2015 29th International Conference on Advanced Information Networking and Applications Workshops

DOCKEMU - A Network Emulation Tool

Marco Antonio To, Marcos Cano and Preng Biba
RLICT
Universidad Galileo
7a. Av. Final, calle Dr. Eduardo Suger Cofiño, Zona 10
Guatemala, Guatemala.
{marcoto, marcos.cano, prengsen}@galileo.edu

Abstract

Dockemu is developed because of the need for a well designed tool for emulating networks. The tool utilizes technologies that are tailored to a current researcher's needs, delivering a robust and dynamic framework. In the past, most emulation tools have tried to provide solutions without taking into consideration that the installation and configuration of the software can be very time consuming, not to mention, the complexity of setting up and running an actual experiment can become also a very complex task. Our approach provides the researcher the flexibility to rapidly create networks (wired or wireless), up to date operating systems and a user friendly method of installation and configuration, that translates into a streamlined workflow for the emulation of experiments. Also, the development of applications or prototypes can be made directly into a real world OS. This saves time, because the prototypes will only be developed once (not for a simulator first and then for a final OS) but the results will also be more accurate. Dockemu utilizes virtualization with Linux Containers through Docker and Linux Bridging along with NS-3 for the emulation of layers 1 and 2 of the OSI model.

***Index Terms*—Emulation, linux containers, network simulator 3, ad hoc networks, virtualization, tap bridge, Docker.**

I. Introduction

The research community is in constant need of new tools in order to ensure better and faster deployments for their experiments. These experiments usually are validated with network simulators, like the Network Simulator 2 (NS-2)[1], the Network Simulator 3, (NS-3)[2], OMNET++[3], among others. Although many protocols have been developed and implemented in simulators, there is still a need in the research community to take those experiments and generate results as *accurate* as possible. As outlined in [1], network simulators do not provide the sufficient accuracy, but are great for a first step in protocol testing. Also, the work of [2] and [3] provide some insightful information about the current state of Mobile Ad hoc Network (MANET) simulation and how we must reconsider the validity of the results obtained through simulators.

Network Emulation provides the means to an experiment to be as close to a real environment, but without the need of actually setting up real networks. In a broader sense, the experiments and scenarios tested through the network emulators should comply with the following requirements [4]:

- *Realism.* The framework should provide the most real conditions available, minimizing the error between the simulated and the real, leading better and more valuable results.
- *Reproducibility.* The scenarios must be reproducible so they can be compared among them. Reproducibility does not mean that the results of a network emulator should always be the same. We must take under consideration that the closer we are to reality, another variables can introduce changes in the experiments.
- *Representativeness.* The framework should provide the desired environmental factors in order for the results to be valid.

The *Dockemu* tool bases its design in the previous requirements, but complies to the emulation benefits mentioned at Puzar et al. [5], which are usage, low costs, scalability, portability, routing and comparability. Furthermore, Dockemu will extend the benefits with the following items:

- The tool is able to handle real world OS and applications.
- The tool is IPv4 and IPv6 capable.
- The tool can emulate wired and wireless networks.
- The tool has a smooth installation and configuration.
- The tool provides a fast scenario deployment.

The main contributions of this paper are:

- An emulation tool that meets current and future needs of the research community utilizing Linux Containers with the Docker framework and Network Simulator 3 (NS-3).
- The Dockemu tool provides a fast and user friendly workflow, from the installation to the configuration of scenarios, providing the accuracy and flexibility of a real world environment.
- A tool that is portable and scalable.

This paper is organized as follows. Section II presents the State of the Art in this area, getting a closer look at past implementations of network simulators and emulators. Section III details the Dockemu tool and its architecture. Section IV explains the implementation of the tool. Section V shows some scenarios and their results and finally Section VI will give our Conclusions and Future Work.

[1]NS-2: http://nsnam.isi.edu/nsnam/
[2]NS-3: http://www.nsnam.org/
[3]OMNET++: http://www.omnetpp.org/

978-1-4799-1775-4/15 $31.00 © 2015 IEEE
DOI 10.1109/WAINA.2015.107

II. Related Work

The research community has many choices at their reach to perform their experiments. Unfortunately not all software satisfies the requirements mentioned before, leading to partial accuracy on the results. There are various tools for simulation or emulation, from open source (NS-2, NS-3, OMNET++) to commercial software (QualNet[4], OPNET[5]), each one with special characteristics and specialities. The Dockemu tool will be distributed to the research community as one more option for emulation of wired or wireless networks, which will follow the guidelines of OpenSource Software community[6]. Due to the benefits that Dockemu will provide the users, were have high expectations that it will be welcomed.

Among other works that have been done, Grajzer et al. [6], do a good job highlighting the current problematic of simulators and emulators. Their work states that (in most cases) simulation is used for testing performance and scalability. As of emulation is used for the final stages of verification of the solution. Grajzer et al. also suggest that given that emulation is "closer" to a real world environment, it would be better not to use it only at the end for validation, but it would save time and effort to use this tools from the beginning, leading to more accurate results. Also, they do a very nice job on stating their experience in implementing IPv6 capable Mobile Ad hoc Network (MANET) routing protocols in simulators and how this time consuming job has to be duplicated later for real world OSs. One of the motivations of creating Dockemu, was in fact the same experience of encountering many out of date MANET routing protocols, or protocols that exist only in simulators that never made it to a real world OS implementation. Moreover, works done by Andel et al. [3] and Hiranandani et al. [2] in the case of MANETs simulation, discuss some of the flaws of tens of dozens of research papers where the simulation results fail to provide the sufficient information so their experiments can be independently repeatable or have tested their experiments with scenarios with default configurations, which do not come close to real life mobility patterns (Example: "Random Waypoint Mobility").

Taking the previous under consideration, many research teams have also created new tools that take network simulation one step further and head towards network emulation. Their goal is to fulfil most of the requirements for simplicity, accuracy, scalability and performance. To name a few of these tools are MobiREAL [7], NEMAN [5], TapRouter [8], JiST [9], Mesh Linux Containers (MLC) [10], among others.

MobiREAL and JiST are speciality simulators, being only used for a specific area of expertise. These two softwares bring specific characteristics for the simulation of wireless ad hoc networks and their scalability. JiST in particular can scale up to millions of nodes. As we have seen, this simulators test the dynamics of new protocols in isolation from external factors, doing their best in introducing interference, speed, etc.

Moving to emulation frameworks, NEMAN, TapRouter and MLC provide similar characteristics of our proposed tool Dockemu, but with slight differences. TapRouter is deprecated because their implementation can now be done directly between Linux Bridges and Tap devices, without the need of a middle layer. MLC uses Linux Containers for node virtualization, but "simulates" the network through the use of a Linux Application called Traffic Control (TC). Even tough, TC can add delay, jitter, bandwidth shaping, among others, it does not provide support for more complex scenarios and mobility patterns. NEMAN is a very good implementation of Linux Containers along with a Simulator, in this case NS-2. NEMAN is very close of what we are trying to achieve but delivers an inherited complexity from NS-2 and its OTCL with C++ implementation. NS-2 can be very time consuming when trying to add or modify existing protocols. Moreover, it is not a modular simulator and fails to scale due to high consumption of computing resources [1].
The NEMAN paper [5] presents a property comparison table between existing emulators. In Table I we have added Dockemu to it.

TABLE I
Properties of Various Network Emulators

	MobiEmu	EMWIN	Mobinet	NEMAN	DOCKEMU
Usage	X			X	X
Low Cost			X	X	X
Scalability		X	X	X	X
Portability	X	X	X	X	X
Routing			X	X	X
Comparability			X	X	X

Table I shows that the Dockemu tool matches one by one the NEMAN tool but has more characteristics that are not mentioned in Table I. For example, NEMAN uses NS-2, a more complex development environment. Dockemu uses NS-3 and is capable of providing not only environments for wireless ad hoc networks, but also is capable of emulating regular ethernet connectivity, making Dockemu a more comprehensive tool for emulating network scenarios. Also, Dockemu (to the best of our knowledge) is the only emulation tool to incorporate the Docker framework for linux container management. As we will describe in the rest of this work, Dockemu requirements are broader, making it a tool capable of satisfying current and future needs of the scientific community.

III. Dockemu Framework

Dockemu is based on current technologies that are paving the way for future generations, which already have proven to be successful in optimizing computing resources as of generating research scenarios.

[4]Qualnet: http://web.scalable-networks.com/content/qualnet
[5]OPNET: http://www.riverbed.com/
[6]OpenSource Website: http://opensource.org/

A. Preliminaries

The Dockemu tool is designed to get scenarios up and running as fast as possible, spending the least time on installing, configuring and getting to know the software. The fact that today, a good part of the research community first develops for a specific simulator and later they have to develop it again for a real world implementation (many times using different Programming Languages) raises the question of how efficient is this method. Dockemu provides the user the benefit of developing applications on current OSs which can be tested on larger networks, leading to more accurate results and saving time in duplicity of the implementation.

In order to satisfy the requirements mentioned before, we use the following technologies and glue them together for a final research tool:

- Virtualization through Linux Containers using Docker.
- Emulation through Linux Network Bridges and Network Simulator 3 (NS-3).
- Dockemu framework which binds the previous together.

Linux Containers provide the *userspace* necessary for the applications to run. Linux Network Bridges and NS-3 provide the emulation needed for wired or wireless hosts to communicate with a chosen layer 2 technology, for example, 802.3 Ethernet for wired scenarios or 802.11 CSMA/CA for wireless scenarios, and finally our implementation which creates the scenarios utilizing the previous two components.

B. Linux Containers

Linux Containers is a form of server-level virtualization which gives the flexibility of "partitioning" the Host Operating System into independent smaller systems. These smaller systems are called *Containers*. Containers provide resource isolation and control for a specific application or system [11]. This technology brings to the researcher a great deal of benefits, including portability, better usage of host resources, allowing to generate hundreds to thousands of nodes (depending on the host hardware), compared to regular OS virtualization technologies like Virtual Box (vBox), VMWare, Citrix, among others.

The fact that Linux Containers can scale experiments to a larger set of nodes, is by itself one of the most important characteristics on why other frameworks ([8], [10], [12]) use them also.

Figure 1 shows the architecture for traditional OS Virtualization on which the Hypervisor sits on top of the host OS. It's important to notice that each Guest OS' resources have to be provisioned beforehand. As contrary to Linux Containers, each container runs only the application needed, not a full OS. Figure 2 shows the architecture of Linux Containers.

C. Docker framework

The Docker Framework *"...is an open platform for developers and sysadmins to build, ship, and run distributed applications."* [13]. Docker builds on the capabilities of LXC and "supercharges" the benefits of Linux Containers. Docker benefits over LXC are (the most important):

- Portable. Docker can make any Linux container portable so it can be migrated between host machines.
- Versioning. Docker enables a history of the changes in a particular Linux container.
- Shared Libraries. The Docker community is uploading to a central repository the containers everyone has made already, leading to less time in implementation.

As Figure 3 shows, Docker takes the place of LXC, but instead of replacing it completely, it uses LXC and converts it to a more powerful, flexible and easier framework for managing Linux Containers.

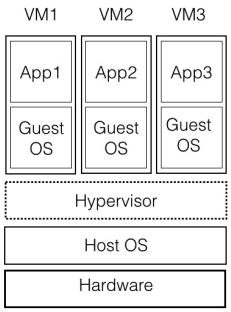

Fig. 1. Traditional OS Virtualization

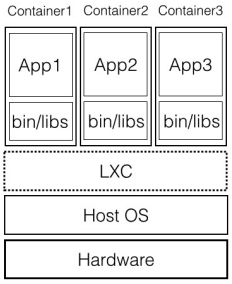

Fig. 2. Linux Container Virtualization

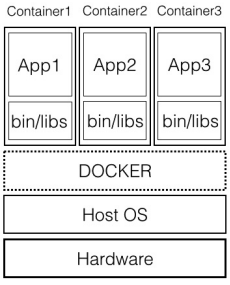

Fig. 3. Linux Container Virtualization

D. Linux Network Bridges

Linux Network Bridges provides a component which behaves like a virtual network switch. Meaning, that it can

connect two or more interfaces together. Moreover, these interfaces can either be real interfaces inside the OS (Ex. eth0) or can be virtual interfaces (Ex. tap).

Typing the following command in a Linux OS using a Shell or Command Line Interface (CLI) shows the current active bridges in the system and which interfaces are connected to them:

```
root@ubuntu:~# brctl show
```

In this first example, the bridge *lxbr0* has associated the interface *eth0* only.

```
bridge name     bridge id          interfaces
lxcbr0          8000.000c297bbed4  eth0
```

The output for the second example, the bridge *lxbr0* has associated two interfaces *eth0* and *veth032*. This means that a real interface like eth0 can communicate with a virtual interface veth032 through a virtual bridge lxcbr0.

```
bridge name     bridge id          interfaces
lxcbr0          8000.000c297bbed4  eth0
                                   veth032
```

E. Network Simulator 3

The Network Simulator 3 or *NS-3 "...is a discrete-event network simulator, targeted primarily for research and educational use. ns-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use"* [14].

NS-3 has many advantages compared to other simulators, being one of them the use of C++ for development of experiments which makes a perfect companion with another OpenSource projects. For example, NS-2 (which is still widely used), has the drawback of having a complex environment to develop new experiments, due to the fact that uses a mix of OTcl with C++.

The role of NS-3 in Dockemu is to provide the means for network scenario generation, which is also the final piece of the Dockemu framework. The *Tap Bridge Model* allows the interface of a node inside NS-3 to be mapped directly into a Tap device (also called interface) in the Host OS.

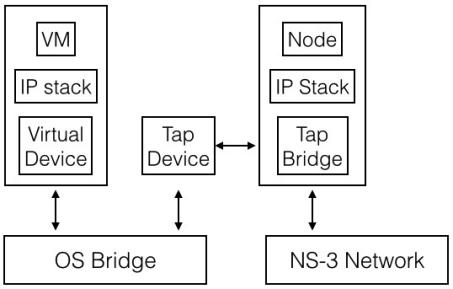

Fig. 4. Tap Bridge Architecture - Simplified Model

Figure 4 shows the flow of information from/to a Virtual Machine (in this case a Linux Container) to/from a node inside the NS-3 simulator.

F. Dockemu Architecture

The Dockemu tool merges the technologies that we mentioned before into a software capable of satisfying the requirements presented at the beginning of this paper. Dockemu has the ability of creating scenarios and then emulating them depending on the need of the experiment. From wired to wireless networks, wireless ad hoc networks and MANETs, which can be further tested by the advantages of using real OSs. For example, most MANET routing protocols are implemented for simulators (NS-2, NS-3, OMNET++, etc.) but most fail to provide an implementation that can handle IPv6. Moreover, there are only a few wireless ad hoc routing protocol implementations in real OSs that are constantly maintained by their authors.

As showed in Figure 5, the Dockemu tool will deploy the corresponding Linux Containers, Linux Bridges, Tap Bridges, Virtual Interfaces (veth) and the needed NS-3 script, in order for the experiment to work properly. Once the experiment is deployed, the user can "attach" to any container and perform metric tests with common and familiar tools, like tcp-dump, ping, ping6, Iperf, etc. If needed, the user can create its customized Container Template, which can later be replicated to the number of nodes needed. As we can see, Dockemu is very powerful because of its flexibility and endless options that can be added directly to the containers.

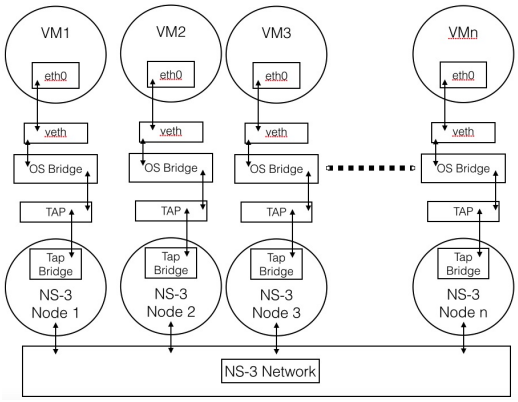

Fig. 5. Dockemu Framework - Top Level View

IV. Implementation

The Dockemu tool was implemented using the Python high-level programming language and bash scripting. The tool is invoked using the *dockemu* command in a CLI environment. The Dockemu gets configured through a text file called *dockemu.conf* which provides all the variables needed to define the scenarios. It is out of scope of this paper to name all of the variables available for the Dockemu tool, were a tutorial

will be available at the URL of the Research Lab[7], but some examples are:

- *layer2*. Defines what kind of layer 2 technology will be used. Ex. Ethernet, CSMA 802.11, among others.
- *number.networks*. The number of simultaneous networks.
- *number.nodes*. The number of nodes per network.
- *ipv4.network*. Defines the IPv4 network.
- *ipv4.ipaddr.range*. Defines the range of IPv4 addresses.
- *ipv4.mask*. Defines the mask for the IPv4 address range.
- *ipv6.network.prefix*. Defines the IPv6 network.
- *ipv6.ipaddr.range*. Defines the range of IPv6 addresses.
- *ipv6.ipaddr.mask*. Defines the mask for the IPv6 address range.
- *adhoc.routing.proto*. Defines the which ad hoc routing protocol the wireless network will use. By default Dockemu comes with OLSR and BMX6.
- *NS-3.adhoc.mobility.pattern*. Defines the node movement inside NS-3.
- *NS-3.time.exp*. Defines for how long the NS-3 scenario should run.

Once the configuration file is set up with the variables needed, the user will type:

```
root@ubuntu:~# ./dockemu -f dockemu.conf
```

The Dockemu tool will take care of the rest, meaning it will configure, create and start the number of "nodes" described in the configuration file. Then, it will create the corresponding networking layer (linux bridges, tap interfaces, virtual interfaces and so on), finishing with the creation of the layers 1 & 2 through NS-3. The experiment will run for the time specified in the configuration file or until it is manually stopped with the keyboard combination *Crtl+C*.

It is important to state at this point, that in order for the Dockemu tool to comply with being easy to install, the team is providing to its users a preconfigured Virtual Machine, based on Ubuntu 14.04 LTS. This VM can be downloaded and added easily to a vBox o VMWare framework, so the user can be up and running as fast as possible.

V. Tests and Results

We have tested the Dockemu tool with various scenarios, which will be available at the Dockemu repository. In this work we will present two different scenarios, one wired and another wireless.

A. Scenario 1

In this scenario Dockemu was set up to provide a simple Ethernet network within the same broadcast domain, with the following characteristics:

- Network IPv4 = 172.17.0.0/24.
- Network IPv6 = 2800:1a0::/64.
- Number of nodes 50.
- Type of layer 2 = Ethernet.
- Use NS-3 = no.

It is important to notice that because we wanted only and Ethernet network, all nodes were configured in the same Linux Bridge, so no NS-3 is necessary. Once the experiment is running we can manage the containers with the docker framework commands. For example, if we would like to show which containers are running, we would type:

```
root@ubuntu:~# docker ns -a
```

This will list all the containers currently running. Then, to get into a container and start measuring network metrics, we would type:

```
root@ubuntu:~# docker attach <nameofcontainer>
```

Once inside the container we can do a throughput test using a well know tool called *iperf*.

```
root@d049153612bc:/# iperf -c 172.17.0.7
------------------------------------------------------------
Client connecting to 172.17.0.7, TCP port 5001
TCP window size:  340 KByte (default)
------------------------------------------------------------
[  3] local 172.17.0.6 port 38465 connected
      with 172.17.0.7 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  27.1 GBytes  23.3 Gbits/sec
```

The results give a very high throughput due to the fact that it is not going through any physical interfaces, just virtual interfaces inside the same OS.

B. Scenario 2

In this scenario Dockemu was set up to provide a wireless CSMA network, with the following characteristics:

- Network IPv4 = 10.0.0.0/24.
- Network IPv6 = 2800:1a0::/64.
- Number of nodes 20.
- Type of layer 2 = CSMA.
- Use NS-3 = yes.

Given the fact that we need to simulate a wireless CSMA network, Dockemu passes through the task to the NS-3 simulator and then Dockemu binds the containers with the nodes inside NS-3. Contrary to Scenario 1, this scenario puts each container in a different Linux Bridge an attaches only two interfaces to each bridge, the virtual interface of the container (veth) and the tap bridge interface which is the link to the NS-3 node. In other words, this scenario creates 20 linux containers, 20 bridges and 20 nodes inside NS-3.

As of the previous scenario, we can attach to a given container and then perform a regular ping between nodes. In this case, we will attach to *node2*:

```
root@ubuntu:~# docker attach node2
```

First we check the connectivity with another node using the *ping* command:

```
root@3ce162a28e95:/# ping 10.0.0.2 -c 5 -vvv
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.534 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.523 ms
```

[7]RLICT Website: http://rlict.galileo.edu/

```
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.462 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.855 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.531 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received,
0% packet loss, time 8999ms
rtt min/avg/max/mdev = 0.462/0.549/0.855/0.109 ms
```

We check the throughput using also *iperf*. This is the output from the Client side:

```
root@794f0113e606:/# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 37166 connected
      with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  86.2 MBytes  71.7 Mbits/sec
```

And the output from the Server side:

```
root@3ce162a28e95:/# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.1 port 5001 connected
      with 10.0.0.2 port 37166
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.2 sec  86.2 MBytes  71.2 Mbits/sec
```

The previous results show that even though we are still using containers in the same computer and OS, the NS-3 simulator gives the characteristic of a CSMA network, limiting the throughput to 71 Mbps approximately.

VI. Conclusion and Future Work

In this paper, we have presented the Dockemu tool for emulation of wired and wireless networks. The tool glues together state of the art technologies of virtualization (Docker and Linux Containers), Linux Bridging and NS-3 in order to provide a complete solution for the research needs of today and tomorrow. Dockemu is limited only by the hardware it is running, but with contemporary PC's processing power, it is capable of creating complex scenarios. Moreover, we have given the user the choice of downloading a VM, so the time of installation and configuration of the tool itself is reduced dramatically, being in a position to start running experiments in a very short time. Also, to the best of our knowledge, Dockemu is the only emulation tool that utilizes the Docker framework for the creation of Linux Containers.

The Dockemu tool turns out to be an efficient and accurate tool for the emulation of wired and wireless networks. Moreover, the potential experiments that can be deployed by the tool are endless because of the flexible nature of it. Furthermore, we have satisfied our initial requirements, proving that our tool can emulate accurately different kind of scenarios.

Although Dockemu has a very strong first version, there is always room for improvement. Our perspective is to keep on developing and refining the tool in various ways. Dockemu will have to consolidate and keep up to date with the Docker framework for the creation of templates for experiments, after all, the emulation of applications and protocols depend on the implementation on Linux Containers. Another improvement, is the switch to a faster form of development, along with being in sync with the Docker framework. This is expected to result in the adoption of the Go programming language. Also, the introduction of a GUI would definitely have a cordial welcome by the research community. Finally we have set up the server architecture to deliver the Dockemu Tool. In the future, if the adoption of the tool is welcomed by the research community, we can move forward to a Cloud Based Dockemu Tool, in order to take advantage of the elastic features of Cloud Computing. This would add the benefits of scaling up to a number of nodes which could only be achieved by huge computing resources. Also, this will remove the installation tasks by the end user and a researcher's benefit of being able to access it over any device.

References

[1] L. Hogie, P. Bouvry, and F. Guinand, "An overview of manets simulation," *Electron. Notes Theor. Comput. Sci.*, vol. 150, no. 1, pp. 81–101, Mar. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2005.12.025

[2] D. Hiranandani, K. Obraczka, and J. Garcia-Luna-Aceves, "Manet protocol simulations considered harmful: the case for benchmarking," *Wireless Communications, IEEE*, vol. 20, no. 4, pp. 82–90, August 2013.

[3] T. Andel and A. Yasinsac, "On the credibility of manet simulations," *Computer*, vol. 39, no. 7, pp. 48–54, July 2006.

[4] *Improvement of IP-based MANET Emulation*. Proceedings of the International Military Communication Conference, 2009.

[5] M. Pužar and T. Plagemann, "Neman: a network emulator for mobile ad-hoc networks," in *Telecommunications, 2005. ConTEL 2005. Proceedings of the 8th International Conference on*, vol. 1, June 2005, pp. 155–161.

[6] M. Grajzer and M. Glabowski, "On ipv6 experimentation in wireless ad hoc networks," *Journal of Telecommunications and Information Technology*, 2014.

[7] K. Konishi, K. Maeda, K. Sato, A. Yamasaki, H. Yamaguchi, K. Yasumoto, and T. Higashinoz, "Mobireal simulator-evaluating manet applications in real environments," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium on*, Sept 2005, pp. 499–502.

[8] J. Zhang and Z. Qin, "Taprouter: An emulating framework to run real applications on simulated mobile ad hoc network," in *Proceedings of the 44th Annual Simulation Symposium*, ser. ANSS '11. San Diego, CA, USA: Society for Computer Simulation International, 2011, pp. 39–46. [Online]. Available: http://dl.acm.org/citation.cfm?id=2048370.2048376

[9] R. Barr, Z. J. Haas, and R. van Renesse, "Jist: An efficient approach to simulation using virtual machines: Research articles," *Softw. Pract. Exper.*, vol. 35, no. 6, pp. 539–576, May 2005. [Online]. Available: http://dx.doi.org/10.1002/spe.v35:6

[10] A. Neumann, E. Lopez, and L. Navarro, "An evaluation of bmx6 for community wireless networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, Oct 2012, pp. 651–658.

[11] Ubuntu-Documentation-Team, "Ubuntu linux containers man page," http://manpages.ubuntu.com/, 2015.

[12] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 275–287, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1272998.1273025

[13] Docker-Staff, "Docker official website," https://www.docker.com/whatisdocker/, 2015.

[14] NS-3-Team, "Network simulator 3 website," http://www.nsnam.org, 2015.