Héctor: A Framework for Testing IoT Applications Across Heterogeneous Edge and Cloud Testbeds

Ilja Behnke
i.behnke@tu-berlin.de
Technische Universität Berlin

Lauritz Thamsen
lauritz.thamsen@tu-berlin.de
Technische Universität Berlin

Odej Kao
odej.kao@tu-berlin.de
Technische Universität Berlin

ABSTRACT

As a result of the many technical advances in microcomputers and mobile connectivity, the Internet of Things (IoT) has been on the rise in the recent decade. Due to the broad spectrum of applications, networks facilitating IoT scenarios can be of very different scale and complexity. Additionally, connected devices are uncommonly heterogeneous, including micro controllers, smartphones, fog nodes and server infrastructures. Therefore, testing IoT applications is difficult, motivating adequate tool support.

In this paper, we present Héctor, a framework for automatic testing of IoT applications. Héctor allows the automated execution of user-defined experiments on agnostic IoT testbeds. To test applications independently of the availability of required devices, the framework is able to generate virtual testbeds with adjustable network properties. Our evaluations show that simple experiments can be easily automated across a broad spectrum of testbeds. However, the results also indicate that there is considerable interference in experiments, in which many devices are emulated, due to the high resource demand of system emulation.

KEYWORDS

application testing, virtual testbeds, IoT architectures, edge computing, distributed stream processing

1 INTRODUCTION

The advances in network technologies and computing capabilities of embedded systems are leading to an increasing amount of small devices being connected to the internet in many fields. This observation, called the Internet of Things (IoT), is already widely in productive use. Examples can be found in smart city applications [4], networked and automated production environments, where production cycles of thousands of parts are monitored individually [11], and in autonomous driving research [13].

This development, however, also comes with several challenges, especially when regarding the feasibility of large-scale implementations. With the rapid growth of the IoT, an increasing amount of data is generated outside of high performing computing backbones such as server farms and clouds. Moreover, data generation happens highly fragmented and in a geo-distributed manner [15]. As many IoT appliances require low latencies, one of the main challenges is the reduction of transport and processing times. This, and the vast amount of data created by large sensor networks, led to the establishment of the edge and fog computing approaches [2, 19]. By pulling computing tasks towards the point of data creation and, thereby, relieving the central computing nodes, networks are disencumbered. Where possible, data filtering and aggregation is moved to small to medium-sized computers in the vicinity of the sensors. Implementing this, extensions for distributed processing platforms that incorporate edge resources have been proposed [10, 18].

However, the resulting complexity of the IoT make the development and testing of such systems a difficult task, where aside of the unsteadiness of many IoT networks, the number of devices also makes it difficult to properly develop and test large-scale systems. Hence, an adequate means to support changes in testbed scale—taking into account heterogeneity of devices, complex dataflows, and with the ability to support different kinds of streaming jobs—is necessary. Since IoT testbeds need to scale not only in heterogeneity but should also match the size of large sensor networks, small lab-based experiments must be expanded into virtual environments [7].

Addressing the challenges of IoT application testing, this paper makes the following contributions:

(1) An approach for a testbed-agnostic IoT testing framework, which we call Héctor
(2) An implementation of a prototype of our approach using a Python API and edge device emulation
(3) An empirical evaluation of the prototypical implementation of our testing framework and emulation approach

The following section will cover some of the technical background. Section 3 presents other approaches for IoT testing. Section 4 explains our approach and testing process. Section 5 contains specifics about the used systems and performed evaluations. Lastly, Section 6 concludes this paper and gives an outlook on future work.

2 BACKGROUND

This section presents the technologies used in our approach.

2.1 Distributed Dataflows

Increasingly IoT applications make use of edge and fog computing approaches due to their locally wide spread device deployment. At the same time, the collected and processed data is produced continuously by sensors. The resulting data streams make the processing of acquired data in batches counter-productive. Therefore, IoT applications typically use stream processing approaches.

Streaming applications are represented by directed graphs where vertices are operations and edges are data streams. Data streams are possibly infinite sequences of data elements, oftentimes represented as messages or key/value pairs. Streaming applications thereby are a highly pipelined model of data-parallel computing [5]. To facilitate the streaming methodology, higher order operators are used for the implementation of functions inside a dataflow graph. These higher order operators constrict functions to certain sets of parameters and results to fit into the streaming framework and to ensure the expected functionality.

2.2 Virtualization

For the deployment of stream processing tasks dataflow platforms oftentimes provide interfaces for operating system level (OS-Level) virtualization solutions. OS-Level virtualization puts the virtualization layer right above the operating system of the host machine. The operating system offers separated operation environments to so called containers which share hardware and operating system. One computer can thereby offer multiple isolated machines with different execution environments simultaneously [6].

The lowest layer of virtualization exists at the instruction set level. Here, instructions of virtual machines never directly go through the host CPU. Instead, all instructions, including calls to hardware resources are translated before being issued to the CPU. This approach facilitates setups where the virtual guests and the host are not running on the same instruction set. Due to the high translation cost, the gained support of a high number of platforms comes with a significant performance penalty [6].

The machine emulator Qemu is one of the most used system emulators. It currently supports the emulation of 22 different platforms. In its system emulation mode a full computer can be emulated, including one or more processors as well as peripheral devices [1]. In addition to the emulation of CPUs, Qemu emulates devices such as displays, serial ports, hard disks, and network cards. The possibility to emulate specific embedded devices by adding new machine descriptions and devices makes Qemu an interesting tool for the parallel virtualization of sets of heterogeneous devices as found in the IoT.

3 RELATED WORK

This section presents an overview of several commercial and academic network emulation tools and simulation platforms tackling the complexity of testing in heterogeneous, geo-locally distributed IoT environments.

3.1 Network Emulation

ns-3 [17] is a free software solution for the simulation of discrete-event networks. Networks can be freely specified by defining nodes and connections including their network capabilities, failure probability and other node properties. Due to the high depth of the simulations and their many configuration parameters, setup and configuration of simulations becomes very complex.

A much simpler network simulation solution is the extension of the Quality of Service (QoS) facilities of the Linux kernel called NetEm [9]. NetEm worsens existing network connections, for example by increasing the delay or losing packages. The objective is to make it possible to test applications that are supposed to run in wide area networks (WANs) with large distances in local area networks (LANs). Instead of having to deploy an actual network with a high number of devices and large distances, the presumed connectivity effects are simply configured into the network devices. The netem kernel module intercepts the regular traffic going over a network interface and applies the configured interference before transmission by implementing two queuing disciplines. The command line utility used for configuring the module is an extension to the traffic shaping tool *tc* and part of the *iproute2* package.

3.2 IoT Simulation Platforms

The commercially available IoTify [1] application development platform was created in order to develop and test applications for the IoT. It is targeted at programmers with little experience concerning the pitfalls of the IoT and available platforms and protocols by providing software building blocks and a suggestion engine recommending APIs and platforms to use. It virtualizes IoT devices in a cloud to test the developed prototypes, making a development effort independent of the availability of hardware. To a large extend, this offer focuses on a quick start for IoT projects. However, since there is no data available on the adequacy of IoTify's virtualization approach and simulation results, it is hard to assess its feasibility in research and production environments.

The European Commission funded program "Federated Interoperable Semantic IoT Testbeds and Applications" (FIESTA-IoT) [20] integrates IoT platforms, testbeds and associated silo applications under one framework. The goal of the project was the creation of a single API to use for the execution of IoT experiments. Currently, it offers testbeds of varying sizes at ten locations in the EU. By providing a unified API, the experiments can be created in a platform agnostic manner. The finite set of testbeds is the main drawback of this project.

The developers of the large-scale IoT emulator MAMMoTH [14] focus on the emulation of edge layer devices and networks. They criticize that while simulators enable a close replication of processes in an IoT environment, they do not actually behave like the target systems. Their solution provides a testbed of VMs to create, deploy and monitor experiments. To support its large scalability, it runs as many nodes as possible inside one VM leading to the conclusion that each VM also emulates a high number of links. Due to this architecture, tens of thousands of nodes can be emulated but experiments are limited to distinct scenarios. Also, the end devices themselves are not emulated fully, but rather certain aspects of them using the COOJA emulator [16].

Focusing on the emulation of fog infrastructures, MockFog [8] implements a testing environment in the cloud where each device is emulated by a virtual machine in a cloud service such as Amazon EC2. Similar to our approach MockFog emulates both, devices and network properties. While computation resources of devices are approximated, their emulation is limited to the available VM configurations.

4 THE HÉCTOR FRAMEWORK

While the presented approaches in Section 3 are quite mature, they do not offer an easily usable and extendible solution for automatic IoT testing. This section illustrates our approach for an IoT testing framework.

4.1 Design

The goal of this research is to investigate if a unified testing methodology can be found that serves the highly divergent IoT architectures. This is approached by the design of an IoT experimenting framework that facilitates arbitrary testbeds to be used with an expandable set of dataflow platforms. The framework should be

[1]https://iotify.io, accessed 2019-05-23

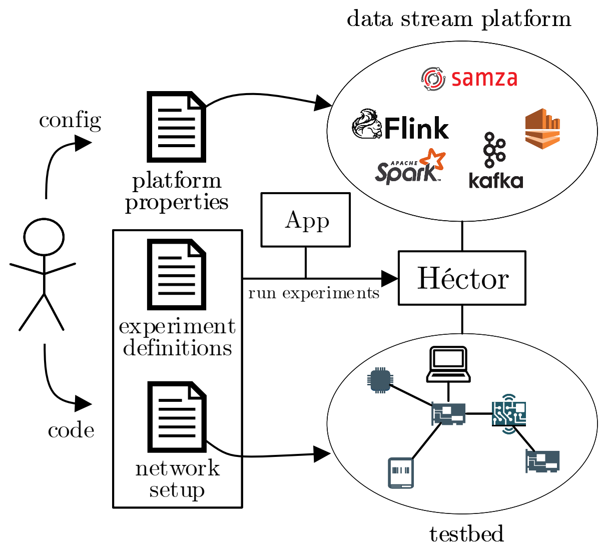

Figure 1: Framework design and testing process

able to be used to easily run chains of automated experiments, independent of their used dataflow platforms, testbeds, and application code. Its design is guided by the following considerations.

1. Flexible Experiment Definitions. To facilitate flexibility, it must be possible to create automated chains of experiments where all properties are variable between runs and benchmark results between the runs are saved in a comparable fashion. At the same time, experiment descriptions should be structured and uniform.

2. Extensible Platform Support. Depending on the application one of several different dataflow platforms might be used. Therefore the framework must implement support for a set of dataflow platforms and provide the potential to simply extend this set.

3. Arbitrary Testbeds. To support different testbeds, it is necessary that existing topologies can be added to a setup, so that the connected machines become available. Next, a means to add virtual topologies is needed. For this, VMs acting as edge or fog devices, linked in a virtual network, are required, allowing to easily perform experiments under different network conditions and on varying scale. The migration among testbeds should be designed to be seamless.

The matter of the wide spectrum of platforms that is taken into consideration for IoT task deployment defines the main imperative of keeping the framework extensible. Additionally, the second design consideration (*Platform Support*) is attended by a non-static way of executing binaries. An API call can be used to define the specific Linux command to use on the executable or upload an additional script that prepares the devices before execution. It is also conceivable that code is uploaded to a certain (possibly emulated) device and compiled there before deployment in the IoT network.

By extending the framework, platform specific functions can be added without changing existing code and risking impairing its functionality. In our case, the API provides a set of methods that takes up the tasks of setting up experiments, preparing IoT environments as virtual testbeds and running the application code in the respective platform's context. Since it still requires programming, experiments can be wrapped in arbitrary code to be chained or to collect metrics at certain points of the execution serving the first design consideration (*Experiment Definitions*).

The third consideration (*Testbeds*) is also attended by the use of an API. All entities of a test can be represented as a data structure. This includes the network, all its nodes (i.e. machines), and the specific experiment definitions with their runtime environments. As long as an IoT device has the necessary properties such as an IP address, it can simply be added to a virtual network as a corresponding data structure.

4.2 Framework Structure

Figure 1 illustrates the resulting framework and the IoT testing process, which we call Héctor. Users have to provide system and dataflow platform specific configurations as well as an experiment definition to the framework. The latter defines emulated network properties and the testing flow through the use of API functions. The following tasks are performed by the framework.

(1) Set up VMs
(2) Connect networks
(3) Set up dataflow platform
(4) Run experiment sequence
(5) Collect metrics
(6) Turn down system

Considering the three design considerations and their discussion as well as the stated goal, we propose a framework consisting of the following main components.

Virtual Network. The Virtual Network forms the core of an experiment. It offers functions to add VMs as well as local host workers to the testbed. This entity is also responsible for the management of all VMs in the testbed, copying and running them. Network conditions can be modified by setting the corresponding properties here. These may include additional delay, random delay variation with an optional correlation, delay distribution, packet loss, packet duplication, packet corruption, and packet reordering emulated using NetEm. Destructing a Virtual Network is synonymic to closing the testbed. All VMs are stopped, their files are deleted, physical machines are reset, and the virtual network is removed.

Virtual Machines. The Virtual Machine represents an instance of a Qemu VM holding all necessary invocation information. It is possible to shape the network traffic to and from a single VM the same way as for the whole virtual network.

Physical Machines. Even though physical IoT devices are running and reachable by the host, a logical representation of them is necessary for management. This representation includes the IP addresses under which they are reachable and authentication information to run worker tasks on them.

Experiment Properties. Next to the IoT devices and their connecting network topologies, experiments must contain the actual context they run in. This includes on the one hand the dataflow platform that is used and on the other hand the actual application code. Data structures are used to hold all configurations necessary to set up the underlying platform for the experiment. The methods that are implemented take care of everything regarding the running of the application. Additionally, the set up experiments are started and stopped with appropriate methods. As the required parameters and set up steps are very different between different platforms, each supported platform needs an implementation of this part.

4.3 Emulation of Devices

To meet the third design consideration (*Arbitrary Testbeds*), the framework has the ability to set up VMs, acting as edge and fog devices. The downside of OS-level virtualization is that it is not able to create heterogeneous testbeds. While the containers are isolated from each other, they all run under the same operating system and system architecture as the host machine (cf. Section 2.2). Since the used edge and fog devices in IoT setups are heterogeneous, we have decided to use a system emulation technology. The operating system and system capabilities such as the number of cores and the amount of memory can be equal to the target systems. Secondly, the migration to a physical testbed with real machines can be designed to be seamless. The edge devices do not have to be able to run the same containers as the development host. When disregarding limitations set by the dataflow platforms, even target systems that do not run operating systems can be included in virtual testbeds and applications can be moved to real world counterparts with no extra effort.

Each VM is spawned as a separate Qemu process. To this end, the used and prepared disk image files must be available on the host. To ensure isolation and immutability by the experiments, each VM invocation is done on a separate copy of the passed disk image which is automatically duplicated by the framework.

4.4 Virtual Networking

To create a viable testbed, all devices are connected to a network set up by the framework. This network connects the invoked VMs as well as the available physical machines. To this end, two tasks are performed:

(1) Creation of a virtual network bridge to connect all VMs to the host.
(2) Configuration of the host's network interface and routing table additions to enable the physical devices to also connect to the virtual network bridge.

The virtual bridge interface is the central part of each topology. Furnished with an IP address, it acts as the access point for all communication. Depending on the used dataflow platform, the respective master entities such as the brokers in a Kafka setup or the JobManager in a Flink setup listen on this device to manage the worker nodes.

VMs are connected to the bridge using using TAP drivers. TAP interfaces are kernel interfaces that are not backed by a real network adapter. They simulate a link-layer adapter to which user-space processes can attach themselves. Here, the emulated network interface of each Qemu VM is connected to a TAP interface, which in turn is connected to the virtual bridge. Since TAP interfaces operate on Layer 2, they simply bridge ethernet frames, without specific routing or an IP address.

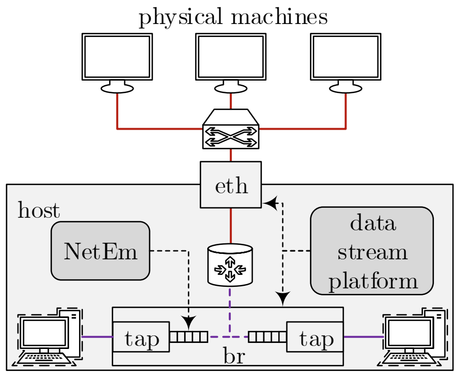

Figure 2: Hybrid testbed setup with Héctor-managed host

Furnishing each VM with a separate and known IP address poses some difficulties. Since the framework uses prepared disk images as blueprints for multiple VMs, it is not possible to preset IP addresses in the disk image's operating system. Additionally, the process needs to stay dynamic so VMs can be added at will. To solve this, the Dynamic Host Configuration Protocol (DHCP) is used. A DHCP server is run on the host that is configured by the framework between setting up the VM properties and starting them. Each registered VM is given a separate Media Access Control (MAC) address which is presented by its emulated network device. Each of these MAC addresses is then assigned a different IP address in the virtual testbed address space and the combinations are stored. The DHCP server can now use the stored MAC-IP combinations to hand out the right IP addresses to the VMs once they have finished booting and broadcast a DHCP discovery message.

Physical devices can be added to virtual testbed by providing their IP addresses and authentication information to the framework. To this end, physical machines need to be up and running Secure Shell (SSH) servers to connect to. Fig. 2 illustrates the necessity for the host to act as a router for the participating networks in a hybrid testbed.

5 EVALUATION

In this section we present experiments run on a working prototype for evaluation purposes.

5.1 Prototype

To test the feasibility of the proposed approach, we implemented a prototype of Héctor. To reflect the design structure of Section 4.2, the API is implemented using an object-oriented Python program, written to work under Python 3.7 on Linux operating systems. Developers of experiments can simply import the IoT API module and write the test cases around its functions. Currently, it supports applications written for Apache Flink [3] and Apache Kafka [12] by containing Experiment-class implementations for both platforms. All networking tasks are performed by calling *iproute2* tools. The physical machines as well as the blueprint of the VMs are prepared with RSA keys for automated access.

Experiments are prepared by providing Python code using the API's functions and a static configuration file that is dependent on the used platform.

5.2 Experiments

We conducted three experiments with different testbeds and dataflow applications.

5.2.1 Setup. All experiments were performed using a host computer with 16GB memory running on an Intel i7 7700K CPU with four cores and simultaneous multithreading to 8 concurrent threads. Different testbeds were used containing only physical machines, only VMs and both. Five Raspberry Pis 3+ running on Cortex-A53 quad-core CPUs with 1GB memory were used as physical edge devices, connected in a wireless LAN through a 300Mb/s IEEE 802.11n switch. The host was connected to the same switch through a 100Mb/s ethernet port. The VMs were emulated Raspberry Pis with ARM1176 CPUs and 256MB memory.

5.2.2 Applications. The used application was similarly implemented for Flink and Kafka and represents a scenario, where multiple edge devices collect sensor data in a locally distributed manner. The messages created by the edge nodes are key-value pairs where the key represents the node id and the value a sensor message containing a measurement and a time stamp. To model real sensor networks, each source is scheduled to a different task slot or producer respectively, starting a processing pipeline in which sensor messages are filtered for significant measurements, grouped by key and averaged over windows of 10ms. Finally, the averaged measurements are submitted to a compute intensive multiplication operation. The Flink application was run with two task slots per machine and the Kafka application with four producers per machine.

5.2.3 Test Experiments. To test the feasibility of the framework and explore the impact of device emulation in IoT testbeds, several experiments were performed using the application. In the following, three of the most significant experiments are outlined.

In the first evaluation, performances of VM testbeds were compared to purely physical testbeds. The Kafka application was run with four to twenty producers and the Flink application with two to ten data source tasks, both running on one to five machines. The experiments were run under the same constraints through the Héctor framework and throughput as well as latencies were measured.

To test the network emulation functionality of the framework and the impact of network properties to IoT applications, a chain of experiments was conducted while varying the network delay of a virtual testbed. In this evaluation, the number of devices was kept constant with five emulated machines, each running two producers or task slots respectively. Both applications ran with eight different configurations. The network emulation applied increasing delays on the connections to all VMs from 10 to 300ms. Additionally, a small probability for package loss of 2-4% was added. It was examined, whether the externally added delay had an impact on the distributed processes that could be seen in the latency and throughput graphs.

To explore the use of hybrid testbeds in the framework, the IoT test applications on Flink and Kafka were run on testbeds including both, physical and virtual machines. To see possible effects of the addition of virtual machines to a network, we performed the experiments eight times, increasing the number of machines in the testbed. The first two tests only included physical edge devices. Virtual edge devices were then added to the five running physical machines. Each machine held two data source threads in the Flink application and four producer threads in the Kafka application.

5.3 Results

The execution of the performed experiments showed, that the developed framework prototype enables the testing of different applications running on different platforms and testbeds in a simple manner. Experiments can be easily performed without the necessity to make changes to the code or setup. The inclusion of agnostic testbeds has shown to be possible without difficulties.

The results of the first experiment from Fig. 3 show that an increase of data producing sensors leads to the expected rise of throughput for both, physical and virtual testbeds. Since more data has to be transmitted and processed, the latencies increases with high numbers of sensors. Especially the performance on virtual testbeds suffers due to the additional resource demands on the

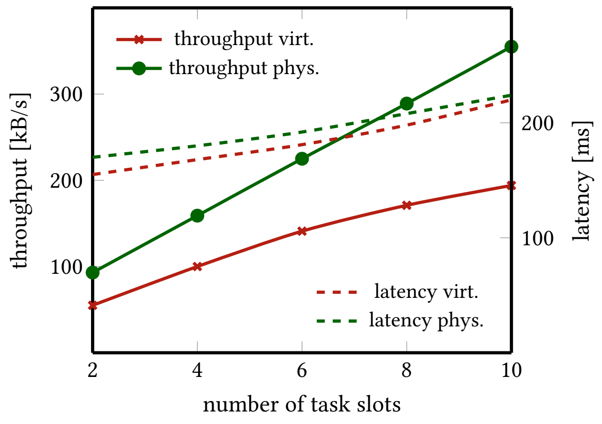

Figure 3: Machine type comparison on Flink setup

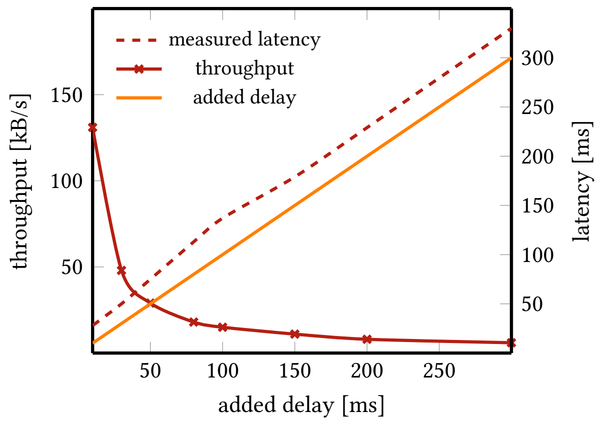

Figure 4: Network emulation evaluation on Kafka setup

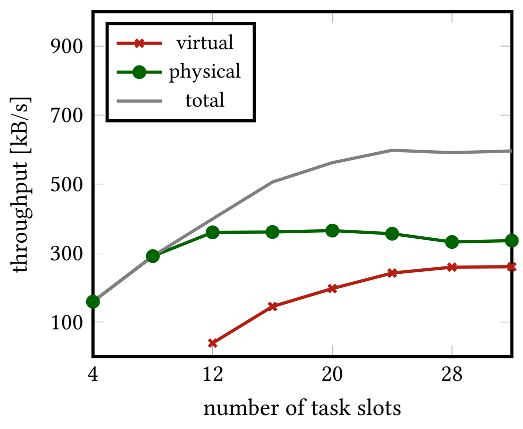

Figure 5: Hybrid testbed experiment on Flink setup

host system. Since this hinders the scalability, physically separating system emulations from experiment processing could be of avail.

The results of the second experiment illustrate the impact of high latencies to distributed Kafka applications. As can be seen from Fig. 4, the throughput generated by Kafka records drops quickly once additional delay is added. The plots also show that an application of additional delay does not furthermore increase measured latencies.

The results of the third experiment from Fig. 5 show the measured throughput in the hybrid testbed when increasing the number of TaskManager nodes. Since no further physical machines are added after 10 task slots are running, their throughput remains constant after that point. Similar to the first experiment, the throughput generated by virtual machines does not increase linearly with the number of task slots due to the high emulation overhead on the host machine.

6 CONCLUSION

In this paper, we investigated solutions for a flexible framework to run IoT experiments on arbitrary and framework-agnostic testbeds. To this end, we designed and implemented the Héctor framework, which allows IoT developers and researchers to run automated experiments. By delivering an experiment definition composed of Python code and a static configuration file, any testbed of Linux-running edge devices can be used to run stream processing applications on. To be able to use many different dataflow engines the framework allows modular extensions for dataflow platforms. Furthermore, we added the functionality to set up virtually networked, emulated devices automatically and further use network emulation to be able to test distributed stream applications intended for the IoT without the need for the actual hardware.

To evaluate our testing approach and explore the limits of the edge device virtualization, several experiments were performed. These experiments compare virtual with physical testbeds and examine the impact of network properties. They demonstrate the functional viability of the framework for IoT applications, yet the measured performance impact of emulation points to the necessity of physical separation between edge device emulation and application data sinks in sensor network IoT scenarios.

To further investigate the use of system emulation in IoT testbeds, additional tests need to be performed. The virtual network set up by the framework is limited in its capabilities since all emulated machines have to be directly connected to the same virtual bridge device. To be able to emulate complex networks of VMs, a network simulator could be added to the framework. The VMs then need to be automatically loaded into the simulated network.

ACKNOWLEDGMENTS

This work has been supported through grants by the German Ministry for Education and Research (BMBF) as Berlin Big Data Center BBDC2 (funding mark 01IS18025A). Moreover, we thank Ilya Verbitskiy for his comments on a draft of this paper.

REFERENCES

[1] Fabrice Bellard. 2005. QEMU, A Fast and Portable Dynamic Translator.. In *USENIX Annual Technical Conference, FREENIX Track*, Vol. 41. 46.
[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12)*. 13–16.
[3] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin* 36, 4 (2015).
[4] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. 2015. Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander. In *2015 IEEE International Congress on Big Data (BigDataCongress '15)*. IEEE, 592–599.
[5] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B Zdonik. 2003. Scalable Distributed Stream Processing.. In *First Biennial Conference on Innovative Data Systems Research (CIDR)*, Vol. 3. 257–268.
[6] Susanta Nanda Tzi-cker Chiueh and Stony Brook. 2005. A Survey on Virtualization Technologies. *Rpe Report* 142 (2005).
[7] João Pedro Dias, Flávio Couto, Ana CR Paiva, and Hugo Sereno Ferreira. 2018. 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, 104–109.
[8] Jonathan Hasenburg, Martin Grambow, Elias Grünewald, Sascha Huk, and David Bermbach. 2019. MockFog: Emulating Fog Computing Infrastructure in the Cloud. In *Proceedings of the First IEEE International Conference on Fog Computing*.
[9] Stephen Hemminger et al. 2005. Network Emulation with NetEm. In *Linux Conference Australia*. 18–23.
[10] G. Janßen, I. Verbitskiy, T. Renner, and L. Thamsen. 2018. Scheduling Stream Processing Tasks on Geo-Distributed Heterogeneous Resources. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 5159–5164.
[11] Hermann Kopetz. 2011. *Internet of Things.* Springer, 307–323.
[12] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A Distributed Messaging System for Log Processing. In *6th International Workshop on Networking Meets Databases (NetDB)*. 1–7.
[13] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. ACM, 751–766.
[14] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Ylä-Jääski. 2012. Mammoth: A Massive-Scale Emulation Platform for Internet of Things. In *2nd International Conference on Cloud Computing and Intelligence Systems*, Vol. 3. IEEE, 1235–1239.
[15] M. Ma, P. Wang, and C. Chu. 2013. Data Management for Internet of Things: Challenges, Approaches and Opportunities. In *GreenCom/iThings/CPSCom 2013*. 1144–1151.
[16] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. 2006. Cross-Level Sensor Network Simulation with Cooja. In *Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*. IEEE.
[17] George F Riley and Thomas R Henderson. 2010. The ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*. Springer, 15–34.
[18] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov. 2016. SpanEdge: Towards Unifying Stream Processing over Central and Near-the-Edge Data Centers. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 168–178.
[19] M. Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (Jan 2017), 30–39.
[20] Martin Serrano, Amelie Gyrard, Elias Tragos, and Hung Nguyen. 2018. FIESTAIoT Project: Federated Interoperable Semantic IoT/Cloud Testbeds and Applications. In *The Web Conference 2018*. Web Conferences Steering Committee, 425–426.