Walker: DevOps Inspired Workflow for Experimentation

Mikołaj Chwalisz
Telecommunication Networks Group
Technische Universität Berlin
Berlin, Germany
chwalisz@tkn.tu-berlin.de

Kai Geissdoerfer
Networked Embedded Systems Group
Technische Universität Dresden
Dresden, Germany
kai.geissdoerfer@tu-dresden.de

Adam Wolisz
Telecommunication Networks Group
Technische Universität Berlin
Berlin, Germany
wolisz@tkn.tu-berlin.de

***Abstract*—Experimentation with computer networks under realistic conditions is a necessary step in debugging, profiling and validation towards real deployments and applications. Although the definition of relevant experimentation scenarios is usually relatively straightforward, their implementation and execution are unfortunately difficult and tedious. Generation of extensive experiment documentation assuring replicability is increasingly challenging even for experienced researchers.**

In this paper, we explain how a typical experimentation workflow can be supported using properly selected tools and components from the DevOps ecosystem, leading to repeatable, well-defined measurements. We start with a general approach using ad-hoc setups. Next, we show how the featured set of tools can be used with, and benefit from, existing testbeds.

***Index Terms*—Experimentation, Wireless Networks, Monitoring, Wireless Testbeds**

I. Introduction

Experimentation allows studying the behavior of networks without any simplifications under real-world conditions. Unfortunately, even having the definition of relevant experimentation scenarios, performing meaningful experiments requires numerous skills and significant effort, the process is complex and time-consuming [1], [2]. In addition, documenting all the necessary steps of an experiment in order to assure its *repeatability* (same team, same experimental setup) and *replicability* (different team, same experimental setup) becomes increasingly difficult even for experienced researchers. This is a major problem across the scientific community, which has recently sparked intense discussions [3]–[5].

We discuss how to support the complete workflow for experimentation, starting just from bare-metal commercial off-the-shelf (COTS) hardware and ending with an analysis of measurement data. We focus on experimentation in ad-hoc scenarios as the most general case requiring also the preparation of the hardware. We show how the approach can be helpful to experimentation using testbeds, benefiting from management capabilities and large-scale node deployment. We re-evaluate prevailing tools for experimentation under new aspects like wide applicability, community support, and workflow coverage. We present a selection of tools from the DevOps community and show how they cover the whole workflow of a typical experiment, closing the gap between ad-hoc and large-scale testbed based experimentation. We provide short examples of how the tools can be used to solve the typical tasks involved in experimentation. We provide a well-documented implementation of an example experiment[1] using the discussed tools.

II. DevOps Approach

DevOps is a set of software engineering practices and cultural values that have been proven to improve the software release cycles, software quality and ability to get rapid feedback on product development [6]. This is achieved by unifying software development (Dev), traditionally responsible for contributing new features, with software operation (Ops), responsible for keeping the system running in production. DevOps practices are strongly advocating automation at all steps of the development and deployment process [7].

The DevOps community has proposed a large set of tools with a broad community of users and maintainers. The Puppet report [6] states the increase in a number of respondents working in DevOps teams from 16% in 2014 to 27% in 2017.

Researchers are involved in developing a new solution (i.e. Dev) and evaluating this solution with experiments (i.e. Ops). The explicitness and the high level of automation that comes with DevOps practices helps to tackle some of the major challenges in experimentation, like replicability. The tools support all aspects of automation, testing, and deployment. Their broad scope allows researchers to apply the obtained skills not only to experimentation, increasing the appeal to get familiar with them as opposed to special purpose tools.

III. Experiment execution workflow

Experiments in different areas of wireless network research often share a common setup, shown in Fig. 1. Examples include research in Internet of Things (IoT) [8], co-existence [9]

The authors would like to thank M. Zimmerling and V. Handziski for their valuable comments.

This work has been partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 645274 (WiSHFUL project).

[1]Publicly available at GitHub under https://github.com/mchwalisz/walker

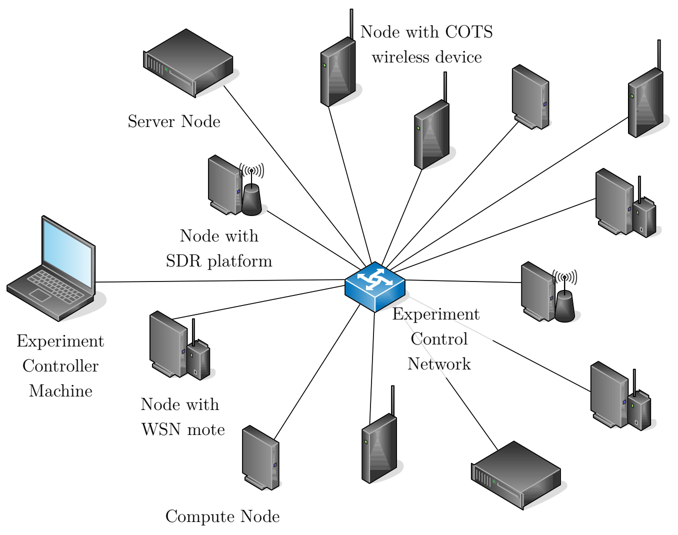
Node with COTS wireless device
Server Node
Node with SDR platform
Experiment Controller Machine
Experiment Control Network
Node with WSN mote
Compute Node

Fig. 1. General experiment architecture

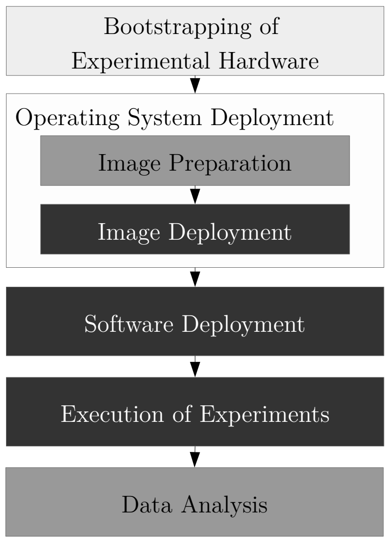

Fig. 2. Experiment execution workflow

or flexible radio architectures [10]. The core of each node is a COTS general purpose computer, e.g. PC machines, single board devices or even virtual machines. The nodes are equipped with the device under tests (DUTs), like sensor nodes, Software Defined Radio (SDR) platforms or PCI Wi-Fi cards. They are used to control the equipment, applications and to collect relevant measurements. Together, the nodes form the experimental networks using the selected radio equipment (DUTs). The nodes are furthermore connected to an experiment controller machine to establish a reliable control channel and thus require an Ethernet interface for connection to the control network and must be able to run the GNU/Linux operating system (OS). Naturally, the **hardware needs to be selected** to meet the requirements of the respective experiment.

A common workflow, shown in Fig. 2, is comprised of the sequence of activities to conduct an experiment. The task of the experimenter is to bootstrap and initialize the hardware, deploy the necessary software, orchestrate the whole experiment by executing commands on each node in the correct order and analyze the results. Focusing on local experiments, we assume that the experimenter has physical access to the hardware.

A. Bootstrapping of Experimental Hardware

The goal of this step is to **prepare the boot manager on the nodes** in order to minimize the manual interaction with the node. After bootstrapping, the nodes should support **remote management** of the deployment and boot process of the experiment-specific OS from the experiment controller machine. An optional, but very useful, feature is the ability to externally **control the power state** of the nodes, which allows for recovery from hard failure without manual intervention. Bootstrapping usually needs to be done only once per node.

B. Operating System Deployment

Having prepared the hardware, the experimenter is required to **select a Linux OS** distribution, that supports the software required to perform the experiments.

Following that, there are two distinct problems that need to be solved:

- How to prepare OS images?
- How to deploy the selected image(s) on a set of selected nodes?

Image Preparation: The lack of precise information about the OS image can cause problems while trying to reproduce an experiment [5]. It is therefore important to start from a well-defined base image and carefully document all installed software. One approach is to **download the default OS images** provided by Linux distributions, like Ubuntu. This requires that all additional software, like extensions or modifications to the standard Linux kernel or measurement tools (e.g. *iperf3*) is deployed at the start of the experiment. This can take significant time with big libraries and applications.

The other option is to **build customized OS images** using well-defined instructions, which benefits repeatability and replicability. It can help to minimize the overall effort for the experimenter, as the OS image can be build once and reused for other experiments. It allows others to review the contents of the OS image or re-use the customized binary. The rebuilding of OS images comes with some overhead, creating a trade-off: the more software is included in the prepared OS image, the less time it takes to start the experiment, but the more frequent it is necessary to rebuild the image. It is thus advisable to include only standard and slowly changing parts of the software stack in the OS image.

It should be possible to flexibly choose the OS and to **modify the image to meet the needs of the experiment**. The chosen OS, naturally, needs to support attached networking hardware, experiment-specific devices under test and software.

Image Deployment: The goal of this step is to provide the ability to **boot the selected OS** on a set of devices. After deployment, the experimenter should have exclusive control of a set of nodes via the control network from the experiment controller. Completely reloading the image on the nodes between experiments avoids leftover configuration from the previous repetition, or even worse, different experiments. Depending on the number of nodes, deploying the OS image can be challenging. One critical factor in this context is the time to get the OS up and running (booted and accessible from the control network). The options for deploying the image on a node depend on the particular hardware and capabilities of the used bootloader. An interesting feature is the ability to **debug the boot process of the experiment-specific OS**, in case of errors in the image preparation or due to modifications of the kernel.

C. Software and Configuration Deployment

The goals of the last preparation step before performing experiments are to **install the remaining software** and assign the proper configuration on a per-node basis. This includes software, that is being developed or is often changing. **Deploying different configurations** allows assigning roles to the respective nodes.

The challenge is to execute this step efficiently on multiple nodes, while allowing dynamic customizations on a per-node basis, using a common description or template. It is also important to be able to automatically verify the configuration of the nodes.

D. Execution of Experiments

Each experiment run is characterized by the corresponding test scenario. It consists of **the setting of configuration parameters**, running various applications, and doing corresponding measurements. Execution of the experiment can be seen as a logically consistent, properly ordered execution of individual actions on the whole set of nodes, including the collection of relevant data. This is by far the most complex part of the experimentation workflow and the concrete actions depend on the particular experiment setting and scenario.

The main issue tackled in this step is **control of actions** on a set of nodes. An action can be changing hardware parameters, for example, transmit power, starting or stopping processes, changing software configuration or triggering software or hardware via an application programming interface (API). The tools should be able to react upon predefined triggers, for example after a Wi-Fi connection has been established, and execute actions, like triggering a packet generator. The interactions, regarding control of the nodes, should be supported in a unified way. Error-handling is also crucial in order to inform the experimenter of a potential misconfiguration or un-expected events early on.

It needs to be possible to easily execute the same experiment multiple times, with the same or different parameters. Finally, **data needs to be retrieved** from the nodes to the experiment controller machine for further processing.

E. Data Analysis

The last step is the analysis of the data gathered from multiple experiment runs. This can be done on the experiment controller machine. It starts with the **loading and preparation of measurement data**. Next, the required **analysis** has to be performed. It usually ends with **the generation of graphs and diagrams**, which provide insight into the obtained data.

Analysis can range from simple statistical modeling to advanced machine learning approaches, depending on the particular experiment. An advanced feature that the data analysis tools could provide is to dynamically control the experiment based on results.

IV. Toolchain for Experimentation

In this section, we discuss tools that support each of the steps within the workflow. A key observation is that many of the tasks, marked in bold in the previous section, are not specific to experimentation, but are supported by widely used IT-automation tools. Therefore, instead of only considering dedicated tools for experimentation, we include state-of-the-art tools from closely related fields. We find that these tools have much wider support, with bugs being fixed and features being continuously added. Additionally, learning a tool that is heavily used in the IT industry is likely to pay out more than investing time for a tool, that could be specific to a single testbed.

The summary of the considered tools and the supported workflow stages is presented in Tab. I. We analyze each mentioned tool with respect to the following functional requirements (when applicable[2]):

- *Modularity*: Parts of the code or configuration created with the tool can easily be modified or re-used in a different context.
- *Scalability*: Easily works with a high number of nodes;
- *Support*: Is under active development and has a big community of users;
- *Ease of installation*: Can be installed and used effortlessly;
- *No additional infrastructure*: Requires no additional infrastructure, like a server or agents installed on each node;
- *Wide applicability*: Can be used in a broad scope of use-cases;
- *Usable in ad-hoc setups*: Is usable in local and ad-hoc experimental setups;
- *Usable in testbeds*: Is usable in a testbed environment;
- *Language*: The programming language used by the user of the tool.

We have selected the smallest set of tools (marked in the table) fulfilling the following criteria: (1) The set supports all steps of the workflow. (2) The individual tools are feature-rich. (3) The tools use the same language. The last criterion reduces the cognitive burden of learning and using different programming languages and concepts for the experimenter. In this case, we have selected *Python*, as it is an open

[2]Marks in brackets denote partial support.

TABLE I
TOOLBOX SUMMARY

		kexec	PXE	**diskimage-builder**	Default OS image	OpenWRT image builder	**Ansible**	jfed gui	omni	**Fabric**	Chef	Puppet	omf	nepi-ng	**jupyter pandas**	LabView	Matlab	R
Workflow stage	Bootstrapping of Hardware	X	X															
	OS Image Preparation			X	X	X							(X)					
	OS Image Deployment						X	X	X	(X)			(X)					
	Software and App. Deployment						X			X	X	X	X					
	Execution of Experiments									X			X	X		X		
	Data Analysis														X	(X)	X	X
Features	Modularity			X			X			X	X	X		X	X		X	X
	Scalability			n/a	n/a	n/a	X		X	X	X	X	X	X	n/a		n/a	n/a
	Support	X	X	X	X	X	X			X	X	X		(X)	X	X	X	X
	Ease of installation			X	X		X			X				X	X			X
	No Additional infrastructure	X		n/a	n/a	n/a	X			X				X	X		n/a	n/a
	Wide applicability			(X)	X		X	X	X	X	X	X			X		X	X
	Usable in ad-hoc setups	X		X			X			X	X	X	(X)	X	n/a		n/a	n/a
	Usable in testbeds	n/a	n/a				X	X	X	X	(X)	(X)	X	X	n/a		n/a	n/a
	User interface or language	n/a	n/a	Python	Shell	Shell	Yaml Python	GUI	Shell	Python	Ruby	DSL	Ruby	Python	Python	GUI	Matlab	R

source programming language, is used to develop many of the automation tools and is available by default in most of the Linux distributions.

For the following step by step description, we assume that the experimenter has installed the marked tools presented in Tab. I on the experiment controller machine[3]. We recommend using the latest available version of the given tool. Python package managers, like `pipenv`, allow reproducing the setup, by keeping track of used versions of all packages. Another general guideline for improving repeatability is to treat storage at the experiment nodes as ephemeral. This ensures that the whole experiment can always be executed from the controller machine.

A. Bootstrapping of Experimental Hardware

To avoid dependency on a hardware specific bootloader, we utilize the *kexec* [11] Linux kernel feature. It is used to dynamically boot into the OS in place, i.e. without having to go through a bootloader. Hence, this method is applicable to a wide range of hardware.

To prepare the node it is necessary to manually install Ubuntu, or any other Linux distribution, on the node. This installation will only be used as a basis to load OS images for experimentation. The kexec method does not require any specific hardware support. Additionally, in Ubuntu, Python support is available by default, ssh access can be enabled during installation and kexec can be added by installing the `kexec-tools` package. To be able to remotely access the node via *ssh* from the controller machine, it is also necessary to set up IP-based networking. Under normal circumstances (no major faults), power control of the nodes can be assured using standard Linux tools.

[3]With the exception of *kexec*, which needs to be installed only on experimental nodes.

B. Operating System Image Preparation

The OS images can be prepared on the experiment controller machine. *diskimage-builder* [12] allows preparing ready-to-use OS disk images of various Linux distributions.

The experimenter can extend a default configuration or develop own elements in order to meet his or her requirements. This approach is relatively easy to use, yet allows for a good level of customization, which is explicit (configuration files) and can easily be reused, modified and shared. In addition to the experiment-specific packages, the created OS image has to include Python, as it is required by other tools like Ansible.

The following example illustrates how to create an Ubuntu 18.04 (bionic) OS image with openSSH-server installed and enabled by default:

```
DIB_RELEASE=bionic disk-image-create ubuntu
 ↪  openssh-server -t tgz -o image.tgz
```

C. Operating System Image Deployment

Next, the experimenter needs a way to deploy and boot the OS image on a specified set of nodes. First, it is necessary to copy the experimental image to a separate partition on each corresponding target node (running the basis OS as bootloader replacement).

Booting a kernel with *kexec* consists of loading the target kernel into memory and then executing it, as shown in an example below. Kexec allows to pass it the kernel command line parameter, specifying the device (partition), on which the experiment-specific OS image resides.

```
kexec -l /mnt/boot/vmlinuz-4.11.3-041103-generic
 ↪  --initrd
 ↪  /mnt/boot/initrd.img-4.11.3-041103-generic
 ↪  --command-line "root=/dev/vg_images/default ro
 ↪  netconsole=6666@10.1.1.101/eth0,5555@10.1.1.5/"
kexec -e
```

The `netconsole` configuration, although optional, allows for remote analysis of the boot process.

Ansible [13] allows to specify the above tasks, like executing kexec commands, in a configuration file, called playbook, once and to execute them on all nodes in parallel using one command.

The feature of remote power state control of the nodes is not explicitly supported. However, with proper use of the remote shell (ssh) and Ansible, reboots can be triggered remotely under normal operating conditions. Manual reboot remains necessary only in case of node misconfiguration and or severe faults in the software or control network.

D. Software and Configuration Deployment

Usage of IT automation tools allows to precisely express the configuration and target state of a node in a standardized language, such that it can be shared and repeated effortlessly. The following example shows an *Ansible* playbook, that makes sure *iperf3* in version `3.0.11` is installed on all nodes.

```
- name: First Playbook
  hosts: all
  become: yes
  tasks:
  - name: Install iperf3
    apt:
      name: iperf3=3.0.11*
```

E. Execution of Experiments

In Linux, actions are typically triggered by executing shell commands. *Fabric* [14] provides a suite of operations for executing local or remote shell commands, uploading or downloading files, as well as auxiliary functionality such as prompting the running user for input. The example shows how to implement a throughput measurement between two nodes together with the collection of measurement results to the experiment controller machine.

```
from fabric import Connection
server, client = "giga1", "giga2"

server_c = Connection(server)
client_c = Connection(client)

server_c.run("iperf3 -s -D -1 -i 1 --json")
client_c.run(f"iperf3 -c {server} -t 15 -i 1 --json
 ↪  --get-server-output > /tmp/{client}.json")

client_c.get(f"/tmp/{client}.json",
 ↪  f"./{client}.json")
```

Experiment control, using Fabric, is based on triggers registered by the Fabric script on the controller machine, for example when an application running on a node finishes or the wireless link is operational. This allows controlling the beginning of the next control action after the completion of the previous one. This level of controlling the experiment execution is frequently sufficient. There are experiments which might benefit from explicit temporal control of events, like the simultaneous execution of actions. Fabric provides only limited support for such operation.

F. Data Analysis

The final step after performing experiments is the analysis of results. The Python ecosystem provides a powerful selection of data analysis tools. An in-depth discussion can be found in [15].

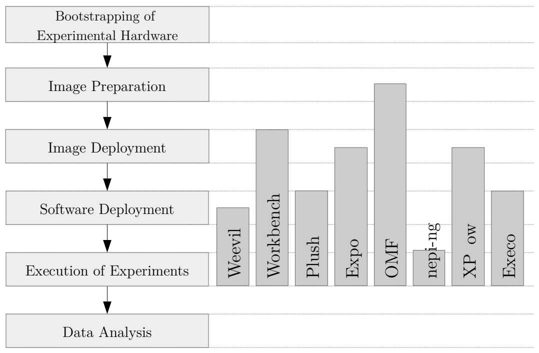

Fig. 3. Dedicated tool overlap with the workflow

V. Using the Workflow with Testbeds

Testbeds (like ORBIT [16], w-iLab.t [1]) offer large-scale experimentation facilities. Testbed providers take care of the experimentation nodes, the environment they are deployed in and provide fast, reliable control channel. Testbeds allow to reserve and access nodes for experimentation over the Internet. After reservation, the experimenter can use the testbed API to select the OS image for each node, boot the nodes and gains full `root` access.

This frees the experimenter from the first few tasks of the workflow, i.e. bootstrapping, OS image preparation and OS deployment. These steps are under the control of testbed provider with functionalities exposed through a testbed specific API. The Slice-Based Federation Architecture (SFA) [17] is currently the de-facto standard for such API. *jfed* [18] and *omni* [19] allow to reserve nodes on the testbed and can be used as a drop-in replacement of the tools suggested for OS deployment.

Many testbeds provide pre-configured disk images for the experiment nodes. Those images are equipped with default software and work out of the box on the respective testbed. The experimenter, having full permissions, can extend the image by installing software, i.e. software deployment workflow step. However, the building process of these images is often nontransparent and base images can be modified (e.g. updated) without notice. The SFA API allows for providing own OS images, like those prepared in the OS image preparation step. Depending on the particular testbed infrastructure, it might be required to modify an image in order to successfully boot it on the respective testbed.

All steps of the workflow following the deployment of the customized OS, are supported by the tools described in Section IV. In summary, the discussed workflow and suggested toolchain can be used together with the testbed APIs, benefiting from the additional support by the testbed.

VI. Related Work

There are several dedicated experimentation tools supporting the experimentation process, summarized in a survey by Buchert et al. [20]. All of those tools closely follow the general experiment architecture but only loosely follow the workflow and do not explicitly name it. A detailed mapping of the tools to their supported workflow steps is provided in Fig. 3. The tools focus on experiment execution with an emphasize of monitoring and data management functionalities, but also provide support for software deployment and configuration management. At the time when most special purpose tools have been developed, the DevOps approach and tools have not yet been mature enough. Out of the tools analyzed in [20], only OMF [21] and *nepi-ng* [22] seem to be used in the Fed4FIRE(+) testbed federation. They have been included in the analysis. We found that the advent of corresponding IT automation tools are supported by a much bigger community and cover the features of the dedicated experimentation tools.

Closely related to the discussion about parallels between experimentation and DevOps, Vucnik et al. [2] adopt the continuous integration and continuous delivery (CI/CD) methodology into wireless network experimentation. They illustrate the capabilities of such approach by extending the LOG-a-TEC testbed and demonstrating the effectiveness in the context of multi-technology 5G machine-type communications.

VII. Final Comments

Usage of the described workflow and tools should improve the structure of the experimentation process by allowing to clearly split concerns and by explicitly solving each subproblem with playbooks and scripts. The resulting implementation of the whole experiment in code aids the application of good experimentation practices and benefits repeatability.

The workflow and proposed toolbox can be used with ad-hoc experimentation networks, is equally applicable for publicly available testbeds and is relevant in the context of experiments running on a commercial cloud. This is particularly useful for doing local experiments and then using the same code at a big scale testbed or cloud for a more in-depth measurement campaign.

We leveraged a set of DevOps related projects and open source tools, that have good documentation, are widely used and can be applied in multiple circumstances. They can be, in some cases, replaced by alternatives. We presented an example experiment and we encourage to repeat our experiments and analysis.

References

[1] S. Bouckaert, W. Vandenberghe, B. Jooris, *et al.*, "The w-iLab.t testbed," in *TridentCom 2010*, vol. 46, 2010.

[2] M. Vučnik, T. Šolc, U. Gregorc, *et al.*, "COINS : ContinuOus IntegratioN in wirelesS technology development," *Netw. Test. Anal. Ser. IEEE Commun. Mag.*, pp. 1–9, August 2018.

[3] Q. Scheitle, M. Wählisch, O. Gasser, *et al.*, "Towards an Ecosystem for Reproducible Research in Computer Networking," in *Reproducibility Workshop*, ACM, 2017, pp. 5–8.

[4] M. Marquis, J. Baillieul, L. Hall, *et al.*, "Report on the First IEEE Workshop on The Future of Research Curation and Research Reproducibility," 2016.

[5] M. N. Mahfoudi, T. Turletti, T. Parmentelat, *et al.*, "Lessons Learned While Trying to Reproduce the OpenRF Experiment," in *Reproducibility Workshop*, ACM, 2017, pp. 21–23.

[6] Puppet and DevOps Research and Assessment, "State of DevOps Report 2017," 2017, pp. 31–31.

[7] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. 2015.

[8] L. Krüger, K. Kuladinithi, M. Mühleisen, *et al.*, "Minimising the Entry Effort in Experimenting with WSN Testbeds to Perform Heterogeneous Research," in *Real-WSN'15*, ACM, 2015, pp. 19–22.

[9] R. Natarajan, P. Zand, and M. Nabi, "Analysis of coexistence between IEEE 802.15.4, BLE and IEEE 802.11 in the 2.4 GHz ISM band," in *IECON 2016*, Oct. 2016, pp. 6025–6032.

[10] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, *et al.*, "srsLTE: An Open-source Platform for LTE Evolution and Experimentation," in *WiNTECH '16*, ACM, 2016, pp. 25–32.

[11] (2018). Kexec(8): Directly boot into new kernel - Linux man page. 07/2018, [Online]. Available: https://linux.die.net/man/8/kexec (visited on 07/17/2018).

[12] OpenStack Foundation. (2018). Diskimage-builder Documentation. 07/2018, [Online]. Available: https://docs.openstack.org/diskimage-builder/.

[13] Red Hat Inc. (2018). Ansible Automation For Everyone. 07/2018, [Online]. Available: https://www.ansible.com/.

[14] J. Forcier. (2018). Fabric's Documentation. 07/2018, [Online]. Available: http://www.fabfile.org/.

[15] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.", Sep. 25, 2017, 786 pp.

[16] D. Raychaudhuri, I. Seskar, M. Ott, *et al.*, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," *IEEE Wirel. Commun. Netw. Conf. 2005*, vol. 3, pp. 1664–1669, 2005.

[17] L. Peterson, R. Ricci, and A. Falk, "Slice-based federation architecture," *Control*, pp. 1–18, July 2010.

[18] (2018). jFed - Java-based framework for testbed federation, [Online]. Available: http://jfed.iminds.be/.

[19] (2018). Omni. 07/2018, [Online]. Available: https://github.com/GENI-NSF/geni-tools/wiki/Omni.

[20] T. Buchert, C. Ruiz, L. Nussbaum, *et al.*, "A survey of general-purpose experiment management tools for distributed systems," *Future Generation Computer Systems*, vol. 45, pp. 1–12, Apr. 1, 2015.

[21] T. Rakotoarivelo, M. Ott, G. Jourjon, *et al.*, "OMF: A Control and Management Framework for Networking Testbeds," *SIGOPS Oper Syst Rev*, vol. 43, no. 4, pp. 54–59, 2010.

[22] T. Parmentelat, T. Turletti, W. Dabbous, *et al.*, "Nepi-ng: An Efficient Experiment Control Tool in R2Lab," in *ACM WiNTECH*, New York, NY, USA, 2018.