Evaluation on UiTiOt Container-Based Emulation Testbed

Chuong Dang-Le-Bao, Nhan Ly-Trong, and Quan Le-Trung(✉)

Department of Computer Networks, University of Information Technology,
Vietnam National University Ho Chi Minh City, Ho Chi Minh City, Vietnam
{chuongdlb, nhanlt, quanlt}@uit.edu.vn

Abstract. In this paper, we present a container-based emulation testbed, namely UiTiOt. The testbed integrated a well-known wireless emulation tool called QOMET to imitate the wireless network models over established wired network. Our testbed was developed based on a state-of-the-art technology called container-based virtualization. With our proposed design, we aim to provide researcher with the capability of running large-scale wireless/IoT experiments at affordable cost. Therefore, we did an insightful evaluation to ensure the feasibility and accuracy of the implementation of UiTiOt testbed. The evaluation includes several test-cases with different network topologies and routing protocols.

Keywords: Network architectures · Network evaluation
Container-based emulation · Network testbed

1 Introduction

As the trend of Internet of Things (IoT) continues to thrive in recent years, where the number of connected devices will reach 20.8 billions in 2020 [1] and leads to rapid movement in development of modern network protocols and standards. It is necessary to test the new device functionality in both hardware and software aspects before mass production. However, there has been a challenge to setup hundred-node or thousand-node in real-world scenarios because one has to consider a number of factors such as geographical position, mobility, transmit power and the budget to afford real devices. Since the simulation only solved the large-scale experiment in theoretical model where many environmental variables would be assumed or randomly set like node mobility, communication range or propagation loss, and in addition it was unable to execute in real-time like real-world testbed as illustrated in Table 1 [2].

Our work [3] has proposed a wireless/IoT Emulation Testbed based on container-based virtualization and QOMET, a wireless network emulation set of tools that made the wired-network adaptable for wireless network experiment. In our testbed system, each node in experimental scenario is represented by a container, which a simple Linux-runtime process that encapsulate user-namespace and relevant binaries and libraries to form a thinner Operating System-like environment. By implementation, we proved the proposed design's feasibility and usability, but we have yet to perform any further evaluation on its performance. Hence in this paper, **we focus on evaluating the performance of our emulation testbed, namely UiTiOt through many test-cases**

Table 1. Comparison of wireless testing techniques

	Simulation	Real world	Emulation
Real-time execution	No	Yes	Usually
Control level	High	Low	High
Condition range	Large	Small	Large
Result realism	Low	High	Medium
Experimentation cost	Low	High	Medium
Ease of use	High	Low	Medium

with different set of parameters. Instead of using OpenStack as underlying infrastructure [3], we setup small scale setup by using VMware virtual machines running on a high-performance laptop. By doing this, we possibly remove any underlying network constraints which had occurred in previous work [3], where traffic flows between containers would be routed to unwanted path despite our configuration. Our paper is organized as follows: the first section is the introduction, Sect. 2 discuss related research in emulation testbed and container-based virtualization, Sect. 3 revisits the architecture of our testbed design, Sect. 4 shows the experimental results of performance evaluation and followed by a conclusion.

2 Related Work

Testbed especially the emulation testbed systems has been playing an important role in development and evaluation of wireless protocols and application. Recent research has shown the advantages of emulation testbed, for instance EmuStack [4] which based on OpenStack and Docker showed the efforts of building a distributed emulation testbed for Delay Tolerant Network (DTN) at a reasonable cost, but no large-scale evaluation has yet been addressed. In addition, QOMB [2] which derived from a general-purpose wireless network test bench StarBED [5], which is large-scale testbed infrastructure comprised of physical computer nodes and network hardware. QOMB and related research and emulation experiments on StartBED system has proven the its efficiency and usability in the field of wireless network experiment but still required a large budget to develop and operate. In addition, recent work of [6] has proposed an emulation testbed based on OpenStack Cloud where an experiment node substituted by a virtual machine instance. On each *experiment* node of QOMET [7], the emulation process would be executed on-the-fly to mimic the wireless behavioral communication over the top of wired network setup. In recent year, container-based virtualization has embarked most aspects of software production in general. Though, many have utilized the advantages of container technology to apply in their research, most used it as an alternative for VM to deploy large-scale network experiment such as reproducing popular network experiments while others used container-based virtualization for Information-Centric Networking (ICN) [8] testbed or even in high-performance computing [9] and ubiquitous computing. Recent research [10] has shown the impressive performance of container technology in cloud context which outperform virtual machines and nearly the same as native performance.

3 Architectural Design

Our testbed system was created by integrating a well-known wireless emulator QOMET into a virtual container, also known as container-based virtualization technique [11]. In a brief, container-based virtualization is a lightweight alternative to the hypervisors, also known as Operating System (OS) Level virtualization. In container-based virtualization, physical resources of one machine are divided into multiple isolated user-space instances called containers. This can be achieved by using a feature of the Linux kernel called kernel namespaces, which allow different processes to have different view on the system, network-namespace feature can also be used to enable network virtualization. The container-based virtualization is basically a piece of software that works at the OS-level, providing abstractions directly for the guest processes. Since it works at OS-level, all virtual containers share the same Linux kernel with the host operating system, hence will have weaker isolation in comparison with hypervisor-based virtualization. However, each container behaves exactly like a stand-alone OS from the user's perspective. In our system, we used Docker [12], the state-of-art software that extended LXC [13] (Linux container – the most popular implementation of virtual container).

3.1 Qomet

QOMET [7] is short for Quality Observation and Mobility Experiment Tools, which is a set of tools used for wireless network emulation. It has been developed by researchers at JAIST since 2006. Our system used the version of QOMET (v2.1), released in June 2013 (Fig. 1).

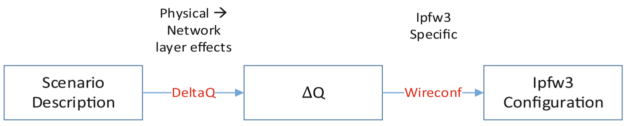

Fig. 1. QOMET model for the emulation.

The emulation mechanism of QOMET consists of two main stages. Firstly, a scenario description was defined by user to represent the wireless communication conditions and topology, that are then used to generate the experimental nodes and computing the network degradation values called *deltaQ*. Secondly, those degradation values were applied to the actual traffic by *wireconf* module (Fig. 2), which is a wrapper for a link-level emulator called DummyNet [14]. In this case, the degradation computation step is very simple, but in order to execute the degradation on real-time traffic of complex topologies was quite a challenge. Our main work focus on creating a Linux run-time environment for QOMET to be deployed and executed the artificial wireless network over wired/virtual network, and virtual container is inevitably the most ideal candidate for creating the QOMET run-time environment.

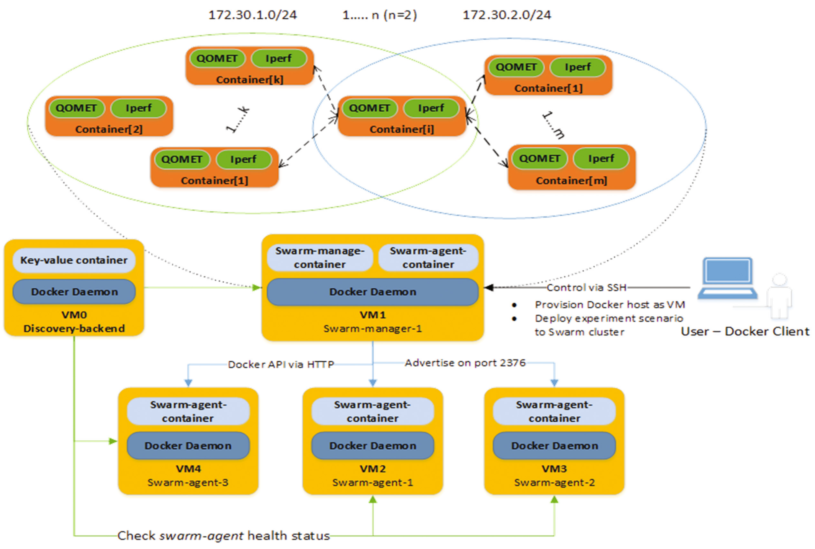

Fig. 2. Architecture of UiTiOt testbed using container-based virtualization.

3.2 UiTiOt Testbed

It is important to mention that the underlying infrastructure for the container-based virtualization is not necessarily the physical machine, a virtual machine can be used to deploy the container runtime software like LXC, Docker. As shown in Fig. 2, we used virtual machines to provide an OS-based computing instances for the upper container-based virtualization. In Fig. 2, a group of virtual machines (VMs) have been created to deploy a Docker-provided technology called Docker Swarm, which enable Docker-running VMs to share their computing resource (e.g. CPUs, memory, NIC) to form a single pool of compute resources that can be interacted transparently from the user point of view. The most notable feature of this distributed containers model is *overlay* networks, which are networks spanning on top of other networks [15]. In other words, one container from one VM will be able to communication with other containers from VMs that running Docker Swarm agents, VM0 in this model was dedicated to be a discovery backend service that stores the identities of all VMs in the Docker swarm setup, this help the VM running Swarm manager discover and check the availability of Swarm agents it manages. In our model, the overlay networks spanned on top of lower virtual networks of virtual machines, which otherwise can be physical networks if we had used physical computers to setup Docker Swarm. In this model, each *experiment node* described in QOMET scenario file is substituted by a container. At first, user had to defined the his experimental scenario in QOMET XML-file, which was then automatically parsed into Docker-compose YML-file. The file contained description about number of containers (each represent an *experiment node)*, number of overlay networks, environment variables inside each container, etc. It important to mention that QOMET executable binaries (e.g. *deltaQ, wireconf, etc.*) into the *container image*, which is a versioning image defined which software should be installed and run inside the container. In Docker engine, those container image configuration is defined a single file called *Dockerfile*. As all relevant file were put into container including a set of routing software like Quagga [16], OLSRd[1]. The *deltaQ* value was computed on-the-fly in the progress of building the image. At the end of *Dockerfile,* we defined an entry point, which points to the application will execute when the container is created and hence we setup multiple start-up applications at once using a custom script. Finally, the container image packaged with all necessary files for specific experiment was distributed across all VMs via a cloud-based container image hosting *Docker Hub*[2]. Finally, the virtual wired-network environment had been established for emulating environment, the system now creates a number of containers and overlay networks according to Docker-compose YML-based file. The container distribution was managed by the Docker Swarm manage and agent to spread the containers across VMs which currently have available resources. We can also manually define the and container distribution and container-consumed resources (CPU, memory, etc.) by specific configuration inside Docker-compose YML-based file[3]. All relevant logs of the experiment will be automatically pushed to a pre-defined central file server.

4 Experimental Results

In this section, we evaluated the performance and accuracy of our distributed container-based testbed through several test-cases. The Docker Swarm setup was hosted on VMware Fusion hyper-visor running inside a laptop, which is a MacBook Pro model 2015 equipped with Core i7 (2.2 GHz) and 16 GB of memory. The virtual machines setup was quite similar to the design, which it has four VM(s): one for discovery service was assign 1 vCPU/ 1 GB RAM, one as a Swarm manager with 2 vCPU/ 4 GB RAM, two as Swarm agents in which each has 1 vCPU/3 GB RAM.

In our experiment, we deploy 32-node scenario, in which these nodes form a tree topology network as illustration in Fig. 3. Besides, two routing schemes were used to enable multi-hop communication among *experiment nodes*, the first routing mechanism was RIP which enable by using Quagga routing suite, the second one was Optimized Link State Routing which implemented by using OLSR daemon (OLSRd). The communications environment was defined in QOMET test-cases using a list of parameters shown in the Table 2. α, σ, W respectively specify the parameters attenuation constant, shadowing parameter, wall attenuation of the log-distance path loss propagation modeltransmit power specifies the power of the power in dBm. The two main testing cases were interference and non-interference where pre-defined conditions shown in Table 2 will artificially affect the link quality between nodes, hence the wireless network communication will be emulated on top of the wired network by execution of *wireconf* module in each node. In the third test-case, the execution of *wireconf* module of QOMET was disabled to capture the normal network performance

[1] *OLSRd:* http://www.olsr.org/mediawiki/index.php/Olsr_Daemon

[2] *Docker Hub:* http://hub.docker.com/

[3] https://docs.docker.com/compose/compose-file/

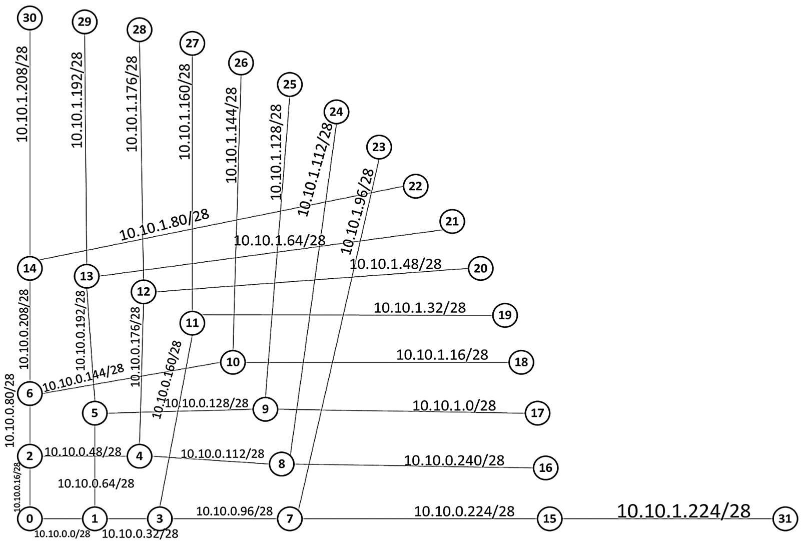

Fig. 3. Tree topology consist of 32 nodes and multiple subnets

Table 2. QOMET XML-file configuration for two main testing cases

Test-case	With interference	Without interference
Transmit power	2.5	
α	3.2	2.0
σ	0	
W	0	
Noise power	−100	0
Standard	802.11a	

without emulation. The popular network performance measurement tool Iperf [17] was used to generate network traffic and capture the bandwidth pattern as well as loss-rate. In detail, the test duration last for 200 s, but the actual duration when *iperf* client really communicate with *iperf* server is approximately 190 s.

We combined 3 mentioned test-cases with two types of routing protocol to create six different testing cases. Two intended connections were chosen to illustrate the emulation effects on **wired** network are: the first connection is from *node15* to *node31* where the transmission occurred directly between two nodes; and the second connection is from *node0* to *node31*, the multi-hop connection is be the longest branch of the tree structure. The measured bandwidth of these connections are shown respectively in following figures Figs. 4 and 5.

In Fig. 4 we continue to discuss the measured bandwidth of a direct connection from *node15* to *node31*. It is clear from the graph that the figures for 6 cases fluctuated in a small range during the period except the scenario without interference using OLSR performed a wide range fluctuation at the beginning of the period, which was till

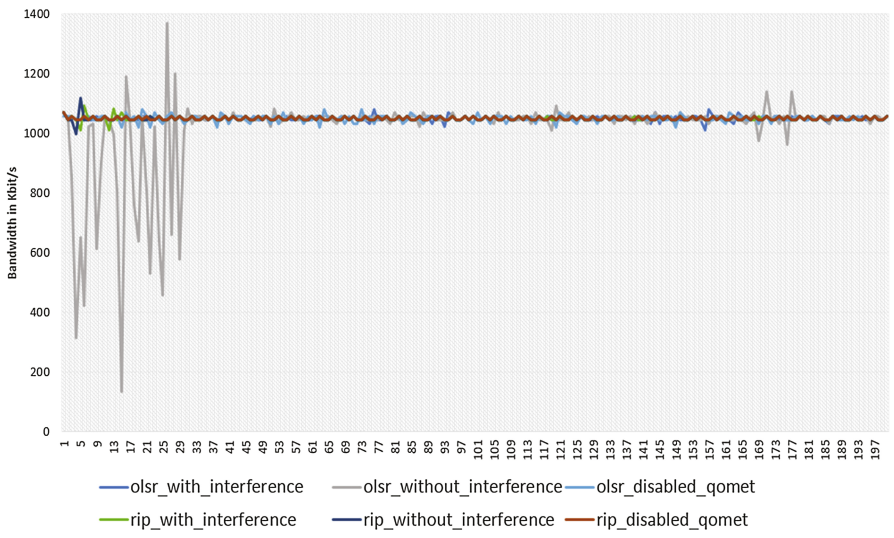

Fig. 4. Bandwidth measured of connection from *node15 to node31*

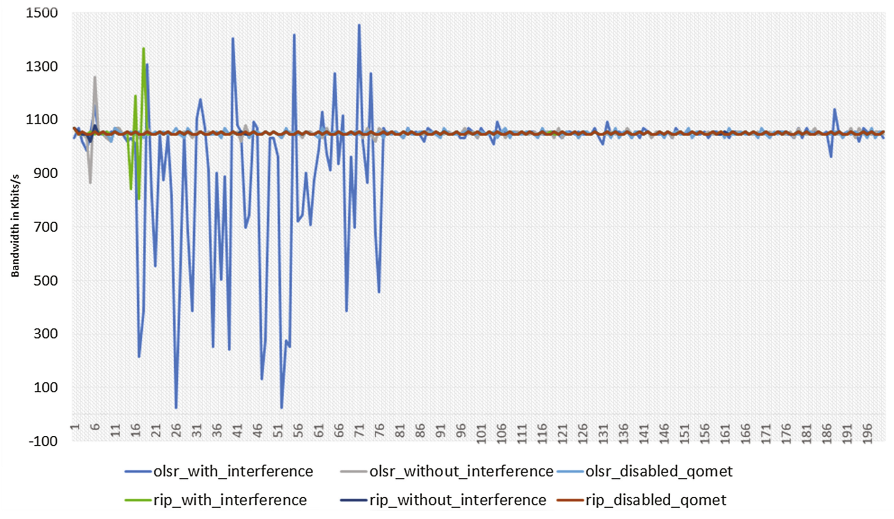

Fig. 5. Bandwidth measured in 6 test-cases from *node0 to node31*

unexplanable at the moment. The average bandwidth in the experiment without interference using OLSR was reported to approximately 1015 kbps. Meanwhile, the figures for the other ones were nearly the same as each other at just under 1050 kbps.

As is shown by the above line chart, the measured bandwidth in both 2 cases which disabled QOMET was seen to be more stable than the other ones. The figures were fluctuated in a tenuous range from around 1020 to almost 1080 kbps. Meanwhile, the 2 test-cases with interference and the one without interference using RIP performed a fluctuation in the bandwidth of the connection from node 15 to node 31 in a wider range. They were changed from under 1000 to just above 1110 kbps. In contrast, the bandwidth in the experiment without interference using OLSR fluctuated in a wide range at the beginning of the period. The minimum and maximum figures for this scenario were reported to approximately 130 and over 1300 kbps respectively.

In general, the proportions of packet loss in the experiment with interference were always at a higher level than the other ones. They were followed by the figures for the experiment without interference and the one that disabled QOME. In Figs. 6 and 7, the connection from node 15 to node 31 which was a direct connection, had the lowest percent of packet loss among 6 connections. The loss-rate for this connection in the experiment with interference was just under 8%, which was almost 3 times higher than the figure for the one without interference. Similarly, only approximately 1% packets of this connection which were unable to reach the destination in the experiment that disabled QOMET. Regarding the figures for indirect connections which were connections consisting intermediary nodes, the measured loss-rates for the scenario with interference were at least 18% and up to approximately 24% whereas the corresponding figures for the one without interference oscillated between 15 and just above 17%. In the scenario that disabled QOMET, the connections were reported with the lowest loss-rates among 3 test-cases which were from around 7.5% to 15%. Similar to Figs. 6 and 7 depicts the loss-rates of 3 test-cases but using OLSR instead of RIP. It is clear from the graph that the experiments using OLSR were reported with a considerably lower loss-rates than the ones using RIP. In most of connections, the loss-rates among 3 experiments followed a descending order from the experiment with interference then without interference to the experiment that disabled QOMET. In contrast, a different order of loss-rates among the 3 test-cases was shown in 2 connections from node 16 to

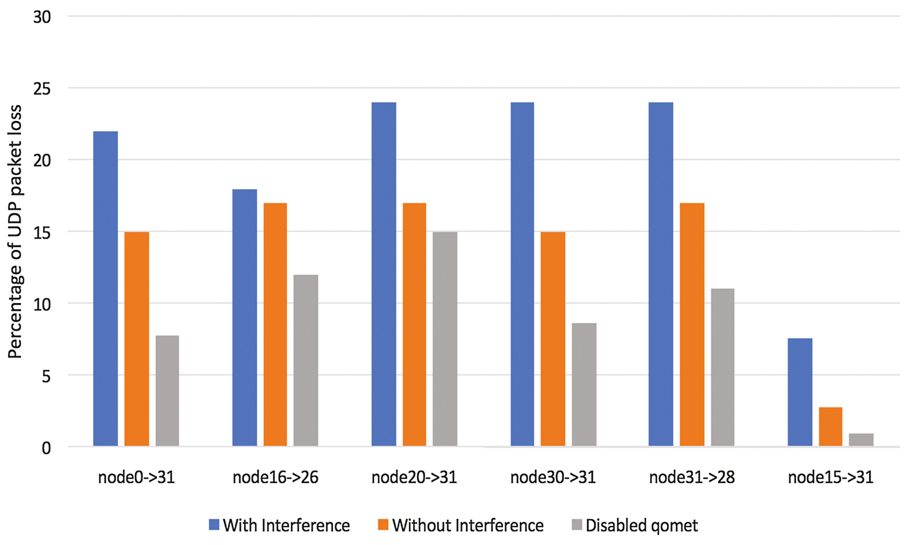

Fig. 6. Lossrates of test-cases using RIPd (Quagga)

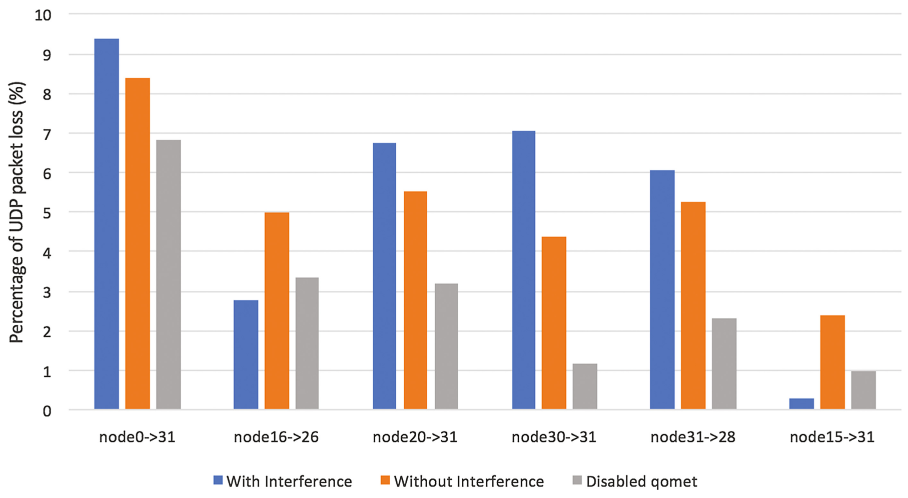

Fig. 7. Lossrates of test-cases using OLSRd

node 26 and from node 15 to node 31. The lowest figure among 3 test-cases was reported to the experiment with interference at about 0.25% for the connection from node 15 to node 31 and under 3% for the one between node 16 and node 26. At around 2.3% and exactly 5% packet sloss respectively for these 2 connections, the scenario without interference was the leader among these 3 test-cases. Meanwhile, the loss-rates for the experiment that disabled QOMET were shown at 1% for the connection between node 15 and node 31 and approximately 3.3% for the one from node 16 to node 26.

5 Conclusion

In a conclusion, we have evaluated the UiTiOt Testbed throughout six test-cases in a complex tree-based topology. The loss-rate results shown that the system work perfectly as our expectation. However, the measured bandwidths showed some abnormal patterns, in which non-interference case have a wide range of fluctuation than the interference case, and we have yet addressed any root causes of this problem. We believe our architecture design have a great potential in large-scale experiment in term of emulation accuracy, automatic deployment and reasonable cost. Though, there are still many to work need to be done to improve the testbed: deploying and extending the underlying infrastructure to more physical or virtual machines to support more *experiment nodes*, and we also aim to support more latest wireless/IoT routing protocol like ZigBee, 802.11n, etc., in the near future. Finally, we do hope that this system will potentially contribute to the development of domestic and international IoTs research and innovation.

Acknowledgement. This research is funded and supported by Department of Science & Technology Ho Chi Minh City, under the contract number: 48/2015/HĐ-SKHCN.

References

1. Gartner.com, 10 November 2015. http://www.gartner.com/newsroom/id/3165317. Accessed 13 April 2017
2. Beuran, R., Nguyen, L.T., Miyachi, T., Nakata, J., Chinen, K., Tan, Y., Shinoda, Y.: QOMB: a wireless network emulation testbed. In: Global Telecommunications Conference, GLOBECOM 2009. IEEE (2009)
3. Chuong, D.L.B., Trong, N.L., Le-Trung, Q.: UiTiOt: a container-based network emulation testbed. In: Proceedings of the 2017 International Conference on Machine Learning and Soft Computing (2017)
4. Li, H., Zhou, H., Zhang, H., Feng, B.: EmuStack: an openstack-based DTN network emulation platform. In: 2016 International Conference on Networking and Network Applications (2016)
5. Miyachi, T., Chinen, K., Shinoda, Y.: StarBED and SpringOS: large-scale general purpose network testbed and supporting software. In: Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools (2006)
6. Le-Trung, Q.: Towards an IoT network testbed emulated over OpenStack cloud infrastructure. In: Recent Advances in Signal Processing, Telecommunications and Computing (SigTelCom) (2017)
7. Razvan, B., Nakata, J., Okada, T., Tan, Y., Lan Tien, N., Yoichi, S.: A multi-purpose wireless network emulator: Qomet. In: 22nd International Conference on Advanced Information Networking and Applications-Workshops, AINAW 2008 (2008)
8. Asaeda, H., Li, R., Choi, N.: Container-based unified testbed for information-centric networking. IEEE Netw. **28**(6), 60–66 (2014)
9. Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., De Rose, C.A.: Performance evaluation of container-based virtualization for high performance computing environments (2013)
10. Seo, K.T., Hwang, H.S., Moon, I.-Y., Kwon, O.-Y., Kim, B.-J.: Performance comparison analysis of linux container and virtual machine for building cloud. Adv. Sci. Technol. Lett. **66**, 105–111 (2014)
11. Soltesz, S., Potzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. ACM SIGOPS Oper. Syst. Rev. **41**(3), 275–287 (2007)
12. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**, 2 (2014)
13. Bernstein, D.: Containers and cloud: From LXC to docker to Kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)
14. Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols. ACM SIGCOMM Comput. Commun. Rev. **27**(1), 31–41 (1997)
15. Chowdhury, N.M.K., Boutaba, R.: A survey of network virtualization. Comput. Netw. **54**(5), 862–876 (2010)
16. Jakma, P., Lamparter, D.: Introduction to the quagga routing suite. IEEE Netw. **28**(2), 42–48 (2014)
17. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: iPerf - The ultimate speed test tool for TCP, UDP and SCTP (2005). https://iperf.fr. Accessed 25 Feb 2017