Virtualization of Radio Access Network by Virtual Machine and Docker: Practice and Performance Analysis

Aravinthan Gopalasingham, Dalia Georgiana Herculea, Chung Shue Chen, Laurent Roullet
Nokia Bell Labs
Route de Villejust, 91620 Nozay, France
Email: {gopalasingham.aravinthan, dalia-georgiana.herculea, chung_shue.chen, laurent.roullet}@nokia.com

***Abstract*—Software defined networking (SDN) and network function virtualization (NFV) are the embraced technologies for the backhauling of future 5G networks. Virtual Machine (VM) and Docker container based deployments have received much attention. This paper presents the virtualization of a prototyped software defined radio access network (RAN) architecture by using VMs and Docker containers. In addition, it provides an analytical model for the generalized software defined RAN architecture with the practice of VM based and Docker container based implementations. Using measurements obtained from the two testbeds and the introduced queuing model, we compare their performances and analyze the two different architectures. Results verify the superiority of the Docker technology. Some observations from the behavior of the testbeds are concluded for a better understanding of the VM and Docker container based technologies for the future development of 5G SDN controller.**

***Index Terms*—5G, radio access network, SDN, Virtual Machine, Docker container, performance analysis.**

I. Introduction

5G will enable new user experience and ultra-broadband service [1]. Future mobile networks is expected to offer a capacity up to a thousand times of legacy 3G and 4G. To meet the network capacity beyond 4G, base stations (BS), also known as eNodeB (eNB), will be deployed in very high density and as heterogeneous cellular networks, consisting of macro cells and small cells of various coverage range. They will support various type of devices and communication setups (e.g., mobile users, vehicular communications for autonomous driving car, Internet of Things) while meeting the requirements of ultra low latency and high quality of experience.

Traditional wireless mobile network infrastructure was originally designed to handle voice traffic. However, today's mobile networks are being overwhelmed by the huge demand for data traffic. It is necessary to upgrade to infrastructures that can support very dense mobile networks as well as the interoperability between vendors, including interference management and dynamic escalation of network bandwidth demand. Besides, low power consumption, agile traffic management and high reliability are important for today's network architectures [2].

In order to meet these requirements and various service demands, the network should be very flexible and easily reconfigurable. The conceptual evolution of SDN that logically centralizes the control plane of the networks leverages standard protocols for managing and re-programming the entire network on the fly. The evolution of SDN is going on for enabling the programmability and flexibility of the radio access networks (RAN). The major benefit of such approach is to allow complete intelligence to the centralized controller so that the functionality of RAN can be better optimized [3]. On the other hand, NFV is the process of relocating or migrating network functions from dedicated hardware to generic servers. NFV often involves the usage of virtualization technologies such as VMs and containers for deployments, which are often called Virtual Network Functions (VNFs) in such use cases. SDN and NFV are two closely related technologies that are often used together. They are complementary and can benefit from each other.

Centralized-RAN, Cloud-RAN, or C-RAN architecture can address the requirements of network self-optimization, self-configuration and self-adaptation in software control through SDN and NFV. Cloud-RAN can support mobile xHaul (Fronthaul and Backhaul) and provide great benefits such as decreasing the operational cost [4] and improving the network security and flexibility. In the context of C-RAN, virtualization plays a key role in deploying and managing the life cycle of mobile network functions and services in the central cloud in order to achieve efficient resource utilization, elasticity, and load balancing. The NFV also results in a significant reduction in the CAPEX as well as OPEX by using automation techniques such as orchestration [5] together with SDN.

Nowadays, there are two major methods for achieving the dynamic resource control for the NFV: virtual machines (VM) and Docker containers. In today's practice, virtual machines are extensively used as they permit workloads to be isolated from each other and for the resources to be well controlled [6]. During the last few months, much attention has been

The research leading to these results has received funding from the European Union Superfluidity H2020 project, https://5g-ppp.eu/superfluidity. A part of the work was carried out at LINCS (www.lincs.fr).

given to Docker container based deployments of NFV. Docker containers have the great advantage of allowing applications to run separately from the host infrastructure and to treat the infrastructure like a managed application [7]. In particular, Docker can more easily adapt to the rapidly evolving infrastructure of virtualized RAN, with the possibility to ship code faster, test faster, deploy faster, and shorten the cycle between writing code and running code.

Some experiments have shown that Docker based deployment can offer superior performance compared to VM based deployment [6]. However, there is a lack of explicit model and study to facilitate the analysis and estimation of possible performance gain and address for example when it is of interest to use Docker containers or to use VM and how do these architectures scale. There are many interesting questions around.

A. Contributions

This paper presents a SDN controller for radio access networks built on virtual machines and Docker containers, respectively. We investigate their performance in our testbeds with VM implementation and Docker container implementation. Lessons and experience learned from the VM implementation and container implementation are reported.

The contributions of the paper are summarized below:

- We describe the practice of NFV implementation for 5G radio access networks and in particular the new option of using the emerging technology Docker containers for the generalized SDN platform introduced in [8].
- We provide a queuing model for analyzing the container based SDN controller in comparison to the VM deployment and show their performance. Using the model, we estimate the server processing time under the two architectures for comparison.
- We conduct experiments to evaluate the two deployments and give analytical results for the future development of SDN controller with different technologies.

B. State of the Art

Docker container is an emerging technology that has been proposed for the deployment of SDN architecture. There are some studies about the practice and performance based on measurements and experiments. The authors of [9] propose Docker containers for building multi-tenant cloud infrastructure. In [7], it is shown that in the described testbed, the performance achieved by using Dockers is the same as the performance in a bare metal system. Today, Docker container is a competitor of the virtual machine technology in the deployment of SDN for RAN.

In what regards evaluations based on analytical models, some investigations were done in order to provide accurate models for characterizing SDN controllers. The authors of [10] use an approach based on network calculus to study SDN architecture. The work in [11] proposes a SDN architecture with analytical model to study user quality-of-service (QoS) under the SDN based cloud computing architecture.

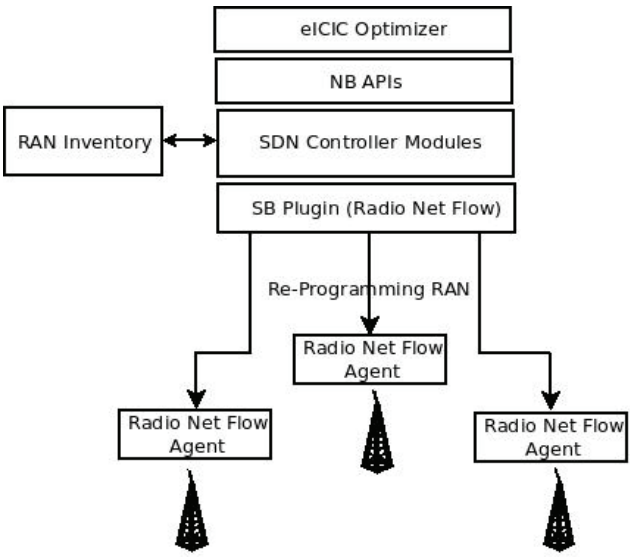

Fig. 1: Generalized SDN platform for RAN

In [12], the authors introduce a model to address multiple node OpenFlow based SDN by approximating the data plane as an open Jackson network with the controller modeled as an M/M/1 queue. An investigation of SDN is performed in [13] to analyze the impact of the network topology and size on the performance of the controller.

To the best of our knowledge, no comparison between the performances obtained in VM based architecture and Docker container based architecture has been done by modeling the controller as a queuing system.

II. Generalized SDN Platform for RAN

The main objective of our research is to bring programmability and flexibility to the RAN using SDN. The major benefit of such approach is to decouple network intelligence to the logically centralized controller(s) so that the functionality of RAN can be better optimized or re-programmed. Fig. 1 shows the general architecture of our SDN controller. It follows the standard three-layered architectural approach with (i) South-Bound (SB) protocol plug-ins to manage the communication between the controller and distributed data-plane entities of the network, (ii) controller modules to implement the network abstraction, and (iii) vendor agnostic Application Programming Interfaces (APIs) for NorthBound (NB) applications and algorithms to program the network.

A. SDN Platform

We implemented our architecture by extending the OpenDayLight (ODL) SDN controller for RAN features along with the integration of MongoDB to maintain the persistence of information and network intelligence. Radio Net Flow (RNF) is the SB protocol plugin introduced to integrate eNBs to this framework. The RNF is a SDN realization of the proprietary

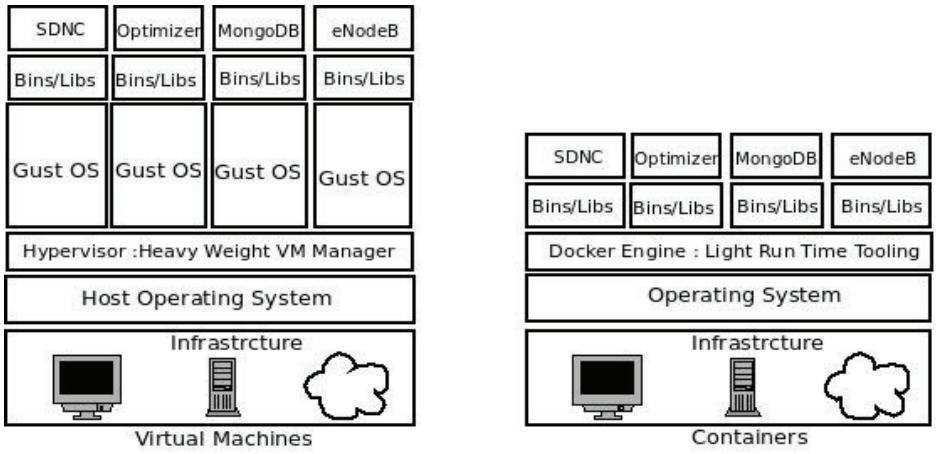

Fig. 2: (a) Virtual machine vs. (b) Docker container architecture

X2 interface [14] used between base stations to facilitate information exchange for the purpose of RAN optimization. Each base station in the network has also an integrated corresponding SDN-Agent called Radio Net Flow Agent (RNFA) [8]. The purpose of the RNFA is to establish communication with the controller. The RAN Configuration Manager, the RAN Statistics Manager, and the RAN topology Manager are the three new modules introduced in the Service Abstraction Layer (SAL) of ODL in our design to abstract and re-program both PHY and MAC layer functions of RAN using the SB RNF protocol.

Each eNB during the initialization establishes a connection to the SDN controller via RNF protocol. The RAN configuration manager in the controller is responsible for validating the initial connectivity to the eNB. Each eNB sends measurements and configuration related information to the controller using RNFA, which will be taken by the RAN Statistics Manager so as to store RAN statistic information in its database (DB). Since RNF protocol is built using Stream Control Transmission Protocol (SCTP), eNBs and the controller communicate in a round robin manner. RAN topology manager creates a network topology based on the neighbor related information that it receives from the eNB. Network applications and optimizations (e.g., enhanced inter-cell interference coordination, a.k.a. eICIC [14]) running on the top of the controller read the network configuration, measurements and topology information, using corresponding HTTP REST (Representational State Transfer) APIs as an input for the execution. Once the application finishes its cycle of execution, it stores the re-configuration parameter in the DB to be sent to corresponding eNBs. RAN configuration manager reads the DB frequently and sends new configuration information to eNB. When the RNFA receives the re-configuration parameters from the controller, it re-configures the corresponding functions in the eNB (e.g., MAC scheduling, frequency or time radio resource allocation decision, power control settings, etc).

B. Testbed Implementation

In the proposed architecture, we deployed 4G/5G eICIC optimizer as a RAN application of the controller which is to coordinate MAC scheduling over LTE networks. In the testbed and experimentation, we employed a Matlab based LTE-complaint simulator from TU Wien's Institute of Telecommunications to emulate LTE mobile network scenarios. It simulates typical 3GPP heterogeneous cellular network (HetNet) scenarios consisting of macro and small cells, and mobile users. For the purpose of experimenting the end-to-end functionality of our architecture, we emulated light-weighted eNBs that implement RNFA and are fed with eNB related information (measurements and configuration) by Matlab using standard User Datagram Protocol (UDP) socket connection. Then, the emulated eNB acts like real base station to the SDN controller. The Matlab simulator and SDN environment (emulated eNBs, SDN controller and MongoDB) are implemented in two physically separated machines. We deployed the SDN controller, MongoDB and eNBs as virtual entities in one physical machine and used another dedicated physical machine for the Matlab simulator.

In the deployment, we implemented the two virtualization techniques, virtual machines and Docker containers, under the same hardware configuration of 16-core CPUs and 16 GB RAM operating with Ubuntu Linux 14.04 version. The motivation of validating our framework under two different virtualization techniques is to analyze the performance and behavior of the system under the two different virtual environments and to provide a comparative study of their suitability for enabling 5G SDN/NFV.

1) Testbed using Virtual Machines: As shown in Fig. 2, the SDN controller, eICIC optimizer, MongoDB and emulated eNBs are deployed in separate VMs using lightweight version Ubuntu 14.04 images. We used Kernel-based Virtual Machine (KVM), that uses Quick Emulator (QEMU) to emulate hardwares for virtual machine deployment. We launched a number

of virtual machines in parallel by executing VM execution command through Python script. The networking between VMs is achieved through network address translation (NAT) through the virtual switch of the hypervisor. Though VMs use limited number of hypercalls or emulated devices via hypervisor to establish communication with the outside world with a certain level of isolation and security, it is known that the hypervisor would add substantial overheads on the performance. These cannot be bypassed or optimized from higher layer. Note that the performance of VM based testbed is often lower when compared to native machines but this can be compensated by the advantage of virtualizing hardware resources of the same physical machine for several entities.

2) Testbed using Docker Containers: As shown in Fig. 3, the SDN controller, eICIC optimizer, MongoDB and emulated eNBs are deployed in isolated Docker containers using Alpine Linux version 3.2 that is 5 MB in size built around musl libc [15] and busybox [16]. In order to achieve the containerization of each entity in our architecture, we have created separate Docker files along with the necessary libraries, including Java 1.7, Maven [17], SCTP for the deployment of ODL controller, json [18], and gcc for eICIC optimizer which is developed using C++. We have used the MongoDB Docker file from Docker hub for the deployment of DB container in our platform. We built all the Docker files using Docker daemon version 1.11 to produce corresponding Docker images and stored them in the local repository before launching our experimentation testbed. During the experimentation, we launched all the containers such as ODL controller, MongoDB, eICIC optimizer and emulated eNBs using a single shell script. The instantiation of all the containers of our experimentation testbed appears very fast: it is less than one minute compared to the VM implementation that may take several minutes.

The advantage of using Docker containers is that a container will virtualize the kernel so that each container uses underlying kernel with its own process and network space which is more light weighted and efficient compared to VMs that run full operating system as shown in Fig. 2. Besides, the communication latency between Docker containers is much lower compared to VMs which have hypervisor overheads. Processes running within containers are more elastic in nature compared to those deployed within VMs which are more static in resource utilization. The entire booting time for our testbed with several Docker containers is of the order of seconds whereas the booting time with VMs is of the order of tens of seconds or even more than a minute. This is due to the fact that Docker containers share the single OS and require to load only the necessary software packages and libraries for particular processes whereas VMs require the entire OS to be loaded irrespective of the requirement of the deployed processes.

III. Controller Queuing Model

According to the previously described architecture, the SDN controller manipulates the fluxes of information (measurements and configuration information) received from the eNBs.

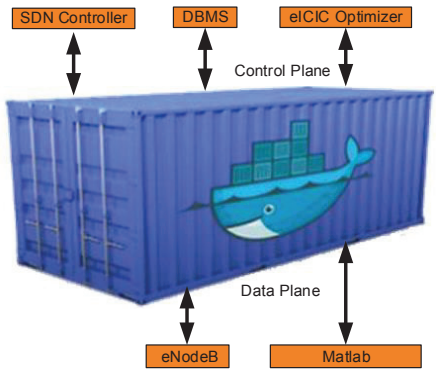

Fig. 3: Testbed using Docker containers

The fluxes are collected in a round robin manner. After the fluxes are collected and aggregated into a single flux, this flux is sent to the central entity. The size of the flux and consequently the load would depend on the number of eNBs for which the SDN controller conduct operations. The i-th flux is assumed as being received in conformity to a Poisson arrival process with parameter λ_i. Note that the parameter λ_i stands for the expectation of the arrival rate of the packet-in messages at the controller.

In order to characterize the processing discipline of the SDN controller, we adopt the M/M/1 queuing model [19]. In a M/M/1 queuing model, the length of packet-in message queue at a SDN controller $N(t)$ is a birth and death process on the countable infinite state set $E = \{0, 1, 2, \ldots\}$. We denote its instantaneous probability by $p_j(t)$ and the stationary probability by p_j such that

$$p_j(t) = P\{N(t) = j\}, \tag{1}$$

and

$$p_j = \lim_{t\to\infty} p_j(t), j \in E. \tag{2}$$

We denote the traffic intensity of the controller by ρ, which is given by λ/μ, where λ is the inter-arrival rate and μ is the service rate. We can use Little's Law to express the length of the waiting queue:

$$\bar{N}_q = \frac{\rho}{1-\rho}. \tag{3}$$

Notice that $p_0 = 1 - \rho$ is the probability of the controller in idle state, and ρ is just the probability of the controller in busy state, where $0 \leq \rho \leq 1$. It is easy to observe that the controller becomes busier with the increase of ρ.

The controller processes the packets on a first-come-first-served basis. Therefore, the distribution of the waiting time of a packet-in message is thus given by:

$$W_{q(t)} = P\{W_q \leq t\} = 1 - \rho e^{-(\mu-\lambda)t}, \ t \geqslant 0. \tag{4}$$

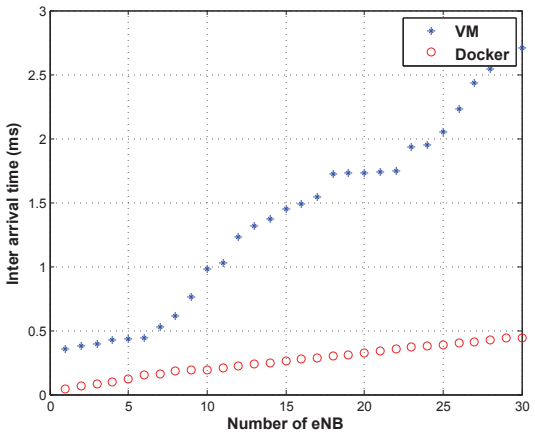

Fig. 4: Inter-arrival time of packets under VM architecture and Docker architecture

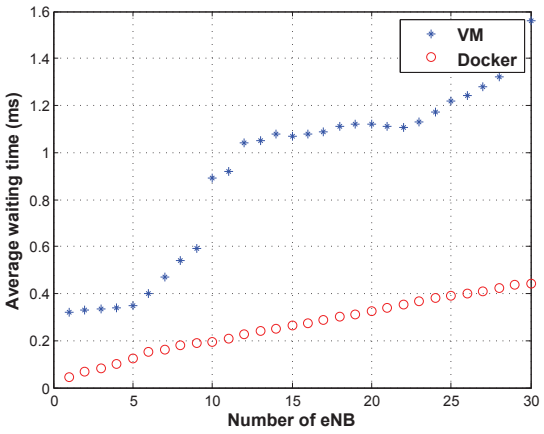

Fig. 5: Average waiting time for the VM based architecture and for Docker based architecture

Using (4), we can express the average waiting time of a packet-in message as:

$$\bar{W}_q = E[W_{q(t)}] = \frac{\rho}{\mu(1-\rho)}. \quad (5)$$

By adding the expected processing time of a packet, which is given by 1/ μ, we have the sojourn time of a packet in the controller expressible as:

$$\bar{W}(t) = \bar{W}_q + \frac{1}{\mu} = \frac{1}{\mu - \lambda}. \quad (6)$$

From (6), we get the processing rate:

$$\mu = \frac{1}{\bar{W}(t)} + \lambda. \quad (7)$$

The above result will allow us to estimate the processing rate in the testbeds by mapping the behavior of the testbeds to the queuing model and the measurements to metrics in terms of inter-arrival rate and average waiting time.

IV. Numerical Evaluation

A. Fitting the Model to the Measurements

The measurements are performed in the two testbeds described in Section II-B1 and II-B2. Fig. 4 illustrates the inter-arrival time of the packets collected from the eNBs served by the SDN controller. We increase the number of eNBs gradually. Results show a significant difference in the behavior of the two testbeds. We find that it is due to the fact the hardware resource of physical machine that is virtualized using VMs would easily get over-loaded with the increase in the number of virtual nodes. The VM based architecture has the intrinsic nature that requires more resources to deploy the whole operating system stack comparing to Docker containers that run only the necessary packages and libraries on top of the existing kernel. As a result, the communication latency between Docker containers is significantly lower compared

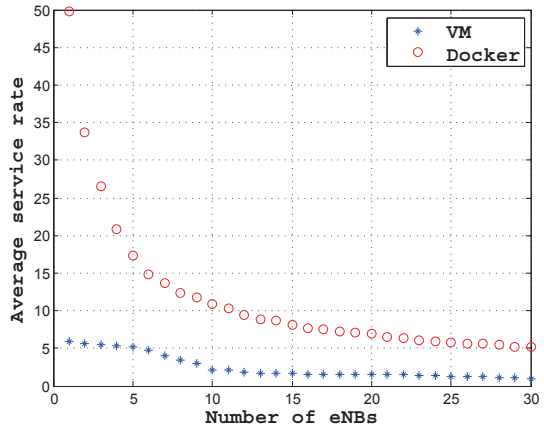

Fig. 6: Service rate for the VM based architecture and for Docker based architecture

with that of VMs which has one extra layer of hypervisor, that will generate additional delay during the data arrival to the centralized controller. The non-linearity of the VM testbed is due to the extra time required by VM platform instantiation.

Next, we measure the average waiting time spent in the two systems from the testbeds. Note that this waiting time refers to $\bar{W}_q$ given by (5). Fig. 5 shows the experimental results. As expected, in both testbeds the average waiting time increases with the number of eNBs due to the increase of the load which will result in a request of more physical resources. However, the Docker container architecture shows clearly its superiority over the VM.

B. VM Architecture vs Docker Container Architecture Service Processing Time

We model the behavior of the two described testbeds with the analytical model described in Section III in order to

estimate the processing time required for the eNBs information update. Expressing the processing rate by (7) and using the measured inter-arrival rate (see Fig. 4) and waiting time (see Fig. 5), we can obtain the service rate for the VM and Docker based testbeds and plot the result.

From Fig. 6, we can see that both the average service rates of Docker container testbed and VM testbed decrease with the number of virtual nodes running in the same physical system. However, the performance due to Docker containers is always higher than that of VMs due to its lightweight and lower resource occupancy in achieving the same amount of tasks. Note that the Docker container testbed shows a very high service rate when the system is started for a small number of eNBs. This behavior is a consequence of the container's virtually zero performance overhead and fast kernel resource allocation. On the other hand, the VM testbed's performance is nearly flat since for any number of eNBs the testbed will load the whole operating system with large overhead and the resource allocation is comparatively slow.

V. Conclusion

This work presents a practice of virtualization of RAN and describes the experimentation in using VMs and Docker containers for implementing a software defined mobile network. The virtualized RAN architecture is evaluated using an analytical model that aims to derive the processing rate under the two NFV deployments. Measurements and comparisons confirm that the Docker container based architecture can provide a superior performance compared to the VM based architecture. This is helpful for the future development of Software Defined Mobile Networks. Note that we also expect the analytical method introduced here could be also useful for studying other similar systems such as Cloud RAN.

References

[1] Nokia Solutions and Networks, "Looking ahead to 5G," *White Paper*, 2014.

[2] Fujitsu Network Communications Inc., "The benefits of cloud RAN architecture in mobile network expansion," *White Paper*, 2014.

[3] D. Boviz, N. Abbas, G. Aravinthan, C. S. Chen, and M. A. Dridi, "Multi-cell coordination in Cloud RAN: Architecture and optimization," in *2016 International Conference on Wireless Networks and Mobile Communications*, Oct 2016, pp. 271–277.

[4] D. Boviz, C. S. Chen, and S. Yang, "Cost-aware fronthaul rate allocation to maximize benefit of multi-user reception in C-RAN," in *IEEE Wireless Communications and Networking Conference*, Mar. 2017.

[5] A. Gopalasingham, Q. Pham Van, L. Roullet, C. S. Chen, E. Renault, L. Natarianni, S. D. Marchi, and E. Hamman, "Software-defined mobile backhaul for future train to ground communication services," in *IFIP Wireless and Mobile Networking Conference*, July 2016, pp. 161–167.

[6] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2015, pp. 171–172.

[7] E. Preeth, F. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," in *2015 International Conference on Control Communication Computing India*, Nov 2015, pp. 697–700.

[8] A. Gopalasingham, L. Roullet, N. Trabelsi, C. S. Chen, A. Hebbar, and E. Bizouarn, "Generalized software defined network platform for radio access networks," in *IEEE Consumer Communications and Networking Conference*, Jan 2016, pp. 626–629.

[9] A. Slominski, V. Muthusamy, and R. Khalaf, "Building a multi-tenant cloud service from legacy code with Docker containers," in *IEEE International Conference on Cloud Engineering*, March 2015, pp. 394–396.

[10] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An analytical model for software defined networking: A network calculus based approach," in *IEEE Global Communications Conference*, Dec 2013, pp. 1397–1402.

[11] T. C. Yen and C. S. Su, "An SDN-based cloud computing architecture and its mathematical model," in *International Conference on Information Science, Electronics and Electrical Engineering*, vol. 3, April 2014, pp. 1728–1731.

[12] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.

[13] C. Metter, S. Gebert, S. Lange, T. Zinner, P. Tran-Gia, and M. Jarschel, "Investigating the impact of network topology on the processing times of SDN controllers," in *IFIP/IEEE International Symposium on Integrated Network Management*, May 2015, pp. 1214–1219.

[14] S. Sesia, I. Toufik, and M. Baker, *LTE - The UMTS Long Term Evolution: From Theory to Practice*. Wiley, 2009.

[15] Standard library musl. [Online]. Available: https://www.musl-libc.org

[16] BusyBox. [Online]. Available: https://busybox.net

[17] Apache Maven. [Online]. Available: https://maven.apache.org

[18] JavaScript Object Notation. [Online]. Available: http://www.json.org

[19] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.