Computer Communications 184 (2022) 73–85

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

An experimental investigation of Round-Trip Time and virtualization

Assis T. de Oliveira Filho [*], Eduardo Freitas, Pedro R.X. do Carmo, Djamel H.J. Sadok, Judith Kelner

Networking and Telecommunications Research Group (GPRT), Federal University of Pernambuco, Brazil

ARTICLE INFO

Keywords:
Virtualization
Active network measurements
Cloud management
Performance metrics
KVM
Container

ABSTRACT

Virtualization is omnipresent in our networks and data centers. Little is known of its overhead, especially regarding the deployment of Virtual Network Functions and the use of existing network active measurement tools. This paper evaluates the impact on Round-Trip Time of parameters such as CPU affinity, the frequency of injection measurements echo packets, the type of virtual network driver, the use of CPU, I/O or network overload, and the number of background VMs. We compare RTT results using three virtualization technologies, namely, KVM, LXC, and Docker containers. To isolate the role of virtualization on packet timing overhead, we also include kernel-level measurement points in our experiments. We discover that the parameters differ in their impact on RTT, and that their effect changes according to how they are combined. The results reported in this paper allow a Cloud administrator to understand the overhead of virtualization technologies such as KVM, LXC, and Docker containers as well as the impact of virtual network drivers, disk I/O overhead, CPU processing and combinations therein, on network performance.

1. Introduction

Cloud Computing revolutionized the allocation of computational resources by providing on-demand and dynamic distribution of hardware components through software settings. At present, approximately 81% of IT organizations use cloud computing or have applications deployed in a cloud, according to a survey from IDG [1]. One of the motivations behind such widespread adoption is the low Total Cost of Ownership (TCO), enabling a quick Return on Investment (ROI) due to reduced capital (CAPEX) and operational (OPEX) costs. In addition, the availability of Cloud open-source software facilitates embracing the technology and simplifies its management at all three levels: infrastructure, platform, and software as a service.

The next generation of data centers, telecommunication architectures, and 5G networks are taking virtualization a step further into their novel architectural design. Not only do they adopt the virtualization of host resources, but also that of networking hardware. The emergence of Virtual Network Function (VNF) promises the *softwarization* of network functions that traditionally run over specialized and dedicated hardware. To put it more simply, network devices such as firewalls, switches or routers are now seen as virtual functions that can be started, migrated, replicated, and terminated just like any other computing service.

However, despite the inherent advantages, virtualization raises some performance drawbacks. This additional software layer introduces processing overhead required to execute virtual machines. In turn, this impacts network performance including that of VNF. In this work, we are particularly interested in the analysis of how virtualization affects network performance in scenarios like cloud computing data centers.

Cloud computing data centers host a large virtualized infrastructure dedicated to running applications, databases, storage services, web services as well as network functions. Most of these services make extensive use of disk I/O, network or CPU resources. High network performance remains a major requirement for the successful deployment of Cloud services. Cloud administrators should understand the benefits and limitations of existing virtualization technology alternatives and weight the impact of the use of competing operations such as disk I/O and CPU processing. This work presents performance evaluation results obtained from running experiments on a testbed especially created to evaluate the impact of virtualization on network performance. In order to make a conscious decision, a Cloud administrator must consider the overhead of virtualization technologies such as Kernel-based Virtual Machine (KVM), Linux Containers (LXC), and Docker containers. This paper reports network performance results from extensive testing that combines changing levels of disk I/O, network and CPU resources with the use of different virtualization technologies.

The rest of the paper is organized as follows: In Section 2, we give more detailed background information about network processing in virtualized systems. We also introduce KVM and Containers virtualization

[*] Correspondence to: Federal University of Pernambuco, Networking and Telecommunications Research Group (GPRT), Recife, Pernambuco, Brazil.
E-mail addresses: assis.tiago@gprt.ufpe.br, atof@cin.ufpe.br (A.T.d.O. Filho), eduardo.freitas@gprt.ufpe.br (E. Freitas), pedro.carmo@gprt.ufpe.br (P.R.X.d. Carmo), jamel@gprt.ufpe.br (D.H.J. Sadok), jk@gprt.ufpe.br (J. Kelner).
URL: http://www.gprt.ufpe.br (A.T.d.O. Filho).

https://doi.org/10.1016/j.comcom.2021.12.006
Received 16 June 2021; Received in revised form 7 December 2021; Accepted 8 December 2021
Available online 16 December 2021
0140-3664/© 2021 Elsevier B.V. All rights reserved.

method. Section 3 presents a description of the experiment setup, along with the testbed specification. Section 4 shows the obtained results, followed by Section 5 where we discuss related work. Finally, Section 6 reports our conclusions and final considerations.

2. Network processing in virtualized systems

Virtualization is the main underlying enabling technology in the design of Cloud Computing systems. It brings service scalability and on-demand elasticity. Its abstraction and logical decomposition of hardware resources such as CPU, memory, and network cards enable the dynamic slicing of resources for their on-demand allocation. On the other hand, this additional software layer has been known to cause considerable processing overhead, slowing down performance, especially in the networking domain [2,3]. More recently, lighter forms of virtual machines, known as containers, have been gaining more popularity and replacing full virtual machines. There are however several competing techniques with different approaches and objectives to choose from. The most popular among these are **Full Virtualization** and **Operating System (OS) Level Virtualization**. Depending on the requirements at hand, a Cloud manager needs to understand their pros and cons.

Full Virtualization consists of running a complete and unmodified guest OS within a host that already has a functional OS. Multiple guests co-execute, and their OS may be different from that of the Host and may even execute a different CPU architecture. The premise of full virtualization is to simply create an environment that supports the execution of a completely different and isolated OS inside an already existing one. The main function needed is the translation of CPU instructions that the guest will send to the Host. This is responsible for its translation to the actual language or format supported by the underlying hardware. This translation process is possible via the Virtual Machine Monitor (VMM) or Hypervisor. In other words, it coordinates the translation of processor instructions between the physical and virtual device. This translation usually executes over two steps. The first one is the binary translation itself of privileged instructions of the Virtual Machine (VM) to unprivileged ones in an equivalent block of the CPU [4]. Then, this is followed by the translated instructions being executed directly on the CPU. The VM's instructions that do not need privileged execution can run directly on unprivileged blocks without translation.

System Level or OS Level Virtualization is a type of virtualization that works at the operating system layer. It is a feature present in the operating system kernel that allows the presence of more than one isolated user-space instance. Thus, the OS kernel will run a single operating system and provide system functionality to multiple isolated partitions that replicate an actual server. These instances are called containers.

These containers use only the system resources and kernel processes to create the environment. Hence a container may only access content and devices assigned to it and does not have an overview of the physical environment outside its space. One of the main advantages of using containers is creating software packages that can easily be moved between different environments.

A commonly used Full VirtualizationVMM infrastructure is KVM. It runs inside the Linux Kernel and supports x86 hardware with virtualization capabilities. KVM uses all of Linux components like its memory manager and process scheduler to interact with the hardware and create virtual environments to execute the VMs. With regard to OS Level Virtualization platforms, Docker and Linux Containers (LXC) are the most widely deployed VMMs. They combine features provided by the kernel to create the isolated processes, namely "chroot", "namespaces", "cgroups" [5].

In addition, this work takes a particular look at the impact of the virtualization technology used to create virtual network cards and present them to the guest operating systems. As a result, the next Section 2.1 will discuss in detail the way KVM creates virtual network devices, interacts with the VMs drivers, and with the physical cards. Similarly, Section 2.2 sheds a light on network processing in the context of containers.

2.1. Network processing inside KVM

Delivering packets to a VM can be achieved mainly through two types of virtual drivers: **emulated** and **para-virtualized**. We describe how each of these driver models operates and exchanges packets with the Host and reference some leading drivers for each technique.

Device emulation mimics the I/O operations the physical card would perform. This is done inside the VMM, in our case KVM, and abstracts the Network Interface Card (NIC) completely, performing CPU instruction translation of all operations. This method provides excellent isolation and security for I/O devices, as processes interact with the VMM and not directly with the network card [6]. A Cloud manager may see security as a deal-breaker and choose device emulation consequently. But this decision has a downside. Emulated drivers impose a technical overhead mainly because of the trap-and-emulate operations performed by the Hypervisor. As emulation requires KVM to translate every instruction from the VM, some instructions may conflict with one that the Host system would execute, this requires the Hypervisor to "trap" the VM's instruction and emulate it instead of running it, which increases the overhead of CPU cycles and context switching.

On the one hand, operating a hardware card in a traditional environment is perfectly supported by the operating system. On the other hand, a virtual device is a software that acts as if it were hardware. These characteristics lead to performance problems since the host CPU must take care of the processing usually performed by a highly specialized and optimized hardware circuit.

Para-virtualized (PV) drivers offer a balance between performance, flexibility, and software support. These drivers are built to "understand" they are used in a virtualized environment and therefore can take advantage of this environment to exchange packets with the hypervisor and NIC in a more efficient way. Instead of emulating a network card inside the hypervisor, para-virtualized drivers pass the network packets directly from VM to the host. This process will be detailed next.

The most used and well-known PV driver is the Linux VirtIO [7]. VirtIO implements its abstraction to the VMs through various structures, the main one being `struct scatterlist`. This struct is a "buffer" that holds two types of entries: **out** and **in**, and each entry stores data destined to one of these directions. The buffer saves every information necessary to exchange data to/from the hypervisor driver and, ultimately, the network card. In other words, the VM sends a buffer of data to KVM, and the hypervisor completes its forwarding. This approach enhances network performance compared to the complexity of driver emulation, supported as the default driver when new VMs are created in KVM.

The VMXNET3 driver is another para-virtualized driver, that simulates 10Gbps NIC developed by VMWare mainly to be used in its proprietary ESXi Hypervisor [8]. It integrates the device driver from the guests and the network processing with the Hypervisor. However, the driver was developed with virtualization in mind. VMXNET3 NIC improves performance compared with its support for multi-queues, Receive Side Scaling (RSS) that distributes network processing among different CPUs, the offloading of IPv4 and IPv6 and Large Receive Offload (LRO) where packets are aggregated into a larger buffer before being passed to higher levels in the network stack, among its support for other features.

As for emulated drivers, the E1000 is another well-known and used one. It emulates a 1 Gbps Intel network card (Intel 82545EM) [9], and as the hardware of the physical E1000 NIC is complex, the driver is estimated to have a considerable overhead. RTL8139 emulates a Realtek NIC and was commonly used as the default driver for KVM virtual machines before the launch of VirtIO.

Table 1
Server's hardware specification.

SO	Ubuntu 18.04.5 LTS
Kernel	5.4.0-47-generic
Processor	Intel(R) Xeon(R) Bronze 3204
RAM	64 GiB
HD	ATA-DISK DELLBOSS VD
NIC	NetXtreme BCM5720 Gigabit Ethernet PCIe

Table 2
Server's major software configuration.

Software	Version
qemu-kvm	2.11.1
virsh	4.0.0
lxc	3.0.3
docker-ce	19.03.13
ping	iputils-s20161105
tcpdump	4.9.3

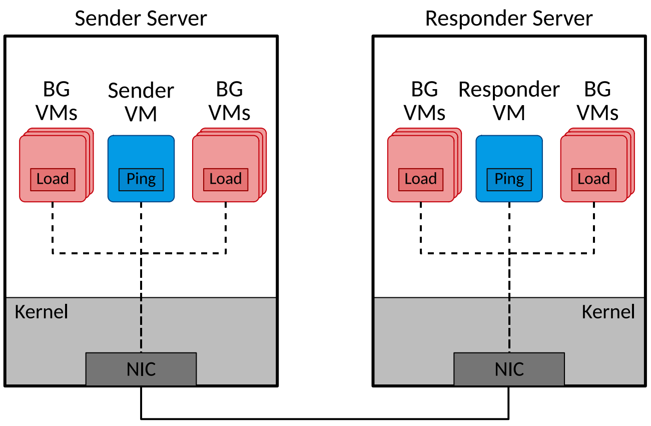


Fig. 1. Testbed configuration.

2.2. Network processing inside containers

In order to allow containers to achieve their desired isolation and performance inside an existing operating system, they rely on specific technologies provided by the OS itself, or more specifically, the kernel. One of these is Namespaces [10]. This particular feature, implemented inside Linux, wraps a specific system resource in an abstraction layer, presenting itself to a specific process as its single own global resource.

As a result, system resources related for networking can be isolated in network namespaces [11], creating virtual network devices with IP addresses, routes, etc., for each process running within a namespace. This virtual device can connect to the Internet through a bridge to the root namespace or configure IP forwarding with NAT rules to the root namespace, among other solutions. Docker and LXC heavily use namespaces to coordinate their networking devices within each container.

Compared to the emulation or para-virtualization of network devices, the network performance of network devices inside namespaces is expected to be higher, as the processing of network packets is not mediated by a VMM that requires heavy processing for their forwarding.

3. Experiment setup

As the main goal of this work is to analyze the influence of a virtual cloud computing environment in network performance, we need to setup a testbed representative of this scenario as faithfully as possible. Our testbed consists of two host servers directly connected in a P2P topology. They have equal hardware and software configurations, listed respectively in Tables 1 and 2. Both servers run a virtual environment that simulates a targeted cloud computing scenario. This virtual environment includes two types of VMs, "main" and "background". The main VMs are responsible for the network performance measurement, whereas the background VMs are responsible for executing a workload to stress server resources. This yields to a common scenario typical of an Infrastructure as a Service (IaaS) provider where different services compete for resource usage. In such scenario, a new tenant would want its VM to have adequate network performance. Measuring such performance falls into the responsibility of the main VM.

The main network performance metric used in this study is the Round-Trip Time (RTT). RTT is an important and well established network metric, used in different performance measurement techniques, such as bandwidth estimation, latency reduction or network buffer assessment as discussed in works like [2,12]. Also, time sensitive applications and services have a direct relationship with the RTT metric as they monitor this parameter. Thus, we direct the experiments to assess network performance through the RTT metric.

The conducted experiments consist of first initiating the virtual environment with the background VMs running the workload to stress the servers and the main VMs executing the network measurement. As the network metric used in this work is the RTT, the measurement happens using the packet echo ping tool inside the main VMs. The main sender VM is always responsible for sending these ping packets, therefore it is referred to as the "Sender VM". The second main VM is responsible for receiving the ping packets and responding them back. This second main VM is therefore called the "Responder VM". After receiving the ping response packets, the Sender VM calculates the RTT and the experiment finishes. Each experiment sends a total of 1000 ping packets and therefore calculates 1000 RTT samples. To better organize the experiments, the Sender VM is always in the first server, that we conveniently call the Sender Server. Respectively, the second server always hosts the Responder VM, thus we call it the Responder Server. Fig. 1 illustrates the testbed scenario.

We establish a timestamp capture point in the Sender Server's kernel through the `tcpdump` packet analyzer tool. In other words, every ping packet that crosses the server's kernel – either a send or response packet – has its timestamp recorded. This allows, for example, calculating the Round-Trip Time of a packet from outside the sender VM and the hypervisor processing, starting from the server kernel. Fig. 2 illustrates the experiment packet flow, including the network capture points from ping and tcpdump.

With both RTT measurement points, we are able to estimate the time that the virtual environment spends processing network packets. We achieve this by calculating the ΔRTT, the difference between the RTT obtained from ping and the one obtained from tcpdump. The result of this calculation is the **Virtualization Overhead**, which is our second obtained experimental result, used to better estimate the impact of virtualization in network performance.

We split the experiments in two parts based on the type of virtualization technology used:

- **Full Virtualization Performance Experiments.** In this part, we create a virtual environment containing the KVM hypervisor.
- **OS Virtualization Performance Experiments.** In this part, we create a virtual environment populated with Container based VMs. As mentioned before, we use LXC and Docker as these are the popular container systems in widespread use.

The factors identified for the experiment factors are generally related to virtualization settings and network configuration, as detailed next:

- **Number of Background VMs**. We use a variable number of background VMs, in the range **0, 1, 2, 4, 8, 16, 32, 64 or 128.** The level 0 represents our baseline and theoretically our best-case scenario, where there is no resource competition and the network performance should be unaffected.

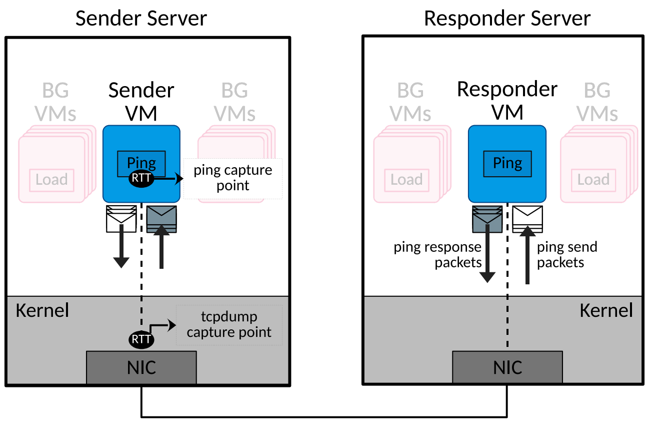


Fig. 2. Experiment flow.

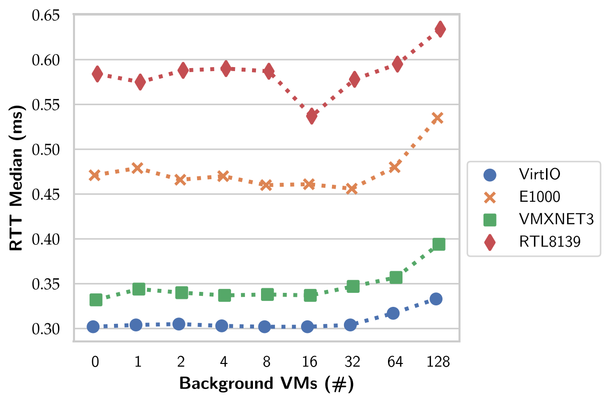


Fig. 3. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median of all experiments.

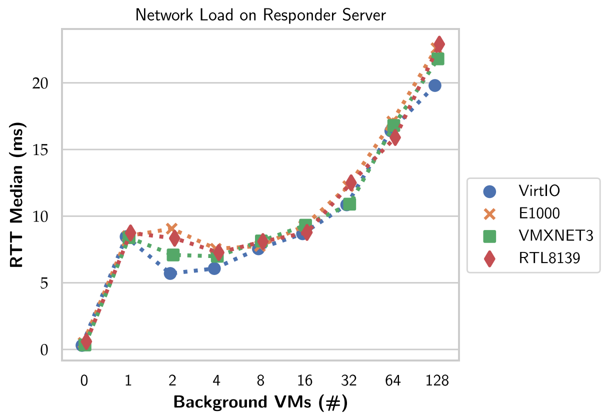


Fig. 4. Results for all experiments conducted, separated by load location and load type.

- **Load Type**. The workload each background VM performs targets a different hardware resource. These include either **CPU, network or disk I/O load**. As earlier mentioned, cloud computing data centers usually host servers dedicated to specific tasks such as storage, running databases, VNFs, web VM servers, among others. Hence the overhead depends on the nature of the service being executed and ranges in its use of CPU, network and I/O resources. The experiments seek to emulate these different types of background loads. For example, the network workload consists of the background VMs sending network packets at the line rate of the virtual network card.
- **Load Location**. The impact of workload that each background VM performs also differs according to whether it is runs at the **Sender Server or Responder Server**.

4. Results and discussion

In this section we present and discuss the results obtained in the experiments carried out. The results for KVM, as well as LXC and Docker virtualization technologies are given in Sections 4.1 and 4.2 respectively.

4.1. Full virtualization performance

This first experiment refers to when the testbed scenario uses the KVM hypervisor, emulating a data center using full virtualization technology. We execute different scenarios that vary the type of virtual network driver used, the frequency of packet exchange and the use or not of affinity, as shown next.

4.1.1. Virtual drivers performance

We initiate the experiments to assess the role of each Virtual Network Driver and understand the way it affects network performance when used with KVM. To this end, we only consider two driver virtualization strategies, **para-virtualized** and **emulated**. Consequently, two drivers representing each technology are tested. To evaluate the performance of using para-virtualization in the design of network drivers, we consider VirtIO and VMXNET3. Both E1000 and RTL8139 represent the class of emulated network drivers.

RTT: Overall, the para-virtualized drivers outperform the emulated ones. More specifically, VirtIO offers the best performance of RTT median compared to all other drivers. This behavior occurs for all factor levels reflecting the different scenario changes and configurations. This was expected as para-virtualized drivers have a more straightforward approach to transmitting and receiving packets than that of emulated drivers which much more complex. In some cases, such as when we apply the Load Location in the Responder Server and with the amount of background VMs above 1, we observe a ≈40% reduction in RTT compared to the emulated RTL8139. The VirtIO driver in scenarios with high VMs number of 128 is the only driver achieving an RTT median value lower than 0.35 ms. This result is detailed in Fig. 3.

Emulated drivers like RTL8139 have the highest RTT values because they do not implement mechanisms that optimize network traffic between the VM and the native host, like “Virtio ring”, provided by VirtIO. The “Virtio ring” allows transmit/receive queues to be shared with the hypervisor in the user space. We also concisely present other arguments as to why para-virtualized drivers are more efficient in Section 2.1. Therefore, it makes network data exchange between the VM and the native machine more straightforward. Still, as for the emulated drivers, the E1000 presents itself as the most optimized in this regard.

Fig. 4 show the specific scenario when we configure Network Load Type as Responder Load Location. It is possible to observe the sharp increase in RTT values as we increase the number of virtual machines in the background. This increase takes place for all drivers. We understand that this result is likely caused by the Responder server being busy processing competing network packets. This leads to delays in the processing of the ping messages. As more background VMs sending heavy network traffic are inserted into the Responder server, this effect becomes more and more accentuated. This result is interesting as it shows the way network performance (RTT) would behave in a VM hosted in a data center acting as a server with lots of concurrent traffic. In all other cases the values are close to 0.01 ms, regardless of the load Type and Location, similar to Fig. 3

We also observe a more significant variation of the RTT when the Load Location is on the Responder server with Network Load regardless of the driver used. The standard deviation for the VirtIO driver – our

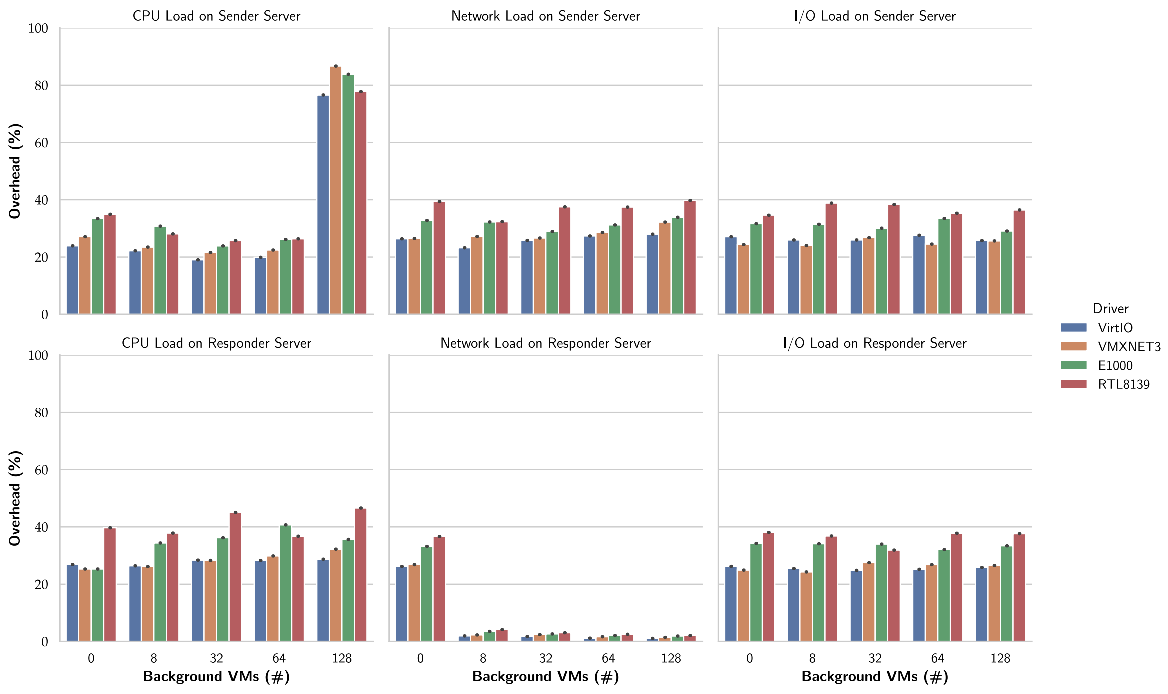


Fig. 5. Results for all experiments conducted, separated by load location and load type.

best-case scenario so far – is 0.35 ms when using 0 background VMs. However, when using 128 background VMs, the standard deviation climbs up to 11.29 ms, and if using RTL8139 it reaches 11.95 ms. This can be explained by the fact that the Responder server network card is overloaded with outgoing network packets from the background VMs. This leads to maintaining the ping response packet queued for a longer period than normal, which makes the network unstable and subject to a higher variation.

Virtualization Overhead: Due to their more simpler nature, para-virtualized drivers such as Virtio, achieve the lowest Virtualization Overhead. The E1000 emulated driver has a smaller overhead compared to that of RTL8139 (similar when we evaluate the RTT median). Fig. 5 displays the percentage overhead for this scenario.

When submitting the Responder server to network load, we observe the lowest levels of overhead percentage for any VM number other than 0, when compared to scenarios with CPU and I/O load. However, it is possible to see that such low percentage does not translate into a good performance. The RTT in these scenarios is higher, as explained before, and the overhead RTT remained the same independently of the number of VMs, causing the percentage to decrease.

There is also a higher overhead percentage suffered by all drivers when using CPU Load in the Responder Server with 128 VMs. This behavior is explained also by the presence of a high number of background VMs with CPU workload that generates a bottleneck for packet processing. This creates a delay inside the server kernel, since it is busy processing the high amount of tasks from the background VMs and also their outbound and inbound ping packets.

Examining these results we can point out some particularities. First, none of the driver technologies exhibits a distorting behavior between its virtualization overhead and their final RTT performance. In other words, we do not experience cases where a high RTT presents a low overhead. Furthermore, considering all scenarios, we can observe that the virtualization overhead got emulated drivers accounts for ≈30% to ≈40% of the final RTT. Para-virtualized drivers contribute approximately 25% to 32% to the RTT.

Observe also a constant performance with VirtIO being the lowest overhead and RTL8139 the highest one. Considering the median overhead for all experiments, including VM number, Load location and type, the VirtIO para-virtualized driver presents ≈25.26% of virtualization overhead. Also, when using the RTL8139 emulated driver, the RTT overhead percentage was the highest compared to other drivers, having ≈35.49% of the RTT being composed of virtualization overhead.

Finally, we emphasize that even considering only the para-virtualized drivers, VirtIO once again presents itself as the most optimized in this regard.

Summary: From our study of driver overhead under KVM, we observed that none of the considered drivers stands out against RTT variation. In other words, they exhibit varying levels of instability. In addition, as expected, we observed that para-virtualization-based drivers trigger a more optimized performance, considering Virtualization Overhead and RTT, as they can use features that make data exchange through direct communication between the VM and the native host. They no longer rely on the hypervisor for such transfer, whereas emulated drivers still do. We also looked at the impact of network load on RTT performance when we instrumented the Load Location on the Responder. This led to higher RTTs as well as a larger RTT variation for all drivers.

In this initial assessment, we note that the number of background VMs 1, 2, 4 and 16 do not induce behavior that requires detailed analysis. Hence, these levels are removed from this point on and not included in the coming experiments. In addition, we also observe the hegemony of the para-virtualized VirtIO driver. As a result, this is the main driver used in the rest of this study.

4.1.2. *The impact of CPU affinity*

We continue the KVM experiments by assessing whether CPU affinity when applied in conjunction of the main VMs impacts RTT performance.

RTT: Overall, CPU affinity usage does not significantly influence RTT performance. Fig. 6 displays the RTT median based on the CPU Affinity. These results demonstrate how ineffective CPU Affinity is in these experiments, which is considered an unexpected behavior. Intuitively, dedicating computational resources to the main VMs ought to speed up the processing of sender and Responder VMs. Apparently, this has not been the case.

Also, the presence of CPU affinity does not impact the variation of the RTT. The cases of higher variation, such as when applying network

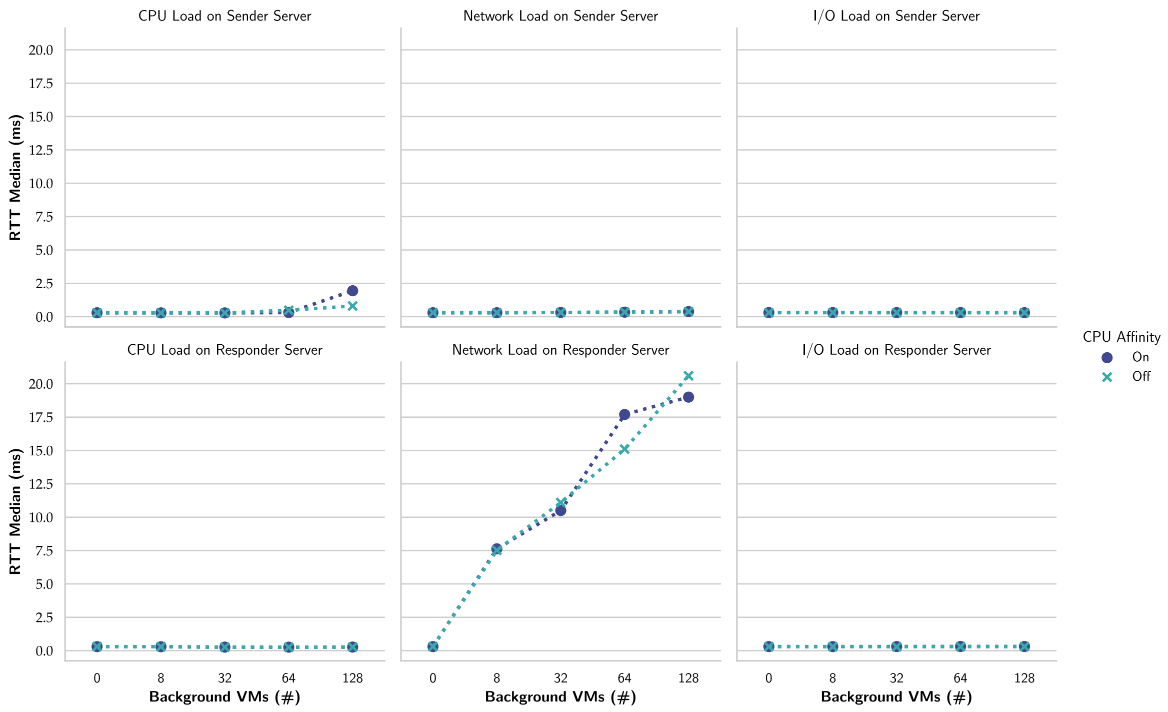


Fig. 6. RTT results for all experiments conducted, using the VirtIO driver and separated by load location and load type.

Table 3
Overhead percentage when using CPU affinity on main VM. The values are displayed in percentage units.

CPU affinity	Avg.	Std. Dev.	Min	25% Quartile	Median	75% Quartile	Max
Off	27.36	15.80	0.15	23.74	26.32	28.43	99.15
On	27.82	16.44	0.06	23.74	26.19	28.66	99.23

load on the Responder server remain the same whether the CPU affinity is on use or not.

Virtualization Overhead: As seen in the RTT results, CPU affinity does not affect the virtualization overhead. Table 3 summarizes the overhead percentage for the experiments with CPU affinity set to on and off. Notice that the metrics displayed in the table are calculated for all parameters of the experiments, including background VM number, Load Type and Location. It is obvious how little difference there between when and not using CPU Affinity.

Summary: The use of CPU affinity in the main VMs introduces no direct performance boost as previously thought. Regardless of whether the background VMs perform CPU or network load, of that the load is placed at the sender or Responder server, CPU affinity has shown no advantages.

4.1.3. *The impact of packet frequency*

We conclude the KVM experiments by investigating the behavior of RTT and its relationship while changing the frequency of echo exchanged. We use three frequencies, 1 pps, 1 Kpps and 1Mpps, the last one representing near line-rate frequency.

RTT: Fig. 7 shows a curious behavior between experiments when contrasting 1 pps results to those for higher frequencies. An overall lower RTT median than that for 1 pps is achieved by the experiments exchanging echo packets at higher frequencies. We believe that this odd result is due to the CPU wake-up time and interrupt delivery response. A frequency of 1 pps causes the CPU to remain in interrupt-based mode, representing a significant overhead in response time. When frequency increases, the CPU switches to using polling mode which allowing higher CPU usage and faster access to packets [13,14]. This can also be seen even within the 1 pps set of experiments. As the number of VMs increases from 8 to 32, the overall median RTT decreases, since the higher CPU load causes the CPU to change to operate in the polling mode.

Another result obtained as part of this set of experiment where packet frequency is changed, is that in most cases, the 1Mpps frequency offers lower RTT. This follows the same explanation from before: as the CPU is working in polling mode, it offers a better performance with faster processing and lower latency. But, it is fair to state that difference between 1 Kpps and 1Mpps scenarios in terms of the RTT median is not as significant as the difference between the 1 pps and 1Mpps scenarios.

Fig. 8 shows a box plot of the results for these experiments with all load types located at the sender server. Observe also here how the frequency also impacts RTT variation. See that 1 pps generates a wider IQR and higher outliers. Also, when using 1 Kpps, RTT variation increases along with the number of VMs. This RTT increase is much larger than the one the 1 Kpps frequency.

Virtualization Overhead. A similar behavior is experienced here, as displayed in Fig. 9. An interesting point observed is that when the network load is applied at the Responder server and the packet frequency is set to the small value of 1 pps, the overhead starts at a 0.36 ms even with 0 background VMs and decreases to 0.21 ms with 128 VMs. This behavior follows the same explanation from before. It is due to the CPU changing from interrupt to polling mode of operation which enhances packet processing. However, with the cases for 1 Kpps and 1Mpps the opposite happens. The overhead starts close to 0.12 ms at 0 VMs and ends at 0.21 ms with 128 VMs. This, on the other hand, is the direct result of existing network overload inside the servers network cards, which increases the virtualization overhead since the hypervisor is dealing with constant bursts of packets.

Summary. A low packet rate may paradoxically create higher RTT in environments with low resource consumption and higher packet packet rates may yield better performance. However, a server dominated by VMs with high outgoing network traffic, creates a considerable virtualization overhead.

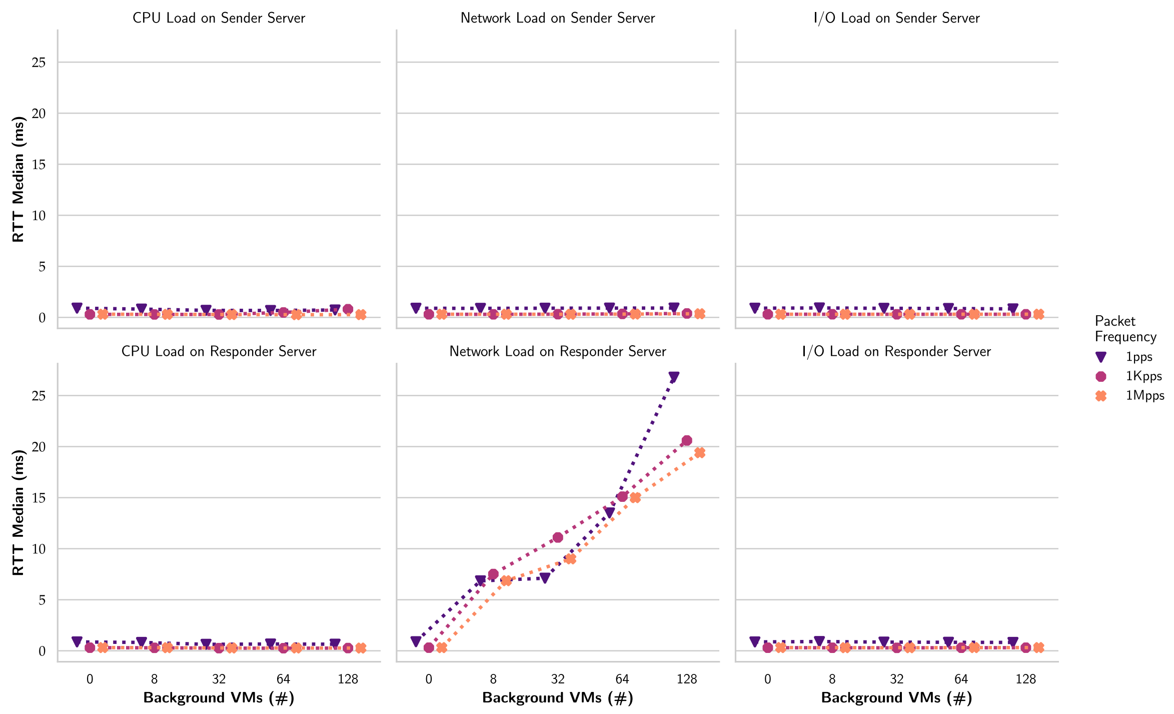


Fig. 7. RTT results for all experiments conducted, using the VirtIO driver and CPU affinity off, separated by load location and load type.

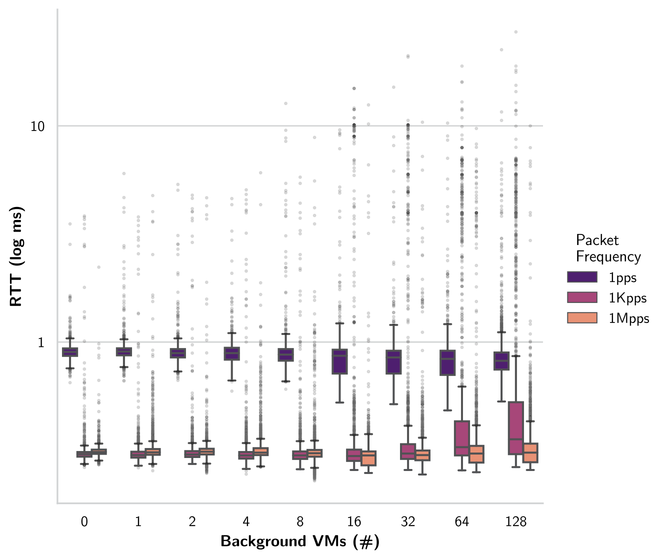


Fig. 8. RTT results for all experiments conducted, using the VirtIO driver and CPU affinity off, considering only the sender server and all load types.

4.2. OS virtualization performance

This next experiment evaluates RTT when using container virtualization, more specifically LXC or Docker containers. Given the previous results, some of the scenarios have been ignored due to their limited role, but we maintain the analysis of the packet exchange frequency of either 1 pps, 1 Kpps or 1Mpps.

RTT: We notice that part of the experiments reported similar RTT results for both Docker and LXC, even though Dockers achieved the lowest RTT values. This assertion can be observed mainly when the load is applied at the sender server, with an echo packet frequency ranging between 1 pps and 1 Kpps, and the application of I/O and CPU load types, as illustrated in Fig. 10. However, when increasing packet exchange to 1Mpps, Docker containers benefited from smaller RTT values with less variation, regardless of the load type, when compared with LXC. Hence, ping packet frequency turned out to be an impacting factor on the network performance when using LXC containers.

Combined with the high frequency factor, we also observe that the Load Type affects the RTT, when it is applied at the sender server. For instance, when combining a 1Mpps frequency with LXC, CPU Load Type and a high number of background VMs the RTT values increase, going from ≈10 ms with 8 VMs to ≈20 ms with 32 VMs. When increasing even further the number of background VMs to 128, RTT tops 52 ms. Also, the variation increases as well demonstrating less RTT stable results, along with the higher number of VMs. This can be considered an intuitive result, since the higher VM number translates directly to higher CPU utilization, which end up competing for resources that otherwise could be dedicated to the packet processing in the sender VM.

On the other hand, when the Load location is applied the Responder server, there is a different outcome, as illustrated in Fig. 11. The use of a high packet ping frequency such as of 1Mpps still seems to affect LXC in a manner different to the one with Docker containers. But, when applying a CPU load type at the Responder, the RTT does not suffer an increase as previously seen when overloading the sender server whereas the network load type causes the RTT to increase even further, reaching an RTT value around 120 ms when using 128 VMs with LXC. This also seems to be the only scenario where Docker suffers an increase in RTT and RTT variation, even though it still outperforms LXC. This behavior follows the one observed under KVM, also justified by the high number of outgoing packets from the background VMs.

Virtualization Overhead. The overhead from container technology is minimal, as expected. Most of the cases, the values are close to 0.01 ms for both LXC and Docker, following the trend seen previously when applying I/O or Network Load. The exception is exactly the scenario when applying CPU load on the sender server with LXC, as displayed in Fig. 12. Here the RTT is dominated by virtualization overhead, where the RTT is 51 ms and the overhead ≈48 ms, when the environment runs 128 VMs. This translates to a significant share (percentage) of the RTT being composed of overhead stemming from the virtualization technology.

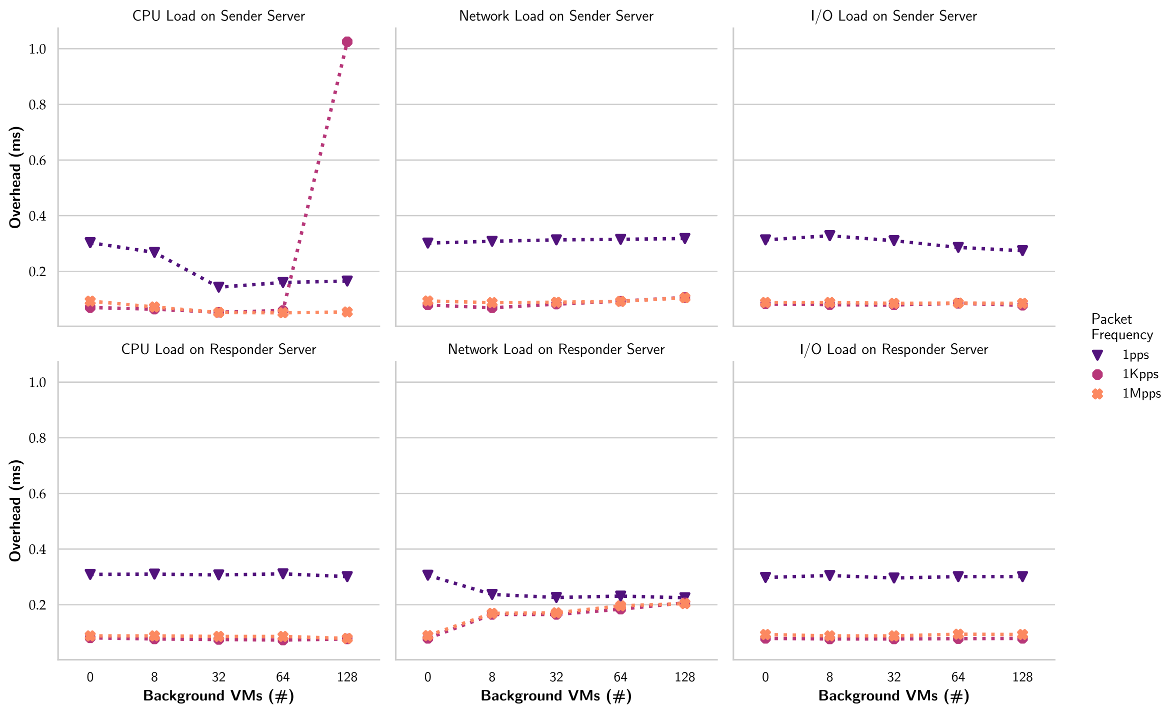


Fig. 9. Virtualization overhead results for all experiments conducted, using the VirtIO driver and CPU affinity off, separated by load location and load type.

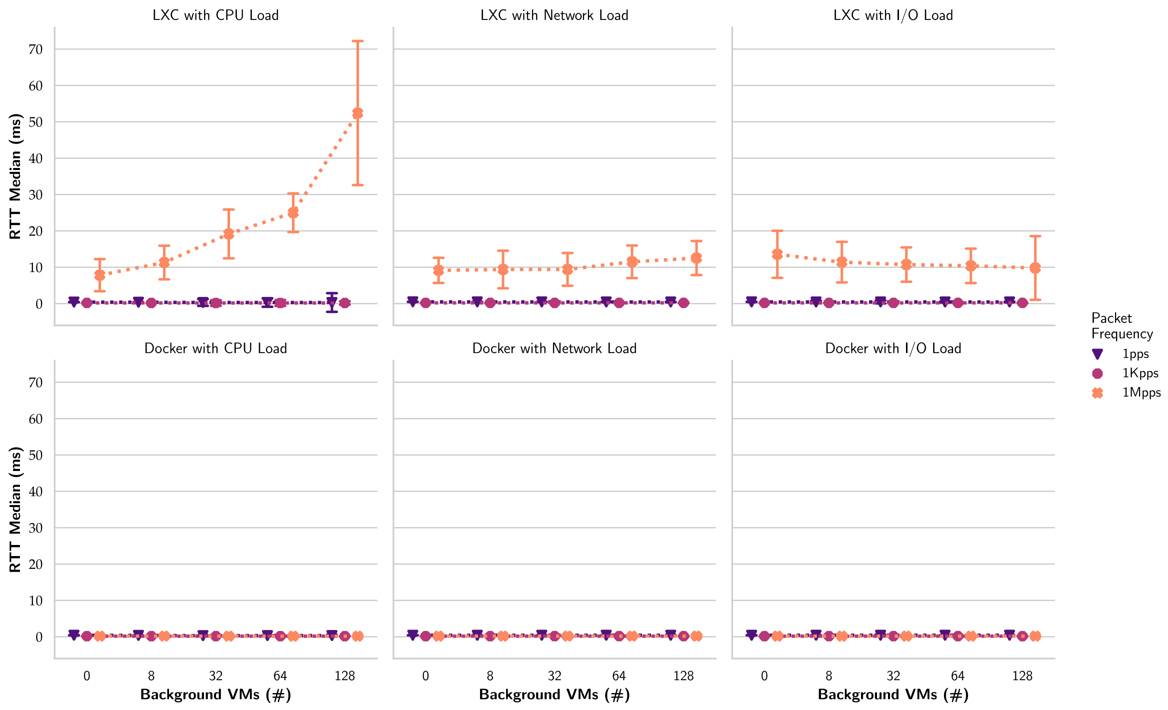


Fig. 10. RTT results for all experiments conducted only on the Sender Server and separated by Load Type.

Summary. The LXC container seems to be most affected by the use of high packet frequency exchange, resulting in high RTT values for any Load Type. LXC also seems to suffer impact of high CPU load when applied in the sender server. In addition, both LXC and Docker suffers with network load when applied in the Responder server. This indicates that, for example, a data center dedicated to VNFs with high packet incoming rate could experience performance loss if using LXC or Docker.

We summarize our results in Figs. 13 and 14.

4.3. Statistical analysis

This section summarizes some of the above results based on the application on widely used statistical techniques. The first one, known as *Kendall rank correlation coefficient*, is a non-parametric hypothesis test regarding statistical dependence based on the tau coefficient. Kendall correlation coefficient returns a value in the range −1 to 1, where 0 is, no relationship, −1 and 1 signal a perfect relationship (the negative sign indicates an inverse correlation). The second statistical technique detects which values in the dataset could be considered “out of the

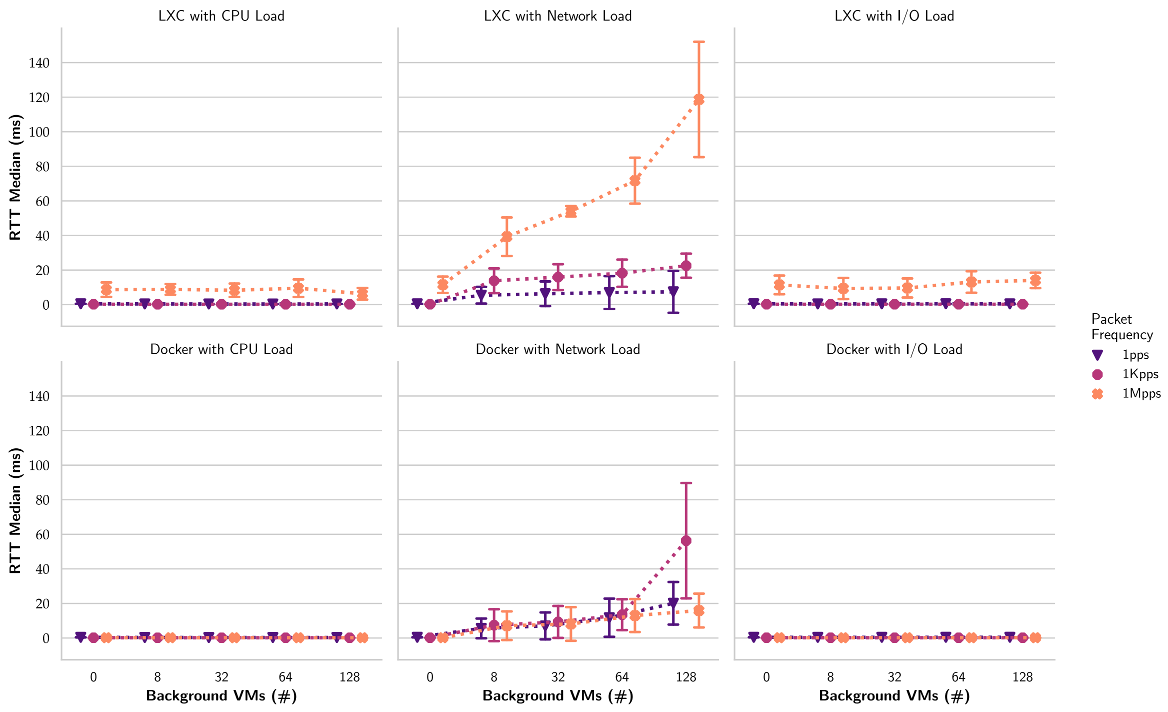


Fig. 11. RTT results for all experiments conducted only on the Responder Server and separated by Load Type.

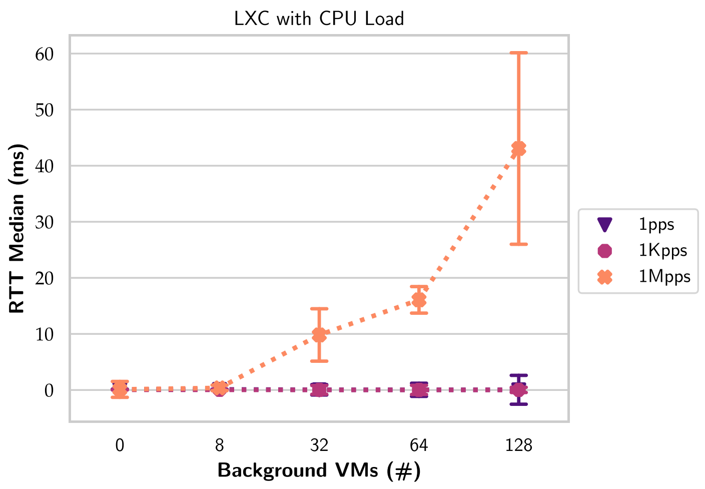


Fig. 12. Overhead results for all experiments conducted only on the Sender Server and separated by Load Type.

curve". It is able to identify how typical or atypical the obtained values were. The technique used was robust Z-score. For the present statistical analysis, we considered all factors with all levels we used in the respective experiments.

4.3.1. *Full virtualization statistics analysis*

Kendall tau. in Fig. 15, we observe the correlation results using Kendall Tau's correlation coefficients for experiments for a scenario running KVM. These results corroborate the results discussed in the previous session. Observe that in general, a lower frequency (in our case 1 pps), is related to a higher RTT and overhead. Also, the VirtIO driver is related to lower RTT values and lower overhead. And also applying the network load type NET is related to the highest values of RTT and higher overhead. This is expected as competing network activities interfere directly with out packet probing activity. Finally, one can confirm that CPU affinity is not related to the increase or decrease of RTT or overhead values.

Table 4
Scenarios where outliers are concentrated (considering all outliers found in our experiments using KVM).

Scenarios		Percentage
Load type	Load location	
NET	Responder server	**95.41%**
NET	Host sender	0.19%
Others	Responder server	1.09%
Others	Sender server	3.3%

Robust Z-score. Table 4 shows in which scenarios are the values considered outliers or atypical. These were mainly generated when using KVM virtualization. In addition, among all the detected outliers, 95.41% of these happen when there is a competing network activity (Load Type NET) running at the Responder Server (i.e. location is Responder).

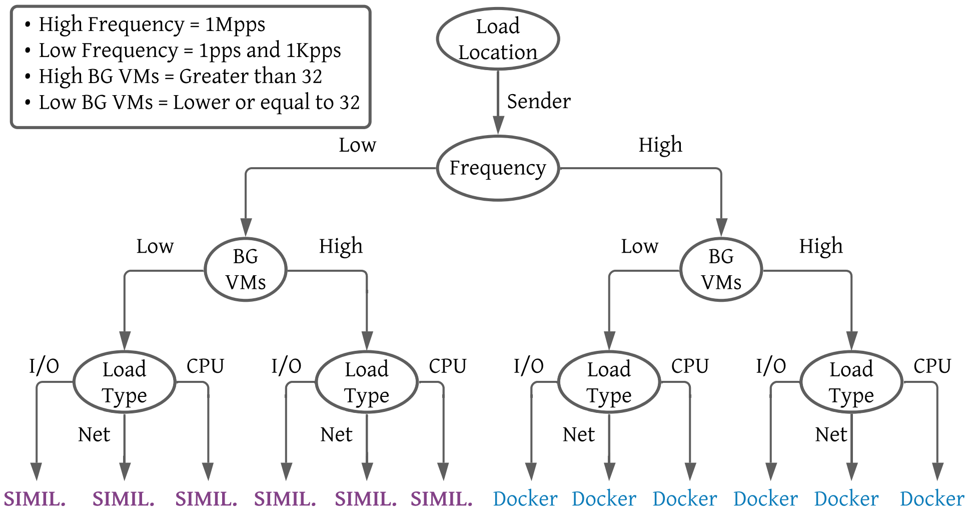


Fig. 13. Summary of network performance for all container experiments on the Sender Server.

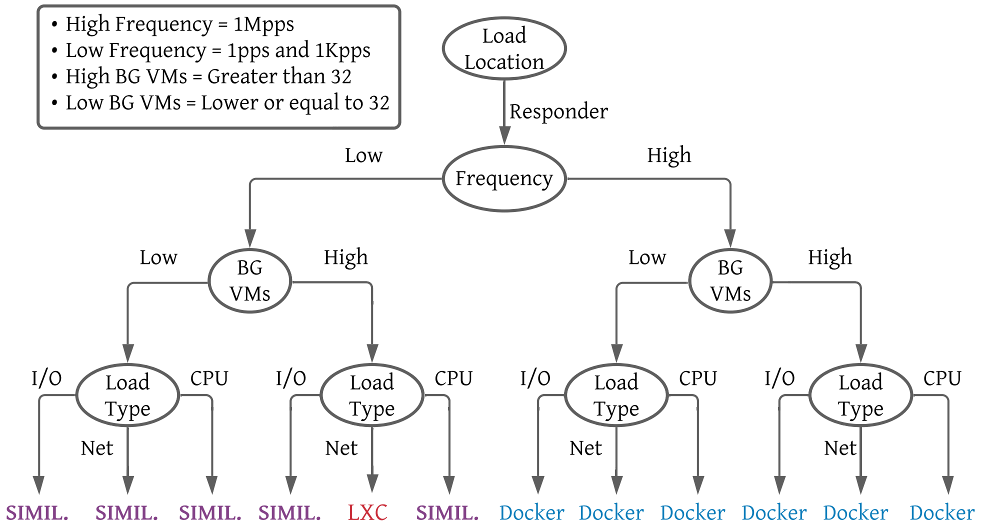


Fig. 14. Summary of network performance for all container experiments on the Responder Server.

4.3.2. *OS virtualization performance statistics results*

Kendall tau. In Figs. 16 and 17, we observe the correlation results using a technique of Kendall Tau's correlation coefficients for experiments using LXC and Docker, respectively. These results corroborate the results discussed in the previous session. We can see here that for Docker, a lower frequency (in our case 1 pps), is related to a higher RTT and overhead, however when we observe the LXC this behavior is inverted. Load type NET is related to the highest values of RTT for both technologies.

Robust Z-score. The Tables 5 and 6 (for LXC and Docker respectively) show the scenarios with outliers when using containers. Contrary to what we see in the case of KVM, with LXC the outliers are distributed across several experiments. This confirms the fact that the observed unstable behavior is inherent to the use of LXC itself. As for the Docker technology, we were able to observe that the values located outside the curve are concentrated. Around 99.6% of them are in the scenario that applies a network load type at the Responder server.

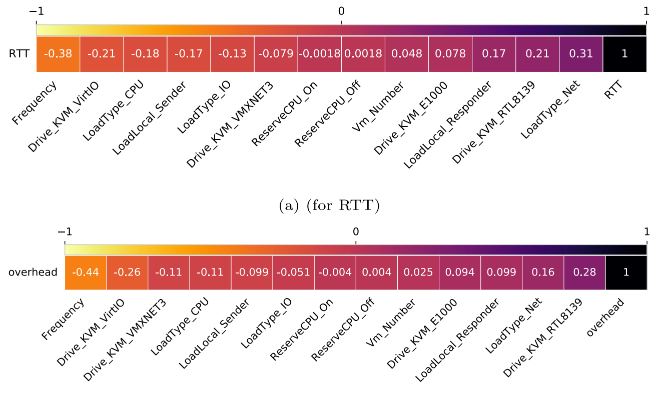
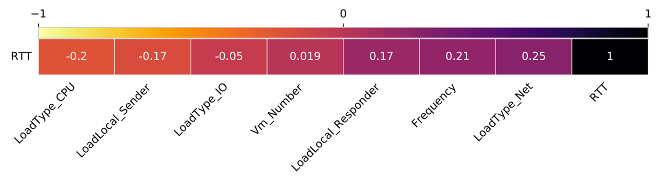


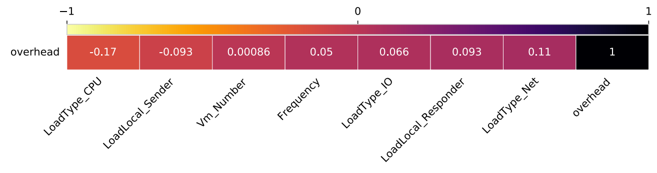
Fig. 15. Kendall rank correlation coefficient - KVM.

5. Related works

Most Cloud Computing Infrastructure management software, such as OpenStack, supports both KVM and container technologies (including Docker and LXC). In this work, we evaluated delay for these main virtualization technologies in a testbed with a variety of cloud computing scenarios. In recent years, several works contributed towards similar comparisons under different data center configurations. Some of these as reviewed next.

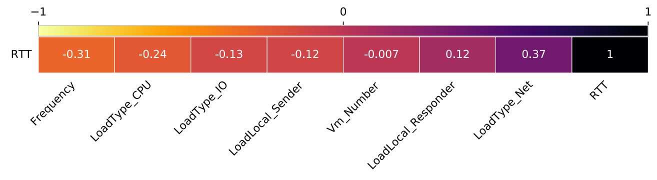


(a) (for RTT)

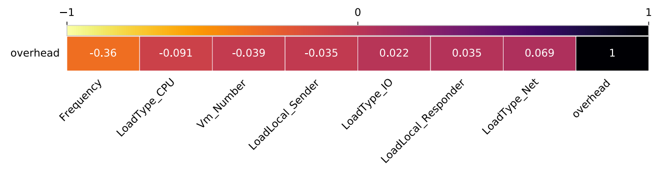


(b) (for Overhead)

Fig. 16. Kendall rank correlation coefficient - LXC.



(a) (for RTT)



(b) (for Overhead)

Fig. 17. Kendall rank correlation coefficient - Docker.

Table 5
Scenarios where outliers are concentrated (considering all outliers found in our experiments using LXC).

Scenarios			Percentage
Load type	Load location	Frequency	
Others	Sender server	**1 Mpps**	25.2%
Others	Responder server	**1 Mpps**	21.9%
NET	Responder server	1 pps or 1 Kpps	21.3%
NET	Responder server	**1 Mpps**	21.2%
NET	Sender server	**1 Mpps**	10.3%
Other scenarios			0.1%

Table 6
Scenarios where outliers are concentrated (considering all outliers found in our experiments using Docker).

Scenarios		Percentage
Load type	Load location	
NET	Responder server	99.6%
Other scenarios		0.4%

The work in [15] compares KVM and Docker performance. The conducted benchmarks evaluated latency in an InfiniBand cluster, a high-performance computer-networking communications standard [16]. The research described in [17] reports a comparison between virtualization technologies, KVM, Docker, and LXC against native host performance, using the throughput metric. It discusses the advantages and limitations of these three virtualization techniques and draws special attention to unexpected processing, memory, and network overload. Unlike [17], our study tackles a wider set of background overhead that impacts packet processing such as the impact of I/O operations. This same study also evaluates the OSv, a hybrid technology seen by many researchers as a viable alternative in the hypervisor versus container dispute. Both works [15,17] point to container virtualization as being more efficient. There have even been experiments where the container achieved a comparable performance to that of a native platform.

The authors in [17] look at the impact of virtualization techniques on packet throughput. Differently, the present study acknowledges a lack of RTT delay based analysis as justified in Section 3.

The authors in [3] examined clock accuracy in virtual machines. They performed several experiments for UDP and TCP transport protocols while also varying I/O and CPU loads. They compared the results with those of a baseline ('Idle') scenario, i.e., without the presence of any significant I/O or CPU loads. A significant finding pointed out by the study is the possibility to optimize network performance by carefully selecting timeout functions such as the sleep function (HPTSleep) from the HighPerTimer library. This is a nanosecond level precision, unified high-performance timer for Linux time counters that automatically identifies the fastest and most reliable one available in user space. It has been reported to reduce the loss of computer resources during the execution of a sleeping process from a 100% level (corresponding to a busy-wait) to a mere 1%–2%, while very accurately maintaining function resuming times on long waits [18].

In [19] the authors examine at packet time-stamping accuracy in the particular case of Docker virtualization. They evaluate scenarios with and without the presence of background overload. A total of eight containers are used to generate CPU, I/O, Memory, and communication resource bottlenecks. They also analyze a passive monitoring system based on containers that depends on time-stamping accuracy for its validation. The work concludes that packet time-stamping under Docker has a reasonable accuracy while exhibiting errors of the order of few microseconds.

The experiments conducted by the research described in [2] discuss packet time-stamping using active measurement and evaluate RTT in a KVM only configuration. They configure a transmitter VM and a receiver VM that share physical resources with up to 128 background VMs responsible for an overload of the CPU, I/O, or network resources. The tests also evaluate the impact of CPU affinity on RTT measurement evaluation. The authors looked at the effect of the actual network drivers themselves. They compared the difference in terms of impact between using the Intel 82545EM Gigabit Ethernet E1000 NIC emulated driver and the VirtIO para-virtualized driver. They concluded that CPU affinity lowers the measured value of RTT, except when there are high levels of I/O overload on the server. Another important result from the study pointed out that under high network traffic conditions, the emulated network driver is more stable than the para-virtualized one in that it obtained lower RTT variation.

Although our work confirms some of the findings in [2], we make some divergent observations. For instance, our measurements report that CPU affinity has no impact on RTT. In the experiments with high network or I/O loads, the results converge with those of [2] as they also confirm that affinity does not lower RTT delay as intuitively expected.

The authors of [20] perform an analysis focused on cloud-to-user (C2U) networks. They investigate the performance of the network versus the geographical position by monitoring C2U latency for two providers (Azure and AWS) at high packet exchange rates and with multiple active measurement methods. They compare the C2U performance of the providers and identify anomalous latency degradation ("badness") events and assess their persistence and dependence on a given provider. They also evaluate the impact of different probing methods on both the observed latency values and the outcomes derived from these observations. Among their findings are that the user's C2U latency is, on average, better on intra-region paths and that generally, a lower latency does not imply less variability. They further report that "badness" events are primarily associated with the Vantage Point (VP) and/or the destination area, and not the infrastructure provider. The authors also argue that, in relation to the adoption of different probing methods, despite some discrepancies in the observed latency values, they marginally impact the results of both the ‘‘badness’’ analysis and the provider comparison.

Similar to the contributions in [20], we also observed that a lower latency does not imply less variability. This can be seen in the results in Section 4.1.3. Furthermore, we also identified ‘‘badness’’ events in our experiments.

The research in [21] carried out an extensive experimental evaluation of layer 3 packet forwarding performance of virtual software routers (VSR) based on the Linux kernel and the KVM virtual machine. The aim of the authors is to analyze complex interactions between hardware and VMs in order to identify and remove performance bottlenecks in a VSR implementation. The authors conclude that VSR performance can be improved by adjusting system parameters such as those related to mechanisms used to move data to and from VMs, thread priorities, and CPU affinity.

One of the relevant results in [21] is the adjusting of CPU Affinity in order to improve performance. In our work, we have not detected any performance improvement due to CPU Affinity or pinning. However, in the cited work, the hardware used is different from ours.

The contribution in [22] compared a traditional virtual machine implementation (using KVM) to a container implementation (using Docker). The authors performed a set of benchmarks that stress different aspects such as CPU, memory bandwidth, memory latency, network bandwidth, and I/O bandwidth. The results indicated that the performance of containers was equal to or better than that of VMs (KVM) in almost all cases. Nonetheless, the authors point out that both technologies require parameter tuning to support I/O intensive applications.

As in [22], our results report a Docker performance higher than that of KVM in nearly all cases. Unlike in [22], no special parameter tuning was necessary perhaps due to the use of more up-to-date software versions.

Despite the presence of multiple and distinct efforts towards analyzing the impact that virtualization has over packet time-stamping and subsequently active network traffic measurement methods, the present work is, to the best of our knowledge, the only study that evaluates this problem while using popular virtualization technologies like Docker, KVM, and LXC under a common perspective. In other words, in the same experiment environment. It develops an experimental scenario seeking to reliably represent the real virtual environments present in the context of cloud computing. It takes into consideration the overload at different levels of computational resources (Network, I/O and CPU) applied either at the packet sending or receiving host and studies the role of CPU affinity. Unlike existing works, our experiments include the exchange of packets at different rates or frequencies. Furthermore, the present study evaluates the impact of using different forms of traditional network access drivers including para-virtualized (VirtIO and VMXNET3) and emulated (E1000 and RTL8139) ones. Finally, we employ two statistical approaches (Kendall and Robust Z-score) to validate and extend our results.

6. Conclusion and future works

Virtualization has become fundamental to cloud computing. For example, through this feature, it is possible to achieve or optimize aspects such as computational flexibility and availability. However, despite the many advantages provided, virtualization suffers from some performance disadvantages, such as the processing overhead on virtual machines introduced through the additional software layer.

In this work, we conduct studies seeking to clarify behavioral aspects related to how virtualization affects some computational practices, such as network performance. In particular, we examine the impact of virtualization on the RTT network metric.

This article analyzes the RTT through the development experiments, with different comprehensive levels of virtualization (KVM, Docker, and LXC) among other parameters, such as CPU Affinity, packet frequency, Virtual Network Driver (on KVM), Load Location and Number of Background VMs, seeking to clarify and identify behavioral aspects not yet known.

In addition, this study also covered statistics evaluation of the results obtained by the experiments, using two methods, Kendall tau, and robust Z-score, seeking to validate and clarify our results. Kendall and robust Z-score methods served to help us identify which variables were more meaningful on RTT results. This was very important, as, in some scenarios, it was naturally impossible to identify causal behavior. Furthermore, these mechanisms corroborated the validation of the obtained results.

The results essentially show the impact that virtualization has and how it interferes with the accuracy of RTT at its different levels and particularities and can serve to guide how the cloud infrastructure can be deployed for better performance. For example, in a data center with predominantly I/O consumption like storage or Database servers, KVM VMs will demonstrate a close-to-zero RTT, especially if using VirtIO virtual network drivers, as evidenced in Fig. 4. Also, depending on how a data center acts, whether as service provider or content provider with high network usage, the results may differ. In a service provider such as a web server or network functions, consisting of sending high network load, using docker containers can introduce lower RTT values with less variation. On the other hand, content providers in data centers like edge, that periodically, receive content updates that consists of receiving high network load. This context yields higher RTT values in any virtualization technology but especially when using LXC containers. In such cases, docker containers can provide better performance when analyzing RTT.

Among the results we present, we can highlight:

- In general, CPU affinity does not improve network performance.
- Higher RTT values and greater RTT variation are observed when Network Load is applied at the Responder server.
- High CPU load on the sender server also creates higher RTT values when using LXC containers.
- In the KVM based experiments, VirtIO achieved the best performance among all Virtual Network Drives. Among the emulated drivers, E1000 outperformed the others.
- LXC triggers a wide variation in RTT values when used in a high packet rate scenario.
- Docker outperforms LXC in all scenarios, presenting either better or similar RTT values and variation.
- The application of I/O Load in KVM does not significantly influence the accuracy of the RTT. In container however, this behavior is different.

Finally, as part of future work, we intend to evaluate additional virtualization technologies, such as XEN, VMware ESXI, LDoms/Oracle VM Server for SPARC, OpenVZ, etc. Also, we intend to use different network capture points, such as in the Responder Server kernel.

CRediT authorship contribution statement

Assis T. de Oliveira Filho: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing, Visualization, Supervision. **Eduardo Freitas:** Conceptualization, Software, Writing – original draft, Writing – review & editing, Visualization. **Pedro R.X. do Carmo:** Software, Validation, Formal analysis, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Djamel H.J. Sadok:** Conceptualization, Project administration, Writing – review & editing, Resources. **Judith Kelner:** Resources, Funding acquisition, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), Brazil.

References

[1] IDG, Cloud Computing Survey, Executive summary, IDG, 2020, URL https://resources.idg.com/download/2020-cloud-computing-executive-summary-rl.

[2] R. Dantas, D. Sadok, C. Flinta, A. Johnsson, Kvm virtualization impact on active round-trip time measurements, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 810–813.

[3] K. Karpov, I. Fedotova, D. Kachan, V. Kirova, E. Siemens, Impact of virtualization on timing precision under stressful network conditions, in: IEEE EUROCON 2017 -17th International Conference on Smart Technologies, 2017, pp. 157–163.

[4] W. Chen, H. Lu, L. Shen, Z. Wang, N. Xiao, D. Chen, A novel hardware assisted full virtualization technique, in: 2008 the 9th International Conference for Young Computer Scientists, 2008, pp. 1292–1297, http://dx.doi.org/10.1109/ICYCS.2008.218.

[5] S. Grunert, Demystifying containers – part I: Kernel space, 2019, Internet, URL https://www.suse.com/c/demystifying-containers-part-i-kernel-space/.

[6] F.-F. Zhou, R.-H. Ma, J. Li, L.-X. Chen, W.-D. Qiu, H.-B. Guan, Optimizations for high performance network virtualization, J. Comput. Sci. Technol. 31 (2016) 107–116, http://dx.doi.org/10.1007/s11390-016-1614-x.

[7] R. Russell, Virtio: Towards a de-facto standard for virtual I/O devices, SIGOPS Oper. Syst. Rev. 42 (5) (2008) 95–103, http://dx.doi.org/10.1145/1400097.1400108.

[8] VMware, Performance Evaluation of VMXNET3 Virtual Network Device, Tech. rep., VMware, 2010.

[9] VMware, Choosing a network adapter for your virtual machine (1001805), 2021, URL https://kb.vmware.com/s/article/1001805.

[10] M. Kerrisk, Namespaces in operation, part 1: namespaces overview, 2013, Internet, URL https://lwn.net/Articles/531114/.

[11] J. Edge, Namespaces in operation, part 7: Network namespaces, 2014, Internet, URL https://lwn.net/Articles/580893/.

[12] A. Botta, A. Pescapé, Monitoring and measuring wireless network performance in the presence of middleboxes, in: 2011 Eighth International Conference on Wireless on-Demand Network Systems and Services, 2011, pp. 146–149, http://dx.doi.org/10.1109/WONS.2011.5720184.

[13] P. Emmerich, D. Raumer, A. Beifuß, L. Erlacher, F. Wohlfart, T.M. Runge, S. Gallenmüller, G. Carle, Optimizing latency and CPU load in packet processing systems, in: Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, in: Spects '15, Society for Computer Simulation International, San Diego, CA, USA, 2015, pp. 1–8.

[14] O. Hohlfeld, J. Krude, J.H. Reelfs, J. Rüth, K. Wehrle, Demystifying the performance of XDP bpf, in: 2019 IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 208–212, http://dx.doi.org/10.1109/NETSOFT.2019.8806651.

[15] J. Zhang, X. Lu, D.K. Panda, Performance characterization of hypervisor-and container-based virtualization for HPC on SR-IOV enabled InfiniBand clusters, in: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 1777–1784, http://dx.doi.org/10.1109/IPDPSW.2016.178.

[16] S. Sur, M.J. Koop, L. Chai, D.K. Panda, Performance analysis and evaluation of mellanox connectx InfiniBand architecture with multi-core platforms, in: 15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007), 2007, pp. 125–134, http://dx.doi.org/10.1109/HOTI.2007.16.

[17] R. Morabito, J. Kjällman, M. Komu, Hypervisors vs. Lightweight virtualization: A performance comparison, in: 2015 IEEE International Conference on Cloud Engineering, 2015, pp. 386–393, http://dx.doi.org/10.1109/IC2E.2015.74.

[18] I. Fedotova, H. Eduard, Hu, A high-precision time handling library, J. Commun. Comput. (2013).

[19] F. Moradi, C. Flinta, A. Johnsson, C. Meirosu, On time-stamp accuracy of passive monitoring in a container execution environment, in: 2016 IFIP Networking Conference (IFIP Networking) and Workshops, 2016, pp. 117–125.

[20] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico, A. Pescapè, Characterization and analysis of cloud-to-user latency: the case of azure and AWS, Comput. Netw. 184 (2020) http://dx.doi.org/10.1016/j.comnet.2020.107693.

[21] L. Abeni, C. Kiraly, N. Li, A. Bianco, On the performance of KVM-based virtual routers, Comput. Commun. 70 (2015) 40–53, http://dx.doi.org/10.1016/j.comcom.2015.05.005, URL https://www.sciencedirect.com/science/article/pii/S0140366415001607.

[22] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2015, pp. 171–172, http://dx.doi.org/10.1109/ISPASS.2015.7095802.