Communication papers of the Federated Conference on Computer Science and Information Systems, pp. 229–235
DOI: 10.15439/2017F394
ISSN 2300-5963 ACSIS, Vol. 13

Leveraging virtualization for scenario based IoT application testing

Tomasz Szydlo
AGH University of Science and Technology,
Department of Computer Science, Krakow, Poland
Email: tszydlo@agh.edu.pl

Joanna Sendorek
AGH University of Science and Technology,
Department of Computer Science, Krakow, Poland
Email: send.joanna@gmail.com

***Abstract*—From day to day we can observe an increasing adoption of Internet of Things in the world. Number of devices connected to the Internet is constantly growing. At the same time, the applications are getting more and more complex and its development becomes more difficult and requires testing. Access to the IoT testbed and possibility to execute the subsequent version of application in the same environmental conditions is desirable. In the paper, we present the concept of virtual testing environment which enables a possibility to create on demand Linux virtual devices which have access to virtual sensors and effectors deployed in the emulated environment driven by the user defined scenarios.**

I. Introduction

THE Internet of Things settles rapidly in the world. Not only in the Smart City initiatives [1], smart homes, agriculture [2], but also enterprise adoption of the Internet of Things (IoT) [3] is starting to gain momentum. The concept of Industry 4.0 where machines will communicate directly with each other providing more flexible production setup and ability to react to changing conditions, increases the data variety, volume and the necessity of data processing at the network edge [4].

IoT applications are getting more and more complex which causes their implementation very challenging and error prone thus the application testing is necessary. There are several tools simplifying testing process. For example, in cloud platforms [5] such as Samsung Artik or IBM IoT there are the device simulation modules which are able to generate randomized data in order to verify the processing in the clouds. At the same time, WSN testbeds [6] are designed to verify low level interactions between devices. Such testbeds constitutes of several, sometimes even hundreds or thousands, of nodes deployed in the particular place. Additional infrastructure enables a possibility for node programming and debugging. There are also projects aimed at testing networking in the virtual environments. The ns-3 project [7] is a discrete event network simulator developed for networking research using the network and protocols models. In contrast, the CORE [8] emulator is a tool for emulating networks on one or more machines using lightweight Linux virtualization. Unfortunately, these solutions does not allow testing the whole IoT stack for data-driven application where the processing depends on the data gathered by the sensors. This is especially important in the Fog Computing concept [9], [10] where computations from the cloud are moved closer to the sensors in order to decrease the processing latency and data throughput from the sensors to the cloud through the Internet [4].

In the paper, we propose the concept of virtualized platform for testing IoT applications which enables a possibility to create virtual devices which along with real embedded devices could be put in the emulated environment driven by the scenarios. Devices instead of reading data from real sensors will be using virtual ones located in the virtual environment. This enables a possibility to repetitively execute the testing scenarios and observing how the new versions of tested application behave facing the same readings from the sensors. Moreover, such virtual environment composed of the virtual devices can be set up on almost any computer simplifying the development process of the IoT applications. In the paper we are focusing on the Linux based IoT devices but the presented concept can be further extended to cover other types of the devices.

The scientific contributions of the paper are: (i) introduction of Linux based embedded devices virtualization concept (ii) introduction of sensors virtualization concept (iii) event-driven environment emulation framework (iv) use case showing the functionalities of the framework.

The rest of the paper is organized as follows. Section II describes the virtualization of linux based IoT devices. In section III the scenario based testing and the environment emulation is discussed, while in section IV case study is presented. The last section summarizes the paper.

II. Virtualization of Linux based IoT devices

Operating systems based on the Linux kernel are used in embedded systems such as consumer electronics, personal video recorders, networking devices, industrial automation and IoT. They differs in kernel versions, available modules and standard libraries. Nevertheless, due to the fact that they are all based on Linux kernels, most of the software can be used on any of these platforms.

In the Linux OS for embedded systems, access to the device peripherals is usually achieved by the files in the $/sys$ folder which are there exposed by the kernel modules. Pin numbers and their functionalities varies between embedded boards, so the applications have to be customized for particular hardware.

In order to simplify the application development, Intel implemented the two libraries - MRAA[1] and UPM[2]. First one is for low speed IO communication in C with bindings for C++, Python, Node.js and Java. It supports generic platforms, as well as Intel Edison, Intel Joule and Raspberry Pi. It provides access, among others, to the GPIO (General Purpose Input/Output), ADC (analog-to-digital converter), I^2C (Inter-Integrated Circuit) and UART (universal asynchronous receiver/transmitter) interfaces. The second one - UPM - is a high level repository that provides software drivers for a wide variety of commonly used sensors and effectors. It contains routines for accessing real sensors such as temperature sensors e.g. Texas Instruments LM35[3], humidity e.g. Sensirion SHT31[4] and others. These software drivers interact with the underlying hardware platform through calls to MRAA APIs. The idea is presented in Fig.1a.

For the purpose of device virtualization, it was necessary to virtualize the device and the hardware interfaces for connecting external peripherals and sensors. We have decided to virtualize the device using Docker containers, while for hardware interfaces virtualization, two solutions were considered:

- implementation of Linux kernel modules emulating the real hardware and placing particular files in $/sys$ folder,
- modification of the system libraries that covers nuances of linux distributions and provides unified API for the users - in that case the aforementioned Intel MRAA which is open sourced and used in several projects.

We have decided to implement interface virtualization by modifying MRAA library. The decision was dictated by the fact, that the library covers several hardware platforms and enables possibility to use UPM library. Communication between the modified v-MRAA library on the Linux based device (real or Docker based) and the emulated environment is message driven. It is based on MQTT which is a much used light-weight publish/subscribe communication protocol. Currently, the GPIO and ADC peripherals are supported. Realization of the I2C and UART protocols is in progress and is out of scope of this paper. Fig.1b depicts the situation where embedded device is connected to virtual sensors through emulated interface. In order to use UPM library for reading values from virtual sensors using routines for real ones, their internal logic must be implemented in environment emulation framework, which is discussed in the next section.

Fig.1c depicts the situation, where the real embedded device instead of real sensors is connected to sensor emulation module. The concept is further elaborated in the previous work [11].

III. Scenario based testing

One of the biggest issues behind complex system testing of IoT solutions, which are based on sensors and real data processing, is lying in the unpredictability of surrounding world and its changing conditions being the input for the system. While the separate components could be mocked for testing purposes, the necessity for being able to reproduce particular changes of the whole conditions, often dependent on each other, seems to be gaining in importance with more emerging IoT applications based on sensors measurements. The concept of applying the scenario-based testing to simulate behavior of external conditions, its influence on devices and interactions between them, is presented in this paper. We have made assumption that not only should person writing a scenario be able to specify exactly what is happening at the given time or under certain conditions, but also be able to apply potentially non deterministic mathematical model as the simulation base. This approach enables testing of the whole application including data analysis models and real data processing. In the least complex variant, the mathematical model could be reduced to replaying already recorded measurements.

A. Environment emulation

Emulation model presented in the paper consists of two main entities: (i) `the world` representing all parameters which can be measured and which are changing along with a particular model and (ii) `observers` which are able to read specified values of world's parameters. World itself has space mapped to measurement vectors V = (v_0, v_1, v_2, ...) where v_i is the specific property of world able to be measured by sensors in the real world and to be observed by observers in the emulation model e.g. temperature, humidity etc. Main idea behind the proposed environment construction is the separation of `the world` which is responsible for simulating physical mechanisms from `observers` which main task is to read world's state and convey it further to the actual device (real or virtual) and sensors. Hence, observers have the role of data providers to the devices and their components and world acts as the source of this data. Observers are reading data from the world in the moments meeting requirements defined in `the scenario` while the world is transforming itself starting from the initial state.

One of the challenges coming with the construction of scenario and its execution have been the ability to make observers responsive to certain situations occurring during world transformation. In certain cases, it could be satisfactory to make observers read world state at specified moments in time domain. However, while this approach is closest to the real-world behavior of sensors, it does not address few important issues. First of all, running any transformation based on mathematical computations on computer gives no guarantee as to the length of intervals occurring between iterations of the simulation. Because complexity of model have impact on this length, this could lead to the point when observer reads data from the world while part of the vectors are still at previous state while the others are already in the next one. Another problem could arise when it would be necessary to synchronize few observers or make particular observations consecutive and dependent on each other. Finally, the nondeterministic

[1] http://mraa.io
[2] https://github.com/intel-iot-devkit/upm
[3] www.ti.com
[4] https://www.sensirion.com

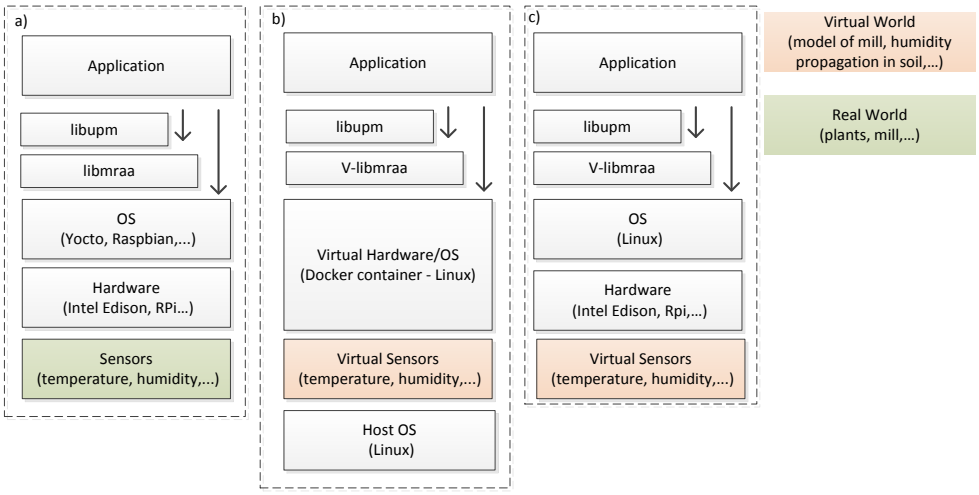

Fig. 1. Access to the sensors on Linux based devices

nature of applied mathematical model may cause some specific circumstances occur in unpredictable moments in time.

All of the mentioned issues decide on insufficiency of the time-based procedural scenario execution. Solution proposed in the paper is inspired by the reactive programming paradigm gaining lately on popularity. This lead to implementation of event-based scenario framework in Python, which enables both time based and reactive observing. In order to inform observers that world's state has already changed, special events can be produced after each iteration cycle which world needs to transform its measurement vectors from state x_k to state x_k+1. In the proposed solution, scenario consist of steps - function declarations written in Python which can be decorated using predefined decorators in order to decide on moment of their execution. Decorators provided by framework for the time of writing are:

Time-based:

- `@every(start=..., seconds=...)` - periodic execution
- `@after(seconds=...)` - one-time execution at given moment

Event-based:

- `@every_event(event=...)` - execution on the each occurrence of event
- `@on_event(event=...)` - one-off execution on occurrence of event

Example 3.1 (Python): Example scenario written in Python framework

```
@every_event(event=World.ITERATION_COMPLETED)
def steps():
    iteration = world.iteration_counter
    if iteration % 3 == 0:
        emit_event("third_iteration")

@every_event(event="third_iteration")
def steps():
    print("Third iteration completed")

@every(start=8, seconds=2)
def steps():
    print("Some task A")

@after(seconds=11)
def steps():
    print("Blocking world")
    emit_event(World.WORLD_PAUSE_EVENT)
    queues_dictionary[World.PAUSED_EVENT]
                .get(block=True)
    print("World is paused")
    print(world.space[24][24]
                .vector.temperature)
    emit_event(World.CAN_RESUME)

if __name__ == '__main__':
        execute(world)
```

Example 3.1 presents scenario which consists of four function declarations, each representing different behavior of observer corresponding with specific event or time moment, declared in the decorator for function. Each event occurring during scenario execution has to be correlated with at most one observer function declaration, however it may be also generated or reproduced during function execution as presented in first declaration.

The idea of observer functions being able to withhold execution of world transformation stems from the necessity for confirming that world state read by observer after one completed iteration has not yet been modified by the following one. Implementation of world being withholdable on WORLD_PAUSED_EVENT should be the responsibility of programmer implementing world thus making it responsive to any events in the more complex example. Described mechanism is also enabling few observers to act during the same interval between particular two iterations.

As the possible development direction for framework, we consider adding effector functions which would have the

ability not only to read world state at particular moment and pause world transformation, but also to influence actual way of transformation e.g. modifying underlying mathematical model responsible for data generation.

Underlying implementation of framework is based on blocking queues stored in the common dictionary, which is mapping event name to corresponding queue. When particular observing function declaration in scenario is decorated, the decorator handles execution of this function based on events appearing in the queue. When new event name is being detected during scenario execution, appropriate queue is added to the queues dictionary.

B. Sensor virtualization

During the scenario, `observers` read the particular values from the virtual `world` as physical values e.g. temperature. In order to read that value by the virtual device using v-MRAA and UPM libraries, the decision should be made which type of hardware sensor should be virtualized - for example DS18B20 by Maxim or LM35 by Texas Instruments.

In the case of LM35 temperature sensor, the voltage provided by the sensor is linearly related to the temperature (0mV + 10.0mV/'C). The voltage can be measured by the ADC input on the virtual device. The python class `LM35_V` in the emulator framework has the method `update_temp(temp)` which computes the voltage that would be generated by the real sensor measuring particular temperature t. The voltage is then converted to the 10bit ADC value $f_{ADC}(t)$ which is sent to the v-MRAA library on the device using MQTT protocol. The maximum voltage measured by the ADC is 5V. The equation is as follows:

$$f_{ADC}(t) = t\frac{10mV}{1^{\circ}C}\frac{2^{10}-1}{5V} = \frac{t*1023}{500} \tag{1}$$

On the virtual device, the application that uses UPM library routine for LM35 sensor will read the already computed voltage value and will convert it to the proper temperature. When the same application would be executed on the real embedded device with LM35 temperature sensor, the value would be read from the real environment.

IV. Automation of environment setup

Building emulation environment described in this paper is done with enclosing each virtual device in one Docker container thus isolating them from platform they would be run on. Usage of Docker containers resolves many problems including running each virtual device independently of the others or ensuring that software required by devices is provided. However, one still has to have Docker platform installed on the machine which will be hosting containers thus making whole environment not fully independent from platform it is build on. What is more, when running multiple virtual devices, one has to repeat the same process of running each container from the same image and setting environment variables for it. In the solution presented in the paper, we aim at environment portable, easy to maintain and as automated as possible. Fig.2 depicts the idea of using automation tools for setting up virtualized devices. In the proposed implementation, we are using `Ansible`, which enables defining all platform setup e.g. installation of required software in YAML files called playbooks. Ansible also has build-in support for docker orchestration which enables creating images from Dockerfiles and running them in the containers.

While usage of Ansible and in general, automation tools, resolves inconvenience stemming from executing repetitive tasks for each instance of virtual device, it does not address issues coming from platform-dependence of installed software. In particular, Linux-oriented playbook will not work when run on Windows operating system and vice versa. Therefore, we have provided additional virtual machine as a docker host and made use of `Vagrant` tool, which enables building virtual environments which are portable and platform-independent. Vagrant has also support for Ansible provisioning thus making all three tools working together well and providing means for easy to build environment.

Fig 2 depicts the tools stack used to set environment up and automate the process.

V. Use case and verification

In order to present the idea of virtual testing environment we have constructed example cloud based IoT application whose purpose was to read temperature from the environment by the two devices and send that data to the cloud platform. As the cloud platform handling devices data, we used the Samsung Artik Cloud[5] platform and applied simple case of plotting data. For temperature reading We decided to use the aforementioned $LM35$ sensors.

In order to test the whole layers of the application during development, some data had to be provided so we constructed two virtual devices. Data was generated by simulation framework and then conveyed to devices as the emulated temperature measurements. Whole test was executed due to scenario constructed in the framework described in section III.

After building emulation environment as described in the previous paragraph, we executed the example scenario. Code for this scenario, composed with framework characterized previously in the paper, is presented by example 5.1.

Example 5.1 (Python): Use-case scenario

```
@every_event(event=World.ITERATION_COMPLETED)
def observe_iterations():
    iteration = world.iteration_counter
    if iteration == 100:
        emit_event("hundred_reached")
    if iteration  == 300:
        emit_event("three_hundred_reached")

@every(start=5, seconds=5)
def read_temperatures():
    temp1 = client1.get_temperature()
    temp2 = client2.get_temperature()
    client1.send_temperature(client1.
```

[5]https://artik.cloud

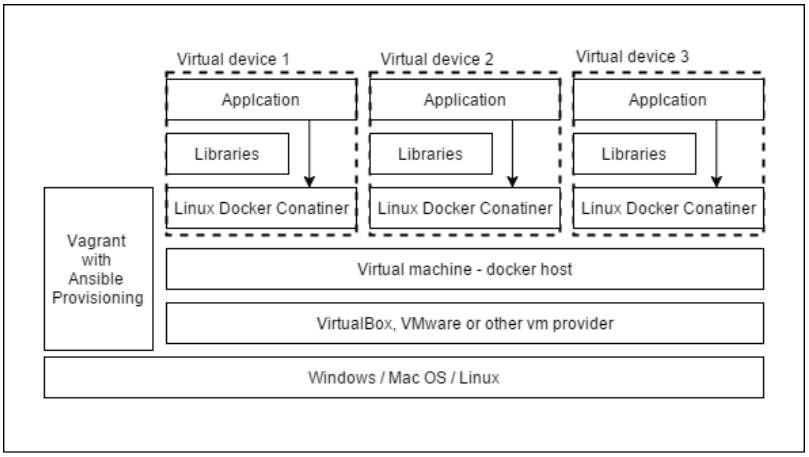

Fig. 2. Virtualization tools applied to building environment

```
                convert_from_celsius(temp1))
    client2.send_temperature(client2.
                convert_from_celsius(temp2))

@on_event(event="hundred_reached")
def add_heater():
    print("Heater added")
    heater1 = World.Heater(size=10,
                           max_temperature=150,
                           bottom_left_x=30,
                           bottom_left_y=20)
    world.heaters.append(heater1)
    world.place_heaters()

@on_event(event="three_hundred_reached")
def remove_heaters():
    print("Heaters removed")
    world.heaters.clear()
    world.is_heater_map.clear()
```

Simulation is based on the simplified heat diffusion process in the two-dimensional space with sensors being able to read temperature at the given points in space. Presented scenario consists of four observer functions. Three of them are event-based and are executed on the occurrence of events connected with iterations. The remaining function is being executed periodically in the time domain, at each five seconds starting from fifth one. Function `observe_iterations` is responsible for emiting events, based on the value of world's iteration counter that are handled by the other functions. Function `read_temperatures` which is working in the time domain, is responsible for providing data to the MQTT clients, which are then sending messages to the devices in simulation environment. Example 5.2 is presenting the initialization code for both clients.

Example 5.2 (Python): Initialization of MQTT clients

```
client1 = MQTTClient.
                LM35_V(1, 0, (20, 25), world)
client2 = MQTTClient.
                LM35_V(2, 0, (30, 25), world)
```

The clients have the role of virtualized temperature sensors LM35 and are instances of class LM35_V, which is taking as constructor arguments: `device id`, `pin id`, `position` of the sensor in the simulation space and the world which is main simulation entity, described in the previous paragraphs. First two parameters are determining MQTT topics, to which the data will be send.

World is starting with two dimensional space of size 50 by 50 units. Space is discretely sampled and each point of the space is mapped to the measurement vector, holding information on temperature value. One heating device with temperature of 150 Celsius is placed at (15, 25) and with each iteration, heat is diffusing on the space with simple mathematical model, reduced to replacing temperature value for the point with average of temperature values of its neighbors. Second heater will be placed in the world during scenario execution. Frequency of world's iterations is 1 sec.

Functions `add_heater` and `remove_heaters` are de facto more effectors than observer functions, because they influence the behavior of the world by placing additional heater and removing all heaters respectively. However, they do not use any synchronization mechanism described in the previous paragraphs and therefore behave in a manner similar to this of observer functions. The example is presenting different possibilities coming with current state of simulation framework, but the aspects of synchronization and separating observer functions from effector functions remain open.

In order to provide some verification for readings made by `read_temperatures` function, we implemented simple plotter that use `matplotlib` library for world simulation. Therefore, each fifty iteration, the plot of world state was generated. Fig. 4 shows plots of heatwave generated after 50, 150, 300 and 350 iterations that corresponds to 50, 150, 300 and 350 seconds since the beginning of the testing scenario, while Fig. 3 depicts graphs of data received by both devices. *"Usecase_temp3"* corresponds to device with virtualized sensor placed at coordinate (20, 25) - device A on figure and *"Usecase_temp4"* corresponds to device with

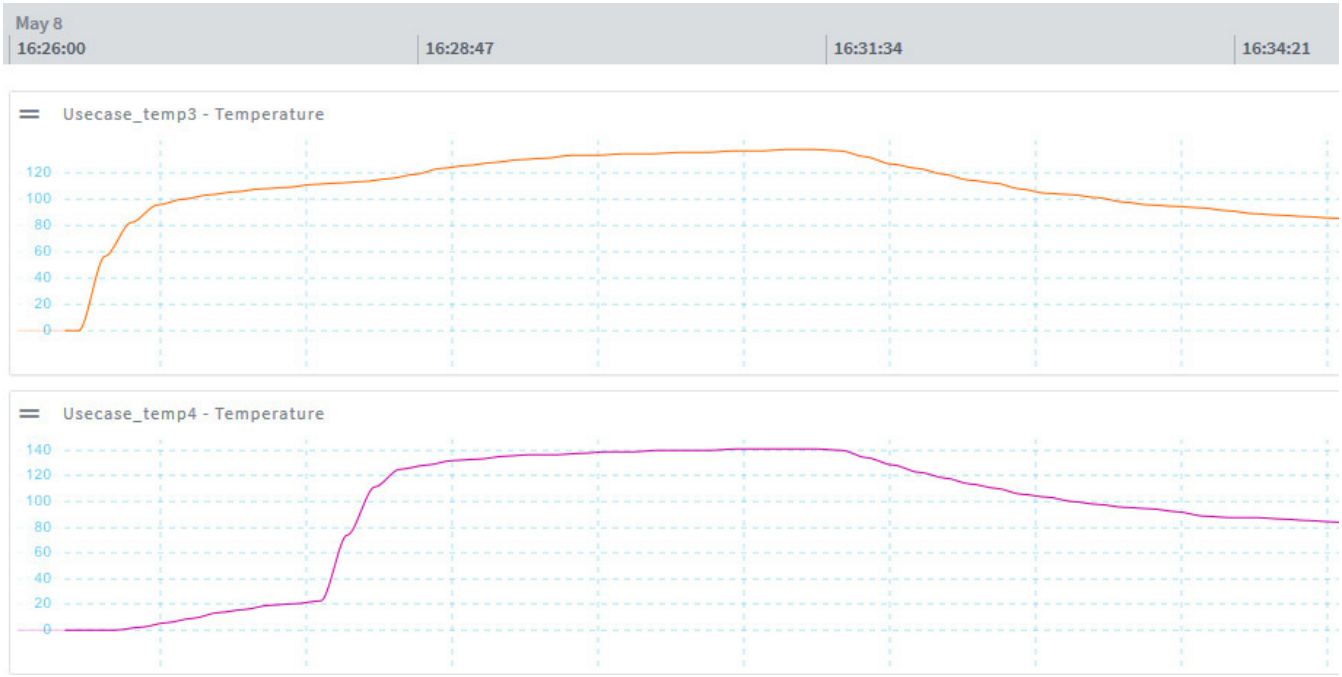

Fig. 3. Plots of data received by device in Samsung Artik Cloud

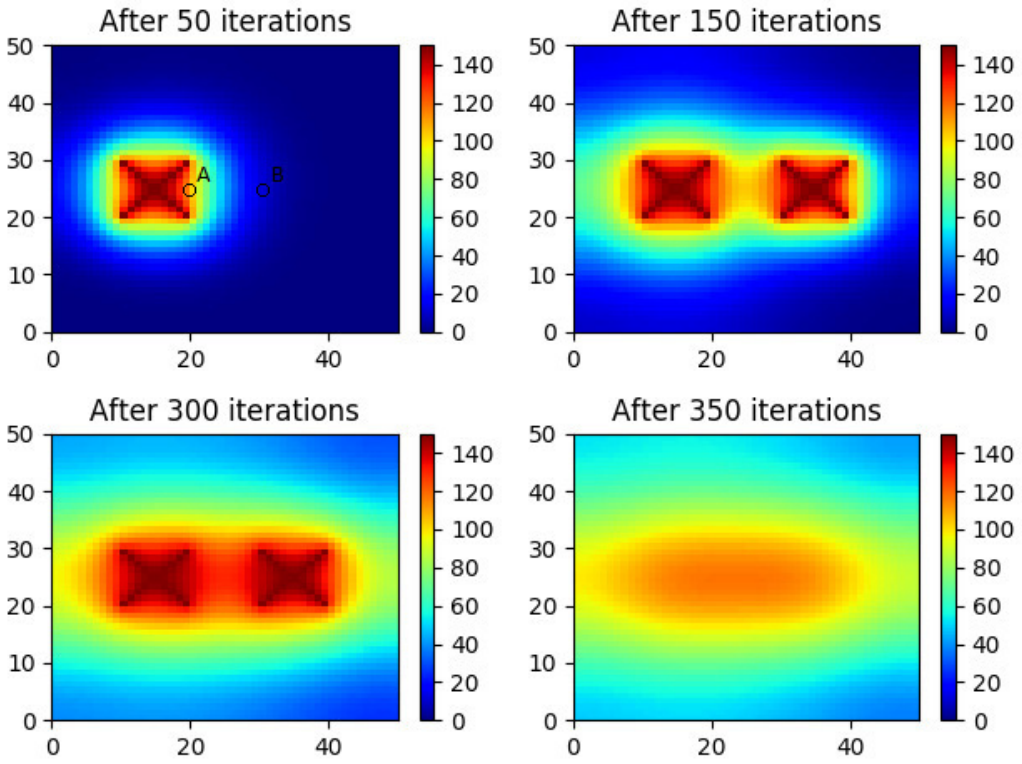

Fig. 4. Plots of world's state corresponding with data received by device

virtualized sensor placed at (30, 25) - device B. There can be observed how firstly, only one sensor was noticing significant changes of temperature, then after 100 seconds the second one started and finally after about 300 seconds, both sensors started noticing declining temperatures.

The presented use case shows how the virtualized testing platform can be used for testing data-driven IoT applications. The virtual machine with docker containers representing virtual devices where executed on i5 notebook with 8GB of RAM, while the emulation environment was executed on i3 desktop computer with 4GB of RAM. During the experiments, the MQTT broker at $iot.eclipse.org$ has been used. It proves that the framework can be executed even on development computers simplifying process of IoT applications development.

VI. Summary

In the paper we have presented the concept of virtualized testbed for scenario based IoT application testing. The concept has been verified by the use case where the virtual environment was used for testing application that measures the temperature using IoT devices and analyzes it in the cloud platform.

As a future work, concept could be extended by adding protocols for I^2C and UART interfaces in v-MRAA library. It would extend the list of possible sensors that could be used by applications. Additionally, the external sensor emulation board would be useful for testing real devices - in that case instead of real sensors, such emulation boards could be connected and driven by the scenarios. Finally, the environment emulation framework can be extended by synchronization concepts simplifying implementation of effectors wanting to change the virtual worlds.

Acknowledgment

The research presented in this paper was partially supported by the National Centre for Research and Development (NCBiR) under Grant No. LIDER/15/0144/L-7/15/NCBR/2016.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[2] J. Shenoy and Y. Pingle, "IOT in agriculture," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, March 2016, pp. 1456–1458.

[3] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, Nov 2014.

[4] R. Brzoza-Woch, M. Konieczny, P. Nawrocki, T. Szydlo, and K. Zielinski, "Embedded systems in the application of fog computing - levee monitoring use case," in *11th IEEE Symposium on Industrial Embedded Systems, SIES 2016, Krakow, Poland, May 23-25, 2016*. IEEE, 2016, pp. 238–243. [Online]. Available: http://dx.doi.org/10.1109/SIES.2016.7509437

[5] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal*, pp. –, 2017.

[6] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental Internet of Things research," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 58–67, November 2011.

[7] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator." in *Modeling and Tools for Network Simulation*, K. Wehrle, M. GÃijnes, and J. Gross, Eds. Springer, 2010, pp. 15–34.

[8] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, Nov 2008, pp. 1–7.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[10] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[11] R. Kaploniak, L. Kwiatkowski, and T. Szydlo, "Environment emulation for WSN testbed," *Computer Science (AGH)*, vol. 13, no. 3, pp. 101–112, 2012. [Online]. Available: http://dx.doi.org/10.7494/csci.2012.13.3.101