Dockemu: Extension of a Scalable Network Simulation Framework based on Docker and NS3 to Cover IoT Scenarios

Antón Román Portabales[1] and Martín López Nores[2]
[1]*Quobis Networks, O Porriño, Spain*
[2]*Department of Telematics Engineering, Universidade de Vigo, Vigo, Spain*
anton.roman@quobis.com, mlnores@det.uvigo.es

Keywords: ns-3 Network Simulation Docker Real-time Containers.

Abstract: The purpose of this project was to extend an existing open-source simulation framework called Dockemu in order to make it suitable to perform IoT simulations. The work covered some improvements with goes from the support of more network technologies to the use of setup and deployment tools used by modern devops professionals. The paper explains the architecture, the newly-added features and the specific advantages it offers for research works in IoT network simulations.

1 INTRODUCTION

The Dockemu project was initially described by Marco Antonio To, Marcos Cano and Preng Biba in (To et al., 2015). It proposes a simulation approach where the nodes can run any client/server Linux software which is executed in a Linux container and the network is simulated used using the open source network simulator ns-3[1].

The Dockemu implementation used to add the extensions described in this work was published by J.A. Álvarez Aldana in Github[2]. The main features of Dockemu simulation framework are as follows:

- It allows to use real software in the simulated nodes, removing the extra effort needed to create ad-hoc simulation software and ensuring more accurate results.
- It is based on open-source projects and the project itself is open-source, so it can be used and extended by anyone at no cost and with total access to the code which is considered to be more effective and cost-efficient for research works. (G Gupta et al., 2013).
- It has a very quick set-up time through a completely automated procedure, minimizing the time the researcher needs to invest in tasks not directly related with the experiments at hands.
- The tools available for the exchange of Docker containers make it very easy to replicate experiments, thus facilitating the validation and analysis of the results included in any publication.

Owing to these facts, Dockemu fosters greater realism, reproducibility and representativeness in network simulations. It supports the most relevant network technologies, and more are expected to be added in the future.

The goal of the extension described in this paper is to enable simulations of different IoT scenarios that can not be covered with the current version of Dockemu. Those scenarios include more complex topologies, including different types of application (e.g. client-server) in the same simulation, and uses of network technologies typically found in IoT networks, namely LTE and 6lowpan over IEEE 802.15.4. This work describes the extensions and the rationale behind the additions of new features needed to conduct experiments that aim to benchmark IoT protocols and measure the impact of the modification of different network and protocol-specific parameters.[3] The key technologies used to implement Dockemu are Linux containers and ns-3.

Both Linux containers are ns-3 are briefly described in Section 2. Then a short state-of-the-art section discusses the existing alternatives to perform IoT simulations. In Section 4 we cover the new features

[1]https://www.nsnam.org/

[2]https://github.com/chepeftw/NS3DockerEmulator

[3]All the changes implemented in Dockemu will be released under GPLv3 license, including the experiments reported in the paper. This tries to follow good practices (Patrick Vandewalle and Vetterli, 2009) in terms of research impact and reproducibility.

added to the existing Dockemu framework and in Section 5 we describe the main implementation details. The paper ends in Section 6 with conclusions and future lines to further enhance Dockemu.

2 TECHNOLOGY BACKGROUND

Container-based virtualization and application containerization is an Operating System (OS) level virtualization method for deploying and executing distributed applications without launching a different VM for each application. The application executed in the containers uses the same lower layers of the OS like the rest of applications, but in an isolated runtime environment. Linux containers take advantage of *namespace* and *cgroups* resource isolation features provided by the Linux kernel.

Docker takes advantage of the aforementioned features to provide a complete framework to create and execute Linux containers. Docker enables the execution of a huge number of isolated containers in a single server. This is the main reason why we considered containers to be the best approach to run the real software to be tested in our simulations, since IoT scenarios normally require a big number of network endpoints. Using Virtual Machines would require much more resources, by far (Sharma et al., 2016). The overhead of virtual machines is even more noticeable in IoT, since applications in this realm have to run in constrained devices, so they are expected to be light in terms of CPU and RAM usage.

Apart from the low use of resources of containers, the use of Docker also facilitates the creation of *images* which contain all the required software to be tested. These can be created from scratch, or derived from existing container images used during the development phase of the project to be automatically tested by CI (*Continuous Integration*) systems or by the developer herself. Currently, there are Docker images available to be used as development environments for the most common IoT operating systems, such as Contiki OS[4] and FreeRTOS[5]. The same Docker container which is used to develop and test IoT applications can be directly connected to Dockemu.

The other key technology of Dockemu, ns-3, is a discrete event simulator (DES) designed for the simulation of data networks of different technologies and topologies. It is targeted primarily for research and educational use and open-source software, licensed under the GNU GPLv2 license, therefore publicly available for research, development and use. There are other open-source network simulators, but ns-3 has been considered one of the best in terms of performance of use of CPU and RAM (ur Rehman Khan et al., 2013). A key feature of ns-3 is a real-time scheduler for simulation events which locks the simulation clock with the hardware clock. This real-time feature and the ability to generate real network packets which its respective checksums enable the integration of ns-3 with real network stacks which and emit/consume packets.

Furthermore, ns-3 enjoys a great diversity of modules to simulate different parts of data networks. Many modules allow to simulate different network technologies implementing realistic models. There are also modules not related with the networking itself but with other characteristics of the simulation such as the mobility model, which allows to simulate mobile nodes deployed in a specific area. One of these modules is the Tap NetDevice[6], which enables the connection of a OS-level tap interface with a node of the ns-3 simulation.

In section 5 we will see how both the real-time feature and Tap bridge module are key characteristics of ns-3 used by Dockemu to be able to run simulations.

3 STATE OF THE ART

Simulation of IoT networks has been widely discussed by the research community which has described a number of requirements every simulation and experimentation platform should meet in order to provide realistic and valuable results. (Gluhak et al., 2011) gathers a comprehensive list of those requirements:

- **Scale:** IoT simulations would normally involve a very big number of nodes;
- **Heterogeneity:** different network technologies, topologies, type of devices and applications must be supported;
- **Repeatability:** experiments must be easily repeatable with the same results;
- **Federation:** testing platforms should be able to federate to each other in order to simulate bigger systems.
- **Concurrency:** several experiments should be able to be performed at the same time;

[4]https://github.com/contiki-ng/contiki-ng/wiki/Docker
[5]https://github.com/megakilo/FreeRTOS-Sim
[6]https://www.nsnam.org/doxygen/grouptap-bridge.html

- **Mobility:** the platform must be able to simulate mobile nodes as this is going to be normal behavior of many IoT devices;
- **User Involvement and Impact:** experiments which involves humans should be able to evaluate its social impact and acceptance.

Though those requirements were initially considered for experimentation testbeds many of them can be applied directly or adapted to pure of hybrid[7] IoT simulations.

Regarding the simulation technologies to be used, DESs have been identified as valid tools in order to simulate the network, but in order to take advantage of multi-CPU and distributed networks (D'Angelo et al., 2016) it is necessary to implement Parallel and Distributed Simulation (PADS)[8]. The authors of (D'Angelo et al., 2016) also proposed the use of multilevel simulations, where some parts of the system are simulated in more detail and other parts are simulated at a higher level in order to save computational resources.

There are several proprietary simulators, such as SimpleIoTSimulator [9], that claim to be able to simulate thousands of sensors implementing different IoT protocols. No technical details about the simulation techniques used are disclosed. Rather, the sensors are pre-built and the code is proprietary, so it is not really usable in typical research environments. Another commercial alternative is Netsim[10], a generic network emulator that can emulate a network of different technologies and connect real applications to it, however it does not handle the setup of the complete scenario. It allows to modify its source code to implement customized protocols to licensed users.

ns-3 has been already use in other simulation frameworks such as Cupcarbon[11]. This is a open-source IoT and smart city simulator that allows to simulate a network of sensors deployed in a city with geographic models[12]. It is largely focused on smart cities and sensor networks, and it does not support the simulation of customized applications.

Node-RED[13] is an open-source flow-based programming tool which is part of the JS Foundation[14]. Flow-based programming is a way of describing the behavior of an application as a network of black boxes. Each black box has a well-defined purpose and exchanges data with other connected boxes. This high-level, functional style of specifications allows the system behavior to emerge from simplified model, but it does not deal with neither implementation nor low-level networking models.

Finally, the most extreme approach in terms of simulation is to use a very large scale open testbed with thousands of real wireless sensors. This is the approach followed by FIT/IoT-Lab[15] which consists of thousands of real IoT wireless sensors spread across six different sites in France. The IoT-LAB is open to research experiments under request.

4 NEW FEATURES ADDED TO SUPPORT IOT SIMULATIONS

This section gathers all the new features and technologies we have added to the Dockemu framework within the scope of this work in order to improve its suitability for advanced IoT simulations and be able to simulate a wider range of scenarios.

4.1 LTE and 6lowpan Over 802.15.4

IoT emulation requires to support a huge number of nodes as there would be in a real environment and also network topologies using adapted technologies as LTE and 6lowpan over 802.15.4. We included support for both in Dockemu by leveraging the LENA, lr-wpan and 6lowpan modules of ns-3. The initial version Dockemu only supported CSMA (model equivalent to Ethernet in terms of simulation) and WiFi (802.11), which are not particularly relevant to emulate IoT scenarios.

Both LTE and 6lowpan over 802.15.4 topologies support mobility, so scenarios where the nodes are not static are supported taking advantage of ns-3 mobility model library[16]. This feature could be especially interesting to simulate connected car scenarios. The type of simulations covered by both technologies is different. LTE can cover wide areas, leveraging the operator infrastructure and relatively expensive devices with SIM card and LTE wireless module which can send and receive high bandwidth traffic. In turn, 6lowpan over 802.15.4 is used in small areas to receive and send information to low power constrained devices. Actually a combination of two modules – which is still not possible in Dockemu– would make

[7] We use the term *hybrid* to refer to simulations where some of the elements is not simulated but a real element, like the client/server software in Dockemu.

[8] ns-3 supports distributed simulations as explained in 6.1.

[9] http://www.smplsft.com/SimpleIoTSimulator.html

[10] https://www.tetcos.com/

[11] http://www.cupcarbon.com/

[12] https://github.com/bounceur/CupCarbon

[13] https://nodered.org/

[14] https://js.foundation/

[15] https://www.iot-lab.info/

[16] https://www.nsnam.org/docs/models/html/mobility.html

sense to cover even more realistic scenarios where a PAN coordinator receives data from remote sensor connected to small devices and this information is sent to a cloud service using LTE.

4.2 Different Types of Nodes

The initial version of Dockemu only considered a single type of node. We have added the capacity to define different types of nodes (e.g. scenarios where one node is a server and the rest of nodes act as clients). This allows, for example, to simulate for IoT protocols such as COAP(Bormann and Shelby, 2016) and MQTT[17] where the nodes can play different roles. Scenarios where there is more than one server can even be easily supported, too. In order to create the different type of nodes the Docker container must be created with the right software or configuration.

4.3 Ansible as Scenario Deployment Tool

The initial setup as well as the scenario setup is a complex process which involves several software elements and configuration files. Ansible –deployment and orchestration software supported by Red Hat– has become an important tool for devops engineers (Ebert et al., 2016) to deploy services since its release in 2012 and it is being used to automate the setup of complex architectures such as NFV-based platforms. The fact that it is a *de facto* standard tool, together with the ease to add new features using existing and well-tested modules, makes it a suitable choice to manage the orchestration of the Dockemu framework. It also allows to replicate the simulation environment in different Linux distributions so it makes the project more portable. These are the reason why we decided to adopt Ansible in Dockemu as the deployment and scenario setup tool.

5 IMPLEMENTATION

5.1 Configuration File

In order to control the parameters of the simulation we had to extend the parameters included in the reference configuration file. The configuration file is written in YAML format, which makes it easy to understand and modify.

[17] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html

Listing 1: Dockemu reference configuration file.

```
#DOCKEMU config file
general:
  waf_path=/home/user/ns-3.22
  logging=true
  verbose=false
  logfile=logs/.log
  containerLogs=/var/logs/containers
  runningTime=300
  templateEthernet=ns3/tap-ethernet
  templateWiFi=ns3/tap-wifi
  templateLTE=ns3/tap-lte
  template6lowpan=ns3/tap-6lowpan

mobility:
  mobility_pattern=1

nodes:
  numberClientContainers:5
  numberServerContainers:1
  clientDockerfile:ns3/docker/cDockerfile
  serverDockerfile:ns3/docker/sDockerfile

networking:
  topology:ethernet
  ipv4Network:10.6.6.0/24
  ipv6Network:2001:DB8::/32
```

Once the configuration file has been provisioned with the right values it is necessary to execute the Dockemu from a terminal command, passing the configuration file as a parameter:

```
$ sudo dockemu -c <conf file>
```

It is required to execute with *sudo*, since it is necessary to create new bridge and tap interfaces. It is also possible to set all the commands using the CLI and passing all the settings defined in the configuration file directly as parameters to the command.

5.2 Steps to Setup the Simulation

In order to set up the environment and prepare all the elements needed by each simulation, Dockemu carries out the following tasks:

1. Build the container images for client and server from the Dockerfiles specified in the configuration. These Dockerfiles must be edited by the authors/testers of the software to be tested in the simulation.
2. It sets the right ns-3 simulation file and does the required modification in the C++ file if required according to the configuration settings.
3. It starts one container running per client and server nodes defined in the configuration file.
4. It creates all the tap and bridge interfaces required to link each container with the ns-3 simulation.
5. It assigns the IP addresses to be used during the simulation to each node.

6. Once the steps below are completed the simulation starts for the duration specified in the configuration file.

5.3 Networking Setup

The networking setup requires the creation of as many bridges and tap interfaces as containers are in the simulation. The number of containers corresponds with the number of nodes desired in the simulation, and it may change from one simulation to another; so, the creation/deletion of bridge and tap interfaces most be done on demand. The number and type of nodes is passed as a parameter. As shown in Figure 1 the virtual interfaces and the corresponding tap interfaces are connected using a bridge interface and the tap bridge object within the ns-3 topology is associated in turn to its corresponding bridge.

Dockemu takes advantage of the Network namespace feature of the Linux Kernel (Pino, 2018) to assign IPv4 and IPv6 IP addresses to each container.

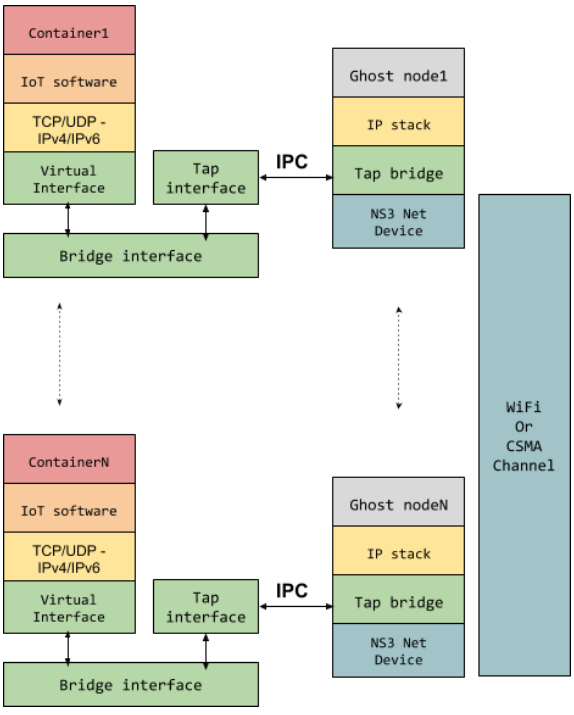

Figure 1: Connectivity between containers and ns-3 ghost nodes for Ethernet (CSMA) and WiFi.

5.3.1 IPv6 Support

In order to support IPv6, an IPv6 address has been added to the Network Space interface created for each container. This way each container will use directly this IP so the application can be tested with a real IPv6 like in a real network. The connection with ns-3 happens at Data Link layer so it is not necessary to handle it in the OS interfaces. Special attention needs to be put on the routing configuration of ns-3 as it has some differences respect to IPv4 routing.

5.4 LTE Support

The only ns-3 networking technologies that support tap bridge are WiFi and CSMA, so a LTE network can not be directly connected to the container through network interfaces. In order to connect the container with the ns-3 simulation, we have followed the approach described and validated in (Hasan et al., 2017). A CSMA link with a huge bandwidth and very low latency is used to connect the tap interface and the UE node in the simulation as depicted in Figure 2. Each container is connected through a bridge with the tap interface installed in a ghost host, connected in turn using the high-bandwidth and very low latency CSMA link. As the CSMA effect is minimized in terms of simulation, it will not add a measurable delay nor will lose any packets. This approach can be followed to simulate more network technologies which can not be connected directly to tap bridge interfaces.

Our implementation allows to connect each container to a UE node, so it simulates the execution of the application in a device with LTE interface. In our initial version all the devices are connected to the same enbNode, but nothing prevents from using several enbNodes in the future, since this is supported by ns-3 LENA module (it has been designed to support from tens to hundreds of eNBs and from hundreds to thousands of UEs, so it is suitable for almost any IoT simulation). The current implementation also allows to create a different role for a server application which may exist or not.

The diagram 2 depicts the general architecture of the system.

5.5 LR-WPAN and 6lowpan Support

Low-rate wireless personal area networks (LR-WPANs) are widely used in IoT deployments as they focus on low-cost, low-speed ubiquitous communication between constrained devices with low-power requirements. IEEE 802.15.4 is a technical standard which defines the physical and MAC (Medium Access Control) layers of LR-WPANs. Other higher-level layers and interoperability sublayers are not defined in the standard. Specifications, such as 6LoWPAN, SNAP, Thread and ZigBee define the layers not covered by IEEE 802.15.4 to implement usable LR-WPANs.

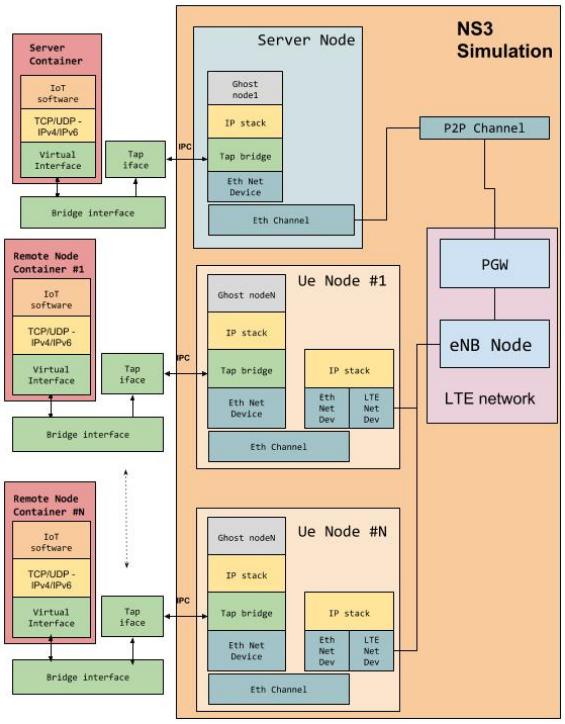

Figure 2: Block diagram of ns-3 configuration to simulate LTE network with *N* UE and a server nodes.

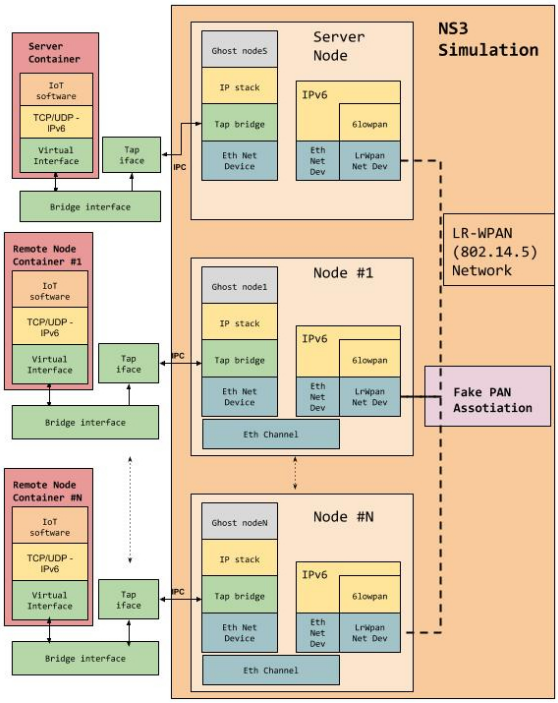

Figure 3: Block diagram of NS3 configuration to simulate LTE network with N UE and a server nodes.

ns-3 has a 6lowpan module, which was used jointly with the lr-wpan module (which implements part of the 802.15.4 standard) in order to simulate a LR-WPAN network. Figure 3 depicts the architecture we use to simulate this type of network. It is pretty similar to the architecture used to simulate the LTE scenario: the ns-3 *tap-bridge is connected to a* ghost node which is connected using an Ethernet[18]. In this architecture the Node container must send the traffic in IPv6 directly. This IPv6 traffic is sent through a very low latency and high bandwidth link to the node which sends the traffic to the LR-WPAN network, after passing through a 6lowpan layer to adapt the packets to the 127-byte maximum size defined by 802.15.4 standard. In order to associate all the nodes to the same network, a fake PAN association is forced in the configuration. In a real network this association would be automatically set up by one of the nodes of the network playing the role of PAN coordinator.

5.6 Docker Setup

In order to execute the software of server and client nodes, Dockemu uses Docker containers. This way a relatively high number of nodes can be simulated at low computational cost (especially considering that the software is expected to have been designed to be executed in constrained devices). For the sake of simplicity and to cover the most common simulation scenarios, the current version allows to deploy one node with the role of server and a configurable number of nodes with the role of client. With this setup we can cover a wide range of IoT simulation scenarios.

It is necessary to build the Docker images which will be instantiated as containers at the beginning of the simulation. In order to create the build, it is necessary to write one *Dockerfile* for the server and another one for the clients. The creation of this *Dockerfile* does not differ from a regular file for other types of containers. It must leave the service running as if it would be in a real device of server. Therefore the simulation will also validate the implementation of the IoT software.

5.7 Software Deployment Automation

Several solutions to automate the deployment of software such Puppet, Chef and Ansible have appeared in the last years. They are intensively used in devops tasks. The initial version of Dockemu was implemented using different Bash scripts to set up the environment (creation of tap and Bridge interfaces

[18]The name of the module used to simulate wired Ethernet networks in ns-3 is called CSMA.

and edition of the C++ ns-3 simulation script) and a Python main script which controls the execution of the simulation.

We have used Ansible to carry out all the environmental tasks and the logic was left in the Python script. Using Ansible instead of Bash scripts has a number of advantages:

- the simulation can be easily deployed in any server, different from the OS where the main script is being executed;
- it is idempotent, and therefore more efficient since it only performs actions when needed reducing setup time;
- it is declarative (not procedural), which added to its smooth learning curve makes the *playbook* files [19] easier to read and modify;
- it will be possible to take advantage of new modules and features added to Ansible.

Once the simulation has finished, Ansible it is also used to gather all the traffic exchanged by all the nodes collected in PCAP files (except the LR-WPAN traffic, which is collected in text files) and also the logs generated by the client and server applications executed in the Docker containers.

6 CONCLUSIONS

Dockemu provides a realistic and easy to setup simulation framework which meets the requirements of a wide range of IoT simulations. It enables the use of real client/server IoT software without any modification by just creating a Docker container following Dockemu guidelines to be included as a node in the simulation.

The fact of using ns-3 to simulate the network means a great advantage since it allows to use all the exiting and future ns-3 modules to increase the number of simulation scenarios covered.

Dockemu can simulate different network technologies (WiFi, Ethernet, LTE and 802.15.4) and can be used with different purposes:

- validate IoT software over realistic network conditions;
- check the impact of modification in the applications or at any level of the network stack;
- carry out performance test of IoT software under specific network conditions.

[19] *Playbook* is the name given by Ansible framework to the scripts where all the instruction needed to carry out a task are written.

6.1 Future Lines

We have identified several work lines to improve and complete Dockemu simulation in the future. These tasks can be divided in three main groups:

1. **Improvement of Existing Framework:** this group of tasks goes from improving and completing the documentation for the framework in order to be used by others to including more configuration parameters for the simulation of the network. This configuration parameters will give the researcher more control about what is happening on the wireless interfaces and enable the execution of more specific experiments.
2. **Addition of More Complex Network Topologies and Support of More Network Technologies:** the network topologies offered are pretty static. The framework could be easily enriched with more topologies and network technologies in order to cover more scenarios.
3. **Support of Parallel and Distributed Simulation Mode:** ns-3 supports parallel and distributed modes for simulation. These modes can handle a higher number of nodes which is essential to cover IoT scenarios with more elements in order to cover real-world scenarios. To support distributed simulation in ns-3, the standard Message Passing Interface (MPI) is used. There is an existing module[20] which already supports it and which makes use of the OpenMPI Linux library[21].

REFERENCES

Bormann, C. and Shelby, Z. (2016). Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959 (Proposed Standard).

D'Angelo, G., Ferretti, S., and Ghini, V. (2016). Simulation of the internet of things. *CoRR*, abs/1605.04876.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. *IEEE Software*, 33(3):94–100.

G Gupta, S., Ghonge, M., D P M Thakare, P., and Jawandhiya, P. (2013). Open-source network simulation tools an overview. 2.

Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., and Razafindralambo, T. (2011). A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*, 49(11):58–67.

Hasan, A. B., Kiong, T., Paw, J. K., Musa, A. B., and Mohamed, H. (2017). Real-time video streaming over ns3-based emulated lte networks. volume 4. International Journal of Electronics, Computer and Communications Technologies.

[20] https://www.nsnam.org/docs/models/html/distributed.html

[21] https://www.open-mpi.org/

Patrick Vandewalle, J. K. and Vetterli, M. (2009). Reproducible research in signal processing. *IEEE SIGNAL PROCESSING MAGAZINE*, pages 37,47.

Pino, D. (2016 (accessed April 22, 2018)). Network namespaces. https://blogs.igalia.com/dpino/2016/04/10/network-namespaces/.

Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. C. (2016). Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 1:1–1:13, New York, NY, USA. ACM.

To, M. A., Cano, M., and Biba, P. (2015). DOCKEMU – a network emulation tool. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. IEEE.

ur Rehman Khan, A., Bilal, S. M., and Othman, M. (2013). A performance comparison of network simulators for wireless networks. *CoRR*, abs/1307.4129.