Continuously Testing Distributed IoT Systems: An Overview of the State of the Art

Jossekin Beilharz[1], Philipp Wiesner[2], Arne Boockmeyer[1], Lukas Pirl[1], Dirk Friedenberger[1], Florian Brokhausen[2], Ilja Behnke[2], Andreas Polze[1], and Lauritz Thamsen[3,2]

[1] Hasso Plattner Institute, University of Potsdam, Germany
{firstname.lastname}@hpi.de
dirk.friedenberger@guest.hpi.de
[2] Technische Universität Berlin, Germany
{wiesner, florian.brokhausen, i.behnke}@tu-berlin.de
[3] Humboldt-Universität zu Berlin, Germany
lauritz.thamsen@hu-berlin.de

Abstract. The continuous testing of small changes to systems has proven to be useful and is widely adopted in the development of software systems. For this, software is tested in environments that are as close as possible to the production environments. When testing IoT systems, this approach is met with unique challenges that stem from the typically large scale of the deployments, heterogeneity of nodes, challenging network characteristics, and tight integration with the environment among others. IoT test environments present a possible solution to these challenges by emulating the nodes, networks, and possibly domain environments in which IoT applications can be executed. This paper gives an overview of the state of the art in IoT testing. We derive desirable characteristics of IoT test environments, compare 18 tools that can be used in this respect, and give a research outlook of future trends in this area.

Keywords: Internet of Things · Cyber-Physical Systems · Fog Computing · Edge Computing · Testing · Iterative Software Development.

1 Introduction

The Internet of Things (IoT) has the potential to transform our lives by connecting everyday objects to the Internet for smarter cities, factories, houses, and more. To realize this vision, distributed software systems will need to integrate IoT devices – usually equipped with sensors and actuators – allowing them to continuously monitor and interact with their environments. These distributed software systems of the IoT will span from devices to clouds and, in many cases, also include intermediate resources at the edge or fog level [7]. Examples of distributed IoT systems include those that control and manage traffic and transportation [19,36], those that enable telemedicine and remote patient monitoring [18,10], and those that detect and predict failures as well as optimize processes in urban infrastructures and manufacturing [13,22,9].

A major remaining challenge to practically developing and deploying distributed IoT systems is the difficulty of adequately testing them [15]. This is complicated due to a number of factors, including the large number of devices, the heterogeneity of devices, mobile nodes resulting in dynamic topologies, network disconnections and node failures, as well as a tight integration of systems with their respective environments. At the same time, properly testing IoT systems in application domains such as traffic and transportation management, patient monitoring, and factory processes is absolutely critical. Consequently, the need for adequate testing of distributed IoT systems has been widely recognized and many solutions have been put forward. Prominent examples include hardware testbeds like StarBED [21] and FIT-IoT [1], hybrid approaches such as Chameleon [14], as well as simulators like IoTSim [34] and iFogSim [11].

Hardware testbeds allow to execute actual application code in realistic settings, yet can be limited in terms of scalability and flexibility. Hybrid test environments address these limitations by incorporating both actual hardware and virtual nodes. Simulations on the other hand enable to flexibly assess the behavior of distributed applications over various scales and possible infrastructures. However, they usually lack the ability to evaluate the non-functional properties of actual application code.

All these approaches have in common that it is typically hard to test distributed IoT systems within their actual environment. Field testing regularly requires a large and coordinated effort, so distributed IoT systems cannot be tested continuously, while lab testing routinely resorts to merely replaying sensor data, so that the distributed IoT systems, despite being equipped to interact with environments, cannot actually influence their domains. This runs contrary to generally understood and widely adopted principles and best practices of iterative software development, where continuous testing of small changes to systems in environments that mirror production environments as closely as possible is a key mechanism for fast feedback and trust in changes. We, therefore, argue that there is a significant lack of approaches and tools for continuously testing IoT systems.

In this paper, we compare currently available IoT test environments to provide an overview over the current state of the art and expose the perceived research gap. For our comparison, we selected test environments that

1. focus on testing software systems on geo-distributed, heterogeneous computing infrastructures such as IoT and edge/fog architectures,
2. allow to run actual system code (i.e., not merely simulating communication),
3. and have the ability to include virtual nodes, allowing tests at large and various scales (i.e., no hardware-only testbeds).

We only discuss general-purpose test environments of which details have been published (i.e., no proprietary offers such as IoTIFY[4] or AWS IoT Device Simulator[5]).

[4] `https://iotify.io/`

[5] `https://aws.amazon.com/solutions/implementations/iot-device-simulator/`

We report the following aspects of IoT test environments in our comparison: how and with which capabilities the tools provide nodes, how the network between nodes is realized, whether domain environments are integrated, as well as general aspects such as project maturity and ongoing development.

The main contributions of this paper are:

- A description of key characteristics of IoT test environments, which can be used to distinguish proposed solutions.
- A point-by-point comparison of state-of-the-art IoT test environments that meet the outlined selection criteria.
- A discussion of current trends and considerable gaps in the state of the art of IoT testing.
- An outlook on future work to close these gaps and an overview of our work in this area.

The remainder of this paper is structured as follows: Section 2 describes central characteristics of test environments. These are used in Section 3 to evaluate and compare concrete test environments. Section 4 discusses the results of our comparison. Section 5 presents the research outlook. Section 6 covers related work. Lastly, Section 7 concludes this paper.

2 Characteristics of Test Environments

Continuously testing IoT systems and applications requires a test environment that reproduces reality as close as possible. To classify and compare existing test environments we derive several quantifiable characteristics from generally desirable properties of test environments.

To be able to continuously develop and test distributed IoT systems in iterative software development processes, we need to be able to deploy and run actual code in flexible, yet realistic environments. To facilitate large-scale deployments while also allowing the realistic testing of system behavior, we believe the support for both virtualized nodes as well as hardware nodes in test environments is crucial. The testing of large-scale deployments further requires the distributability of not only the nodes, but also the network representation and the simulation of the domain environment. Another important aspect relating to the three feature dimensions here — nodes, networks, and the domain environments — is the meaningful testing of fault tolerance of IoT systems by precisely injecting faults. Lastly, because IoT systems are inherently integrated tightly with their specific environment through sensors and actuators, we believe that the simulation of the domain environment is a key characteristic for test environments.

The remainder of this section discusses the characteristics that will be used in the following comparison of test environments in Section 3. We identified 16 different attributes, which are organized into four overarching categories, regarding general features, the nodes, the network and the domain environment.

2.1 General

First, we describe general attributes of test environments. We present the *initial* year of *publication* along with the information if the project is *actively maintained*, which is assessed based on whether there has been a new release or active collaboration (e.g., commits to the repository) in 2021.

As the *maturity* of a project is subjective, we try to formalize it as follows. An empty circle (○) denotes the lowest maturity, meaning that the specific test environment only exists as a concept in form of a publication, but no actual tool is available. We did not assess whether such concepts are actively maintained, as this cannot be sensibly judged. The second degree of maturity (◑) is reported if the tool is available only as a prototype without good documentation. If there is a full system available with detailed documentation, we denote it as the third degree of maturity (●).

Next, we classify if a test environment is *offered as a service*. This indicates whether there is a service where the test environment can be used without manually deploying and operating it.

Lastly, we asses the property *scriptable scenarios*, which is fulfilled if the execution of experiments can be controlled via a script. With the capability to predefine schemes to alter parameters and characteristics of a simulation at runtime, much more complex scenarios can be implemented. This is highly important when systematically approaching an investigation with a test environment.

These general information about a test environment can serve as an indicator for the applicability to current challenges, but they are also used to identify recent trends in test environments in Section 4.

2.2 Nodes

An essential aspect of test environments is which type of nodes can be used. The attributes investigated here determine if a scenario or application of interest can at all be properly implemented or analyzed with a given test environment.

The first attribute, *hardware integration*, classifies the test environments according to their capability to integrate physical hardware nodes. The availability of hardware integration enables the inclusion of embedded systems and facilitates testing of applications in realistic environments.

The *virtualization type* describes how virtual nodes are represented, namely via virtual machines (V), containerized nodes (C), or a combination of the two (VC). Depending on the application under test, the differentiation between containerized and virtualized nodes can be crucial. Virtual machines enable a more realistic execution environment for the application under test, while containerization is a more light-weight approach.

For the *energy consumption* characteristic, we investigate if a test environment facilitates modeling (or, in the case of hardware nodes, monitoring) the power consumption of nodes and network. In any use case where energy is a scarce resource, for example for battery-constrained IoT devices, this feature allows testing the effect of software changes to a node's power usage.

The *distributability* describes whether virtual nodes of the test environment can be spread across multiple physical host nodes, enabling large-scale scenarios.

Finally, we investigate the possibility of *fault injection*. For individual nodes, examples include the purposeful shutdown or internal failure of a given node at a given time. By simulating such faults, the robustness of an application or network setup towards faults can be tested. We analyze the characteristics of distributability and fault injection as well for the network and the domain environment categories.

2.3 Network

Regarding network, we first analyze the *network type*, namely how network is emulated within the test environment. Traffic shaping (TS) allows users to change network parameters, like delay or bandwidth. Examples of this are the Linux Traffic Control (*tc*) or the more advanced NetEm. Tools based on software-defined networks (SDN) use a virtualized network such as provided by Mininet or MaxiNet. Lastly, network simulators (NS) can be used to model the underlying network. In our understanding, network simulators can simulate different kinds of networks, also future ones, without having them physically available. Common network simulators are ns-3 or OMNet++ with INET.

Network *distributability* regards the possibility of the test environment to span the network across multiple physical hosts, leveraging more complex routing schemes in a physical network. For traffic shaping-based approaches this comes naturally if nodes are distributed, for network simulation-based approaches also the simulation has to run in a distributed manner.

Fault injection entails active support of the tool to purposefully alter network connections at runtime, e.g., the increase of latency or the loss of packets. Such capabilities are important when comparing fault tolerance of network setups and in general testing of network robustness.

2.4 Domain Environment

As IoT applications run in embedded, real-life settings like traffic control, water management or smart homes, simulating the domain environment is of high importance. First, we asses the general *domain environment support* of a tool, meaning whether there is an API for connecting domain-specific simulators that can interact with the test environment at runtime. Examples include a traffic simulation, such as SUMO, that can send the coordinates of mobile nodes to the test environment for it to adapt its networking parameterizations.

Similar to the node and network categories, we also asses the *distributability* of the domain environment to see if the execution of the environment can be spread across multiple hosts.

Last, we report the capability of *fault injection* inside the domain environment. We define this functionality to be present when the test environment supports to alter the domain environment during runtime in a way that is expected to introduce faults in the application running on the nodes.

Table 1. An overview of test environments for IoT systems. A gray background marks works where code is not openly available. All characteristics are described in Section 2.

	General					Nodes					Network			Domain Env.		
	Initial Publication	Actively Maintained	Maturity	Offered as a Service	Scriptable Scenarios	Hardware Integration	Virtualization Type	Energy Consumption	Distributability	Fault Injection	Network Type	Distributability	Fault Injection	Domain Env. Support	Distributability	Fault Injection
EMU-IoT [27]	2019	○	◑	○	●	○	C	○	●	○	○	-	-	○	-	-
ELIoT [23]	2017	○	◑	○	○	○	C	○	●	○	○	-	-	○	-	-
IOTier [24]	2021	-	○	○	●	○	C	○	●	●	TS	●	●	○	-	-
Fogify [30]	2020	●	●	○	●	○	C	●	●	●	TS	●	●	○	-	-
MockFog [12]	2019	●	◑	○	●	○	V	○	●	●	TS	●	●	○	-	-
Blockade [33]	2014	●	●	○	○	○	C	○	○	●	TS	○	●	○	-	-
Distem [28]	2013	○	●	○	●	○	C	○	●	●	TS	●	●	○	-	-
Fogbed [6]	2018	○	◑	○	○	○	C	○	●	○	SDN	●	○	○	-	-
EmuFog [20]	2017	○	◑	○	○	○	C	○	●	●	SDN	●	●	○	-	-
Dockemu [26]	2015	○	◑	○	●	○	C	○	●	○	NS	○	○	○	-	-
EmuEdge [35]	2019	○	◑	○	●	●	VC	○	○	●	TS	○	●	○	-	-
Héctor [2]	2019	○	◑	○	●	●	V	○	○	●	TS	○	●	○	-	-
Sendorek et al. [29]	2018	-	○	○	●	●	V	○	○	○	SDN	○	○	○	-	-
Chameleon [14]	2015	●	●	●	○	●	V	○	●	○	SDN	●	○	○	-	-
StarBED [21]	2002	●	●	●	●	●	V	○	●	○	SDN	●	○	○	-	-
UiTiOt [16]	2017	-	○	○	○	●	C	○	●	○	NS	●	●	○	-	-
WHYNET [38]	2006	-	○	○	●	●	V	●	●	○	NS	●	●	○	-	-
MobiNet [17]	2005	-	○	○	○	●	V	○	●	○	NS	○	○	○	-	-

3 Comparison of Test Environments

Based on the selection criteria defined in Section 1, we selected 18 environments for testing IoT applications and evaluated them on the characteristics described in Section 2. Table 1 provides an overview of all evaluated tools. We clustered the test environments (1) by their ability to integrate real IoT devices in their experiments and (2) by the type of network modeling approach they use.

3.1 Test Environments without Hardware Integration

First, we cover test environments that emulate IoT environments without the possibility to integrate real IoT devices in experiments. These test environments are further categorized based on whether they use network simulation, SDN-based solutions, or simple traffic shaping to emulate realistic network traffic.

Using no Network Model. EMU-IoT [27] is a container-based test environment with a focus on defining, orchestrating and monitoring reproducible experiments. Although the authors describe the many challenges faced by developing IoT test environments, their implementation does not consider any kind of network emulation and has no mechanism for injecting faults into the system.

ELIoT [23] is based on Docker containers and supports the IoT protocols CoAP and LWM2M by using the open-source projects Leshan and coap-node. While ELIoT includes the interaction with the environment for the use case described in the paper, this interaction is only modeled within the nodes (i.e., they implemented a simple calculation of an illuminance sensor value based on the time of day). It does not integrate an environment emulation that would allow for two-way interaction between IoT systems and the environment.

Using Traffic Shaping. IOTier [24] is a virtual testbed for tiered IoT environments that is unfortunately not openly available. Nodes are represented via resource-constrained containers while networking is based on NetEm. A special focus is grouping emulated components into tiers with comparable capabilities, and enabling inter-tier as well as intra-tier communication. It features a testbed controller in which operators can define desired runtime states over time. However, there is no API for integrating simulators of domain environments. Its simulation engine uses fixed-increment time progression and can modify experiments via scheduled and conditional events.

Fogify [30] appears to be one of the most capable tools according to our criteria. It uses Infrastructure-as-Code descriptions for containerized deployments to define experiment settings (i.e., Docker Compose) and features the possibility to adapt configurations at runtime (e.g., for injecting faults). Fogify uses Virtual eXtensible LAN (VXLAN) for overlay networks and is distributable across multiple physical hosts. We classified this tool as being able to model energy consumption as this feature is described in the paper. However, this is currently not implemented in code. Although Fogify has an API for interacting with experiments during runtime, there is not yet a uniform way to integrate simulations of domain environments.

MockFog [12] is a tool for automated execution of fog application experiments. It consists of three modules: one for infrastructure setup, one for application management, and one for experiment orchestration, which enables the scripting of scenarios. The experiment infrastructures are set up automatically in public cloud environments via dockerized application containers. Hence, applications must support running inside Docker and must be available as container image. Experiment descriptions can be used to generate events, such as traffic scenarios and network or machine failures.

Blockade [33] is a test environment based on Docker containers and traffic shaping. The user creates a setup similar to a Docker Compose file and Blockade manages the set-up as well as tear-down processes. Each node is implemented as a separate Docker container. Blockade offers basic networking capabilities by

using the Docker network and integrates the manipulation of network parameters via, e.g., NetEm settings.

Distem [28] is a virtual testbed using Linux Containers (LXC) that can be executed on multiple physical hosts. One focus of Distem is resource allocation and assignment to achieve realistic setups for special devices (like IoT devices). Network parameters can be adapted using NetEm. Distem can be used via the command line and allows scriptable scenarios via its Ruby library.

Using Software-Defined Networking. Fogbed [6], as described by the original paper, uses Mininet for networking and is hence bound to a single host. The latest prototype additionally extends MaxiNet, which enables emulating environments that span several physical machines. Fogbed furthermore enables the testing of third-party systems such as resource management, virtualization, and service orchestration through standard interfaces.

EmuFog [20] is a fog computing emulation framework based on the distributable network emulator MaxiNet. The framework does not resort to simulations but is able to span an emulated network of thousands of virtual devices over multiple physical machines. EmuFog focuses on the networking components of fog computing by embedding a network topology generator, enhancer, and node placement algorithm. Applications have to be deployed as Docker containers.

Using Network Simulation. Dockemu [26] is the only tool without hardware integration that uses network simulation. It utilizes the network simulator ns-3 to model the communication between nodes, which in turn are represented by Docker containers. The paper recognizes the importance of providing realistic conditions and environmental factors for the applications under test. The tool itself, however, is restricted to controlling properties of nodes and the network but does not include mechanisms to provide a domain environment in which the application operates.

3.2 Test Environments with Hardware Integration

Next, we describe hybrid tools that offer the possibility to integrate real IoT devices in otherwise emulated environments to make experiments more realistic.

Using Traffic Shaping. EmuEdge [35] is an openly available, hybrid simulator that can represent nodes using containers, virtual machines, and physical devices. It supports OS-level as well as system-level virtualization and can interface simulators and real testbeds. Networking is based on networking namespaces (*netns*) and can replay real-world network traces.

Héctor [2] is an IoT testing framework with the main goal of representing devices realistically. Devices are emulated with QEMU in system mode, allowing fine grained performance moderation of individual devices and testing on the target platform, including its corresponding microarchitecture. Specifically, this

allows testing of devices that are not able to run Docker containers (e.g., microcontrollers). Physical as well as emulated devices can be part of the network, which itself can have emulated properties such as added delay and packet loss.

Using Software-Defined Networking. Sendorek et al. [29] describe an elaborated concept for a software-defined virtual test environment for IoT systems. Their system supports three so called "immersion levels" that range from fully virtualized environments for low-cost, scalable experiments to environments with real devices and sensors for testing under realistic conditions. The authors do not cover distributability or fault injection in their concept.

Chameleon [14] builds upon OpenStack to deliver a testbed that can be used like a cloud. Chameleon is both a concept with an open-source implementation and a platform service supported by hardware at University of Chicago and at the Texas Advanced Computing Center that includes different nodes and setups including GPUs, FPGAs as well as ARM and x86 cores. In addition to bare metal nodes, nodes virtualized with KVM can be used. Besides the concept of an OpenStack-based testbed, the Chameleon project has some insights regarding the operational side of such a testbed, like user management, fair resource allocation with leases and lease reapers, security attacks etc.

StarBED [21] is a large-scale general purpose network testbed based on colocated physical nodes which uses SpringOS to build experiment topologies and drives experiments. Its updated fourth version implements additional features, such as wireless network emulation and a background traffic generator. Although StarBED aims to enable Internet-scale experiments, it apparently lacks the possibility to emulate IoT characteristics (e.g., resource constraints, heterogeneous network capacities) and mainly acts as a resource management system.

Using Network Simulation. UiTiOt [16], meanwhile in its third version, is a test environment for large-scale wireless IoT applications. Instances of the application under test are executed using Docker Swarm on top of an OpenStack instance. The network connections between the application instance (e.g., IEEE 802.11a/b/g, ZigBee) are emulated using the wireless emulator QOMET. Apart from the virtual resources, UiTiOt can integrate physical nodes into the network. The authors also introduce a web interface for users of the testbed and a load-balanced database for receiving and storing logs from the application under test.

WHYNET [38] is a hybrid testbed that focuses on mobile communication and applications, using a combination of simulation, emulation, as well as physical nodes and connections. It simulates the network via the QualNet simulator and the sensor network simulation framework sQualNet, which is one of the few tools that model energy consumption. Using the TWINE framework [37], it emulates the network stack and the execution of applications to provide a scalable but realistic test environment. WHYNET includes a basic concept of mobility but does not allow the integration of domain-specific simulators for this purpose.

MobiNet [17] focuses on the evaluation of applications and network setup in ad hoc wireless networks. The tool allows the testing of different deployment

schemes for applications and includes the simulation of movement of nodes. The core of MobiNet takes care of emulating the physical, data link, and network layers. Edge nodes can be distributed across machines and can host multiple virtual nodes for large-scale environments. Unfortunately, the code for MobiNet is not publicly available.

4 Discussion

We identified themes that emerged in our comparison of test environments in each of our categories of characteristics: general characteristics, and those that relate to representation of nodes, network and domain environment.

4.1 General

Testing of IoT systems is an active research area and many solutions try to help the developers of IoT systems in this regard. In our comparison, most systems were initially published within the last five years. These works include mature and widely adopted projects, but also ideas and research prototypes. Only two of the examined test environments are offered as a service.

4.2 Nodes

In our comparison we investigated the ability of test environments to use virtual and hardware nodes for the testing of IoT systems. For the virtual nodes, both containers and virtual machines are used, with recent works showing a tendency to use more lightweight container virtualization. This choice of virtualization type correlates with the integration of hardware nodes: Systems that include hardware nodes mostly use virtual machines, while the others mostly use containers. The ability to execute some nodes on actual hardware is missing from more than half of the test environments, even though this is especially important in many IoT use cases because often highly customized hardware is used. While energy consumption modeling is crucial to test the behavior of battery-constrained IoT devices, this is barely considered in the tools covered. A better integration of power models, for example using simulators built for this purpose [32], would be an important next step for virtual test environments.

4.3 Network

The environments included in our comparison contain a mix of different methods to model the network. This includes two systems that do not even include the ability to specify a network topology, seven systems that support traffic shaping (usually via tc and NetEm), as well as full network simulation (four systems) and software defined networking (five systems). The scalability to large networks that need to be realized on multiple execution nodes is possible in almost all test environments that can distribute nodes. Network distributability only seems to

still be a challenge when network simulators are used. Fault injection is an important feature for IoT testing, but dedicated support for defining and executing specific failure scenarios is missing from many IoT test environments.

4.4 Domain Environment

Despite the tight integration of distributed IoT systems with their environment, support for the simulation of domain environments is missing from all testing tools included in our comparison. Accordingly, system developers have to resort to expensive and time-consuming field testing, when they want to test the interaction of IoT systems with their environment. While some environmental factors can be integrated in the testing by feeding applications recorded streams of sensor data, this integration is naturally limited and cannot model the manipulation of the environment by IoT systems.

5 Research Outlook

While many tools exist that tackle the problem of testing distributed IoT systems, there are still important open challenges.

Research Gap. Currently, there is limited support for assessing key system requirements such as high resilience and low energy consumption. However, the biggest gap in our view is the missing integration with domain environment simulations. This integration is particularly important for IoT systems, because the tight coupling and interaction with the environment is a fundamental property of the Internet of Things. The integration of domain environment simulations like traffic or infrastructure simulations would allow for meaningful and continuous testing of these interactions.

An Ideal IoT Test Environment. As we have derived the characteristics described in Section 2 from our understanding of the needs of a test environment for continuous testing, an ideal IoT test environment would fulfill all these characteristics. Specifically, an ideal test environment would:

- support testing on virtual and hardware nodes,
- model and monitor the energy consumption,
- include a network representation that allows complex network topologies and dynamic changes thereof,
- integrate domain environment simulations,
- enable the distribution of nodes, networks, and domain environments across multiple physical nodes to allow the testing of large-scale deployments,
- and also support fault injection on these three dimensions to evaluate the fault tolerance of the system under test.

The Marvis Testing Framework. We are working on a framework towards our vision of an ideal IoT test environment, called Marvis [3]. By combining virtual nodes (containers) with hardware nodes, Marvis offers capabilities for

hybrid setups to combine the advantages of scalability and realism. Nodes can communicate via a simulated network realized by the network simulator ns-3.

A focus of our work is the integration of domain environment simulators to enable the continuous testing of the often intricate interactions between the IoT system and the environment. Currently, Marvis integrates the traffic simulator SUMO to demonstrate this, allowing the testing of interactions between the real software systems that run on the nodes and the movement of road users in the traffic simulation. This integration is bidirectional, meaning both, the change of the movement of road users by the applications under test, and the change of connectivity in the network simulation by the traffic simulation is possible.

Besides this, Marvis also offers fault injection in the three feature dimensions: It is possible to inject faults in the nodes (e.g., start and stop nodes, or execute commands), the network simulation (e.g., connect or disconnect nodes, change network parameters like delay), or the domain-specific simulation (e.g., changing speed of vehicles).

6 Related Work

Testing has been recognized as an important topic in IoT systems research since its beginning. Consequently, several related works provide an overview of testing research, environments and frameworks.

Tonneau et al. [31] presented an extensive work focusing on the question of choosing the right wireless sensor network testing platform for specific environment characteristics in 2015. It is the only related work in which all presented testbeds consist of devices carrying real sensors – no platforms were presented that only simulate or emulate devices under test. Tonneau et al. considered seven platform features: experimentation, scale, repeatability, mobility, virtualization, federation, and heterogeneity.

Dias et al. [8] identified the motivation and challenges of testing planetary-scale, heterogeneous IoT applications and devices. Surveyed testing tools were chosen with no specific properties in mind, making 16 IoT testing platforms that were available in 2018 part of the survey. Tools were compared based on ten properties, including supported IoT layers, test level, test method, supported platforms, and scope (market/academic). The authors conclude that further research and development in the area of IoT testing is necessary, given the criticality of many IoT systems and the challenges of testing them.

A journal article from the same year by Chernyshev et al. [5] discusses the state of IoT research, simulators and testbeds. They defined a set of relevant research topics, including eight goals for the IoT. Furthermore, they performed a comparative study of nine simulation tools, categorized by the scope of coverage of the IoT architecture layers, as well as a comparison of three large-scale IoT hardware testbeds. They identified three open challenges concerning IoT testing: A lack of support for common IoT communication standards, a lack of end-to-end service simulation across all IoT layers, and a large discrepancy between simulator and real-world test results.

Patel et al. [25] compared a total of 26 simulators, emulators, and physical testbeds for the IoT. The authors discussed these groups of test environments independently from each other on characteristics such as scope, scale, and security measures. While there is no specific survey system or selection method given, the comparison is followed by a short analysis of the usage of simulators, emulators, and physical testbeds in the different stages of software development.

Bures et al. [4] performed a systematic mapping study on interoperability and integration testing of IoT systems. Rather than comparing specific tools, they analyzed 115 out of 803 identified papers in the general area of IoT. The literature study was guided by seven research questions regarding research trends, researchers, publication media, topics, challenges, and limitations mentioned in the surveyed works. They conclude that there is a need for more specific testing methods for IoT systems.

7 Summary

This paper presented the current state of the art in continuous testing of distributed IoT systems. Specifically, we described desirable characteristics for test environments in this context and compared IoT test environments that allow to run system code on virtual nodes. Many solutions have been put forward, implementing various approaches to providing execution hosts and realizing network conditions. However, no currently available solution provides support for domain simulations, even though IoT systems form cyber-physical systems that make sense of and interact with their surroundings.

We believe that systems that monitor and affect the real world should be tested comprehensively, especially in critical application domains such as traffic management, patient monitoring, and manufacturing. Future work should therefore focus on providing comprehensive test environments, including simulation of domains and modeling of system characteristics such as energy consumption.

References

1. Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., Watteyne, T.: FIT IoT-LAB: A large scale open experimental IoT testbed. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). IEEE (2015)
2. Behnke, I., Thamsen, L., Kao, O.: HéCtor: A framework for testing IoT applications across heterogeneous dge and cloud testbeds. In: 12th International Conference on Utility and Cloud Computing Companion. ACM (2019)
3. Beilharz, J., Wiesner, P., Boockmeyer, A., Brokhausen, F., Behnke, I., Schmid, R., Pirl, L., Thamsen, L.: Towards a staging environment for the Internet of Things. In: 2021 IEEE International Conference on Pervasive Computing and Communications (PerCom Workshops). IEEE (2021)
4. Bures, M., Klima, M., Rechtberger, V., Bellekens, X., Tachtatzis, C., Atkinson, R., Ahmed, B.S.: Interoperability and integration testing methods for IoT systems: A systematic mapping study. In: International Conference on Software Engineering and Formal Methods. Springer (2020)

5. Chernyshev, M., Baig, Z., Bello, O., Zeadally, S.: Internet of things (IoT): Research, simulators, and testbeds. IEEE Internet of Things Journal (2017)
6. Coutinho, A., Greve, F., Prazeres, C., Cardoso, J.: Fogbed: A rapid-prototyping emulation environment for fog computing. In: 2018 IEEE International Conference on Communications (ICC). IEEE (2018)
7. Dastjerdi, A.V., Buyya, R.: Fog computing: Helping the Internet of Things realize its potential. Computer (2016)
8. Dias, J.P., Couto, F., Paiva, A.C., Ferreira, H.S.: A brief overview of existing tools for testing the Internet-of-Things. In: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE (2018)
9. Geldenhuys, M.K., Will, J., Pfister, B., Haug, M., Scharmann, A., Thamsen, L.: Dependable IoT data stream processing for monitoring and control of urban infrastructures. In: IEEE International Conference on Cloud Engineering. IEEE (2021)
10. Gontarska, K., Wrazen, W., Beilharz, J., Schmid, R., Thamsen, L., Polze, A.: Predicting medical interventions from vital parameters: Towards a decision support system for remote patient monitoring. In: International Conference on Artificial Intelligence in Medicine (AIME). Springer (2021)
11. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Software: Practice and Experience (2017)
12. Hasenburg, J., Grambow, M., Bermbach, D.: Mockfog 2.0: automated execution of fog application experiments in the cloud. IEEE Transactions on Cloud Computing (2021)
13. Kang, H.S., Lee, J.Y., Choi, S., Kim, Hyun andPark, J.H., Son, J.Y., Kim, B.H., Do Noh, S.: Smart manufacturing: Past research, present findings, and future directions. International journal of precision engineering and manufacturing-green technology (2016)
14. Keahey, K., Anderson, J., Zhen, Z., Riteau, P., Ruth, P., Stanzione, D., Cevik, M., Colleran, J., Gunawi, H., Hammock, C., Mambretti, J., Barnes, A., Halbach, F., Rocha, A., Stubbs, J.: Lessons learned from the Chameleon testbed. In: 2020 USENIX Annual Technical Conference (USENIX ATC 20) (2020)
15. Kim, H., Ahmad, A., Hwang, J., Baqa, H., Le Gall, F., Ortega, M., Song, J.: IoT-TaaS: Towards a prospective iot testing framework. IEEE Access (2018)
16. Ly-Trong, N., Dang-Le-Bao, C., Huynh-Van, D., Le-Trung, Q.: UiTiOt v3: A hybrid testbed for evaluation of large-scale IoT networks. In: 9th International Symposium on Information and Communication Technology. ACM (2018)
17. Mahadevan, P., Rodriguez, A., Becker, D., Vahdat, A.: MobiNet: A scalable emulation infrastructure for ad hoc and wireless networks. ACM SIGMOBILE Mobile Computing and Communications Review (2006)
18. Malasinghe, L.P., Ramzan, N., Dahal, K.: Remote patient monitoring: A comprehensive study. Journal of Ambient Intelligence and Humanized Computing (2019)
19. Masek, P., Masek, J., Frantik, P., Fujdiak, R., Ometov, A., Hosek, J., Andreev, S., Mlynek, P., Misurec, J.: A harmonized perspective on transportation management in smart cities: The novel IoT-driven environment for road traffic modeling. Sensors (2016)
20. Mayer, R., Graser, L., Gupta, H., Saurez, E., Ramachandran, U.: EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures. In: 2017 IEEE Fog World Congress (FWC). IEEE (2017)

21. Miyachi, T., Chinen, K.i., Shinoda, Y.: StarBED and SpringOS: large-scale general purpose network testbed and supporting software. In: 1st Intternational Conference on Performance Evaluation Methodolgies and Tools. ACM (2006)
22. Mohammadi, M., Al-Fuqaha, A.: Enabling cognitive smart cities using big data and machine learning: Approaches and challenges. IEEE Communications Magazine (2018)
23. Mäkinen, A., Jiménez, J., Morabito, R.: ELIoT: Design of an emulated IoT platform. In: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC). IEEE (2017)
24. Nikolaidis, F., Marazakis, M., Bilas, A.: IOTier: A virtual testbed to evaluate systems for IoT environments. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE (2021)
25. Patel, N.D., Mehtre, B.M., Wankar, R.: Simulators, emulators, and test-beds for internet of things: A comparison. In: 2019 Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud). IEEE (2019)
26. Petersen, E., Cotto, G., To, M.A.: Dockemu 2.0: Evolution of a network emulation tool. In: 2019 IEEE 39th Central America and Panama Convention. IEEE (2019)
27. Ramprasad, B., Fokaefs, M., Mukherjee, J., Litoiu, M.: EMU-IoT - a virtual Internet of Things lab. In: 2019 IEEE International Conference on Autonomic Computing (ICAC). IEEE (2019)
28. Sarzyniec, L., Buchert, T., Jeanvoine, E., Nussbaum, L.: Design and evaluation of a virtual experimental environment for distributed systems. In: 21st Euromicro Int. Conference on Parallel, Distributed, and Network-Based Processing. IEEE (2013)
29. Sendorek, J., Szydlo, T., Brzoza-Woch, R.: Software-defined virtual testbed for IoT systems. Wireless Communications and Mobile Computing (2018)
30. Symeonides, M., Georgiou, Z., Trihinas, D., Pallis, G., Dikaiakos, M.D.: Fogify: A fog computing emulation framework. In: 2020 IEEE/ACM Symposium on Edge Computing (SEC). IEEE (2020)
31. Tonneau, A.S., Mitton, N., Vandaele, J.: How to choose an experimentation platform for wireless sensor networks? A survey on static and mobile wireless sensor network experimentation facilities. Ad Hoc Networks (2015)
32. Wiesner, P., Thamsen, L.: LEAF: Simulating large energy-aware fog computing environments. In: 2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC). IEEE (2021)
33. Worstcase: Blockade. `https://github.com/worstcase/blockade` (2021)
34. Zeng, X., Garg, S.K., Strazdins, P., Jayaraman, P.P., Georgakopoulos, D., Ranjan, R.: IOTSim: A simulator for analysing IoT applications. Journal of Systems Architecture (2017)
35. Zeng, Y., Chao, M., Stoleru, R.: EmuEdge: A hybrid emulator for reproducible and realistic edge computing experiments. In: 2019 IEEE International Conference on Fog Computing (ICFC). IEEE (2019)
36. Zhao, Y., Yu, X., Chen, M., Zhang, M., Chen, Y., Niu, X., Sha, X., Zhan, Z., Li, W.J.: Continuous monitoring of train parameters using IoT sensor and edge computing. IEEE Sensors Journal (2021)
37. Zhou, J., Ji, Z., Bagrodia, R.L.: TWINE: A hybrid emulation testbed for wireless networks and applications. In: INFOCOM. vol. 6. Citeseer (2006)
38. Zhou, J., Ji, Z., Varshney, M., Xu, Z., Yang, Y., Marina, M., Bagrodia, R.: WHYNET: A hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. In: 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization. ACM (2006)