Performance Evaluation on Virtualization Technologies for NFV Deployment in 5G Networks

Rasoul Behravesh, Estefanía Coronado, and Roberto Riggio
Wireless and Networked Systems, FBK CREATE-NET, Trento, Italy
Email: {rbehravesh, e.coronado, rriggio}@fbk.eu

***Abstract*—Multi-access Edge Computing (MEC) is regarded as a pivotal pillar to grasp the particularized 5G goals by shifting network intelligence from the cloud to the edge. Network Function Virtualization (NFV) emerged as a paradigm intending to replace traditional vendor-specific network appliances with software instances of the network functions capable of running on standard devices. Recently, deploying softwarized network functions at the network edge has gained an unprecedented attention. Multiple virtualization technologies can be utilized to deploy virtualized network functions including Virtual Machines (VMs), containers, and unikernels. However, each virtualization platform has specific advantages and disadvantages, which makes worthy studying their real performance. This is specially important when it comes to implement network functions at the edge of 5G networks, where resources are scarce and quick response to user requests is needed. In this regard, this paper studies the performance of virtualization technologies by deploying two services namely Apache and Redis and provides an extensive experimental campaign and conclusive results.**

***Index Terms*—5G, Multi-access Edge Computing, Virtualization, VMs, Containers, Unikernels**

I. Introduction

While 4G LTE networks are present in daily communications, ranging from calls to Internet surfing, future 5G systems are already a reality. Comparing to its predecessor, 5G promises significantly higher data rate, ultra-low latency, and improved Quality of Service (QoS). In fact, numerous applications like Virtual Reality (VR), autonomous driving, and Internet of Things (IoT) benefit from the eye-catching advantages of 5G networks. Nevertheless, this pervasive connected world requires to handle real-time massive amounts of data from users and applications. So far, this processing burden has been shifted to resource-rich cloud data centres [1]. However, transmitting data to/from the cloud induces significant delay. This issue can be addressed by setting the resources closer to the user and implement them at Network Points-of-Presence (N-PoP) sites. This paradigm known as Multi-access Edge Computing (MEC) decreases network latency and bandwidth consumption and prevents from single point of failure, hence providing better user experience [2].

Network Function Virtualization (NFV) has attracted considerable attention from network operators in the transition from 4G to 5G networks. NFV encourages a more scalable, agile, and inexpensive network by decoupling network functions from the underlying hardware and implementing them in the form of software instances called Virtual Network Functions (VNFs). Until now, network operators leveraged cloud computing for deploying network functions as a way to avoid investing a huge amount of money on dedicated devices. However, MEC allows them not only to host their own desired network functions but also to be prepared to host third parties services, making a new revenue stream [3].

Traditionally, Virtual Machines (VMs) were the main technology for VNF deployment, providing full-featured and isolated environments but with a huge overhead on the system. Recently, containers have emerged as a promising solution to replace heavy-weight and resource hungry VMs that are very costly to migrate and scale. Containers encapsulate applications and their dependencies in a light-weight, highly-portable, and executable entity, able to run on different platforms [4]. However, they face security vulnerabilities caused by sharing the Operating System (OS) kernel between all the containers in the host [5]. Similarly, unikernel technology has also absorbed considerable attention from research community. Unikernels are light-weight machine images aiming at reducing memory footprint and image size of applications by integrating application code and its dependencies into a single bootable binary images [6].

The substantial benefits of virtualization technologies have led Telecommunication Service Providers (TSPs) to choose them as a solution for service deployment. Nevertheless, they can dramatically impact the performance of applications, as well as the agility and scalability of resource-scarce deployments. For that reason, if virtualization technologies are to be applied to 5G networks, it becomes of vital importance to analyse the capabilities that they can provide. Although studies in this respect have been done in the past, many focus on a subset of the platforms [7], [8], or do not consider the most determinant indicators in resource-scarce scenarios [9]. To the extent of our knowledge, a direct performance evaluation across the three main virtualization technologies for VNF deployment, i.e. VMs, containers and unikernels, covering various networking aspects has not been performed before. In particular, in this paper we discuss and compare the performance of the aforementioned technologies by deploying two network services, namely Redis and Apache HTTP, as platforms enabling VNF deployment in 5G networks.

This work has been supported by the European Union's H2020 Research and Innovation Action under Grant Agreement H2020-ICT-761592 (5G-ESSENCE Project).

978-1-5386-9376-6/19/$31.00 ©2019 IEEE

The rest of the paper structured as follows. Section II overviews the studies performed on virtualization technologies, followed by an in-depth description of each technology in Sec. III. Section IV presents the system setup, and discusses the results of the experimental campaign. Finally, we draw our conclusions and present the future work in Sec. V.

II. Related Work

This section reviews the efforts made on assessing the existing virtualization technologies for VNF deployment. While the majority of the literature has chosen Mirage [10], OSv [11], and Rumprun [12], [13] for unikenrel construction, Docker and LXC are the mainstream for container management. Similarly, since KVM is the native hypervisor of Linux, it is the preferred option of the works presented in this section for VM deployment.

OpenStack [14] has been widely used for service provisioning. In this regard, the work presented in [9] investigates the service provisioning time of utilizing different virtualization systems including containers, unikernels, and virtual machines. The authors compare the provisioning time of OSv unikernels against Docker containers and KVM virtual machines. The main intention of the work is to analyze the impact of executing concurrent services on the provisioning time of the services.

The study presented in [8] conducts a deep analysis of the performance of containers against unikernels for REST services with different languages including Java, Go, and Python. They evaluate the execution time and memory footprint for the REST services implemented in OSv and Docker across all the languages. Moreover, the impact of using single-thread and multi-thread processes is also investigated. Another work in [16] extends the experiments domain by involving multiple platforms for each container, unikernel and VM. Authors study the memory footprint and throughput of Nginx and Redis key-value store on OSv and Rumprun for unikenrels, LXD and Docker for containers and KVM for virtual machines.

As opposed to other works, this paper performs a deep study on the performance evaluation of three main enabling technologies for VNF deployment. We investigate the performance of Apache HTTP service and Redis key-value store on three different virtualization systems. Our work, is different from the others since we study the performance of services with different network and computing demands. We tried to make minimum changes to the applications and select an unikernel solution that supports POSIX applications. Contrary to some works that compare the services developed in different programming languages, this work keeps the application unchanged for different virtualization systems since the programming language used for the application development greatly impacts the overall performance of the application. We aim to conduct a fair evaluation completely focused on the virtualization capabilities.

III. Background on Virtualization Technologies

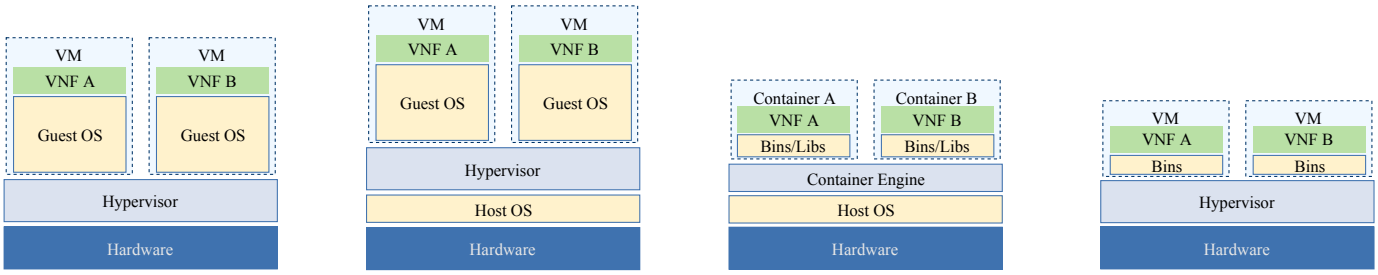
This section presents an overview on different virtualization technologies currently being used in the cloud and at the edge to run multiple applications on the same underlying server while keeping them isolated. The objective of all the virtualization technologies is to break up the bulk physical resources into smaller instances allocated to different users or aggregating resources on different servers to provide a common view of the pool of resources that can be assigned or released dynamically.

Hypervisor-based virtualization is the main technology to provide isolated environments on top of a shared pool of resources. Hypervisor is a software layer that abstracts the underlying physical resources and provides virtual machines with full functionalities of a real system. Hypervisors are classified into (i) *Hypervisor Type-1*: as depicted in Figure 1 the hypervisor is directly installed on the bare-metal hardware, providing better performance by eliminating the need for the host OS. KVM, Xen, and HyperV are examples of Type-1 hypervisor. (ii) *Hypervisor Type-2*: as shown in the Figure 1 the hypervisor is installed as an application on top of the operating system. The installation is simple but generates more overhead than the Type-1. VirtualBox and VMWare Workstation are examples of this type of virtualization.

Currently, employing VMs for VNF deployment is the main trend in the cloud. VMs provide a good level of security for the applications due to the solid isolation that happens at the hardware level. Usually general purposed operating systems are installed inside VMs to run applications, which makes them heavyweight instances that consume a considerable amount of resources. This, also makes the instantiation time, migration and scaling very costly.

As noticed, the main predicament with the VM is the bulky size of the OS, which is intentionally designed generic to meet the needs of users with different application scenarios. The principal inclination in the cloud is to use general-purposed OSes and assign each VM to run a single service, which results in large size OSes with the full list of features that are mostly unusable for the service. Unikernels are completely opposed to the monolithic approach of OS design and emerged as a solution to provide OS services and features including the networking stack, memory access, and scheduling in the form of libraries to construct light-weight, single-purpose, and secure virtual machine images. As it can be perceived from Figure 1, unikernels are extremely light-weight, eliminating all the OS services, features, and libraries that are not required for running an application. Regarding the limited OS libraries that are used for each application, the attack surface shrinks significantly that leads to better security levels. Unikernel approach allows to run only one application inside each unikernel instance and cannot be expanded after construction [10].

Since the introduction of Mirage as the first platform for constructing unikernels, many other platforms such as Light-weight Virtual Machine (HaLVM) [17], IncludeOS [18], OSv [19], and Rumprun [12] have emerged but few of them



(a) Type-1 Hypervisors. (b) Type-2 Hypervisors. (c) Containers. (d) Unikernels.

Fig. 1. High-level comparison between virtualization technologies.

gained attention. Rumprun is different from other solutions, in the sense that device drivers that constitute the majority of the code lines of an unikernel can be deployed in the user-space. By this approach, crashes in drivers do not lead to the failure of the whole VM. Rumprun employs NetBSD [20] kernel drivers to support application execution. POSIX applications can be easily ported to Rumprun unikernel without imposing any changes to the application. Moreover, it supports different platforms including KVM, Xen, and bare-metal platforms supported by NetBSD.

Containerization is another virtualization approach that has gained attention, in which virtualization happens at the host level and provides isolated environments for applications as depicted in Figure 1. Containers are lightweight and portable application instances containing the libraries and dependencies required for the application. Linux by default has namespaces and cgroups features, which gives the ability to create isolated environments and limiting the access of each application to the shared resources, respectively. Being small-size, agile, and scalable are some features of containers that make them a greatly used technology for VNF deployment. Security vulnerabilities originating from the shared kernel is the main predicament with the container technology. Multiple container engines have been introduced for simplifying creation, running, and termination of containers, among which LXC, Docker, and rkt have gained more reputation.

IV. Performance Evaluation

This section provides comprehensive details about the methodology followed for the performance evaluation, including the Key Performance Indicators (KPIs) considered. Then we provide a description of the services we have implemented in three different virtualization platforms. Moreover, we describe the results produced by various experiments performed on each service.

A. System Setup

The system setup comprises an Intel NUC device equipped with a Kingston SODIMM DDR4 RAM with 16GB capacity and an Intel(R) Core(TM) i7-7567U CPU with 3.50GHz clock rate. Ubuntu 18.04.1 LTS is utilized as the host OS for all the platforms. The experiments for each deployment is performed independently. Regarding the need of VM and unikernel scenarios for a hypervisor, Qemu alongside KVM is used to achieve a near-native performance for virtual machines. Furthermore, Docker container engine (version 18.06.1-ce) is utilized for running containers on the system.

The communication between containers and the host operating system is established through Docker's default (Docker0). Moreover, the guest OS of virtual machines, as well as unikernels, are connected to the host operating system through Linux default bridge and a static IP address that is assigned to *veth* pairs. The evaluated services are executed in order and independently from each other in all the platforms.

B. Methodology

Our evaluations examine the performance of two services namely Apache Hyper Text Transport Protocols (HTTP) server and Redis key-value store on three different platforms: VMs, containers, and unikernels. The main objective is to reach an understanding on the impact of virtualization platforms on the performance of the service and clarify the pros and cons of each platform.

Aiming at studying the impact of the virtualization technologies and the real performance of the services, as well as discovering the limitations of the platforms, Memtier-benchmark [21] and ab (Apache HTTP server benchmarking tool) [22] benchmarking tools are employed. Both benchmarking tools are installed on the host operating system to send requests to the servers and collecting results of the experiment.

Apache is a cross-platform open-source HTTP server responsible for managing web pages in an efficient manner. It is a fast, secure and reliable web server, known to be the most widely used on the Internet. Many factors such as design, usability, and web page content are critical for the success of a website but the most important factor is the real performance of the web server. Therefore, evaluating such performance becomes an integral part of the deployment.

We evaluate the performance of the Apache HTTP server based on different KPIs. First, we evaluate the image size of the service in three different platforms to see the amount of storage required to host the application. Next, we analyze the CPU and memory utilization of the service, which has a great impact on the number of services that a physical server can run simultaneously. Linux *top* command is employed for VM

and unikernel deployments alongside *Docker status* command in container deployments.

Delay in responding to user requests can drastically degrade the user's satisfaction from a website. A considerable portion of the overall delay depends on the server's performance in responding to the requests. First, we analyze the delay in the HTTP server by varying the number of requests to the server. In this regard, we increase the number of requests in each step from 100 to 30000 and repeat the experiments for 10 times. Second, we keep the number of requests unchanged and repeat the tests to measure the transfer rate of the HTTP server.

Redis is a data structure server employed for storing different type of values. Redis can be used for storing data, caching data, or simply as a message broker. Similar to Apache, Redis server is deployed in three different platforms. Memtier-benchmark is a tool capable of generating different traffic patterns to evaluate the performance of the Redis server. Memtier-benchmark has the ability to generate traffic for multi-thread and multi-client scenarios. It provides different options to be configured for benchmarking Redis service like the number of requests per each client and data size to examine the performance in different situations.

In the first two experiments, image size and memory consumption of the service is analyzed. The third KPI is the CPU utilization performed in two different scenarios. One examines the CPU utilization when the service is running and does not serve any request from the user, and the other employs the benchmarking tool to configure 50 clients and send 10000 requests per client to assess the impact of the requests on the service. The experiments are repeated 10 times.

Regarding the importance of the latency in storing and retrieving data, we examine the latency performance of Redis server with different traffic patterns. The latency is calculated as the maximum latency from the time that a client issues the request to the server until the time the reply is received by the client. In the first scenario, we examine the impact of increasing the number of requests per client. The data size is fixed and it is considered to be 32KBytes for all the requests. We repeated the experiments 10 times for each request number, which varies from 100 to 30000. In another experiment, the data size is increased from 16 to 512 KBytes, while the number of requests is fixed to 1000 per client.

C. Results

This section illustrates the results acquired from the experiments. First, we investigate the image size of the services implemented in different platforms, in this respect, the values obtained are shown in Table IV-C. Given the architecture of the various technologies the image size of Rumprun unikernels for both services are significantly lower than other platforms. The main reason comes from the fact that unikernels just contain the dependencies required to run the application and remove all other libraries and binaries necessary for running the application. Contrary to the unikernels, applications running in containers require the normal operating system operations, which are provided by Ubuntu OS for both applications.

TABLE I
IMAGE SIZE OF THE SERVICES DEPLOYED IN DIFFERENT PLATFORMS

Service Name	Rumprun Unikernel	Container	VM
Apache	6.6 MB	115 MB	14.6 GB
Redis	3.7 MB	101 MB	14.5 GB

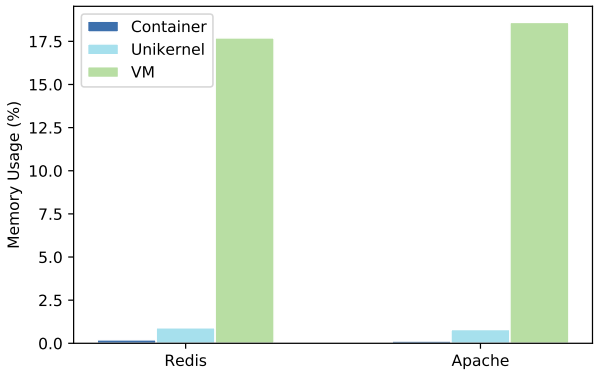


Fig. 2. Memory utilization of Redis and Apache HTTP services running in different virtualization environments.

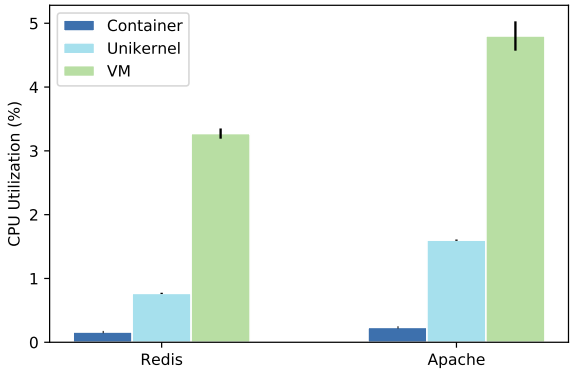


Fig. 3. CPU utilization of the services in the idle mode.

The Ubuntu OS image size is 88.1 MB and the rest of the size remain for the application and dependencies. Obviously, VMs are not comparable with other platforms because of the general-purpose OS that is running inside the VMs and it reaches about 15 GB with minimal utility installation.

As illustrated in Figure 2 the memory usage of containers is much less than Rumprun unikernel and VMs. The main reason is the efficient and dynamic usage of memory done by containers, which is opposed to the fixed size memory allocation in Rumprun unikernels. Although VMs dynamically use the memory, due to the huge number of services running on the general-purpose OS they cannot compete with containers.

Regarding the importance of CPU utilization of each service on the overall performance of the system, Figure 4 depicts the CPU utilization of both services running on different

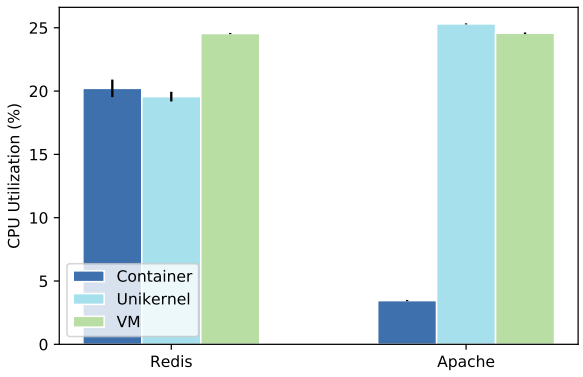


Fig. 4. CPU utilization of services with 1000 requests destined to the servers.

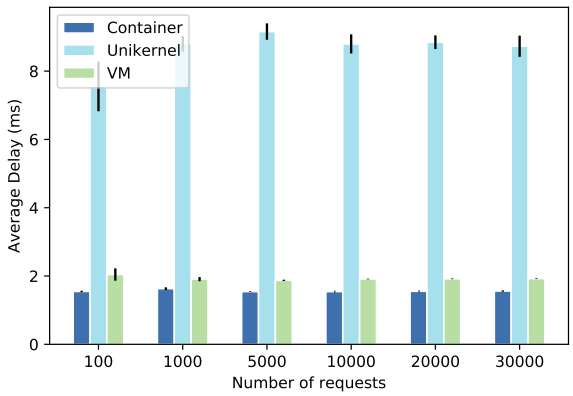


Fig. 5. Redis server latency with 32 KBytes packet size for an increasing number of requests.

platforms. However as sketched in Figure 3, the CPU utilization of the services is very low in idle mode, it increases drastically when requests are destined at the service, especially for Rumprun unikernels. Apart from that, unikernels and VMs employ a hypervisor layer (KVM + Qemu in our case) that uses the acceleration methods to speed up running applications, which in return add some extra processing to the system. The confidence interval for all the plotted results is set to 0.95.

The services studied in this paper have different demands in terms of memory and CPU usage, therefore considering them separately and study their performance can help to better understand the advantages and limitations of the services on different virtualization environments. First, we examine the average latency of SET and GET operations in Redis server. To this end, the number of requests sent to the server is increased and data size is fixed to 32 KBytes. As it can be inferred from Figure 5, increasing the number of requests does not affect the server's performance, while the Rumprun unikernels perform poorly comparing to containers and VMs. Second, the impact of data size is studied by generating 1000 requests and increasing the packet size of the requests ranging from 16 to 512 KBytes. Although the delay increases slightly while increasing data size of packets, the virtualization platform's performance remains similar to the first case. This effect is shown in Figure 6.

Similar to the Redis server, we conducted some experiments on the Apache HTTP server to evaluate the behavior of the application on the different technologies from different points of views. We aim to evaluate the service in terms of the time required to serve each request and the transfer rate of the server. As depicted in Figure 7 the time per request for the Rumprun unikernels is very high and growing by the increment in the number of generated requests, which is mostly because of the poor process management in which fork() and execve() system calls for making child processes are not supported and no other process will be created to handle the requests. Therefore, the delay for serving requests

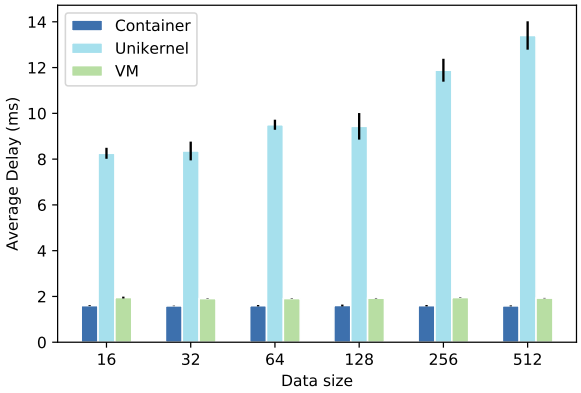


Fig. 6. Redis server latency for 1000 requests vs increasing packet data size.

increases in unikernels, while containers and VMs can make child process and concurrently serve requests to increase the performance. The same experiments are performed to evaluate the transfer rate of the service. As it is expected from the previous results, Rumprun performs poorly and has a lower transfer rate. In Figure 8 it can be observed that the transfer rate of Rumprun unikernels is much lower than containers but comparable to VMs.

V. Conclusion

This paper contributed a comprehensive study on three virtualization environments namely containers, unikernels, and VMs. To do this, we implemented Redis and Apache services alongside three virtualization systems and evaluate different KPIs. While unikernels have a smaller image size and very small memory consumption, the time required for process completion is increasing mainly due to the inefficient memory management and the hypervisor overhead. Containers are also very light-weight both for memory and CPU utilization but they perform well for both services. The process management is more efficient comparing to unikernels and

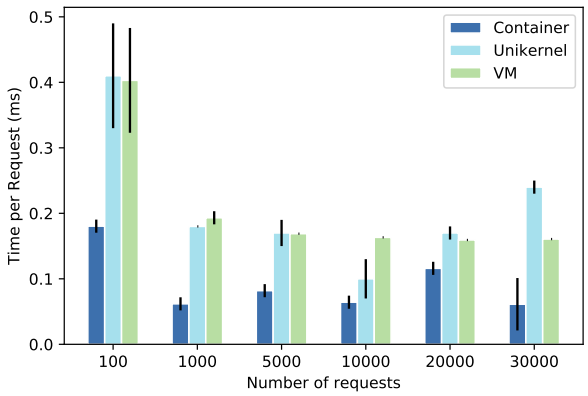


Fig. 7. Time to serve each request in Apache HTTP server fro an increasing number of requests.

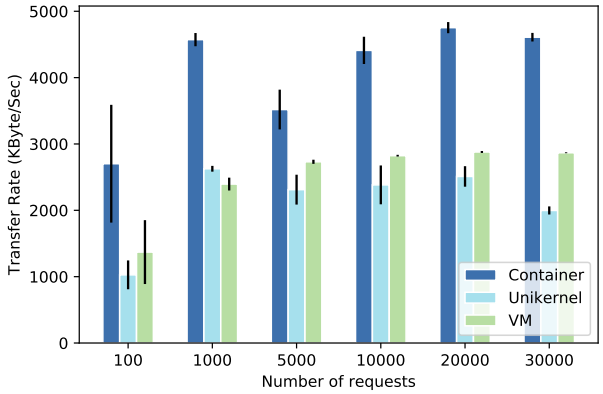


Fig. 8. Transfer rate of the Apache HTTP server for an increasing number of requests.

the overhead produced by the hypervisor is eliminated. The performance of VMs is compatible with containers except for the memory and CPU consumption, caused by the high number of services running in background. In fact, although containers and VMs showed a stable performance comparing to the Rumprun unikernel for both services, we can claim that actual performance of unikernels can be achieved by selecting the proper application as well as the proper platform for constructing the unikernel. Applications with high context switching between user and kernel mode can perform better when deployed as unikernels because unikernels eliminate the overhead of context switching by only presenting one execution mode, which is the privileged mode. Regarding the obtained results, in the future works we are going to choose among the virtualization systems to deploy VNFs at the cloud or at the edge.

REFERENCES

[1] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5G wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016.

[2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 1, 2018.

[5] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My VM is Lighter (and Safer) than your Container," in *Proc. of ACM OSP*, Shanghai, China, 2017.

[6] A. Madhavapeddy and D. J. Scott, "Unikernels: the rise of the virtual library operating system," *Communications of the ACM*, vol. 57, no. 1, pp. 61–69, 2014.

[7] I. Briggs, M. Day, Y. Guo, P. Marheine, and E. Eide, "A performance evaluation of unikernels," in *Technical Report*, 2014.

[8] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, and F. De Turck, "Unikernels vs containers: An in-depth benchmarking study in the context of microservice applications," in *Proc. of IEEE SC2*, Paris, France, 2018.

[9] B. Xavier, T. Ferreto, and L. Jersak, "Time provisioning evaluation of kvm, docker and unikernels in a cloud platform," in *Proc. of IEEE CCGrid*, Cartagena, Colombia, 2016.

[10] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *Proc. of ACM SIGPLAN Notices*, Houston, Texas, USA, 2013.

[11] OSv, a new operating system for the cloud. Accessed on 12.02.2019. [Online]. Available: https://github.com/cloudius-systems/osv

[12] "The Rumprun unikernel and toolchain for various platfrms ," Accessed on 04.02.2019. [Online]. Available: https://github.com/rumpkernel/rumprun

[13] A. Kantee and J. Cormack, "Rump Kernels No OS? No Problem!" *USENIX; login: magazine*, 2014.

[14] "OpenStack: open source software for creating private and public clouds," Accessed on 07.02.2019. [Online]. Available: http://www.openstack.org/

[15] M. J. De Lucia, "A Survey on Security Isolation of Virtualization, Containers, and Unikernels," US Army Research Laboratory Aberdeen Proving Ground United States, Tech. Rep., 2017.

[16] M. Plauth, L. Feinbube, and A. Polze, "A performance evaluation of lightweight approaches to virtualization," *Cloud Computing*, vol. 2017, pp. 15–19, 2017.

[17] The Haskell Lightweight Virtual Machine (HaLVM). Accessed on 13.02.2019. [Online]. Available: https://github.com/GaloisInc/HaLVM

[18] A. Bratterud, A. A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum, "IncludeOS: A minimal, resource efficient unikernel for cloud services," in *Proc. of IEEE CloudCom*, Vancouver, BC, Canada, 2016.

[19] A. Kivity, D. L. G. Costa, and P. Enberg, "OSv Optimizing the Operating System for Virtual Machines," in *Proc. of IEEE USENIX*, Philadelphia, USA, 2014.

[20] "Unix-like open source operating system," Accessed on 04.02.2019. [Online]. Available: https://www.netbsd.org/

[21] "NoSQL Redis and Memcache traffic generation and benchmarking tool," Accessed on 02.02.2019. [Online]. Available: https://github.com/RedisLabs/memtier/benchmark

[22] "ab-Apache HTTP server benchmarking tool," Accessed on 02.02.2019. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html