Integrating research testbeds into social coding platforms

Matevž Vučnik, Carolina Fortuna, Tomaž Šolc and Mihael Mohorčič
Jožef Stefan Institute, Ljubljana, Slovenia
{matevz.vucnik, carolina.fortuna, tomaz.solc, miha.mohorcic}@ijs.si

***Abstract*—The adoption of social coding platforms among software developers is particularly high because it increases collaboration and productivity as well as code re-use. On the other hand research testbeds do not generally have such high adoption rate. The learning curve of adopting new technology represents an initial drop in performance before it increases. However this effect is unexpected by many users and there is a risk that they will abandon new technology before their performance increases. The main contribution of this paper is to introduce methodology to take advantage of the high adoption rate of social coding platforms to improve the adoption of research testbeds. The proposed approach is that a social coding platform serves as a common gateway to various testbeds since it is something many developers are comfortable with and thus the initial effort needed to start using the testbed is decreased and the probability that experimenters continue to use a testbed is higher.**

I. Introduction

Working with testbeds [1] we observed a trend that testbeds sometimes are not utilized to their full potential since the initial effort required to bring to life even the simplest example is often very high. Testbed providers many times implement their own special registration system responsible for user registration, creating user accounts and managing user rights [2]. This system is the first thing an experimenter will encounter. Extensive studies have been done in the field of technology adoption and have proven that difficulty of use can discourage adoption of an otherwise useful system [3].

In [4] the authors go even further and present the learning curve of relative performance over time Figure 1. Initially the learning curve goes below 0 on the relative performance scale which means that at the start of using a new system user performance is worse than before although in time the performance will increase. This effect is unexpected by many users and there is a risk that they will abandon new technology before their performance increases and they realize the benefit of it.

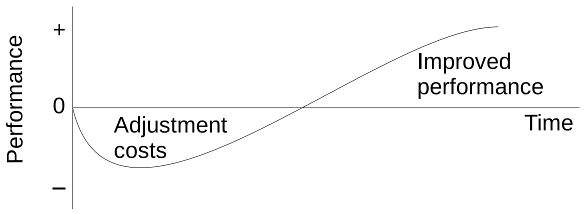

Fig. 1. Relative performance over time with learning effect as illustrated in [4, Fig. 1].

Based on this knowledge we can safely assume that the initial interaction with the system needs to be as pleasant as possible this way keeping the time spent "below 0" as short as possible thus increasing the probability of adoption. One way to achieve this goal is to integrate the testbed into an already successfully adopted platform. Social coding platforms adoption is considerably high among software developers [5] therefore the research testbed integrated in one of those platforms would automatically benefit from its adoption rate.

The proposed methodology of integrating a wireless experimentation testbed into a social coding platform is suitable for any testbed provider that offers some kind of hardware to experimenters. The idea is that a social coding platform serves as a common gateway for various testbeds since it is something many developers are familiar with and thus the initial effort needed to start using the research testbed is decreased and the probability that experimenter continue using the testbed are higher. Furthermore many potential experimenters already have user accounts created on at least one social coding platform therefore there is no need to create a new one and the administrator can simply grant an existing user the right to access the testbed repository and she/he can start experimenting.

At the moment of writing three the most popular social coding platforms are GitHub[1], Bitbucket[2] and Gitlab[3] [6]. Our reference implementation of the proposed methodology is based on social coding platform GitHub although near identical integration could be achieved using any of the two remaining platforms.

The rest of the paper is structured as follows. Section II positions our work against the others. Section III details the reference architecture and implementation of the integration. Finally, Section IV identifies the future work and concludes the paper.

II. Related work

Open access testbeds in Europe are brought together under various European projects more notably FED4FIRE[4] and WiSHFUL[5]. FED4FIRE defines a common access mechanism for different testbeds while also taking care of unifying the experiment control based on OMF framework [7]. Wishful

[1]GitHub, https://github.com
[2]Bitbucket, https://bitbucket.org
[3]Gitlab, https://gitlab.com
[4]FED4FIRE, https://www.fed4fire.eu
[5]WiSHFUL, http://www.wishful-project.eu

978-1-5386-1478-5/18/$31.00 © 2018 IEEE

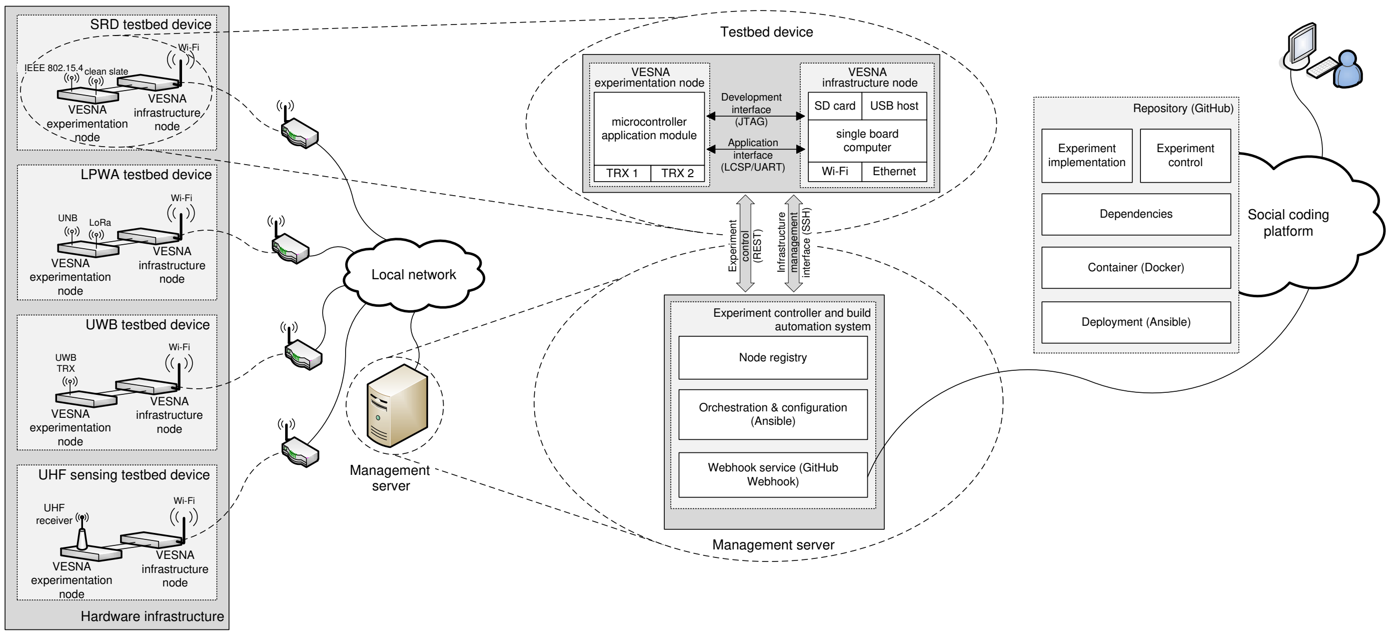
SRD testbed device
IEEE 802.15.4
clean slate
Wi-Fi
VESNA experimentation node
VESNA infrastructure node
LPWA testbed device
UNB
LoRa
UWB testbed device
UWB TRX
UHF sensing testbed device
UHF receiver
Hardware infrastructure
Local network
Management server
Testbed device
VESNA experimentation node
microcontroller application module
TRX 1
TRX 2
Development interface (JTAG)
Application interface (LCSP/UART)
VESNA infrastructure node
SD card
USB host
single board computer
Ethernet
Experiment control (REST)
Infrastructure management interface (SSH)
Experiment controller and build automation system
Node registry
Orchestration & configuration (Ansible)
Webhook service (GitHub Webhook)
Repository (GitHub)
Experiment implementation
Experiment control
Dependencies
Container (Docker)
Deployment (Ansible)
Social coding platform

Fig. 2. Architecture of the reference integration.

project develops unified programming interfaces (UPIs) as a radio and network control abstraction [8] which are particularly useful as a basis for experiment automation. Some testbeds follow the FED4FIRE authentication scheme and the others have developed their own user registration and user management systems.

US research testbeds are federated under the Global Environment for Networking Innovation GENI[6] [9]. GENI approach to testbed experimentation is more traditional trying to replicate the network simulator experience by implementing a fully functional GUI [10]. Current trends in 5G go in the direction of software defined radios and networks [11], [12] however testbed experimentation have yet to follow the software defined approach. Popular 5G related SDR social coding projects among others include Microsoft OTP4LTE-U[7] and SRS srsLTE[8]. OTP4LTE-U project develops experimental LTE/LTE-U PHY and MAC layers while srsLTE offerers a more complete LTE software suite up to IP layer. Both projects support commercial off-the-shelf SDR hardware such as bladeRF[9] and USRP[10].

After examination of different testbed access systems we observed that it is still a common practice for testbeds to use a custom user registration and management system. For each testbed an experimenter needs go through a different registration process to create a new user account before she/he can start experimenting. By integrating testbeds into a social coding platform and automating the build process it is possible to completely avoid multiple user accounts furthermore an experimenter can immediately start by running an example experiment with minimal effort. After the example experiment is done an experimenter can continue by adjusting some trivial parameters and observing the results to get the filling of using the testbed therefore not spending so much time reading the documentation and possibly losing the interest to go deeper.

Beyond the user registration and management we observed that testbeds generally already provide good abstraction layers or frameworks required to support automation. Furthermore many times the code is hosted on one of the social coding platforms i.e. GitHub while the integration with the testbed itself is typically missing. From what we learned about different experimental testbeds we can conclude that social coding platform integration is achievable by any of the testbeds we examined.

III. Integration

In order to make low level wireless firmware experiments based on social coding platforms functionality possible, we had to introduce multiple layers of abstraction. The abstraction is in form of software modules and libraries as well as hardware. After researching the related work we can conclude that testbeds generally already implement most of the needed abstractions for successful social coding platform integration. Before the integration we need to ensure that the experiment build is completely automated and that everything needed for the actual build is hosted inside the repository located on social coding platform.

We identified three features a social coding platform needs to support for the successful integration with wireless experimentation testbed:

1) Creating releases or tags from repository
2) Triggering a webhook
3) Uploading and appending files to the created releases

Creating a release or tag takes a current snapshot of the repository and attaches a version number and the description

[6]GENI, http://www.geni.net
[7]OTP4LTE-U, https://github.com/Microsoft/OTP4LTE-U
[8]srsLTE, https://github.com/srsLTE/srsLTE
[9]bladeRF, https://www.nuand.com
[10]USRP, https://www.ettus.com

this way we can reproduce this experiment results at any time. Social coding platform needs to support webhook triggering mechanism this way the testbed management server can initiate a build wen a new release is created. There should be an option to append a file to the created release when the experiment is done thus the collected results are made available to the experimenter for download and analysis. Additionally Wiki support is also desired to host the testbed documentation and testbed topology along with the code this way the testbed experimenter has all the needed information in one place.

During the extension of our testbed LOG-a-TEC [13] we upgraded the architecture depicted on Figure 2, by defining a completely new experiment deployment and execution system based on Ansible[11] orchestration and container technology Docker[12]. The reference implementation combines several features that comprise a complete testbed management, monitoring and experimentation solution. Internally, it is a complex setup composed of self-sufficient systems packaged in Docker containers. Each container includes all dependencies with the purpose of being easily redistributable. Externally, the management server exposes a user-facing web interface and an HTTP REST API.

A. *Hardware*

The hardware infrastructure is composed of several types of testbed devices, which are comprised of two VESNA (VErsatile platform for Sensor Network Applications) nodes, as depicted on Figure 3. Notably, the SNA-LGTC experimentation nodes host several different radio interfaces and thereby support experiments in various frequency bands. In particular, we classify them to those enabling experimentation with: (i) LPWA communications, (ii) short range UWB communication, (iii) advanced spectrum sensing in sub-1 GHz bands, (iv) efficient lightweight protocols in 2.4 GHz and sub-1 GHz bands that are not based on IEEE 802.15.4 (clean slate) and (v) spectrum sensing and signal/interference generation by reconfigurable transceivers.

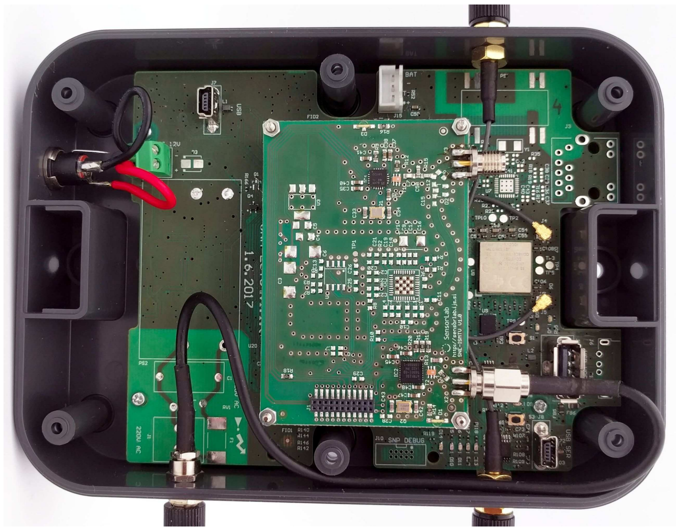

Fig. 3. SNA-LGTC assembled.

[11]Ansible, https://www.ansible.com

[12]Docker, https://www.docker.com

The infrastructure node is a custom-designed single board computer based on the BeagleCore[13] module running the GNU/Linux operating system. It features wired Ethernet and Wi-Fi for infrastructure connectivity, SD card slot for storage expansion, VESNA and USB interfaces for extensibility and development/debug process. The interfaces are depicted on Figure 4. The experimentation node is custom-designed board based ARM Cortex-M3 microcontroller running low level bare metal firmware. It has a modular design and is able to host different application modules with dedicated experimentation transceivers. Both nodes are interconnected via application and development interfaces, providing the exchange of application data as well as remote low-level application debugging. The development interface is used for programming and debugging of the experimentation node, while the application interface is used for communication while the experimentation node is in operation. The development interface is based on JTAG and for the application interface protocols such as LCSP (Light-weight Client Server Protocol; an HTTP-inspired protocol) running on top of serial interface are used. Optionally more standard protocols can be used for the application interface such as CoAP protocol on-top of IPv6.

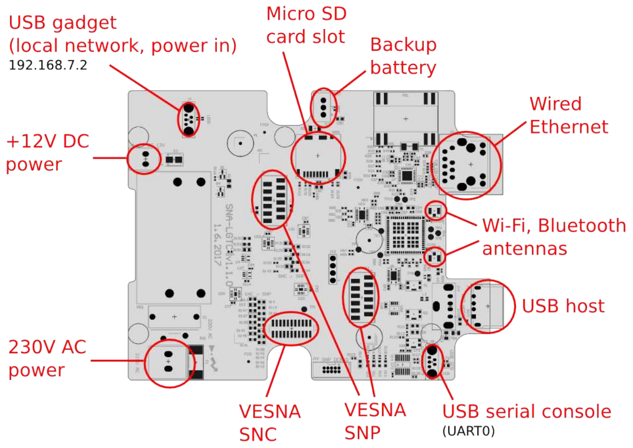

Fig. 4. SNA-LGTC connectors.

The main reason for the separation of the hardware infrastructure in two functional blocks is to have a generic infrastructure node that can be combined with various experimentation nodes to support a range of heterogeneous experiments but keep a unified way of nodes management and reconfiguration. This way, the infrastructure node which takes care of communication for the remote management and hosting containers can use more reliable components ensuring better uptime, whereas the experimentation node that is more prone to errors and crashes is executed on a separate hardware and configured within a container. An additional benefit is that the required dependencies do not need to be installed on the infrastructure node; instead, all the required experimentation tools including all the dependencies are packaged in a redistributable container image which can be safely downloaded and executed on the infrastructure node without sacrificing reliability.

[13]BeagleCore, http://beaglecore.com

B. Experiment description

The experiment description is based on the VESNA Management System (VMS) with custom developed protocol LCSP (Lightweight Client Server Protocol) for communication between VESNA and the infrastructure node. On top of this protocol we implemented an abstraction library called ALH Tools which serves as basis for developing distributed wireless experiments.

The LCSP protocol is based on client-server architecture. In our case the server side is on VESNA sensor nodes and the client side is on a PC hosting VMS. In order to access the resources of nodes, the gateway has to establish a connection with the management system. This is done by setting up a serial/TCP/SSL link. Resources that are pre-prepared on the nodes are exposing different features. In the following we describe the LCSP protocol characteristics which are essential for understanding of the management system.

LCSP was developed for communication between sensor network and a remote server. It was inspired by the HTTP protocol, but purposely kept simple for fast and easy implementation on VESNA nodes. The protocol defines two methods, GET and POST, which are understood by each VESNA node. GET method is used for "safe" requests which do not change the state of the system. POST is used for "unsafe" requests which can change the state of the system. The response is considered to be in a binary format although it is normally in a text format except the spectrum sensing data which is in binary. Each response ends with a sequence `OK\r\n` to mark the end of the response. The format of the request and reponse is shown in Listing 1.

The protocol includes simple and efficient error handling mechanism. There are two types of errors that can be encountered as a response to these requests. The first type is `JUNK-INPUT` which is the more common situation when someone mistypes the resource name and the parser on the node does not recognize it. After this response, the parser on the node expects 5 new lines which reset the parser before one can try to access the resource again. The second type of error is `CORRUPTED-DATA`, meaning that CRC check did not pass successfully. Thus we can conclude that the error happened somewhere on the link between the infrastructure and the gateway. This situation will occur with lower probability.

Additionally VMS implements an HTTP API which translates HTTP requests to LCSP requests. The call to the API has to meet the specified form for GET and POST request. In order to make a GET or a POST request we have to make a call to the handler called "communicator" located on the web server inside the management system. Requests that do not meet the template are rejected by the VMS. The call includes several parameters:

- Cluster - corresponds to the serial/TCP/SSL/ port which VESNA is connected to.
- Method - can be GET or POST.
- Resource - corresponds to the resource name located on the target node.
- Content - a specific parameter of POST requests specifies the data transmitted to nodes.

```
Client requests must be formated:

GET resource?arg1=val1&...&argN=varN\r\n

  resource: abstract resource identifier
  examples: - firmware/version
            - sensors/temperature
  arg1:     parameter 1 name
  val1:     value of parameter 1

  argN:     parameter N name
  valN:     value of parameter N

POST resource?parameters\r\n
Length=len\r\n
<data having len bytes length>\r\n
crc=crc_value\r\n

  resource:   abstract resource
              for example: firmware
  parameters: parameters given to the
              POST handler same format
              as GET parameters
  len:        length of the data that
              will be written to the
              specific resource
  data:       possibly binary data
              to be transmitted
  crc_value:  CRC value calculated on all
              the previous content except
              the line starting with crc=;
              value represented as an
              unsigned decimal number

Server responses must be formated:

<response to a specific request>\r\n
\r\n
OK\r\n

Error messages must be formated:

<response with error description>\r\n
<JUNK-INPUT or CORRUPTED-DATA>\r\n
\r\n
OK\r\n
```

Listing 1. LCSP request and response format

```
api?cluster=port&method=get&resource=name
...method=post&resource=name&content=content
```

The ALH Tools package provides utilities and modules for managing VESNA-based wireless sensor networks that are using the proprietary LCSP protocol. In a typical setup, VESNA nodes participate in the network with an infrastructure PC connected over serial/TCP/SSL. In this network, each sensor node exposes an HTTP-like interface, supporting two types of requests: GET and POST. The TCP/SSL tunnel terminates in an infrastructure server that performs the translation between the LCSP protocol and a proper HTTP REST interface exposed on the web.

```
              serial/TCP/SSL
+-----------+          +-----------+
|   VESNA   | LCSP --- |    VMS    | HTTP -->
+-----------+          +-----------+
```

Alternatively VESNA can also be directly connected to a client over a serial line. This setup is typically used for the development or debugging. The ALH Tools with LCSP protocol transparently support both modes of operation. Optionally either an URL of an HTTP REST endpoint is given or a character device for the serial line.

ALH Tools are implemented in Python programming language and can be installed as a standard Python package by running:

```
$ pip install vesna-alhtools
```

C. Experiment deployment and build automation

The experimentation system was designed to abstract the underlying complexity and enable the desired workflow. Everything needed for the experiment is available in a GitHub repository starting with the Ansible playbook which gets executed when new release is created in the repository by triggering GitHub Webhook listening on experiment controller, which is part of the central management server. The playbook contains a description of the experiment setup, which includes the list of testbed devices performing the experiment while the node registry holds all available testbed devices. There are two parts of the experiment description first will be executed on the target experimentation nodes and the second in the experiment controller. The experimentation node will download the GitHub repository containing the Docker file which is the description of the automated build and the actual experiment code. The Docker image will not be built each time from scratch; the testbed generic image will be downloaded from Docker store, which has all the needed dependencies and services already installed and set up. Thus, the experimenter only needs to write the code for the actual experiment performed on the node and the part performed by the experiment controller.

An important part of the experiment setup process is the flashing of the experimentation node. This can be done by choosing an existing embedded binary from the list of pre-prepared embedded images suitable for the experiment, or by specifying own source to be compiled for the experimentation node.

D. Result collection and publishing

Results produced by the distributed experiment framework ALHtools are copied using Ansible commands to the experiment controller which will eventually compress these results in a single zip file along with the other documents produced by running the experiment. When the process is complete, the experiment controller will automatically upload the resulting zip file to social network platform repository created by the new release which initiated the experiment build. This way the results become available to to the experimenter for download and post processing. The results can either be in a raw binary format or text format or some other standard format i.e. JPEG or PNG. The format of the results is actually defined by the experiment description.

IV. Conclusions and future work

In this paper we showed that wireless testbeds can be integrated with social coding platforms thus avoiding unnecessary user account management and making testbeds more attractive by lowering the initial effort needed to start using experimental facilities. One direction of future work is the automated experiment scheduler. Currently testbed resources are manually and statically assigned during the experiment description therefore only one user at the time is allowed to access the testbed. The idea is to implement automated time, frequency and space devision multiplex scheduler for the testbed while also keeping experiment results reproducible.

Acknowledgment

This work was partly funded by the Slovenian Research Agency (Grant no. P2-0016) and the European Community under the H2020 eWINE (Grant no. 688 116), WiSHFUL (Grant no. 645 274) and Fed4FIRE+ (Grant no. 732 638) projects.

References

[1] T. Šolc, C. Fortuna, and M. Mohorčič, "Low-cost testbed development and its applications in cognitive radio prototyping," in *Cognitive radio and networking for heterogeneous wireless networks*. Springer, 2015, pp. 361–405.

[2] P. Mueller, "Software defined testbed." Joint Expert Group and Vision Group Workshop, 2015. [Online]. Available: http://dspace.icsy.de/handle/123456789/407

[3] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[4] C. F. Kemerer, "How the learning curve affects case tool adoption," *IEEE Software*, vol. 9, no. 3, pp. 23–28, May 1992.

[5] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (r) evolution of social media in software engineering," in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. New York, NY, USA: ACM, 2014, pp. 100–116.

[6] J. Jiang, L. Zhang, and L. Li, "Understanding project dissemination on a social coding site," in *2013 20th Working Conference on Reverse Engineering (WCRE)*, Oct 2013, pp. 132–141.

[7] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "Omf: A control and management framework for networking testbeds," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 54–59, Jan. 2010.

[8] C. Yang, S. Byeon, P. Ruckebusch, S. Giannoulis, I. Moerman, and S. Choi, "Implementation of phy rate and a-mpdu length adaptation algorithm on wishful framework," *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 8–13, 2017.

[9] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014.

[10] J. Duerig, R. Ricci, L. Stoller, M. Strum, G. Wong, C. Carpenter, Z. Fei, J. Griffioen, H. Nasir, J. Reed, and X. Wu, "Getting started with geni: A user tutorial," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 1, pp. 72–77, Jan. 2012.

[11] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "Openairinterface: A flexible platform for 5g research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, Oct. 2014.

[12] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, H. Taoka, H. Tullberg, M. A. Uusitalo, B. Timus, and M. Fallgren, "Scenarios for 5g mobile and wireless communications: the vision of the metis project," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 26–35, May 2014.

[13] C. Fortuna, A. Bekan, T. Javornik, G. Cerar, and M. Mohorčič, "Software interfaces for control, optimization and update of 5G machine type communication networks," *Computer Networks*, 2017.