Open access • Proceedings Article • DOI:10.1109/GIOTS.2017.8016222

Towards an emulated IoT test environment for anomaly detection using NEMU

Shane Brady, Adriana Hava, Philip Perry, John Murphy ...+2 more authors

Institutions: University College Dublin, University of Bordeaux

Published on: 06 Jun 2017 - The Internet of Things

Topics: Smart objects and Testbed

Title	Towards an emulated IoT test environment for anomaly detection using NEMU
Authors(s)	Brady, Shane; Hava, Adriana; Perry, Philip; Murphy, John; Magoni, Damien; Portillo Dominguez, Andres Omar
Publication date	2017-06-09
Publication information	Proceedings of the Global Internet of Things Summit (GIoTS) 2017
Conference details	Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6-9 June, 2017
Publisher	IEEE
Item record/more information	http://hdl.handle.net/10197/9051
Publisher's statement	© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works
Publisher's version (DOI)	10.1109/GIOTS.2017.8016222

Towards an Emulated IoT Test Environment for Anomaly Detection using NEMU

Shane Brady*, Adriana Hava*, Philip Perry*, John Murphy*, Damien Magoni†, and A. Omar Portillo-Dominguez*
*Lero, School of Computer Science, University College Dublin, Ireland
Email: shane.brady.1@ucdconnect.ie, {adriana.hava,philip.perry,j.murphy,andres.portillodominguez}@ucd.ie
†LaBRI, University of Bordeaux, France
Email: damien.magoni@u-bordeaux.fr

***Abstract*—The advent of the Internet of Things (IoT) has led to a major change in the way we interact with increasingly ubiquitous connected devices such as smart objects and cyber-physical systems. It has also led to an exponential increase in the number of such Internet-connected devices over the last few years. Conducting extensive functional and performance testing is critical to assess the robustness and efficiency of IoT systems in order to validate them before their deployment in real life. However, creating an IoT test environment is a difficult and expensive task, usually requiring a significant amount of physical hardware and human effort to build it. This paper proposes a method to emulate an IoT environment using the Network Emulator for Mobile Universes (NEMU), itself built on the popular QEMU system emulator, in order to construct a testbed of inter-connected, emulated Raspberry Pi devices. Additionally, we experimentally demonstrate how our method can be successfully applied to IoT by showing how such an emulated environment can be used to detect anomalies in an IoT system.**

***Index Terms*—Internet of Things, Emulation, Testing, NEMU.**

I. Introduction

Internet of Things (IoT) has become a major technological revolution which is leading to an explosive growth in the number of internet-connected devices globally. Experts estimate that IoT will consist of approximately 50 billion devices by 2020 [1]. Consequently, IoT is gradually impacting every business area and industry field. Moreover, how effectively these devices are able to communicate (particularly over potentially heavily congested networks) can have a major impact on the quality of the services provided, as well as in their degree of success (or failure). Therefore, properly testing such IoT services is essential [2]. However, testing in IoT is a very challenging task. This is because the intrinsic characteristics of IoT (e.g., the normally vast number of devices and the complex nature of the inter-connectivity among the devices) make the creation of an appropriate test environment of real devices particularly effort-intensive and costly. Likewise, monitoring the health and performance of an IoT system presents similar challenges. This is the result of producing and transmitting huge amounts of data, which makes considerably more complex the detection of anomalous behaviours within the system.

To address these challenges, our research work has focused on developing techniques to improve the processes involved in testing IoT solutions, with a special interest on anomaly detection. In particular, this work presents a novel method to create an emulated IoT environment based on the Network Emulator for Mobile Universes (NEMU), which is a powerful state-of-the-art tool used to create highly customisable virtual networks of virtual machines [3]. This work also shows, through experimental evaluation, how the emulated environment can produce similar results when used at specific settings (compared to a real one), while also significantly simplifying the effort required to build a test environment. This was assessed within an anomaly detection scenario, in which the health of the IoT systems was monitored with log analytic techniques.

II. Related Work

Several research works have focused on developing techniques to simulate IoT environments. For example, the authors of [4] presented an approach to improve the scaling capabilities of the NS3 simulator, so that it is usable for large problem sizes (such as IoT). Despite the fact that simulations are useful to test IoT advancements to a certain degree, they are not suitable for all scenarios. For instance, when the IoT services need to be tested in real-time to assess if they can fulfill their expected *Service Level Agreements*. Consequently, other works have focused on emulating IoT environments where applications can be deployed and executed exactly as they would in real life. Despite the progress in this area, there is still a need for approaches that allow to create more easily, and with less effort, emulated IoT systems [5]. This is because most of the current alternatives are typically hard to use, and difficult (or impossible) to configure at run-time. In 2015, Bagula and Erasmus have proposed an IoT emulation environment with COOJA [6]. However, this environment is targeted only at Contiki systems and emulate motes such as Tmote Sky, Z1 mote, MicaZ mote, etc. It can not be used for Raspberry Pi emulation. More recently, Le-Trung has proposed an IoT testbed emulated over an OpenStack Cloud infrastructure [7]. However, his testbed is based on the QOMET emulator which uses `wireconf`, a software only available on RHEL6 systems.

For these reasons, this paper presents a method that leverages on NEMU to improve that process. NEMU is a tool to create (and manage) realistic virtual dynamic networks, allowing practitioners to test and evaluate prototypes of complex applications (such as IoT services) with a complete control over the characteristics of the emulated environment (e.g., network topology and link parameters). Also, NEMU can create such environments with relatively limited hardware resources, without any administrative rights, and/or distributed over several physical machines [3]. As NEMU uses QEMU for system emulation, it can run a large variety of hardware including Raspberry Pis. For these reasons, NEMU is an attractive candidate to be used in the IoT domain. Finally, anomaly detection is a relevant scenario in IoT, where sensor data and sensor faults are continuously monitored. For instance, the authors of [8] presents a comprehensive list of potential solutions to discover anomalies (e.g., using supervised and unsupervised machine learning techniques). In contrast to other works in the area, this work proposes the usage of log analytic techniques to detect such anomalies.

III. Proposed Method

A. Testbed Architecture

The architecture of the IoT testbed (i.e., the emulated IoT environment) contains multiple components:

1) NEMU is used for the creation and management of a network of virtual machines (VM). An important characteristic of NEMU is that it allows to modify the system at run-time. For instance, by allowing to add (or remove) devices to the network, as well as network links between the devices. This capability allows the introduction of anomalies into the IoT system.
2) QEMU is used to emulate the IoT devices (e.g., Raspberry Pis in our case). QEMU was chosen because it is an open source machine emulator that allows to accurately emulate an IoT device (especially the processor, such as the ARM family of low power processors typically used in SoC-based boards). Alternatively, the practitioner can also use real IoT devices and hook them to the virtual network. This is useful if some specific hardware can not be emulated, such as specific shields.

These components are shown in Fig. 1, which depicts the high-level architecture of the testbed.

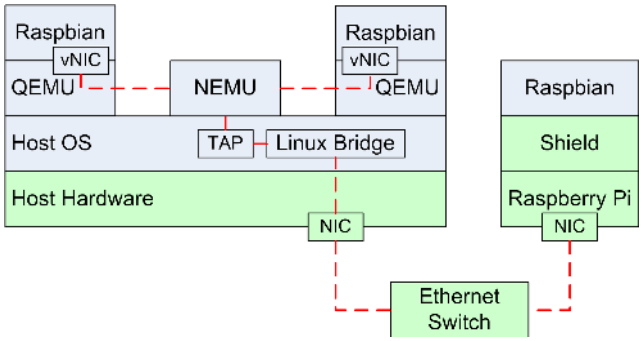

Fig. 1. High-Level Architecture

B. Testbed Generation

The following steps are required to create a working emulated IoT environment:

1) Define the network topology. For this work, we focused on the widely-used tree topology [9], [10] typically used to manage large scale networks. However, any topology can be built with NEMU. At the bottom of the tree structure, there are sensors constantly monitoring and reporting their measured values. On the next layer up there are a series of Raspberry Pis, each responsible for a cluster of sensors. Raspberry Pis are used as they are small, affordable computers, widely deployed in IoT environments. As we proceed up the tree, each higher level Raspberry Pi is responsible for a number of Raspberry Pis below it in the tree. Each Raspberry Pi will report both the sensor data it receives, as well as metrics about the performance of the device itself up the tree. These metrics, including the CPU and memory usage of the device as well as network statistics (such as throughput, packet loss and latency) provide insight into the health of the system as a whole, enabling the detection of anomalies in the network.
2) Create Virtual Host Configurations (VHostConf object in NEMU). Each VHostConf acts as a template For Virtual Hosts (VHost object in NEMU) to expedite the generation of VMs (based on the templates).
3) Specify the architecture (e.g., ARM), the processor (e.g., arm1176) and to provide the kernel for the chosen operating system.
4) Create VHosts that represent the devices (i.e., Raspberry Pis). For each VHost, the experimenter can modify the configuration parameters defined on the VHostConf. A virtual drive image must be provided for the host (i.e., Raspbian Wheezy).
5) Specify the parameters for the virtual Network Interface Cards (VNics). These virtual network interface cards can be attached to a virtual node (VHost) in order to allow it to communicate with other virtual nodes through virtual switches. A VHost can have multiple VNics. As optional parameters, each VNic can be set to a specific card model and hardware address (otherwise, default values are used).
6) Create VLinks, which are virtual networking elements to inter-connect the other virtual entities. A commonly used type of VLink is the VSwitch. A VSwitch is a virtual Ethernet switch device with a configurable number of interfaces. Using NEMU's link command, one can link virtual hosts and virtual Ethernet elements to create various network topologies. For instance, VSwitches, can be used at appropriate position in our scenario, to create a tree topology to connect a set of emulated Raspberry Pis.
7) NEMU provides pre-configured virtual routers

(VRouters) to simplify the management of the network. A VRouter provides services such as DHCP and IP forwarding. A VRouter can also be linked to a VSlirp. A Slirp is a particular virtual point-to-point link which enables a VHost to communicate with the Internet through NAT emulation inside QEMU. Slirp links also contain internal DHCP and DNS servers. This capability is needed to send information from the emulated IoT system to a machine outside of the virtual network (e.g., on the Internet).

8) An optional step is to connect real hardware devices to the emulated network (as shown in Fig. 1). This is done by creating a Linux bridge and a TAP interface on the machine hosting the testbed (i.e., where the QEMU devices are being created). The TAP interface allows software in the Linux user space to be able to use a virtual Ethernet interface on the host machine. A real device can be connected to the host either straight into the host physical NIC or through a physical switch as shown on the figure. Both the host's NIC and the TAP are then connected to the Linux virtual bridge thus creating an hybrid virtual/real network.

Finally, to complement the previous discussion, Fig. 2 presents an exemplary tree topology created using the above discussed method.

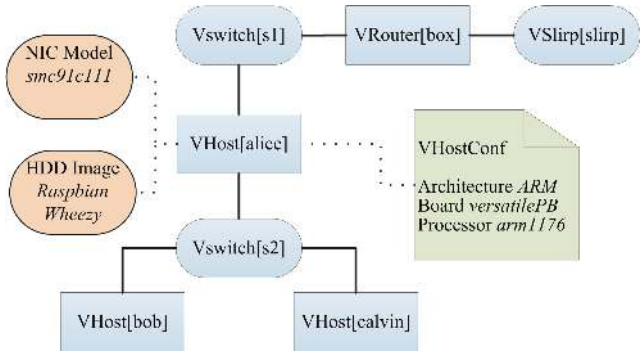

Fig. 2. NEMU Scripting Components

C. Anomaly Detection Use Case

Among the range of available use cases, our work has initially focused on anomaly detection due to its relevance to IoT. In this scenario, an IoT system, composed of multiple devices interconnected through a network topology, periodically transmits their data to a consumer layer such as a NodeRED application. This layer is then responsible for consolidating the received data (potentially from several IoT systems) before sharing it to an analytics service, such as Logentries's log analytics platform, with the aim of identifying abnormal behaviours that might occur in the system.

D. Log Analytics Platform

To interface with the anomaly detection logic, the device at the top of the topology (i.e., the root of the tree) is set responsible for forwarding the sensor data and usage metrics of all devices in the topology from the IoT system to a NodeRED application. It runs independently on its own machine and is used to share the data to a real-time log analytics platform (e.g., Logentries), where the data is analysed to detect anomalies in the IoT system. The NodeRED application can also generate logs (i.e., reflecting its own performance), information which can be correlated with the other logs to deepen the detection of anomalies. Fig. 3 shows the temperature values and anomalies reported in the log analytics system (i.e., Logentries). Temperatures below 19.6C and above 20.4C are flagged as warnings (see section IV).

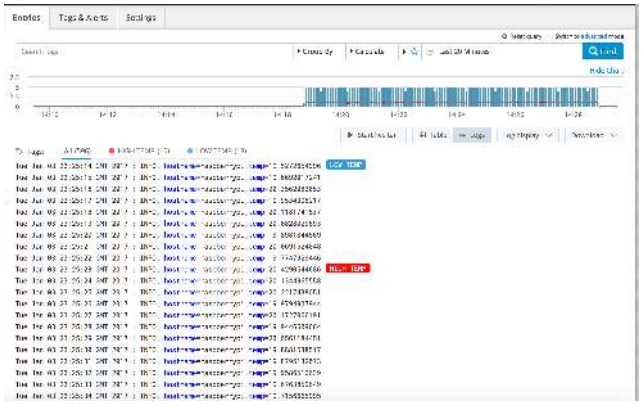

Fig. 3. Temperature data reported at the log analytics platform

IV. Experimental Evaluation

The performed experiments aimed to assess how well an emulated IoT system created with our proposed method and software was able to act as a real one. For this purpose, experiments addressed three scenarios:

1) Measurements of an IoT application using the MQTT protocol.
2) Processor and memory usage of a benchmark application.
3) Network throughput of iperf and FTP applications.

A. Setup

Three types of experiments were done:

1) The first type used a real IoT system built with Raspberry Pi hardware. This real environment was composed of Raspberry Pi 3, model B, and connected by a 100 Mbps Ethernet switch.
2) The second type used an emulated IoT system built with ARM emulation software performed by the QEMU system emulator. The emulated environment was composed of the same number of equivalent emulated Raspberry Pis, using a QEMU specific kernel[1] and Raspbian Wheezy as the Operating System (OS).
3) The third one used a virtualized IoT system built with x86-64 virtualization performed by the QEMU-KVM virtualization software.The virtualized environment was

[1]https://github.com/dhruvvyas90/qemu-rpi-kernel

composed of x86-64 virtual machines leveraging VT-x hardware acceleration through the KVM module and running Debian Wheezy as OS. As the Raspbian distribution used on Raspberry Pis is based on the Debian distribution, these VMs may run code compiled for the same major release without problems.

Both emulated and virtualized environments were running on a Dell Precision T5500 workstation equipped with an Intel Xeon at 2.40Ghz (4 cores/8 threads), 24 GB of RAM, under Linux Ubuntu 15.10 64-bit. All experiments were done in isolation, so that all load was controlled. Also, we used LogEntries as real-time log analytics platform due to its strong analytic capabilities [11].

Two representative anomaly detection scenarios were evaluated: sensor temperature warnings and processor and memory load warnings as already discussed in section III-C. The sensor temperature data was simulated by using a script which randomly generated temperature values by following a normal distribution centered at 20 C and a 2σ of 0.2 C. It ran on the devices found on the leaf nodes of the network, transmitting this information along with the other metrics. To generate processor and memory loads, we introduced some controlled noise to the system by running on the devices a subset of the widely-used Java Dacapo benchmarks [12].

As evaluation criteria, the main metrics were temperature measurements and anomaly detection, processor and memory usage (in %), and network throughput (in Mbps). These metrics were collected with the following tools: wireshark (for MQTT packets) and Logentries (for anomalies), htop and psutils (for processor and memory usage), iperf and the GNU/Linux ftp client (for network throughput). Monitored events (i.e., detected anomalies) were retrieved from LogEntries' reports, which was fed by the NodeRED node. Temperature readings below or above 2σ of the mean were considered as anomalies and flagged as warnings.

B. Results

1) Experiment 1 – MQTT-based IoT application: The MQTT protocol was used to send messages from fake temperature sensors running on the Raspberry Pis to an MQTT broker (server). MQTT runs on the TCP protocol thus it is reliable at the transport layer. We have used the QoS setting of 0 for MQTT which means that messages are send in best effort mode at the application layer. Fig. 4 shows the temperature measured and sent in the real IoT system over time. Fig. 5 shows the temperature measured and sent in the emulated IoT system. For this experiment, each run fired 1800 messages (lasting around 30 mn when firing 1 message per second), and 10 runs were performed per testbed type. Min, max and average values are shown on the figures. We can observe that the received values are similar and not affected by the type of testbed used. We have observed that for each type no messages were lost and no messages were delivered out of order. This means that the server has not lost any received message. Only the fact that random values produced by a normal distribution were used as inputs (temperature values) explains the variations on the figures. We haven't shown the results produced by the virtualized testbed as they are similar to both figures already shown.

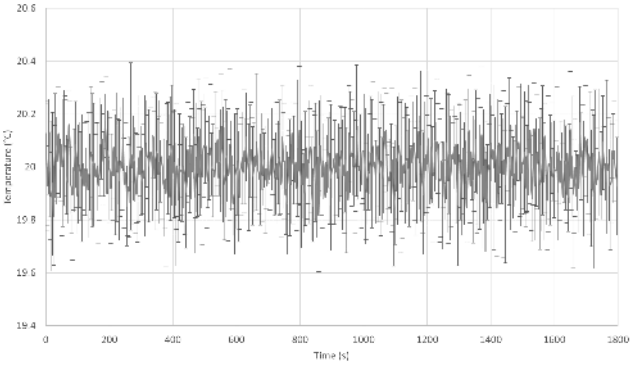

Fig. 4. Temperatures reported by the Physical Raspberry Pi Network

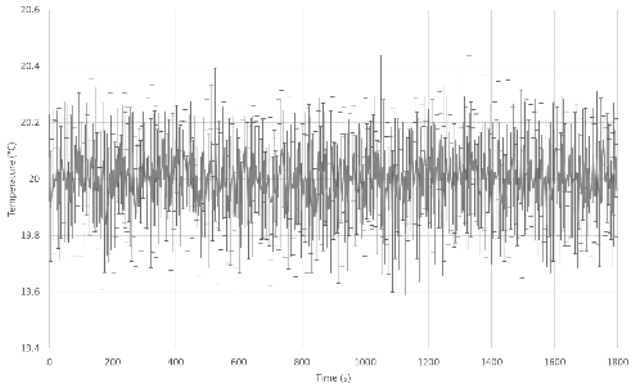

Fig. 5. Temperatures reported by the Emulated Raspberry Pi Network

We have also observed the total duration for firing the 1800 messages. We have tested with various message rates from 0.1 to 100 per second. We have collected those values in Table I. We notice that the total time for sending those messages are matching the frequency parameter for the real and virtualized testbeds. The virtualized testbed is a bit faster because hardware accelerated x86 virtual machines are faster than real RPis. The emulated testbed performs similarly at low message frequencies but shows an increased lag when the frequency increases. At 100 messages per second, the emulated testbed is 86% slower than the real one. This issue must be taken into account when using the emulated testbed and depends on the IoT application used. In most cases, 1 message per second would be considered high, but in some cases, monitoring may require up to 100 messages per second, maybe more.

To understand the behaviour of the systems across the evaluated experimental configurations, our analysis focused on comparing the number of anomalies detected per environment as shown in Table II. It can be noticed how the number of detected anomalies over half-an-hour (at 1 message/s) exhibited a comparable behaviour regardless of the type of

TABLE I
AVERAGE DURATION FOR SENDING 1800 MQTT MESSAGES

Testbed	Real	Emulated	Virtualized
1 message/10s	5:00:09.742	5:00:13.297	5:00:08.840
Δ *vs* Real		+0.02%	-0.005%
1 message/5s	2:30:05.894	2:30:08.066	2:30:04.777
Δ *vs* Real		+0.024%	-0.012%
1 message/s	30:02.772	30:17.073	30:01.663
Δ *vs* Real		+0.8%	-0.1%
5 messages/s	06:01.706	06:17.801	06:00.822
Δ *vs* Real		+4.4%	-0.3%
10 messages/s	03:01.972	03:17.902	03:00.733
Δ *vs* Real		+8.7%	-0.7%
50 messages/s	00:37.279	00:53.978	00:36.700
Δ *vs* Real		+44%	-1.5%
100 messages/s	00:19.278	00:35.982	00:18.701
Δ *vs* Real		+86%	-3%

test environment. Results for the real network are more spread out as shown by the higher standard deviation values, due to the use of real network hardware. An example of the observed anomalies (i.e., temperature thresholds) at Logentries is depicted in Fig. 3.

TABLE II
AVERAGE NUMBER OF ANOMALIES REPORTED

Testbed	Low Temp Anomalies Average (Std-Dev)	High Temp Anomalies Average (Std-Dev)
Real	85.4 (12.8)	86.9 (12.5)
Emulated	85.1 (6.3)	82.7 (8.6)
Virtualized	82.2 (7.8)	83.7 (7.5)

2) Experiment 2 – processor and memory intensive application: A second set of experiments was performed to assess the behaviour of our solution in more resource-intensive scenarios, where one or more resources, such as CPU or memory, would frequently reach 100% usage. Even though such scenarios might not be the most commonly found in IoT (as the data gathering and processing usually tend to be light-weight in terms of CPU and memory), the aim was to strengthen the validation of our solution by identifying either other potentially applicable usage scenarios or limitations on its usage. The experimental set-up was similar to that used in the previous experiment, with the following difference: instead of using MQTT, the tested devices ran a subset of the widely-used Java Dacapo Benchmarks [12]. This strategy allowed us to diversify the assessed behaviours (i.e., CPU and memory usage) by introducing some variability on the workloads processed by the system. The chosen benchmark programs were luindex, sunflow and eclipse, which used three considerably different levels of resource utilization. Each test run lasted 10 minutes and was repeated 10 times. All the plots show for each timed data point (one per second), its average, min and max values over the 10 runs.

Obtained results are illustrated in Figures 6, 7, and 8. They show how an experimental configuration behaved on each of the three tested types of environments. Intuitively,

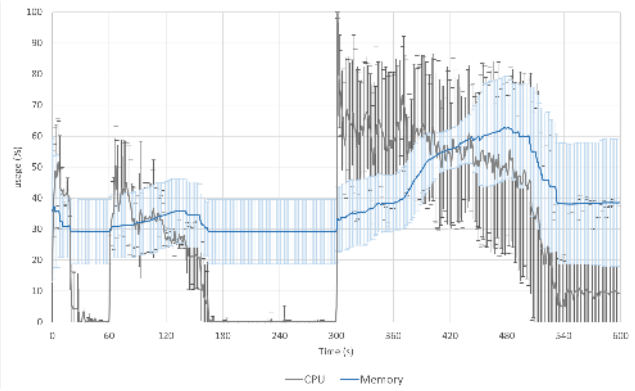

Fig. 6. Test Load of the Physical Raspberry Pi Network

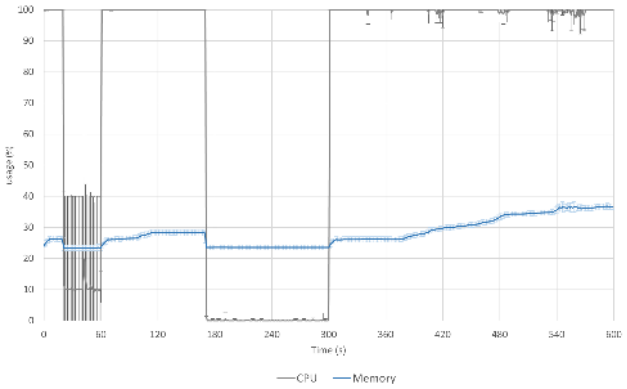

Fig. 7. Test Load of the Emulated Raspberry Pi Network

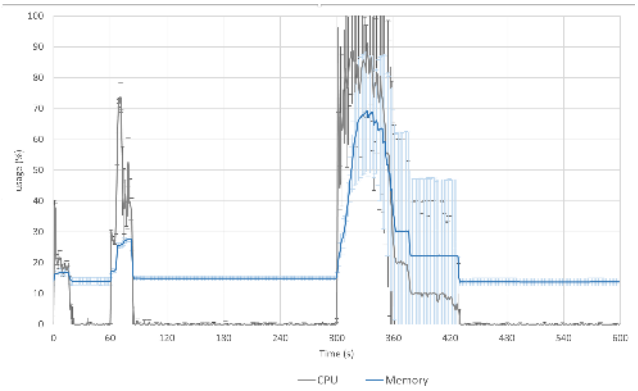

Fig. 8. Test Load of the Virtualized x86-64 Network

if the figures depicted similar shapes among them (like in the previous experiment), the environments would exhibit a comparable behaviour. However, it can be easily seen, by visually inspecting the figures, that it is not the case. Figure 6 depicts the real Raspberry Pi, hence, this exemplifies a baseline behaviour. Meanwhile, Figure 7 and 8 depict the emulated and virtualized Raspberry Pi networks, respectively. It can be noticed how the trends in Figure 8 (i.e. the three peaks caused by the execution of the chosen benchmarks) resemble the ones of the real Raspberry Pi, although they are less sharp.

Furthermore, it can be noticed how the trends depicted on 7 considerably differentiate against the other two figures. In the case of the CPU, its usage practically reaches 100% during the three peaks, situation which indirectly influenced the memory usage (which showed a relatively flat behaviour). This is due to the software-only emulation of the ARM processor which is very demanding for the host environment.

3) Experiment 3 – network intensive application: The last experiment focused on measuring network throughput between the devices in each testbed type and also between different types (e.g., between a physical Raspberry Pi (RPi) and an emulated RPi). These results are summarized in Table III, depicting the throughput obtained for each combination of the three tested environment types. Considering that the first line in the table (i.e., RPi $\rightarrow$ RPi) is the performance baseline, it can be noticed how, whenever an emulated RPi is involved, there is a drastic reduction in throughput (e.g., of approximately 75% in the case of the FTP traffic). This is because in these scenarios, the relatively slowness of the emulated RPi makes it to become a bottleneck in the system (hence provoking the observed impact on throughput). In contrast, it is worth noticing that, even in such saturated scenario, a virtualized x86-64 works fine as a replacement of a real Raspberry Pi. This is reflected in the fact that all combinations using a virtualized x86 achieved levels of throughput comparable to the corresponding baseline value. The only exception was the experimental configuration which only used virtualized environments. This was the result of using hardware acceleration and not capping the NEMU links (i.e., letting them reach maximum throughput), which made the system achieve a throughput higher than what would be actually possible with the real hardware.

TABLE III
NETWORK THROUGHPUTS IN MBPS BETWEEN DEVICES

Link	iPerf	FTP
RPi $\rightarrow$ RPi	95.1	80.04
RPi $\rightarrow$ Emulated RPi	24.1	19.76
Emulated RPi $\rightarrow$ RPi	20.4	16.32
Emulated RPi $\rightarrow$ Emulated RPi	20.2	20.16
RPi $\rightarrow$ Virtualized x86	94.0	89.6
Virtualized x86 $\rightarrow$ RPi	95.6	88.32
Virtualized x86 $\rightarrow$ Virtualized x86	280	277.6

V. CONCLUSION

Performing proper testing in IoT is very challenging. In particular, creating a real testing environment is difficult and expensive. This is because a large amount of human effort and investment in hardware are typically required to create such environments. To address this problem, this paper proposed a novel method to create a realistic emulated IoT testing environment, based on NEMU (a state-of-the-art network emulator). The aim is to increase IoT practitioners' productivity by facilitating the creation of suitable IoT testing environments in which IoT advancements can be easily tested during their research and development cycles. That is, before their level of maturity justify the costs involved on testing in a real IoT environment.

Our experimental results have shown how such emulated environment closely resembles a real IoT environment, as both types of environments produced similar results when used for IoT applications. However, results also showed that processing and memory intensive applications shall not be evaluated by emulation as the performances are still widely different from a real system. Practitioners should therefore check that the IoT application under test is not stressing the devices in terms of processing, memory or network bandwidth, otherwise results obtained on the emulated testbed will be incorrect.

These experiments were done within a log analysis context, where the aim was to monitor and detect anomalies in the IoT environments in real-time. The overall results showed that the emulated IoT system worked well, as the behaviours of the detected anomalies (across all the tested experimental configurations) were similar between the two environments, as they achieved comparable numbers of anomalies.

Future work will focus on strengthening the experimental validation of our approach. For instance, by diversifying the characteristics of the emulated environment (e.g., topologies, number of nodes, link characteristics). Additionally, we plan to investigate ways to improve the detection (and potentially the alleviation and/or elimination) of the anomalies discovered in the IoT system.

ACKNOWLEDGMENT

Supported, in part, by Science Foundation Ireland grant 13/RC/2094. Supported, in part, by Agence Nationale de la Recherche grant ANR-10-IDEX-03-02.

REFERENCES

[1] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," Cisco, Tech. Rep.
[2] R. H. Weber, "Internet of things new security and privacy challenges," *Computer Law & Security Review*, vol. 26, no. 1, pp. 23–30, 2010.
[3] V. Autefage and D. Magoni, "Nemu: A distributed testbed for the virtualization of dynamic, fixed and mobile networks," *Computer Communications*, vol. 80, pp. 33–44, 2016.
[4] P. M. d. Silva, J. Dias, and M. Ricardo, "Cidrarchy: Cidr-based ns-3 routing protocol for large scale network simulation," *ICSTT*, pp. 267–272, 2015.
[5] Y. D. A. Y.-J. Vilen Looga, Zhonghong Ou, "Mammoth: A massive-scale emulation platform for internet of things," *ICCCIS*, 2012.
[6] B. Bagula and Z. Erasmus, "Iot emulation with cooja," in *Workshop on Scientific Applications for the Internet of Things*, ICTP, Ed.
[7] Q. Le-Trung, "Towards an iot network testbed emulated over openstack cloud infrastructure," in *2017 International Conference on Recent Advances in Signal Processing, Telecommunications Computing (SigTelCom)*, Jan 2017, pp. 246–251.
[8] S. M. M. P. Jayavardhana Gubbi, Rajkumar Buyya, "Internet of things (iot): A vision, architectural elements, and future directions," *FGCS*, vol. 29, pp. 1645–1660, 2013.
[9] U. W. Bishnu Kumar Maharjan and R. Zandian, "Tree network based on bluetooth 4.0 for wireless sensor network applications," *EEDER*, 2014.
[10] S.-k. K. Joonkyo Kim and J. Park, "Bluetooth-based tree topology network for wireless industrial applications," *ICCAS*, 2015.
[11] B. Siniarski, P. A. Perry, C. Olariu, J. Murphy, and T. Parsons, "Real-time monitoring of sdn networks using non-invasive cloud-based logging platforms," *PIMRC*, 2016.
[12] A. O. Portillo-Dominguez, P. Perry, D. Magoni, M. Wang, and J. Murphy, "Trini: an adaptive load balancing strategy based on garbage collection for clustered java systems," *Software: Pract. and Exp.*, 2016.