OpenBenchmark: Repeatable and Reproducible Internet of Things Experimentation on Testbeds

Malisa Vucinic, Bozidar Skrbic, Enis Kocan, Milica Pejanovic-Djurisic, Thomas Watteyne

OpenBenchmark: Repeatable and Reproducible Internet of Things Experimentation on Testbeds

Mališa Vučinić*, Božidar Škrbić§, Enis Kočan§, Milica Pejanović-Djurišić§, Thomas Watteyne*
* EVA team, Inria, Paris, France
§ University of Montenegro, Faculty of Electrical Engineering, Podgorica, Montenegro
{malisa.vucinic,thomas.watteyne}@inria.fr, {bozidars,enisk,milica}@ucg.ac.me

***Abstract*—Experimentation on testbeds with Internet of Things (IoT) devices is hard. The tedious firmware development, the lack of user interfaces, the stochastic nature of the radio channel, the testbed learning curve, are some of the factors that make the evaluation process error prone. The impact of such errors on published results can be quite unfortunate, leading to misconclusions and false common wisdom. The selection of experiment conditions or performance metrics to evaluate one's own proposal may not lead to perfectly fair comparisons with state-of-the-art, either. Our research community is well aware of these problems and is actively working on solutions. We present `OpenBenchmark`, a cloud-based, reproducible, repeatable and comparable IoT benchmarking service. `OpenBenchmark` facilitates and improves the IoT experimentation workflow: it runs the experiments on supported testbeds, instruments the supported firmware according to the industry-relevant test scenarios, and collects and processes the experiment data to produce Key Performance Indicators (KPIs). This paper introduces the `OpenBenchmark` platform, discusses its applicability, design and implementation.**

***Index Terms*—Internet of Things, Experimentation, Testbed, Repeatability, Reproducibility, 6TiSCH.**

I. Introduction

Experimental research has been an indispensable instrument in transitioning the Internet of Things (IoT) from a theoretic concept to state-of-the-practice. Testbed infrastructure on IoT communication technologies is present worldwide [1], [2]. It is freely open in the majority of cases. Large research projects [1] even fund its use. Even so, community in our domain seems to agree [3]: we still have quite some room for improvement in making experimental evaluations on testbeds fairly comparable, reproducible and repeatable, and most of all, relevant to the industry [4].

Open-source IoT operating systems have been around for over a decade [5]. Networking stacks [6] implementing the latest Internet standards are freely available. Hardware platforms are offered at prices several times lower than in the past[2]. Yet, tedious firmware development and debugging, the stochastic nature of the radio channel and unpredictable results, lack of user-friendly interfaces on IoT devices, remote testbed access, all add up to the list of challenges a researcher needs to overcome to develop and evaluate her proposal in real-world conditions.

To obtain results, a researcher typically needs to write custom evaluation-specific serial logging code in firmware, log parsing and statistical processing scripts tailored to the experiment goals. When running the experiment in a remote testbed, there is an additional step of resource reservation and testbed-specific logging. These practices are error-prone, and may result in corrupt data or published results. This is not to mention the inherent bias of evaluating one's own proposal and comparing it to state-of-the-art in favorable conditions.

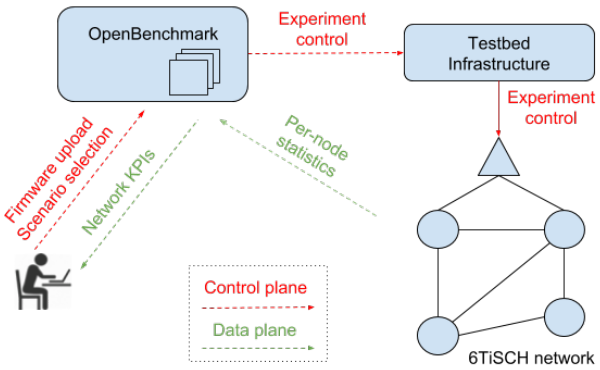

Fig. 1. Overview of `OpenBenchmark` functionality.

One consequence of this widespread practice is that we end up with a plethora of academic papers evaluating important but proposal-tailored metrics in conditions that each researcher selects on its own (fair) ground. Comparing results published by different authors is therefore challenging, at best. Reproducing the experiment of a different author typically involves extensive work, as hardware may not be the same and the change in experiment conditions may lead to unexpected results. Finally, the end users of academic research – industry stakeholders – may not always find the used metrics or conditions very relevant to their needs [4].

We present `OpenBenchmark`[3], a cloud-based benchmarking service that facilitates reproducible, repeatable and comparable IoT experimentation in industry-relevant *test scenarios*. By acting as an intermediary between the user and the testbed infrastructure, `OpenBenchmark` evaluates the networking performance of a firmware image. `OpenBenchmark` hides the testbed infrastructure complexity from the user, launches the experiment and instruments the supported firmware in real-time according to a test scenario. As shown in Fig. 1, `OpenBenchmark` collects the performance data from net-

[1] https://www.fed4fire.eu
[2] See for example http://www.ti.com/tool/cc2650stk.
[3] https://benchmark.6tis.ch

work nodes, generates an interoperable performance dataset, and processes it in real time to calculate and plot Key Performance Indicators (KPIs).

The design requirements of OpenBenchmark are:

- **Industry relevance**. Test scenarios and KPIs must be relevant to industry stakeholders.
- **Experiment reproducibility**. The user must be able to easily (e.g. with a single click) reproduce an experiment in different conditions, including on another testbed.
- **Experiment repeatability**. The user must be able to easily repeat an experiment in the exact same conditions.
- **Human and machine-user accessibility**. The user can be human, or a machine, so running an experiment can be automated.

The remainder of this paper is organized as follows. Section II details the design and implementation of the OpenBenchmark platform. Section III describes the main envisioned use cases of OpenBenchmark. Section IV discusses related work. Finally, Section V concludes this paper.

II. The OpenBenchmark Platform

OpenBenchmark automates the experimentation and network performance benchmarking on selected testbeds supporting Internet of Things devices compliant with the IEEE802.15.4 standard. OpenBenchmark instruments the execution of an experiment in real-time, following the predefined test scenarios, and collects the data to calculate the network KPIs in a fully automated manner. OpenBenchmark was designed with an IEEE802.15.4-based wireless communication technology called 6TiSCH in mind: while a majority of the KPIs are generic and applicable to other low-power IoT networking technologies, some are specific to 6TiSCH. Test scenarios are generic and derived from industrial requirements.

A test scenario is mapped to an executable logic that runs concurrently with the experiment in the testbed. OpenBenchmark sends commands to trigger the desired actions of the firmware: configure radio transmit power, trigger application packet, generate interference, ... The commands are destined to the Network Gateway, which processes and translates them into the potentially proprietary format expected by the firmware Implementation under Test (IUT). The Network Gateway may run at the testbed infrastructure and be physically connected to the serial port of IUTs, or run at OpenBenchmark premises and communicate with the IUTs over an emulated serial port. This emulated serial port is provided through the software component of the companion OpenTestbed project [2], which transports the serial data over the MQTT protocol. OpenBenchmark provides the necessary integration and provisioning of the OpenTestbed software on supported testbeds, such that this complexity is hidden from the user. This allows the user to focus on the protocol aspects of the firmware, while the performance evaluation is entirely handled by OpenBenchmark through the Application Programming Interfaces (APIs) exposed by compliant firmware projects.

TABLE I
Scenario "Building Automation": logical roles in the network.

Logical role	Occurrence	Description
Monitoring Sensor (MS)	3 / area	Sensor monitoring a physical value: temperature, flow, light
Event Sensor (ES)	4 / area	Asynchronous event detection sensors: smoke, window/door reed switches
Actuator (A)	2 / area	Node performing some physical action, e.g. light dimmer
Area Controller (AC)	1 / area	Node controlling a given area, communicating with the zone controller.
Zone Controller (ZC)	1 / network	Node controlling a given zone and potentially forwarding traffic outside the local network.

A. Support

At the time of writing, OpenBenchmark supports experimentation on the IoT-LAB testbed in Saclay, France, the w-iLab.t testbed in Ghent, Belgium, and (work ongoing) the OpenTestbed in Paris, France. The OpenWSN project serves as the proof-of-concept firmware implementation, while support for other IoT operating systems and networking stacks, such as Contiki-NG, is envisioned.

B. Test Scenarios

The goal of an OpenBenchmark *test scenario* is to capture real-life use cases of a technology in order to benchmark its performance in a setting that is relevant to the end users: companies adopting the technology for their products and their customers. A test scenario also allows the experiment to be fully reproducible and the results easily and fairly comparable, desirable properties from a research point of view.

Each scenario describes the application traffic pattern and load, and the desirable coverage requirements in terms of number of IEEE802.15.4 hops. At a later stage, we plan on adding support for controllable interference generation. The description of a scenario is generic, with testbed-specific mappings.

1) Scenario: Building Automation: Building automation systems typically consist of different areas and zones. As per RFC5867 [7], an area corresponds to a physical location within a building, typically a room with different types of sensors to feed HVAC (heating, ventilation, and air conditioning) or lighting subsystems. Each area has its own area controller. A zone represents a logical partition of the system, consisting of multiple areas. A zone has its zone controller, that is fed with data originating at the area controllers.

Within a building, there may be multiple zones depending on the specifics of the use case: multi-tenants vs single-tenant, separation by floors, etc. In terms of the definition of this scenario, we consider a multi-area, single-zone building automation system. Each area encompasses devices serving multiple subsystems like HVAC, lightning or fire detection.

Table I lists different logical roles a node in the network can have, and their occurrence.

TABLE II
SCENARIO "BUILDING AUTOMATION": TRAFFIC PATTERN.

Sender	Dest.	Traffic pattern and load	Ack	Action
MS	AC	Periodic, uniformly in [25, 35] seconds	Yes	None
ES	AC	Poisson, mean of 10 packet-s/hour	Yes	Forward to ZC
A	AC	Periodic, uniformly in [25, 35] seconds	Yes	None
AC	ZC	Periodic, uniformly in [120,140] milliseconds	No	None

TABLE III
SCENARIO "BUILDING AUTOMATION": OTHER RELEVANT SETTINGS.

Setting	Value
Coverage Requirement	4-6 hops
Application Payload Size	100 bytes

During the mapping of this scenario to testbeds, devices within a logical area should be placed in close physical proximity of each other and the area controller. Table II summarizes the traffic pattern for each logical role; Table III lists other relevant settings. The coverage requirement is an approximation based on RFC5867 deployment requirements.

2) Scenario: Home Automation: The scenario has been derived from the requirements discussed in RFC5826 [8] and the emulated topology of a smart house discussed in Vučinić *et al* [9]. Tables IV-VI list different logical roles a node in the network can have, the traffic pattern for each logical role, and other relevant settings, respectively.

3) Scenario: Industrial Monitoring: The scenario has been derived from the requirements discussed in RFC5673 [10]. Tables VII-IX list different logical roles a node in the network can have, the traffic pattern for each logical role, and other relevant settings, respectively.

TABLE IV
SCENARIO "HOME AUTOMATION": LOGICAL ROLES IN THE NETWORK.

Logical role	Occurence	Description
Monitoring Sensor (MS)	49%	Sensor monitoring a physical value, e.g. temperature, humidity
Event Sensor (ES)	21%	Asynchronous event detection sensors, e.g. human presence
Actuator (A)	30%	Node performing some physical action, e.g. light dimmer, relay
Control Unit (CU)	1/network	Central unit controlling the automation system

TABLE V
SCENARIO "HOME AUTOMATION": TRAFFIC PATTERN.

Sender	Dest.	Traffic pattern and load	Ack	Action
MS	CU	Periodic, uniformly in [3, 5] minutes	No	None
ES	CU	Poisson, mean of 10 packet-s/hour	Yes	Trigger burst
A	CU	Periodic, uniformly in [3, 5] minutes	Yes	None
CU	multiple A	Poisson, mean of 10 5-packet bursts/hour	Yes	None

TABLE VI
SCENARIO "HOME AUTOMATION": OTHER RELEVANT SETTINGS.

Setting	Value
Coverage Requirement	2-4 hops
Application Payload Size	10 bytes

TABLE VII
SCENARIO "INDUSTRIAL MONITORING": LOGICAL ROLES IN THE NETWORK.

Logical role	Occurence	Description
Sensor (S)	90%	Traditional monitoring sensor: temperature, pressure, fluid flow, ...
Bursty Sensor (BS)	10%	Monitoring sensor transmitting large quantities of data: e.g. vibration monitor
Gateway (G)	1/network	Application gateway

C. Key Performance Indicators (KPIs)

`OpenBenchmark` calculates high-level KPIs based on the events generated by the SUT[4]. For example, `OpenBenchmark` triggers the sending of an application packet by sending a `Send Packet` command to the Network Gateway, including a unique packet token, the identifier of the sender and the recipient of the packet. The Network Gateway then communicates with the firmware IUT using a proprietary format transported over the (emulated) serial port. The IUT handles this command by sending an actual packet over the network, including in its payload a unique token generated by `OpenBenchmark`, and creates a log entry over the serial port that is handled by the Network Gateway. This log entry contains the originator of the packet, the intended recipient, the unique token, the timestamp and other relevant information. Similarly, upon reception of the packet over the network, the recipient logs the time instant, the packet token, and other useful information. Based on the uniqueness of the token, `OpenBenchmark` processes the logs, matches the corresponding send and receive events, and calculates relevant KPIs. In the following, we give a brief summary of implemented KPIs.

Reliability. Refers to the ratio between packets received and packets sent by the application. Therefore, this KPI refers

[4] The complete specification of the `OpenBenchmark` APIs for experiment instrumentation and performance event handling is available at https://benchmark.6tis.ch/docs.

TABLE VIII
SCENARIO "INDUSTRIAL MONITORING": TRAFFIC PATTERN.

Sender	Dest.	Traffic pattern and load	Ack	Action
S	G	Periodic, uniformly in [1, 60] seconds	No	None
BS	G	Periodic, uniformly in [1, 60] minutes	No	None

TABLE IX
SCENARIO "INDUSTRIAL MONITORING": OTHER RELEVANT SETTINGS.

Setting	Value
Coverage Requirement	5-10 hops
Bursty Sensor Payload Size	100 bytes
Packets per Burst	10 packets

to the end-to-end reliability. A packet may fail a transmission on a given link and be later re-transmitted. However, a failed packet transmission on a given link does not influence the end-to-end reliability if the packet eventually arrives at the destination.

Latency. Refers to the time interval between the instant packet is generated at the application layer of the sender, and the instant the packet is received by the application layer of the destination.

Radio Duty Cycle (RDC). Refers to the ratio between the cumulative time that the radio chip is powered, and the measurement period. The firmware IUT needs to locally keep track of the duty cycle and generate a corresponding event that is processed by `OpenBenchmark`.

Number of Hops Traversed per Packet. A research-relevant KPI that refers to the number of nodes in the network that have forwarded a given packet before it reaches its final destination. When sending and receiving an application packet triggered by `OpenBenchmark`, the IUT logs the value of the time-to-live field in the generated/received IPv6 header.

Synchronization Precision. A research-relevant KPI that refers to the average clock drift measured between a pair of nodes. In a 6TiSCH network, nodes indicate clock de-synchronization as a field in the link-layer acknowledgment frame. The IUT needs to log the measured clock drift and the identifier of the peer.

Network Formation Time. Refers to the initial phase when the network is forming. It is an important KPI from the installation point of view. The IUT needs to log the timestamp of the corresponding events. We consider 3 different sub-phases described in the following.

Synchronization Phase Duration. A 6TiSCH-specific KPI that refers to the time interval between the instant a device is booted, and the instant that device gets synchronized with the network and starts duty cycling. In 6TiSCH networks, a device is synchronized upon reception of a first Enhanced Beacon (EB) frame.

Secure Join Phase Duration. Refers to the time interval between the instant a device gets synchronized with the network, and the instant corresponding to the end of the authentication, key and parameter distribution protocol. Once a device completes the secure join phase, it becomes a network node.

Parent Selection and Bandwidth Assignment Phase Duration. A 6TiSCH-specific KPI that refers to the time interval between the instant corresponding to the end of the authentication, key and parameter distribution protocol, and the instant the node has been successfully assigned the minimum bandwidth needed for it to start sending application traffic.

For a node to complete this phase, it first needs to select a routing parent and then request bandwidth. Once the bandwidth with the default (preferred) parent is assigned, the node can start sending application traffic.

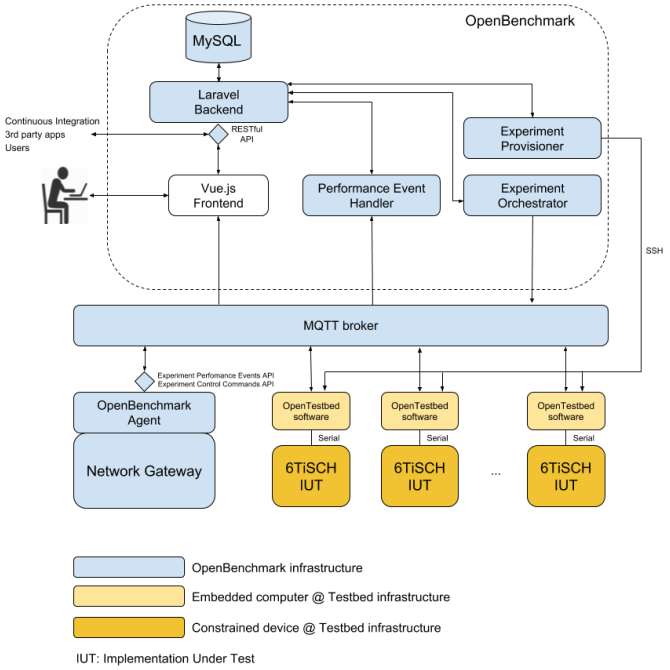

Fig. 2. Software architecture of the OpenBenchmark platform. The System under Test (SUT) consists of the Network Gateway and firmware Implementations under Test (IUTs).

D. Software Architecture

The `OpenBenchmark` platform consists of the following components:

- Agent. A component running at the Network Gateway side, translating `OpenBenchmark` commands to the format that the IUT implements, and also converting performance data from the IUT to the format expected by `OpenBenchmark`.
- Experiment Provisioner. A component in charge of testbed node reservation, firmware flashing, and launching the necessary software components that run at testbed infrastructure side. These include the Network Gateway, and the serial port emulation software (OpenTestbed) that make the testbed nodes appear to the Network Gateway as if they were physically connected.
- Experiment Orchestrator. A component in charge of orchestrating the SUT according to the selected test scenario. The Experiment Orchestrator interprets the test scenario files and instruments the experiment based on the interpreted data.
- Performance Event Handler. A component in charge of handling performance data events coming from the SUT. Based on these events, Performance Event Handler generates the experiment data sets and calculates the KPIs.
- Web server. A Laravel-based (PHP) backend and Vue.js-based frontend allowing the user to access the `OpenBenchmark` platform through a graphical interface. The backend serves as a bridge between the frontend

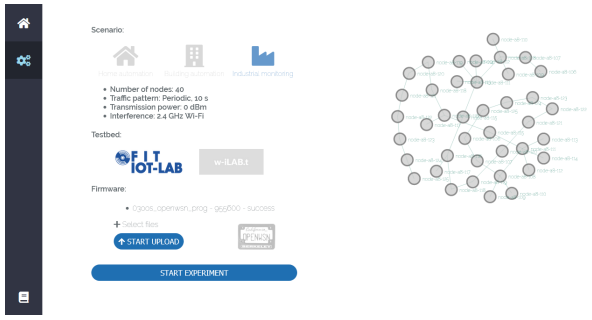

Fig. 3. Screenshot of the OpenBenchmark GUI section for scenario and testbed selection and firmware upload.

Fig. 4. Screenshot of the real-time KPI monitoring as the experiment progresses. The figure on the bottom depicts the increase in assigned bandwidth once the node has joined the network. The figure on the top shows how duty cycle varies depending on the phase.

and the rest of the OpenBenchmark components that are implemented in Python. The backend provides a RESTful API that enables the use of OpenBenchmark by 3^{rd} party applications.

E. User Interface

Fig. 3 presents a screenshot of the graphical user interface (GUI) that allows the user to upload the firmware, select a test scenario and the testbed. The GUI allows KPI monitoring in real-time (see Fig. 4) as the experiment is progressing, as well as the experiment report dashboard. Each report summarizes the main results of the experiment through the KPI plots. The dashboard further enables the user to reproduce the experiment at a later time or compare the results of different experiments.

Multiple users can access the platform concurrently, run the experiments, generate experiment reports and store experiment history. The same scenario can be concurrently triggered on different testbeds by different users, but due to the overlapping physical testbed resources in use among different scenarios, one testbed can only support one OpenBenchmark experiment at a time. OpenBenchmark functionalities can also be accessed through the RESTful API.

F. Scenario Implementation

A test scenario is defined in a JSON config file. The JSON file consists of a generic part describing the scenario "instance", and a testbed-specific part describing how the instance is mapped to a specific testbed through physical nodes to use and their transmission power. Application traffic is encoded as an array of objects carrying the time instants relative to the beginning of the experiment when a node is instructed to send an application packet. These time instants follow the distributions discussed in Section II-B. Listing 1 depicts an JSON snippet describing a building automation test scenario instance and its mapping to the IoT-lab testbed.

Listing 1. Example JSON snippet describing a "test scenario".

```
{
 "identifier": "building-automation",
 "duration_min": 300,
 "number_of_nodes": 47,
 "nodes": [
  { "generic_id": "openbenchmark00",
    "role": "monitoring-sensor",
    "area_id": 1,
    "traffic_points": [
     {"time_instant_s": 2596, "destination": "
         openbenchmark01"},
     {"time_instant_s": 3784, "destination": "
         openbenchmark02"},
     {"time_instant_s": 5190, "destination": "
         openbenchmark03"}, ... ]    },    ],
 "iotlab": { "openbenchmark00": {
             "node_id": "a8-100",
             "transmission_power_dbm": 0 },
                 ...   }    }
```

III. Example Use Cases

We envision three main use cases of OpenBenchmark, with target different groups: IoT industry stakeholders, research community, and firmware developers.

A. Referent Benchmark of an IoT Technology

Although there are many variants of IoT communication stacks (e.g. 6TiSCH, WirelessHART, ZigBee, ZigBee IP, Thread), it is quite challenging to point to a document that gives a fair and industry-relevant performance comparison among them. We designed OpenBenchmark to be used to tackle this challenge. We are coordinating with the IETF 6TiSCH standardization group to use OpenBenchmark to produce a *reference* performance dataset of the 6TiSCH technology, as standardized in the IETF [11]. The standardization group plans to use the dataset to promote the technology to the industry stakeholders, and also to identify the performance bottlenecks that need to be addressed in the next generation of standards.

B. Research Proposal Benchmarking

The research community also benefits from OpenBenchmark. We hope to attract researchers to use our benchmarking service for the evaluations of their research proposals. OpenBenchmark facilitates the extraction of experiment data by hiding the unnecessary testbed complexity. Moreover, it also leads to the increased confidence in the results: OpenBenchmark is in its entirety open source and can be reviewed and improved by the community. By sharing the pointers to OpenBenchmark experiment reports (e.g. in a scientific paper), a researcher allows her colleagues to easily reproduce the experiment through the preserved firmware image, compare results with her own, all in a fully automated

manner. This is to be compared with a tedious process we know today of requesting the source code, exchanging about the steps taken (potentially a couple of years ago), debugging the hardware that executes someone else's code, ...

C. Continuous Delivery Benchmarking

Firmware always evolves. Updates to the standards, newly discovered security vulnerabilities in the code, new features, all require the firmware development community to constantly update the code base of different IoT open-source projects. The best practices of continuous integration testing are already in place for the popular repositories. However, unit and functional testing do not indicate whether a software patch introduces unwanted performance loopholes. *Does the proposed patch improve or degrade existing performance? In what conditions was the "existing performance" measured couple of years ago when we first merged that feature?* To answer such questions, `OpenBenchmark` is designed to provide a "continuous delivery benchmarking" service to firmware developers. We are working on integrating `OpenBenchmark` with the continuous integration procedures of the OpenWSN firmware project, the referent implementation of the 6TiSCH protocol stack. This allows the code maintainers to run automated nightly experiments and assess the performance of the latest patches, before their release.

IV. Related Work

The need for a standardized benchmark in low-power networking community was first proposed in 2015 by Duquennoy *et al.* [3]. Ever since, colleagues from 18 different institutions have gathered to launch an "IoT Benchmarks" initiative. As stated on the initiative website[5], the goal is to *raise the bar in the quality of experimental data, and provide researchers and engineers in both academia and industry with an objective view of the strengths and weaknesses among existing protocols*. There are many activities going on in parallel, the organization of scientific workshops (CPSBench) and competitions (EWSN) being some of the prominent examples. Researchers are working on related topics such as controllable interference generation [12], definition of a generic benchmark for low-power wireless networking [13], warning against (un)realistic properties of testbeds [14]. These efforts are perfectly complementary to each other and to our effort on the `OpenBenchmark` platform. We plan on integrating the controllable interference generation tools and we already exploit the methodology to collect dense connectivity datasets in order to provide a mapping of the generic test scenarios to physical devices in the testbed.

V. Conclusion

This paper presents `OpenBenchmark`, a benchmarking service for the IoT networking community that facilitates repeatable and reproducible experimentation on testbeds. `OpenBenchmark` expects the user to upload a firmware image and select a test scenario to obtain the network KPIs. The platform takes care of testbed reservation, instruments the firmware in real time according to the scenario, collects the performance data from supported firmware projects and allows the researcher to focus on the protocol development. `OpenBenchmark` is developed open source[6] in order to maximize the confidence of the community in the provided results through code reviews and improvements.

[5] https://www.iotbench.ethz.ch/

[6] https://github.com/openwsn-berkeley/openbenchmark

References

[1] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 459–464.

[2] J. Muñoz, F. Rincon, T. Chang, X. Vilajosana, B. Vermeulen, T. Walcarius, W. Van de Meerssche, and T. Watteyne, "OpenTestBed: Poor Man's IoT Testbed," in *IEEE INFOCOM, CNERT workshop (under review)*. IEEE, 2019.

[3] S. Duquennoy, O. Landsiedel, C. A. Boano, M. Zimmerling, J. Beutel, M. C. Chan, O. Gnawali, M. Mohammad, L. Mottola, L. Thiele *et al.*, "A benchmark for low-power wireless networking," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*. ACM, 2016, pp. 332–333.

[4] B. Martinez, C. Cano, and X. Vilajosana, "A Square Peg in a Round Hole: The Complex Path for Wireless in the Manufacturing Industry," *arXiv preprint arXiv:1808.03065*, 2018.

[5] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. IEEE, 2004, pp. 455–462.

[6] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "OpenWSN: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[7] J. Martocci, P. De Mil, and N. Riou, *Building Automation Routing Requirements in Low-Power and Lossy Networks*, Internet Engineering Task Force Std. RFC5867, June 2010.

[8] A. Brandt, J. Buron, and G. Porcu, *Home Automation Routing Requirements in Low-Power and Lossy Networks*, Internet Engineering Task Force Std. RFC5826, April 2010.

[9] M. Vučinić, B. Tourancheau, and A. Duda, "Performance comparison of the RPL and LOADng routing protocols in a home automation scenario," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*. IEEE, 2013, pp. 1974–1979.

[10] K. Pister, P. Thubert, S. Dwars, and T. Phinney, *Industrial Routing Requirements in Low-Power and Lossy Networks*, Internet Engineering Task Force Std. RFC5673, October 2009.

[11] M. Vučinić, M. Pejanović-Djurišić, and T. Watteyne, "SODA: 6TiSCH Open Data Action," in *2018 IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench)*. IEEE, 2018, pp. 42–46.

[12] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. Zúñiga, "Jamlab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *International Conference on Information Processing in Sensor Networks (IPSN), April 2011, Chicago, IL, USA*, 2011.

[13] C. A. Boano, S. Duquennoy, A. Förster, O. Gnawali, R. Jacob, H.-S. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G. P. Picco *et al.*, "IoT-Bench: Towards a Benchmark for Low-power Wireless Networking," in *2018 IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench)*, 2018.

[14] K. Brun-Laguna, P. H. Gomes, T. Watteyne, and P. Minet, "Moving Beyond Testbeds? Lessons (We) Learned about Connectivity," *IEEE Pervasive Computing*, 2018.