Computer Communications 228 (2024) 107961

Contents lists available at ScienceDirect

Computer Communications

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

Analysis of SR-IOV in Docker containers using RTT measurements

Assis T. de Oliveira Filho [a,b,c,*], Eduardo Freitas [a,b], Pedro R.X. do Carmo [a,b], Eduardo Souto [d], Judith Kelner [a,b], Djamel F.H. Sadok [a,b]

[a] *Networking and Telecommunications Research Group (GPRT), Universidade Federal de Pernambuco (UFPE), Recife, 50730-120, Pernambuco, Brazil*
[b] *Centro de Informática (CIn), Universidade Federal de Pernambuco, Recife, 50740-560, Pernambuco, Brazil*
[c] *ICAM-Tech, Universidade Católica de Pernambuco (UNICAP), Recife, 50500-900, Pernambuco, Brazil*
[d] *ICE, Universidade Federal do Amazonas (UFAM), Manaus, 69077-000, Amazonas, Brazil*

ARTICLE INFO

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

ABSTRACT

Cloud computing, a central pillar of modern IT infrastructure, faces constant challenges in provisioning and optimizing network performance, specifically regarding low-latency communication. This study investigates the impact of Single Root I/O Virtualization (SR-IOV) as a critical Quality of Service (QoS) enabler in virtualized environments. Data plane innovative technologies for virtual servers, especially SR-IOV technology, emerged as a promising solution adopted in data centers. When combined with Peripheral Component Interconnect (PCI) Passthrough in Docker environments, SR-IOV promises significant network performance gains. Our rigorous experimental methodology demonstrates that integrating SR-IOV reduces Round-Trip Time (RTT) latency by up to 15 times compared to the traditional Linux based Bridge configuration used in Docker, without significant additional costs. This research is particularly relevant for system administrators, data center professionals, and network traffic engineers, providing them valuable information into optimizing communication in cloud computing environments. By addressing this critical gap in knowledge, our study serves as a practical guide for the effective implementation these emerging technologies for network virtualization. In terms of practical applicability, the results raise valuable insights into the performance and implications of implementing SR-IOV and PCI Passthrough in a Docker environment. As a result, more informed decisions are tailored to the specific requirements of different usage scenarios.

1. Introduction

The landscape of computing resources experiences profound transformations sustained by virtualization technology, particularly within data center and cloud environments. One prominent approach for host and server virtualization that has gained widespread recognition is containerization due to its relatively lightweight and efficient nature, unlike traditional virtualization architectures. While it is deeply integrated with the operating system, the container approach encapsulates applications and their dependencies within self-contained virtual containers. These Linux containers exhibit seamless adaptability across various environments, from on-premises configurations to public and private clouds, cementing their role as indispensable assets in contemporary computing.

In the realm of Linux containerization, two notable contenders have emerged: Linux Containers (LXC)[1] and Docker.[2] Recent studies and industry trends have demonstrated Docker's significance, making it the *de-facto* standard in the field, as highlighted by [1–3]. Docker's meteoric rise in popularity can be attributed to its remarkable adaptability, efficient resource utilization, and standout performance compared to its counterparts [4].

The growing use of Docker presents new challenges, such as the need to improve packet processing speed between virtualized resources and the physical infrastructure to minimize packet latency. Recent research initiatives have witnessed numerous proposals aimed at the optimization of packet processing latency. The engineering and provisioning of cloud and data center network architectures have evolved from fat tree to spine-leaf topology. This trend is motivated by increasing network usage efficiency and reduced latency [5].

Network engineers dedicate a considerable effort towards optimizing throughput, delay, jitter, and packet loss, seen as the main design

* Corresponding author.
E-mail addresses: assis.tiago@gprt.ufpe.br (Assis T. de Oliveira Filho), eduardo.freitas@gprt.ufpe.br (E. Freitas), pedro.carmo@gprt.ufpe.br (Pedro R.X. do Carmo), esouto@icomp.ufam.edu.br (E. Souto), jk@gprt.ufpe.br (J. Kelner), jamel@gprt.ufpe.br (D.F.H. Sadok).

[1] https://linuxcontainers.org/.
[2] https://www.docker.com/.

https://doi.org/10.1016/j.comcom.2024.107961
Received 1 April 2024; Received in revised form 13 September 2024; Accepted 23 September 2024
Available online 2 October 2024

metrics. Current advancements in Network Interface Cards (NICs) design have led to the deployment of 400 Gbps port NICs in data centers. However, despite the availability of high-speed interfaces, latency remains a hurdle and requires special consideration. Additional work is required to optimize the Linux data plane [6]. For instance, a Vanilla Linux distribution with a 10 Gbps network port has been shown to only manage a mere 1 Gbps when sending small 64-byte Ethernet packets [7]. Thus, there is a need to discuss some of the solutions that have been put forward to deal with this issue. This paper delves into an essential and widely used Linux data plane setup, namely, the SR-IOV technology in conjunction with the PCI Passthrough mechanism.

SR-IOV technology has garnered attention as a critical player in network interface virtualization because it reduces hypervisor overhead due to packet processing and enhances network performance. It is a smart hardware-based solution that allows a single physical Input/Output (I/O) device to appear as multiple separate virtual devices with equivalent capabilities on the Peripheral Component Interconnect Express (PCIe) bus [8]. On the other hand, PCI Passthrough is a mechanism that enables a physical device, such as a NIC, to present its PCI bus directly to user applications in user space. In other words, this solution cuts the middleman as it bypasses the kernel. As a result, applications gain access to the network card directly, enabling Virtual Machines (VMs) to access multiple Virtual Network Interface Cards (vNICs) without traversing the standard kernel packet processing path [9]. This direct access reduces data copy operations, packet processing overhead, and end-to-end latency [10].

Integrating SR-IOV and PCI Passthrough with Docker containers presents an opportunity to significantly improve network performance. By multiplexing physical NICs into virtual ones and providing direct access to network resources, Docker applications can achieve reduced latency and higher packet transfer rates. However, more comprehensive research is needed to explore the benefits and limitations of such integration, highlighting the need for a detailed investigation.

This study aims to fill this gap by analyzing SR-IOV's role in enhancing network capabilities and performance within Docker applications. Specifically, it explores the impact of Docker virtualization and SR-IOV on end-to-end latency, establishing a methodologically sound evaluation environment for a cloud computing setup. Additionally, it examines the challenges and considerations involved in implementing SR-IOV in virtualized environments and provides practical guidance for optimizing latency in a Docker-based network.

Despite the apparent and potential advantages, such as improved network performance, reduced latency, and efficient resource utilization, of integrating SR-IOV with Docker, more comprehensive and formal research still needs to be conducted to explore both the benefits and limitations of the combined use of these mechanisms. This gap in the literature signals a critical need for an in-depth investigation into how SR-IOV technology can be harnessed to augment network performance when using Docker containers. The present research addresses this void by thoroughly analyzing SR-IOV's role and efficacy in enhancing the network capabilities of Docker applications. This study aims to contribute to the broader understanding of Docker virtualization's potential in leveraging advanced network technologies to optimize packet processing at the data plane.

In summary, this paper explores the impact of Docker virtualization and SR-IOV on end-to-end latency represented by the RTT performance metric. It establishes methodologically a sound evaluation environment for cloud computing. The work assesses the benefits and limitations of Docker virtualization, SR-IOV, and PCI Passthrough technologies and examines their potential influence on latency under various configurations. Additionally, we delve into the challenges and considerations involved in implementing SR-IOV in a virtualized environment and provide insights for optimizing end-to-end latency in such scenarios. This work provides a comprehensive overview of the Docker and SR-IOV interaction and its impact on RTT, along with practical guidance for optimizing RTT in a Docker-based network environment.

The remainder of the paper is structured as follows: Section 2 provides the theoretical background necessary for the discussions presented in this paper. Section 3 reviews related work. Section 4 outlines the adopted experimental setup and testbed design. Section 5 presents some of the latency performance results obtained. Finally, Section 6 summarizes our conclusions and offers final insights.

2. Containerization and SR-IOV

SR-IOV is a key data plane technology developed to allow I/O devices, such as NICs, to be used more efficiently. Proposed by the PCI-SIG (PCI Special Interest Group), SR-IOV achieves its by goal virtually slicing the actual physical device Physical PCIe Function (PF) into multiple Virtual PCIe Functions (VFs). Each VF can be directly assigned to applications independently of the Information Technology (IT) infrastructure resources and virtualization architecture, offering them isolated and faster access to the hardware [11]. Next, we introduce the key elements of the SR-IOV architecture, specifically the PFs and VFs.

- **PF**: Represents the complete PCI hardware devices. It includes all device capabilities and manages its underlying VFs.
- **VFs**: Offer efficient subdivisions of a given PF and are designed to optimize I/O operation. Each VF can be assigned to a VM or container, providing isolation for its processes. This allocation is critical to mitigating multi-tenancy issues common in cloud and data center environments, where resource contention can significantly impact application performance.

SR-IOV reduces contention for software and hardware resources by creating VFs, leading to reduced CPU overhead associated with packet processing. To achieve this, SR-IOV multiplexes packet processing among the VFs and isolates containers by placing packets in their respective different queues. This is particularly advantageous for latency-sensitive applications. Additionally, the direct path of SR-IOV provides increased security by isolating traffic at the hardware level, thus offering a robust solution for multi-tenant environments.

SR-IOV is a hardware-based technology that offers multiple methods to integrate into container user-space processes. In the context of Docker virtualization, two main integration methods stand out, each with its peculiarities, advantages, limitations, and specific configurations. We detail these approaches in the following subsections while also discussing their adaptation to align with Docker-based virtualization practices and paradigms.

2.1. *Integration of SRIOV within docker environments*

Considering the Docker platform, the integration of SR-IOV to improve the network efficiency of containers can be achieved mainly by three methods: **Direct access**, integration via **Network Plugin** and **PCI Passthrough** [12]. These methods provide different approaches to how SR-IOV VFs and PFs can be allocated or used by containers to ensure optimized network performance, isolation, and security.

Knowing container namespaces is crucial to understanding and fully implementing these methods. Container namespaces provide the necessary isolation and control over network resources, which is fundamental for the optimal functioning of direct access and network plugin integration methods.

A namespace in Linux encapsulates a global system resource in an abstraction, making it appear that processes within the namespace have their isolated instance of the resource. For containers, namespaces ensure that processes within a container do not interfere with those outside it, providing isolation for network interfaces, IP addresses, routing tables, and more [13,14].

Network namespaces allow each container to have its own network stack, which includes IP addresses, routing tables, and network devices. This is crucial for effectively managing network resources and ensuring secure and efficient communication between containers and the external network [14,15].

2.1.1. Network plugin

Integration via network plugin uses third-party software or plugins explicitly developed for Docker, which automates configuring and assigning VFs to containers. This software or plugins manage the complexity of SR-IOV configuration and VF assignment, providing a more straightforward, more accessible, and more automatic interface for the user.

Examples of plugins that allow the abstraction of the complexity of SR-IOV configuration for Docker are Intel® SR-IOV Container Network Interface (CNI) Plugin [16], Multus CNI [17], SR-IOV Network Device Plugin for Kubernetes [18] and Kuryr-Kubernetes [19]. These commonly deployed examples include and offer simplicity and automation in using SR-IOV with Docker. However, it is essential to highlight that several additional alternative plugins are also available to choose from.

Despite the benefits listed above, using network plugins has challenges. A significant issue associated with this SR-IOV management strategy lies in the reliance on third-party abstractions, often written for specific network cards and the possible loss of granular control over the configuration and allocation of hardware resources [20]. This specificity can cause compatibility issues between plugins and the SR-IOV hardware [21]. Furthermore, a second challenge is the reduced flexibility, as administrators may lose the ability to perform specific configurations or fine-tuning due to automated approaches that enforce more generic and general configurations [22]. For instance, certain plugins may not support advanced features available on specific network cards, such as custom queue management or proprietary traffic shaping algorithms, forcing administrators to adopt less efficient alternatives. This occurs because many network plugins are designed to accommodate a wide range of hardware setups, thus prioritizing broad compatibility over specific optimizations [23]. In other words, the main relevant challenges encountered include an undesirable heavy dependence on other inherent third-party solutions and the potential loss of control over specific configurations and optimization of network resources [24].

To summarize, the integration via Network Plugin significantly simplifies Docker and SR-IOV operational procedures. This simplification allows developers to focus on developing and delivering solutions rather than worrying about and spending effort needed to understand and deal with the technical details of network configuration. However, it is essential to note that there may be drawbacks that cannot be overlooked and require special attention to ensure alignment with specific hardware and application requirements. In other words, network plugin integration for SR-IOV in Docker offers a valuable path to optimizing virtualized network infrastructure however administrators need to carefully balance ease of use and the need for customization and direct control. The Fig. 1 illustrates the process.

2.1.2. Direct access

Direct integration of SR-IOV into Docker environments represents a method by which VFs are manually assigned to specific containers, requiring careful administrative configuration. This process begins with preliminary host configuration, where the administrator enables SR-IOV through Basic Input/Output System (BIOS)/Unified Extensible Firmware Interface (UEFI) settings and proceeds to create VFs on a NIC that supports SR-IOV. Subsequently, these VFs are assigned to the desired containers using the Docker Command-Line Interface (CLI) or Dockerfiles resources, where the VF is specified as a device resource, thus allowing the container direct access to a "slice" of the network hardware.

This configuration involves setting up a standard Linux bridge to provide network packet switching between container namespaces. By default, this setup can be achieved with the docker network command, which oversees both the bridge creation and linking container namespaces to this newly established bridge (look at Fig. 1).

This integration approach offers significant advantages, notably the near-native network performance provided by containers [1]. This is possible due to the direct access to hardware offered by VFs, thus eliminating intermediate virtualization layers that would introduce additional and unwanted latency. Furthermore, unlike the Network Plugin method, direct integration allows for granular control over the allocation of network resources, enabling more efficient and specific management of available resources, which is particularly advantageous in environments where network performance tuning is required. Thus, direct integration, although more complex and laborious to operate, offers complete control over the configuration process and allocation of SR-IOV resources, allowing adjustments that maximize network performance and resource efficiency. Network engineers with advanced technical knowledge may prefer this approach to optimize every aspect of the interaction between containers, VF, and the underlying network hardware.

Despite the advantages, direct integration has associated limitations. Manual configuration on the host and the assigning of VFs to containers is prone to errors. It can be complex, especially in large-scale deployments like data centers and cloud computing environments. The need for individual management of VFs poses significant scalability challenges, making the approach less viable for environments with many containers. These limitations suggest careful evaluation before adopting direct integration, considering performance benefits versus operational requirements and management complexity.

2.1.3. PCI passthrough and SR-IOV in Docker

Some proposals for optimizing the Linux data plane view the kernel and hypervisor software as complicated intermediaries between VMs or containers and the NIC. The copy of packets between these spaces as well as the management of process interrupts among them is known to slow down the data plane. Among the technologies that bypass the host hypervisor is PCI Passthrough. PCI Passthrough is a virtualization technique that allows physical PCI devices to be directly accessible by VMs or containers, removing the hypervisor or host operating system from the network processing stack [11]. In the context of Docker containers, PCI Passthrough is used to assign VFs from an SR-IOV compatible NIC to a specific container. This gives the container exclusive, low-level access to the network device, resembling the performance achieved in a non-virtualized environment. It presents a promising strategy designed to enhance the performance of Docker environments beyond the straightforward integration of SR-IOV. By allowing physical PCI devices to be dedicated directly to specific containers, this technique promises a significant reduction in latency and increased network bandwidth. With PCI Passthrough, containers can have almost native access to network hardware by bypassing traditional virtualization layers. This approach complements the direct integration of SR-IOV by offering an efficient and low-overhead communication path with the hardware and aligning with it to optimize available resources, thus ensuring superior performance for critical applications running on Docker.

While direct integration of PCI Passthrough offers an experience closer to direct and exclusive access to specific hardware resources within containers, integration via network plugins abstracts this complexity, focusing on ease of use and automation. This comparison highlights the trade-offs between performance optimization and operational simplicity. Fig. 1 illustrates how the techniques presented and discussed in this section works.

2.2. Challenges and considerations

The deployment of SR-IOV in a Docker environment is expected to achieve a fundamental improvement in the performance of containerized networks, similar to its benefits achieved under Kernel-based Virtual Machine (KVM) virtualization, as detailed in a recent study [1].

Despite its advantages, deploying SR-IOV in Docker environments requires careful consideration of hardware compatibility, configuration complexity, and potential scalability constraints. The static nature of VF

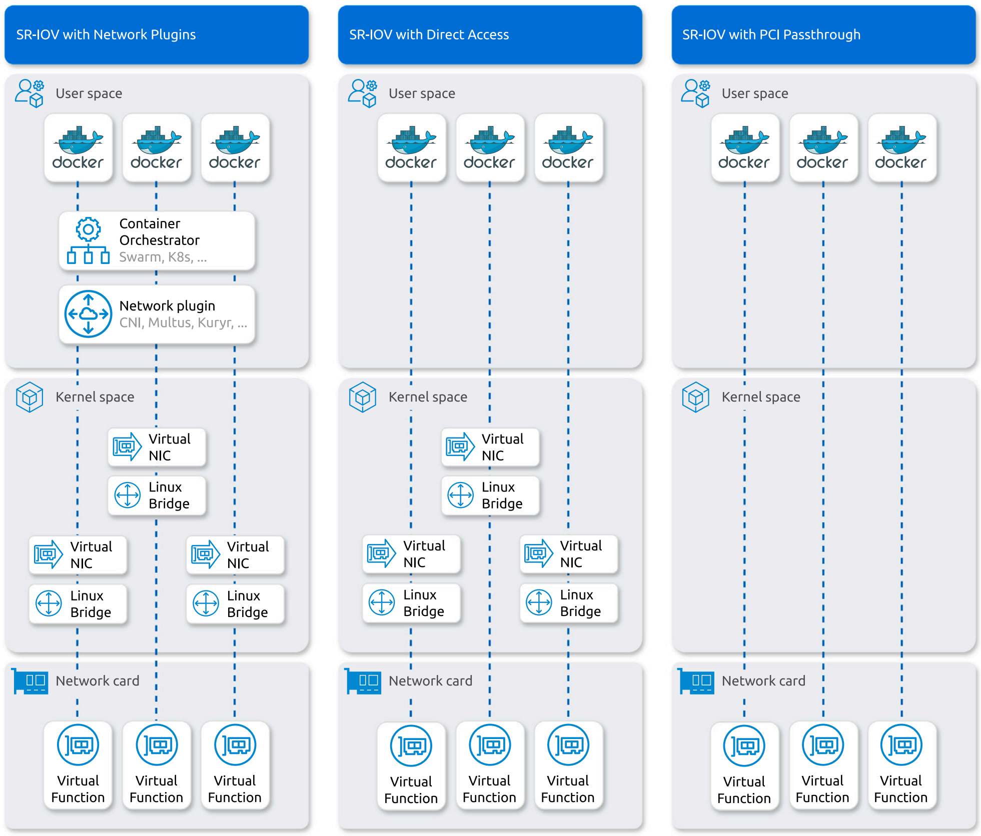

Fig. 1. Illustration of SR-IOV integration for container environment.

allocation can pose challenges in dynamic scaling scenarios, requiring advanced orchestration tools to manage resource allocation effectively.

Integrating SR-IOV technology into Docker environments presents a viable path to optimizing network performance in virtualized infrastructures. By leveraging the direct access capabilities of PCI Passthrough, SR-IOV facilitates improved latency, throughput, and CPU efficiency. However, a successful deployment requires addressing the inherent challenges associated with hardware dependency and configuration management.

3. Related work

The continuous evolution of virtualization and networking technologies plays a crucial role in optimizing the performance of network services, especially in contexts and services where end-to-end latency is a critical factor. Integrating technologies such as Docker and SR-IOV has improved efficiency significantly and QoS in virtualized environments. This section explores recent research and developments in network virtualization, focusing on metrics such as latency represented by the RTT. Analysis of these works provides a deeper understanding of current trends, challenges, and innovative solutions shaping the future of network virtualization, data plane technology, and containerized communications performance.

In our previous work, titled “Measuring the Impact of SR-IOV and Virtualization on Packet Round-Trip Time” [25], we focused on studying the effects of using Kernel-based KVM virtual machines combined with SR-IOV on network performance. Latency in the form measured by the RTT metric was considered the main evaluation metric. The study sought to present the relevance of SR-IOV in improving network efficiency in virtualized and cloud networks. Using a test platform that simulated common network scenarios, we compared the performance of technologies for connecting virtual machines to the network interface, such as Bridge, SR-IOV, and SR-IOV with PCI Passthrough. Among the results obtained, we showed that SR-IOV with PCI Passthrough outperformed standalone SR-IOV and traditional Bridge configurations in terms of latency while maintaining similar CPU usage levels. This highlights the advantage of using SR-IOV with PCI Passthrough to improve network performance without additional CPU overhead. Furthermore, the study has shown that *sender* CPU overhead is much higher than that of a *receiver* virtual machine.

Based on these findings, our current work focuses on studying latency, not using KVM virtualization as previously conducted, but container virtualization, specifically Docker. Following similar footsteps, We explore the integration of SR-IOV with PCI Passthrough in Docker while evaluating its potential to lower packet processing latency at the Linux data plane. The adopted experimental methodology seeks to understand whether integrating SR-IOV in Docker can reduce data plane latency compared to the traditional Bridge configuration, and whether this optimization does not result in significant additional CPU processing costs. Observe that a cloud server may not opt to dedicate considerable CPU resources to optimize packet processing at the data plane. This is the reason behind considering CPU overhead as an important decision metric for selecting a data plane configuration. The practical implications of these findings are substantial for systems administrators and networking professionals, offering them more profound insights into the performance impacts of implementing SR-IOV and PCI Passthrough in Docker environments. This study addresses a critical gap in understanding cloud communication optimization and is a practical guide for leveraging emerging technologies in network virtualization.

Continuing with the analysis of current literature, Midha, Tripathi, and Sharma [26] examine the impact of integrating Docker technology with Software Defined Networks (SDN) in a cloud environment. Their study focuses on network performance parameters such as latency, throughput, bandwidth usage, availability, connectivity, and packet loss. The authors performed simulations on the Google Cloud Platform to analyze SDN performance with and without Docker containers. Key results indicated that Docker containers demonstrate better resource utilization than VMs, especially regarding CPU and memory usage. Additionally, the startup latency for Docker containers was significantly lower than for VMs. This study offers insights into the advantages of Docker containers in virtualized SDN environments, particularly regarding resource efficiency and latency reduction. Although the study provides a comprehensive look at various network performance metrics in cloud environments, it suffers from some limitations. While authors focus on a wide range of performance parameters in SDN, our study is more specific, focusing on SR-IOV in Docker containers and its implications on latency measured reflected on the RTT. Furthermore, we explore how SR-IOV influences resource utilization, offering new insights into resource efficiency in virtualized network scenarios. We also consider different network configurations, use cases, and experimental environments distinct from those used by Midha et al. [26], providing a distinct understanding of the practical implications of SR-IOV and Docker.

In the context of virtualization in high-speed networks, we highlight the study by Jiuxing Liu, [27]. This work investigates the efficiency of I/O virtualization, focusing on 10 Gigabit Ethernet (10GbE) NICs using SR-IOV. Authors report that CPU and memory virtualization have already achieved notable advances, but I/O virtualization, especially for high-speed network devices, still faces considerable challenges. Liu's research evaluates SR-IOV in a virtual KVM environment, comparing it to native bare metal configurations and the Virtual Input/Output (VirtIO) driver. The results indicate that the native bare metal configuration outperforms SR-IOV, mainly due to the overhead inherent to virtualization. However, it is essential to note that SR-IOV performed better when compared to the VirtIO driver, although it required more CPU cycles and context switches than the native configuration. A vital observation of the study is its use of a less mature KVM PCI Passthrough implementation, as reported by the authors, which may have contributed to the performance loss observed in the SR-IOV scenario. This study provides important results into the performance of different virtualization configurations in high-speed networks, highlighting the limitations and potential of SR-IOV compared to other approaches. While Liu's work provides insight into the performance of SR-IOV in an KVM environment, our research offers a more focused examination of the impact of SR-IOV on latency measurements when using Docker containers.

Jae-Geun Cha and Sunwook Kim [28] focus on optimizing network latency for edge services through container technology. The study emphasizes using CNI plugins to enable container interaction, identifying that different plugins result in varying network latency values. The research evaluates the performance impact of using multiple plugins simultaneously. Key findings include the observation that SR-IOV plugins offered twice the throughput compared to kernel-based virtual network interfaces and maintained over 50% of the bandwidth despite adding other virtual network interfaces over a single connection. The authors bring essential findings into the performance of low-latency networks in container-based edge computing environments. Cha and Kim provide an overview of network performance in Docker containers in edge computing; our differs as it focuses on SR-IOV end-to-end latency measured using the RTT metric for applications in cloud environments.

The paper by Liu et al. [29] presents a performance comparison of different container networking solutions, including Flannel, Calico, Weave, and SR-IOV in conjunction withData Plane Development Kit (DPDK). The study aimed to provide a fair comparison for managing hybrid and heterogeneous cloud environments. Key findings include that SR-IOV demonstrated significantly better performance than other solutions, achieving near-linear speed even with small 256-byte packets. Although the authors cover a broader spectrum of containerized networking technologies, they do not examine the impact of SR-IOV on RTT in Docker containers.

The paper [30] by Jie Zhang et al. investigates the performance of hypervisor (e.g., KVM and container-based virtualization containerized (e.g., Docker) in the context of high-performance computing (HPC) on SR-IOV enabled InfiniBand clusters. The main focus of the study is to understand the performance characteristics of different virtualization solutions and virtualized I/O technologies for InfiniBand clusters in the High-Performance Computing (HPC) domain. Researchers conducted a comprehensive evaluation using IB verbs, MPI benchmarks, and applications, characterizing hypervisor and container-based virtualization performance with PCI Passthrough and SR-IOV. The main results include that VMs with PCI Passthrough outperformed those with SR-IOV, even though SR-IOV enabled efficient resource sharing. They also point out that container-based solutions performed better than hypervisor-based solutions. In addition, the work reports that the PCI Passthrough container incurred only 9% overhead in HPC applications compared to native performance.

The study [8] also evaluates the benefits achieved by SR-IOV in a HPC context while using InfiniBand NICs in an Amazon Web Services (AWS) Amazon Elastic Compute Cloud (EC2) cloud environment. Results show that SR-IOV improves network performance, reduces latency overhead, and increases bandwidth to near-line rate. However, the SR-IOV setup still suffers twice the latency compared to the bare metal baseline scenario with no virtualization. Observe that both configurations managed to reach similar bandwidth levels.

The study in [31] explores Network Function Virtualization (NFV) challenges and focuses on Deep Packet Inspection (DPI) Virtual Network Functions (VNFs). It compares the performance of a typical DPI solution running in user space and the same service as a virtual function in an KVM environment with SR-IOV support. It also examines the impact of using the DPDK, a framework that improves packet processing speed, on VM performance and user space. The results show that DPDK achieves a line rate of 10 Gbps for the NIC in the bare-metal environment, whereas the standard Linux kernel reaches a maximum throughput of 1 Gbps. In a virtualized environment, the standard Linux kernel reaches 1 Gbps throughput but is unstable, whereas DPDK experiences a 19% performance degradation compared to the corresponding bare-metal experiment. However, the study does not evaluate SR-IOV performance in a cloud computing environment with multiple VMs and competing background loads.

One can observe significant trends, challenges, and innovations raised by network virtualization and container performance. Studies like those by Midha, Tripathi, and Sharma [26] provide valuable insights into the efficiency of Docker containers in virtualized SDN environments, highlighting benefits such as better resource utilization and reduced latency. On the other hand, research such as that by Liu et al. and Cha and Kim expand the understanding of network performance to include other contexts, such as KVM and edge computing, but may did investigate the specific impacts of SR-IOV on RTT measurements. Furthermore, other works that focused on HPC and NFV environments, such as those by Zhang et al. [30] and the study on Network Function Virtualization, addressed different aspects of network virtualization, from the efficiency of SR-IOV in InfiniBand clusters to the impact of DPDK on virtualized network functions.

These studies highlight the complexity of the different approaches to optimizing network performance in virtual servers, while underscoring the need for more specific research. Our study, which examines the impact of SR-IOV on network performance within cloud computing data centers using the Docker platform, fills a gap in this field. Unlike the works cited or other research in the literature on SR-IOV technology, our study addresses methodologies to understand the nuances of SR-IOV's influence in Docker environments. This distinct focus advances our understanding of network performance optimization in cloud-based infrastructures.

Furthermore, we exhaustively compare the current SR-IOV Docker-based setup with the results from our previous KVM-based study [1]. This discussion exhibits the advantages of container with SR-IOV and

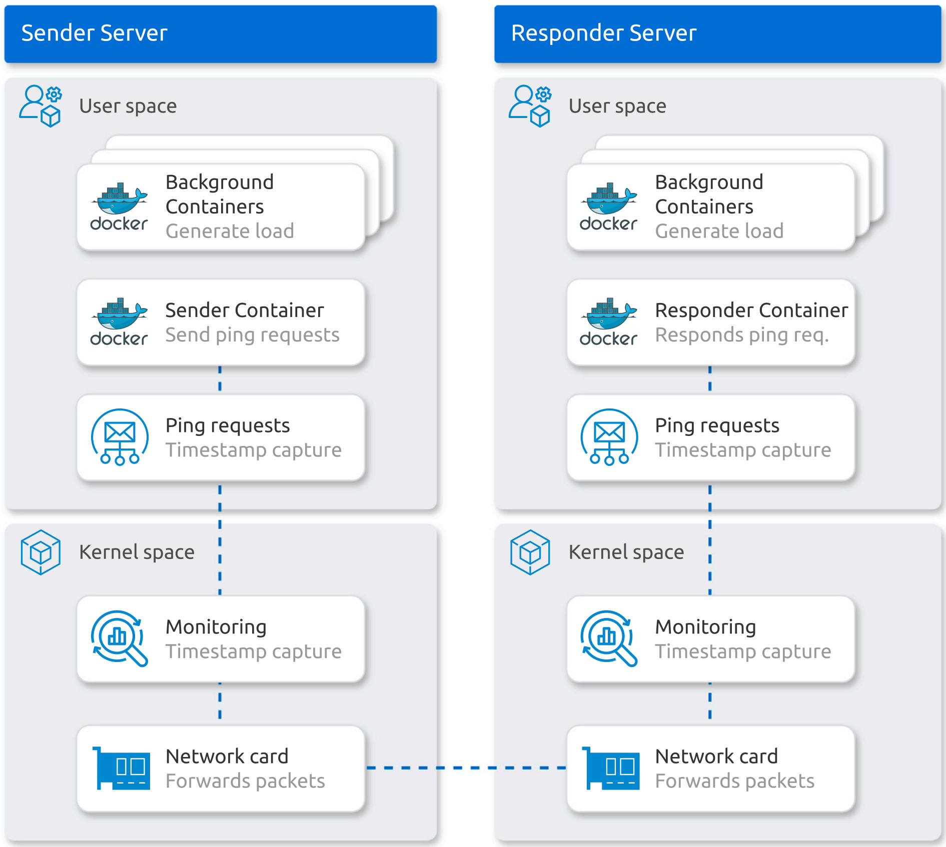

Fig. 2. Experiment setup and flow with timestamp capture points.

identifies some of its shortcomings when using traditional virtualization technologies. Such analysis offers a detailed overview of the nuances of each technology, leading to an understanding of their ideal applications and usage scenarios.

4. Experimental setup

To evaluate the performance of SR-IOV when used with Docker virtualization, we built a testbed consisting of two physical host machines directly (peer-to-peer) connected. As the virtualization platform, the experiments utilize Docker version 26.0.0, build 2ae903e. The host machines are equipped with CPU Intel® Xeon® Bronze 3204, 64 GB of RAM, and Ubuntu Server 20.04 with kernel v5.4.0-171-generic as Operational system. Also, the NICs used were Intel® x540-AT2 with 10 Gigabit ports.

The experiment initiates by booting up the Containers. Next, some background workload is also generated. The **number of Background containers** ranges in the set **0, 8, 32, and 64**. These containers run on either the **Sender** or the **Responder** server. In addition, we run one *main* container (to differentiate if from background containers) on each server or host. The experiment is designed to measure the RTT of packets between the two main containers generated using the ICMP based *ping*[3] packet echo tool. The containers are called Sender containers and Responder containers according to their role. The Sender container runs on the Sender Server and sends ping echo requests to the Responder container. In turn, the latter responds to the received ping echo requests with responses returned back to the Sender container. The adopted experimental testbed is depicted in Fig. 2, with server and container placement.

[3] https://linux.die.net/man/8/ping.

Servers hosted in a Datacenter dedicated to specific tasks (e.g., storage, databases, VNFs, web container servers) are subject to variable overheads based on the type of service and its usage of CPU, Network, and I/O resources. The experiments aim to emulate these different types of background loads. Thus, we measure the impact of creating typical background stress conditions and overload caused by the containers due to the excessive consumption of resources on the host, namely **CPU** and **Network**. We choose these as they seem to influence data plane latency more than I/O operations, based on previews works [1,32]. We call this the **Load Type**, which in turn may be located at the Sender or Responder server.

In each experiment, we send 1000 Internet Control Message Protocol (ICMP)-based echo packets at two frequencies: 1000 packets per second (1 Kpps) and 1,000,000 packets per second (1 Mpps). Whereas 1 Kpps corresponds to a low bandwidth scenario, 1 Mpps represents a high bandwidth scenario. These two rates were selected to represent distinct levels of network load and stress testing conditions. When we deploy CPU load scenarios, we analyze specific CPU resource competition. When we deploy network load scenarios, we want to stress the server and, therefore, the network card with network resource competition; thus, the background VMs send and receive network packets at the highest bandwidth occupation of each VM. Because of this, we use the standard 64-byte ICMP packet size [33]. Smaller packets introduce less fragmentation and processing overhead, which leads to more precise latency measurements [34,35]. For context, 1 Kpps equates to a bandwidth of approximately 8 Mbps (assuming each packet is 1000 bytes - 1 kB). In contrast, 1 Mpps equates to approximately 8 Gbps, demonstrating the system's capability to handle significantly higher throughput. These frequencies were chosen based on prior research aimed at assessing the effects of virtualization on RTT, allowing us to analyze performance under both normal and high-load conditions. This methodology and its justification are well-supported by the literature, including our previous studies [1,36,37].

We considered using metrics other than RTT in our analysis. However, we decided not to include three common network considerations–packet loss, jitter, and throughput–for the following reasons:

- Packet Loss: In our experiment, we used ping to evaluate RTT, sending 1000 ICMP packets in each test. If a packet failed, we repeated the experiment until all packets were successfully transmitted. This ensured that momentary errors or congestion would not affect the results. However, this approach eliminated the possibility of observing small intermittent packet losses that could occur in individual runs. In addition, the ICMP protocol used by ping has a high priority in the network, which increases the chance of transmission failures, making the metric irrelevant in this context [38].
- Jitter: We initially considered measuring jitter, but ping could be better suited to capture temporal variations between packets, which is essential for accurately analyzing this metric. Ping measures response time aggregated with the sending time without the granularity necessary to calculate instantaneous variations, which characterize jitter (work [39]). Furthermore, the fixed interval of sending ICMP packets in ping makes capturing these variations with corrections difficult. The work [38] suggest using specific concepts, such as OWAMP or TWAMP, to measure jitter adequately. Since our main focus was stability and response time (RTT), the jitter measurement was considered redundant and uninformative.
- Throughput: Although we initially considered measuring throughput, this metric was restored for several reasons. Ping uses small ICMP packets (usually 64 bytes plus the header), which naturally results in very low throughput, regardless of network conditions. Thus, measuring throughput with ping does not reflect the actual transmission capacity of the network. Even though we generated throughput graphs, we did not observe any relevant variations or behaviors that would justify their inclusion in the results since they consisted of a constant and predictable throughput, given the behavior of our experiments.

On the other hand, alternative analyses, such as CPU utilization, provided more relevant information about the network performance in the context evaluated, reinforcing our decision to focus on metrics more specific to the controlled scenario.

Although the integration via network plugin for SR-IOV in Docker presents ease of use in the technology, this paper aims to perform a performance evaluation focused specifically on RTT. In this context, we chose not to use the integration strategy via the network plugin. The reason for this choice lies in the premise that using third-party software or plugins to manage the SR-IOV configuration and the assignment of VFs may introduce additional overhead. Such overhead could be a critical component when the evaluation focuses on lowering latency, as an extra layer of abstraction or automation may occasionally adversely affect network performance. To successfully obtain sufficiently accurate latency performance through RTT measurements, minimizing the influence of external factors, including those introduced by additional configuration automation software such as over the shelf plugins, is crucial. Consequently, our study prioritizes approaches that enable more direct and granular control over the configuration of hardware and network resources. This ensures a more precise analysis of the impact of SR-IOV on network performance in Docker environments.

We also refer to how a container connects to a NIC as the container's **Driver**. It can be configured as **Bridge** when the container connects to the default virtual bridge within the Linux Kernel or as **SR-IOV with PCI Passthrough** if the container gets direct access to the NIC. In the Bridge configuration, we use the MAC Virtual Tap (Macvtap) driver to create a virtual bridge between the physical and virtual networks of the containers, allowing isolated communication. Fig. 3 shows how the experiments applied the adopted configurations.

Table 1
Experiment's parameters.

Number of BG VM	0, 8, 32 and 64
Load type	Network Load, CPU Load
Load location	Sender Server, Responder Server
Driver	Linux Bridge, SR-IOV and PCI Passthrough (Fig. 3)
Ping frequency	1 Kpps, 1 Mpps

These configurations are complementary because they represent two distinct approaches to using SR-IOV with containers through direct connections without relying on network plugins. The Bridge configuration with Macvtap allows isolated communication via a virtual bridge, whereas SR-IOV with PCI Passthrough provides direct access to the NIC for dedicated network resource use. This distinction is essential when evaluating RTT performance, as the configuration choice can significantly impact latency measurements. Table 1 shows all parameters used by the experiments.

We added extra timestamp capture points on both servers. Packets are timestamped with the `tcpdump` tool[4] (*v4.9.3*) as they pass through the server's kernel, allowing us to extract additional RTT measurements. These timestamp capture are displayed in Fig. 2.

In addition to RTT, our primary evaluation metric, a comprehensive comparison of CPU consumption under different network configurations is of paramount importance, as highlighted in prior research [1,32,36,37]. Inspired by the results from our previous study [25], we examined the variations in CPU usage between the different evaluated scenarios. We measure this metric using the sysstart tool, part of the mpstat package,[5] in version 12.2.0.

We split the experiments into two parts based on the network configurations implemented in the experiments. Based on whether there is competition for resources with the SR-IOV technology we separated the experiments in the following groups:

- **The Impact of SR-IOV with PCI Passthrough in Docker.** In this part of the experiments, SR-IOV technology is free from external competition for CPU and network resources.
- **Concurrency analysis of SR-IOV with Bridge Configuration in Docker.** In this part of the evaluation, we create an experiment where all Containers use SR-IOV, both the *Main Container* and the background ones. As a result of this, there the background containers compete with the main containers at the sender and receiver hosts.

5. Measurement results

As seen previously, the main objective of this study is to explore the impact of virtualization technology SR-IOV on the packet processing latency represented by the use of RTT and CPU. In this section, we detail the experiments and present their results. Given the large number of considered parameters, we divided the results into subsections based on the following load scenarios: (a) The Impact of SR-IOV with PCI Passthrough in Docker, (b) Concurrency analysis of SR-IOV and Bridge in Docker, and (c) Statistical analysis.

5.1. The impact of SR-IOV with PCI passthrough in Docker

These initial experiments seek to evaluate the impact that SR-IOV in conjunction with PCI Passthrough technology (referred to simply as SR-IOV) have on end-to-end latency. At first, only the *Main VM* utilized

[4] https://manpages.ubuntu.com/manpages/focal/en/man8/tcpdump.8.html.

[5] https://manpages.ubuntu.com/manpages/xenial/man1/mpstat.1.html.

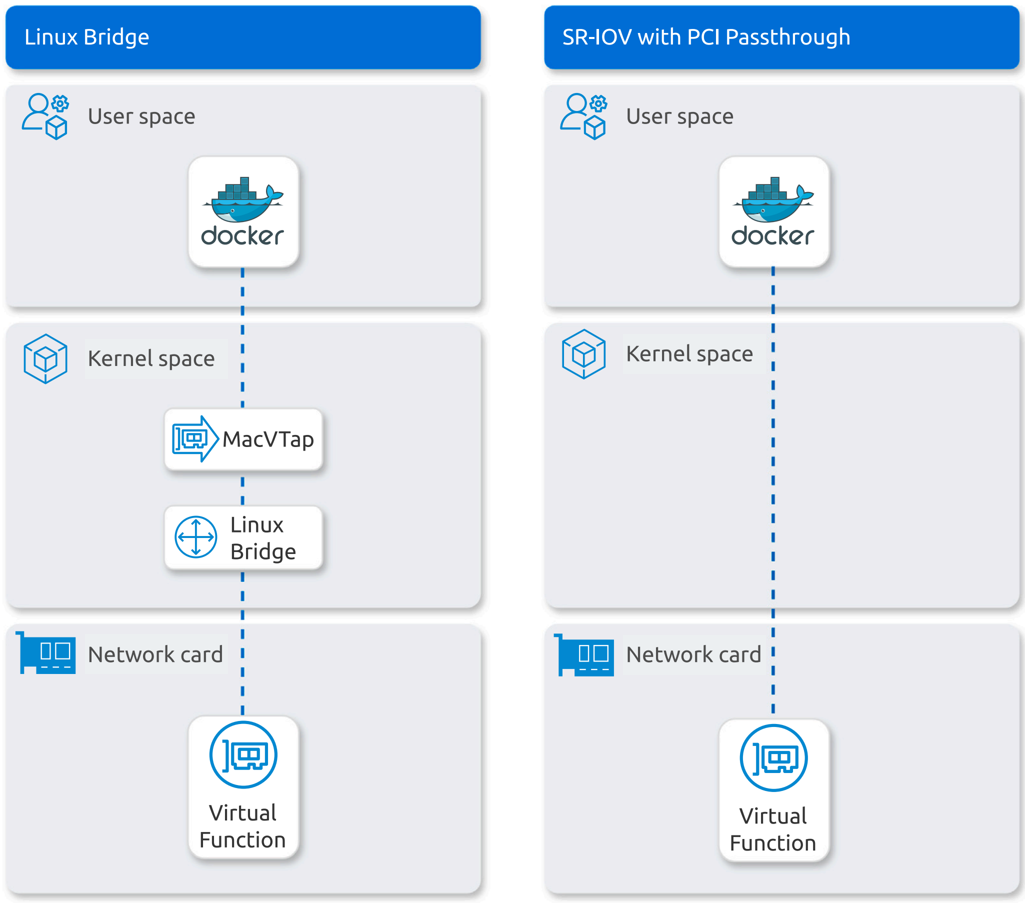

Fig. 3. Illustration the connections used in the experiments.

SR-IOV with PCI Passthrough, while background containers employed the traditional Bridge mode for NIC connectivity.

Latency measure using RTT: Overall, as expected, using SR-IOV with PCI Passthrough positively benefits RTT performance. The RTT median for SR-IOV scenarios is considerably lower than for scenarios with Bridge configuration. This separation becomes evident in the presence of background network load, i.e., when using *Network Load* with a high number of background containers. The achieved RTT in these scenarios with *Network Load* was about ≈10–15 times lower than when using Bridge. This behavior can be observed regardless of whether the *Network Load* Location is at the *Responder* or *Sender* Server, as shown in Figs. 4 and 5, respectively.

Moreover, our analysis indicated that variations in *Ping Frequency* and *Load Location* did not significantly impact the collected latency values. This observation differed from the outcomes observed when employing CPU load type with a high number of background containers on the Sender Server. In such case, we observed a higher variation of RTT samples with 1 Kpps ping frequency and less variation with 1Mpps.

CPU Usage: The results show that the combined SR-IOV and PCI Passthrough configuration generally does not negatively influence CPU usage under Docker, demonstrating that using these technologies shows little impact compared to standard bridging. This result is observed with the *Load location* both on *Sender* and *Responder*, as observed in Figs. 6 and 7.

As evidenced in our investigations, implementing SR-IOV in Docker environments demonstrates a substantial latency improvement similar to the one reported previously by [25]. Specifically, the SR-IOV configuration in conjunction with Macvtap, as explored in [25], highlights a significant decrease in RTT compared to the "Bridge" configuration. These findings reinforce our observations about SR-IOV in Docker, demonstrating the efficiency of this technology in optimizing packet processing latency. The displayed behavior demonstrates the remarkable advantage of using SR-IOV + PCI Passthrough in docker containers. One can achieve considerably lower RTTs while maintaining similar and stable CPU usage regardless of the server load and application type.

5.2. Concurrency analysis of the use of SR-IOV and bridge in Docker

Next, we want to evaluate if a container on a server can achieve similar latency results if all other containers also use SR-IOV. Therefore, in this scenario, every background container uses SR-IOV while the main container uses either standard Bridge or SR-IOV. Since this type of competition can be typical in data centers, it is essential to observe whether increasing the number of VFs used concurrently can overload the network interface.

RTT: Regardless of the *Load Location*, we observed a significant improvement in the RTT in the experiments where the SR-IOV is applied. Simultaneous use of all available VFs on the board does not harm latency values. In fact, we observe that the highest RTT values occur when using Bridge mode on all virtual machines. Concisely, the findings show that using SR-IOV (combined with PCI Passthrough) reduces the RTT, whereas utilizing a Bridge configuration leads to higher packet latency.

Fig. 8 illustrates how using SR-IOV on all containers can still provide lower delays, showing that the concurrency of VFs does not directly degrade performance. This means that even if you have a datacenter with multiple containers using SR-IOV combined with PCI Passthrough, you can still benefit from using one more VF on your container. A similar result is observed in Fig. 9, where the *Load Location* is on the

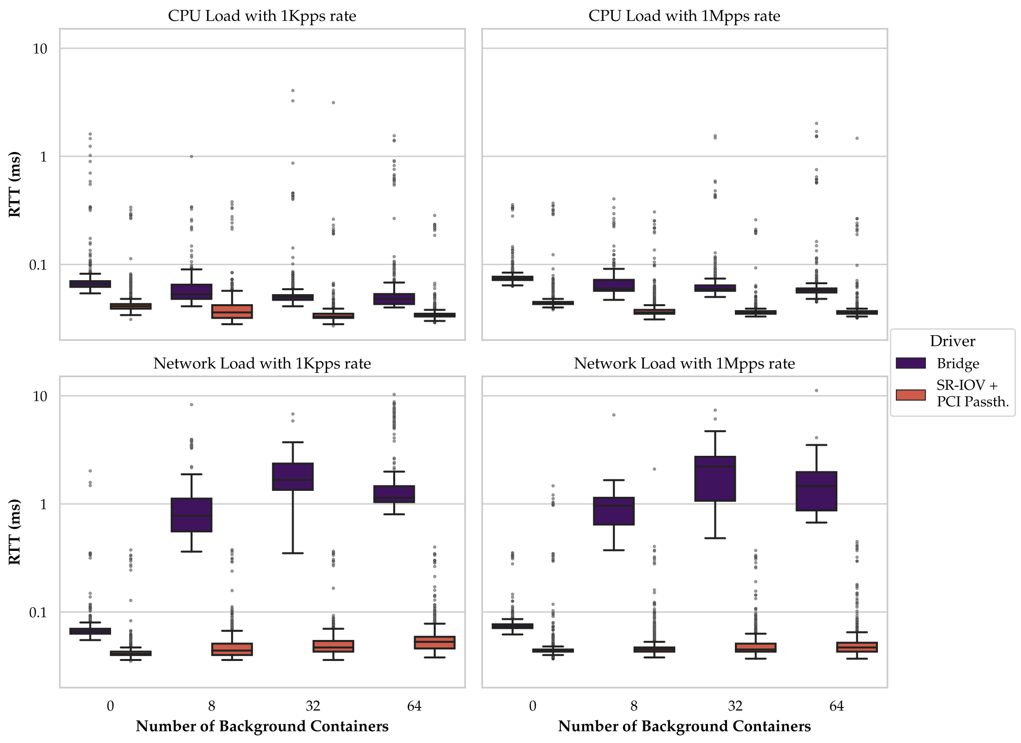

Fig. 4. Results for all experiments conducted to observe the Impact of SR-IOV with PCI Passthrough in Docker. This graphic summarizes the performance of each virtual driver, considering the RTT values of all experiments on the Responder Server.

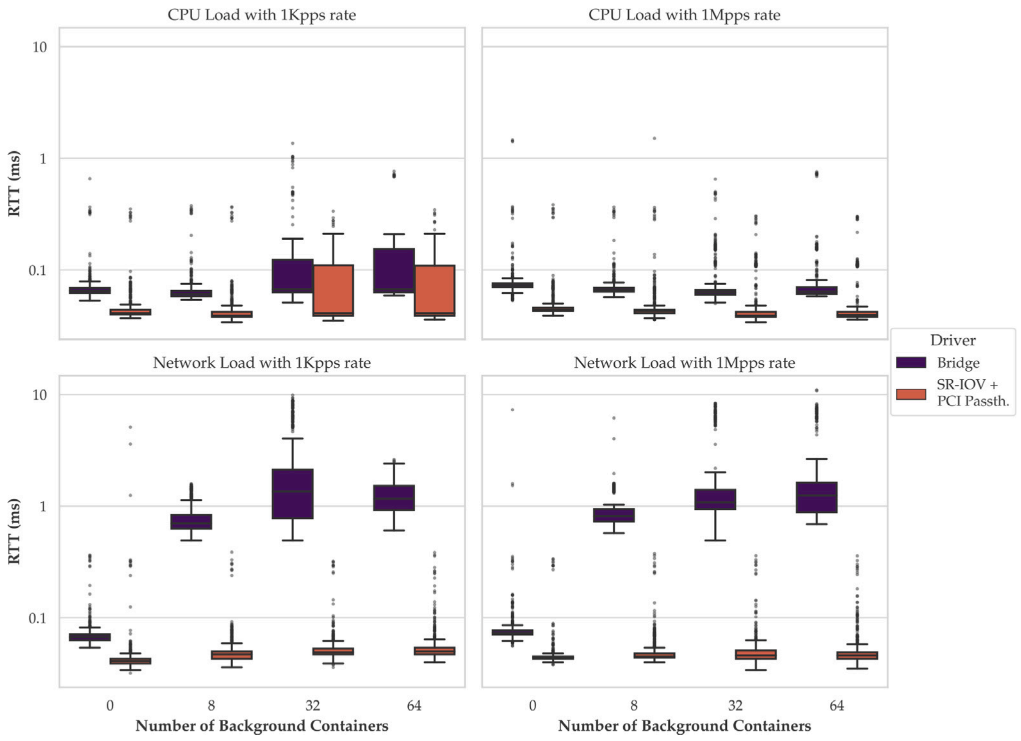

Fig. 5. Results for all experiments conducted to observe the Impact of SR-IOV with PCI Passthrough in Docker. This graphic summarizes the performance of each virtual driver, considering the RTT values of all experiments on the Sender Server.

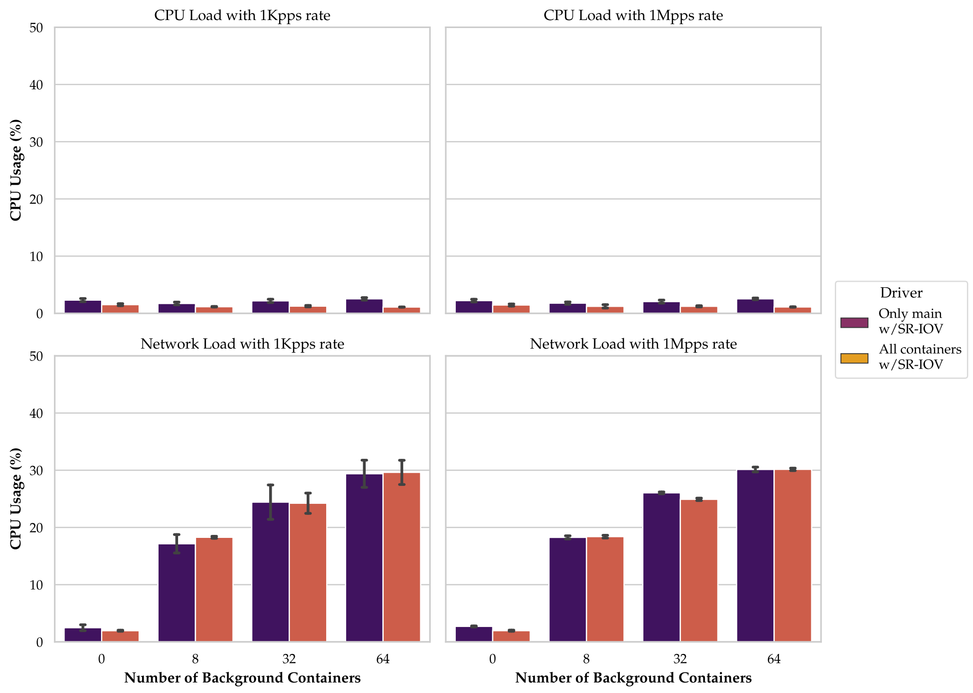

Fig. 6. Results for all experiments conducted to observe the Impact of SR-IOV with PCI Passthrough in Docker. This graphic summarizes the performance of each virtual driver, considering the CPU values of all experiments on the Responder Server.

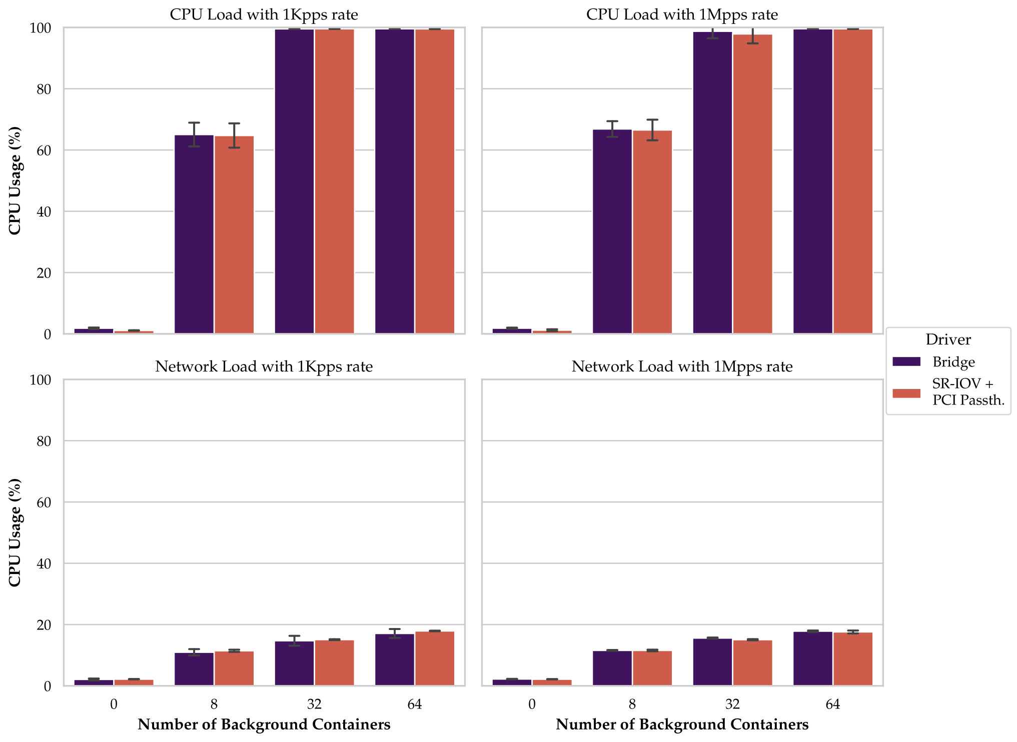

Fig. 7. Results for all experiments conducted to observe the Impact of SR-IOV with PCI Passthrough in Docker. This graphic summarizes the performance of each virtual driver, considering the CPU values of all experiments on the Sender Server.

sender server. There is no difference in the RTT variation in either two scenarios.

We found that when the main container uses SR-IOV and the background containers use a *Bridge*, the results are very similar to when every container uses SR-IOV. When there is a network load with a 1 Kpps rate, there is a slight decrease in the RTT median, showing that the concurrency of SR-IOV usage does not translate to performance loss. We combined the results of the previous section with the ones in this section to illustrate this case and plotted a specific graph, illustrated in Fig. 10.

CPU Usage: Overall, there is a noticeable similarity in CPU usage performance between the two scenarios. The results depicted in Fig. 11 mirror those observed in the previous section. We summarized the results of the responder server in Table 2, incorporating all experiments.

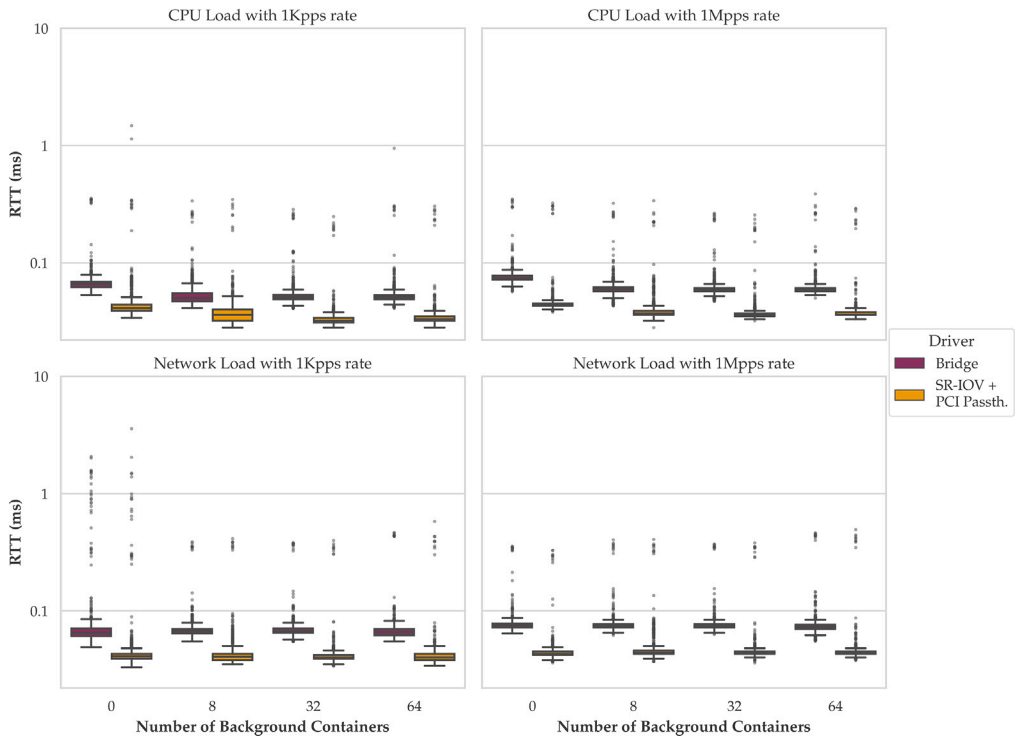

Fig. 8. Results for all experiments conducted to observe the Concurrency analysis of the use of SR-IOV and Bridge in Docker. This graphic summarizes the performance each virtual driver, considering the RTT values of all experiments on the Responder Server.

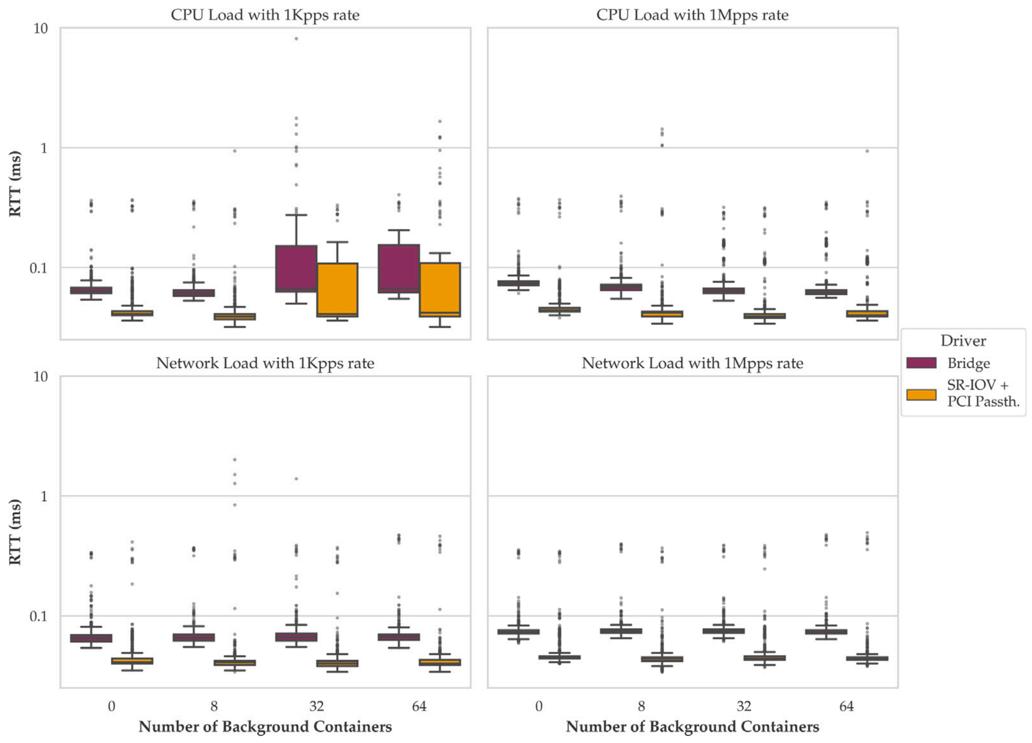

Fig. 9. Results for all experiments conducted to observe the Concurrency analysis of the use of SR-IOV and Bridge in Docker. This graphic summarizes the performance each virtual driver, considering the RTT values of all experiments on the Sender Server.

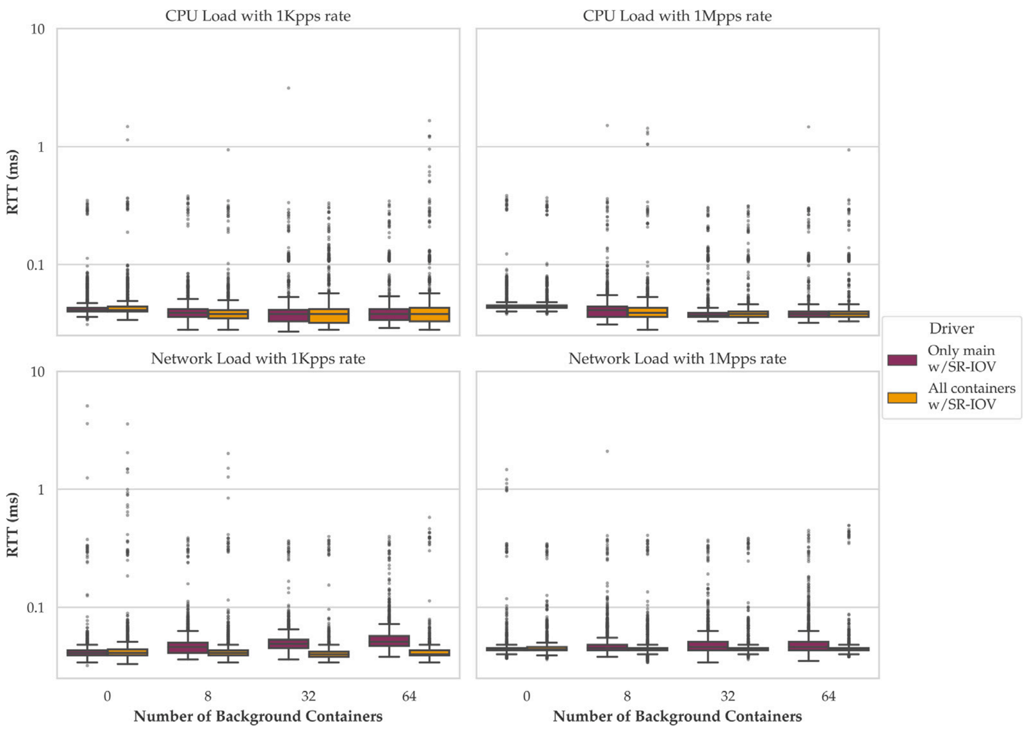

Fig. 10. Combination of both of our scenarios: This graphic compares the scenarios of when only the main container uses SR-IOV and background containers uses Bridge vs. when all containers uses SR-IOV.

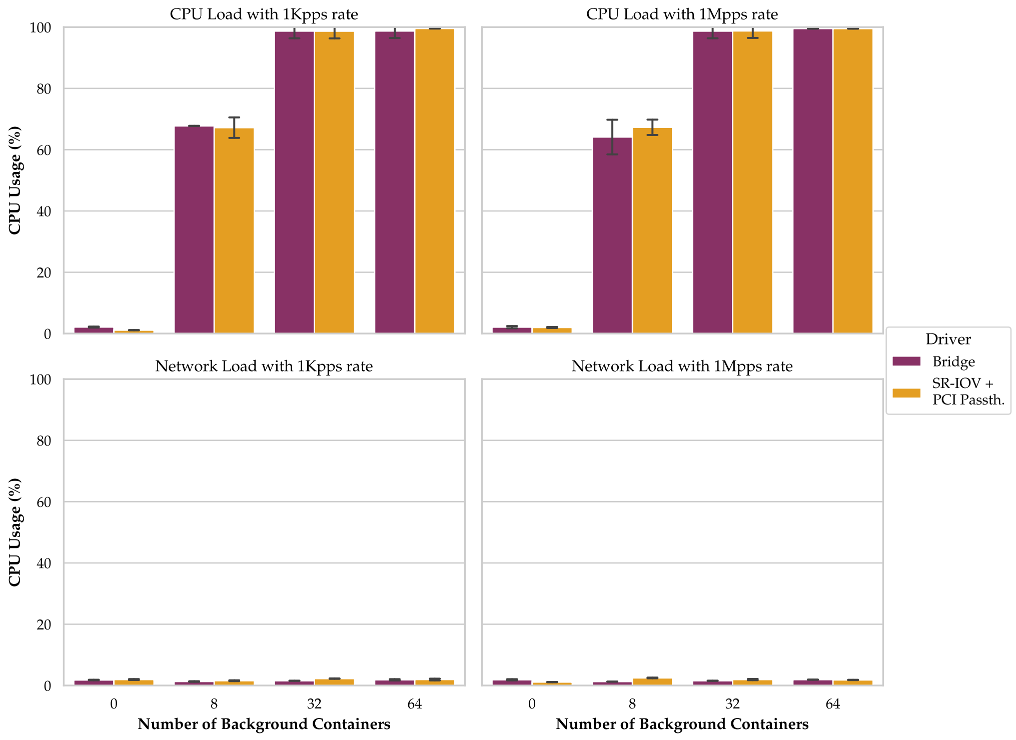

Fig. 11. Results for all experiments conducted to observe the Concurrency analysis of the use of SR-IOV and Bridge in Docker. This graphic summarizes the performance each virtual driver, considering the CPU values of all experiments on the Sender Server.

Notably, the variation remains low, as evidenced by the min and max values across all parameters (load location, load type, ping frequency), with the mean ranging between 1.2% and 2.1%. These findings contrast with those reported in [25], which indicated that under high network load, employing SR-IOV with PCI Passthrough with all KVM VMs might lead to higher CPU usage on the Sender Server.

5.3. Statistical analysis

In this section, statistical tests were conducted using evaluation metrics as indicators to determine any differences in network performance between various technologies. For our research, the introduction of statistical analysis was vital, as it allowed us to analyze the results more

Table 2
Results for all experiments conducted to observe the Concurrency analysis using SR-IOV and Bridge in Docker. This table summarizes the performance of each virtual driver, considering the CPU values of all experiments on the responder server.

Number of background	CPU usage (%)			
containers	Min.	Mean	Max.	Std. deviation
0	1.50	2.125972	2.36	0.121294
8	1.01	1.192083	1.76	0.103326
32	1.05	1.409167	2.30	0.215472
64	1.06	1.569861	1.96	0.308632

precisely. The *Wilcoxon Signed-Rank Test* [40] was adopted with a confidence level of 95% to compare the distribution of two paired groups with two attributes. This non-parametric method indicates whether a pair of two groups with two attributes has significantly different distributions. The hypothesis was based on the question of whether there is a statistical difference in CPU utilization or latency values between using one technology versus another.

Question: Is there a difference in CPU utilization or RTT values between the use of one technology versus another?

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

Table 3 lists the results obtained in the statistical test. The null hypothesis is rejected for all tests, indicating that the distributions are different. This suggests a statistical difference in RTT and CPU usage values for each compared scenario, which shows us that we have different distributions. However, it is essential to note that a statistically significant difference does not necessarily imply a difference in performance when using SR-IOV (PCI) versus bridge. Observations from the results graphs demonstrate that the CPU usage between SR-IOV and bridge is quite similar, and the minor differences and variations are not relevant to the virtualization context analyzed. The statistical test indicates a difference in the data distributions, which may reflect variations in data dispersion or variability rather than a true performance discrepancy. This result could be attributed to factors such as inherent variability in network traffic, differences in workload patterns, or even measurement noise. Therefore, while the statistical test highlights differences in distribution, these do not translate into meaningful differences in CPU performance for the given virtualization scenarios. This reinforces the need to consider statistical results and practical performance metrics when evaluating virtualization technologies.

In addition, the *Kendall rank correlation coefficient* [41] was utilized. It is a non-parametric hypothesis test that evaluates statistical dependence based on the tau coefficient. The Kendall correlation coefficient returns a value from −1 to 1, where 0 indicates no relationship, and −1 and 1 suggest a perfect relationship (with a negative sign indicating an inverse correlation). These tests enabled us to identify which technologies exhibit a stronger or weaker relationship, improving or degrading the performance metrics applied to our scenario.

The evaluation was conducted by dividing the scenarios into two categories: *Load Location on Responder Server* and *Load Location on Sender Server*. An analysis was performed for both categories based on the two observed metrics: CPU utilization at the sender server and RTT values. The correlation showed that in the scenario of all containers using a Bridge (both main and background containers), there is a greater increase in CPU utilization.

On the other hand, in the scenario where all containers use SR-IOV, there is a more significant impact resulting in reduced CPU utilization. This effect is observed when the load is on Sender and Responder servers. This is likely due to the role of the SR-IOV hardware in handling packet processing, alleviating the CPU workload.

Regarding latency, a wider difference between the technologies was observed. The scenarios where all containers used SR-IOV had the most significant impact on reducing RTT, regardless of the *Load Location*. Conversely, the scenario where all containers use a bridge setting strongly correlated with increased RTT values (see Fig. 12).

6. Discussion and conclusion

Virtualization is known to affect the accuracy of RTT measurements [36]. However, with container-based virtualization, such as Docker, this impact is reduced but not totally eliminated [37]. This work evaluated SR-IOV, a technology designed to optimize the packet processing data plane within Docker container virtualization environments. Specifically, we examined the benefits and challenges of integrating SR-IOV with Docker. Our experiments utilized active ICMP ping monitoring to measure network performance across various scenarios, identifying key variables and system resources that impact latency measurements.

The applicability of SR-IOV in Docker environments was highlighted through detailed case studies. These studies demonstrated SR-IOV's ability to significantly decrease packet data plane latency while maintaining data integrity and security in highly virtualized environments. The optimized configuration of SR-IOV resulted in tangible performance improvements, providing practical references for professionals and researchers.

Our measurements showed that traditional bridging techniques result in less efficient network performance than SR-IOV. Bridging involves routing packets through the host's kernel, adding overhead due to context switching and copying between user and kernel space. This process increases latency and reduces throughput, especially under high network loads. In contrast, SR-IOV minimizes latency and maximizes throughput by bypassing these intermediate layers. Additionally, while our lightweight measurements did not significantly impact CPU overhead, the efficiency gains with SR-IOV are notable without additional CPU load, making it a viable option for data centers. The results suggest that SR-IOV positively improves RTT. Therefore, it is highly recommended that cloud operators deploy SR-IOV technology with PCI Passthrough, especially when offering time-critical services related to financial trading centers, online gaming platforms, virtual and extended reality, high-performance computing applications, and cloud computing providers.

While integrating SR-IOV with Docker environments offers significant performance improvements over traditional bridging techniques, cloud operators face several challenges:

- Hardware Compatibility: Not all NICs support SR-IOV, needing careful planning and potential hardware upgrades.
- Configuration and Management: SR-IOV requires precise setup and maintenance, including enabling BIOS/UEFI settings, creating VFs, and dealing with PCI specification.
- Dynamic Scaling: SR-IOV's static nature can pose challenges in environments requiring flexibility and rapid provisioning.
- Expertise: The initial setup and ongoing management require significant and specialized expertise.

The following are some findings from this research that we highlight:

- SR-IOV and PCI Passthrough provide a more efficient and direct communication path between virtualized resources and physical infrastructure, improving network performance and lowering RTT by approximately 10 to 15 times.
- The use of SR-IOV and PCI Passthrough does not increase CPU usage, making it a viable option for data centers to enhance network performance without additional CPU overhead.
- Network performance RTT in Docker containers suffer more impact from SR-IOV than by the overall data center environment.

Table 3
Wilcoxon Signed-Rank Test.

Virtualization type	Server	Sub-topic	Group	p-Value (Wilcoxon)	Comprehension
Docker	Sender	CPU	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)
		RTT	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)
	Responder	CPU	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)
		RTT	SRIOV (PCI) vs. Bridge	0,0000000	Different distribution (reject H0)

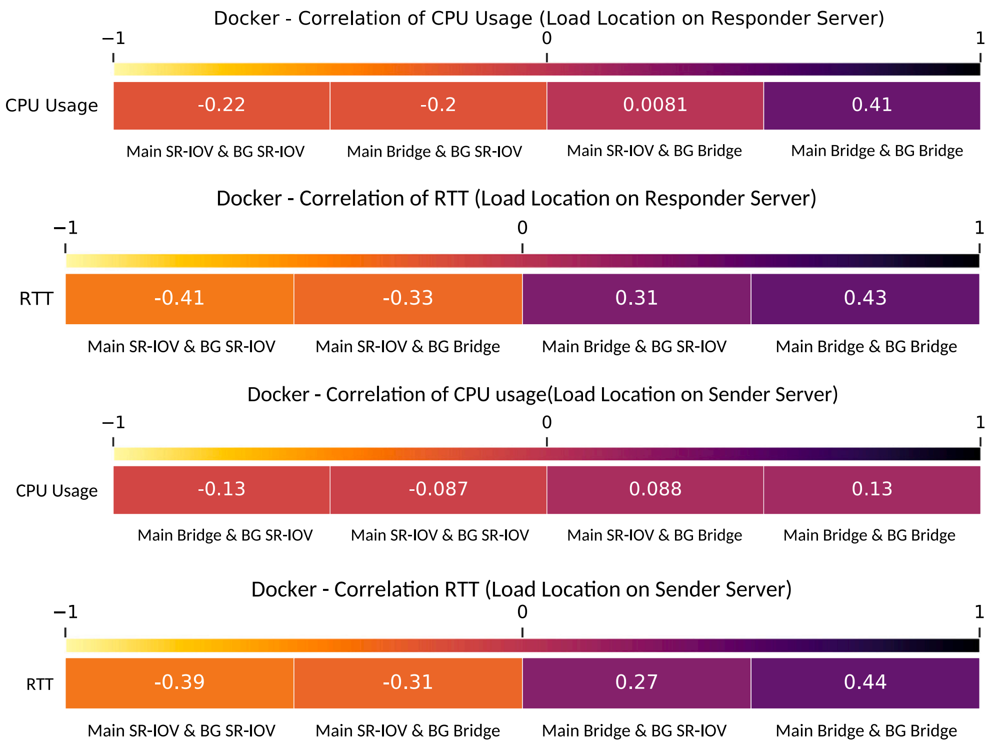

Fig. 12. Kendall rank correlation Coefficient.

- SR-IOV used in Bridge mode with Macvtap is beneficial for scenarios requiring straightforward setup and ease of management, where network traffic isolation is important but performance demands are moderate. Conversely, SR-IOV with PCI Passthrough is optimal for high-performance environments where direct and low-latency access to the NIC is essential, providing enhanced network throughput and reduced overhead.

In summary, integrating SR-IOV within Docker environments significantly reduces end-to-end latency and enhances the overall performance of virtualized systems. Docker can provide fast and efficient container networking by leveraging direct access to physical NIC resources, improving application performance and communication.

Future research should evaluate the response of other virtualization technologies, such as VMware, to SR-IOV combined with PCI Passthrough. Additionally, explore alternative mechanisms to PCI Passthrough in conjunction with SR-IOV. We would also like to make a more significant variation about some factors, such as the size of the packages used in the experiments, and analyze how this influences them.

CRediT authorship contribution statement

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

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 research was supported by *Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP)*, Brazil, *Fundação de Amparo à Ciência e Tecnologia de Pernambuco (FACEPE)*, Brazil - *process number 2023/00420-1*, and *Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)*, Brazil.

References

[1] A.T. de Oliveira Filho, E. Freitas, P.R.X. do Carmo, D.H.J. 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/abs/pii/S0140366421004722.

[2] M.K. Patra, A. Kumari, B. Sahoo, A.K. Turuk, Docker security: Threat model and best practices to secure a docker container, in: 2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security, ISSSC, 2022, pp. 1–6, http://dx.doi.org/10.1109/iSSSC56467.2022.10051481.

[3] A. Solis, W.J. Allen, E. Ferlanti, Containerizing visualization software: Experiences and best practices, in: Practice and Experience in Advanced Research Computing, Association for Computing Machinery, 2022, pp. 1–8.

[4] D. Berkholz, Docker index shows continued massive developer adoption and activity to build and share apps with docker, 2021, URL https://www.docker.com/blog/docker-index-shows-continued-massive-developer-adoption-and-activity-to-build-and-share-apps-with-docker/, Internet.

[5] W.-H. Chiu, C.-T. Chang, S.-T. Liu, L.-S. Chu, B.-W. Cheng, H.-R. Chi, Exploring innovation model and evolution of medical cloud service: A case study of Chung-Hwa telecom, in: ICSSSM12, 2012, pp. 435–438, http://dx.doi.org/10.1109/ICSSSM.2012.6252272.

[6] C.-H. Hong, K. Lee, J. Hwang, H. Park, C. Yoo, Kafe: Can OS kernels forward packets fast enough for software routers? IEEE/ACM Trans. Netw. 26 (6) (2018) 2734–2747, http://dx.doi.org/10.1109/TNET.2018.2879752.

[7] M. Bencivenni, D. Bortolotti, A. Carbone, A. Cavalli, A. Chierici, S. Dal Pra, D. De Girolamo, L. dell'Agnello, M. Donatelli, A. Fella, D. Galli, A. Ghiselli, D. Gregori, A. Italiano, R. Kumar, U. Marconi, B. Martelli, M. Mazzucato, M. Onofri, G. Peco, S. Perazzini, A. Prosperini, P.P. Ricci, E. Ronchieri, F. Rosso, D. Salomoni, V. Sapunenko, V.M. Vagnoni, R. Veraldi, M.C. Vistoli, S. Zani, Performance of 10 gigabit ethernet using commodity hardware, IEEE Trans. Nucl. Sci. 57 (2) (2010) 630–641, http://dx.doi.org/10.1109/TNS.2009.2032264.

[8] 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, p. 7, http://dx.doi.org/10.1145/2616498.2616537.

[9] E. Freitas, A.T. de Oliveira Filho, P.R.X. 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, URL https://www.sciencedirect.com/science/article/pii/S0140366422003917.

[10] 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.

[11] PCI-SIG, Peripheral component interconnect special interest group, 2023, URL http://www.pcisig.com/home. (Accessed 12 December 2023).

[12] Intel Corporation, Configure SR-IOV network virtual functions in linux* KVM, 2023, URL https://www.intel.com/content/www/us/en/developer/articles/technical/configure-sr-iov-network-virtual-functions-in-linux-kvm.html. (Accessed 12 December 2023).

[13] X. Merino, C.E. Otero, The cost of virtualizing time in linux containers, in: 2022 IEEE Cloud Summit, 2022, pp. 63–68, http://dx.doi.org/10.1109/CloudSummit54781.2022.00016.

[14] I.-M. Stan, D. Rosner, Ş.-D. Ciocîrlan, Enforce a global security policy for user access to clustered container systems via user namespace sharing, in: 2020 19th RoEduNet Conference: Networking in Education and Research, RoEduNet, 2020, pp. 1–6, http://dx.doi.org/10.1109/RoEduNet51892.2020.9324866.

[15] B.K. Aakriti Sharma, A. Sangwan, Optimization of docker container security and its performance evaluation, J. Discrete Math. Sci. Cryptogr. 24 (8) (2021) 2365–2375, http://dx.doi.org/10.1080/09720529.2021.2014142, arXiv:https://doi.org/10.1080/09720529.2021.2014142.

[16] T. Nagendra, R. Hemavathy, Unlocking kubernetes networking efficiency: Exploring data processing units for offloading and enhancing container network interfaces, in: 2023 4th IEEE Global Conference for Advancement in Technology, GCAT, 2023, pp. 1–7, http://dx.doi.org/10.1109/GCAT59970.2023.10353542.

[17] A. Ayub, M. Ishaq, M. Munir, Enhancement in multus CNI for DPDK applications in the cloud native environment, in: 2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN, 2023, pp. 66–68, http://dx.doi.org/10.1109/ICIN56760.2023.10073475.

[18] S. Qi, S.G. Kulkarni, K.K. Ramakrishnan, Understanding container network interface plugins: Design considerations and performance, in: 2020 IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN, 2020, pp. 1–6, http://dx.doi.org/10.1109/LANMAN49260.2020.9153266.

[19] D. Haja, M. Szalay, B. Sonkoly, G. Pongracz, L. Toka, Sharpening kubernetes for the edge, in: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, in: SIGCOMM Posters and Demos '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 136–137, http://dx.doi.org/10.1145/3342280.3342335.

[20] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, H. Guan, High performance network virtualization with SR-IOV, J. Parallel Distrib. Comput. 72 (11) (2012) 1471–1480, http://dx.doi.org/10.1016/j.jpdc.2012.01.020, URL https://www.sciencedirect.com/science/article/pii/S0743731512000329, Communication Architectures for Scalable Systems.

[21] L.-M. Wang, A. Zelezniak, E.S. Daniels, T. Miskell, L.-D. Chen, Build an SR-IOV hypervisor, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM, 2019, pp. 539–544.

[22] J. Li, S. Xue, W. Zhang, R. Ma, Z. Qi, H. Guan, When I/O interrupt becomes system bottleneck: Efficiency and scalability enhancement for SR-IOV network virtualization, IEEE Trans. Cloud Comput. 7 (4) (2019) 1183–1196, http://dx.doi.org/10.1109/TCC.2017.2712686.

[23] S. Bauer, D. Raumer, P. Emmerich, G. Carle, Intra-node resource isolation for SFC with SR-IOV, in: 2018 IEEE 7th International Conference on Cloud Networking, CloudNet, 2018, pp. 1–6, http://dx.doi.org/10.1109/CloudNet.2018.8549547.

[24] S. Cirici, M. Paolino, D. Raho, SVFF: An automated framework for SR-IOV virtual function management in FPGA accelerated virtualized environments, in: 2023 International Conference on Computer, Information and Telecommunication Systems, CITS, 2023, pp. 1–6, http://dx.doi.org/10.1109/CITS58301.2023.10188786.

[25] A.T. de Oliveira Filho, E. Freitas, P.R. do Carmo, D.F. Sadok, J. Kelner, Measuring the impact of SR-IOV and virtualization on packet round-trip time, Comput. Commun. 211 (2023) 193–215, http://dx.doi.org/10.1016/j.comcom.2023.09.013, URL https://www.sciencedirect.com/science/article/pii/S0140366423003237.

[26] S. Midha, K. Tripathi, M. Sharma, Practical implications of using dockers on virtualized SDN, Webology 18 (SI01) (2021) 312–330, http://dx.doi.org/10.14704/WEB/V18SI01/WEB18062.

[27] 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.

[28] J.-G. Cha, S.W. Kim, Design and evaluation of container-based networking for low-latency edge services, in: 2021 International Conference on Information and Communication Technology Convergence, ICTC, 2021, pp. 1287–1289, http://dx.doi.org/10.1109/ICTC52510.2021.9620212.

[29] H. Liu, Y. Luo, B. Chen, Y. Yang, Performance evaluation of container networking technology, in: 2023 IEEE 3rd International Conference on Information Technology, Big Data and Artificial Intelligence, ICIBA, Vol. 3, 2023, pp. 815–818, http://dx.doi.org/10.1109/ICIBA56860.2023.10165552.

[30] 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.

[31] 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.

[32] 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.

[33] Linux man-pages project, Ping(8) - linux man page, 2024, URL https://man7.org/linux/man-pages/man8/ping.8.html. (Accessed 03 October 2024).

[34] K. Phemius, M. Bouet, Monitoring latency with OpenFlow, in: Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, 2013, pp. 122–125, http://dx.doi.org/10.1109/CNSM.2013.6727820.

[35] S. Gallenmüller, F. Wiedner, J. Naab, G. Carle, Ducked tails: Trimming the tail latency of(f) packet processing systems, 2021, pp. 537–543, http://dx.doi.org/10.23919/CNSM52442.2021.9615532.

[36] 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.

[37] A.T.d. Oliveira Filho, Uma análise experimental de desempenho do protocolo QUIC, in: Mestrado em Ciência da Computação, UFPE, Recife, 2020, p. 101, URL https://repositorio.ufpe.br/handle/123456789/37646.

[38] C. Pelsser, L. Cittadini, S. Vissicchio, R. Bush, From paris to tokyo: on the suitability of ping to measure latency, in: Proceedings of the 2013 Conference on Internet Measurement Conference, 2013, http://dx.doi.org/10.1145/2504730.2504765.

[39] P. Skurowski, R. Wójcicki, Z. Jerzak, Evaluation of ip transmission jitter estimators using one-way active measurement protocol (owamp), 2010, pp. 153–162, http://dx.doi.org/10.1007/978-3-642-13861-4_15.

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

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