of RoboNet is to support experiments in wireless networks with mobile robots, such as mobile hotspots [30] and wireless UAVs [31]. The testbed is located in 238 Davis Hall on the University at Buffalo's



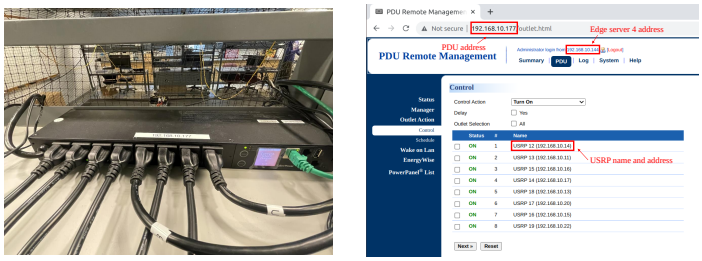
(a) (b)

Figure 3: (a) Snapshot of the RoboNet testbed; (b) RoboNet network topology.

North Campus. Figure 3 shows a snapshot of RoboNet and the corresponding topology. At the center of RoboNet is a netted enclosure of dimension $6 \times 4 \times 2.1$ m^3, providing a safe space for robot navigation. For mobile nodes, three wireless robots have been designed based on SuperDroid vehicles and universal software radio peripheral (USRP) SDRs. An indoor navigation system is also designed based on Marvelmind beacons to provide indoor localization for the robots. For static nodes, a set of USRP SDRs have been deployed over the shelves on the left and right sides of the netted enclosure. All the static software radios are controlled by a server rack of five Dell workstations. The mobile software radios are controlled by the robots' onboard computing hosts.

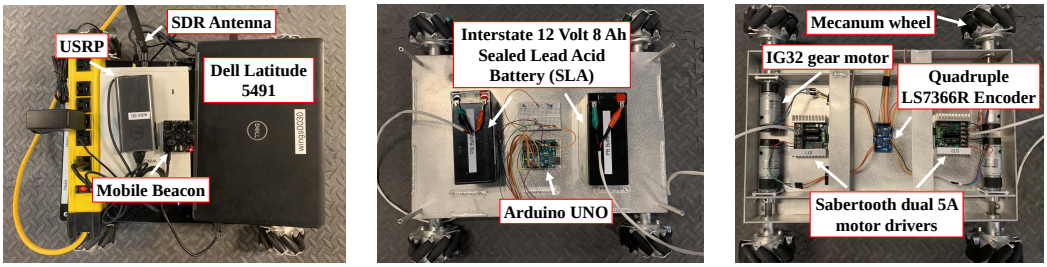
Static Nodes. The static nodes consist of 19 USRP N210, 5 USRP B210 SDRs and 1 wAP 60G (AP). Each USRP N210 operates at frequencies from DC to 6 GHz and can process up to 50 mega samples per second (MS/s). Each USRP N210 is equipped with a CBX daughterboard and two VERT900/VERT2450 antennas. These USRP SDRs are connected via two switches to a server rack, comprising four Dell EMC R340 PowerEdge workstations for baseband signal processing. Each USRP B210 is designed for low-cost experimentation with continuous frequency coverage from 70 MHz to 6 GHz. Each USRP B210 is also equipped with two VERT2450 antennas. The five USRP B210s provide flexibility because they can be deployed to any place depending on the requirements. The wAP 60G (AP) router is a product from MikroTik [32] and can be used either as a point-to-point primary or a point-to-multi-point primary.

The USRP SDRs are powered via three remotely accessible CyberPower Power-Distribution-Units (PDUs), as shown in Figure 4(a). These PDUs



(a) (b)

Figure 4: (a) Snapshot of PDU setup; (b) PDU remote management interface.



(a) (b) (c)

Figure 5: Snapshots of mobile node. (a) USRP software radio, control host, laptop, and mobile beacon; (b) Power unit and Arduino controller; and (c) Bottom view: motors, motor drivers and encoder.

are assigned with Ethernet LAN IP addresses 192.168.10.175, 192.168.10.176 and 192.168.10.177 and connected to edge servers via switches. By getting access to the three default IP addresses, experimenters can power on, shut down and make a schedule with all static USRPs remotely. Figure 4(b) shows the PDU remote management interface, via which experimenters can power on/off USRPs in real time or at scheduled times.

Mobile Nodes. Three software-defined robot vehicles have been designed for RoboNet based on a combination of SuperDroid robots and USRP SDRs. Snapshots of the robot vehicles are shown in Figure 5. The SuperDroid robot serves as the mobile carrier of the software radios. A programmable Mecanum wheel vectoring robot has been used in the current design of the mobile nodes. Each robot comprises 4 Mecanum wheels, 4 IG32 gear motors, 2 Sabertooth dual 5A motor drivers, 1 Quadruple LS7366R Encoder and 1 Arduino UNO controller. Each robot is powered by two

(a) (b)

Figure 6: (a) Controller modem; (b) Super beacon.

18V/2.4A PB (lead-acid) batteries. This allows each robot vehicle to carry up to 50 lbs of payload, including the USRP SDRs and their controlling host. Each robot is equipped with USRP SDRs for programmable wireless communications. Currently, both USRP N210 and B210 can be supported by mobile nodes. Each robot can also carry a wAP 60G to enable mmWave communications.

A Dell Latitude 5491 laptop with Intel Core^{TM}i7-8850H CPU@2.6GHz*12 is used for robot control, USRP SDR control and baseband signal processing. The connection between the controlling laptop and the robot vehicle is established by an Arduino via USB port "/dev/ttyACM0". The mobile beacon is connected to the laptop via USB port "/dev/ttyACM1". The two default serial ports provide more flexibility of our testbed. For example, by accessing the USB serial port, experimenters can access the raw beacon location information and design their own position algorithms, rather than using algorithms that we provide. Finally, the movement of the robot is controlled and navigated by the Arduino and the beacon via serial communications.

Indoor Positioning System. Because of the poor reception of GPS signals in indoor environments, an indoor positioning system has been deployed, as shown in Figure 6. The system consists of a controller modem (Figure 6(a)) and 7 precise (with accuracy of ± 2 cm) Marvelmind Super-Beacons (Figure 6(b)). Based on this system, the location of the mobile beacon can be calculated using trilateration based on the propagation delay of ultrasonic signals to a set of stationary beacons.

The 7 super beacons are divided into two groups: 4 static and 3 mobile beacons. As shown in Figure 3(b), the 4 static beacons, b_1, b_2, b_3 and b_4, are attached to the four sides of the protective net. For example, Figure 6(b) shows the deployment of b_1, which can communicate with the controller mo-

dem, its neighbour beacons and the mobile beacon using the selected frequency (19/25/31/37 kHz). According to the exchanged information among the static beacons, the mobile beacon and the modem, the robot locations will be updated in real time. We adopt a Non-Inverse Architecture to set up the navigation system and 31 kHz is used as the communication frequency.

Finally, the controller modem is connected to the edge server via a USB port. Through the control dashboard at the server, experimenters can define a network map by assigning the origin point of the 3D network, configuring beacon parameters (e.g., beacon address and mode), and monitoring the movements of the mobile beacons mounted on the robots.

Robot Self-Adjustment Scheme. Since we focus on investigating the wireless communication network, we always hope that the robot will move as prescribed and arrive at its target location. However, with inaccurate readings from the encoder and different speeds of the four Mecanum wheels (shown in Figure 5(c)), the robot may fail to arrive at the expected position and collisions may happen when multiple robots exist. In order to focus on the wireless network study itself without worrying about the negative impacts induced by the robot, we propose a beacon-based robot self-adjustment scheme to allow the robot to automatically adjust its position and posture during experiments. The overall robot self-adjustment scheme is summarized in Algorithm 1.

There are two phases of the self-adjustment scheme: i) beacon-based robot posture adjustment and ii) beacon-based robot position adjustment. Due to the different speeds of the Mecanum wheels, there is a divergence angle θ between the movement direction of the robot and the network's x-axis, as shown in Figure 7, especially when the robot moves left or right. At the beginning of the adjustment, the robot records its beacon-based position (x_1, y_1). Since movement errors are negligible when moving short distances forward or backward, we have the robot move forward for τ seconds ($\tau = 3$ by default), record its new beacon-based position (x_2, y_2), and then move backwards for τ seconds back to its original position (x_1, y_1). With the recorded two positions, the divergence angle θ can be calculated based on

$$\theta = \begin{cases} 90^\circ, & \text{if } x_1 = x_2 \text{ and } y_1 < y_2 \\ -90^\circ, & \text{if } x_1 = x_2 \text{ and } y_1 > y_2 \\ \arctan(\frac{y_2 - y_1}{x_2 - x_1}), & \text{otherwise.} \end{cases} \tag{1}$$

With the divergence angle θ, the movement option and the movement distance can be obtained by referring to Table 1, in which d_1 is the measured reference distance, which is obtained as follows: When a robot turns left or right, one of its four wheels (left-back wheel by default) does not move and the other three wheels do. By reading the encoder value of one of the non-static wheels (the left-front wheel by default) when the robot rotates 360°, the value of d_1 can be obtained. The two adjustment parameters (θ and d_1) will then be packed in a message and sent to the onboard Arduino controller. With the received message, the Arduino will control the robot to finish the beacon-based robot posture adjustment.

In the second phase, the robot first measures its new position (x_3, y_3) and compares it with the measurement-based beacon state information (x, y) which can be obtained via a one-time beacon-based measurement. The obtained distance divergence d_x and d_y for the x and y axis will be calculated and transformed to the corresponding movement direction and distance as shown in Table 1, in which d_2 and d_3 are the measured reference distance when the robot moves forward and backward for 1 meter, respectively. Similarly, the obtained adjustment parameters will be packed and sent to the Arduino. Once the Arduino receives the movement command, the robot will adjust its position and then finish the second-phase adjustment.

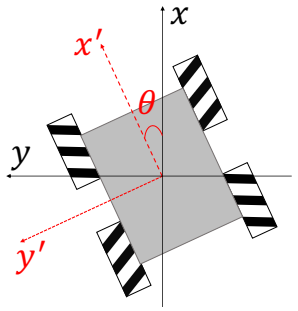


Figure 7: Mecanum wheel robot with angular deviation.

Table 1: Robot movement operation

Parameter	Movement Option	Movement Distance	Parameter	Movement Option	Movement Distance
$\theta > 0$	Rotate Left	$\lvert\theta\rvert/360 * d_1$	$\theta < 0$	Rotate Right	$\lvert\theta\rvert/360 * d_1$
$x_3 < x$	Move Forward	$d_x = (x - x_3) * d_2$	$x_3 > x$	Move Backward	$d_x = (x_3 - x) * d_2$
$y_3 < y$	Move Left	$d_y = (y - y_3) * d_3$	$y_3 > y$	Move Right	$d_y = (y_3 - y) * d_3$
Otherwise	Stop	0			

Algorithm 1: Robot Self-Adjustment Scheme

Beacon-based Robot Posture Adjustment:
- Measure current position (x_1, y_1) via beacon
- Robot moves forward for τ seconds
- Measure new position (x_2, y_2)
- Robot moves back to (x_1, y_1)
- Calculate the angle deviation based on (1)
- Determine the rotation direction and calculate rotation distance based on Table 2
- Arduino movement control

Beacon-based Robot Position Adjustment:
- Measure current position (x_3, y_3) via beacon
- Look up state position table and obtain target state position (x, y)
- Determine the movement direction and calculate movement distance based on Table 2
- Arduino movement control

4. Control Plane Design

The control plane supports both traditional model-based control, enabled by WNOS, and emerging data-driven control, enabled by the RL repository. A set of APIs are developed for WNOS to enable automatic generation of distributed cross-layer control algorithms. While the RL repository is combined with a set of experiment management APIs and multiple communication protocols to ease the use of NeXT testbed and enable broadband wireless communication. The control plane is deployed over the edge servers which are placed in the shelf labeled as "UB NeXT" in Figure 3(a).

4.1. Network Modeling and Optimization Support

It is typically tedious and error-prone to manually model and optimize forwarding infrastructure in the data plane. To address this challenge, we deployed our previously designed WNOS [11] over NeXT. The primary benefits of WNOS are that it abstracts the data plan forwarding infrastructure, allows experimenters to define control objectives in a centralized manner using high-level APIs, and then automatically generates distributed cross-layer control

algorithms that can be deployed on NeXT's data plane, e.g., *UBSim* and *RoboNet*. At a high level, WNOS comprises two key components: network abstraction and network control problem decomposition and control program generation. The network abstraction provides a set of APIs, based on which experimenters can characterize in a centralized manner the desired network behaviors before actual deployment. The network control problem decomposition and control program generation is enabled by *disciplined instantiation (DI)* [11], based on which user-defined abstract centralized network control problems can be decomposed into a set of distributed subproblems. WNOS is designed based on a three-level hierarchical architecture to enable scalable network deployment. Specifically, at the first-level, the WNOS control host is connected to all second-level SDR control hosts via wireless interfaces (Wi-Fi in our current prototype). The generated distributed algorithms are automatically pushed over the wireless interfaces and installed at each of the SDR control hosts which form the third-level. Hence, one only needs to create a single piece of code to control all the SDR devices.

WNOS supports a wide set of network control problems in both static and mobile networks. These include, but are not limited to, *rate maximization, power minimization, end-to-end delay minimization, and movement optimization*. WNOS also provides a rich set of APIs, based on which experimenters are allowed to define more sophisticated control problems in next-generation broadband networks spanning across multiple frequency bands, e.g., microwave, mmWave as well as THz bands. Below are some examples of the APIs.

Table 2: Example APIs of WNOS

API	Description
`attach(·)`	Add elements to the network
`connect(·)`	Link one or more network elements
`install_model(·)`	Install an expression model for a network element attribute
`get_expr(·)`	Get the expression of a network element
`mkexpr(·)`	Construct the new expression
`record_expr(·)`	Store the expression in the database
`set_para(·)`	Designate a specific expression as a utility function, constraint, or optimization variable
`set_soln(·)`	Select the solution method to optimize the designated variables
`record_expr(·)`	Store the expression in the database

4.2. Data-Driven Network Control Repository

The second part of the control plane is the data-driven network control repository which enables data-driven control on RoboNet and makes it easy

to modify advanced AI/ML algorithms to be compatible with our testbed. This repository consists of two classes of APIs for data-driven control, i.e., *Basic Class* and *Advanced Class*. The basic class is responsible for network initialization. Examples include the *Environment Initialization API, Variable Initialization API* and *Feedback List Initialization API*. The *Advanced Class* APIs are designed based on *Basic Class* and are used for policy training, including updating states, actions and a value table. Given the number of states and actions specified using the *Configuration API*, the environment can be initialized using the *Environment Initialization API*. Key variables involved in learning algorithms, such as the current state and next state, can be initialized via the *Variable Initialization API*. One is also allowed to choose the *Reward Type* and *Calculator Mode* through the *Configuration API*. Based on these APIs, four classes of RL algorithms have been implemented in the advanced class and can be called via the *RL Algorithm API*. These are epsilon-greedy search, upper confidence bound (UCB) action selection, Q-learning and State–action–reward–state–action (SARSA). Different reward types and calculator modes have been defined in advance, while experimenters can define custom reward types and calculator modes for their own experiments.

4.3. NeXT Experiment Management APIs

Extensive experiments can be conducted over the NeXT testbed, especially on *RoboNet* discussed in Section 3.2. To help experimenters use our testbed efficiently, we design a set of experiment management APIs, by which elements deployed on RoboNet can be coordinated. As shown in Figure 8, there are three classes of APIs, as discussed next.

Network Configuration APIs. APIs in this class are used to define various network environments. We provide three different APIs, *Network Configuration API*, *Host Configuration API* and *USRP Configuration API*. Parameters that can be configured via network configuration APIs include network area, center frequency, bandwidth, transmission power, modulation type, slot duration, the number of robots, etc. Through host and USRP configuration APIs, experimenters can manage Ethernet addresses, wireless network addresses, and port numbers for the SDRs and their controlling hosts.

Nodes Synchronization APIs. To easily coordinate the server and mobile hotspot controllers, Nodes Synchronization APIs are provided. With

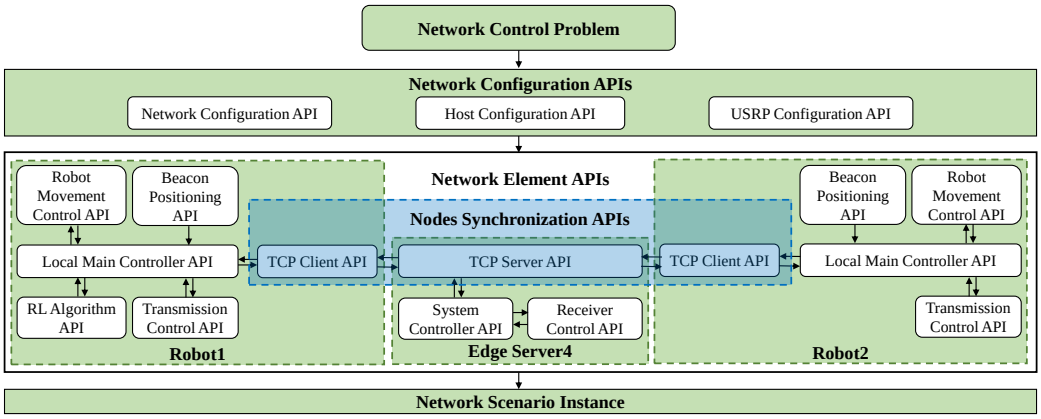
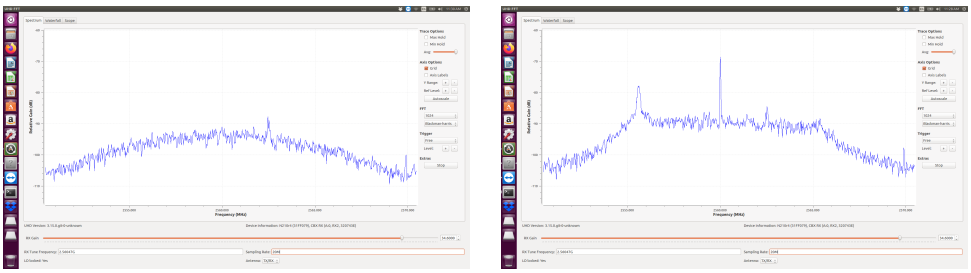


Figure 8: Network element control interface and experiment management APIs.

these APIs, for example, experimenters can start the experiments with just one command executed on the edge server.

These APIs are based on a Transmission Control Protocol (TCP) connection established over WiFi to provide communications among different nodes. The WiFi wireless local area network is enabled by TP-Link Archer A7 AC1750 Wireless Dual Band Gigabit Router, which follows wireless LAN 802.11a/b/g/n/ac standards. The 2.4 GHz and 5 GHz bands are dedicated for node synchronization and we avoid using the two bands for conducting experiments. Thus, the WiFi will not cause any interference to our target experiments. The other potential external interferes are mainly from wireless devices like phones. However, since most wireless devices get access to the internet via University at Buffalo's WiFi network, which also works in the 2.4 GHz and 5 GHz bands, the interference to measurements is limited. In Figure 9, we show the spectrum comparison without and with ongoing experiments, respectively. The results show that the possible interference (-92 dBFS) to our experiments is much smaller than our signal strength (-68 dBFS). Thus we can neglect the possible interference.

Network Element APIs. After the experiment profile has been configured, one can further control various network components via a set of system control APIs deployed at the edge server and mobile hotspot controller. These include the *Transmission Control API*, which can be used to control the transmissions of the USRP N210 carried by the robot vehicle; the *Receiver Control API* for controlling data receiving; the *Robot Movement Control API* for controlling robot movement; and finally the *Beacon Positioning API*, based on which experimenters can obtain the robots' real-time



(a) (b)

Figure 9: Screenshot of 2.56GHz spectrum monitor of network (a) in idle mode with -92 dBFS peak interference; (b) during experiments with -68 dBFS peak signal.

positions.

In these network element APIs, logging features are enabled to record system status like transmission process startup, beacon positioning updates, robot movements and so on. Data that will be used for analysing and processing, like throughput for each time slot, are stored and updated in dictionaries/tables during the tests and saved automatically once an experiment finishes.

```
1  import Reinforcement_Learning_API as rli
2  import Robot_Movement_Control_API as rmi
3  import Beacon_Positioning_API as bpi
4
5  while experiment_running:
6      curt_state = bpi.operation()
7      next_state = rli.operation()
8      updt_rbt_movement_ctrl(curt_state, next_state)
9      updt_usrp_commn_ctrl()
10     updt_fdbk_request()
11     reward = rli.fdbk_processing(fdbk, fdbk_type, rwd_calc)
12     rli.updt_value_table(reward)
13     if rbt_adjustment_status:
14         rmi.rbt_adjustment(next_state)
```

Listing 1: Example of Experiment Management APIs

In Listing 1, we show an example of using the aforementioned APIs to conduct experiments on the NeXT testbed. While an experiment is running (line 5), the user calls *Beacon Positioning API* to get the current state information (line 6) and calls *Reinforcement Learning API* to get the next state

information (line 7). The robot updates its location by calling *Robot Movement Control API* (line 8). After the robot arrives at the target location, the communication begins (line 9). After a pre-defined communication time in *Network Configuration API*, the robot requests feedback (line 10) and obtains the current reward (line 11), and the value table is then updated (line 12). Before conducting the next time-slot experiment, the robot adjustment status parameter defined in *Network Configuration API* will be checked (line 13). If the status is *True*, the robot will adjust its posture and position based on Algorithm 1 (line 14).

4.4. Communication Protocols Management

We consider three different communication protocols in our testbed: GNU Radio Benchmark, srsRAN and mmWave communication protocols.

GNU Radio Benchmark Protocol. This is developed based on GNU Radio narrow-band benchmark library [33]. Specifically, we extend the original benchmark narrow-band library by designing three additional APIs. These are *Benchmark Interaction API*, *Benchmark Transmission Control API* and *Benchmark Receiving Monitor API*. For example, the *Benchmark Transmission Control API* is used to control when and what data is transmitted. The transmission duration and transmission information can be configured in *Network Configuration API* in Section 4.3 and can then be transmitted to the basic benchmark module via *Benchmark Interaction API*. *Benchmark Receiving Monitor API* is used to monitor the status of the receiver. If the receiver detects disconnected links, it will restart the transmitter by sending a request to the edge server via *Benchmark Interaction API*.

```
1  import Network_Configuration_API as ncfg
2  import Host_Configuration_API as hcfg
3  import Benchmark_Interaction_API as bmia
4
5  def Benchmark_Transmission_Control_API():
6
7      if bmia.tmr1 == 0:
8          data = ncfg.cxn_data
9      elif bmia.tmr2 <= ncfg.ts_len:
10         data = ncfg.comm_data
```

```
11      else:
12          data = ncfg.cxn_data
13          bmia.socket.sendto(ncfg.cmd_done, (hcfg.wl_host, hcfg
                .port))
14
15      return data
```

Listing 2: Example of *Benchmark Transmission API*

Listing 2 shows an example of how *Benchmark Transmission Control API* is used to control the transmission of data at run time. There are two types of data that can be transmitted: regular *transmission data*, which is the actual data that we want to deliver over the network, and dummy *connection data*, which we use to keep the network connection alive. The connection data is needed because GNU Radio does not provide an auto re-connection scheme if a connection is lost. In Listing 2, the experimenter first calls the *Benchmark Transmission Control API* (line 5) to determine the data to be transmitted based on the two timers received from *Benchmark Interaction API*. If timer 1 (*bmia.tmr1*) equals 0 (line 7), the data is set as connection data (*ncfg.cxn_data*, line 8); if timer 2 is (*bmia.tmr2*) smaller than a predefined transmission duration (*ncfg.ts_len*, line 9), the data is set as communication data (*ncfg.comm_data*, line 10); otherwise, the data is set to *ncfg.cxn_data* (line 12) and the *one-time-slot-finished* information will be sent to *Local Main Controller API* via *Benchmark Interaction API* (line 13).

Software-Defined RAN Protocol. This is developed based on srsRAN, an open-source 4G and 5G software radio suite developed by Software Radio Systems (SRS) [34]. It contains three different modules, srsEPC, srsENB and srsUE. We design a set of *srsRAN Configuration APIs* to manage the three modules based on the parameters in *Network Configuration API*. For example, the user dataset information can be generated automatically and stored in "user_db.csv" via *srsEPC configuration API*, and srsENB operation parameters like communication frequency can be generated automatically via *srsENB configuration API*. The *srsUE configuration API* is used to generate "ue.conf" file which contains srsUE operation information, such as IMSI information.

```
1 import os
2 import srsEPC_configuration_API as epca
3 import srsENB_configuration_API as enba
4 import srsUE_configuration_API as suea
5
```

```
6 epca.srsepc_operation()
7 os.system("gnome-terminal␣--␣bash␣-c␣\"sudo␣srsepc;␣exec␣bash
    \"")
8 enba.srsenb_operation()
9 os.system("gnome-terminal␣--␣bash␣-c␣\"sudo␣srsenb;␣exec␣bash
    \"")
10 suea.srsue_operation()
11 os.system("gnome-terminal␣--␣bash␣-c␣\"sudo␣srsue;␣exec␣bash
    \"")
```

Listing 3: Example of *srsRAN Configuration APIs*

Listing 3 shows an example of how to generate srsRAN configure files and run the corresponding programs. Users call *srsEPC_operation* (line 6) to generate "user_db.csv" and start up srsEPC program in line 7. Similarly, "enb.conf" and "ue.conf" are generated by calling *srsENB_operation* (line 8) and *srsUE_operation* (line 11), respectively.

Millimeter Wave Communication Protocol. The mmWave communication protocol is supported by MikroTik mmWave routers [32]. These mmWave routers can be configured to form a point-to-point network or point-to-multi-point network based on requirements. When integrating the mmWave communication protocol with srsRAN to enable large scale wireless network communication, we connect both the USRP B210 (running srsEPC and srsENB) and the mmWave router (primary) to a single laptop and design a *Gateway Setting API* to navigate data traffic between srsUE and mmWave subordinate.

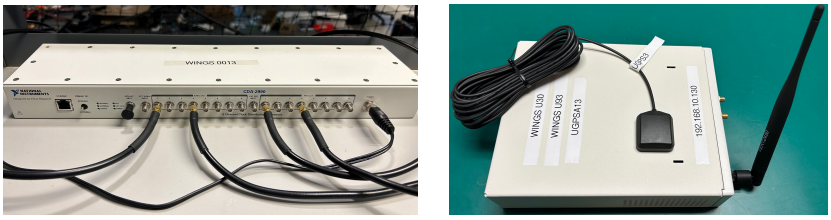
```
1 import os
2 import Network_Configuration_API as ncfg
3 import Host_Configuration_API as hcfg
4
5 def Gateway_Setting_API():
6
7     subnet = hcfg.srsRANsubnet
8     eth_addr = getattr(hcfg, ncfg.laptop_name).get("eth_host"
        )
9     gw_setting_cmd = "gnome-terminal␣--␣bash␣-c␣\"sudo␣ip␣
        route␣add␣" + str(subnet) + "␣via␣" + str(eth_addr) +
        "␣;␣exec␣bash\""
10    os.system(gw_setting_cmd)
```

Listing 4: Example of *Gateway Setting API*

Listing 4 shows an example of how to set the gateway to enable communication between srsUE and mmWave subordinate. The key is to get the subnet address of srsRAN (line 8) and Ethernet address of the current laptop (line 9). The command is generated based on the above two information (line 10) and the gateway is set by executing the command (line 11).

Since the three communication protocols have their own logic stacks and the interactions with controllers are processed in different ways, we implement three different profiles for each communication protocol. These three profiles are stored in laptop controllers and edge servers. Each profile is independently stored in different folders but they share the same components except communication protocol. Experimenters can choose the profile to load, i.e., select the communication protocol they want to use, before conducting experiments. By providing different profiles for different communication protocols, we can easily integrate more communication protocols, like direct sequence spread-spectrum (DSSS) [35], to our testbed in the future.

The three communication protocols that we have implemented all rely on self-synchronization schemes. For example, srsRAN implements the Precision Time Protocol (PTP), which is a standard protocol used for time synchronization in packet-based networks, so no clock/time synchronization features are needed. However, protocols that rely on clock/time synchronization can be implemented with support of additional hardware, such as an Octoclock or GPS module, as shown in Figure 10. The accuracy of Octoclock is within a few 100 ns and GPS module is within 50 ns.



(a) (b)

Figure 10: (a) Snapshot of the Octocolock; (b) GPS module

5. Example Experiments over NeXT

We now test NeXT and showcase its capabilities of optimization, simulation and experimentation considering different network control problems.

These include user scheduling in a cellular network, trajectory optimization for a mobile hotspot, and joint rate and power control in multi-hop networks. A comprehensive overview of these experiments is summarized in the Table 3.

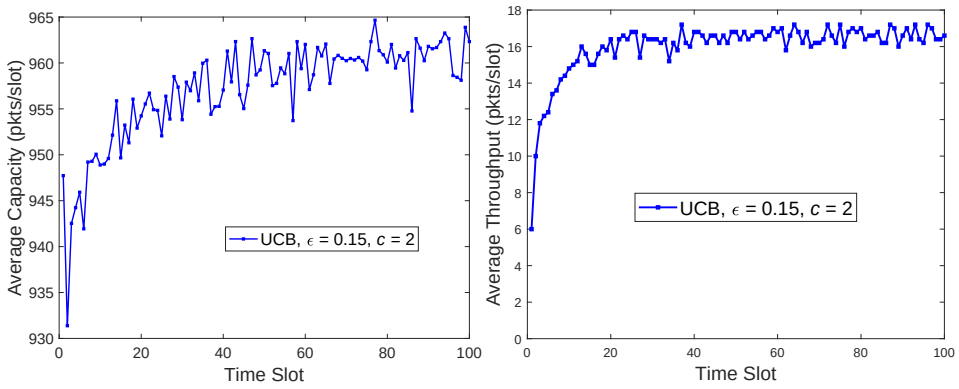
Table 3: Experiments Overview

Experiment No.	**Name**	**Type**
1	User Scheduling	simulation & experimentation
2	Overflow Control	experimentation
3	Mobile Hotspot Navigation	experimentation
4	Multi-Mobile Hotspots Navigation	experimentation
5	Mobile Hotspot Navigation with srsRAN	experimentation
6	Mobile Hotspot Navigation in IAB	experimentation
7	Multi-hop Network Optimization	optimization & simulation & experimentation

5.1. Experiment 1: User Scheduling

In the first experiment, we consider a wireless network with a hotspot serving a set of users. The transmission time is divided into a set of consecutive time slots. In each time slot, we consider that the hotspot can serve at most one user. The objective of the hotspot is to maximize the aggregate throughput by selecting a user to serve in each time slot. We design control algorithms for the hotspot based on the data-driven network control repository as discussed in Section 4.2. Specifically, we consider the upper confidence bound (UCB) action selection algorithm and test it over both UBSim and RoboNet developed in Section 3. First, we test the effectiveness of the UCB algorithm in UBSim. Figure 11(a) plots the achievable capacity averaged over 20 episodes each with 100 time slots. It can be seen that the average capacity improves over time, and this validates the effectiveness of the data-driven network control repository.

Then we further test the data-driven network control repository over RoboNet considering SDRs and real-world wireless channels. USRP20 is selected as the transmitter and five USRPs (USRP2, USRP5, USRP9, USRP11 and USRP19) are selected as receivers (see Figure 3). The time slot duration is set to 3 seconds. The exploration parameter ϵ and UCB control parameter c are set to 0.15 and 2, respectively. We run 10 episodes of robot navigation, with each episode consisting of 100 time slots. We calculate the average number of received packets in each time slot and the results are shown in Figure 11(b). It can be seen that the highest throughput can be achieved in around 20 time slots. This further validates the effectiveness of the data-driven network control repository. Comparing Figures. 11(a) and



(a) (b)

Figure 11: User scheduling scenario: (a) Average capacity obtained over UBSim; and (b) average throughput using RoboNet.

(b), we found the average capacity in UBSim is much larger than the average throughput on RoboNet. This is because we use different protocols in each system. In UBSim, the network capacity is calculated based on the Shannon capacity formula while on RoboNet, the throughput is obtained based on GNU Radio's narrowband communication protocol. Besides, the transmission power, bandwidth and so on are different in UBSim and RoboNet. For these reasons, the gap between UBSim and RoboNet is large. Since we focus on the verification of algorithms effectiveness, we neglect the gap between the simulator and reality. However, it would be interesting to investigate how to mitigate the reality gap, which is also a potential function provided by the NeXT system. We discuss this further in Section 6. Recall that the primary objective of NeXT is to provide an integrated environment for optimization, simulation, and experimentation in software-defined wireless networks. This experiment illustrates the benefits of using the NeXT testbed. Conducting experiments in the real-world is time consuming while simulation-based experiments can be done much quicker. With the NeXT testbed, users can test their algorithm in UBSim first and check the performance of the proposed algorithm. If the results show that the algorithm needs improvement, they do not need to do the tests in the real-world, which saves time. By conducting experiments on our testbed, users also avoid directly collecting data in the real-world, which can sometimes be unsafe.

```
usrp_rx_list = ["usrp2","usrp5","usrp9","usrp11","usrp19"]
ts_len = 3                  # time slot length; unit: second
ts_nim = 100                # time slot number
```

```
4
5 Q_Epsilon = 0.15
6 Q_StepSize = 0.2
7 Q_DiscountRate = 0.95
```

Listing 5: User scheduling configuration parameters in *Network Configuration API*

Listing 5 shows user scheduling configure parameters in *Network Configuration API*, which can be used to configure the parameters involved in this experiment. Experimenters can specify the USRPs they want to use as the receiver in line 1. The time-slot length and time-slot number for each episode can be set via line 2 and line 3, respectively. The parameters used for the UCB action selection algorithm configuration can be configured from line 5 to line 7.

5.2. Experiment 2: Overflow Control

In this scenario, Robot1 is adopted as the transmitter and three USRPs (USRP1, USRP9 and USRP11) are adopted as receivers. Each time slot is set to 5 seconds and the transmitter transmits data every 0.15 seconds. Assume that the arrival rate follows a Poisson distribution and the average arrival rate is set to 1 packet per time slot. The maximum data buffer length is set to 8 packets for each receiver. Q-learning is adopted in this case to control data buffer overflows. As shown in Figure 12, we run one episode with 500 time slots and calculate the number of transmitted packets and the corresponding running average for each time slot. It can be seen that, in some time slots the number of transmitted packets is 0. This happens when there are no packets available in the buffer to transmit or when the channel conditions are bad. For Poisson distributed packet arrivals with average arrival rate 1, the expected number of packets arriving at each user in each time slot is 1. Thus, the total expected arrivals for three users in each time slot is 3. From Figure 12, the running average is around 3 packets per time slot which matches the above mathematical analysis. The cumulative overflows are shown in Figure 13, in which the slope converges gradually over time towards the optimal achievable packet overflow rate.

```
1 pkts_arrival_rate = 1                 # unit: packet/slot
2 queue_max_pkts_num = 8                # maximum buffer size
3 queue_overflow_reward = -20
```

Listing 6: Overflow control configuration parameters in *Network Configuration API*

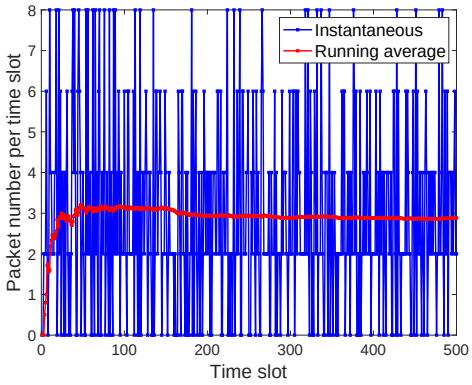


Figure 12: Overflow control scenario: Transmitted packet number vs. time slot

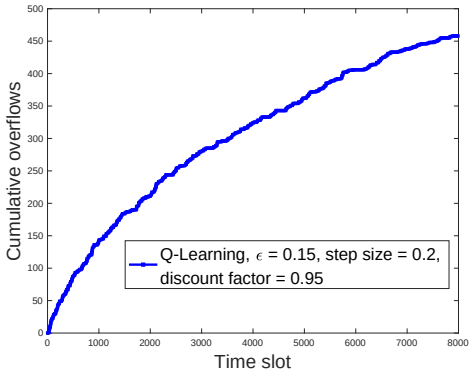


Figure 13: Overflow control scenario: Cumulative overflows vs. time slot

As shown in Listing 6 experimenters can modify the parameters in *Network Configuration API* to meet their needs. Experimenters can modify the packet arrival rate and maximum buffer length for each user via line 1 and line 2, respectively. Line 3 can be configured to set an overflow reward (such that a negative value corresponds to a penalty).

5.3. Experiment 3: Mobile Hotspot Navigation.

In the third experiment, we consider a wireless network where a robot carrying a mobile hotspot moves around to serve a set of users. The objective is to maximize the users' aggregate throughput by controlling the robot's trajectory. The network is divided into a set of grid cells, each corresponding to a state of the environment. In each grid cell, the robot has five action options, i.e., move forward, move backward, move left, move right and stay. The reward for each state-action pair is defined as the sum throughput of

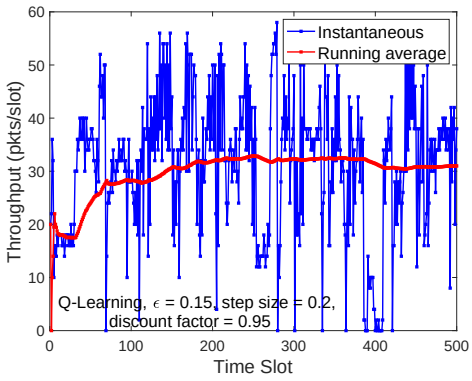


Figure 14: Single mobile hotspot scenario: instantaneous and running average of throughput.

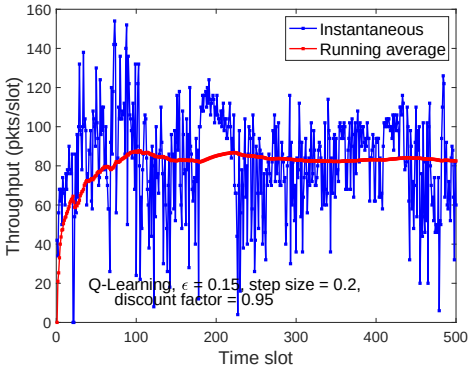


Figure 15: Two mobile hotspots scenario: instantaneous and running average of throughput.

users. Q-learning is considered in this experiment with exploration probability ϵ set to 0.15, step size of 0.2 and discount factor 0.95. Each episode consists of 500 time slots, corresponding to 3 hours. We measure the number of received packets and calculate the corresponding running average in each time slot. The experimental results are reported in Figure 14. It can be seen that the running average converges to around 30 packets/slot. The drop of instantaneous throughput around time slot 400 is caused by the imperfection of the wireless link, which got disconnected as the robot moved.

5.4. Experiment 4: Multi-Mobile Hotspots Navigation.

In the fourth experiment, we consider the same wireless network scenario as the third except that we adopt two mobile hotspots. To avoid collisions,

the network is divided into two regions and each robot can only move within one region. Five USRPs (USRP0, USRP1, USRP2, USRP3 and USRP19) are configured as users to receive service from the two robots. In each time slot, a user is only allowed to connect to the robot with the shortest distance to it. Q-learning with the same parameters as in the second experiment is adopted. Similarly, we measure the number of correctly received packets and calculate the corresponding running average in each time slot. The experimental results are reported in Figure 15. It can be seen that the running average converges to around 80 packets/slot.

```
1  import struct
2  import Network_Configuration_API as ncfg
3  import Beacon_Positioning_API as bpi
4  import Host_Configuration_API as hcfg
5
6  def update_rbt2_state():
7
8      curt_state = bpi.operation()
9      s_addr = hcfg.rbt2.get("code")
10     d_addr = hcfg.rbt1.get("code")
11
12     data = struct.pack('!H', s_addr & 0xffff) + struct.pack('
           !H', d_addr & 0xffff) + struct.pack('!H', curt_state &
            0xffff) + ncfg.rbt2_state_info
13     rb2_tcp_skt.sendto(data, ((hcfg.rbt1.get('host'), hcfg.
           rbt1.get('port'))))
```

Listing 7: Example of multi-robots interaction

In Listing 7 we give an example showing how Robot2 sends its state information to Robot1 during the experiments. Robot2 first gets its current state information via interaction with *Beacon Positioning API* (line 8). By calling *Host Configuration API*, the message source code (line 9) and message destination code (line 10) are obtained for message routing. Then the data is constructed (line 12) and sent to Robot1 via *TCP Client API* (line 13).

In the above four experiments, we adopt the GNU Radio Benchmark communication protocol. In the following two experiments, we adopt srsRAN (and mmWave) as the communication protocol.

5.5. Experiment 5: Mobile Hotspot Navigation with srsRAN

Similar to *Mobile Hotspot Navigation*, we want to maximize the users' aggregate throughput by controlling a robot's trajectory but with a different

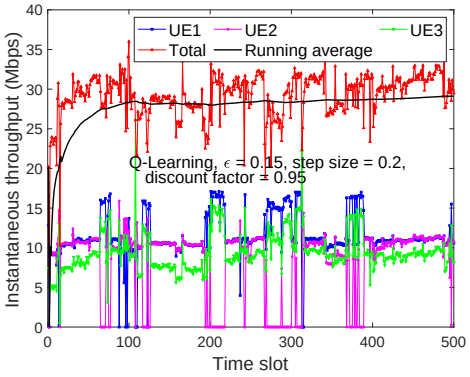
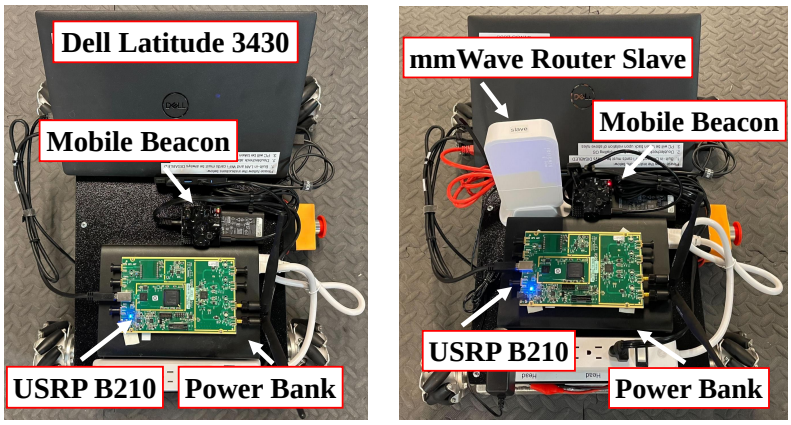


Figure 16: Single mobile hotspot scenario: instantaneous and running average of throughput.

communication protocol, namely, srsRAN. The robot carries a USRP B210 which works as a base station to serve three users (each being a USRP B210). The three USRP B210s are located at the position of USRP0, USRP5 and USRP14 as shown in Figure 3(b), respectively. In each time slot, we use *iperf3* to measure instantaneous throughput for 3 seconds and calculate the corresponding average throughput. The results are shown in Figure 16. It can be seen that the total running average converges to around 28 Mbps. The drop of instantaneous throughput of UE2 near the 200th time slot results from it losing its connection and thus leads to increased throughputs of UE1 and UE3. We can also find that the three UEs can achieve similar throughput if no connections are lost. This is because we set the srsRAN MAC layer scheduling mechanism to proportional fair (PF), which aims to balance system throughput and fairness. Based on the PF scheduling mechanism, UE's with relatively better instantaneous channel quality indicator (CQI) compared to their historic average rates will be allocated with more resources. Experimenters can also choose a round-robin scheduling method by specifying it in *Network Configuration API* at the MAC layer.

```
import os
import Host_Configuration_API as hcfg
import srsUE_configuration_API as suea

ue_name = suea.get_srsRANue_name()
ue_port = getattr(hcfg, ue_name).get("port")
iperf3_server_cmd = suea.cmd_gen(ue_port)
```



(a) (b)

Figure 17: (a) Snapshot of srsRAN based-robot; and (b) Snapshot of IAB based-robot.

```
8 os.system(iperf3_server_cmd)
```

Listing 8: Example of starting *iperf3* server

Listing 8 shows how to generate an *iperf3* server on srsUE side. Firstly, users need to obtain the UE name (line 5) and the predefined port number (line 6). Then *iperf3* server command is generated (line 7) and executed (line 8) to run the *iperf3* server, which is waiting for *iperf3* client connection from the client side (i.e., the robot).

5.6. Experiment 6: Mobile Hotspot Navigation in IAB

The sixth experiment is mobile hotspot navigation in an integrated access and backhaul (IAB) network setting. As shown in the Figure 17(b), a robot carries a USRP B210 and a mmWave router slave as a relay to bridge three users and the base station (mmWave router primary). The three users are located at the positions of USRP0, USRP5 and USRP19 and the mmWave primary is located at (7.1 m, 0 m, 1.5 m) in the RoboNet network. Q-learning is adopted to optimize the robot trajectory. The results of the experiments are shown in Figure 18. Similarly, the running average throughput converges to 22 Mbps.

In the above experiments, the RL algorithms are adopted to improve network performance. However, we found that sometimes RL does not work well

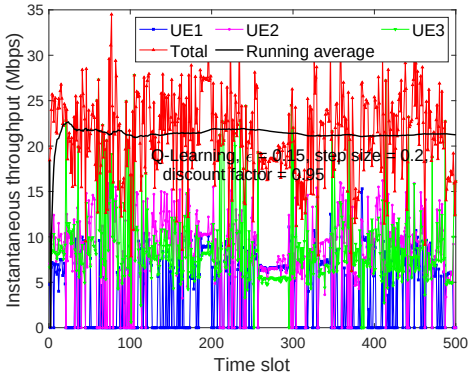


Figure 18: Single mobile hotspot scenario in IAB setting: instantaneous and running average of throughput.

in practice. Randomness (like the user disconnection, time-varying channel, movement of robot and so on) could affect the RL performance. For example, during the mobile hotspot navigation in IAB experiments, the robot spends most time in a state (state 6) after 200 time-slots but the running average throughput is lower as shown in Figure 18. This could be due to the laptop's limited computation capabilities, which can degrade the USRP B210s performance. In this case, if we want to apply RL in a wireless network, we need to take randomness into consideration and design a more sophisticated reward (not simply taking throughput as the only one criteria).

5.7. Experiment 7: Multi-hop Network Optimization

In the seventh and final experiment, we consider a multi-session multi-hop network with two sessions and eight nodes. Each session consists of four nodes, namely one source node, two relay nodes and one destination node. The objective is to maximize the network throughput while minimizing the interference between the two sessions by jointly optimizing the physical and transport layers. The optimization algorithms are generated automatically by WNOS, which has been deployed over the control plane of NeXT, as described in Section 4. The resulting algorithms are deployed over the data plane. Similar to the *User Scheduling* experiment discussed above, we conduct this experiment over both UBSim and RoboNet. The results are reported in Figure 19. We can see that the control algorithms converge over both UBSim and RoboNet. It is worth pointing out that different link models have been considered in UBSim and RoboNet in their current implementations. In future research, we will create a digital twin of RoboNet based on

UBSim and test the gap between simulated and real-world performance.

6. New Research Topics Enabled by NeXT

In this section we discuss the new research topics that NeXT can enable, including *sim-to-real transfer learning*, *robust wireless network control*, *online digital twin construction and optimization*, and *multi-agent reinforcement learning*.

Sim-to-real transfer learning: Towards zero-touch wireless network self-configuration, the proposed framework will connect accelerated learning in the virtual domain with performance evaluation in the real domain. With the proposed framework, novel machine learning algorithms can be designed and tested rapidly in the virtual domain in a variety of configurable networking scenarios, and the converged algorithms can be deployed on SDR hardware for practical evaluation. Making use of a digital twin for initial policy iteration can significantly reduce the time required to generate an optimal control policy, especially in the case of deep learning or deep reinforcement learning. These transfer learning experiments will be used to understand the performance discrepancy between simulation and hardware evaluation, which will be necessary for designing repeatable experiments towards accelerated learning for wireless network self-configuration. This investigation into efficient transfer learning will start with experimental benchmarks to quantify the reality gap between UBSim and RoboNet and then designing methods to minimize the impact of this gap through an experimental campaign of domain adaptation and novel twin-domain learning algorithms.

Robust wireless network control: The use of robust learning for domain adaptation in the wireless domain has been introduced in [27]. By introducing noise to the training data or training environment during policy iteration, it has been shown that the resulting control policy will provide improved performance when faced with unexpected observations or perturbations compared to a non-robust policy. In this line of research, this uncertainty can be interpreted as the set of all physical phenomena which contribute to the performance gap between simulation and hardware scenarios, such as unpredictable RF interference or hardware nonlinearities. The programmable SDR hardware provided by the RoboNet testbed coupled with the virtualization of the RoboNet environment in UBSim enables investigation into robust learning to improve sim-to-real transfer learning performance in a wide variety of networking scenarios, with or without knowledge of the reality gap. We

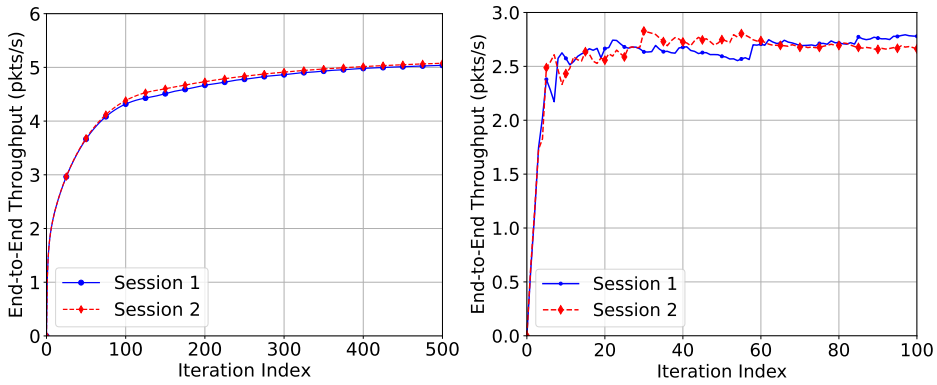


Figure 19: Multi-hop network optimization scenario: Average end-to-end throughput with (a) UBSim simulator and (b) NeXT testbed.

plan to build on findings in [27] by applying the experimental robust learning framework to the sim-to-real capabilities presented in this work, exploring robust learning as a method of mitigating performance degradation due to sim-to-real policy transfer.

Online digital twin construction and optimization: Existing methods for generating a virtual model for digital twin applications typically rely on human expertise, and can be tedious and error-prone. This motivates autonomous virtual environment construction based on mobile sensing techniques such as simultaneous localization and mapping (SLAM). Using SLAM with remote-control hardware such as the robots introduced in Section 3, it is possible to record observations and generate a 3D environment map with configurable fidelity in real time without significant human intervention. This capability can significantly accelerate the digital twin construction process by automating the collection and import of environmental data into the desired simulation environment, such as UBSim. With integrated simulation and experimentation capabilities, the NeXT testbed can enable research of online digital twin construction by providing configurable network simulation environments in UBSim, and verifying the accuracy of the autonomously generated digital twin with ground truth obtained through testbed experiments.

Multi-agent Reinforcement Learning (MARL): The NeXT testbed can support MARL research for development and evaluation of algorithms such as REINFORCE policy gradient (PG)[36], gradient-based partially observable

MDP (G(PO)MDP)[37], actor-critic (A2C)[38], or asynchronous actor-critic (A3C)[39]. In general, these algorithms require significantly more time to converge to an optimal policy than their single-agent counterparts. Additionally, debugging MARL algorithms can be complicated due to the distributed nature of data collection and processing. The architecture of UBSim and its supporting APIs can significantly simplify the simulation design process by streamlining user-configurable parameters such as the number of nodes, distributed or centralized control algorithms, and reward function related to the environment. This can save time, provide configurable online feedback to display only target data points, and limit redundancy in coding for large-scale MARL problems. Finally, the configurable SDR topology and the hardware available in the RoboNet testbed can provide a framework through which simulation results obtained in the virtual digital twin environment can be verified through real-world experiments.

7. Conclusions and Future Work

In this work, we introduced the software-defined testbed *NeXT*, which enables integrated simulation, experimentation and optimization for wireless research. We designed the data plane with both the simulator *UBSim* and the testing facility *RoboNet*. We designed the control plane in which a software toolchain is developed to support both traditional model-based and new data-driven control techniques. We presented the communication protocols deployed on our testbed. We verified the effectiveness and flexibility of *NeXT* considering both simulation and testbed experiments. We also discussed the new research topics that can be enabled by *NeXT*. In future work, we will *i) enable experiments in flying networks by integrating UAVs into NeXT; ii) enable digital twin for testing self-optimizing networks; and iii) allow remote access to the NeXT platform via CloudRAFT, a cloud-based framework for remote access of experimentation platforms that has been developed at the University at Buffalo [40].*

References

[1] A. Gosain, Platforms for Advanced Wireless Research: Helping Define a New Edge Computing Paradigm, in: Proc. of Technologies for the Wireless Edge Workshop, New Delhi, India, 2018.

[2] J. Breen, A. Buffmire, J. Duerig, K. Dutt, E. Eide, M. Hibler, D. Johnson, S. K. Kasera, E. Lewis, D. Maas, A. Orange, N. Patwari, D. Reading, R. Ricci, D. Schurig, L. B. Stoller, J. Van der Merwe, K. Webb, G. Wong, POWDER: Platform for Open Wireless Data-Driven Experimental Research, in: Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization, London, United Kingdom, 2020.

[3] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, C. Gutterman, Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless, in: Proc. of the 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, 2020.

[4] M. L. Sichitiu, I. Guvenc, R. Dutta, V. Marojevic, B. Floyd, AERPAW Emulation Overview, in: Proc. of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization, London, United Kingdom, 2020.

[5] L. Bonati, S. D'Oro, M. Polese, S. Basagni, T. Melodia, Intelligence and Learning in O-RAN for Data-Driven NextG Cellular Networks, IEEE Communications Magazine 59 (10) (2021) 21–27.

[6] J. Hu, S. K. Moorthy, A. Harindranath, Z. Guan, N. Mastronarde, E. S. Bentley, S. Pudlewski, SwarmShare: Mobility-Resilient Spectrum Sharing for Swarm UAV Networking in the 6 GHz Band," , in: Proc. of IEEE International Conference on Sensing, Communication and Networking (SECON), Virtual Conference, 2021.

[7] Z. Shi, S. He, J. Sun, T. Chen, J. Chen, H. Dong, An efficient multi-task network for pedestrian intrusion detection, IEEE Transactions on Intelligent Vehicles 8 (1) (2023) 649–660.

[8] Y. Mi, D. Mohaisen, A. Wang, AutoDefense: Reinforcement Learning Based Autoreactive Defense Against Network Attacks, in: 2022 IEEE Conference on Communications and Network Security (CNS), Austin, TX, USA, 2022.

[9] F. Wen, M. Qin, P. Gratz, N. Reddy, Software Hint-Driven Data Management for Hybrid Memory in Mobile Systems, ACM Transactions on Embedded Computing Systems 21 (1) (2022) 1–18.

[10] F. Tian, Y. Zhang, W. Ye, C. Jin, Z. Wu, Z.-L. Zhang, Accelerating Distributed Deep Learning Using Multi-Path RDMA in Data Center Networks, in: Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR), Virtual Event, USA, 2021.

[11] Z. Guan, L. Bertizzolo, E. Demirors, T. Melodia, WNOS: Enabling Principled Software-Defined Wireless Networking, IEEE/ACM Transactions on Networking 29 (3) (2021) 1391–1407.

[12] A. Gosain, Platforms for Advanced Wireless Research: Helping Define a New Edge Computing Paradigm, in: Proceedings of the 2018 on Technologies for the Wireless Edge Workshop, New Delhi, India, 2018, p. 33.

[13] L. Bonati, S. D'Oro, S. Basagni, T. Melodia, SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems, in: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services, Wisconsin, USA, 2021.

[14] R. Doost-Mohammady, O. Bejarano, L. Zhong, J. R. Cavallaro, E. Knightly, Z. M. Mao, W. W. Li, X. Chen, A. Sabharwal, RENEW: Programmable and Observable Massive MIMO Networks, in: 2018 52nd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, USA, 2018, pp. 1654–1658.

[15] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, P. Ruth, FABRIC: A National-Scale Programmable Experimental Network Infrastructure, IEEE Internet Computing 23 (6) (2019) 38–47.

[16] J. Mirkovic, T. Benzel, Deterlab Testbed for Cybersecurity Research and Education, Journal of Computing Sciences in Colleges 28 (4).

[17] R. Zhao, T. Woodford, T. Wei, K. Qian, X. Zhang, M-cube: A millimeter-wave massive MIMO software radio, in: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, 2020, pp. 1–14.

[18] S. Wang, J. Huang, X. Zhang, Demystifying millimeter-wave V2X: Towards robust and efficient directional connectivity under high mobility, in: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, 2020, p. 677–690.

[19] A. Song, X. Hong, F. Zhang, Z. Peng, Z. Wang, Mu-Net: Community-shared infrastructure for mobile underwater acoustic networks, The Journal of the Acoustical Society of America 150 (4) (2021) A197–A198.

[20] NSF Testbeds, `https://nets-vo.org/resoures/nsf-testbeds/`.

[21] CISE Community Research Infrastructure, `https://www.ccrivo.org/projects/`.

[22] L. Bertizzolo, L. Bonati, E. Demirors, T. Melodia, Arena: A 64-Antenna SDR-Based Ceiling Grid Testbed for Sub-6 GHz Radio Spectrum Research, in: Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization, Association for Computing Machinery, Los Cabos, Mexico, 2019, p. 5–12.

[23] R. K. Sheshadri, E. Chai, K. Sundaresan, S. Rangarajan, SkyHaul: An Autonomous Gigabit Network Fabric in the Sky, ArXiv abs/2006.11307.

[24] P. Sen, D. A. Pados, S. N. Batalama, E. Einarsson, J. P. Bird, J. M. Jornet, The TeraNova Platform: An Integrated Testbed for Ultra-Broadband Wireless Communications at True Terahertz Frequencies, Computer Networks 179 (2020) 107370.

[25] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, J. Beutel, FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems, in: 2013 ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Philadelphia, USA, 2013, pp. 153–165.

[26] R. Trüb, R. D. Forno, T. Gsell, J. Beutel, L. Thiele, A Testbed for Long-Range LoRa Communication: Demo Abstract, in: Proceedings of the 18th International Conference on Information Processing in Sensor Networks, Montreal, Canada, 2019, pp. 342–343.

[27] M. McManus, Z. Guan, N. Mastronarde, S. Zou, On the Source-to-Target Gap of Robust Double Deep Q-Learning in Digital Twin-Enabled Wireless Networks, in: Proc. of SPIE Conference Big Data IV: Learning, Analytics, and Applications, Orlando, Florida, 2022.

[28] SimPy, https://pypi.org/project/simpy/.

[29] Y. Jiang, T. Zhang, D. Ho, Y. Bai, C. K. Liu, S. Levine, J. Tan, SimGAN: Hybrid Simulator Identification for Domain Adaptation via Adversarial Reinforcement Learning, in: Proceedings of the 2021 IEEE International Conference on Robotics and Automation, Xi'an, China, 2021, pp. 2884–2890.

[30] S. Barrachina-Muñoz, B. Bellalta, E. W. Knightly, Wi-Fi Channel Bonding: An All-Channel System and Experimental Study From Urban Hotspots to a Sold-Out Stadium, IEEE/ACM Transactions on Networking 29 (5) (2021) 2101–2114.

[31] J. Buczek, L. Bertizzolo, S. Basagni, T. Melodia, What is A Wireless UAV? A Design Blueprint for 6G Flying Wireless Nodes, in: Proc. of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation CHaracterization (WiNTECH'21), New Orleans, LA, USA, 2022.

[32] mmWave Router, https://mikrotik.com/product/wap_60g.

[33] GNURadio Benchmark, https://github.com/n4hy/gnuradio/tree/master/gr-digital/examples/narrowband.

[34] srsRAN, https://www.srslte.com/.

[35] G. Sklivanitis, A. Gannon, K. Tountas, D. A. Pados, S. N. Batalama, S. Reichhart, M. Medley, N. Thawdar, U. Lee, J. D. Matyjas, S. Pudlewski, A. Drozd, A. Amanna, F. Latus, Z. Goldsmith, D. Diaz, Airborne Cognitive Networking: Design, Development, and Deployment, IEEE Access 6 (2018) 47217–47239.

[36] T. Zhang, H. Wen, Y. Jiang, J. Tang, Deep Reinforcement Learning Based IRS for Cooperative Jamming Networks under Edge Computing, IEEE Internet of Things Journal.

[37] R.-T. Ma, Y.-P. Hsu, K.-T. Feng, A POMDP-Based Spectrum Handoff Protocol for Partially Observable Cognitive Radio Networks, in: 2009 IEEE Wireless Communications and Networking Conference, Budapest, Hungary, 2009.

[38] T. Niu, Y. Teng, Z. Han, P. Zou, An Adaptive Device-Edge Co-Inference Framework Based on Soft Actor-Critic, in: 2022 IEEE Wireless Communications and Networking Conference (WCNC), Austin, TX, USA, 2022.

[39] H. Zhou, Z. Wang, H. Zheng, S. He, M. Dong, Cost Minimization-Oriented Computation Offloading and Service Caching in Mobile Cloud-Edge Computing: An A3C-based Approach, IEEE Transactions on Network Science and Engineering.

[40] S. K. Moorthy, C. Lu, Z. Guan, N. Mastronarde, G. Sklivanitis, D. Pados, E. S. Bentley, M. Medley, CloudRAFT: A Cloud-based Framework for Remote Experimentation for Mobile Networks, in: Proc. of IEEE International Workshop on Communication and Networking for Swarms Robotics (RoboCom), Virtual Conference, 2022.