

Received October 7, 2019, accepted November 10, 2019, date of publication November 19, 2019, date of current version December 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2953356

Wipi: A Low-Cost Large-Scale Remotely-Accessible Network Testbed

ABDELHAMID ATTABY^{1,4}, NADA OSMAN², MUSTAFA ELNAINAY^{2,3}, (Senior Member, IEEE), AND MOUSTAFA YOUSSEF², (Fellow, IEEE)

¹Wireless Research Center, Egypt Japan University of Science and Technology, New Borg El Arab 21934, Egypt

²Department of Computer and Systems Engineering, Alexandria University, Alexandria 21544, Egypt

³Information Systems Department, Faculty of Computer and Information Systems, Islamic University of Madinah, Medina 42351, Saudi Arabia

⁴(on leave) Faculty of Engineering at Shoubra, Benha University, Banha 13518, Egypt

Corresponding author: Abdelhamid Attaby (abdelhamid.rabia@ejust.edu.eg)

This work was supported in part by a grant from the Egyptian National Telecommunication Regulatory Authority (NTRA).


ABSTRACT The high cost of establishing a network experimental lab obstructs researchers from fulfilling and validating their research proposal. Remotely accessible testbeds overcome this difficulty by allowing researchers to access the testbed and the attached expensive wireless devices through the Internet. In this paper, we introduce WiPi as a low-cost networking testbed that can be utilized remotely and supports large-scale experiments. WiPi is implemented from the available off-the-shelf computing nodes, such as standard laptops and Raspberry Pis, with the goal to be affordable to many institutions, especially in developing countries. Multiple features, including users' isolation, disk protection, ease of user experience, power efficiency, multiple application domains, efficient disk utilization, and resource pooling, are implemented as part of the testbed. The interface and functionality of WiPi target three different levels of researchers in terms of their research experience: Expert, users with no prior knowledge with ns3, and novice. Besides, WiPi can combine simulation, emulation, and experimentation over real devices in the same experiment to further support larger-scale experiments. The web interface of the testbed allows researchers to partition and map a virtual network with a large number of virtual nodes to a physical network with a limited number of real nodes. Evaluation results show that WiPi can be utilized by a wide range of researchers and can support different networking applications. Furthermore, it can reduce the execution time of large-scale experiments by almost 40%, highlighting its suitability as a low-cost, large-scale remotely-accessible network testbed.

INDEX TERMS Network experimentation, wireless networks, remotely accessible testbeds, low-cost testbeds, network emulations.

I. INTRODUCTION

¹Technologies in the field of wired and wireless networks are evolving rapidly, leading to the emergence of new protocols and standards. However, these protocols and standards need to be tested and validated by their inventors and other researchers. Typically, this testing and validation can be achieved by multiple approaches, including networks simulations, emulation, and/or implementation on real devices.

Researchers often prefer simulations at the beginning of their work and when they want to validate their results and idea initially. This is due to the low-cost and ease of use of simulators. For instance, ns-2 [2], ns-3 [3] and OPNET [4]

The associate editor coordinating the review of this manuscript and approving it for publication was Wenchi Cheng .

¹An earlier version of this paper appeared in the proceedings of the 2018 IEEE 87th Vehicular Technology Conference (VTC Spring) [1].

provide scalable, easy and modular tools for researchers to build simulated wired or wireless networks.

Emulations, on the other hand, can take the validation of the results to the next level by providing manageable and reproducible environments while mitigating the experiments with real applications. For instance, CORE [5], EMPOWER [6] and ns-3 [3] provide tools for researchers to build emulated wired or wireless networks. Each node in these networks can be managed like a separate node with full control over its TCP/IP layers. Nonetheless, emulations; similar to simulations; do not provide the experimenters with real device nor practical conditions to test their work which, in turn, affect the research ability to match the real network. Furthermore, dealing with such emulators requires some experience to configure the system parameters and its nodes.

Implementation on real devices is the optimal choice to handle real conditions on different devices under practical scenarios. However, these implementations have multiple

obstacles including the high cost of the real devices, the overhead of setting up the hardware and software on a scale and the experience required to run the desired experiment on the installed testbed efficiently.

Recently, researchers have begun to rely on the remotely accessible testbeds in testing their research over real devices. Testbeds such as CRC [7], CORNET [8], CogFrame [9], ORBIT [10], NITOS [11], and OneLab [12], have gained momentum as they allow researchers to validate their results through pre-installed, pre-configured and open testbeds that can be used from anywhere through the Internet. The main architecture of these testbeds consists of a number of processing nodes that are connected to external networking devices, e.g., RF devices, and managed by extra software tools for configuration and collection of the results. Managing the testbed is implemented using an abstract interface layer that can be used to manage the users and deployed experiments. These leading testbeds, though allowing a wide range of researchers to validate their work, results, and systems in realistic environments, still have space for improvements to scale to new users, especially in developing countries where cost is the main concern.

In this paper, we present WiPi as a remotely-accessible testbed which has been implemented with particular design goals including leveraging low-cost devices, utilizing heterogeneous devices, resource pooling, and supporting large-scale experiments through mixing simulation and emulation. The testbed leverages commodity hardware devices such as standard laptops/desktops and Raspberry Pis (RPIs) as low-cost computing devices. Besides, it leverages low-cost RF nodes such as standard WiFi adapters and low-cost Software Defined Radios (SDRs) for communication. Also, it can virtualize the testbed nodes and mix emulation and simulation in the same experiment to support large-scale networks. Moreover, it allows resource pooling to separate the computing nodes (i.e., laptops or RPIs) from the communication nodes (i.e., WiFi, low-cost SDRs, or USRPs) to efficiently utilize the available resources.

WiPi facilitates experimentation of large-scale networks through two techniques: node virtualization and combining simulation, emulation, and implementation on real devices in the same experiment. In node virtualization, we multiply the number of testbed nodes by virtualizing each physical node to several virtual nodes to increase the testbed resources. On the other hand, combining simulation, emulation, and experimentation on real devices in a single experiment enables evaluation of practical applications and protocols over a wide range of wired and wireless network scenarios while achieving scalability of the testbed. For instance, in some experiments, researchers are interested in probing and evaluating the behavior of a part of the network (sub-network) in a system consisting of a large number of nodes. Executing the entire scenario on real devices can be resource exhaustive and of little value, as the user is not interested in the exact behavior of every single node but the outcome of the big network on a subset of nodes. For such scenarios, the researcher may

implement the part of the interest on some of the WiPi nodes, emulate the links between them and simulate the rest of the experiment.

WiPi is designed to support three different levels of researchers' experience: Expert, users with no prior knowledge with ns3, and novice. Expert users can utilize the testbed devices directly without using the testbed utilities. Users that have no prior knowledge about ns-3 [3], the used emulation software in the testbed, can use the code generator utility provided by WiPi to map a drawn network typology to scripts required by the testbed nodes to execute. Finally, novice users with low experience can use the testbed to partition and map a network topology to several testbed nodes and monitor the results.

The WiPi architecture uses a controller server that runs the cControl and Management Framework (OMF) experiment controller that manages and controls the testbed nodes [13]. In addition, the controller server runs a web portal that manages researchers' reservations, authentications and authorizations, a VLAN control module to isolate concurrent users in a separated environment, a power control module that controls powering on/off the Raspberry Pi nodes to add a control layer over the Raspberry Pi nodes, a disk imaging module to save and manage users' images, and some pre-configured scripts that can be used by researchers for the execution of large-scale experiments. In addition, as part of its design, WiPi needs to address a number of specific challenges related to the low-cost and heterogeneous hardware including users' isolation, disk images handling, ease of use, the implementation of large-scale experiments' support, and power efficiency.

We evaluate WiPi from three perspectives: the power of the low-cost devices in the middle of live experiments, the performance of combining simulation and emulation in the same experiment, and the performance of executing experiments using WiPi. The evaluation using heterogeneous devices and interfaces shows that the throughput of the low-cost nodes can support a wide range of wireless networking applications. The evaluation of combining the simulation, emulation, and implementation on real devices in the same experiments against pure simulation or emulation only shows that this approach can reduce the total execution time of experiments by almost 40% as compared to simulating the same experiments on a single physical node.

The rest of this paper is organized as follows: Section II discuss related work. Our design goals and architecture are explained in Section III. The implementation and different execution scenarios of large-scale experiments are discussed in Section III. The evaluation of the WiPi testbed is detailed in Section IV. Finally, we conclude and discuss future directions in Section VI.

II. RELATED WORK

In this section, we discuss the public wired and wireless remotely accessible testbeds that are open for researchers to sign up and use the hardware devices installed by the testbeds' operators. We will also cover the related work to overcome

TABLE 1. A features comparison between WiPi and other testbeds.

	Cost	Heterogeneity	Resources Pooling	Power Efficiency	Storage Management	Disk Protection	Applications Domain	Large-Scale Support	Testbed Usability
Emulab [21]	High	High	✓	×	✓	N/A	Diverse	✓	Usable
ORBIT [10]	High	Medium	✓	×	✓	N/A	Diverse	✓	Uncomfortable
CRC [7]	Medium	Low	×	×	✓	N/A	Limited	×	Usable
NITOS [11]	High	High	✓	×	✓	N/A	Diverse	✓	Usable
PlanetLab [29]	High	High	✓	×	✓	N/A	Diverse	✓	Comfortable
QOMB [28]	Low	Low	×	×	×	N/A	Limited	×	Uncomfortable
EmPower [30]	Medium	Low	×	×	×	N/A	Limited	×	Uncomfortable
WiPi	Low	Medium	✓	✓	✓	✓	Diverse	✓	Comfortable

the main limitation of using real hardware devices: the lack of scalability, reliability, and consolidation.

A. REMOTELY ACCESSIBLE NETWORKING TESTBEDS

One of the most popular wired and wireless testbeds, is ORBIT [10]. It consists of 20×20 grid of processing nodes. Each node is connected to a software-defined radio device (SDR). The researchers can reserve the testbed to perform their experiments. ORBIT is using OMF to manage controller servers and other resources. ORBIT depends on relatively high-cost nodes and devices.

CORNET [8] is another wireless testbed that consists of 48 software-defined radio nodes. The researchers interested in the wireless field, especially in the algorithms of Cognitive Radios (CR) and its applications, can use CORNET to perform their tests and validate their results. The testbed still has a lack of important functionalities such as a disk imaging system and a reservation system.

NITOS [11] presents a testbed built from multiple wireless interfaces to support experimentation with heterogeneous (Wi-Fi, WiMAX, LTE, Bluetooth) wireless technologies in different environments (e.g., indoor, outdoor). It consists of 100 nodes (some of them are mobile). Similar to ORBIT [10], it relies on high-cost nodes and devices to build a NITOS testbed.

CRC [7] is another wireless testbed that is using the OMF framework, Frisbee disk imaging system, a reservation system, and other features. However, it does not support large-scale experiments and still require expensive hardware.

Federated testbeds is another class of testbeds in which multiple testbeds located in different sites can be integrated to facilitate cross-domain experiments. Fed4Fire [14] is a project that is open, remotely accessible, and introduces reliable facilities that supports a wide variety of diverse Internet research areas. GENI (Global Environment for Networking Innovation) [15] is an open infrastructure that facilitates research and education for large-scale networks and distributed systems. FIBRE [16] is a testbed that connects local laboratories together to form a virtual network laboratory to be accessible by students and researchers.

WiPi, on the other hand, is implemented using heterogeneous low-cost devices. Besides, it facilitates large-scale wired and wireless experiments. The homogeneity in the

installed operating systems across all the different heterogeneous used devices improves the user experience. Besides, the dynamic network topology that is created on-demand utilizes the testbed resources more efficiently and increases the accessibility of the testbed expensive devices. Furthermore, the storage management system of WiPi offers a set of pre-defined standard disk images, allows saving user images, and utilizes the time and bandwidth needed to load such images to multiple nodes. Finally, the interface and functionality of WiPi support different level of users in terms of their research experience.

A comparison between the features implemented in WiPi and other testbeds is summarized in table 1.

B. APPROACHES TO SCALE NETWORKING EXPERIMENTS

In this section, we explore the different approaches used to perform large-scale experiments and emulate wireless environments, including simulation, node virtualization, network emulation, and wireless emulation.

1) NETWORK SIMULATION

Network simulation is one of the most effective evaluation methodologies in the area of computer networks, mainly for the development of new communication architectures and network protocols. Network simulators allow researchers to model computer networks by defining both the behavior of the network nodes and communication channels. Popular simulators such as ns-2 [2], ns-3 [3], OMNeT++ [17], and OPNET [4] has been used by a wide range of researchers.

Ns-2 [2] is one of the early efforts in this field. Network simulations using ns-2 require researchers to write a C++ code to model the behavior of the simulation nodes. Also, they need to write oTcl scripts to control the simulation and specify additional aspects, for example, the network topology. However, a significant shortcoming of ns-2 is its limited scalability in terms of simulation time and memory consumption. This shortcoming affects the usability of ns-2 because new research domains in the field of computer networks, such as peer-to-peer networks, wireless sensor networks (WSNs), and grid architectures, require the simulation of extensive networks, probably with thousands of nodes.

Ns-3 [3] is the successor of ns-2 that overcomes the shortcomings of ns-2. Network simulation using ns-3 can be

performed entirely using C++ code only while Python code can be used to release parts of the simulation. Ns-3 models are not compatible with the ns-2 simulator, so, they to be ported to ns-3 manually.

OMNeT++ [17] is a general-purpose discrete event-based simulator that is applied mainly in the network simulation domain. Network simulation using OMNeT++ can be performed using C++ code to generate so-called simple modules. Simple modules can be merged using NED, the network description language of OMNeT++, to form so-called compound modules. NED supports writing variable parameters, e.g., the number of nodes, in the network description. In this case, the modules representing the nodes can be dynamically instantiated by the simulator during the time of running.

OPNET (Optimized Network Engineering Tools) [4] is a discrete event commercial network simulator. Network simulations in OPNET are performed using its GUI interface to define a network topology and to configure parameters starting from the application layer to the physical layer. OPNET uses a predefined set of devices and protocols. Therefore, researchers cannot test new protocols nor alter the behavior of existing ones.

In summary, simulation tools offer a flexible and scalable way to produce reasonably detailed physical and link models and to repeat and control target network conditions at the user's requirement. Whereas they are beneficial in evaluating specific solutions, they become inaccurate and computationally unmanageable when used for cross-layer evaluation of applications running on large-scale networks. As the TCP/IP protocol stack is written from scratch in most of the simulation tools, it does not precisely emulate real-world protocol stacks, which can significantly affect performance.

2) NODE VIRTUALIZATION

In this approach, the virtual nodes are hosted on the physical nodes of the network infrastructure. Each virtual node can be implemented using one of the virtualization software such as OpenVZ [18], Xen [19] or VMware [20]. For large-scale experiments, various virtual nodes can be defined, established, and mapped onto the corresponding resources of the testbed. Users can define the number of nodes as well as the connectivity/topology of the different virtual and physical nodes. Each virtual node has its own private filesystem, process hierarchy, network interfaces, and IP addresses, and set of users and groups. The parameters of the connection channel between different virtual and physical nodes (e.g., packet loss, bandwidth, and delay) have to be implemented to emulate wireless channels. Furthermore, the user has to model, calculate, and implement the effect of map position, e.g., GPS coordinates, of each wireless node over the channels between nodes. Therefore, the researcher has to define and implement all these parameters. As an example, Emulab [21] uses the concept of virtual nodes to involve a large number of nodes in the same experiment by allowing testbed users to control virtual nodes using either Emulab commands or Xen Project Management API (XAPI)

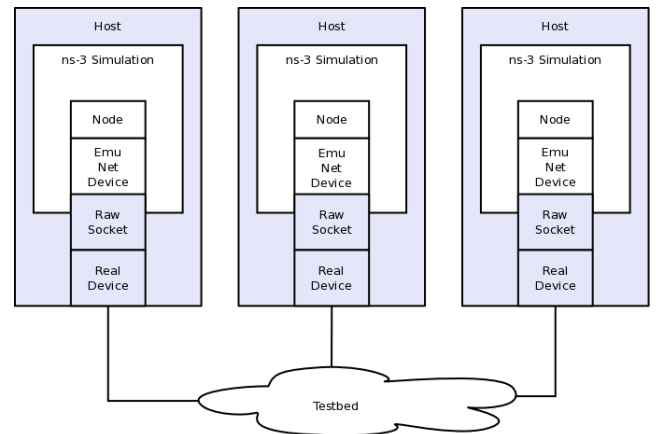


FIGURE 1. An example of three physical nodes of the same testbed and each node has its own ns-3 simulated network and connected together through the Emu NetDevice type [3].

commands [22]. Emulab uses the concept of node virtualization to mainly support the researchers that target the wired network experiments.

ORBIT [10], as another example, uses virtual machines to facilitate the on-demand creation of additional nodes to allow researchers to run non-performance critical back-end software, various controllers, or monitoring and reporting tools. Similarly, CRC [7] used the concept of system virtualization to allow slicing the physical node into static virtual nodes with predefined specifications (disk space, RAM, network interfaces, etc.). Each virtual machine has exclusive access to one of the wireless interfaces attached to the physical node.

In summary, the node virtualization approach has the flexibility of having a number of virtual nodes implemented over one physical node. However, it suffers from requiring the user to manually emulate the wireless channel characteristics between physical or virtual nodes.

3) NETWORK EMULATION

This approach extends the node virtualization technique to add more control over the whole network, including the wired and wireless devices. Users can draw the physical and the logical topology of the network, define the nodes parameters and their locations, and control each virtual node separately. The software automatically defines the channel characteristics based on the locations of the user-defined nodes and the parameters specified by the users. For example, the Common Open Research Emulator (CORE) [23] is a software that can emulate networks on one or more physical machines. Emulated networks can be connected to real networks through physical interfaces.

Ns-3 [24], as another example, has been designed to integrate with testbeds and virtual machine environments. ns-3 supports two kinds of network devices. The first kind, called Emu NetDevice, allows the nodes in the simulation to send data on the physical network. The second kind, called Tap NetDevice, allows physical nodes to participate

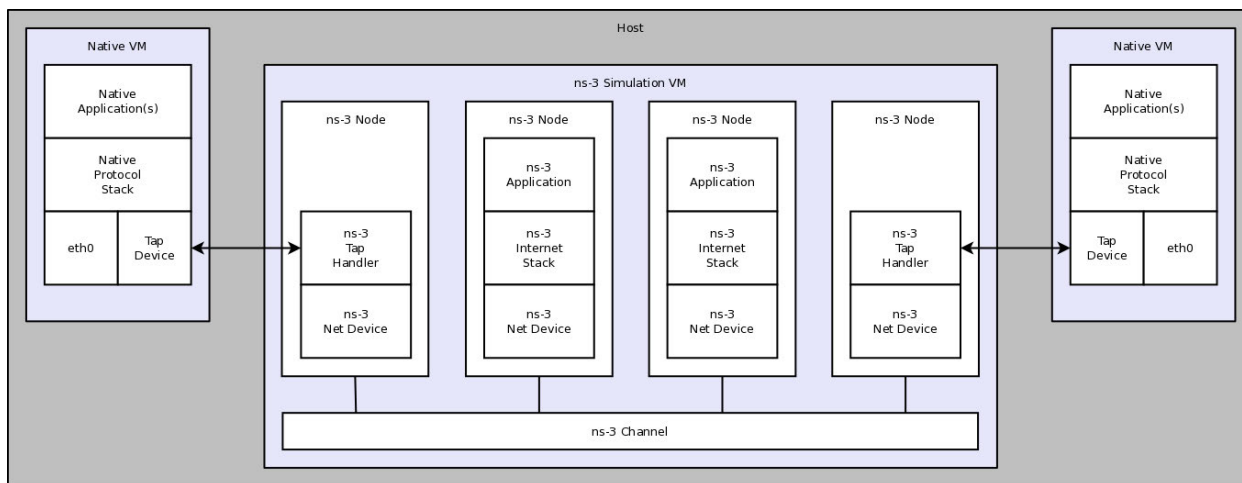


FIGURE 2. An example of connecting virtual machines to the ns-3 simulation virtual machine on the same physical node through the Tap NetDevice [3].

in the simulation as if they were one of the simulated nodes. The simulation can include a combination of the two kinds. Figures 1 and 2 illustrate the Emu NetDevice and Tap NetDevice in the ns-3 simulation respectively. Expert researchers can integrate ns-3 with ORBIT [10] and Emulab [21] testbeds in their experiments to benefit from ns-3 simulation and emulation capabilities.

To further support large-scale experiments, a virtual network with a large number of simulated nodes can be mapped to a physical network with a small number of physical nodes. This mapping operation varies from testbed to another based on the definition of the cost of each physical node. For example, in EmuLab [21], a random search method is used to find the mapping. EmuLab divides the search into two stages: a minimum graph cut to initially partitioning the topology to smaller topologies and then solving the mapping problem for each sub-topology. By random search, each simulated node is assigned to an available physical node, assuming all available physical nodes are used in the mapping. For each assignment, the performance of the mapping is evaluated, and the mapping that gives the best performance is used. However, the consumed time by random search could always be a problem for large topologies.

MaxiNet [25] proposed an emulation technique to mix simulation and emulation using a set of physical nodes. They used the METIS [26] graph partitioning technique to partition the topology over the available physical nodes. The minimum graph cut is found based on edges weights and nodes weights. MaxiNet proposes to set edges weights to the capacity of the links in the topology and nodes weights to the cardinality of the nodes. This way, the topology will be partitioned so that less communication is used in the emulated links.

In WiPi, we use the two kinds of network devices provided by ns-3 to mix simulation and emulation at the same time in order to support large-scale experiments. We take into consideration the different specifications, the demand rate,

and the maximum rate of real links of each physical node in the mapping operation. In addition, WiPi facilitates the execution of these experiments by automating the generation of ns-3 scripts for non-expert researchers.

To sum up, the network emulation approach has both the benefits of virtualization and wireless channel characterization. It also integrates physical nodes and virtual nodes into the same experiment to achieve the user need for monitoring the behavior of one or more nodes in the middle of large-scale networks.

4) WIRELESS EMULATION

In this subsection, we will cover the different approaches used to achieve emulated **wireless** environments, including radio propagation emulators and channel emulators. The approaches used in this subsection are used in the experiments where multiple wireless communications and channels need to be tested and evaluated.

a: RADIO PROPAGATION EMULATORS

In this type of wireless emulation, computers with only wired connectivity behave as if they have wireless connectivity. A controller creates a network quality degradation description, which corresponds to the real-world events to emulate the wireless network features (e.g., network delays, bandwidth availability, and packet loss). For example, QOMET [27] implemented a two-stage scenario-driven technique for wireless network emulation to convert a real-world scenario to a sequence of network-condition descriptors that support a wide range of experiments.

QOMB [28], NITOS [11], PlanetLab [29] and EmPower [30] testbeds use these emulators to emulate wireless connections in experiments.

In summary, this type of wireless channel emulation requires a central software that maps the nodes virtual locations, channels noise, and other parameters that affect the

TABLE 2. The virtualization and emulation approaches provided by the current testbeds.

	Node Virtualization	Network Emulation	Radio Propagation Emulators	Channel Emulators
Emulab [21]	✓	✓	×	✓
ORBIT [10]	✓	✓	×	✓
CRC [7]	✓	×	×	×
NITOS [11]	×	×	✓	×
PlanetLab [29]	×	✓	✓	×
QOMB [28]	×	×	✓	×
EmPower [30]	✓	×	✓	×
WiPi	✓	✓	×	×

wireless channel into quality degradation effects. The software also needs to have control over the physical interfaces of different nodes to apply these effects.

b: CHANNEL EMULATORS

In this type of wireless emulation, testbeds use Field Programmable Gate Array (FPGA) to emulate radio wave interference and multiple-input and multiple-output (MIMO) channel-like software-based discrete event simulators. For example, in [31] the emulator takes in the signals generated by wireless network cards through the antenna port, subjects the signals to the same effects that occur in a real physical space (e.g. attenuation, multi-path fading, etc), and feeds the combined signals back into the wireless cards while FPGAs transform the signals using realistic signal propagation models.

ORBIT [10] and Emulab [21] are typical examples of testbeds that use wireless channel emulation devices to support researchers need to test sophisticated wireless channel characteristics. Through time, the demand for these devices raises to test new protocols and technologies. Recently, many commercial wireless channel emulators appeared on the market to ensure accurate testing of sophisticated technologies such as LTE, HSPA, HSPA+, EV-DO, WLAN, and WiMAX.

In conclusion, this type of wireless channel emulation need special hardware or devices to add the wireless channel characteristics and parameters to the connections between different physical nodes.

Table 2 provides a brief comparison of the virtualization, network emulation, and wireless emulation implementation in different testbeds. One testbed may implement different techniques at the same time to provide wide choices to their users.

The WiPi testbed, as compared to testbeds mentioned above, considered the most cost-effective testbed. The low-cost of RPi and its ability to be connected with external sensors through GPIO pins make the testbed flexible to extension and supportive to next-generation applications, especially in Internet of Things (IoT) and wireless fields, and large-scale experiments. It also presents added features such as a reservation system, disk imaging system, users isolation, device pooling, and others.

**FIGURE 3.** Example of a Raspberry Pi node connected to a USRP1 via a USB interface in WiPi.

III. THE WIPi TESTBED

This section introduces the WiPi testbed including the testbed design goals and the proposed architecture.

A. DESIGN GOALS

1) LOW COST

We designed the testbed with low-cost devices to make the final cost of building similar testbeds affordable as well as scale the testbed to a larger number of nodes. For example, a single RPi node costs around \$25 - \$35. Similarly, the testbed uses off-the-shelf laptops to act as powerful processing nodes in case computational-heavy experiments are needed. In addition, one workstation acts as a server that holds the required services and software used to operate the necessary packages to run the testbed. Figure 3 shows an example of a typical node in the testbed, where a Raspberry Pi node is connected to a USRP1 device through a USB cable.

2) HETEROGENEITY

We utilized off-the-shelf heterogeneous devices to establish a useful testbed that supports the wireless research community and inspires builders of testbeds and network labs with the ability to handle heterogeneous devices in the same testbed. The proposed testbed contains different types of devices, including Raspberry Pis 3 Model B, laptops, and a workstation.

3) BETTER USER EXPERIENCE

The testbed users may face difficulties when dealing with different operating systems to run their experiments and collect the results. We designed the testbed with an eye on

improving the user experience because it is hard for users to learn different coding styles to deal with different operating systems. One of the main features in the proposed testbed is the homogeneity in the installed operating systems across all the different heterogeneous devices used. We use Raspbian, Ubuntu and Debian operating systems for Raspberry Pi nodes, laptop nodes, and the controller server, respectively. All the mentioned operations systems are Debian-based, which is common and well known in the community. This provides users with better user experience.

4) RESOURCES POOLING

The limited number of USRPs and their relatively high cost raises the need for efficient utilization of these high-cost devices. Other testbeds typically assign a USRP device to each node so that users can configure and use the USRPs from the connected nodes. Instead, we amortize the cost of the high-cost USRPs using the concept of resource pooling, i.e., putting all USRP devices in a pool and assigning a USRP only to the node upon request. The node-to-USRP connection is established only during the reserved time slot and based on the user’s request. This pooling approach, however, raises multiple challenges, including USRPs and nodes isolation.

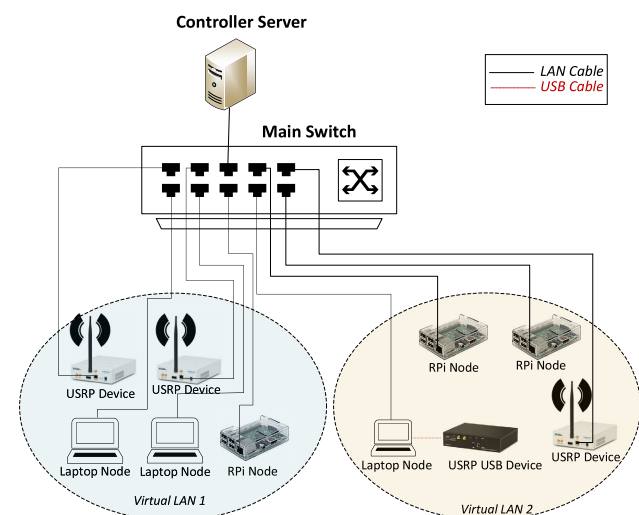


FIGURE 4. Dynamic VLANs creation for resources pooling and users isolation as used in WiPi.

5) DYNAMIC NETWORK TOPOLOGY

The concurrent users and different USRP-to-Node assignment may conflict if there is no control over the network of the testbed devices. The reserved nodes and devices have to be isolated from other concurrent users’ ones. For that, we group different users’ devices and nodes by controlling the topology of the testbed network dynamically to ensure the isolation of different reservations using virtual LANs. Figure 4 illustrates how two concurrent users with two different reservations can be isolated dynamically based on the reserved devices. As shown, each user’s reserved devices are attached to a virtual LAN dynamically at the time of reservation.

6) POWER-EFFICIENT TESTBED

The low-power consumed by Raspberry Pi devices saves much power in the whole testbed as compared to using traditional computing devices. To further reduce the energy consumption of the WiPi testbed, we control the powering operations of Raspberry Pis through the Power over Ethernet (PoE) technology. Specifically, each Raspberry Pi is connected to a PoE module that can power on/off the RPi. The PoE module takes power from a PoE port of the main switch through LAN cable (Pin 4,5) and outputs the power to the power-in ports of RPi. Through a PoE switch and PoE modules, we can power on/off RPi nodes using scripts that are integrated with the reservation system. The system powers on the reserved nodes only at the reserved time and turns off the nodes when the reservation ends or users log out at the end of their reservation times. Figure 5 illustrates how a PoE module of a Raspberry Pi work; a PoE switch provides the module with the data and the necessary power through a single cable, while the module separates them into proper terminals.

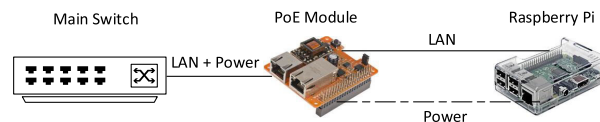


FIGURE 5. The PoE module used in controlling Raspberry Pis in WiPi.

PoE modules can also be used to force Raspberry Pis, when required, to reboot and initiate the process of loading a user’s image from the network. This is useful for user isolation and controlling nodes reservations. More details about the booting procedure of Raspberry Pis are covered in Section III-B2.

7) EFFICIENT STORAGE AND BANDWIDTH UTILIZATION

We provide base images that can be loaded into the reserved nodes based on the user choice. These pre-built images save learning time and setup problems for testbed users. In addition, images built and shared by users can be utilized by other users to save their time. After the users finish their experiments, they need to save back the changes made to the base image to their allocated space in the storage server. We use Frisbee [32] to handle disk imaging operations. Frisbee enables multi-cast operations in transferring disk images to targeted nodes. This allows a single disk image to be loaded into multiple nodes at the same time with minimal bandwidth overhead. We conducted an experiment to evaluate the efficiency of using Frisbee as a multi-cast disk imaging tool in loading the same disk image, i.e., in parallel, into multiple RPi nodes, instead of loading the disk image sequentially into each node. We created a compressed base disk image of Raspian operating system using the frisbee [32] client software with a total size of 2.43 GB. A storage server with frisbeed [32] software was used to load the same disk image to multiple Raspberry Pis in parallel. Figure 6 shows the average loading time, starting from initiating the transfer until successfully loading the base image when loading the

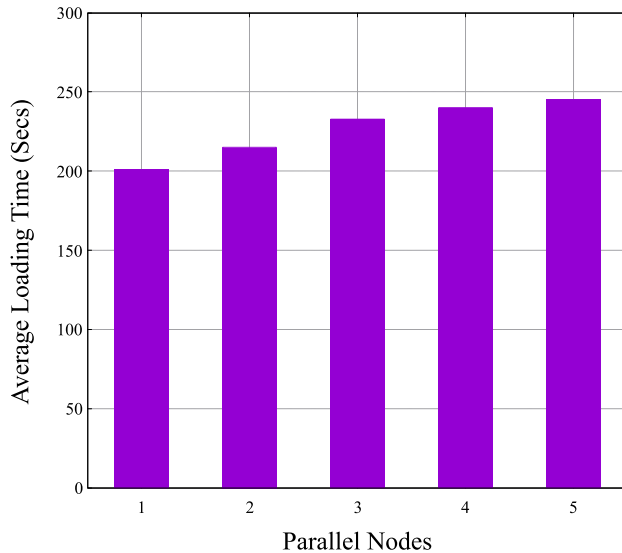


FIGURE 6. The average loading time for loading a base image into a different number of nodes in parallel.

same disk image to a different number of nodes in parallel. From the results, we can see that using Frisbee [32] can speed up the process of loading the same disk image into multiple nodes instead of loading it sequentially.

8) DISK PROTECTION

All users to the WiPi testbed get root access to the RPi nodes. This gives them full flexibility in running their experiments. However, having root access, users can overwrite the SD card plugged into Raspberry Pis, which can affect the entire operation of the Raspberry Pi. To avoid that, we force RPi3-B models to boot from the network upon restart and ignore the SD card. Then, the booted operating system will initiate the disk image writing procedure. For RPi1-B, the model does not support booting from the network; so, we use u-boot [33] to add network booting feature to RPi1-B nodes.

Implementing features such as disk imaging, storage server, and disk protection add complexity for the users to run their experiments. The users have to load the desired disk image, either their saved or shared images, to the nodes before starting using them. The added features increase the time needed to start the experiment. However, we overcome this issue by loading the required images to the desired nodes before the reservation starts, besides, implementing these features allow each user to have root access to the testbed nodes so that they can fully control them.

9) MULTI-APPLICATIONS DOMAIN

Besides the importance of the testbed in wireless research, it can be extended to include more research fields. Specifically, the Raspberry Pi's GPIO pins can be used to connect a different kind of sensors to support new application domains, e.g., sensor networks and IoT.

B. ARCHITECTURE

Figure 7 shows the architecture of the testbed. It consists of Raspberry Pi nodes installed with their PoE modules to control the nodes' power, in addition to, USRP devices that can be connected via LAN port and laptop nodes that are connected, in some of them, to USRP1 devices via USB ports. All the nodes and USRP devices are connected via a managed PoE-support switch and a controller server. The controller server plays a set of roles and can be accessed by researchers at their reservation time with limited privileges. Details about the hardware and software used are covered in the next subsections.

1) HARDWARE

We implemented the WiPi testbed using the following hardware:

a: LAPTOP NODES

The testbed includes laptops with various brands and different specifications. As USRP1 devices can be accessed from its USB port only, some of the laptops are connected directly to USRP1 devices through a USB cable. The researcher can reserve a laptop connected to a USRP1 device or a standalone laptop. Standalone laptops can run various applications or be connected to one of the USRP2 pooled devices attached to the testbed network. Therefore, the testbed utilizes available devices and nodes efficiently instead of dedicating a USRP2 device to each laptop or RPi.

b: RASPBERRY PI NODES

The testbed includes RPi3-B nodes installed with their PoE modules. The testbed can be extended using other models of Raspberry Pis. There are few differences in their specifications and booting procedure that may affect their integration with the system. We show later how this can affect images loading procedure and how we solved it.

c: PoE MAIN SWITCH

A GigaEthernet PoE switch was installed to connect RPi nodes, laptop nodes, USRP devices, and the controller server. The switch provides the necessary power to Raspberry Pi nodes through their PoE modules. Also, it can be configured using its console port by the VLANs control module and power control module. The reserved nodes and devices are grouped in a new dynamically-created VLAN that prevents other users from accessing the reserved devices. We developed a VLAN control module that automatically connects to and configures the main switch to add the ports assigned to the reserved nodes into a new VLAN during the reserved time. As shown in Figure 4, suppose that two independent users reserved different devices and nodes at the same time. Assume that User 1 reserved two USRP devices, two laptop nodes, and one Raspberry Pi node. Also assume that User 2 reserved one laptop node connected to a USRP1 device

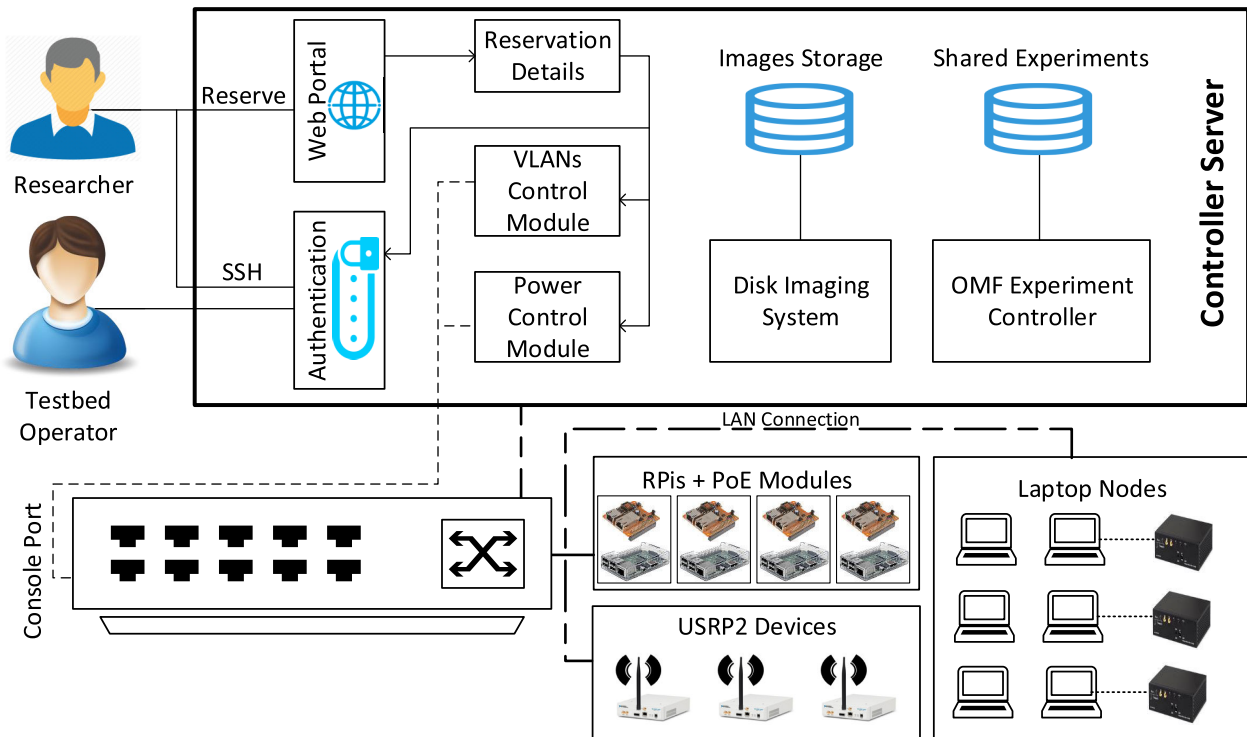


FIGURE 7. The WiPi testbed architecture.

through a USB connection, one USRP device, and two Raspberry Pi nodes. The system will automatically detect the concurrency, and the controller server will run scripts that will access the main switch to create two VLANs, one for each user. The nodes and devices reserved by the first user will automatically be assigned to VLAN 1. Similarly, the nodes and devices reserved by the second user will automatically be assigned to VLAN 2. Pooling and isolation can be achieved without any interference with other users zones using this method.

d: USRP DEVICES

Software Defined Radio (SDR) devices with model number USRP1 and USRP2 are installed with their daughter boards. The high-bandwidth and high-dynamic-range processing capability of the product provide researchers with powerful devices for testing their experiments and results.

e: CONTROLLER SERVER

We prepared a workstation to act as the controller server that manages the web portal, reservation system, experiments, VLANs control module, power control module, and disk imaging software. The server can be fully controlled by testbed operators or partially accessed by researchers during their reservation times with limited privileges.

2) SOFTWARE

The controller server and the nodes were prepared to accept different packages to facilitate the operation of the testbed. The main packages include:

a: RESOURCE CONTROLLER AND EXPERIMENT CONTROLLER

To provide an efficient framework to manage and measure different researchers' experiments, we used OMF [13] to manage the experiments and OML to measure the results. Using OMF, researchers can write the experiment description into a script that specifies the participating nodes and the required commands to be executed in each node at specific times. The script written by the OMF Experiment Description Language (OEDL) can realize the experiment using a sequence of events to be executed on each node. During that, OML collects the results and stores them in a database to be recovered later. We use OMF because of its popularity and efficiency in experiments' management and measurement. OMF and OML source codes do not support Raspbian operating system and are not compiled for ARM processors. To integrate the required software with our testbed, we edited the source code and cross-compiled it to work over Raspberry Pis with Raspbian operating system.

b: DISK IMAGING

Before the user reservation time begins, the system loads the user-selected base-images into the reserved nodes' disks. We provide base-images installed with common protocols and tools e.g., NS2 [2], ns-3 [3]. After researchers finish their edits and other software installations, they can save the entire disk image into the server's images storage. The saved image can be loaded next time by the same user. Using this,

the users cannot affect each other, and they can resume from their final edits. To achieve that, we configure the laptops and Raspberry Pis to boot initially from the network, pull their IP address from the DHCP server, and load from the TFTP server a minimal operating system that later loads the user-specified OS. Specifically, we compiled a tiny operating system (TinyOS) [34] with the required Frisbee [32] client and disk imaging tools. The TinyOS kernel and files are loaded into the testbed nodes first. After that, the Frisbee client call the Frisbee server to load the image of the specific user into that node.

Note that older models of Raspberry Pi, e.g., Raspberry Pi 1, do not support network booting. To overcome this problem, we install a u-boot operating system [33] on the SD Card of Raspberry Pi 1 nodes to initiate the TFTP booting procedure. Furthermore, the low-memory of Raspberry Pis (512 MB for RPi1 and 1 GB for RPi3) obstruct Frisbee client's operation during the loading process. We overcome the problem by dividing the image into small size blocks and load each block separately.

c: WEB PORTAL

The web portal allows both testbed operators and researchers to authenticate and access the system. Researchers have to reserve the required nodes and devices from the reservation system module of the web portal. Their reservation details are provided to both the VLANs Control Module to separate the reserved items during reservation time and Power Control Module to power on the reserved Raspberry Pis before the reservation time and initiate the user's disk images loading procedure. Users can access the controller server and run their experiments either through a web interface or SSH.

IV. SUPPORTING LARGE-SCALE EXPERIMENTS

Supporting large-scale experiments that include a large number of resources is a complex task for testbeds with limited physical resources. In WiPi, we enhance the testbed scalability using two techniques: Node Virtualization and Node Emulation.

In node virtualization, system virtualization is used to allow slicing the physical node into virtual nodes with predefined specifications (Disk Space, RAM, Network Interfaces, etc.) The created virtual nodes are able to dynamically access any of the available physical resources such as network interfaces.

For simplicity, we use static slicing of the physical nodes in our implementation. A set of static virtual nodes with predefined specifications are created in each physical node. Each virtual node has exclusive access to one of the wireless interfaces attached to the physical node.

To further increase the scale of the testbed, node emulation is used. In some experiments, researchers are interested in probing and evaluating the behavior of a small number of nodes in a system consisting of a large number of nodes. For example, in evaluating a wireless routing protocol,

the researcher might want to physically test a scenario on a few nodes that are part of a bigger wireless network. Running the entire scenario on real hardware can be resource exhaustive and of little value, as the user is not interested in the exact behavior of every single node, but rather the behavior of the big network on a subset of nodes. For such scenarios, combining real-world and simulation experiment through emulation can be advantageous.

ns-3 is one of the simulation tools that provide emulation capabilities to support the integration of simulated networks with real network devices. ns-3 supports two kinds of network devices [3]: The first kind is called Emu NetDevice. It allows the nodes in the simulation to send data on the physical network. The second kind is called Tap NetDevice. It allows physical nodes to participate in the simulation as if they were one of the simulated nodes. The simulation can include a combination of the two kinds. As mentioned before, Figure 1 and Figure 2 illustrate the Emu NetDevice and Tap NetDevice in the ns-3 simulation respectively.

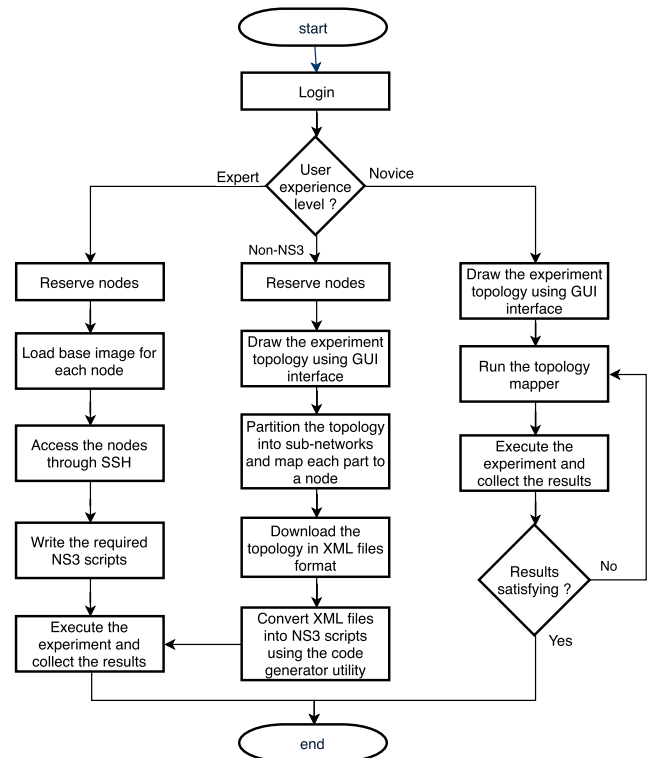


FIGURE 8. Steps required by the user to run a large-scale experiment.

The proposed node emulation approach integrates the ns-3 tool with the testbed physical nodes. We include ns-3 base images to the predefined list of base images in WiPi. The balance of this section covers the proposed node emulation approach, starting with implementation goals, and going through the detailed implementation and supported features. Figure 8 summarizes the steps required by the users to accomplish their large-scale experimentation.

A. NODE EMULATION IMPLEMENTATION GOALS

1) MIXING SIMULATION AND EMULATION IN LARGE-SCALE EXPERIMENTS

Fulfilling the resources requirements of large-scale experiments could exhaust the resources of the testbed, while the researcher may only be interested in the behavior of a part of the network. Therefore, it is reasonable only to use the physical resources to emulate the part of the network under concern, while simulating the rest of the network. The main goal of node emulation is to enhance the scalability of the testbed by applying the concept of mixing simulation and emulation in large-scale experiments. A simulation will be used for most of the network, while emulation will only be used for the parts that have the greatest effect on the evaluation study. Figure 9 provides an example of a topology that is partitioned into two parts, where the sub-network in each part will be simulated on a different physical node, while the link in red color is the one of interest; and it will use a real network interface between the two physical nodes (Emulated).

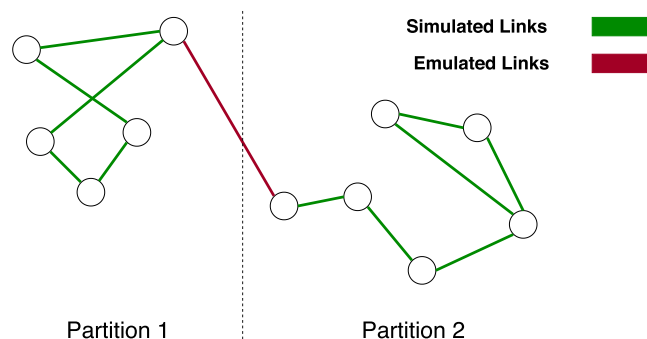


FIGURE 9. An example of mixing simulation and emulation, so that only the parts of interest in the network are emulated, while the rest are simulated.

2) TARGETING DIFFERENT USER EXPERIENCES

Designing and running node emulation experiments using ns-3 requires a certain level of experience. Three node emulation levels are provided to the users to increase the testbed usability. Each node emulation level targets one of the following three different users based on their experiences:

- 1) **Expert User:** This user is the one with the highest experience. He or she can design and partition the experiment using the node emulation concept. He or she also can write and run the ns-3 emulation scripts as well as work with shell commands to prepare and initialize the used resources.
- 2) **Non-ns-3 User:** This user is expected to be able to design his or her own partitioning of the experiments, but he or she is not an ns-3 user and cannot write ns-3 scripts. The testbed is supposed to provide a way of taking users' experiment designs as an input and automatically generate and run the required ns-3 scripts.
- 3) **Novice User:** This user only has his or her experiment topology but does not know how to partition it to

use node emulation and cannot write ns-3 emulation scripts. The testbed shall provide automatic partitioning tool to find the best mapping between the user topology and available physical nodes, in addition to generating and running the required ns-3 scripts.

3) RESOURCES UTILIZATION

The main goal of node emulation is to support large-scale experiments with limited available resources. Hence, the mapping technique that maps emulated partitions into physical resources should take into account resources utilization. In particular, it should minimize the resources used and lower the experiment time as much as possible, while not affecting the expected performance of the experiment.

When choosing the physical nodes that will be participated in emulation experiments, three aspects should be taken into consideration:

- 1) The number of physical resources available in the node (RAM, CPU, ...): These resources should be optimized with respect to the part of the topology to be run on that physical node.
- 2) The demand rate of the node: A resource that is highly demanded in the testbed should be excluded or used less in the automatic emulation mapping.
- 3) The frequency of choosing a physical node in the automatic mapping: If the same physical node is always selected to be used in emulation experiments, this node will be exhausted, while there could be other nodes that are less frequently used. A round-robin approach is used for selecting nodes in emulation experiments to avoid exhausting physical nodes.

B. IMPLEMENTATION

As mentioned earlier, node emulation is divided into three levels based on the user experience. In this subsection, we describe the implementation of each level in details.

1) MANUAL EXPERIMENTS

Manual experiments are considered to be the highest level of node emulation, i.e., only expert users would be able to perform them. To manually design and run node emulation experiments, the user must have at least a basic experience in the following:

- Topology partitioning techniques, to be able to design the emulation experiment, given the available resources.
- ns-3 and its emulation capabilities, to be able to write the required ns-3 scripts.
- UNIX shell commands, to set up testbed nodes and network interfaces to be used for node emulation.

The following steps are needed to run manual experiments:

- a) **Reserve Testbed Nodes:** The user needs first to find and reserve testbed nodes. When a node is reserved, the user is permitted to choose the initial operating system image to be loaded on the node. As node emulation uses ns-3, the WiPi testbed provides a basic image that has ns-3 installed and ready to be used.

- b) Partition the large-scale network topology: The large network topology needs to be partitioned into smaller sub-networks. Also, these sub-networks need to be mapped to the reserved testbed nodes. The users are fully responsible for the partitioning task because they are fully aware of their experiments, they can determine the parts of interest in the topology, and they can specify the parts to be emulated not simulated. Given the number of reserved nodes and the computing power of each node, the user should partition the topology, and design the emulation experiment.
 - c) Write the required ns-3 scripts: As shown in Figure 1, each testbed node should run an ns-3 emulation module. The user should prepare the ns-3 script that will be run on each reserved testbed node. Each ns-3 module should use the suitable emulation device provided by ns-3, either Emu NetDevice, or Tap NetDevice. In addition to emulation devices, each module needs to contain the part of the topology to be simulated on the corresponding testbed node.
 - d) Prepare the network interfaces: The user should define and prepare the network interfaces that will be used for each emulation module. If the experiment uses only Ethernet devices, Ethernet connections between nodes are already set up by the testbed. Otherwise, the user needs to set up the needed connections. For example, if the experiment will use WiFi interfaces, an Ad-hoc wireless network should be created to allow for the needed connections. Furthermore, the used network interfaces should be put in promiscuous mode, as it is a mandatory requirement by ns-3 emulation devices.
 - e) Run the experiment and acquire results: ns-3 provides two approaches to obtain experimental results. The first approach is by creating PCAP files that contain experiment traces. The second is by using the flow monitor module. This module computes most of the performance metrics that could be needed in an experiment, such as throughput, delay, packet loss, etc. After running the experiment, the user can acquire the results by finding and extracting the needed performance metrics from the generated PCAP and flow monitor files.
- a) Experiment Design: The goal of this phase is to design an experiment and to provide the design to the testbed. An XML description language based on the Topology_Generator tool [35] is used to describe the experiment. This description-language includes the following main elements:
 - Physical node: This element specifies the physical testbed nodes to be used in the experiment. A physical node is described by four values: a virtual name chosen by the user, the real name of the node in the testbed, the IP address of the interface attached to the testbed's network, and the MAC address of that interface.
 - Node: This describes a simulated node in the topology, and it could be one of three types: PC node, Emu node, or Tap node. It also contains a value that refers to the physical node that hosts the simulated node.
 - Hardware device: A simulated hardware device, and it could be one of the following: Hub, Switch, or Router.
 - Link: A wired or a wireless link, and it points to a list of physical nodes that host its connected nodes.
 - Application: This element describes a running application (sender, receiver, UDP/TCP, transmission rate, and transmission time).

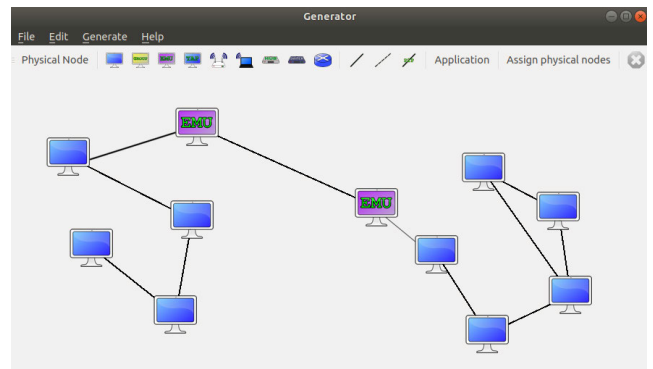


FIGURE 10. An example of designing an emulation experiment using the GUI application provided by the WiPi testbed.

2) NS-3 EXPERIMENT AUTOMATION

In this level of node emulation, the testbed assumes a user with no prior experience in writing and running ns-3. The user is required to provide a detailed design of his or her emulation experiment that specifies the following:

- Physical nodes that will be used in the experiment.
- The number of required partitions (sub-networks).
- The sub-networks to nodes map.

The testbed will use these parameters to produce the required ns-3 scripts automatically and to run the generated scripts.

The implementation of the experiment automation is executed in two phases, experiment design and experiment execution:

To facilitate the creation of the XML description file, we have developed a graphical user interface application with drag and drop capability to allow for an easy way for designing an experiment. The GUI application allows the user to create physical nodes and use them to host nodes, links, and hardware devices. It then gives the user the option to assign each created physical node to a reserved node from the testbed. Figure 10 provides an example of an experiment designed using the GUI application.

- b) Experiment Execution: After the experiment design is provided in XML format, the testbed is required to perform the following tasks:

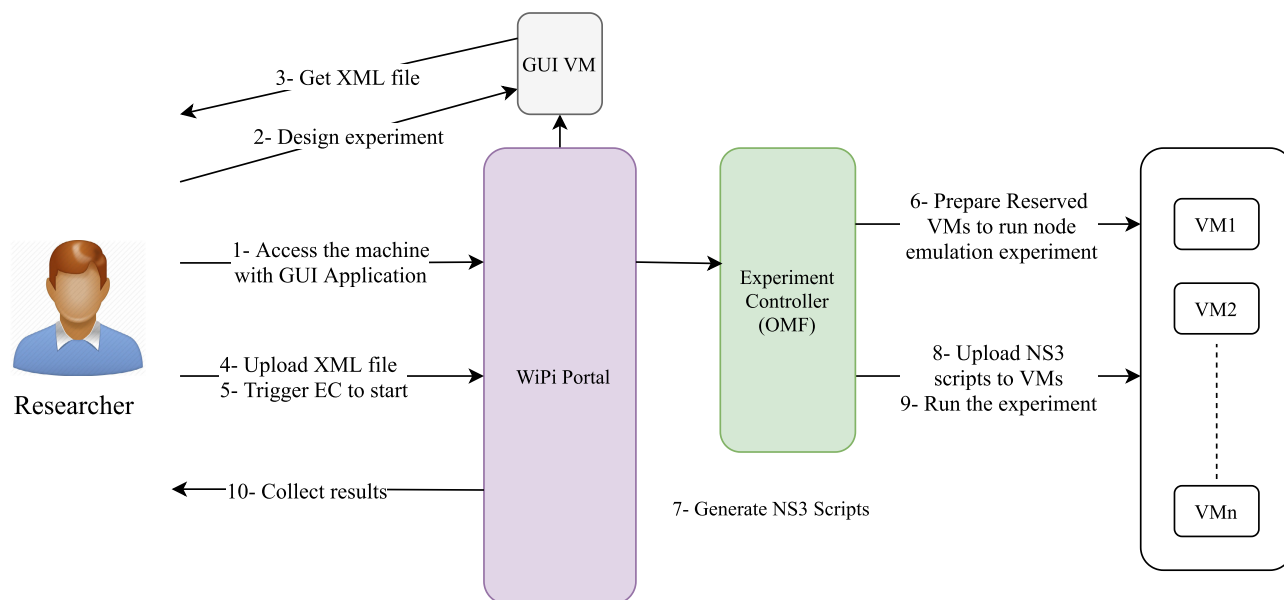


FIGURE 11. End-to-end flow of the ns-3 automation level of node emulation in WiPi.

- Prepare and initialize the testbed nodes to run in node emulation mode.
- Generate the required ns-3 scripts based on the provided experiment design.
- Run the generated ns-3 scripts on their correspondent testbed nodes.
- Provide a way to collect the experiment results.

The experiment controller module of WiPi is the entity responsible for performing all of the required tasks to run the experiment. When the experiment controller is triggered to run an experiment, it first defines and prepares the used testbed nodes by setting up the needed network connections between them and putting the used network devices in promiscuous mode.

The next step is to generate ns-3 scripts; the controller uses an ns-3 code generator application that is based on the Topology_Generator [35] to generate the needed ns-3 scripts. The generator takes the given XML experiment design as an input and generates a number of ns-3 scripts equal to the number of physical machines defined in the design. Each generated ns-3 scripts contains only the part of the topology that will be simulated on its correspondent testbed node.

After generating ns-3 scripts, the experiment controller moves each generated ns-3 script to its correspondent reserved physical node. To control and run an ns-3 script on testbed nodes, the OMF controller is used. An OMF script is used to create the needed ns-3 applications on the reserved nodes and to run the created application for the required simulation time. The simulation time of the experiment is defined by the maximum time needed for all of the running applications in the experiment to terminate.

WiPi allows two ways to provide the results to the user:

- Allow the user to collect it manually by viewing and analyzing the generated PCAP and flow monitor files.

- Use OML to collect the results and store them to an accessible database for the user to acquire. OML defines a set of measurement points and computes performance metrics at each defined point. The collected points are then stored in the measurements database.

Figure 11 illustrates the end-to-end flow of the ns-3 automation level of node emulation. As shown, the user can first use the provided GUI application to generate the XML file that describes the experiment, then trigger the experiment controller to start running the experiment by providing the input XML file. To be able to trigger the controller for running the experiment, the used testbed nodes must be reserved in advance.

The experiment controller generates the required ns-3 scripts and uses OMF to run them on the reserved testbed nodes based on the assignment of nodes provided in the given experiment design.

3) TOPOLOGY MAPPING

In this level of node emulation, the WiPi testbed targets novice users who do not have any indication on how to partition the network topology of their experiments over physical nodes nor the number of physical nodes required to complete their experiments. The testbed interface is supposed to take the virtual network topology as an input and automatically produce the near-optimal partitions of this topology as an output. This operation is divided into two parts: The first part is mapping each simulated node in the virtual network to a physical node in the testbed. The second part is emulating a set of virtual links established between simulated nodes in the virtual network to a set of real links in the testbed.

The topology mapping requires each physical node to be represented by a cost function to distinguish between

different types and specifications of each physical node. The following parameters are taken into consideration:

- 1) The physical capacity of the physical node (CPU, RAM, etc.), which defines the load on the node, given the part of the network simulated on it.
- 2) The demand ratio of the node, which ensures not to exhaust the same nodes every time a new topology mapping is performed.
- 3) The maximum data rate of the real links attached to the node, which ensure not to overload real links with data more than their maximum capacity.

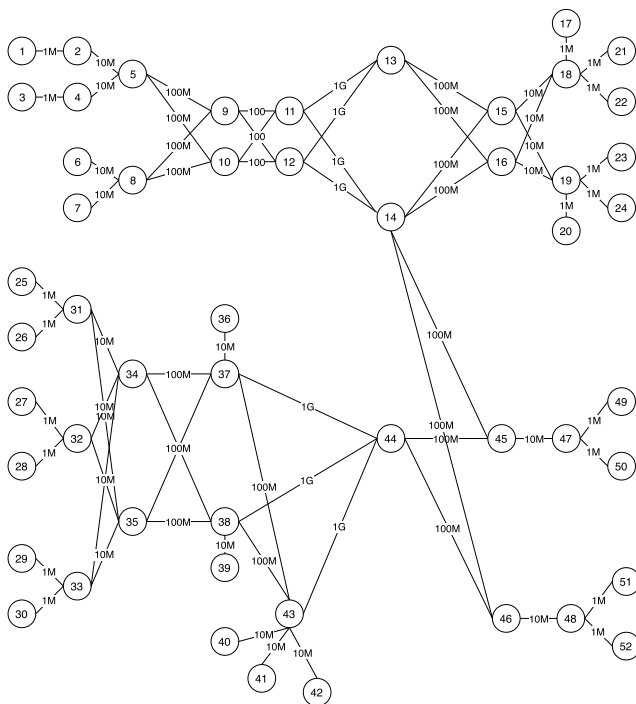


FIGURE 12. Experiment topology.

Traditionally, when performing topology mapping [26], the load of physical nodes is not taken into account nor the demand ratio of physical nodes when performing partitioning. To illustrate the inefficiency of the current traditional solutions, let us assume that we have the topology shown in Figure 12. The topology has 52 nodes to be simulated, and links are labeled with their capacities. Assume the testbed has eight available physical nodes, then the topology can be partitioned into up to eight testbed nodes. We use the METIS [26] partitioning software to perform the required partitioning, with each link weight is equal to the link capacity, and each node weight is the cardinality of the node. Figure 13 shows the resulted partitions over the eight testbed nodes. In the previous example, each physical node runs 4-8 simulated nodes, while the available resources in a physical node could afford to simulate more virtual nodes, leading to a waste of testbed resources.

On the other side, not only the load on the nodes but also the total simulation time of the experiment must

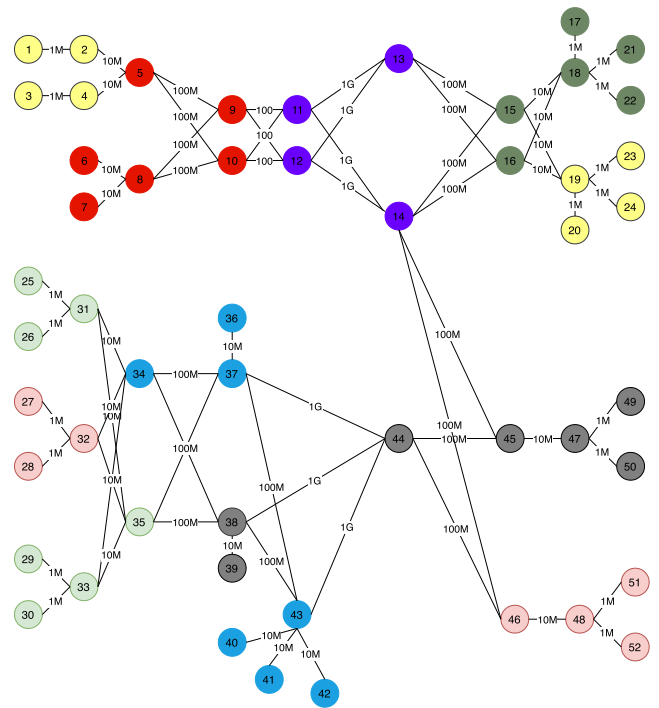


FIGURE 13. Partitioned topology. Different colors represent a different mapping of virtual nodes to a physical node.

be considered. From our experiments, there is a positive relationship between the number of simulated nodes run per physical node and the total simulation time. So, a good distribution of virtual nodes over physical nodes in the mapping operation will decrease the total simulation time.

Searching the mapping space for the best-evaluated point given a cost function is an NP-hard problem [36]. Therefore a greedy search method could be used to find a near-optimal solution for the mapping.

Our approach in finding the near-optimal solution is divided into an initial run and further runs. Initially, the user can select a performance level on a scale from 1-10. This performance level is a direct map to the total simulation time, where the 10th level will run the whole virtual network topology on a single physical node (pure simulation), and the 1st level will use all of the available physical nodes to run the experiment. The levels in between will use a portion of the available nodes based on the selected level.

After specifying the number of physical nodes, the partitioning problem can be described as follows: let n be the number of virtual nodes, k is the number of physical nodes, and \mathbf{W}_{ij} is the virtual link capacity between node i and j . We construct an undirected graph that consists of n vertices. Each edge lies between vertex i and vertex j has a weight \mathbf{W}_{ij} . Subsequently, the partitioning technique tries to find the minimum k cuts based on the edge weights. Hence, it will tend to avoid emulating virtual links with high capacities (high weight on edges).

In practice, the actual amount of traffic passing through a real link could differ from the capacity of the mapped

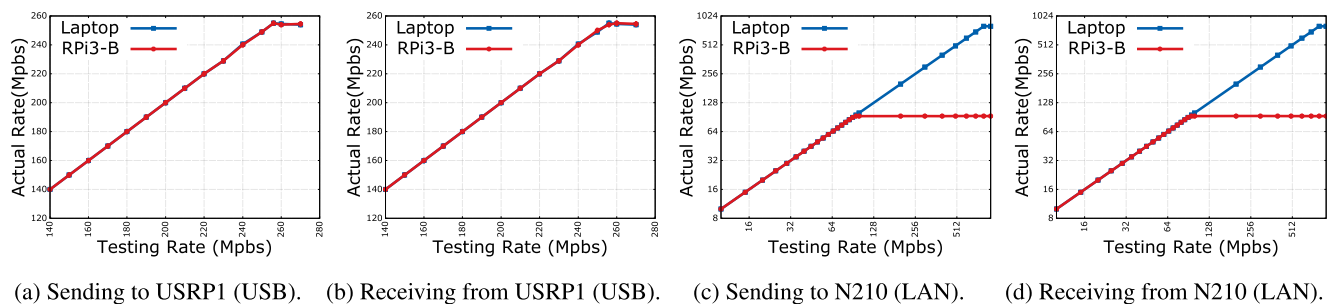


FIGURE 14. Evaluating the data rate between the testbed nodes and the USRPs using the USB or LAN interfaces.

virtual link. The traffic could be larger or lower than the capacity by a considerable deviation. Therefore, using link capacities is not assured of achieving the target of lowering the traffic passing through emulated links, and it would be better to use the actual amount of traffic instead of link capacities. This is achieved incrementally by monitoring the output of the different runs of the same simulation experiment. Specifically, if the users are not satisfied with the emulation performance, i.e., the total simulation time, they can use the extra information collected during the run in next runs to enhance the partitioning performance (as usual, any performance evaluation is based on multiple runs, and the average results are reported). In the new run of the experiment, instead of using link capacities, the amount of actual traffic passing through each link will be used as the weight of the link/edge. Thus, this second run will tend to lower the actual traffic passing through emulated links, which is expected to improve the performance of the partitioning.

V. EVALUATION

In this section, we evaluate WiPi from three perspectives. The first one is the capabilities of low-cost devices when participating in testbeds as processing nodes. The second one is the performance of our approach of combining the nodes/networks simulation and emulation in the same experiment as compared to using the simulation or emulation independently. The last one is the performance of executing large-scale experiments using WiPi testbed as compared to using a single machine.

A. LOW-COST DEVICES

In this part, we evaluate the capabilities of low-cost devices, as processing nodes, when connected to external SDR devices. The evaluation studies the throughput (data rate) between nodes, either Raspberry Pis or laptops and USRP devices. Tables 3 and 4 summarize the specifications of the testbed processing nodes and the SDR devices, respectively.

The data can be transferred in terms of streams between the FPGA of a USRP device and a node using a host interface. We evaluated the streaming rate through *benchmark_rate* command that inputs the testing rate in Mega Samples per Second (MS/s), the sample type, the direction of the transfer,

TABLE 3. Installed nodes specifications.

Type	Processor	CPU Cores	RAM	Storage	LAN	USB
RPi3-B	1.2 GHz Quad Core	4	1 GB	N/A	10/100 BaseT Ethernet	USB 2.0
Laptop node (Example)	2.4 GHz Core i3	2	4 GB	256 GB	10/100/1000 BaseT Ethernet	USB 2.0

TABLE 4. Installed SDR devices specifications.

Type	LAN	USB	Max. Host Streaming Rate
USRPI	NA	USB 2.0	256 Mbps
USRP2 N210	Gigabit Ethernet	NA	800 Mbps

and other options. We fixed the type of samples to be I/Q sample, which is used by most applications and the data format to be 16-bit complex< float>. The total size of each sample is 32-bit. We monitored the number of samples that can be sent/received/dropped between nodes and different USRP devices in addition to the number of overflows/underflows detected, and then we calculated the actual rate. Figure 14 summarizes the results obtained using the two types of USRPs and the testbed nodes.

1) RASPBERRY PI NODES

Using a Raspberry Pi as a software-defined radio processing node has some limitations due to the limited capabilities of Raspberry Pis. However, if the experiment and the required processing are small, it can be handled by Raspberry Pis. We show here the throughput between Raspberry Pis and USRP devices. We connected Raspberry Pis with a USRP1 device via a USB cable and monitored the samples sent/received to/from USRP1 device. Figure 14a, 14b summarize the 95% confidence interval of the average of sending/receiving rate to/from a USRP1 device through USB2 connection. As shown in the figures, a Raspberry Pi 3-B can perfectly handle the small and high date rates sent/received to/from a USRP1 device up to the maximum host streaming rate of the USRP1 device (256 Mbps). Furthermore, due to the maximum connection rate of Ethernet technology (100 Mbps), a Raspberry Pi 3-B cannot handle, on average, more than 93.5 Mbps practically when connected to an N210 USRP device through a LAN connection. Figure 14c, 14d shows 95% confidence interval of the average

of the actual rates that can be streamed between a Raspberry Pis and a USRP N210 device.

2) LAPTOP NODES

Nodes consisting of laptops or laptops connected to USRP1 devices are considered the main processing unit to the researchers who are using the WiPi testbed. We evaluated the maximum throughput that a laptop can handle. Figure 14a, 14b shows the relation between the testing rate and the actual rate in 10 seconds when a laptop is connected to a USRP1 device. The results show that a laptop can work as a perfect software-defined radio processing node using a USB connection and up to the maximum allowed rate of USRP1 devices (256 Mbps). Figure 14c, 14d summarize the same test when using a LAN connection to USRP N210 device; it shows that a laptop equipped with a GigaEthernet interface can handle data rates sent/received to/from an N210 USRP devices up to its maximum capabilities (800 Mbps).

From the results above, we can conclude that low-cost devices can be utilized in different applications and experiments. Using such devices in the WiPi testbed will not obstruct the experimentation process.

B. COMBINING SIMULATION AND EMULATION

We conducted some experiments to evaluate the effect of mixing simulation and emulation, against having a pure simulation or pure emulation. In the first experiment, the effect of using a real link/emulated link was evaluated with respect to simulation time and CPU load. For simulation, a single physical node was used to simulate both the server and the client. This way, the traffic between the two nodes is simulated by CPU clocks, as shown in Figure 15.

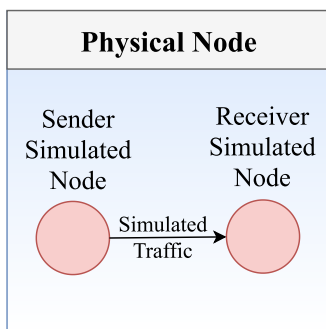


FIGURE 15. Simulation using a single physical node.

For emulation, instead of using one physical node, two physical nodes were used, where one physical node used to simulate the sender and one to simulate the receiver. The traffic between the two simulated nodes passes through the real link between the two physical nodes, as shown in Figure 16. The capacity of both the real link and the simulated link in this experiment is 1Gbps, and hence, the traffic passing through the links is limited to this capacity.

In Figure 17, we change the traffic passing through the links (simulated/emulated) on the X-axis, and record the



FIGURE 16. Emulation using two physical nodes.

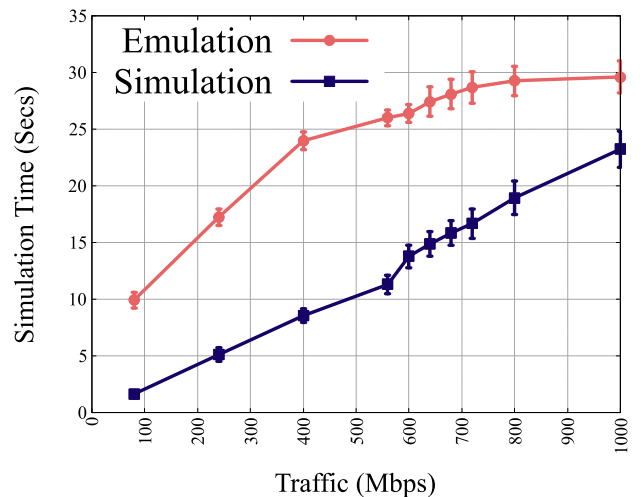


FIGURE 17. Simulated link vs. emulated link with respect to simulation time (Link capacity = 1Gbps).

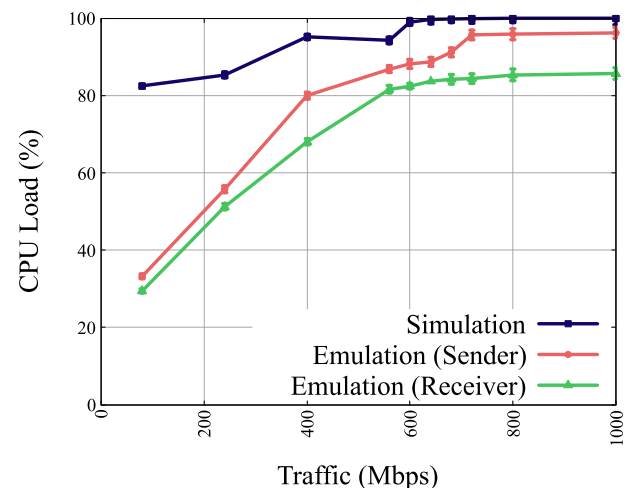


FIGURE 18. Simulated link VS emulated link with respect to CPU load (single node vs sender & receiver nodes).

corresponding average experiment time on Y-axis. From the figure, we can see that time is always greater for emulation than simulation. This is due to the real movement of the traffic in real links instead of just performing CPU clocks in the simulation.

Figure 18 compares average CPU load, where the average load on the single physical node used in the simulation is compared to the average load on the two physical nodes used

in the emulation. It is noticeable that simulating the sender and the receiver on the same physical node consumes more CPU than simulating them in two separate physical nodes. Accordingly, using simulation only in large-scale experiments is limited to the CPU power of the machine running the simulation.

In the second experiment, we evaluated the effect of partitioning the traffic on two physical nodes and an emulated link (mixing simulation and emulation). Instead of simulating/emulating the whole traffic, the traffic is partitioned over simulation and emulation. As shown in Figure 19, the traffic is partitioned on three links: a real link, a simulated link on the first physical node, and a simulated link on the second physical node.

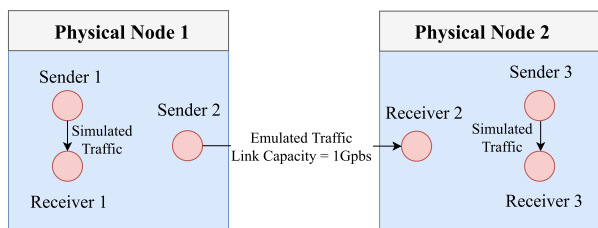


FIGURE 19. Mixing simulation and emulation using two physical nodes.

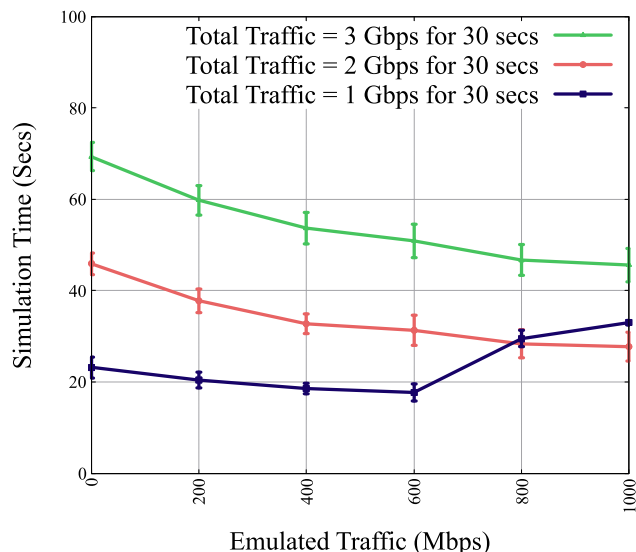


FIGURE 20. Examples on partitioning the traffic using both simulation and emulation. This is an example of the used approach in supporting large-scale experiments in the WiPi testbed.

Figure 20 shows the total time required to finish the experiment for different partitioning scenarios in terms of the actual traffic passed through the emulated link. Note that when the emulated traffic is equal to zero; this means that this is a pure simulation executed internally inside the physical nodes. However, when the emulated traffic is equal to the real link capacity (1G in this experiment), this means that this is a pure emulation, i.e. all traffic is carried on the emulated link between the two physical nodes without any simulation

traffic between the internal virtual nodes. We conducted the experiment for three different values of the total traffic sent between nodes in the same network topology: 1Gbps, 2Gbps, and 3Gbps. The capacity of the simulated links was set to the value of the traffic passing through the link to prevent losses. From Figure 20, we can see that at the beginning of the 1Gbps total traffic curve (the blue curve), when the emulated traffic is equal to 0, the CPUs on the physical nodes are fully utilized by the simulated traffic. This CPU bottleneck makes the total simulation time takes 22 seconds on average. However, the total simulation time starts to decrease when the Network Interface Cards (NICs) start to handle part of the traffic (when emulation starts) until it reaches an optimum point at 600Mbps emulated traffic, where the total simulation time is 18 seconds on average. After this point, the performance starts to decrease (increase in the simulation time) as the bottleneck is moved to the buffers of the NICs that become overloaded. The total simulation time in this case, where the emulation overhead is the dominating factor, is 37 seconds on average. Note that the maximum emulated traffic cannot exceed the link capacity (1000 Mbps in this scenario). This is why we did not see this behavior in the case of 2Gbps and 3Gbps total traffic curves (red and green curves). The previous results show that combining simulation and emulation in one experiment enhances the total experiment time.

The previous experiment shows that partitioning the traffic over physical nodes and increasing the traffic on the emulated link instead of running the total traffic on one physical node, reduces the experiment time. Furthermore, increasing the emulated traffic, which is the traffic passing through the real link, also reduces the time, where, the minimum experiment time is reached when the real link is fully utilized. The total execution time of experiments can be reduced by almost 40% as compared to simulating the total traffic on a single physical node.

In the analysis above, we show that combining simulation and emulation in the same experiment enhances the total time required to perform experiments. In addition, the CPU load of each of the devices that participated in these experiments is minimized. In large-scale experiments, where a single low-cost device cannot hold out the CPU and memory required to perform such experiments, combining simulation and emulation can be advantageous.

C. LARGE-SCALE EXPERIMENTATION

In this part, we evaluate WiPi in terms of the performance of executing large-scale experiments. The setup of the experiment is shown in Figure 21, where the total number of simulated nodes is $n \times m$ divided over m physical nodes. Each physical node has two network interfaces, wired and wireless, and connected to its neighbor node through one of these interfaces. One physical node is simulating a random network topology that consists of n nodes. The experiment ends when each simulated node sends or receives an ICMP packet from all other nodes. We performed two experiments in this context. In the first one, the number of physical testbed

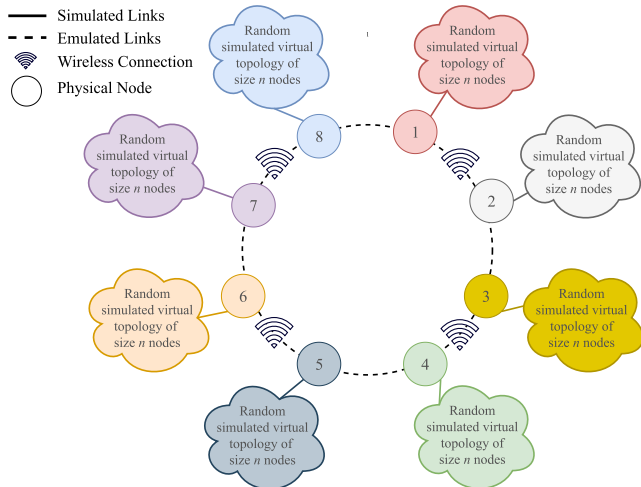


FIGURE 21. A large-scale experiment that consists of m physical nodes (eight in the figure) connected together using real links (eight in the figure), each node is connected to a random virtual topology that consists of n simulated nodes (maximum of 10000). The total simulated nodes vary between 8000 and 80000.

nodes m participated in the experiment is fixed while the number of nodes n simulated on each physical node is dynamic. In the second one, the total simulated nodes $n \times m$ is fixed, and the number of physical testbed nodes m participated in the experiment is dynamic. The details of each experiment results are shown below.

TABLE 5. Simulation machines specifications.

Machine	CPU Cores	CPU Speed	Memory	Disk Storage
Machine 1	4	2.5 GHz	8 GB	512 GB
Machine 2	8	3.2 GHz	16 GB	1 TB
Machine 3	16	3.9 GHz	32 GB	1 TB
Machine 4	32	4.0 GHz	64 GB	2 TB

The first experiment evaluates experimentation over WiPi using a significant amount of physical nodes as compared to using a single machine with different specifications. Table 5 shows the specifications of the machines used in this evaluation. These machines are not part of the WiPi testbed and they are used for evaluation purposes only. Note that the whole topology that consists of $n \times m$ nodes is simulated on each of these machines. We performed the experiment multiple times and recorded the 95% confidence interval of the average. Figure 22 summarizes the average time required to finish the experiment. We can see that executing large-scale experiments over WiPi can reduce the total time required to finish the experiment as compared to using a single machine.

The second experiment shows the effect of using more physical nodes in the total experiment time. We ran the same experiment but using a different number of the testbed nodes m . We performed the experiment multiple times and recorded the 95% confidence interval of the average. Figure 23 summarizes the average time required to finish the experiment. From the figure, we can notice that increasing the

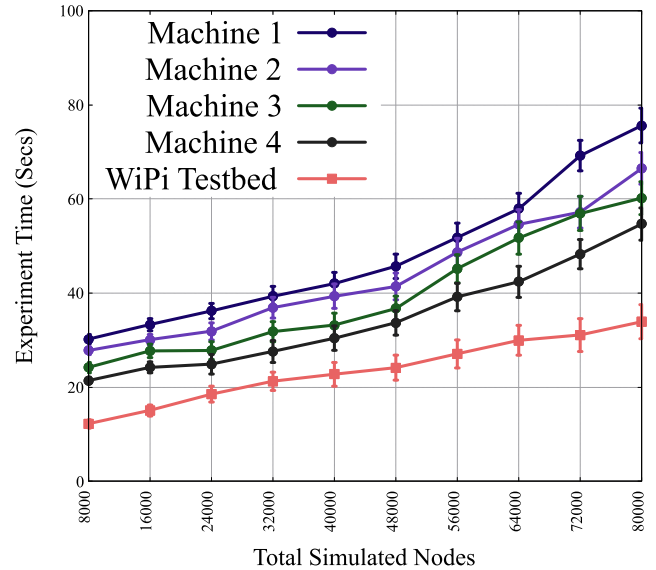


FIGURE 22. The total experiment time required to run a large-scale experiment with topology shown in Figure 21 over WiPi compared to running it over different machines.

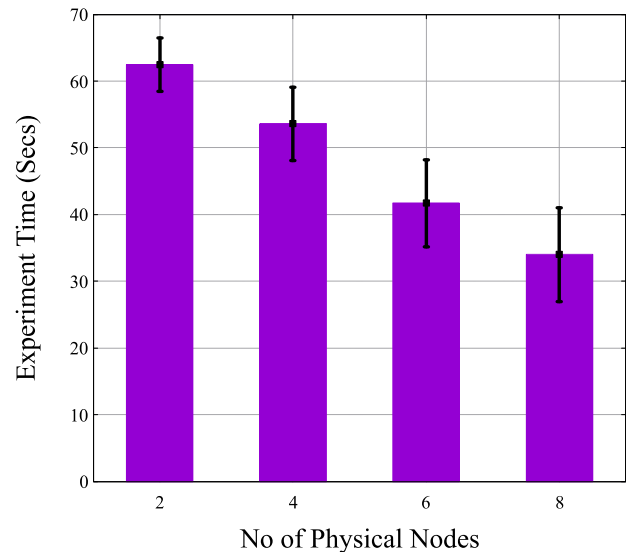


FIGURE 23. The total experiment time required to run a large-scale experiment with topology shown in Figure 21 using a different number of physical nodes.

number of participated physical nodes m in the same experiment can reduce the total experiment time by a significant amount.

In this subsection, we noted the total time of large-scale experiments executed in single machines with different specifications as compared to executing the same experiment in the WiPi testbed. In summary, executing large-scale experiments using more physical nodes of the WiPi testbed minimizes the time required to complete the experiment.

VI. CONCLUSION AND FUTURE DIRECTIONS

We presented the WiPi testbed as a low-cost testbed that can be accessed remotely to provide researchers with processing nodes and wireless devices to test their work and results in

small and large-scale experiments; therefore, increase their research credibility. The testbed is implemented using heterogeneous devices and supports multiple features including homogeneous operating systems, resource pooling, powering control, users isolation using VLANs, disk protection, multi-application domain, and efficient disk utilization. We also addressed the challenges we faced during implementation and how we handled them, especially large-scale experimentation support. The results show that Raspberry Pis can act as software-defined radio processing nodes, in the case of small data rates, and the used laptops can process wireless physical operations perfectly. The evaluation of large-scale experiments shows that WiPi can handle different types of users experience level to support a wide range of researchers.

Currently, we are expanding the testbed in different directions, including handling mobile nodes, supporting sensor and IoT networks, as well as providing better mapping algorithms.

REFERENCES

- [1] A. Attaby and M. Youssef, "WiPi: A low-cost heterogeneous wireless testbed for next generation applications," in *Proc. IEEE 87th Veh. Technol. Conf. (VTC Spring)*, Jun. 2018, pp. 1–5.
- [2] Fall. (2007). *The Network Simulator (NS-2)*. [Online]. Available: <http://www.isi.edu/nsnam/ns>
- [3] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.
- [4] X. Chang, "Network simulations with OPNET," in *Proc. Winter Simulation Conf. Simulation—A Bridge Future*, Dec. 1999, pp. 307–314.
- [5] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *Proc. IEEE Mil. Commun. Conf.*, Nov. 2008, pp. 1–7.
- [6] P. Zheng and L. M. Ni, "EMPOWER: A network emulator for wireline and wireless networks," in *Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, Mar./Apr. 2003, pp. 1933–1942.
- [7] S. S. Hanna, A. Guirguis, M. A. Mahdi, Y. A. El-Nakieb, M. A. Eldin, and D. M. Saber, "CRC: Collaborative research and teaching testbed for wireless communications and networks," in *Proc. 10th ACM Int. Workshop Wireless Netw. Testbeds Exp. Eval. Characterization*, 2016, pp. 73–80.
- [8] T. R. Newman, A. He, J. Gaedert, B. Hilburn, T. Bose, and J. H. Reed, "Virginia tech cognitive radio network testbed and open source cognitive radio framework," in *Proc. 5th Int. Conf. Testbeds Res. Infrastruct. Develop. Netw. Communities Workshops*, Apr. 2009, pp. 1–3.
- [9] A. Saeed, M. Ibrahim, K. A. Harras, and M. Youssef, "A low-cost large-scale framework for cognitive radio routing protocols testing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 2900–2904.
- [10] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremono, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2005, pp. 1664–1669.
- [11] K. Pechlivanidou, K. Katsalis, I. Igoumenos, D. Katsaros, T. Korakis, and L. Tassioulas, "NITOS testbed: A cloud based wireless experimentation facility," in *Proc. 26th Int. Teletraffic Congr. (ITC)*, Sep. 2014, pp. 1–6.
- [12] S. Fdida, T. Friedman, and T. Parmentelat, "OneLab: An open federated facility for experimentally driven future Internet research," in *New Network Architectures*. Springer, 2010, pp. 141–152.
- [13] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "OMF: A control and management framework for networking testbeds," *ACM SIGOPS Operating Syst. Rev.*, vol. 43, no. 4, pp. 54–59, 2010.
- [14] A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts, "Future Internet research and experimentation: The FIRE initiative," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, pp. 89–92, Jul. 2007.
- [15] M. Berman, J. S. Chase, L. Landwebe, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Comput. Netw.*, vol. 61, pp. 5–23, Mar. 2014.
- [16] T. Salmito, L. Ciuffo, I. Machado, M. Salvador, M. Stanton, N. Rodriguez, A. Abelem, L. Bergesio, S. Sallent, and L. Baron, "FIBRE—An international testbed for future Internet experimentation," Tech. Rep., 2014.
- [17] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proc. 1st Int. Conf. Simulation Tools Techn. Commun. Netw. Syst. Workshops*, 2008, p. 60.
- [18] OpenVZ. *A Container-Based Virtualization For Linux*. [Online]. Available: <http://wiki.openvz.org/MainPage>
- [19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Dec. 2003.
- [20] VMware. [Online]. Available: <https://en.wikipedia.org/wiki/VMware>
- [21] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the Emulab network testbed," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 113–128.
- [22] XAPI. [Online]. Available: https://wiki.xen.org/wiki/XAPI_Command_Line_Interface
- [23] J. Ahrenholz, "Comparison of CORE network emulation platforms," in *Proc. Mil. Commun. Conf.*, Oct./Nov. 2010, pp. 166–171.
- [24] G. Carneiro, "Ns-3: Network simulator 3," in *Proc. UTM Lab Meeting*, Apr. 2010.
- [25] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "MaxiNet: Distributed emulation of software-defined networks," in *Proc. IFIP Netw. Conf.*, Jun. 2014, pp. 1–9.
- [26] G. Karypis and V. Kumar. (2009). *METIS—Unstructured Graph Partitioning and Sparse Matrix Ordering System Version 2.0*. Accessed: Jul. 2013. [Online]. Available: <http://glaros.dtc.umn.edu/gkhome/views/metis>
- [27] R. Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, and Y. Shinoda, "A multi-purpose wireless network emulator: QOMET," in *Proc. 22nd Int. Conf. Adv. Inf. Netw. Appl.—Workshops*, Mar. 2008, pp. 223–228.
- [28] R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K.-I. Chinen, Y. Tan, and Y. Shinoda, "QOMB: A wireless network emulation testbed," in *Proc. IEEE Global Telecommun. Conf.*, Nov./Dec. 2009, pp. 1–6.
- [29] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An overlay testbed for broad-coverage services," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, Jul. 2003.
- [30] R. Riggio, T. Rasheed, and F. Granelli, "EmPOWER: A testbed for network function virtualization research and experimentation," in *Proc. IEEE SDN Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–5.
- [31] K. C. Borries, G. Judd, D. D. Stancil, and P. Steenkiste, "FPGA-based channel simulator for a wireless network emulator," in *Proc. IEEE 69th Veh. Technol. Conf.*, Apr. 2009, pp. 1–5.
- [32] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb, "Fast, scalable disk imaging with frisbee," in *Proc. USENIX Annu. Tech. Conf. Gen. Track*, 2003, pp. 283–296.
- [33] U. Das. (2002). *Boot—The Universal Boot Loader Verfügbar unter r*. [Online]. Available: <http://www.denx.de/wiki/U-Boot>
- [34] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for sensor networks," *Ambient Intell.*, 2005.
- [35] [Online]. Available: https://github.com/idaholab/Topology_Generator
- [36] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Annu. Acm Symp. Theory Comput.*, 1971, pp. 151–158.



ABDELHAMID ATTABY was born in Cairo, Egypt, in 1988. He received the B.S. and M.S. degrees in computer science engineering from the Faculty of Engineering, Benha University, Egypt, in 2009 and 2014, respectively. He is currently pursuing the Ph.D. degree with the Egypt Japan University of Science and Technology (EJUST). He is a Teaching Assistant of computer engineering with the Computer Science Engineering Department, Benha University. His research interests include image processing, computer networking, and software architecture.



NADA OSMAN was born in Alexandria, Egypt, in 1993. She received the B.S. degree in computer engineering from the Computer and Systems Engineering Department, Alexandria University, Egypt, in 2016, where she is currently pursuing the M.S. degree in computer engineering.

From 2016 to 2018, she was a Software Engineer with Ejada Systems Ltd., Alexandria. Since 2017, she has been a Teaching Assistant of computer engineering with the Computer and Systems Engineering Department, Alexandria University. In 2018, she started being a Research Assistant with CRC Laboratory, Alexandria University. Her research interests include computer networking, machine learning, and deep learning.



MUSTAFA ELNAINAY (M'08–SM'17) received the B.Sc. and M.Sc. degrees in computer engineering from Alexandria University, in 2001 and 2005, respectively, and the Ph.D. degree in computer engineering from Virginia Tech, in 2009. He is currently on leave and affiliated with the Faculty of Computer and Information Systems, Islamic University of Madinah, Saudi Arabia. He is also an Associate Professor of computer engineering with the Computer and Systems Engineering Department, Alexandria University, Egypt. His research interests include wireless

and mobile networks, cognitive radio and cognitive networks, as well as software testing automation and optimization. His focus is on the use of artificial intelligence to solve problems in different domains, including communications and networking, indoor localization, and software engineering. He has served as a Reviewer, TPC member, and TPC chair/track chair for various international journals and conferences.



MOUSTAFA YOUSSEF is currently a Professor with Alexandria University, and also the Founder and Director of the Wireless Research Center of Excellence, Egypt. He has tens of issued and pending patents. His research interests include mobile wireless networks, mobile computing, location determination technologies, pervasive computing, and network security. He is a recipient of the 2003 University of Maryland Invention of the Year Award, the 2010 TWAS-AAS-Microsoft Award

for Young Scientists, the 2013 and 2014 COMESA Innovation Award, the 2013 ACM SIGSpatial GIS Conference Best Paper Award, the 2017 Egyptian State Award, multiple Google Research Awards, among many others. He is the Lead Guest Editor of the upcoming IEEE Computer Special Issue on Transformative Technologies, an Associate Editor for the IEEE TRANSACTIONS ON MOBILE COMPUTING (TMC) and ACM TSAS, and has served as an Area Editor for ACM MC2R as well as on the organizing and technical committees of numerous prestigious conferences. He is an ACM Distinguished Speaker and an ACM Distinguished Scientist.

• • •