Unikernel Network Functions: A Journey Beyond the Containers

Tytus Kurek

ABSTRACT

In the last few years, network functions virtualization technology has matured and become a de facto standard in the implementation of modern telecommunications infrastructure. At the same time, telecommunication service providers began exploring container technologies in search of better performance results. However, the use of containers for network functions virtualization purposes entails a number of limitations, such as security and isolation problems resulting from the use of a shared kernel. This article introduces unikernel network functions as the next step toward network functions softwarization. The use of unikernels enables the implementation of network functions in the form of lightweight images that can run either on a hypervisor or directly on the hardware layer. It is shown that unikernel network functions achieve comparable performance results to those based on containers, while providing a significantly higher security level.

INTRODUCTION

The rapid increase in demand for data transmission services that has happened in recent years has enforced the need to modernize the existing telecommunications infrastructure. Such a venture, however, not only means huge expenses related to the purchase of dedicated hardware, but also problems related to a swift transformation. In order to cope with the aforementioned challenges, telecommunication service providers (TSPs) have started adopting network functions virtualization (NFV) technology [1]. By implementing traditional network functions in software and running them inside virtual machines (VMs) on the hardware layer, they managed to satisfy the requirements, reduce operating expenses, and achieve the agility level of modern IT companies.

Example areas of NFV adoption are shown in Fig. 1 [2]. Among the use cases defined by the European Telecommunications Standards Institute (ETSI), the following two are the most widely used. The virtual customer premises equipment (vCPE) use case defines all types of network services that are typically run on the network edge, such as firewalls or routers. In turn, the virtual Evolved Packet Core (vEPC) use case covers all services required to converge voice and data on core fourth generation (4G) and 5G networks. While vCPE does not have high performance requirements and can easily be implemented as virtual network functions (VNFs), vEPC is mainly driven by performance. Therefore, the implementation of vEPC based on traditional VMs has been causing problems to TSPs for years.

In the search for solutions that can meet the performance requirements of vEPC, TSPs have begun exploring alternative technologies. One of the interesting solutions proposed recently is container network functions (CNFs) [3]. Such services run inside containers rather than VMs, which makes them lightweight and helps to achieve almost bare metal performance. However, the use of containers for NFV implementation purposes entails a number of limitations, such as security and isolation problems resulting from the use of a shared kernel [4].

In order to meet the performance requirements of vEPC, while providing the security level needed by all VNFs, unikernel network functions (UNFs) are proposed and presented in this article. Such network functions are implemented in unikernels and can run either as VMs or directly on the hardware layer. UNFs achieve very good performance results, which makes them a serious competitor to container technologies. The article proceeds with a brief overview of the unikernel technologies and follows with UNF presentation. The experimental results are shown at the end.

UNIKERNELS

A unikernel is a specialized, single-address-space machine image constructed by using library operating systems [5]. When constructing a unikernel, the minimal set of libraries required to run an application on specific hardware is selected from a modular stack. These libraries are later compiled together with the application code into an image that can be run either on the hypervisor or directly on the hardware layer. In this way, unikernels eliminate the overhead introduced by traditional operating systems [6].

Figure 2 summarizes the resource isolation methods used in the technologies discussed. Both machine containers and process containers run directly on the kernel layer, but the images they are using and the container construct are different. Similarly, what distinguishes regular VMs from unikernel-based VMs is the image construct, which in the case of unikernels does not introduce a division into kernel and service parts. Each kernel and image is unique, hence the name unikernel. Unikernels facilitate specialization of services, which is important due to the growing popularity...
of microservices. Each application runs inside its own VM. Thanks to the lightweight nature of unikernels, this solution scales out and allows running multiple VNFs in parallel.

**CONTAINERS CRITICISM**

Over the past few years, TSPs have started exploring container technologies as a potential replacement of traditional VMs. So-called CNFs, however, have a number of limitations. The following subsections briefly describe the most important ones.

**PERFORMANCE**

It is claimed that the main advantage of containers is their performance. Containers run directly on the kernel layer, eliminating the overhead introduced by a hypervisor and the virtual hardware layer. This results in low resource consumption and very good performance results. However, unikernels achieve comparable results, as shown in the Experimental Results section. This is because of the decoupling of the application not only from other services, but also unnecessary libraries, which partially compensates the overhead introduced by the hypervisor and the virtual hardware layer. Therefore, other factors, such as security, isolation, and minimization, become decisive when choosing a technology for NFV implementation.

**SECURITY**

One of the biggest limitations of containers is the security level they provide. Containers are less secure than VMs. While VMs rely on a hypervisor, containers use cgroups and namespaces, which are kernel features. This means that resource isolation is provided directly by the kernel. If one container gets compromised, other containers are impacted too. For example, a compromised container can start utilizing all available network resources. In turn, unikernels run as VMs, benefiting from full resource isolation provided by a hypervisor and the virtual hardware layer. Moreover, unikernel images do not contain POSIX tools, so even if the application is exploited, it is very difficult to compromise the underlying operating system, not to mention compromising the host. Obviously, a potential attacker could still first compromise a vulnerable hypervisor and then exploit unikernels running on it, but this applies to containers too. As security requirements of vEPC are almost as important as performance requirements, this positions unikernels as a perfect candidate for NFV implementation.

**ISOLATION**

Another disadvantage of using a shared kernel is isolation. Any change to the kernel, any upgrade, or even loading and unloading modules affects all running containers. In turn, unikernels can either run as VMs or directly on the hardware layer, benefiting from hypervisor-based or full isolation. Therefore, they are more resilient to such changes. It is only important to ensure that the changes do not affect the hypervisor or the underlying hardware.

**MINIMIZATION**

Although containers have significantly improved applications’ minimization ability, they still carry the overhead introduced by traditional operating system tools. While CNFs are usually implemented based on regular cloud images, such as Ubuntu, most of the tools that are part of the image are not used by CNFs. In turn, unikernels provide minimization up to an absolute limit. This is because unikernels contain only the application code and necessary libraries. As a result, unikernels are usually smaller than containers. For example, an image size of a firewall with 1,000,000 rules, discussed in the Experimental Results section, is only 30 MB for a unikernel, while it is 133 MB for an Ubuntu-based container.

**BUSINESS INVESTMENTS**

Finally, it is important to consider the business aspects of CNF adoption. In recent years TSPs have invested billions of dollars in NFV infrastructure (NFVI) based on OpenStack [7] and other similar cloud platforms. Although such solutions
In order to combine the benefits of using container and virtualization technologies, Unikernel Network Functions are proposed and presented in the following part of the article. It is shown that using unikernels for network functions implementation purposes allows comparable performance results to leading container technologies to be achieved while providing the security level of VMs.

**Unikernel Network Functions**

In order to combine the benefits of using container and virtualization technologies, UNFs are proposed and presented in the following part of the article. It is shown that using unikernels for network functions implementation purposes allows comparable performance results to leading container technologies to be achieved while providing the security level of VMs.

**Fit for Purpose**

The advantages of UNFs result from the simplicity of their implementation in unikernels. This is because network functions use kernel libraries to a limited extent. For example, a network function implementing the firewall service uses only those libraries that allow the network function to interact with a network interface card (NIC) and interpret the TCP/IP stack. The vast majority of resources are consumed by the application itself. Libraries responsible for an interaction with block devices, graphic cards, and so on become unnec-

---

**Adaptability**

As shown in the next section, it is very easy to run UNFs from an existing image; however, creating the image is usually not that easy. Most of the existing unikernel projects require the application code to be written in C. This tends to be problematic for DevOps engineers, who prefer using high-level programming languages, such as Python or Ruby. Adding a new firewall rule is no longer as easy as executing a shell command. Fortunately, there are unikernel projects that provide a translation from the application code to C.

An interesting example is the IncludeOS project [10]. It provides a configuration language called NaCl that allows defining network functions in the form of JavaScript Object Notation (JSON) files. This tool fits much better for people with a DevOps background and eliminates the blocker for swift UNFs adoption. Another interesting unikernel project is the Unikernel Based on Linux (UKL) which was released in November 2018 [11]. UNFs based on UKL could not only be adopted very easily, but would also benefit from

---

**Figure 2.** Resource isolation methods.

**Figure 3.** Example types of unikernel functions.
the maturity of the Linux kernel and its community of 15,000 people.

**Use Cases**

UNFs can run on either VMs or bare-metal machines. These will be referred to as virtual UNFs (VUNFs) and bare-metal UNFs (BUNFs), respectively, in the rest of the article. Each of these variants fits into different NFV use cases defined by ETSI and has its own advantages.

**Virtual Machines**

In the first case, UNFs are run on VMs. A cloud controller such as OpenStack can be used to automate and orchestrate the process. VUNFs achieve comparable performance results to network functions implemented in traditional virtualization and container technologies, as shown in the next section of the article. In combination with the small image size and relatively low RAM consumption, this allows them to be successfully used in edge devices of the telecommunications infrastructure. VUNFs fit perfectly into the vCPE [12] and mobile edge [13] use cases.

**Bare Metal**

UNFs can, however, run on bare-metal machines too. In this case, metal as a service (MAAS) can be used as a cloud controller [14]. The UNF image is launched directly on the hardware layer, which in theory should allow even better performance results to be achieved. This is important, as high performance is required by the vEPC, virtual IP multimedia services (vIMS), and virtual provider edge (vPE) use cases [12]. BUNFs could be used in any type of hardware that is located in the network core and is not able to run a full Linux kernel. This is a huge advantage of BUNFs over network functions based on traditional virtualization and container technologies.

**Experimental Results**

In order to test the performance of VUNFs, the following experiment was carried out. How the basic performance parameters [15] of a firewall service change depending on the number of rules was examined. The firewall service was implemented in three different technologies — KVM, Docker, and IncludeOS — each of which is the leading technology in the field of virtualization, containers, and unikernels, respectively. The Ubuntu image and the iptables software were used to build the firewall service for KVM and Docker, as they are the most popular cloud image and the most popular firewall software for Linux, respectively.

The test lab consisted of three bare-metal nodes: Sender, Firewall, and Receiver, which were connected in a chain with a dedicated 1 Gb/s network. In order to measure TCP throughput, an iperf session was initiated between the Sender and the Receiver. In turn, to measure TCP requests per second, a netperf session was initiated. Finally, ICMP latency was measured with ping. For TCP throughput and TCP requests per second, the measurement lasted for 30 s. For ICMP latency, an average of 100 measurements was drawn.

The results of the experiment are shown in Fig. 4. For the axis of ordinates, a logarithmic scale was used. The value of a given performance parameter was measured for the number of firewall rules of 1, 10, 100, 1000, 10,000, 100,000, and 1,000,000, respectively. On the basis of the graphs presented, it can be observed that the firewall service implemented in IncludeOS achieves better performance results than those implemented in KVM and Docker. This is particularly evident in the case of TCP requests per second and ICMP latency.

IncludeOS deals much better with firewall services with a large number of rules. In the case of a small number of rules, all three technologies achieve similar results. This is because firewalls with fewer rules have less impact on the statistics of the network traffic that is passing through them. Therefore, the link throughput starts playing the primary role.

**Limitations and Future Work**

When considering the benefits of using UNFs, it is also worth mentioning their limitations. During the experiment, it was observed that building firewall images with more than 100,000 rules takes a lot of time. It took more than a week to build a firewall with 1,000,000 rules. In addition, it was observed that the RAM usage during building a firewall image reaches about 6 GB/100,000 rules. However, the above problem applies only to IncludeOS and does not mean that other unikernel projects suffer from similar limitations. In addition, firewalls with more than 100,000 rules are rare. All observations were reported to the IncludeOS project team.

Further work on vUNFs should focus on conducting detailed performance tests with regard to other network functions than the firewall and a larger range of alternative technologies. It would also be worthwhile to conduct an experiment involving the launch of BUNFs. Examining the performance of such network functions would not only confirm the legitimacy of using UNFs...
for bare metal clouds, but would also provide the foundations for such solutions.

**Conclusions**

In the last few years, TSPs have begun exploring container technologies in search of better performance results than those achieved by traditional VMs. Container technologies, however, have a lot of limitations resulting from the use of a shared kernel, such as security and isolation problems. In order to solve these problems, unikernel network functions have been proposed and presented in this article. Such network functions can either run on a virtual machine cloud or a bare metal cloud, which makes them suitable for all NFV use cases. By using modern unikernel technologies, such as IncludeOS and UKL, UNFs can be adopted by regular DevOps teams. One of the main advantages of unikernels over containers is the high level of security they provide, which results from the use of the hypervisor and the unikernel construct. Along with the satisfying performance results achieved by unikernels, it makes them an excellent tool for implementing the NFV technology of the future.

**Acknowledgments**

The author would like to thank Kristian Jerpetion and other members of the IncludeOS project for their help in solving problems with the IncludeOS and NaCl software. Thanks to their support, the research was carried out very smoothly, and all problems encountered were quickly solved.

**References**


**Biography**

**Tytlus Kurek** (tkurek@itsteer.com) received his M.S. degree and Ph.D. degree in telecommunications from the AGH University of Science and Technology, Krakow, Poland, in 2011 and 2018, respectively. His current research interests focus on network functions virtualization, 5G networks, container network functions, open source MANO, fast packet processing, and unikernels.