Computer Communications 211 (2023) 193–215

Contents lists available at ScienceDirect

Computer Communications

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

Measuring the impact of SR-IOV and virtualization on packet round-trip time

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

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

ARTICLE INFO

Keywords:
Virtualization
SR-IOV
KVM
Active network measurements
Cloud management
Performance metrics

ABSTRACT

This article explores different data plane configurations for data center servers, focusing on scenarios involving SR-IOV in conjunction with KVM virtualization. Specifically, it evaluates the performance of different ways of using SR-IOV, including MacVTap "passthrough" mode with SR-IOV and PCI Passthrough on the Linux data plane. The study aims to shed light on how SR-IOV can be leveraged to optimize round-trip time (RTT) and resource usage. Through the experiments, we confirmed the expected behaviors of the technology but also clarified and identified behavioral aspects not yet known. For example, we confirmed that SR-IOV could improve RTT and outperform more traditional configurations, but we identified that this comes at the price of utilizing more computational resources, such as the CPU, depending on the orchestrated usage configuration. Furthermore, our evaluations also found that configuring the SR-IOV MacVTap in "passthrough" mode appears to be the best configuration when CPU resources are scarce or low or when scalability and flexibility are design requirements. In this article, we also expanded the SR-IOV assessment regarding the concurrency of this feature. Through these assessments, we identified that a VM without SR-IOV in a data center with other VMs that benefit from SR-IOV is inefficient from a performance point of view of communication. Thus, in this way, this study provides valuable information on how to optimize the use of SR-IOV in KVM environments to design efficient and high-performance virtualized data centers.

1. Introduction

Resource virtualization has proven beneficial across data centers, telecommunication systems, and computer networks. Virtualization is a software technology that slices down hardware and physical infrastructure, such as servers, storage, routers, switches, hard disks, network interface cards (NICs), and even software, such as operating systems, into logical, seemingly independent instances. Although it dates back to 1972, when it was first introduced by IBM in its 370 mainframe series to concurrently run different operating systems and applications in an isolated and protected manner, virtualization has gained significant momentum in the last decade with its use in the design of data center servers and emerging telecommunication technologies, such as the 5G architecture and Software Defined Networks (SDN). The sharing of server resources within data centers and the need to offer customers different operating systems and processing resources led to the adoption of server virtualization. Numerous data center hardware, communication, and software resources have been virtualized.

The result is cost reduction, optimization in the use of hardware, energy savings, and simplification in the management of data centers. From a software engineering perspective, virtualization obeys fundamental software design pillars such as reliability, flexibility, information security, integrity, and user privacy.

Despite its popularity and widespread adoption, several performance issues need consideration when using virtualization in data centers, the cloud, and telecommunication systems. They include performance loss experienced across I/O, network, and processing operations. Such a problem is primarily due to the overhead introduced through the presence of additional software processing layers that implement virtualization. Many strategies and solutions have been suggested to mitigate such performance problems, as pointed out, for example, in [1]. A case in point is Para-virtualization, a technology where a guest operating system is modified before deployment over a virtual machine (VM) instead of emulating an entire hardware environment [2]. Additionally, containers are increasingly seen as a more streamlined alternative to hypervisors with support for superior system

* Corresponding author.
E-mail addresses: assis.tiago@gprt.ufpe.br (A.T. de Oliveira Filho), eduardo.freitas@gprt.ufpe.br (E. Freitas), pedro.carmo@gprt.ufpe.br (P.R.X.d. Carmo), jamel@gprt.ufpe.br (D.F.H. Sadok), jk@gprt.ufpe.br (J. Kelner).
URLs: http://www.gprt.ufpe.br (A.T. de Oliveira Filho), http://www.djamelsadok.gprt.ufpe.br (D.F.H. Sadok).

https://doi.org/10.1016/j.comcom.2023.09.013
Received 18 April 2023; Received in revised form 12 July 2023; Accepted 8 September 2023
Available online 12 September 2023

efficiency. As a result, container adoption in data centers and cloud computing systems has recently been expanding [3,4].

In the wake of the growing demand for virtual environments to achieve performance close to bare-metal environments [5], lower processing latency, and reliability, several networking solutions have emerged. Nonetheless, this work mainly focuses on advances made in the context of the data plane responsible for packet processing. For instance, in 2008, Intel introduced the Virtual Machine Device Queues (VMDQ) technology, an Intel (R) Virtualization Technology for Connectivity (Intel VT-c) component. VMDQ offloads packet sorting from the virtual machine manager (VMM), the hypervisor, to the network controller to accelerate network I/O throughput. VMDQ optimizes VM data traffic processing to improve bandwidth and CPU usage. Around the same time, the data plane introduced PCI Passthrough [6] and Single root I/O virtualization (SR-IOV) technologies. Both aimed at lowering packet processing latency by acting at the hardware level. Other emerging packet processing frameworks with similar goals include new packet frameworks such as the Berkeley Software Extensible Switch (BESS) and Fast Data I/O (FD.IO) [7], and Vhost/Vhost-User protocol. Unlike the previous approaches, which focused on portability, the Vhost/Vhost-User technology can be used in either user-space or kernel space, depending on the implementation. One aspect is implemented on the host (VHOST), while the other is implemented in the guest operating system. Despite the current presence of several technologies competing for data plane performance optimization, these can only be considered the ultimate solution that can stand out in some scenarios. Therefore, choosing the appropriate technology for a given use case is still an open research issue. The present work dedicates its effort to examining how a major supported data-plane technology, namely SR-IOV, affects the network performance of virtualized servers.

Section 2 provides an overview of Single root I/O virtualization technology, detailing its operation and use cases. SR-IOV is a hardware-based solution for virtualizing network interfaces to optimize network performance and reduce CPU overhead. In Section 3, we identify a set of key performance metrics (KPIs) and describe our measurement setup, including the testbed configuration and the software tools used to measure performance. Section 4 presents and discusses the achieved performance results for several different data plane configurations, including the use of SR-IOV in combination with MacVTap and the standard Linux bridge. We also evaluate two ways of deploying SR-IOV and MacVTap in "passthrough" mode and examine competition between VMs for SR-IOV resources and its impact on CPU usage and RTT metrics. In addition, we perform a statistical evaluation seeking to validate all our results.

Section 6 reviews related work on virtualization technologies and their impact on network performance. Finally, in Section 7, we report our conclusions and final considerations, highlighting the importance of SR-IOV for optimizing network performance in virtualized data centers and recommending its use in combination with other technologies such as containers and software-defined networks to achieve the best results.

This article comprehensively analyzes the Single root I/O virtualization technology and its impact on network performance in virtualized data centers. It presents a detailed measurement methodology and experimental scenarios, providing valuable information for researchers and practitioners working in virtualization and network performance optimization.

2. Virtualization and single root I/O virtualization

As noted in [8], virtualization support can impose a processing overhead that negatively impacts network application performance. Several software and hardware-based technologies have been developed to address this challenge to optimize cloud server data plane performance. Cloud networking often adopts a mix of solutions, where service-level agreements with tight performance indicators must be enforced. This study focuses on Single Root I/O Virtualization (SR-IOV), a powerful and widely adopted packet processing technology.

Single Root I/O Virtualization (SR-IOV) is a hardware-based networking technology that allows a single physical input/output (I/O) device to present itself as *multiple virtual devices* with similar capabilities [9]. I/O devices that support SR-IOV can range from Graphics Processing Units (GPUs) to Network Interface Cards (NICs). The SR-IOV architecture is defined by both physical and virtual PCIe functions, as outlined below:

- Physical Function Physical PCIe Function (PF): PF is the physical I/O device itself. PFs are "functions" that support the operational capability of the complete PCIe device. PFs are configured similarly to a conventional physical device and are responsible for advertising the SR-IOV capabilities to the remaining data-plane modules [9].
- Virtual Function Virtual PCIe Function (VF): A VF results from a virtual split of a PF performed by the SR-IOV technology. A VF does not have the same privilege and cannot access the resources of the physical device [10]. It is exposed to the system as a virtual network adapter. The maximum number of VFs in a single I/O device can vary among different PFs. VFs are mainly concerned with sending and receiving data and may not be used for supporting more advanced protocols and services, such as the Precision Time Protocol (PTP) or the IEEE MAC Security (MACsec), which are usually supported by common NICs [11].

VM access to SR-IOV VFs must carefully consider the advantages and potential limitations. Some of the latter are detailed in the coming sections. As SR-IOV is "simply" a physical technology, there are multiple ways to logically connect it to user space processes. However, regarding interfacing virtual machines and SR-IOV, two main designs dominate the scene. Both approaches are presented next.

2.1. PCI passthrough

The first architectural design for interfacing a VM to SR-IOV is PCI Passthrough. This technology utilizes the PCI bus to improve device I/O. PCI Passthrough is a mechanism that assigns host PCI devices directly to the virtual machines (VMs) and acts as a hypervisor bypass, as described by [6]. It allows a Virtual Machine (VM) to directly connect to the PCI device, behaving as if it was physically connected. Hence, with PCI Passthrough, the hypervisor no longer has to perform packet translation between the virtual machine and the host. Removing the hypervisor from the data plane path is a clear step to mitigate performance degradation caused by context switches and memory copies between VMs and the hypervisor during packet processing.

Although SR-IOV and PCI Passthrough appears to have distinct designs, they can work simultaneously. SR-IOV is typically utilized in combination with PCI Passthrough. Hence, instead of creating multiple VFs using SR-IOV and allowing each VMs to use standard drivers to gain access to a given VFs, each VF is assigned to a VM via PCI Passthrough, creating a direct PCI connection between the VM and the VF, one that bypasses the hypervisor. This setup enables further performance and scalability enhancements since each VM will directly gain access to a virtual function running on the network card.

2.2. Standard virtual drivers

Of course, SR-IOV can, by default, also be accessed by VMs via standard Linux drivers and virtual devices without the need for PCI Passthrough. There are, in fact, multiple network device drivers, including TUN/TAP and MACVLan (used to connect container interfaces directly with host interfaces) that offer VM an access interface to SR-IOV VFs. MacVTap is regarded as a preferred alternative due to its higher performance. Next, there is a brief description of both drivers.

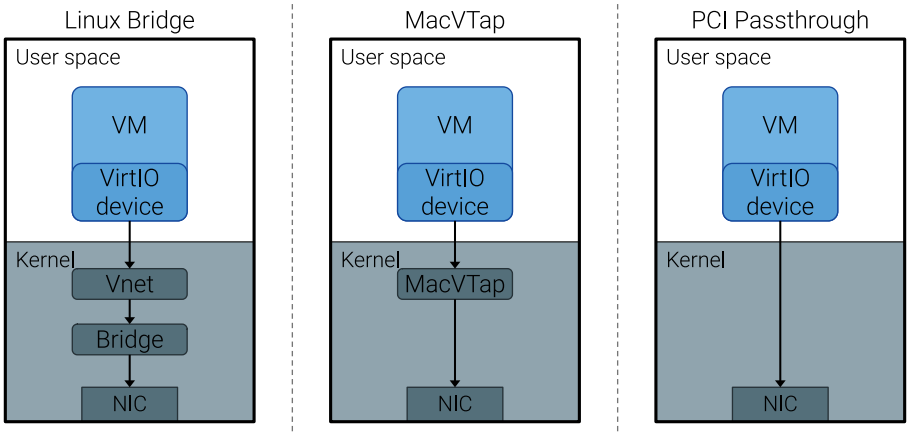

Fig. 1. Comparison of the discussed technologies to provide network access to virtual machines.

As a module for the administrator responsible for configuring network connectivity policies for virtual machine controllers, a Virtual Machine Manager (VMM) or hypervisor creates special devices inside the Linux OS to enable network communication with the VMs. Depending on how each VM connects with the NIC, these devices can be virtual switches or simple network interfaces. For instance, TUN/TAP devices implement layer two protocols. Note that these devices can be used for purposes other than virtualization, where they can also serve layer three protocols. They are executed in kernel space and are usually named as **Vnet**. TUN/TAP devices can be considered as virtual network interfaces. Instead of requiring a virtual switch to connect them to the physical card, they interact directly with the underlying network interfaces. These network interfaces are typically implemented using Linux **Bridges**, which facilitates the transmission of packets between the VMs and the physical NIC. For example, the standard configuration of KVM VMs uses Vnet devices connected to the physical NIC to provide network access among interconnected VMs. We also address this configuration in our experiments and examine its performance.

MacVTap is another virtual device that aims to improve how virtual machines connect to the network card and the external network [12]. It combines the TUN/TAP interface with MACVlan to create a single module with multiple layers of two virtual devices that can connect directly with KVM in user space – similar to TAP devices – but also where each device can have its own unique MAC address – like MACVlan device drivers. MacVTap combines the TUN/TAP and Bridge drivers with a single module based on the MACVlan device driver. Hypervisors such as KVM/QEMU directly access MacVTap. Both the guest VMs and the host show up directly on the switch that the host is connected to.

Similarly to MACVlan, MacVTap can work in private, Virtual Ethernet Port Aggregator (VEPA), Bridge (routing), and "passthrough" modes.

In private mode, the network card is exclusively assigned to the VM, allowing it to access the adapter's physical resources more directly without needing an additional processing layer, such as going through the hypervisor. Also, in private mode, each virtual machine connected to the physical card via MacVTap cannot connect to other guests and must go through an external router or gateway to communicate. This is useful in scenarios where the virtual machine needs to communicate with other network devices without sharing the network with other virtual machines. In VEPA mode, traffic from multiple virtual machines is *aggregated* and passed to a single physical network device (NIC) via MacVTap. The packets are sent to an external switch, which will forward packets back to the NIC and then to the VMs. This allows multiple virtual machines to share a single physical network adapter, which can be more efficient in using physical resources and managing network policies. Unlike the Linux Bridge device previously explained, Bridge mode connects all VMs, enabling communication between direct internal VMs without needing external switch forwarding.

Finally, "passthrough" mode, which **should not** be confused with PCI "passthrough", is when one virtual NIC attaches to the physical NIC and **prevents** the NIC from creating other virtual devices, giving a sense that the vNIC "takes over" the NIC. The vNIC inherits the NIC's MAC address and sets it to promiscuous mode, which enables the vNIC to receive every packet sent to the NIC.

Because it supports various features, MacVTap is the appropriate virtual driver when using SR-IOV [13]. As the network card is virtually sliced into VFs, connecting each VM to a VF through the MacVTap allows every VF to guarantee a dedicated MAC address with virtual layer two device functionality. Fig. 1 illustrates the discussed network access technologies.

3. Experiment setup

This section describes the testbed used to run the experiments presented in this study. Given our interest in network performance within a cloud computing environment, typical cloud virtualization software, VMs, and their interfaces comprise the adopted testbed partially inspired by [8,14].

The testbed contains two host servers, emulating typical cloud servers, connected following a peer-to-peer (P2P) topology. Each server runs a virtual environment, which includes two types of VMs. The first type refers to the "Main" VMs. These are the monitored VMs where we extract the main performance metrics. The second type of VMs is the Background ones. These VMs are there to emulate the presence of competition for processing and network resources. As such, the Background VMs are responsible for running additional workloads designed to stress the servers for resources. We use the Kernel-based Virtual Machine (KVM) hypervisor to drive our virtual environment.

A given experiment must first configure and launch the "Main" VMs, followed by the background VMs. Then, the background ones. We are particularly interested in evaluating SR-IOV performance. The Round-Trip Time (RTT) between two "Main" VMs is commonly used to indicate network performance, including SR-IOV. However, it is important to note that using Ping to measure fine-grain RTT may not provide precise results, as it involves additional steps such as ARP REQ, ARP REP, and unicast traffic. One expects that the presence of different virtualization layers involved in packet processing combined with the data considered plane technologies will impact packet processing times which in turn leads to changes to RTT. We used Ping, the Packet Internet Groper echo-based tool, to estimate the RTT, although we cannot provide accurate results for RTT measurements in detail due to the additional steps involved. However, Ping offers several benefits, making it a valuable tool for our experiments. Its simplicity and wide availability allow administrators and network administrators to use it easily. In addition, Ping is versatile, allowing the measurement of RTT

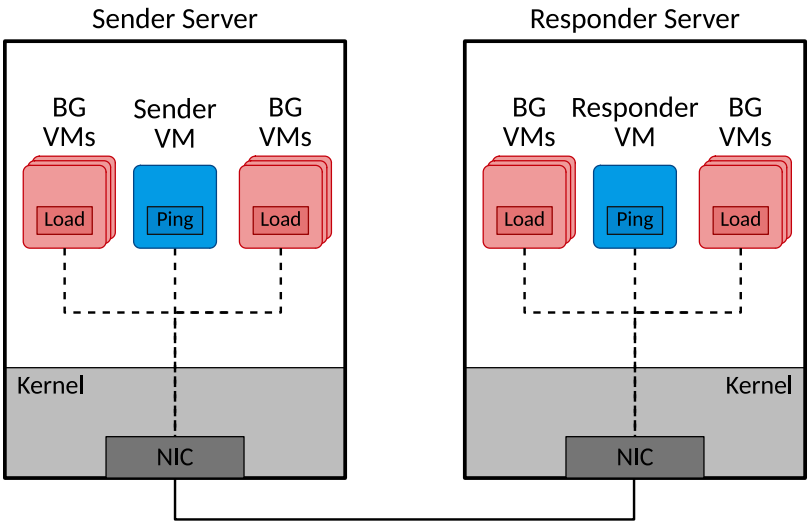

Fig. 2. Testbed configuration.

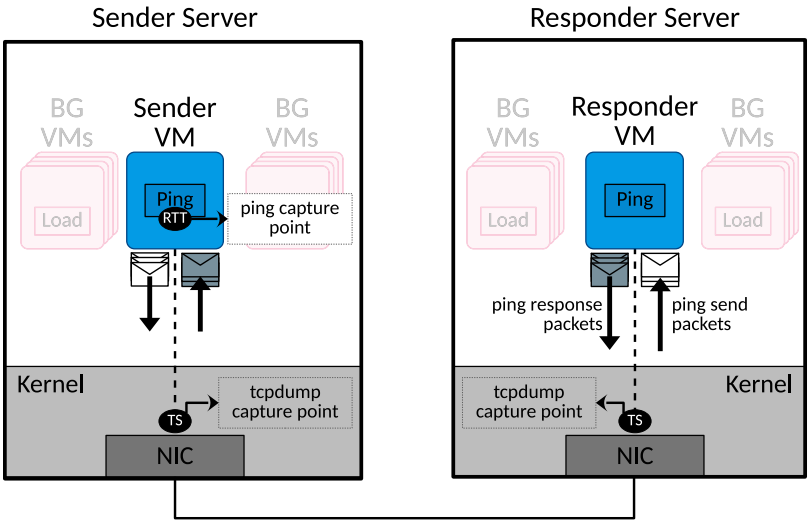

Fig. 3. Experiment flow.

between different network devices, such as servers, virtual machines, or routers, which offers flexibility in evaluating network performance. Its ICMP-based standardization ensures compatibility across different platforms and networks. Additionally, Ping can be used to troubleshoot network connectivity issues, and identify packet loss or high latency, aiding in optimization and troubleshooting. By comparing RTT values obtained from different network configurations or experimental conditions, Ping enables an analysis of the impact of various factors on network performance. However, it is important to note that Ping may not be suitable for RTT measurements in details that provide granularity and high accuracy, and more advanced tools or methods are recommended in these situations. The VMs are named according to their type ("Main" or background VM) and location (Sender or receiver). The **Sender VM**, hosted on the **Sender server**, is responsible for *sending* the ping request packets to the **Responder VM**, hosted on the **Responder server**. The *Responder VM* echos back the received ping packet requests to the **Sender VM**. Fig. 2 depicts the experiment testbed and its server and VM setup. Tables 1 and 2 list the hardware and software specifications of the used servers, respectively.

More importantly, we also introduced extra timestamp capture points on both servers. These additional capture points improve the evaluation of the Round-Trip Time (RTT) in our experiments, making it possible to record the timestamp values of the ping packets in

Table 1
Server's hardware specification.

SO	Ubuntu 20.04.6 LTS
Kernel	5.4.0-47-generic
SO(Guest)	Debian 11
Kernel (Guest)	5.10.0-16-generic
Processor	2x Intel(R) Xeon(R) Bronze 3204
RAM	64 GiB
HD	ATA-DISK DELLBOSS VD
NIC	Intel x540-AT2 10 Gigabit

Table 2
Server's major software configuration.

Software	Version
qemu-kvm	4.2.1
virsh	6.0.0
ping	iputils-ping 3:20190709-3
tcpdump	4.9.3
sysstat (mpstat)	12.2.0

several steps, and we use the tcpdump tool for this. By recording the timestamp of all ping packets as they pass through the server kernel, we make it possible to calculate the RTT from different locations beyond the "Main" virtual machines (VMs), including the RTT between the network interface card (NIC) and virtual network interface.

Fig. 3 illustrates the capture points adopted for the ping packets and the corresponding locations where the timestamp values are collected. This expanded set of RTT observation points gives us a more comprehensive understanding of the packet journey and allows us to identify possible causes of delays in packet processing.

The tool tcpdump has been widely adopted as a network monitoring tool for various fundamental reasons. Its compatibility and availability across multiple operating systems make it a convenient and flexible choice for deployment in different network environments. By allowing real-time packet capture and analysis, tcpdump provides immediate information about current traffic, enabling immediate detection of bottlenecks and anomalies. Through advanced filtering capabilities, it is possible to carry out an accurate and targeted analysis, focusing only on packets relevant to the desired assessment. Detailed analysis of packets captured by tcpdump reveals valuable information about headers, data, and other metadata, enabling an in-depth understanding of network performance. In addition, its tight integration with other data analysis and visualization tools, such as Wireshark, further extends its capabilities, allowing for comprehensive network assessment.

While tcpdump may not provide absolute accuracy in measuring RTT, it does play a crucial role in real-time packet capture. With other metrics and monitoring tools, tcpdump complements our RTT assessment, providing valuable insight into network behavior, including identifying bottlenecks, anomalies, and latency variations. We gain a holistic understanding of network performance across our experiments by leveraging tcpdump with other performance indicators such as CPU load, memory utilization, and network throughput. In this way, tcpdump stands out as an essential tool for the comprehensive evaluation of the RTT, even considering its possible accuracy limitations.

3.1. Experiment variables

This section describes the metrics and factors adopted in the experiments and explains their selection to analyze the impact of the SR-IOV and virtualization on packet processing delay.

3.1.1. Performance metrics

As earlier stated, there is a need not only to observe the end-to-end overall RTT but also to isolate delays caused by virtualization from the one the NIC is responsible for. Both metrics are described next.

- **RTT-based Evaluations**: The RTT end-to-end user level metric is established as an essential network performance indicator [8,14–16]. It can be used to extract estimates of delay, congestion, and bandwidth that network flows experience over a network.
  - **RTT Overhead**: This second metric expresses the difference between the RTT computed by the ping tool on the "Main" VMs and the RTT recorded by the tcpdump kernel-level tool reporting when packets cross the NIC of the *Sender Server*. This metric represents the time required for packets to travel between the *Sender VM* and the network adapter of the *Sender Server*, providing insights into intra-server packet transmission.
  - **Response Time**: This third metric measures the RTT portion of a packet, which encompasses the time taken from the *Responder Server's* NIC to the *Responder VM* and back to the NIC, highlighting the internal response time within the Responder.
    It represents the time a packet travels from the NIC to the VM and back to the NIC again. The response time, internal to the Responder, can help identify areas where network latency may be causing delays or packet transfer slowdown.

Apart from measuring RTT, we also capture additional metrics that contribute to obtaining a more comprehensive understanding of the behavior of the data plane. One such metric is CPU usage.

- **CPU usage**: Indicates the amount of processing power being utilized by the Sender Server. By monitoring CPU usage, one can evaluate the impact of network traffic on the Sender Server's resources and assess how efficiently it is handling the load.
  When evaluating packet processing optimization solutions, it is crucial to consider their potential side effects, as these proposals can significantly affect the system's overall performance. Although designed to improve network performance and reduce latency, they can also, as a side effect, require the use of other critical computational resources, such as CPU, causing unwanted or unexpected consequences such as higher energy consumption and/or causing possible performance bottlenecks. Monitoring CPU usage aids in root cause analysis. Higher packet processing delays may be attributed to the performance of the packet processing strategies, the type of virtualization and its overhead, or the presence of high demand for CPU resources due to background processing or packet processing software itself. For example, some well-known packet processing technology solutions for optimization, such as GRO (Generic Receive Offload), GSO (Generic Segmentation Offload), LRO (Large Receive Offload), and TSO (TCP Segmentation Offload), require more CPU cycles to process or perform packet segmentation in exchange for better network performance. For instance, RSS (Receive Side Scaling) enables distributing the processing of network traffic across multiple CPU cores.
  Therefore, evaluating the tradeoffs between CPU usage and performance is essential when considering these technologies. Additionally, in our experiments, we utilized the *sysstat* application, specifically the *mpstat* command, to measure CPU usage (Table 2) and assess its impact on packet processing and overall system performance. This allowed us to analyze the correlation between CPU utilization and network performance, providing insights into the efficiency of the packet processing strategies and their interaction with CPU resources. It is important to note that high CPU usage can slow down other processes running on the system, resulting in degraded performance of other applications. This can be particularly problematic in virtualized environments where multiple virtual machines share a single physical host.

It is essential to draw attention to the fact that not all metrics are available for all experimental setups. For example, in the experiments where we used PCI Passthrough, we could not calculate *RTT overhead* nor *Response Time* since the NIC's PCI bus used by the main VM is directly attached to the VM (hence bypassing the kernel), preventing the tcpdump kernel tool from accessing the NIC to timestamp the packets. For each scenario, the metrics that could not be collected are mentioned.

3.1.2. Adopted performance factors

There is a need to select the right factors for any experiment. The factors employed in this study are described as follows:

- **Number of Background VMs**. This factor indicates the number of virtual machines that will share and compete for resources in the testbed's cloud environment. The values taken are **0, 8, 32, 64, or 128** virtual machines. These values were defined according to previous results obtained by [8,14].

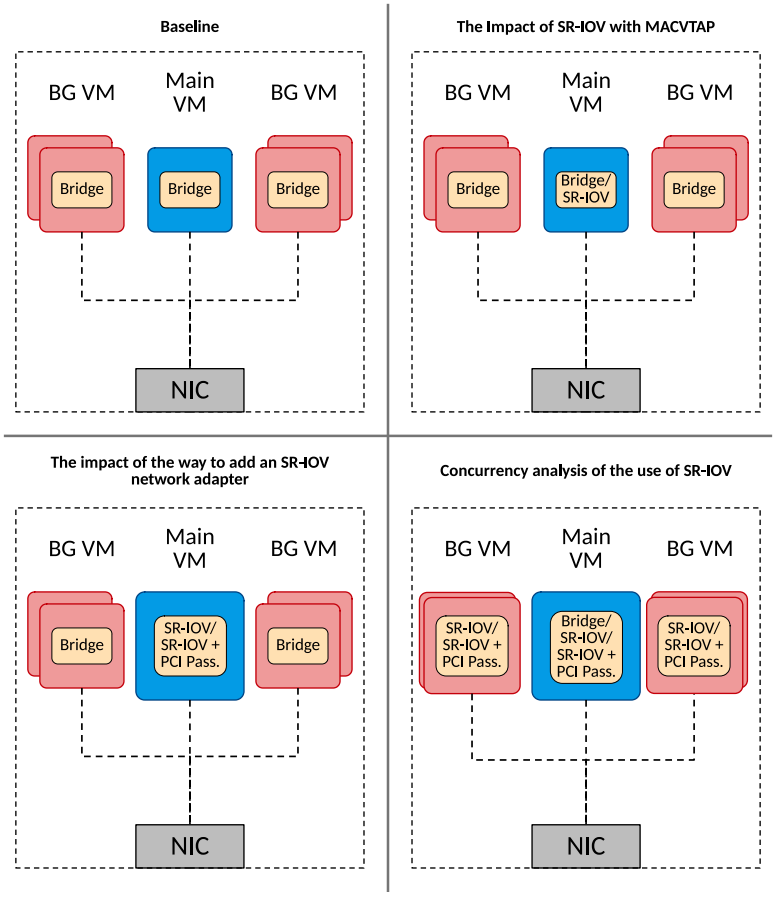

Fig. 4. SR-IOV scenarios on every experiment.

- **Load Type**. Indicates the type of resource usage the background VMs will execute. The type of load may either be **Network Load** or **CPU Load**. The network load results from creating VMs on both servers and sending packet bursts from all the VMs on one server to the VMs of the other one.
- **Load Location**. This factor specifies the location of the load type which may either be at the **Sender Server** or the **Responder Server**. In the case of using **Network Load** type, this factor represents the direction of packet bursts. For example, if the load location is said to be *Responder server*, this corresponds to having packet bursts being transmitted from the background VMs on the *Sender Server* to the *background VMs* on the *Responder Server*.
- **Driver**. The Driver factor is an important consideration when connecting virtual machines (VMs) to network cards. Several drivers are available to choose from, including E1000, Bridge, SR-IOV, and SR-IOV + PCI Passthrough.
  - **E1000**: The E1000 device is used inside the VMs and connects to a Linux bridge inside the kernel.
  - **Bridge**: The VirtIO device is used inside the VMs and connects to a Linux bridge inside the kernel.
  - **SR-IOV**: The VirtIO device connects to an SR-IOV VF via a MacVTap driver through the "passthrough" mode. It is important to note that this mode should not be confused with PCI Passthrough, which provides a VM with direct access to PCI hardware. Initially, we evaluated the joint use of SR-IOV and MacVTap in two modes: "passthrough" and "bridging". However, due to its implementation characteristics, we excluded the "bridging" mode from our evaluation. In the "bridging" mode, MacVTap functions as a virtual router, routing packets to the SR-IOV adapter and allowing the VM to access the adapter through MacVTap. This introduces additional overhead and potential bottlenecks in the communication structure. In "passthrough" mode, the SR-IOV adapter is directly assigned to the virtual machine, enabling direct access to the adapter's physical resources. To clarify the configuration, SR-IOV is implemented on the host, while MacVTap is created through the host's SR-IOV. In this setup, MacVTap exists on the host, and VirtIO is on the virtual machine (guest). We omitted the VEPA and private modes in our experiments as their characteristics were incompatible with the objective, methodology, and scope of this work, as explained in Section 2.
  - **SR-IOV + PCI Passthrough**: The VirtIO device connects to an SR-IOV VF directly to the PCI bus via PCI Passthrough.
- **Ping Frequency**. This factor specifies the frequency of packets sent from the *Sender VM* to the *Responder VM*. The available frequencies are:
  - **1 kpps**: 1000 packets per second;
  - **1 Mpps**: 1,000,000 packets per second;

The experiment's adopted packet size is the minimum Ethernet packet size of 64 bytes. The number of packets sent per experiment is set to 100 whereas all experiments are repeated 10 times. Consequently, the total number of packets sent in each evaluated scenario is 1000.

4. Results and discussion

This section showcases the outcomes of our experiments and offers a comprehensive analysis of the reported findings. We designed four experiments, each intended to evaluate a specific group of network drivers. Fig. 4 summarizes the experiments and illustrates the considered driver diversity.

4.1. Baseline experiments

Initially, we conducted an experiment seeking to validate the behavior obtained in [8] in the new testbed and isolate the performance of the RTT when not using the SR-IOV technology. In addition, we update this study with newer 10GbE network cards, as listed in Table 1. This time, we only used two virtual network drivers: the E1000 emulated and para-virtualized VirtIO drivers. This decision was driven by the results reported in [8], where the authors concluded that overall the E1000 driver provides better performance over the emulated network drivers and that VirtIO outperforms all the reing tested drivers. For ease of understanding, the results are individually compared for each adopted metric.

RTT: This metric understandably tops the list of metrics five it reflects the end-to-end delay. Our experiments show that the VirtIO driver consistently outperforms the E1000 driver regarding median RTT, even when no *Background VMs* are present. Thus, overall, VirtIO demonstrates superior RTT performance as depicted in Figs. 5 and 6. This behavior was expected since the para-virtualized drivers have been purposely optimized for virtual environments. VMs using VirtIO do not have to go through most of the steps needed to emulate a virtual NIC [17]. For example, one of the mechanisms used by the VirtIO driver is the technique called "virtqueue", which allows direct communication between the virtual machine and the host machine without emulating a network card [8]. In other words, it has been designed to work closely with the hypervisor to provide better performance and lower overhead.

On the other hand, emulated drivers, such as E1000, are entirely virtual network cards emulating actual physical network adapters through software. The E1000 driver, for example, emulates an Intel 82545EM Gigabit Ethernet Controller. Furthermore, as the VM kernel will need to emulate the exact behavior of the given card in real-time, this requires more processing power, time, and CPU cycles, as each frame transmitted or received from the guest machine's OS will require extra processing (such as multiple hypervisor queries or data copies), which results in this category of drivers processing fewer packets per second compared to other methods.

In addition, RTT is greatly affected by the *Load Type* and *Load Location*, regardless of the virtual driver used. As shown in Fig. 5, the CPU load on the Sender Server significantly impacts the RTT, resulting in measurements ranging from 0.2 ms to 25 ms when 128 background VMs are used. This indicates a high variation in the measured values. Additionally, the median increases from approximately 0.1 ms (with no background VMs) to 0.9 ms (when having 128 background VMs). Conversely, Fig. 6 shows that when the load is applied at the *Responder server*, the RTT is more precise, particularly with the VirtIO driver, and overall lower, with only a few outliers exceeding 10 ms.

On the other hand, the Network Load has a similar impact on both the Sender and Responder. As soon as 8 background (BG) VMs are deployed, the RTT gains a low variation but higher values until there are 128 VMs when the RTT becomes more imprecise reporting outlier RTT values reach close to 1000 ms.

The E1000 emulated driver features a bulky overhead RTT that considerably influences the final performance. In some cases, as with the experiments where load location is the *Responder server*, *Load Type Net* and having more than 64 background VMs, the *RTT Overhead* dominates the RTT and contributes to its value by approximately 60% to 80%. When changing the load location to *Sender server*, we also obtain this effect: the overhead being about ≈25% to ≈45% of the total RTT. On the other hand, the para-virtualized VirtIO driver offers lower overheads compared to the E1000. Overall, the highest *RTT Overhead* reached when using the emulated E1000 driver with load location *Responder server*, Load Type is *Network Load*, and with the number of VMs above 64, where we observed ≈20% contribution to the total RTT. Thus, we identified that the *Type of Load* and *Load Location* factors significantly impact this metric. Load Type being *Network Load* affects RTT very negatively. In contrast, this does not impact RTT greatly when *Load Type* is *CPU Load*.

Response Time: In general and as expected, VirtIO's para-virtualized driver provides the shortest times relative to a metric response time. VirtIO always more or less halves the response time compared to the one obtained when using the E100 driver (e.g., with response time values of ≈0.1 and ≈0.05 ms, when the number of background machines ranges between 0 and 128, respectively). The highest response time achieved using the E1000 driver (for example of ≈0.2 and ≈0.1 while using background machine values of 0 and 128, respectively) ends up, once again, higher than the ones reached when using the para-virtualized driver.

When applying background network traffic (i.e. selecting *Load Type* as *Network Load*) at the Responder server and having 128 background VMs, the response time to a ping frequency of 1Mpps contributes to RTT with ≈0.6 and ≈0.35 for the E1000 and VirtIO drivers respectively.

Similar performance is observed with the experiment with *Load Location* set as *Sender server* and *Load Type* set to *Network Load*. However, the difference between the VirtIO stream yield is smaller than what was observed and commented earlier when using 128 background VMs for the E1000 equivalent, with a time of ≈0.17, against ≈0.48. As previously stated, in all our scenarios where the *Load Type* is *CPU Load* and *Load Location* is *Sender server*, VirtIO outperforms E1000. The response time has been defined, even if the flow is higher or much higher (about an amount of higher background machines).

Summary: Our baseline experiments validated as expected the fact that the VirtIO driver achieves the lowest RTT values, particularly regarding CPU usage. Furthermore, overall, we found that the Load Type and Local Load factors are the ones that impact the most RTT.

Regarding the variation of the RTT, we observed the presence of a minor one, in other words, more stability, in the experiments configured with load location in the *Responder server*. In contrast, when we conducted our evaluations with Load Type as being *CPU Load*, and its location being *Sender server*, we noticed a significant relative variation in the results. This was evident when using 32 background machines. In this same configuration, both the para-virtualized VirtIO E1000 emulated drivers achieved close results. Furthermore, with this first set of experiments, the frequency of echo messages did not significantly influence the RTTs' medians, behavior, or other measured metrics.

Regarding the *RTT Overhead* metric, we noticed that setting the Load Type to *Network Load* causes this metric to deteriorate when using both drivers, even more in the case of the E1000.

Concerning the *Response Time* metric, we found that the VirtIO para-virtualized driver reaches the shortest response times in all controlled experiments.

As mentioned in [8], emulated drivers face performance issues as they attempt to recreate the functionality of a network card in software, leading to increased complexity and processing overhead. In contrast, Virtio is a para-virtualized driver technology consisting of a front end (software driver) and a back end (device driver). The front end, located within the virtual machine, emulates a Virtio-compatible device interface and communicates with the back end on the hypervisor side. This para-virtualized architecture makes drivers aware of the virtualized environment, enabling more direct processing by utilizing the simplified Virtio interface and leveraging the hypervisor as an intermediary for communication with the physical device. Consequently, para-virtualized drivers eliminate the need to replicate the entire network card architecture and achieve more efficient and streamlined performance.

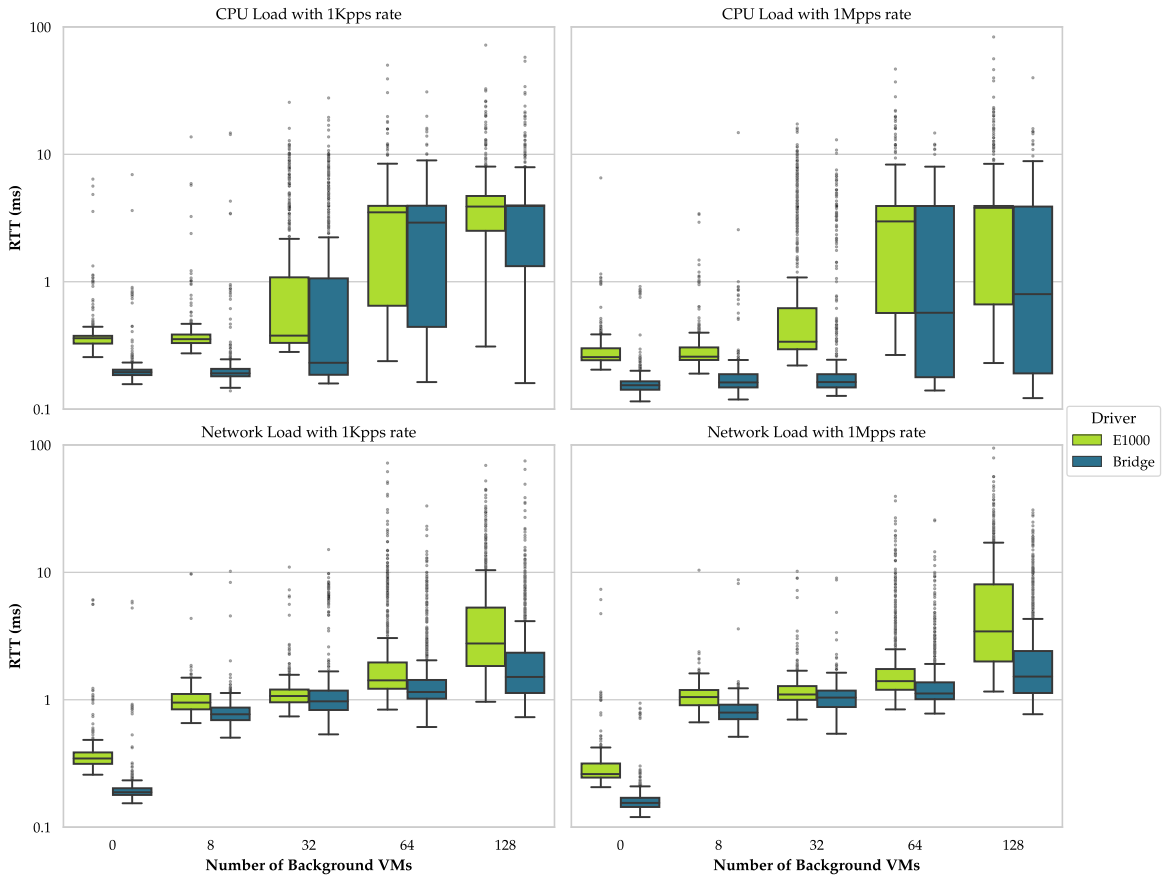

Fig. 5. Results for the baseline experiments. This graphic summarizes the performance of each virtual driver without SR-IOV, considering the RTT when the Load is applied on the Sender Server.

Finally, from these experiments, we also observed that the increase in the network card's transfer capacity due to the use of new and faster network interfaces does not seem to change the behavior of the virtual drivers.

4.2. Impact of SR-IOV with MacVTap

We proceeded with the execution of the planned experiments using the VirtIO driver consistently. This time, the testbed hosts were equipped with SR-IOV-capable network interfaces. In this section, we present the results obtained from utilizing SR-IOV in conjunction with MacVTap network devices. We provide a detailed explanation of how SR-IOV and MacVTap operate in Section 2. We then compare the performance of SR-IOV with that of the baseline standard Linux bridge described in Section 4.1.

As part of these new experiments involving SR-IOV and MacVTap, we designed two network settings for the KVM-based testbed. The first setup, referred to as "SR-IOV", utilizes the MacVTap virtual device in its "passthrough" mode. The second setup, named "Bridge", represents an alternative to the traditional implementation of virtual network configuration, utilizing the Vnet device with Linux Bridge. It is important to note that both scenarios exclusively use the VirtIO network driver. Initially, only the "Main" VMs on both Sender and Responder servers leverage the use of SR-IOV, as depicted in Fig. 4. This enables us to observe the impact of the other background VMs that do not utilize SR-IOV.

RTT: The results demonstrate that SR-IOV when used in conjunction with MacVTap can provide an overall lower RTT when compared to the "Bridge" network, as seen in Figs. 8 and 7. The Main VM with SR-IOV gains access to a dedicated virtual network card with an exclusive virtual PCI bus. In contrast, the "Bridge" configuration mode introduces an extra processing layer in the form of a virtual bridge on the host machine, which causes overhead in network performance due to additional packet processing.

Considering the type and location of the load, there is a pattern similar to the one exhibited by the baseline, where the CPU load on the Sender server significantly increases RTT variation, resulting in a higher RTT median.

Despite this, the most notable performance differences between the two approaches in this first set of experiments are apparent when the Load Type is CPU and Load Location is set to Sender (Fig. 7). In this case, using the "Bridge" operation mode with VirtIO resulted in an RTT approximately twice as high as SR-IOV's. For instance, when running 64 VMs in the background, the RTT was approximately ≈0.45 with SR-IOV and approximately ≈0.98 with the "Bridge" mode technology.

In some cases, SR-IOV technology may be more advantageous. For example, both technologies present similar RTT results when conducting experiments with load location being the *Responder server* and Load Type being the Network Load while subjected to more than 64 background VMs 8. We believe that the high volume of data generated in such configurations causes the network interface to excessively use the queue, creating a bottleneck that inhibits the advantages reached under other configurations.

Regarding RTT variation when using SR-IOV, we could not identify any significant performance improvement. This behavior was unexpected, as we believed isolating the SR-IOV and MacVTap mechanisms would provide better stability than using the "Bridge" configuration. However, this observed behavior is no longer surprising. When using SR-IOV with MacVTap, the virtual network adapter still needs to go through some abstraction layers, including the virtualization layer, network stack, and physical network adapter, consequently causing

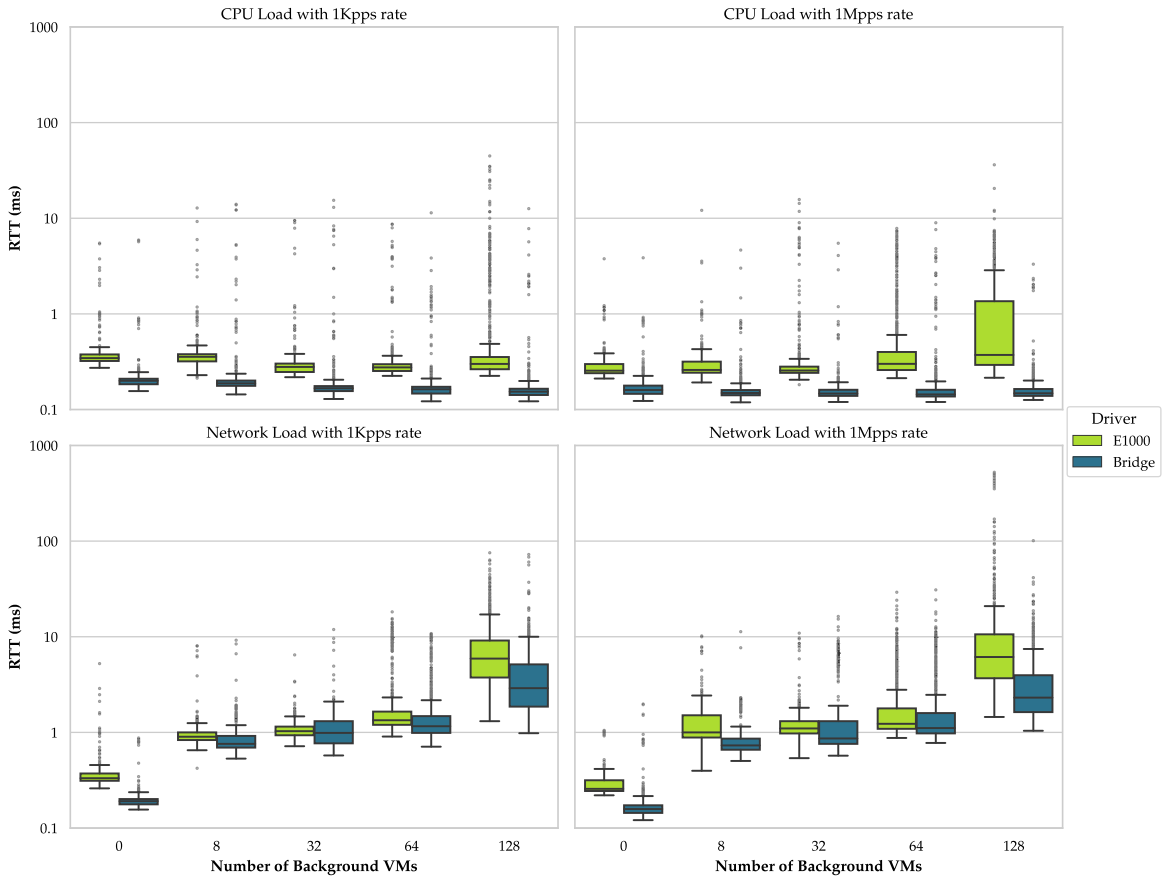

Fig. 6. Results for the baseline experiments. This graphic summarizes the performance of each virtual driver without SR-IOV, considering the RTT when the Load is applied on the Responder Server.

additional processing delay. Following such a path directly influences the observed increase in RTT variation.

Overall, virtualization overhead is lower with SR-IOV than the traditional "Bridge" configuration shown in Fig. 9. However, exceptions exist, such as when conducting experiments on the *Responder server* subjected to network load (Fig. 10). In these cases, when the number of background machines reaches 128, we observe that RTT overhead in milliseconds for SR-IOV is almost twice smaller than the "Bridge" configuration. In this case, the SR-IOV and "Bridge" scenarios achieve ≈2.0 ms and ≈1.0 ms respectively packets are echoed at the rate of 1 kpps. For this same configuration, however, when increasing the Ping Frequency to 1 Mpps, we identified that SR-IOV's RTT increases by as much as four times reaching ≈2.0 ms against ≈0.5 ms for the "Bridge" setup. This behavior raised our curiosity, especially given that despite the higher overhead, at the end of the day, this influence was not impacting enough to make the final median RTT of the SR-IOV greater than that of the "Bridge" mode configuration scenario.

Furthermore, it is worth noting that several parameters can affect the RTT time, such as interrupt coalesce time even with a kernel device driver. These parameters may introduce additional variability in the observed RTT values. While our experiments focused on comparing the overall RTT overhead between SR-IOV and the "Bridge" configuration, it is important to acknowledge that specific system configurations and parameter settings can influence the RTT measurements. Future investigations could explore the impact of these parameters in more detail to provide a comprehensive understanding of the RTT performance under different configurations.

A possible explanation of this behavior could be because when the Load Type is *Network Load*, and the Load Location is the *Responder server*, the latter sends excessive network packets to the *Sender server*. As a result, the exchanged packets enter the receive queue, occupied by other packets from the background network traffic load, causing a high *RTT Overhead*. There is no difference in behavior if we also look at the number of packets sent per second. This evidence shows that in some cases, the processing of packets inside the SR-IOV + MacVTap can be higher than when using the standard Linux bridge.

Response Time: Due to their more direct nature than the Bridge methodology, the experiments conducted with SR-IOV and MacVTap achieved the lowest Response Time.

Although we expected a more significant performance difference between the two approaches when evaluating the results where load location is the *Sender server* due to the Bridge's intermediary mechanisms (such as vnet, among others previously mentioned). Overall, the obtained values were very close, with SR-IOV having a slightly shorter time.

Note that the observed performance gap becomes more dominant as the number of background virtual machines increases beyond 64, regardless of whether the load type is CPU or Network. For instance, with 64 and 128 background machines (1Mpps), we observed *RTT Overhead* values of approximately ≈0.06 and ≈0.09 using SR-IOV. This contrasts with the higher values of ≈0.12 and ≈0.16 observed using the "Bridge" mode. That is the amount of data and not the type of Load seems to have more influence when we observe the metric Response Time in this context 12. When Local Load is located at the *Responder server*, we witness a more significant disparity in the performance of Response Time over the variation in the number of background virtual machines between the two approaches. Having the configuration of 128 background machines and the Load Type being *Network Load* at its highest rate, namely 1Mpps, the collected times were ≈0.40 and ≈0.20 for "Bridge" and SR-IOV, respectively as depicted in 11.

The explanation for this phenomenon is consistent with the hypothesis discussed earlier in this subsection when examining the median RTT metric.

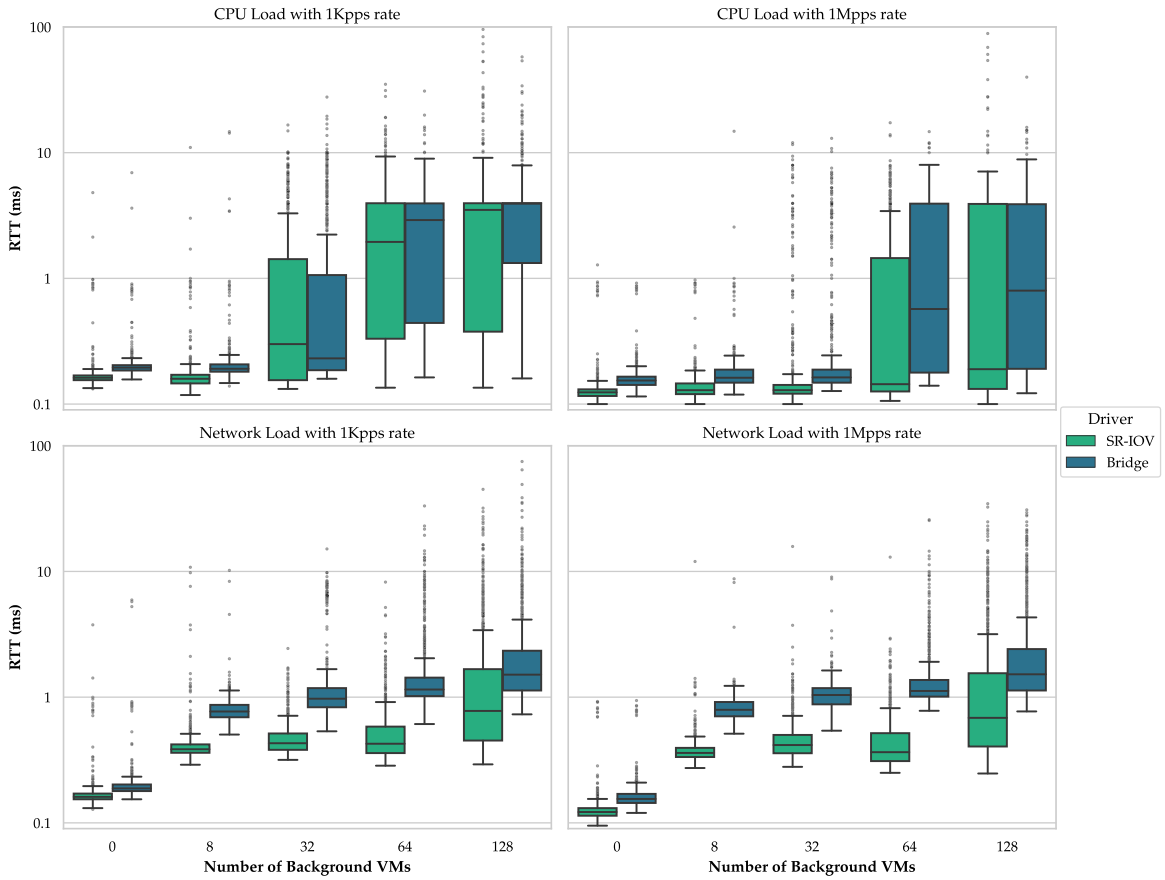

Fig. 7. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median of all experiments on the Sender Server.

Summary. Overall, one observes that the joint use of SR-IOV and MacVTap reduces the overhead and processing required to manage virtual networks, thereby lowering RTT and speeding up packet processing at the data plane. The result is a better overall network performance when compared to when using the traditional "Bridge" network approach. However, this research identified some cases where SR-IOV technology with MacVTap did not make any noticeable difference. This was the case when the traffic destination server was subject to considerable background network load, and hosted many VMs (128 background VMs in our tests) that competed for resources.

Regarding the Response Time, the experiments performed with SR-IOV and MacVTap obtained the lowest response time compared to when using MacVTap with the "Bridge" mode. However, there was a limited performance difference between the two data plane configurations when evaluating the results with *Sender server*, even though the "Bridge" mode employs several intermediate mechanisms. As the number of background machines increased beyond 64, the performance gap between the two approaches became more prominent, regardless of the background Load Type being applied (network or CPU), with SR-IOV outperforming Bridge. The amount of data has more influence on the response time metric in this context. When the Load Location was on Responder, there was a significant disparity in performance between the two approaches. Bridge suffered a higher response time than SR-IOV, particularly when reaching the 128 background VM configuration and the competing injected Load Type of additional network traffic.

On a positive note, neither SR-IOV nor MacVTap caused a noticeable variation in RTT.

Based on the real results presented, it is evident that the use of SR-IOV in conjunction with MacVTap can be precious compared to the Bridge and VNET options. In high-performance virtualization environments where latency and bandwidth are critical to performance, SR-IOV can significantly reduce CPU overhead and network latency. Furthermore, the combination of SR-IOV with MacVTap allows for additional flexibility in network configuration, allowing system administrators to adjust network settings to meet the specific needs of their workloads. In summary, SR-IOV with MacVTap can be a powerful choice for virtualization environments demanding high performance and network flexibility.

Some real examples, in high-traffic load environments, SR-IOV can avoid network bottlenecks, while using MacVTap minimizes virtual switch overhead. In environments where latency is critical, SR-IOV and MacVTap can help minimize latency by allowing traffic from the VM to be sent directly to the underlying physical device. Additionally, in critical security environments, using SR-IOV and MacVTapp can help ensure the integrity of VM traffic by preventing other users or VMs on the network from intercepting or manipulating the traffic.

4.3. The influence of the SR-IOV connection method

Recall that MacVTap can be configured in any of four different modes. This determines how a MacVTap device communicates with the lower device in the KVM host. Communication between a source device in the KVM host using the Virtual Ethernet Port Aggregator (VEPA) default mode requires an external switch. As a result, it moves to switch out of the server and frees resources for use by the VMs. This mode is mainly used in data centers and cloud systems as it allows administrators to manage virtual machine networking at the switch level, which is desirable. Note that switches aware of the VEPA guests can enforce filtering and bandwidth limits per MAC address without the host knowing about it. Unlike VEPA, the Bridge mode directly connects VMs to each other. It, therefore, removes the round trip through the external bridge. This setup is used when inter-guest communication is

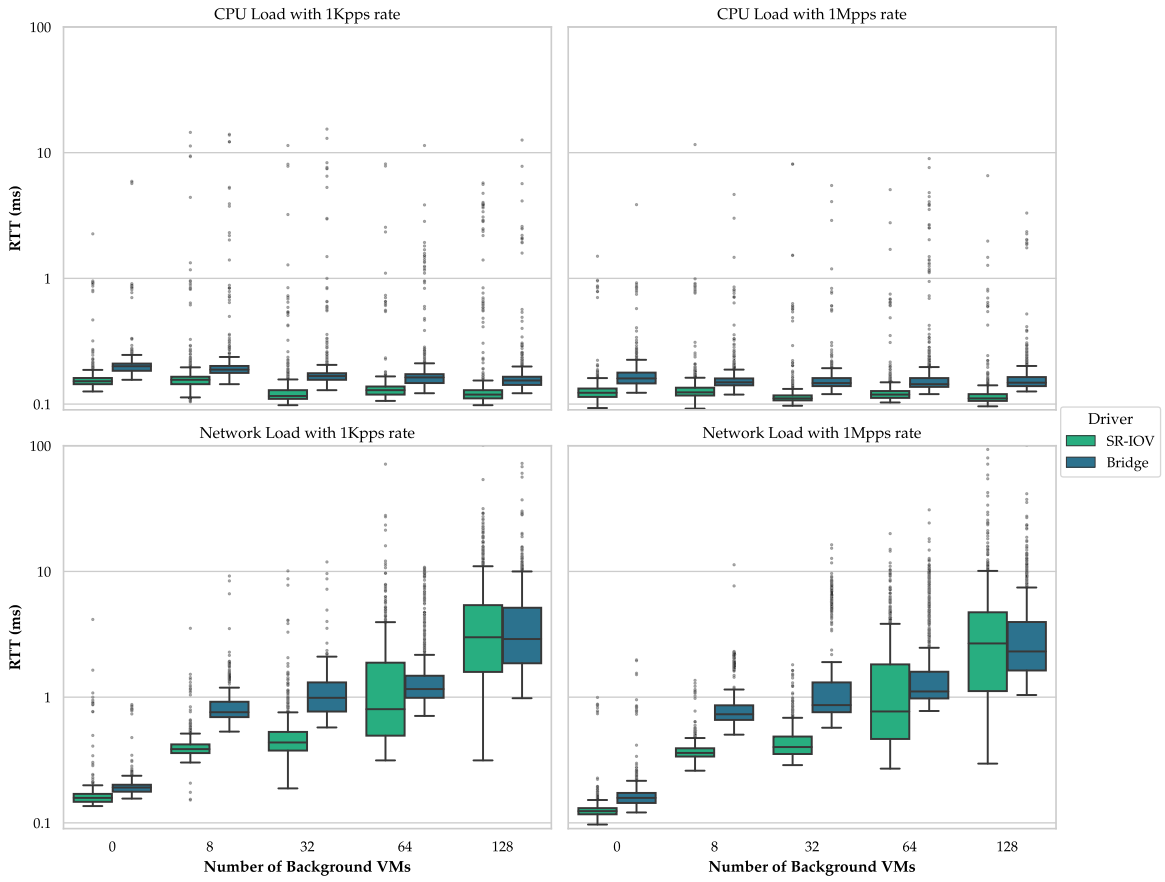

Fig. 8. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median of all experiments on the Responder Server.

performance critical as it tends to reduce RTT in such cases. The MacVTap "passthrough" mode, not to be confused with the PCI-Passthrough technology which may apply to any PCI device, attaches a physical interface device or a SR-IOV Virtual Function (VF) directly to a guest VM. As a result, all packets are sent directly to the designated network device. Note that there is a one-to-one correspondence between the guest and the network device. In other words, guests cannot share a device or a VF in "passthrough" mode. This mode is interesting to this study as it skips traffic processing in kernel MacVTap. Instead, the hardware takes processing in charge, releasing Host CPU resources. Finally, the third MacVTap mode, known as private, operates similarly to VEPA but does not allow nodes on the same MacVTap device to talk to each other. This mode is selected when the goal is to isolate VMs connected to the same physical interface.

Given the encouraging results achieved by the combined use of MacVTap "passthrough" and SR-IOV, presented in Section 4.2 we chose to compare these with an alternative data plane setup that uses SR-IOV and PCI-Passthrough. PCI Passthrough connects a VM directly to the VF as explained in Section 2.1. Note that only the "Main VM" of both servers uses these two configurations whereas the remaining background VMs continue using the standard bridge to connect to the NIC (i.e. without using VF and SR-IOV). Fig. 4 illustrates the adopted scenarios.

RTT: The joint use of SR-IOV and PCI Passthrough, as expected, achieves the lowest RTT values. This is justified due to giving VMs, in our case the server's "Main VM", direct access to the virtual PCIe. The achieved maximum and median RTT values were ≈118 ms and ≈0.09 ms, respectively. The latter represents a value around 1.723 times lower than when using SR-IOV and MacVTap. A close look at Fig. 13 also shows that the experiments where the *Responder server* competed with 128 background VMs, used a ping rate of 1Mpps while applying a Load Type in the form of background network traffic, achieved the highest RTT.

In comparison, even when the background load is located at the *Responder server*, the experiments based on the use of MacVTap have a maximum RTT in the order of ≈199 ms, obtained when such background load is of the type *network Load* combined with a high number of machines in the background (more significant when there are 64 or more). In these experiments, the performance discrepancy between the approaches becomes more significant due to the steep increase in the number of packets that need to be processed.

From the *Sender server* perspective, the median RTT values are similar to those reached in *Responder server* based experiments. However, the experiments orchestrated with increased CPU Load Type exhibited significant variability in RTT values. The medians for SR-IOV + PCI Passthrough configured experiments continued to achieve a lower RTT, with values ranging between ≈0.047 and ≈119 ms, obtained when using a 1Mpps ping frequency to measure RTT while running 64 and 128 background VMs, respectively 14.

CPU Usage: The goal next was to capture CPU usage (remembering, we use *mpstart*, see the Table 2 in Section 3.1.1) levels when having different background loads located at the Sender or Responder servers. Initially, *CPU Usage* is extracted for a "Bridge" (Vnet VirtIO), the results of which are shown in Figs. 15 and 16.

The two approaches, namely SR-IOV with MacVTap, and SR-IOV + PCI Passthrough exhibited similar behavior with that of the "Bridge" approach. The observed differences are not substantial, with the values ranging from 45.94% to 46.9%. This was the case when there was network background traffic at *Sender server* and 128 VMs competing for processing resources with "Main VM".

CPU usage partially was mostly as expected. This performance metric did not differ a great deal among SR-IOV + PCI-Passthrough

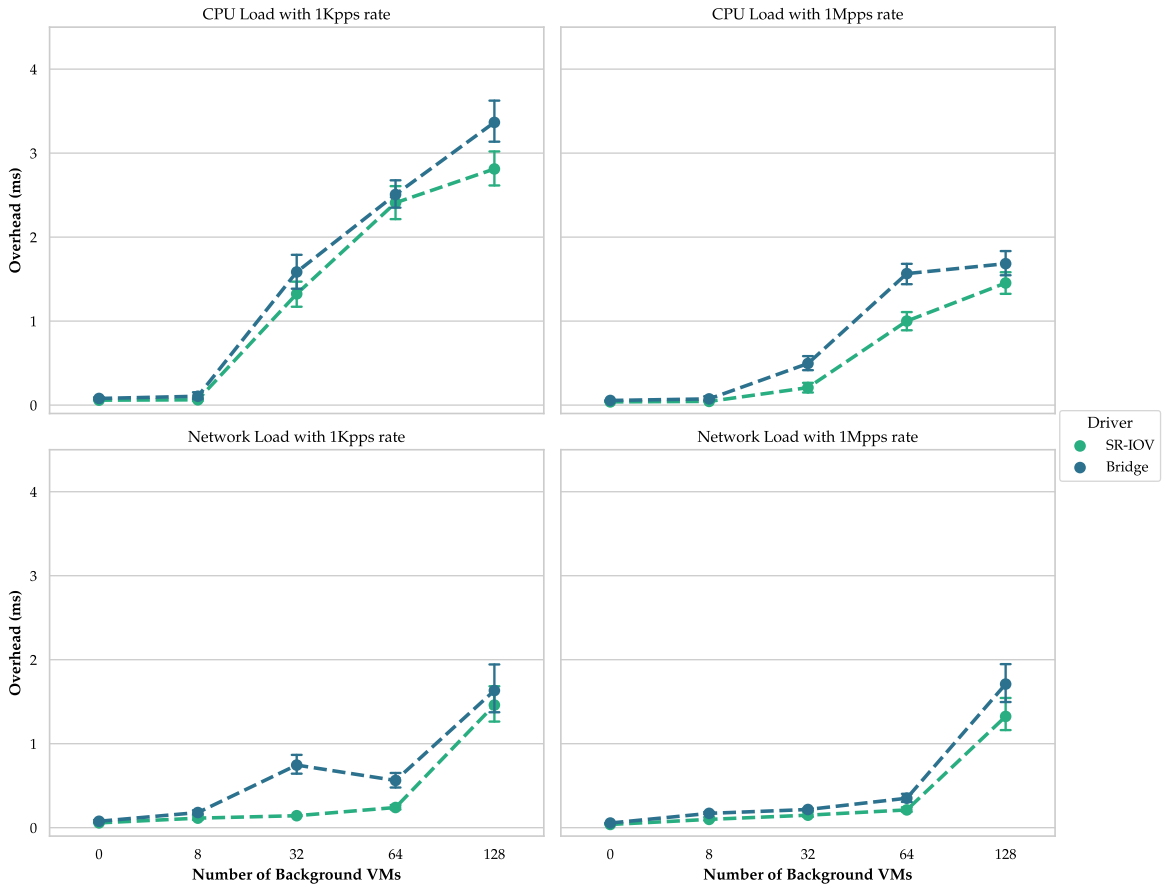

Fig. 9. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the Overhead of all experiments on the Sender Server.

and SR-IOV with MacVTap because both data plane strategies rely on hardware offloading, which allows data transfer to be handled by the physical network interface card (NIC), hence reducing CPU load. Observe that factors such as the need for a more complex NIC allocation involving the hypervisor and the virtual interface in packet processing tasks when using the SR-IOV MacVTap did not influence CPU usage. However, this was hardly the case for the "Bridge" approach. Using the Virtio Vnet Bridge would require the CPU to perform additional work as it has to process and fully manage the virtual network interfaces.

Note that the simple topology used in this scenario somehow explains the above performance similarity in CPU usage between the approaches SR-IOV with MacVTap and SR-IOV with PCI-Passthrough. Recall that only the two "Main" VMs (each running at a server) use SR-IOV and share server resources with the colocated background VMs using the traditional Bridge configuration. Since SR-IOV only applies to a single VM, the CPU load from one driver is not as strong. Hence, in such scenarios, the CPU load will not directly affect the packet processing performance at the host regardless of the driver used whether SR-IOV with MacVTap, SR-IOV with PCI-Passthrough, or the standard "Bridge". To get around this, we developed a new set of experiments with configurations that allowed us to more accurately contemplate the SR-IOV technology impact on the CPU usage metric.

Summary: The choice between the SR-IOV + PCI Passthrough and SR-IOV with MacVTap approaches depends on various factors and considerations. The SR-IOV + PCI Passthrough approach offers the lowest RTTs, as demonstrated in the experiments. However, it comes with certain limitations, such as hardware support requirements and challenges in sharing physical devices among multiple virtual machines. On the other hand, SR-IOV with MacVTap provides more flexibility. Directly assigning virtual functions (VFs) to individual virtual machines using the available virtual functions provided by SR-IOV-compatible hardware allows multiple virtual machines to share network resources from a single physical card while maintaining their MAC addresses and network settings. However, this approach may involve some performance trade-offs compared to SR-IOV + PCI Passthrough. The choice between these approaches should be based on specific use case requirements, desired performance levels, and the feasibility of hardware modifications. It is essential to carefully evaluate these factors to determine the most suitable approach for achieving the desired balance between performance, hardware support, and flexibility in a virtualized environment.

For example, the SR-IOV + MacVTap approach can be beneficial in a flexibility-oriented data center profile. Imagine a scenario where the data center hosts multiple virtual machines with dynamic network requirements. By utilizing SR-IOV + MacVTap, administrators can efficiently allocate and reconfigure network resources for different virtual machines without requiring extensive hardware modifications. This allows for quick adjustments to network configurations, making it ideal for environments with constantly changing workloads, diverse networking needs, rapid provisioning of virtual machines, and varying traffic patterns.

On the other hand, in a performance-centric data center profile where achieving the highest level of performance is crucial, and hardware modifications are feasible, the SR-IOV + PCI Passthrough approach can be employed. Consider a scenario where a data center requires ultra-low latency and high network performance for specific virtual machines. In this case, SR-IOV + PCI Passthrough can be utilized. By directly assigning virtual functions (VFs) to individual virtual machines using the available virtual functions provided by SR-IOV-compatible hardware, these VMs gain exclusive access to the physical network resources, resulting in lower latency and improved network performance. This approach may involve more significant hardware

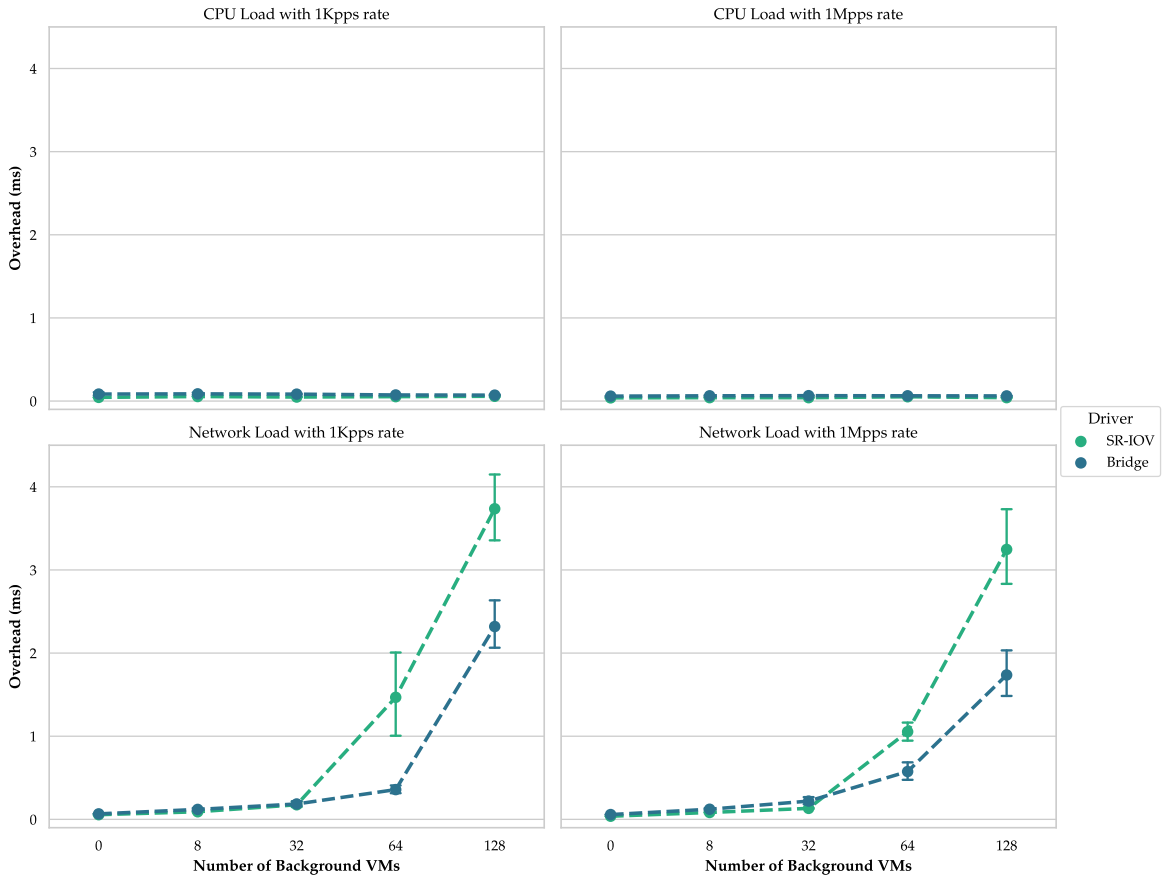

Fig. 10. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the Overhead of all experiments on the Responder Server.

modifications, such as dedicated SR-IOV-capable network interface cards (NICs) or additional physical network infrastructure, to support the SR-IOV feature. However, it ensures optimal performance for critical workloads that demand high network efficiency and low-latency communication.

Furthermore, when considering availability, combining MacVTap and SR-IOV in "passthrough" mode offers advantages over PCI Passthrough. With MacVTap "passthrough", multiple virtual machines can share the network resources of a single physical card. In the event of a failure in one virtual machine, the remaining VMs can continue functioning without interruption, ensuring high availability of network services. On the other hand, with PCI Passthrough, each VM has exclusive access to the physical network interface, and a failure in one VM can lead to the unavailability of the assigned network resources until the issue is resolved.

These examples illustrate how the choice between SR-IOV + MacVTap and SR-IOV + PCI Passthrough depends on the specific requirements of the data center profile. By carefully considering factors such as flexibility, performance needs, the feasibility of hardware modifications, and availability requirements, administrators can select the most appropriate approach to achieve the desired balance between flexibility, performance, and availability in their virtualized environment.

4.4. Analyzing competition by SR-IOV VF usage

This scenario examines how SR-IOV performs concurrently in a data center environment. An important difference introduced to the data plane is that unlike in the previous experiments where only a server's "Main VM" may gain access to SR-IOV, we now allow all VMs to access the network interface using SR-IOV. The idea is understanding how the different VMs' competition for SR-IOV resources affects RTT and CPU usage metrics. In addition, VMs usage of SR-IOV varied across the different experiments. Overall, these experiments were divided into: (a) a scenario where all VMs (background and the two main ones) use SR-IOV + PCI Passthrough; (b) a second one where the VMs use SR-IOV with MacVTap in "passthrough" mode and (c) Finally a third data plane configuration allows only the background VMs to gain access to SR-IOV + PCI Passthrough. Fig. 4 details the above three configurations. Due to the limited number of VFs supported by the used NIC, the number of background VMs was limited to 64 instead of 128 in these new experiments.

RTT: when all VMs share access to an SR-IOV with PCI Passthrough data plane, the lowest RTT values are achieved as shown in Figs. 17 and 18. There are nonetheless some specific cases, such as when we select to apply background network Load at the *Responder server* while having a large number of background VMs (32 and 64), the SR-IOV MacVTap (without PCI Passthrough although using MacVTap "passthrough" mode) configuration outperforms the others.

Regardless of the frequency level used for sending echo packets by the "Main VM", all the above three data plane configurations exhibited a high variation in the RTT values for cases with more than 32 background VMs when the locating load on the *Sender server*. Note however that RTT median values were lower due to using SR-IOV.

CPU Usage: The use of PCI Passthrough (in first and third data plane configurations) influences CPU usage. The highest CPU usage levels ranged between ≈95.15 and ≈93.19% registered at the *Responder server*. This was the case when network background traffic was inserted together with the increased number of background VMs to 32 and 64, see Fig. 19. The combined use of SR-IOV and MacVTap reports a ≈18.81% reduction in CPU usage. However, a strange behavior was observed when only 8 background VMs were used. The scenarios

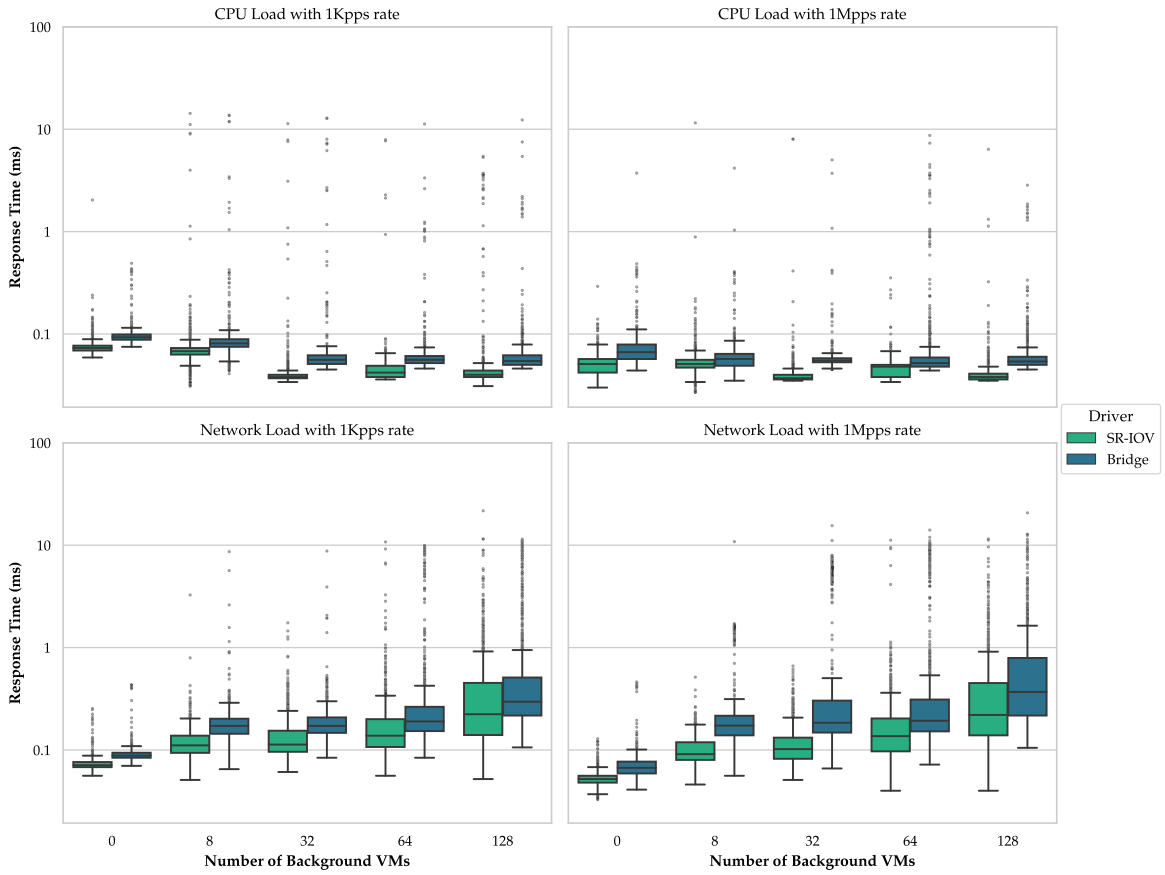

Fig. 11. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the Response Time median of all experiments on the Responder Server.

using PCI Passthrough, corresponding to the first and third data plane configurations, reduced CPU usage of ≈25.91% and ≈26.09%.

Also, when the two SR-IOV usage approaches compete (first and second scenarios), distinct CPU usage levels are observed. This depends on the number of background VMs present and if there is background network traffic. The MacVTap approach reduces CPU usage by ≈20% when using a high number of background VMs (such as 32 and 64), see Fig. 20.

SR-IOV with PCI Passthrough setup allows VMs to access the NIC directly, offloading network processing tasks to the physical device. This results in higher CPU usage, particularly under high network traffic loads. In contrast, the SR-IOV MacVTap configuration has a more complex NIC allocation that involves the hypervisor and virtual interface in the necessary packet processing tasks, leading to a distribution of activities and lower CPU usage. On the other hand, the "Bridge" approach fully processes network traffic through a virtual network adapter, emulated by the hypervisor, resulting in high CPU usage. The hypervisor must emulate the entire network stack, including the NIC hardware, driver, and firmware, and perform memory mapping, interrupt virtualization, and resource allocation. These factors explain the observed behavior.

This behavior of the CPU usage contemplated with the experiments indicates that the SR-IOV MacVTap approach seems justifiable when achieving good performance while maintaining computational resources at manageable levels.

Summary: When the background VMs use SR-IOV, except the "Main VM", the tendency is to suffer RTT performance loss. In addition, RTT worsens as more background VMs are introduced into a server due to their competition for CPU consumption. This suggests that having a VM that does not use SR-IOV in a context where others have access to SR-IOV VFs will lead to sub-optimal RTT performance for such a VM.

The combination of SR-IOV and PCI Passthrough and its use for all VMs withstands higher CPU usage to a greater degree than when using SR-IOV MacVTap at the data plane. This is especially true when in the presence of many background VMs (such as 32 and 64). On the other hand, when there are few background VMs, such as 8 in our experiments, located at the *Responder server*, SR-IOV MacVTap consumes more CPU than the other two evaluated alternatives. All methods suffered from high RTT variation when the *Sender server* a considerable number of background VMs (as many as 64) while subjected to additional CPU overhead caused by the consumption by background tasks.

When the goal is to control CPU usage for a data center server and that all its VMs are using SR-IOV, the *Sender server* should use MacVTap in "passthrough" mode. This is evident with a high level of background traffic and many background VMs.

5. Statistical analysis

In this section, statistical tests were performed using indicators presented as evaluation metrics. The goal is to confirm the observed performance levels across the different data plane technologies considered in this work. When conducting our research, we chose to use the Wilcoxon Test and Kendall Correlation due to the intrinsic characteristics of these statistical tests and the nature of our data.

To this end, we first used the Wilcoxon Signed Rank Test (Wilcoxon) [18]. This non-parametric method indicates whether a pair of two paired groups with two attributes have a significantly different distribution. We selected a confidence level of 95% for our observations. The proposed hypotheses tests are based on the following simple question:

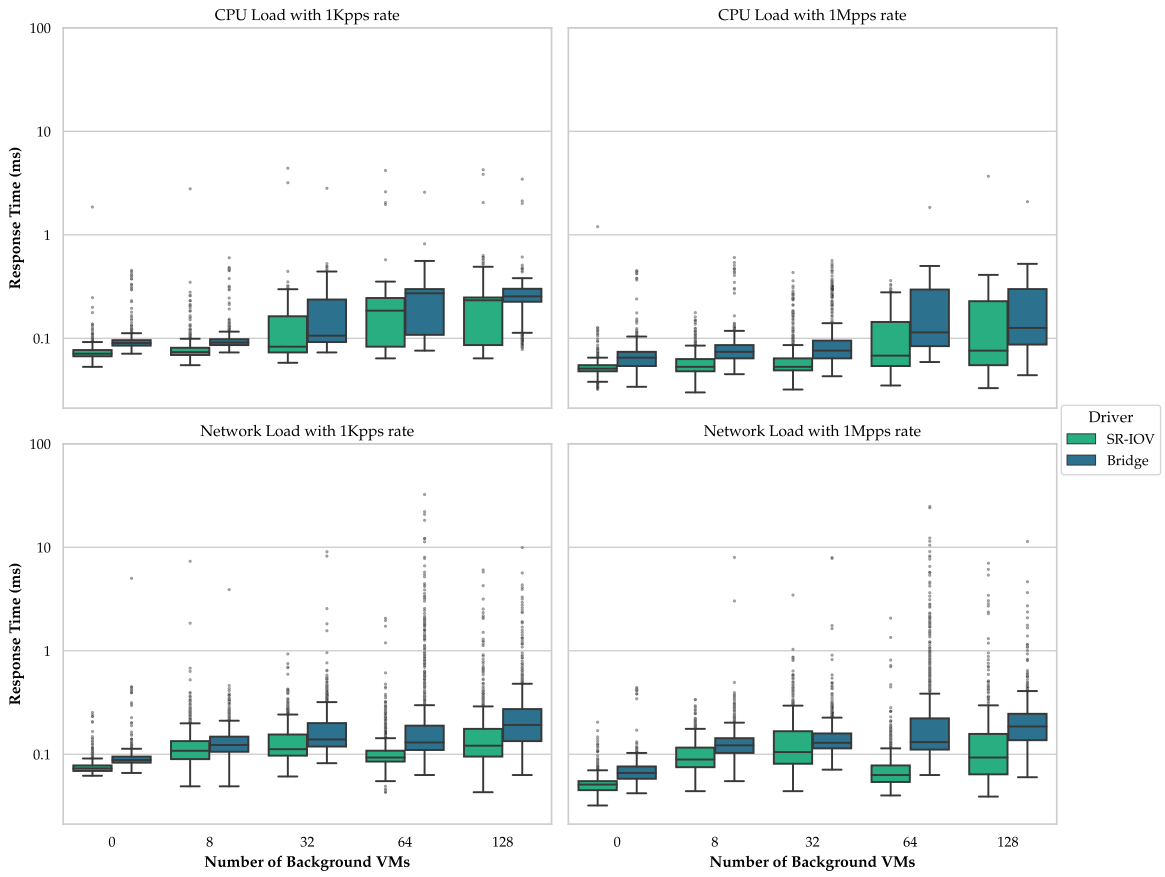

Fig. 12. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the Response Time median of all experiments on the Sender Server.

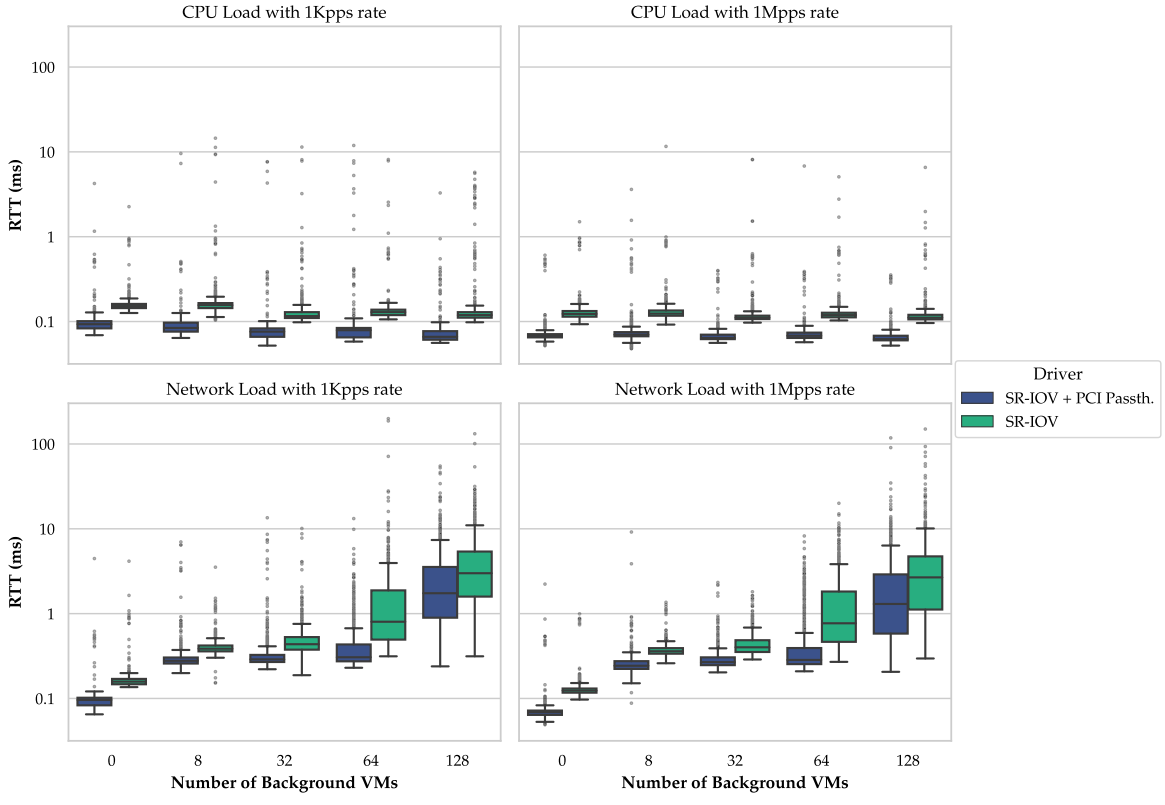

Fig. 13. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median values of all experiments on the Responder Server.

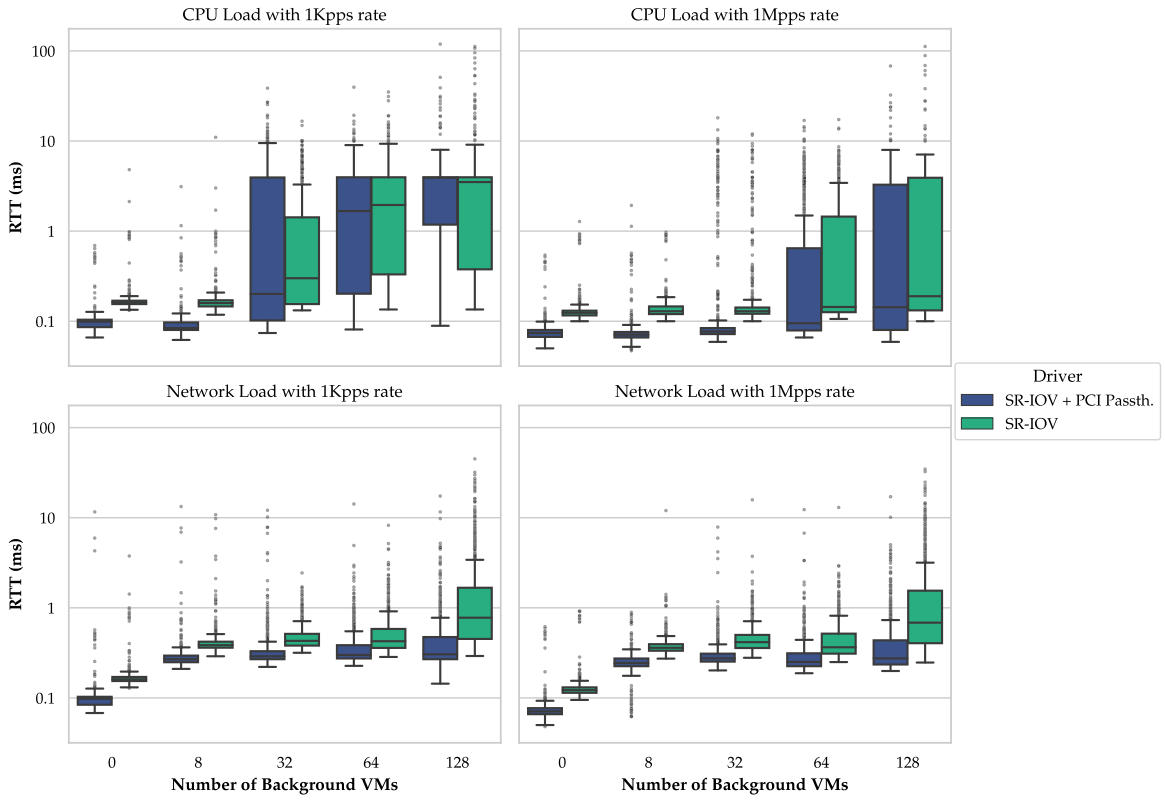

Fig. 14. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median values of all experiments on the Sender Server.

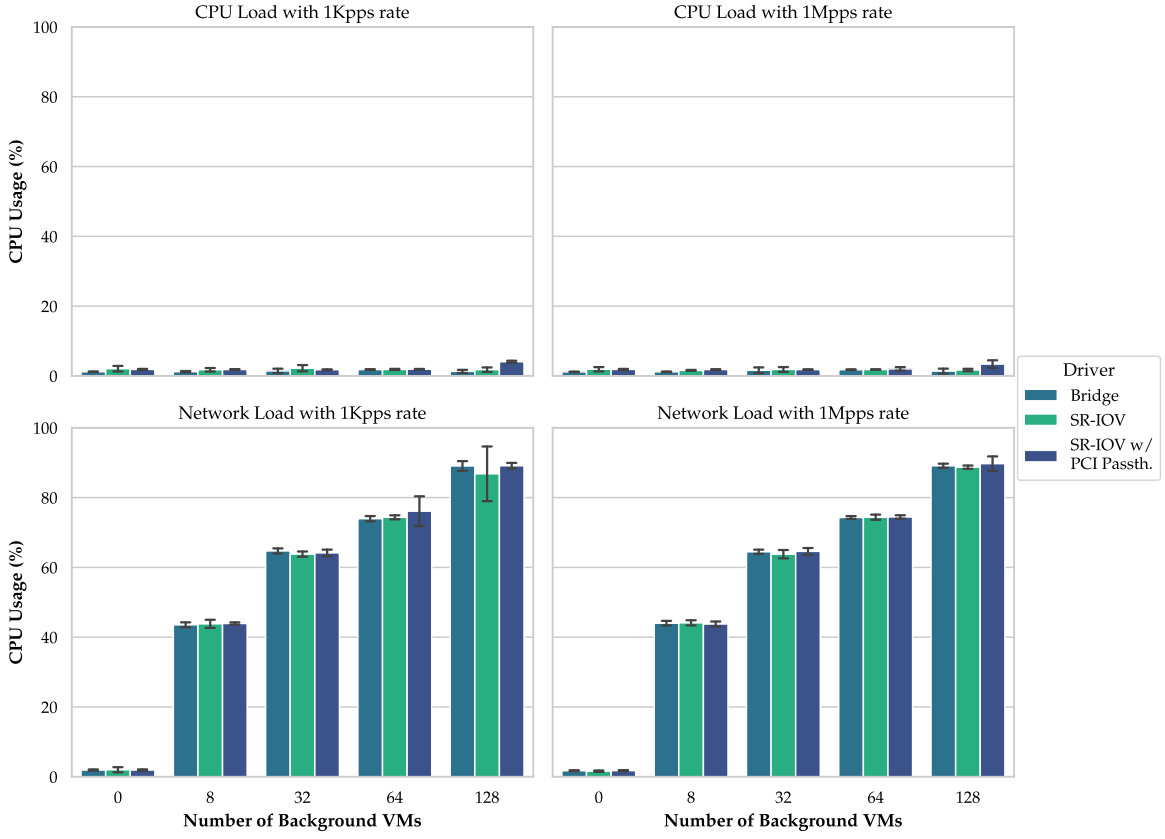

Fig. 15. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the CPU values of all experiments on the Responder Server.

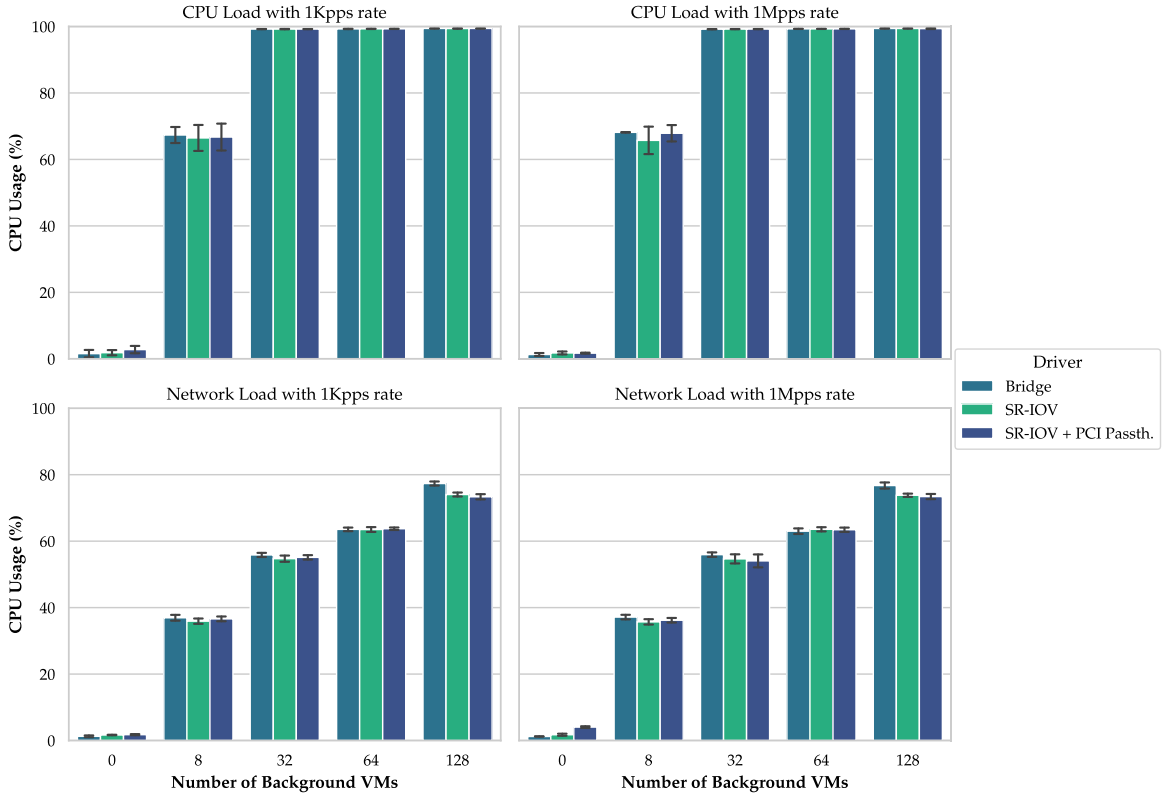

Fig. 16. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the CPU values of all experiments on the Sender Server.

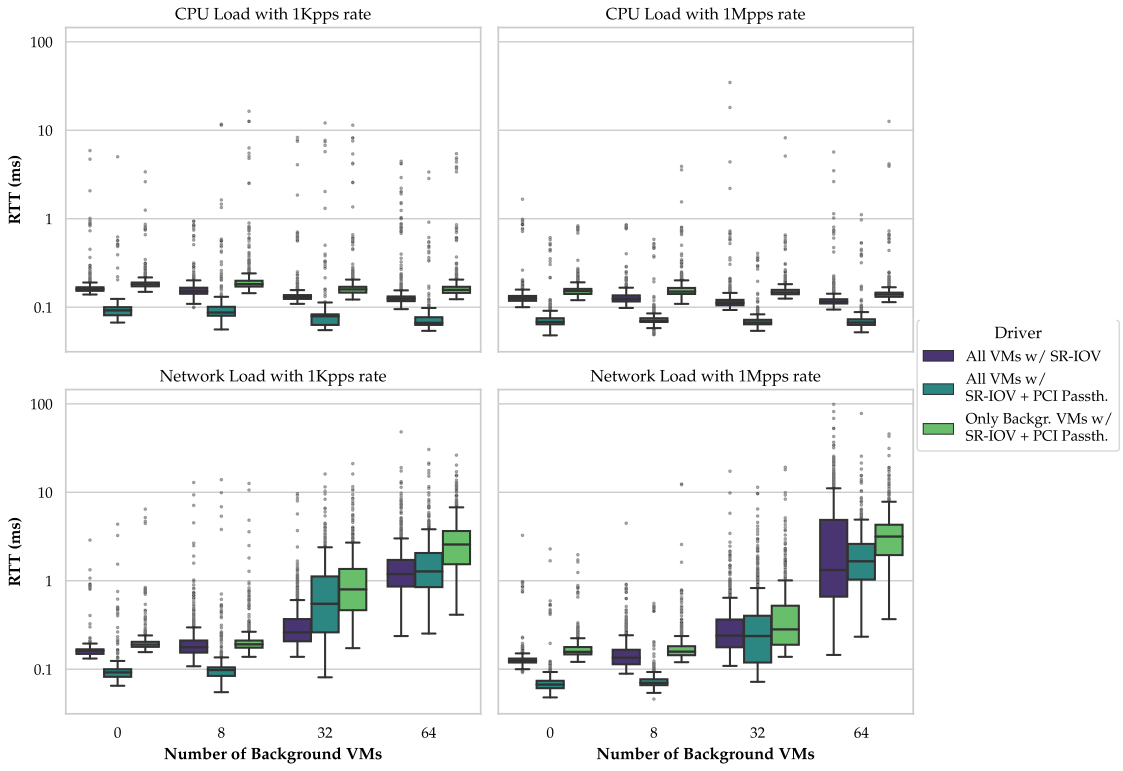

Fig. 17. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median of all experiments on the Responder Server.

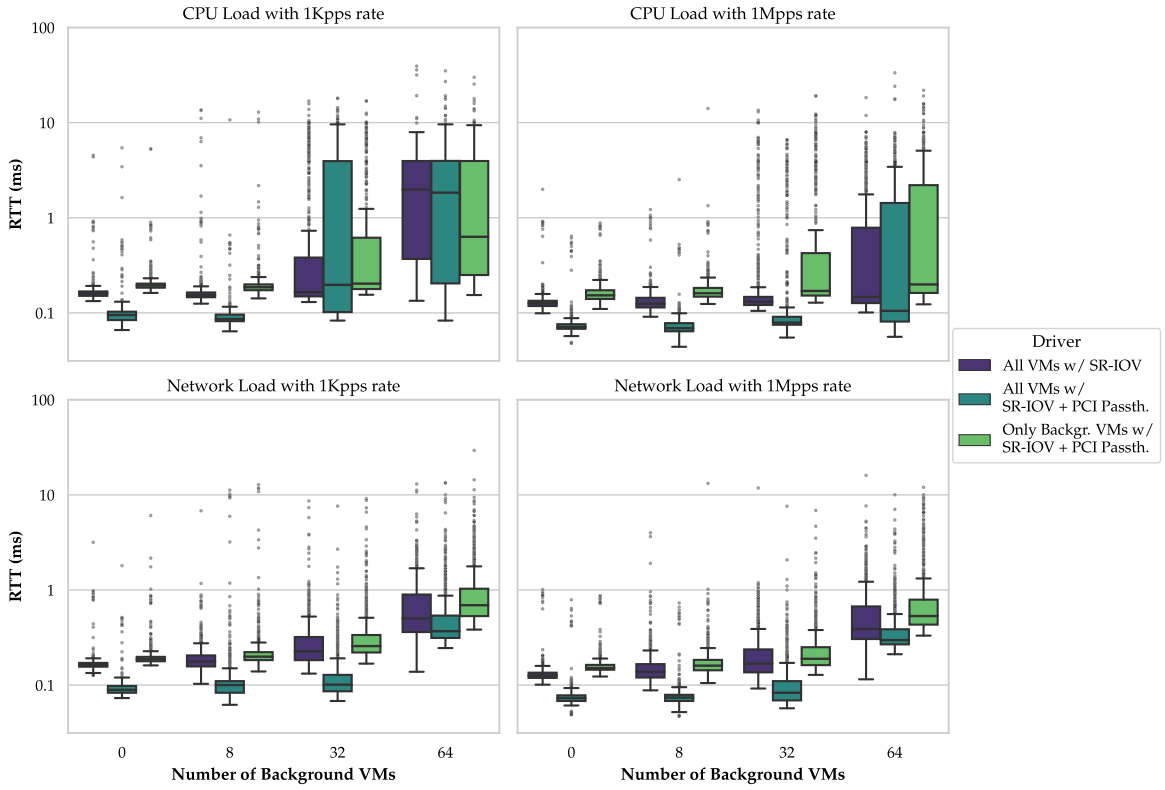

Fig. 18. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the RTT median of all experiments on the Sender Server.

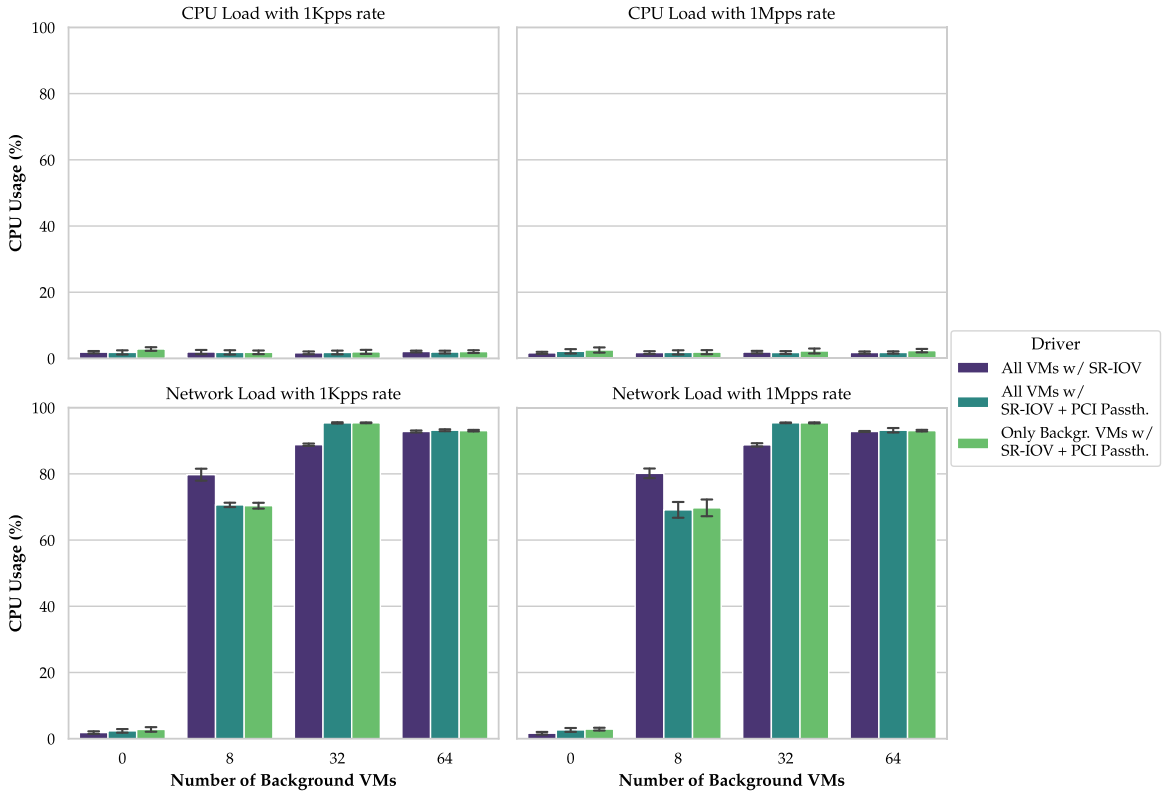

Fig. 19. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the CPU values of all experiments on the Responder Server.

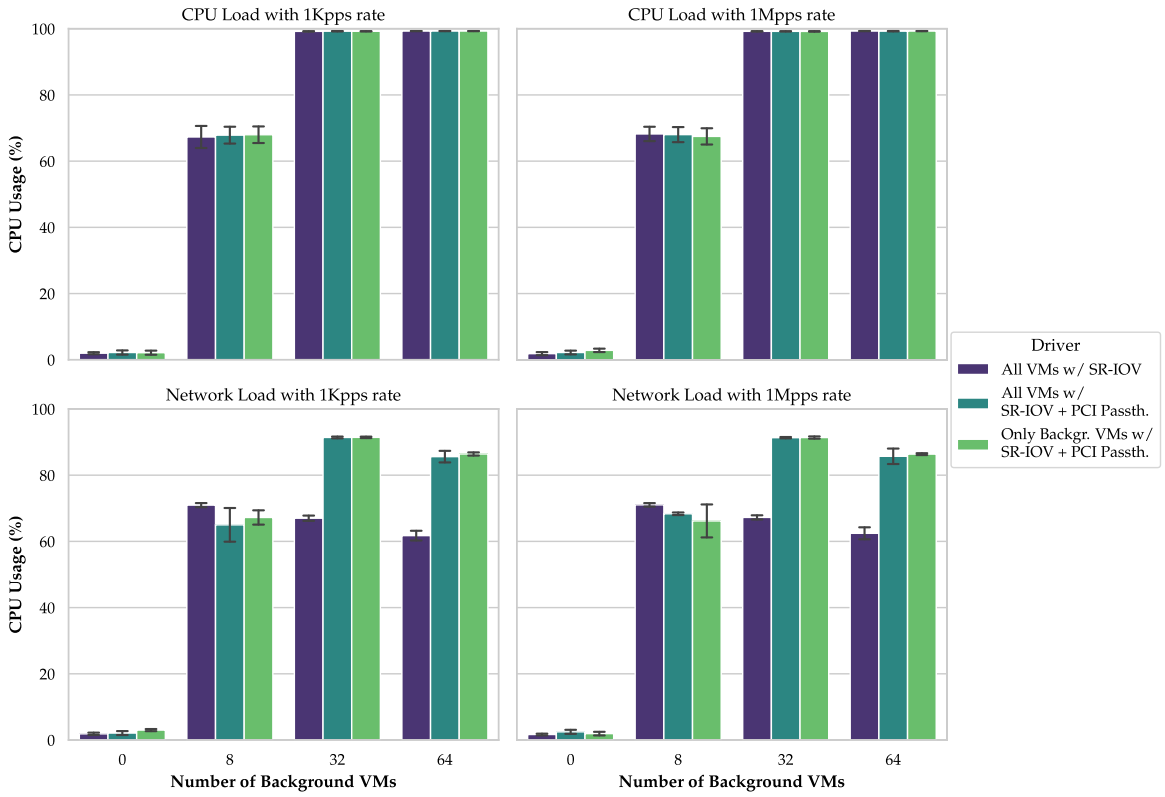

Fig. 20. Results for all experiments conducted. This graphic summarizes the performance of each virtual driver considering the CPU values of all experiments on the Sender Server.

Table 3
Technologies behavior – hypothesis testing.

Server	Sub-topic	Group	p-Value (Wilcoxon)	Comprehension
Sender	CPU	SRIOV (PCI) vs. Bridge	0,218411	Same distribution (fail to reject H0)
		SRIOV (PCI) vs. SRIOV (Macvtap)	0,142760	Same distribution (fail to reject H0)
		SRIOV (PCI) vs. SRIOV (PCI - BG)	0,0000000	Different distribution (reject H0)
		SRIOV (Macvtap) vs. Bridge	0,4608560	Same distribution (fail to reject H0)
	RTT	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)
		SRIOV (PCI) vs. SRIOV (Macvtap)	0,0000000	Different distribution (reject H0)
		SRIOV (PCI) vs. SRIOV (PCI - BG)	0,0000000	Different distribution (reject H0)
		SRIOV (Macvtap) vs. Bridge	0,0000000	Different distribution (reject H0)
Responder	CPU	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)
		SRIOV (PCI) vs. SRIOV (Macvtap)	0,0000000	Different distribution (reject H0)
		SRIOV (PCI) vs. SRIOV (PCI - BG)	0,0000000	Different distribution (reject H0)
		SRIOV (Macvtap) vs. Bridge	0,0000000	Different distribution (reject H0)
	RTT	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)
		SRIOV (PCI) vs. SRIOV (Macvtap)	0,0000000	Different distribution (reject H0)
		SRIOV (PCI) vs. SRIOV (PCI - BG)	0,0000000	Different distribution (reject H0)
		SRIOV (Macvtap) vs. Bridge	0,0000000	Different distribution (reject H0)

Is there a difference in CPU usage or RTT metrics when using a given data plane technology when compared to another one?

$$Hypotheses \begin{cases} H_0 : \text{No significant difference between any two groups} \\ H_1 : \text{Different distributions} \end{cases} \quad (1)$$

In Table 3, it is possible to observe the results obtained in the conducted statistical test. According to the values obtained, it is possible to observe that for KVM, there is no significant difference in performance between the technologies: SRIOV(PCI) vs. Bridge, SRIOV (PCI) vs. SRIOV (MacVTap), and SRIOV (MacVTap) vs. Bridge when the *sender server* is stressed for resources. Observe the observed metric being analyzed is CPU usage. The null hypothesis was rejected for all other remaining hypotheses, indicating that the distributions are indeed different.

In addition to the Wilcoxon technique, we used the Kendall rank correlation coefficient [19], a non-parametric hypothesis test regarding statistical dependence based on the tau coefficient. This choice reflects our commitment to selecting statistical techniques more aligned with our data's properties and challenges, thus ensuring more accurate and reliable analysis. Kendall correlation coefficient returns a value from −1 to 1, where 0 indicates no relationship, and −1 and 1 signal a perfect relationship (the negative sign indicates an inverse correlation). By performing these tests, one can confirm if there is a difference in performance when one uses a particular technology. In addition, the analysis identifies which configurations have a larger or smaller relationship with the increase or worsening of the metrics used to measure the performance of our scenario.

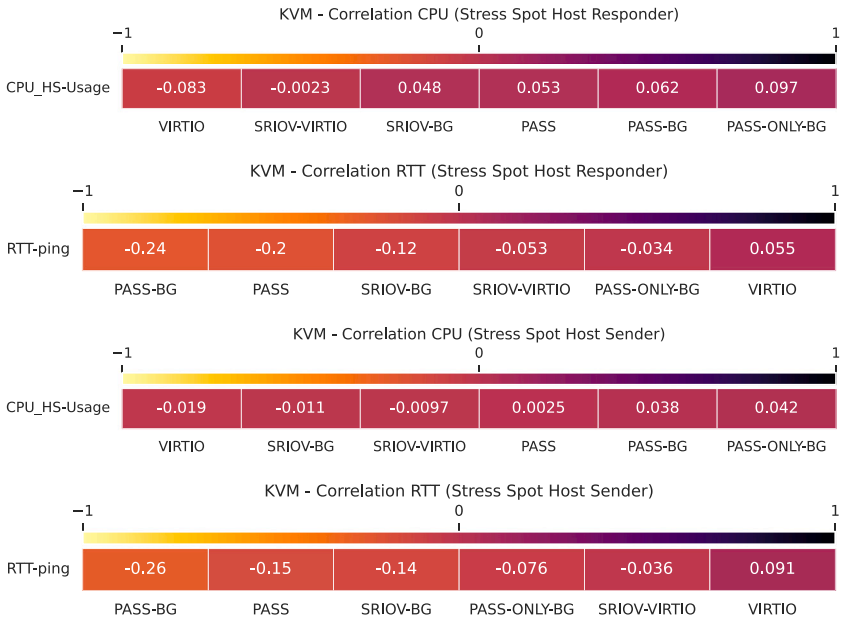

Fig. 21. Kendall rank correlation coefficient – KVM.

5.1. Full virtualization analysis - Kendall rank

To evaluate the technologies and architectures previously proposed, we divided the following scenarios: Stress Spot in Host Responder and Stress Spot in Host Sender. For both scenarios, we performed an analysis based on the metrics of CPU usage on the sender host and RTT-Ping value. The Kendall ranking results allow us to observe that in terms of CPU usage, regardless of the location of the stress spot, there was no significant difference in impact for the tested data plane configurations. Still, even so, we see that VirtIO has the most negligible impact on CPU usage while using SRIOV PCI Passthrough on background machines has more impact on CPU usage. Regarding the value of RTT-Ping, we observed a more significant difference between the technologies. The use of SRIOV PCI Passthrough in the background machines significantly influences the reduction of RTT-Ping regardless of the location of the applied background stress (see Fig. 21).

Summary: The results indicated that there is no significant performance difference between SR-IOV (PCI) vs. Bridge, SR-IOV (PCI) vs. SR-IOV (MacVTap), and SR-IOV (MacVTap) vs. Bridge technologies when resources stress the sending server. However, the null hypothesis was rejected for all remaining hypotheses, indicating that the distributions are different.

6. Related works

[20] reviews SR-IOV (Single Root I/O Virtualization) performance with a focus on latency and a conclusion highlighting the need for further research in this area. The study discusses the advantages of SR-IOV in para-virtualized solutions, such as VIRTIO, and points out that recent studies need to address latency. Furthermore, the article highlights the need for more investigation into SR-IOV scalability and latency optimizations. It is concluded that more work is needed to deepen the understanding of SR-IOV and its impact on latency. The lack of comprehensive studies on SR-IOV latency raises the need for additional research to understand its performance better. Latency is critical in many applications, such as network function virtualization, and can significantly affect system quality and efficiency. Therefore, it is essential to explore SR-IOV optimizations and scalability strategies further, taking latency into account as a critical consideration. The expansion of this knowledge will contribute to the advancement of virtualization technologies and the creation of more efficient and high-performance solutions.

The research described in [21] focuses on Network Function Virtualization (NFV) challenges. It considers Deep Packet Inspection (DPI) Virtual Network Functions (VNFs). The experiments compare the performances of a normal DPI program running in user space with the same DPI service when executing as a virtual function inside a Kernel-based Virtual Machine (KVM) environment with added SR-IOV support. The work also analyzes the performance gained when using the Data Plane Development Kit (DPDK) packet processing framework over a VM and at the user space. Recall that DPDK is a framework initially developed by Intel that provides a kernel-bypass technique to process network packets at the user-space level, improving the packet processing speed. The research focuses on bench-marking and analyzing how virtual network functions differ from bare-metal network programs in terms of performance. Nonetheless, the study falls short of assessing SR-IOV performance when used in a cloud computing environment, with multiple VMs and subject to different competing background resource loads.

The work in [22] evaluates SR-IOV in a KVM virtual environment. The authors conduct experiments to evaluate SR-IOV and compare it with other configurations, such as using a VirtIO standard driver, with no SR-IOV VFs applied, and a native configuration, where the same network application is executed in the native host. In this work, SR-IOV is implemented in combination with the PCI Passthrough. The authors evaluate network metrics like bandwidth and latency of TCP connections and system metrics like host CPU utilization, memory access, interrupts, and context switches.

As expected, the native experiment case achieved better results since it runs bare-metal and suffers no virtualization overhead. SR-IOV shows better performance than the VirtIO driver. SR-IOV also consumes more CPU cycles than the native case since it deals with virtualization interrupts. It is essential to remember that the experiments in this work used an early implementation of KVM PCI Passthrough, which may result in less optimal performance, while the VirtIO implementation was already well-tested and mature. This explains how the experiments showed that VirtIO uses fewer context switches between host and VMs and fewer guest interrupts than when using SR-IOV.

The work presented in [23] experiments with and compares SR-IOV and PCI Passthrough technologies when deployed in Docker Containers and KVM VMs. This work focuses on the High-Performance Computing

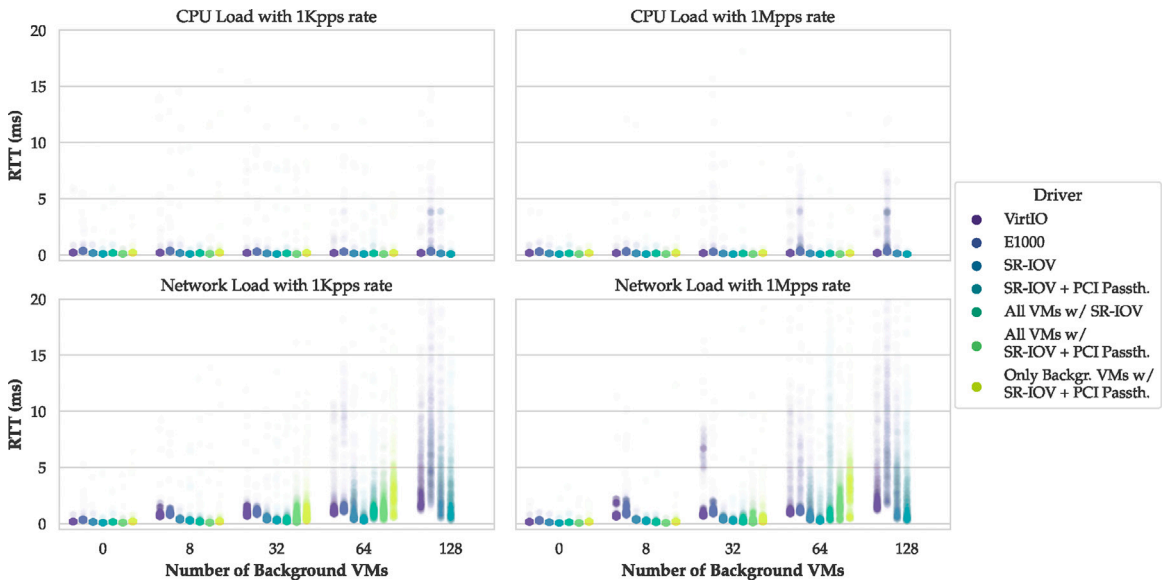

Fig. 22. Results for all experiments conducted with load on the Receiver server. This graphic summarizes the performance of each virtual driver considering a scatter plot of the RTT values.

(HPC) context and applies InfiniBand NICs to assess high network processing. The main goal of this work is to evaluate whether SR-IOV and PCI Passthrough can improve the network performance of virtual machines when applied in an HPC environment. To answer this research question, the authors built a testbed with HPC-specific applications deployed in Docker containers and KVM VMs and in bare metal to serve as a baseline for the results. The network metrics obtained were mainly latency and bandwidth. The results show that SR-IOV has the worst performance in both latency and bandwidth since this technology still goes through hypervisor network management. Container and VM with PCI Passthrough configuration achieve more optimal performance, with results close to bare-metal. In some cases, containers with PCI Passthrough show a performance degradation for latency when processing big-sized messages like 256 bytes. However, overall, containers perform better than VMs, displaying a mere 9% overhead on HPC applications. The work [9] also performs SR-IOV evaluation in the HPC context using InfiniBand cards. Its authors conduct experiments using an AWS EC2 cloud environment to compare network performance evaluation with and without SR-IOV. The results show that SR-IOV when running over the InfiniBand networking technology yields better performance, lowering the latency overhead and increasing the bandwidth to a level close to line-rate performance. In an EC2 cluster, SR-IOV can reduce latency compared to not using it. Nonetheless, it still results in twice the latency for the network functions compared to not using any form of virtualization.

The article [24] analyzes the performance of different I/O technologies that provide network access to virtual network functions (VNFs). The work focuses on assessing the maximum possible physical throughput as a growing number of VNFs (up to 10) are introduced. Each VNF uses OVS-DPDK, FD.io VPP, or SR-IOV (with PCI Passthrough) as the packet I/O technology. The VNFs are created using a KVM hypervisor, and they perform either a simple IPv4 forwarding or other more demanding services such as NAT, Firewall, QoS, and DPI. The results show that the combination of SR-IOV and PCI Passthrough can achieve close-to-physical-limit throughput when using more than 3 VNFs while performing simple IP forwarding. OVS-DPDK and VPP show similar performance, stabilizing their achievable throughput at around half the physical limit of the network card. With the introduction of more demanding VNFs, SR-IOV remains optimal, while VPP outperforms OVS-DPDK.

In the research described in [25], the experiments were carried out with the combination of SR-IOV and PCI Passthrough packet processing technologies while also using KVM and LXC. The object is to evaluate the usage of hardware assistance on VMs and Containers, in contrast to standard virtual drivers like MacVTap (for KVM) and Macvlan (for LXC). Although the authors do not mention PCI Passthrough, it is intuitive to understand that Passthrough is applied since the given explanation in the paper exposes the SR-IOV NIC directly to the user space VM, and the obtained results show that only SR-IOV VFs would not enable such high performance. The authors created an environment with up to 8 VMs/Containers and a 1GbE NIC, where they measure the achievable throughput, latency, and CPU usage when processing a variable incoming packet rate. The results show that LXC and KVM have comparable throughput performance when using SR-IOV. The baseline of using MacVTap and Macvlan shows how suboptimal the performance of KVM can be, significantly when increasing the incoming network load. The CPU usage also benefits SR-IOV. KVM with MacVTap has 100% CPU usage when increasing the incoming load from 200 Kpps and beyond, especially because of software interrupt request (SoftIRQs) handling inside the kernel. When using SR-IOV, the KVM VM-based configuration reaches a 100% packet processing level (no loss) only when processing an incoming rate of 1400 kpps, which is close to the NIC line rate and represents the high processing capacity of the VM.

Concerning packet processing optimization for servers, existing works for datacenter design failed to provide in-depth evaluations of SR-IOV. There is a need to look at how SR-IOV may be deployed jointly with other data plane technologies such as PCI-Passthrough and MacVTap to increase performance further. Thus, to our knowledge, this is the only study that develops an assessment considering other factors such as traffic behavior, background overhead including VMs, and competing network traffic not handled by SR-IOV. We create a testbed that represents a virtual environment typical to the context of cloud computing. We vary several factors that will allow the study to delve deeper into the obtained results and interpret them. In addition, our study also assesses the impact of using different forms of SR-IOV, including MacVTap and PCI-Passthrough. Last but not least, we apply known statistical approaches (Kendall rank and Wilcoxon Signed Rank Test) to validate the observations made during the empirical study.

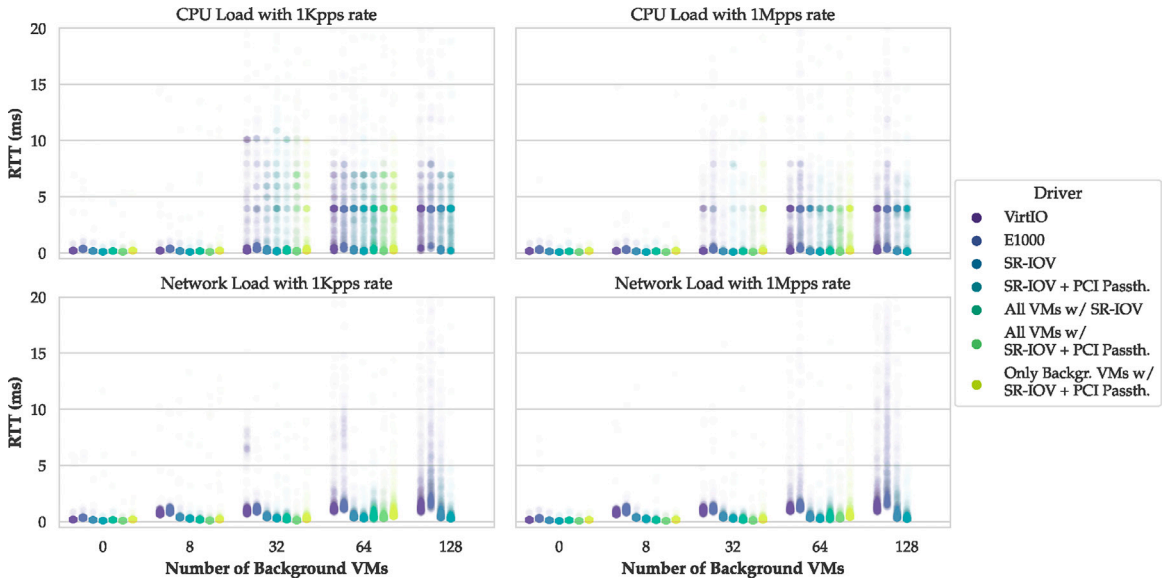

Fig. 23. Results for all experiments conducted with load on the Sender server. This graphic summarizes the performance of each virtual driver considering a scatter plot of the RTT values.

7. Conclusion and future works

Virtualization is one of the technological foundations for the cloud computing paradigm. It enables cloud service providers to deliver infrastructure and services through the logical software-based slicing of computing, storage, and network resources on physical servers. However, it is known that virtualization can adversely affect overall performance due to the additional layers of software required for its operation. This is true about network latency. As a result, several proposals emerged to solve this performance challenge. Among these is the SR-IOV data plane optimization technology.

In this work, we conducted comprehensive studies to investigate the influence of the SR-IOV technology on latency in a KVM virtualization environment. Specifically, we examine the impact of SR-IOV on the critical Round Trip Time (RTT) networking metric and gain valuable insights into its behavior. This article analyzes SR-IOV through a series of experiments that accurately represent the technology's use case in the KVM virtualization platform. We explore various factors, such as SR-IOV utilization methods, network virtual drivers, load location, the number of background VMs, and variations in the number of VMs using SR-IOV. Our aim is to shed light on behaviors that are not yet fully understood. To evaluate our experiments, we employ statistical evaluation methods, including the Wilcoxon Signed Rank Test and Kendall rank. The summarized results of our experiments are presented in Figs. 22 and 23.

The results indicate that the SR-IOV, regardless of how its use, has a significant positive influence on the reduction of RTT. Among the results we present, we can also highlight the following:

- The combination of SR-IOV + PCI Passthrough in the data plane achieves the lowest RTTs, even when used with background traffic.
- Using the Bridge Vnet driver and SR-IOV + PCI Passthrough consumes more CPU, especially in background CPU competition. The SR-IOV MacVTap setup in 'passthrough' mode is the best suitable configuration when CPU resources are scarce or low.
- SR-IOV and MacVTap in 'passthrough' mode is preferred when scalability and flexibility are design requirements. This data plane configuration outperforms a traditional "Bridge" Vnet.
- The use of SR-IOV with either MacVTap or PCI Passthrough, by no means, can reduce the RTT variation in scenarios with high CPU usage.
- It is inefficient from the point of view of communication performance to configure a VM that does not use SR-IOV in a Datacenter in the presence of other VMs that benefit from SR-IOV.

Our study highlights the importance of considering SR-IOV in virtualization environments to improve communication performance and provides useful guidance for selecting the most appropriate configuration. In conclusion, our study provides valuable insights into the impact of SR-IOV on network latency in a KVM virtualization environment.

Finally, as part of future work, we also aim to evaluate the performance of SR-IOV using alternative methods, such as the IEEE 1588 Precision Time Protocol (PTP), in place of traditional ping-based measurements. By incorporating PTP, we can assess the accuracy and precision of time synchronization between virtual machines (VMs) and the underlying physical infrastructure. This evaluation will provide valuable insights into the effectiveness of SR-IOV in achieving precise time synchronization, which is crucial for applications requiring strict timing requirements, such as real-time communication and synchronization in distributed systems. Additionally, we will explore the impact of SR-IOV on PTP performance metrics, including clock accuracy, offset, and network delay variation. By incorporating PTP evaluation alongside other metrics, such as the ones above, we can better understand SR-IOV's performance and its suitability for time-sensitive applications in various virtualization environments. We also plan to explore the evaluation of SR-IOV on other virtualization platforms, including Docker and LXC platforms. In addition to this, we aim to introduce new metrics to enhance our analysis further. These metrics may include application-level response time, bandwidth, packet loss, interrupt latency for the kernel driver, packet size variations, memory bandwidth stress, cache stress, and other relevant factors. By incorporating these additional metrics into our evaluations, we can gain a more comprehensive understanding of the performance and capabilities of SR-IOV in various virtualization environments. These future endeavors will contribute to expanding our knowledge and insights into the potential benefits and limitations of SR-IOV technology.

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, Visualization. **Pedro R.X. do Carmo:** Software, Validation, Formal analysis, Data curation, Writing – original draft, Visualization. **Djamel F.H. 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.

Data availability

Data will be made available on request

Acknowledgments

This work was supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), FACEPE (Fundação de Amparo a Ciência e Tecnologia de PE) and CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

References

[1] E. Freitas, A.T. de Oliveira Filho, P.R. do Carmo, D. Sadok, J. Kelner, A survey on accelerating technologies for fast network packet processing in Linux environments, Comput. Commun. 196 (2022) 148–166, http://dx.doi.org/10.1016/j.comcom.2022.10.003.

[2] J.P. Walters, V. Chaudhary, M. Cha, S. Guercio, S. Gallo, A comparison of virtualization technologies for HPC, in: 22nd International Conference on Advanced Information Networking and Applications (AINA 2008), 2008, pp. 861–868, http://dx.doi.org/10.1109/AINA.2008.45.

[3] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, L. Peterson, Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors, SIGOPS Oper. Syst. Rev. 41 (3) (2007) 275–287, URL https://doi.org/10.1145/1272998.1273025.

[4] M.G. Xavier, M.V. Neves, F.D. Rossi, T.C. Ferreto, T. Lange, C.A.F. De Rose, Performance evaluation of container-based virtualization for high performance computing environments, in: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013, pp. 233–240, http://dx.doi.org/10.1109/PDP.2013.41.

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

[6] A. Richter, C. Herber, T. Wild, A. Herkersdorf, Denial-of-Service attacks on PCI passthrough devices: Demonstrating the impact on network- and storage-I/O performance, J. Syst. Archit. 61 (10) (2015) 592–599, http://dx.doi.org/10.1016/j.sysarc.2015.07.003.

[7] D. Marion, VPP architecture, 2016, URL https://www.youtube.com/watch?v=d6Fn5cuLcVk.

[8] A.T. de Oliveira Filho, E. Freitas, P.R. do Carmo, D.H. Sadok, J. Kelner, An experimental investigation of Round-Trip Time and virtualization, Comput. Commun. 184 (2022) 73–85, http://dx.doi.org/10.1016/j.comcom.2021.12.006, URL https://www.sciencedirect.com/science/article/pii/S0140366421004722.

[9] G.K. Lockwood, M. Tatineni, R. Wagner, SR-IOV: Performance benefits for virtualized interconnects, in: Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment, XSEDE '14, Association for Computing Machinery, New York, NY, USA, 2014, http://dx.doi.org/10.1145/2616498.2616537.

[10] A. Viviano, SR-IOV virtual functions (VFs), 2012, URL https://docs.microsoft.com/en-us/windows-hardware/drivers/network/sr-iov-virtual-functions--vfs-.

[11] V. Shankarkumar, L. Montini, T. Frost, G. Dowd, Precision Time Protocol Version 2 (PTPv2) Management Information Base, RFC 8173, 2017, http://dx.doi.org/10.17487/RFC8173, URL https://www.rfc-editor.org/info/rfc8173.

[12] I. Corporation, Using the MacVTap driver, 2021, URL https://www.ibm.com/docs/en/linux-on-systems?topic=choices-using-macvtap-driver.

[13] C.B. Robison, Configure SR-IOV network virtual functions in linux* KVM*, 2017, URL https://www.intel.com/content/www/us/en/developer/articles/technical/configure-sr-iov-network-virtual-functions-in-linux-kvm.html.

[14] 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, http://dx.doi.org/10.1109/INM.2015.7140382.

[15] A.T.d. Oliveira Filho, Uma Análise Experimental De Desempenho Do Protocolo QUIC, Mestrado em ciência da computação, UFPE, Recife, 2020, URL https://repositorio.ufpe.br/handle/123456789/37646.

[16] 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.

[17] 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.

[18] R.F. Woolson, Wilcoxon signed-rank test, in: Wiley Encyclopedia of Clinical Trials, Wiley Online Library, 2007, pp. 1–3.

[19] H. Abdi, The Kendall rank correlation coefficient, Encycl. Meas. Stat. 2 (2007) 508–510.

[20] M. Fischer, F. Wiedner, Survey on SR-IOV performance, Network 43 (2021).

[21] M.-A. Kourtis, G. Xilouris, V. Riccobene, M.J. McGrath, G. Petralia, H. Koumaras, G. Gardikis, F. Liberal, Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration, in: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015, pp. 74–78, http://dx.doi.org/10.1109/NFV-SDN.2015.7387409.

[22] J. Liu, Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support, in: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2010, pp. 1–12, http://dx.doi.org/10.1109/IPDPS.2010.5470365.

[23] 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.

[24] N. Pitaev, M. Falkner, A. Leivadeas, I. Lambadaris, Characterizing the performance of concurrent virtualized network functions with OVS-DPDK, fd.io VPP and SR-IOV, in: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 285–292, http://dx.doi.org/10.1145/3184407.3184437.

[25] P.S. Muhammad Siraj Rathore, KVM vs. LXC: Comparing performance and isolation of hardware-assisted virtual routers, Am. J. Netw. Commun. 2 (2013) 88–96, http://dx.doi.org/10.11648/j.ajnc.20130204.11.