Comparing Virtualization Solutions for NFV Deployment: a Network Management Perspective

Lucas Bondan*, Carlos Raniery Paula dos Santos†, Lisandro Zambenedetti Granville*
*Institute of Informatics – Federal University of Rio Grande do Sul
†Department of Applied Computing – Federal University of Santa Maria
Email: {lbondan, granville}@inf.ufrgs.br, csantos@inf.ufsm.br

***Abstract*—Network Functions Virtualization (NFV) is a paradigm designed to promote service agility and able to quickly generate revenue, thus encouraging competition among companies in the computer network industry. Besides the advocated benefits of NFV, management requirements should be properly taken into account. The choice of a particular NFV-based technology must consider its management requirements. However, there is still no evaluation of virtualization solutions providing an in-depth analysis from the management point-of-view. This paper presents a performance analysis of three prominent virtualization solutions: ClickOS, CoreOS, and OS^v. Our results place ClickOS and CoreOS as the best solutions regarding boot time, response time, and memory consumption. Moreover, based on the results obtained for each performance metric, we provide a broad discussion about the effectiveness of each virtualization solution in fulfilling qualitative management requirements.**

***Keywords*—NFV, network management, virtualization solutions, performance analysis, management requirements**

I. Introduction

Network Functions Virtualization (NFV) [1] is a networking paradigm where network functions (*e.g.,* firewalls, load balancers, NATs), often requiring dedicated devices, are deployed on virtual servers running on commodity hardware. Introduced by the European Telecommunications Standards Institute (ETSI), NFV complements the more established paradigm of Software-Defined Networking (SDN) [2]. While SDN focuses in decoupling the network control plane, NFV is concerned with moving network functions from dedicated hardware appliances into software running on standard commercial off-the-shelf (COTS) servers [3]. Different than SDN, however, NFV is much less mature, and more concrete developments are just emerging.

As in any novel networking technology, the proper management of its functional aspects is fundamental for its adoption. Unfortunately, the network management discipline is not in the focus of current NFV efforts. We argue, however, that the identification of concrete management functions cannot be neglected. As such, we identified a set of NFV management requirements from the perspective of the network operator [4]. By listing the difficulties faced using a virtualization solution for an NFV deployment, we evidence the importance of the virtualization solution choice for the network management.

Once identified the management requirements, network operators must choose the most appropriate technology to fulfill these requirements. Recently, some efforts from industry and academia led to the development of virtualization solutions aimed to run Virtualized Network Functions (VNFs) [5]–[10]. Although some of these solutions are not directly designed for NFV, they present essential virtualization properties, which turns them promising NFV enablers. The right choice of the virtualization solution is crucial for the network operator, since it has direct implications on the network performance and management support. Besides those efforts and to the best of our knowledge, no investigations were conducted to show how effective virtualization solutions are with regard to management requirements. We argue that the analysis of different virtualization solutions is fundamental to help network operators interested in adopting NFV on their environments.

In this paper, we analyze three different virtualization solutions: ClickOS [5], CoreOS [6], and OS^v [7], which are open source solutions frequently considered NFV enablers. To perform the comparison, we selected the NFV management requirements most appropriate to be analyzed using quantitative metrics, *i.e.,* virtual machines (VMs)/containers instantiation, VNF deployment, VM/container & VNF monitoring, and physical/virtual network functions coexistence. An experimental network setup was deployed using each virtualization solution, supporting their evaluation based on selected performance metrics. The main contributions of this paper are (*i*) a performance evaluation of different virtualization solutions and (*ii*) an in-depth discussion on the effectiveness of each solution regarding the management requirements.

The remaining of this paper is organized as follows. In Section II, we present a background and related work on NFV and emerging solutions. The methodology and the experimental scenario applied to evaluate the performance metrics are presented in Section III. In Section IV, we show the results obtained in the experimental evaluation. In Section V, the results achieved are discussed, relating the performance metrics to the management requirements. Finally, we present the conclusions and perspectives of future work in Section VI.

II. Background and Related Work

NFV is a new networking paradigm where functions (*e.g.,* firewalls, DNS, IDS), traditionally performed by dedicated physical devices, are virtualized and deployed on commodity hardware. Initially, NFV not only enables to reduce both capital and operational expenditures (CAPEX and OPEX) by virtualizing network functions (NFs), but it is also about business and service agility, and the ability to quickly generate revenue, thus encouraging competition among companies in the computer network industry. Moreover, in academia NFV represents a way to develop innovative solutions, by simplifying the design and deployment of network functions [11].

To promote NFV adoption, ETSI released a series of documents detailing NFV concepts. We highlight the NFV Management and Orchestration (MANO) document, which deals with these aspects in the context of NFV [12]. ETSI MANO aims to propose an architecture for NFV management and orchestration, defining some reference points and an information model to manage important data regarding operational NFV elements (*e.g.,* virtual links, VNFs, VMs). However, the reference points defined in ETSI MANO are too vague to be implemented in practice, and the information model seems like a set of abstract requirements for building a model, insufficient for the management of an NFV deployment in practice.

Similarly to ETSI, the Internet Research Task Force (IRTF) is also concerned with promoting NFV adoption. IRTF hosts the NFV Research Group (NFVRG) focused on issues related to NFV environments. NFVRG already produced a set of Internet-Drafts dealing with policy-based management, service verification, resource management, among others research topics[1]. Despite the efforts from both ETSI and IRTF, we identified a lack of practical analysis over virtualization solutions from the management point-of-view. This analysis can certainly help to refine the reference points (*i.e.,* interfaces and functional blocks) and data model proposed in ETSI MANO, highlighting practical necessities of virtualization solutions.

In a previous work, we took a first step in identifying key management requirements of NFV, by deploying a network setup request using a virtualization solution [4]. Now, we advance the research on management requirements, evaluating different virtualization solutions from the management point-of-view. NFV solutions are emerging, as the case ClickOS, CoreOS, OS^v, NetVM (or openNetVM), CirrOS, Alpine Linux, among others [5]–[10]. In this paper, we concentrate our evaluation in three solutions: ClickOS, CoreOS and OS^v.

Our choice was based on three main aspects. The first one is code availability, since that all solutions chosen are open-source and available for download with no cost. Next, these solutions still are under development, so developers are constantly improving and/or fixing issues to guarantee stability for their solutions. Finally, all selected solutions are in accordance with the NFV Virtualization Requirements document published by ETSI [13], which presents low-level requirements for NFV adoption. In our investigations, ClickOS, CoreOS, and OS^v differ from the others by presenting all these aspects together, *i.e.,* in the same solution. We present each one of the selected virtualization solutions in details as follows.

A. ClickOS

ClickOS is a Xen-based software platform optimized for fast network packet processing and designed to support typical network requirements such as high throughput, low latency, and isolation [5]. ClickOS consists of the Click Modular Router running on top of a minimalist Linux [14]. Using ClickOS, network functions are provided by Click libraries, which allows implementation of complex network functions processing configurations by using simple, well-known processing elements. In addition, ClickOS VMs present small sizes and memory (with basic Click libraries, ClickOS VMs are 6 MB in size).

[1]https://datatracker.ietf.org/rg/nfvrg/documents/

B. CoreOS

CoreOS allows the easy deployment of a wide range of isolated functions using Linux containers (LXC) virtualization [6]. LXC provides similar benefits as complete virtual machines (or full virtualization) but focused on functions instead of entire virtualized hosts. Container-based virtualization allows code to run in isolation from others but safely share the machine resources. Moreover, it is not necessary a dedicated Linux kernel or hypervisor for managing containers, thus presenting almost no performance overhead. In container-based virtualization, the main element is the container engine responsible for containers lifecycle management. In CoreOS, VNFs can be represented by Linux functions (*e.g.,* $iptables$ for proxy, firewall, and NAT; $Snort$ for IDS Sensor) running on the same server but with isolated memory spaces.

C. OS^v

OS^v is an open source operating system based on the library OS design [7]. Although its standard version is distributed based on the QEMU Kernel Virtual Machine (KVM), OS^v can be managed using other hypervisors (*e.g.,* Xen, VirtualBox) with a minimal amount of architecture-specific code. Moreover, OS^v is flexible, able to run functions designed in different languages, such as C/C++ and Java. In OS^v, each VM runs a single function with its dedicated copy of the library OS. Library OS attempts to address performance and functionality limitations in functions that are caused by traditional operating systems abstractions.

Different works can be found in the literature performing comparisons among virtualization solutions. Estrada *et al.* [15] compared the KVM hypervisor, the Xen para-virtualised hypervisor, and LXC, also performing changes over these solutions in order to improve the runtime to solve sequence alignment problem in bioinformatics. In another work, Felter *et al.* [16] conducted an analysis of the two most common types of virtualization available: VMs and containers based. In their analysis, KVM hypervisor and LXC were compared regarding input/output (IO) operations. Finally, the work of Reddy and Rajamani [17] presents a comparison of different operating systems over the same hypervisor (KVM) in a cloud environment, using performance metrics like CPU usage, memory management, and network communication.

Our focus in this work is neither on the implementation and evaluation of the descriptors proposed by ETSI MANO nor into only comparing different virtualization solutions as the works aforementioned. Our main objective is mapping lower level performance metrics into management requirements, based on the comparison of different virtualization solutions.

III. Methodology

An in-depth discussion about NFV management requirement is fundamental. More specifically, the list previously investigated must be revisited, considering different virtualization solutions [4]. Such discussion is provided in the following.

A. Management Requirements Classification

Management requirements can be observed from two perspectives. The (*i*) **qualitative** analysis requires a **subjective** evaluation, with network operators being interviewed or observed when using a virtualization solution. In the other hand, the (*ii*) **quantitative** analysis is performed by measuring **objective** performance metrics from the related systems. In the context of this paper, we applied the second approach, at the same time we limited the set of requirements for the following:

VMs/containers Instantiation – Once properly configured the VNF servers, network operators have to instantiate the VMs or containers hosting VNFs. Depending on the strategy and technology used, each VNF needs a dedicated VM or container to host it, probably resulting in wide ranges of instances over the NFV Infrastructure (NFVI);

VNF Deployment – Deploying VNFs in the NFVI involves both the configuration and placement of VNFs. The VNF placement is a well-known problem in the NFV literature [18], [19], with solutions focused on optimizing the placement of VNFs to avoid unnecessary migrations over the NFVI;

VM/container & VNF Monitoring – The monitoring of both VMs/containers and VNFs is essential to guarantee the proper service operation. In the context of NFV, the monitoring activity includes the VNFs status acquisition, which may require the instrumentation of VNFs to expose their internal state through management interfaces;

Physical and Virtual NFs Coexistence – The management of both physical and virtual network functions should mostly be transparent for network operators. One exception to this rule is the case where computing resources allocated to VNFs dynamically change. Network operators must be aware of the underlying workload before making any changes to the VNFs.

We performed a quantitative analysis due to the possibility to recreate the evaluated scenario for different virtualization solutions, thus giving us a better perspective on their operation. In order to evaluate the effectiveness of each virtualization solution considering the presented management requirements, we selected three performance metrics: boot time, response time, and memory consumption, which are directly related to each other due to their quantitative nature. In Section V, we provide a detailed discussion regarding the performance metrics and its mapping into the selected management requirements.

B. Evaluation Scenario

We first must configure the VNF servers which are responsible for hosting VMs or containers that will run VNFs. In our experiments, each server is an AMD 3.6 GHz with 4 GB of memory, placed according to Figure 1.

Boot time and memory consumption were measured directly from the VNF servers using scripts running on the Server OS. To measure the response time, however, we deployed a VNF acting as network proxy on the respective Server OS for each virtualization solution. This VNF is responsible for forwarding packets from Host A to B and the reply from Host B to A. In Host A, a script is responsible for sending a request, while another one will calculate the elapsed time between Host A send a request and identify the reply sent by Host B. By measuring the elapsed time between a request and its respective reply, it is possible to evaluate the response time inserted by the VNF running in the VNF server. In Algorithm 1 we present the pseudo-code detailing the execution sequence of our proxy.

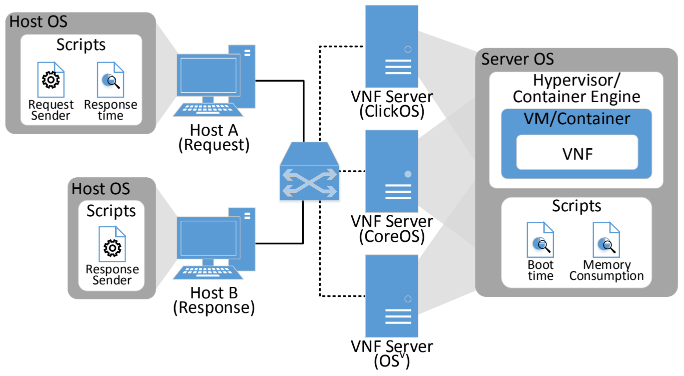


Fig. 1. Evaluation Setup

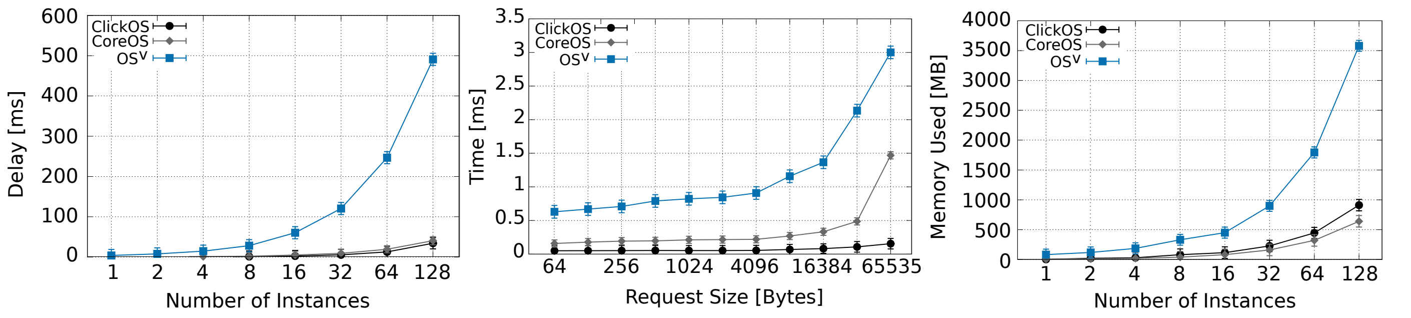
Algorithm 1 Proxy Operation Pseudo-code

```
1: while $TRUE$ do
2:     $incoming\_pkt \leftarrow listenInterface(eth0)$
3:     if $incoming\_pkt \neq NULL$ then
4:         $outgoing\_pkt \leftarrow incoming\_pkt$
5:         if $incoming\_pkt.src\_addr == host\_A$ then
6:             $outgoing\_pkt.dst\_addr \leftarrow host\_B$
7:         else if $incoming\_pkt.src\_addr == host\_B$ then
8:             $outgoing\_pkt.dst\_addr \leftarrow host\_A$
9:         end if
10:    end if
11:    $sendPacket(outgoing\_pkt, eth0)$
12: end while
```

The proxy starts to monitor its network interface ($eth0$) until an incoming packet be received (lines 2 and 3). Then, the incoming packet is copied as an outgoing packet to be forwarded to the correct destination (line 4). Next, the proxy verifies the source of the incoming packet: if the packet is from host A, the destination address is changed to host B (lines 5 and 6). Otherwise, the destination receives host A address (lines 7 and 8). Once configured the new destination, the proxy forward the incoming packet (line 11). Our proxy operates in a limited way purposely, since the objective is to evaluate the performance of the virtualization solutions. Thus, the proxy does not add any significant packet processing time, enabling a clear evaluation over the time spent by each solution.

The next step is the setup of the servers responsible for host VNFs. We first configure the operating system (Server OS), which will provide routines needed by the hypervisor/container engine to access the VNF server functionalities. Next, for each virtualization solution a different kind of VM or container is instantiated to host VNFs, which are implemented according to the libraries provided by the virtualization solution during the creation of a VM/container, *i.e.,* the VNF description/configuration language. For each virtualization solution, a different set of configurations was applied, explained in in the following.

ClickOS: a Xen hypervisor running on top of a Linux-based system is deployed in the VNF server. ClickOS images are responsible for hosting specific Click configurations, *i.e.,* one VNF per ClickOS image. A set of network elements available on Click libraries supports the creation of VNFs. Description files containing network elements for the functions are interpreted and executed by ClickOS. In this way, the proposed



(a) Average boot time (b) Average response time (c) VNF server memory consumption

Fig. 2. Evaluation results comparison

proxy function (explained in Algorithm 1) was implemented using some Click elements, making a description file inserted in the ClickOS images. The VNF management in ClickOS is only possible using Cosmos, a tool developed to allow the communication between user and ClickOS domains.

CoreOS: a Linux-based server was used to host containers for each VNF. The main building block of CoreOS is the Docker engine responsible for guarantee VNFs running isolated from each other. In our setup, VNFs in CoreOS are represented by Linux functions installed on the CoreOS server. In our experimental setup, we used a CoreOS container running *iptables* as the proxy VNF. Two rules were configured on *iptables*, operating like the proxy pseudo-code presented in Algorithm 1: one rule to forward packets coming from host A to B and another one for the opposite direction.

OS^v: a KVM hypervisor was configured on a Linux-based server since it is the standard hypervisor for OS^v management. Each OS^v VM was set to host one particular VNF, which is the recommended behavior by OS^v developers. Once designed, VNFs source codes are compiled and transferred to OS^v VMs, running immediately after the VMs startup. Each OS^v VM can be accessed using a shell, easing the access and operation control of VNFs inside them. An OS^v proxy image available at the OS^v project repository was used to instantiate the proxy. Algorithm 1 was implemented in the C programming language using Socket RAW libraries and deployed inside OS^v images.

For all virtualization solutions, configuration templates were used to specify VMs characteristics (*e.g.,* processing power, storage, memory, and networking). In this work, we configured the minimum recommended amount of resources needed to start a simple VNF to reply network requests: for ClickOS, one virtual CPU with 12 MB of memory; for OS^v, one virtual CPU with 64 MB of memory. In the CoreOS configuration, however, the processing and memory available for containers are based on the total amount of these resources on the server due to its container-based nature.

IV. Evaluation Results

In this section, the results obtained in the evaluation of each performance metric are presented in the following order: boot time, response time, and memory consumption. All measurements were repeated until guarantee a minimum confidence interval of 95%.

A. Boot Time

Fast boot time is important in any NFV deployment since that VNFs must be available as soon as possible. For example, when a VM or container that hosts a VNF crashes, a new one must be started-up as fast as possible to avoid losses in packet processing. The boot time evaluation of each virtualization solution was performed using a script to measure the time needed by a VM or container to start its operation and response to a network request (*i.e.,* send an ICMP Echo Reply in response to an ICMP Echo Request packet). It represents the effective time the VM/container needs to be operational in the network. The script used can be configured to measure the boot time for one VM/container or a group of VMs/containers, starting all the instances and calculating the total amount of time to receive the response from all VMs/containers. The results of this evaluation are summarized in Figure 2(a).

OS^v boot time increases faster than the others. We credit this poor performance to the wide number of libraries and commands that must be loaded during OS^v VMs startup. Moreover, we can conclude that ClickOS has the fastest boot time among the virtualization solutions evaluated – less than 0.25 milliseconds for 1 VM and close to 50 milliseconds with 128 VMs. This good performance, however, has a drawback: the optimizations made turned ClickOS not very user-friendly. For example, there is not a shell to access ClickOS VMs, requiring use the Cosmos tool to have a minimal level of access to ClickOS VMs domain. CoreOS also presented good performance results, being just some milliseconds slower than ClickOS, since that the functions are already installed on the CoreOS server, and the container engine (*i.e.,* Docker) needs only to manage server resources sharing among containers. Thus, CoreOS appears as a good alternative to ClickOS, by presenting a very similar performance, with the benefit of having friendly access to containers domain.

B. Response Time

The main objective here is to analyze how much time each virtualization solution spends to receive, process, and response packets. A simple VNF was implemented for each virtualization solution, acting as a proxy (*i.e.,* receiving packets from a host and forwarding them to another) according to the experimental scenario presented in Section III-B. An important point to highlight is that all virtualization solutions were analyzed without any additional network improvements, providing a fair evaluation of each virtualization solution.

Analyzing Figure 2(b) we can conclude that ClickOS presents the smallest response time for all packets sizes, being around 0.16 milliseconds in the worst case (*i.e.,* packets 64 KB sized). We credit this performance to the improvements made to directly map packet buffers into ClickOS VMs memory space. Very close to ClickOS performance we found CoreOS – under 0.5 milliseconds until packets reach sizes bigger than 16 KB. OS^v presented the worst results, showing a response time close to 3 milliseconds with the maximum packets size analyzed, twice as much time as the worst case of CoreOS and around eight times than the worst case of ClickOS.

C. VNF Server Memory Consumption

We measured the available memory before and after VMs/containers instantiation. We varied the number of VMs/containers instantiated from 1 to 128. Inside each host, the same function used in the boot time evaluation was instantiated. Despite the available memory for each VM can be configured using predefined templates, the total amount of memory used for each solution is also influenced by the hypervisor/container engine. The results obtained are presented in Figure 2(c).

Despite all solutions presented the same behavior, the smallest consumption was presented by CoreOS, with less than 700 MB used with 128 containers instantiated. We credit this performance to the container-based virtualization approach used by CoreOS, where the server memory is allocated according to the necessity of the functions running in each container. Very close to CoreOS performance we found ClickOS, with less than 1000 MB (or 1 GB) of memory used in the last case, which is a good performance considering the full virtualization approach used by ClickOS. Finally, OS^v presented the worst memory consumption, using more than 3500 MB to instantiate 128 VMs. We credit this behavior to the complexity of OS^v VMs, since that each VM is a single function with its own copy of OS^v libraries, that should be loaded with each VM. Despite different evaluations, our results for ClickOS and OS^v were very close to those achieved by their authors [5], [7].

V. Results Discussion

In this section, we discuss in details the relationship of the performance metrics evaluated in the previous section with the management requirements presented in Section III.

A. VMs/containers Instantiation

The process of instantiating a VM or container must be performed as fast as possible, in particular when the VNF to be hosted on it is part of an entire service chaining (or VNF-FG). Further, in order to instantiate a new VM or container, the network operator (or the placement algorithm) must be aware of how much memory will be required and, consequently, the amount of available memory in the VNF server. If the memory required by the new VM/container is high enough to compromise others VMs/containers or even the VNF server operation, another VNF server with sufficient memory must be selected. For these reasons, we argue that boot time and memory consumption must be considered for the VMs/containers instantiation management requirement.

Based on the results obtained, ClickOS and CoreOS presented best boot time and memory consumption performances, respectively. However, OS^v has a differential: a tool called Capstan allows the network operator to access a wide set of OS^v VMs images available in the project repository, facilitating the deployment of new VMs. In this way, CoreOS is a good choice regarding performance, due to its fast boot time and low memory footprint. However, if the priority is more flexibility in the overall process of VM instantiation, OS^v appears as an alternative, able to ease this process with the drawback of worse memory consumption.

B. VNF Deployment

The VNF deployment is related to two tasks performed over VNFs. The first one is the VNF location, also known as VNF placement problem, widely investigated in NFV literature [18], [19]. The VNF placement problem consists in the distribution of VNFs in a pull of servers according to some criteria. One of the main criteria to be considered in this problem is the resource utilization. The second task involved in the VNF deployment is the configuration/reconfiguration of VNFs. It must occur as fast as possible to avoid information loss or even service outages. Moreover, depending on the strategy used more memory may be required for VNF reconfiguration [20]. For this reason, memory consumption is as important as boot time regarding VNF deployment.

ClickOS and CoreOS are the best virtualization solutions regarding boot time and memory consumption, characterizing them as good choices in terms of VNF configuration/reconfiguration, with a small advantage for ClickOS regarding boot time, and for CoreOS concerning memory consumption. However, despite results showed CoreOS as the best choice regarding memory consumption, there is a drawback in using CoreOS: it is not suitable for migration due to its container-based virtualization. Then, we believe CoreOS is the best choice regarding memory consumption and suitable for use in static scenarios, *i.e.,* network scenarios where VNF migration is not needed. However, in cases where VNF migration is often performed (*e.g.,* for load balancing), we recommend the use of ClickOS, which is more suitable for migration with memory consumption results close to the values presented by CoreOS.

C. VM/container & VNF Monitoring

VNFs must be continuously monitored to keep the network working properly. In this way, the network response time of VMs and containers must be as fast as possible, to quickly obtain information from their operation, such as the current VNF location and its status. When the monitored VNF is a part of a VNF-FG, delays in the VNF response may impact the entire service provided by the VNF-FG. For this reason, the network response time of VMs/containers is an important metric to reflect the VNF monitoring requirement.

In our results, ClickOS appears as the best choice in terms of response time and, consequently, covering the VNF monitoring requirement better than the others solutions. Moreover, ClickOS is very suitable for VNF migration due to the small size of ClickOS VMs (approximately 12 MB), avoiding bottlenecks in the network paths during the migration and its full virtualization approach, turning the memory dump easily.

D. Physical and Virtual NFs Coexistence

Memory consumption is a key metric to reflect how good a virtualization solution is regarding VNF coexistence. If the amount of memory needed to instantiate and manage VMs or containers is too high, probably a few number of VNFs may coexist in the same VNF server. As well known by network operators, the network elements management is easier when they are close to each other. Another important metric is the network response. In cases where VNFs should communicate with each other, forming a VNF-FG, they need to fast reply to others VNFs of the VNF-FG. Moreover, delays in the network response from a VNF belonging to a VNF-FG may affect communication among VNFs composing another VNF-FG.

In our evaluation, CoreOS presented the best results regarding memory consumption while ClickOS is the best virtualization solution regarding network response time. Considering the similarity in the results of both ClickOS and CoreOS concerning memory consumption, we believe that ClickOS is the best choice regarding VNF coexistence, since that the network response of ClickOS is better than CoreOS.

We can conclude that there is no definitive virtualization solution for all evaluated performance metrics and, consequently, for all management requirements considered in this work. Despite the best performance presented by ClickOS, the creation, access, and management of ClickOS VMs is a hard task. CoreOS provides a simple way to create and access containers when compared to ClickOS, but container-based virtualization may impose some difficulties related to VNF migration due to its shared memory approach. Finally, OS^v appears as an alternative, by given up performance to achieve easy creation, access, and management of VMs. The choice of one or another virtualization solution depends on the network operator needs and the network scenario.

VI. Conclusion and Future Work

In this paper, we provided a comparison among emerging NFV enablers: ClickOS, CoreOS, and OS^v. We refined and classified management requirements from the literature in two classes, selecting quantitative ones to be evaluated based on three different performance metrics: boot time, response time, and memory consumption. Our main objective is to help network operators to choose a virtualization solution to for their NFV deployment.

Our results show that ClickOS and CoreOS are superior to OS^v , with advantage for ClickOS in terms of boot time ($18,61\%$ faster than CoreOS in the best case) and response time (approximately 10 times faster than CoreOS for packets 64 KB sized). Nevertheless, CoreOS required $42,34\%$ less memory than ClickOS for 128 instances. However, MANO solutions should also consider the drawbacks using one or another solution in NFV environments. For example, in more dynamic scenarios where VNFs are often updated or reconfigured, solutions like OS^v could be an promising alternative, due its diversified images database.

Now, we plan to design a comprehensive management system for NFV, selecting the best virtualization solution to cover as many as possible the management requirements. We also plan to investigate orchestration mechanisms for NFV, working in a synergy among all the good practices proposed by ETSI and the management requirements evaluated in this work, proposing a unified design pattern for NFV MANO.

References

[1] M. Chiosi *et al.*, “Network Functions Virtualisation (NFV),” ETSI NFV ISG, White Paper, 2012, https://portal.etsi.org/nfv/nfv_white_paper2.pdf.

[2] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN: An Intellectual History of Programmable Networks,” *ACM Sigcomm Computer Communication*, vol. 44, no. 2, pp. 87–98, 2014.

[3] F. Risso, A. Manzalini, and M. Nemirovsky, “Some Controversial Opinions on Software-Defined Data Plane Services,” *IEEE SDN for Future Networks and Services (SDN4FNS)*, pp. 1–7, Nov. 2013.

[4] L. Bondan, C. R. P. d. Santos, and L. Z. Granville, “Management requirements for ClickOS-based Network Function Virtualization,” in *International Workshop on Management of SDN and NFV Systems (ManSDN/NFV) collocated with the International Conference on Network and Service Management (CNSM)*, Nov 2014, pp. 447–450.

[5] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, and M. Honda, “ClickOS and the Art of Network Function Virtualization,” *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[6] CoreOS: Open Source Projects for Linux Containers. Available at: https://coreos.com/. Accessed December, 2015.

[7] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har’El, D. Marti, and V. Zolotarov, “OSv—Optimizing the Operating System for Virtual Machines,” in *USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 61–72.

[8] J. Hwang, K. Ramakrishnan, and T. Wood, “NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms,” *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, March 2015.

[9] CirrOS: a Tiny OS that specializes in running on a cloud. Available at: https://launchpad.net/cirros. Accessed December, 2015.

[10] Alpine Linux. Available at: http://www.alpinelinux.org/. Accessed December, 2015.

[11] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network function virtualization: State-of-the-art and research challenges,” *IEEE Surveys & Tutorials*, 2015, to appear.

[12] J. Quittek *et al.*, “Network Functions Virtualisation (NFV) - Management and Orchestration,” ETSI NFV ISG, White Paper, 2014.

[13] M. Chiosi *et al.*, “Network Functions Virtualisation (NFV) - Virtualisation Requirements,” ETSI NFV ISG, White Paper, 2013.

[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click Modular Router,” *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[15] Z. J. Estrada, F. Deng, Z. Stephens, C. Pham, Z. Kalbarczyk, and R. Iyer, “Performance comparison and tuning of virtual machines for sequence alignment software,” *Scalable Computing*, vol. 16, no. 1, pp. 71–84, 2015.

[16] 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 (ISPASS)*, March 2015, pp. 171–172.

[17] P. Reddy and L. Rajamani, “Performance comparison of different operating systems in the private cloud with kvm hypervisor using sigar framework,” in *International Conference on Communication, Information Computing Technology (ICCICT)*, Jan 2015, pp. 1–6.

[18] H. Moens and F. D. Turck, “VNF-P : A Model for Efficient Placement of Virtualized Network Functions,” in *International Conference on Network and Service Management (CNSM)*, Nov 2014, pp. 418–423.

[19] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, “The dynamic placement of virtual network functions,” in *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[20] V. A. Olteanu and C. Raiciu, “Efficiently migrating stateful middleboxes,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, p. 93, Sep. 2012.