Research, Society and Development, v. 11, n. 12, e26111234200, 2022
(CC BY 4.0) | ISSN 2525-3409 | DOI: http://dx.doi.org/10.33448/rsd-v11i12.34200

Takeaways from an experimental evaluation of eXpress Data Path (XDP) and Data Plane Development Kit (DPDK) under a Cloud Computing environment

Aprendizados de uma avaliação experimental do eXpress Data Path (XDP) e Data Plane Development Kit (DPDK) em um ambiente de Computação em Nuvem

Conclusiones de una evaluación experimental de eXpress Data Path (XDP) y Data Plane Development Kit (DPDK) en un entorno de Computación en la Nube

Received: 08/17/2022 | Reviewed: 08/27/2022 | Accept: 08/28/2022 | Published: 09/06/2022

Eduardo Freitas
ORCID: https://orcid.org/0000-0002-5752-525X
Universidade Federal de Pernambuco, Brazil
E-mail: eduardo.freitas@gprt.ufpe.br
Assis T. de Oliveira Filho
ORCID: https://orcid.org/0000-0001-9873-6929
Universidade Federal de Pernambuco, Brazil
E-mail: assis.tiago@gprt.ufpe.br
Pedro R. X. do Carmo
ORCID: https://orcid.org/0000-0002-7952-3239
Universidade Federal de Pernambuco, Brazil
E-mail: pedro.carmo@gprt.ufpe.br
Djamel F. H. Sadok
ORCID: https://orcid.org/0000-0001-5378-4732
Universidade Federal de Pernambuco, Brazil
E-mail: jamel@gprt.ufpe.br
Judith Kelner
ORCID: https://orcid.org/0000-0002-2673-5887
Universidade Federal de Pernambuco, Brazil
E-mail: jk@gprt.ufpe.br

Abstract
The popularization of the Linux kernel at data center servers became a significant factor to consider when developing or deploying network applications. New "packet processing frameworks" emerged to ensure fast packet processing inside these servers that rely on the Linux kernel, XDP and DPDK being the two main ones. Alongside this, Cloud Computing environments dominated the data center scenario, bringing virtualization to the center of these servers. This research compares DPDK and XDP packet processors when subject to a cloud computing environment with I/O, CPU, and network load. We first describe an architecture that implements frameworks and emulates a cloud environment. We measure throughput and packet loss while varying the number of competing VMs, CPU usage by the frameworks, and packet size and present our results in the form of takeaways. Furthermore, we conclude that the virtual environment can impact the performance of both frameworks depending on the received packet size and the type of workload the cloud environment performs. A significant case to highlight is when the cloud environment performs network load, and the servers receive small-sized packets, creating both throughput degradation and increasing packet loss. We conclude our experiments by conducting statistical non-parametric hypothesis tests to enhance our conclusions and have a closer look at the results.
Keywords: XDP, DPDK; Linux kernel; Fast Packet processing frameworks; Cloud computing.

Resumo
A popularização do "*kernel*" Linux nos servidores de data centers tornou-se um fator significativo a ser considerado ao desenvolver ou implantar aplicações de rede. Novos "*frameworks* de processamento de pacotes" surgiram para garantir o rápido processamento de pacotes dentro desses servidores que dependem do "*kernel*" Linux, sendo o XDP e o DPDK os dois principais. Paralelamente, os ambientes de computação em nuvem dominaram o cenário de *data centers*, trazendo a virtualização para o centro destes servidores. Esta pesquisa compara os processadores de pacotes DPDK e XDP quando submetidos a um ambiente de computação em nuvem com E/S, CPU e carga de rede. Primeiro descrevemos uma arquitetura que implementa os frameworks e emula um ambiente de nuvem. Medimos o rendimento e a perda de pacotes enquanto variamos o número de VMs concorrentes, o uso de CPU pelos frameworks e o tamanho dos pacotes e apresentamos nossos resultados sob a forma de aprendizados. Concluímos que o ambiente virtual pode impactar o desempenho de ambos os frameworks, dependendo do tamanho do pacote recebido e da carga de uso de

recursos que o ambiente em nuvem realiza. Um caso significativo a ser destacado é quando o ambiente em nuvem realiza carga de rede, e os servidores recebem pacotes pequenos, criando tanto a degradação do rendimento quanto o aumento da perda de pacotes. Concluímos nossos experimentos conduzindo testes estatísticos de hipóteses não paramétricas para aprimorar nossas conclusões e analisar mais de perto os resultados.
Palavras-chave: XDP, DPDK; Kernel Linux; Frameworks de processamento de pacotes; Computação em nuvem.

Resumen

La popularización del kernel Linux en los servidores de los centros de datos se convirtió en un factor importante a tener en cuenta a la hora de desarrollar o implantar aplicaciones de red. Surgieron nuevos "frameworks de procesamiento de paquetes" para garantizar un rápido procesamiento de paquetes dentro de estos servidores que dependen del kernel Linux, siendo XDP y DPDK los dos principales. Junto a esto, los entornos de Computación en la Nube dominaron el escenario de los centros de datos, llevando la virtualización al centro de estos servidores. Esta pesquisa compara los procesadores de paquetes DPDK y XDP cuando se someten a un entorno de Computación en la Nube con carga de E/S, CPU y red. Primero describimos una arquitectura que implementa marcos y emula un entorno de nube. Medimos el rendimiento y la pérdida de paquetes mientras variamos el número de máquinas virtuales en competencia, el uso de la CPU por parte de los marcos y el tamaño de los paquetes, y presentamos nuestros resultados en forma de conclusiones. Llegamos a la conclusión de que el entorno virtual puede afectar al rendimiento de ambos marcos en función del tamaño de los paquetes recibidos y del tipo de carga de trabajo que realiza el entorno de la nube. Un caso significativo a destacar es cuando el entorno de la nube ejecuta carga de red y los servidores reciben paquetes de pequeño tamaño, lo que genera una degradación del rendimiento y un aumento de la pérdida de paquetes. Concluimos nuestros experimentos haciendo pruebas de hipótesis estadísticas no paramétricas para mejorar nuestras conclusiones y tener una visión más cercana de los resultados.
Palabras clave: XDP, DPDK; Kernel Linux; Frameworks de procesamiento de paquetes; Computación en la nube.

1. Introduction

Over the recent years, the Internet has become a vital asset to our society in their daily activities. Internet services are essential and increasingly demand more resources. Emerging applications such as audio/video streaming services, 5G for Industry 4.0, and Automotive Networks are sensitive and seek guarantees for bounded latency. Data center servers have been fitted with Network Interface Card (NIC) running at 40, 100, and more recently 200Gbps. To speed up packet processing, new programmable NICs were introduced [Liu et al., 2019] to unload part of the packet processing from the host onto dedicated processing units inside the network card.

Despite the performance gains achieved using new hardware, there is a need to examine and tune the associated operating software to take full advantage of hardware advances. The operating system's kernel is at the heart of such interfacing software, responsible for the interaction between the hardware and user space applications. According to [Surveys, 2021], the GNU/Linux Operating System (OS) is the most frequently used one in data centers, which makes the Linux kernel a widely used kernel in data center servers and consequently a significant performance decisive component.

As a general-purpose kernel, Linux implements a wide range of communication services with varying requirements, including security, performance, and high availability. As a result, it has a complex communication protocol stack that may cause unforeseen overheads in packet processing. The works in [Rizzo, 2012], [Belay et al., 2014], [Høiland Jørgensen et al., 2018] and [Jeong et al., 2014] report multiple Linux communication-related performance issues in the context of data centers. Without the necessary changes to Linux packet processing, this may be viewed as a bottleneck, providing only suboptimal throughput well below the supported NIC line-rate performance.

In addition, Linux supports a wide variety of communications protocols, devices, and use-cases. This can vary from Internet of Things (IoT) sensors to smartphones, data center servers, or embedded devices. This wide variety of concerns requires Linux to provide a platform-agnostic network packet processing, which does not leverage platform-specific features. This ultimately leads to the embedding of costly packet processing.

Some Linux shortcomings in network packet processing can be addressed. First, the frequent allocation and reallocation of the **sk_buff** buffer cause a significant CPU stress [Han et al., 2010] to deal with related operations, like buffer

initialization, allocation, and request control. Also, System Calls are responsible for lowering performance as well. They require a process to switch between user and kernel modes. This not only introduces CPU overhead in the form of wasted cycles spent on the mode switch [Tsuna, 2010], but also leads to a phenomenon known as Processor State Pollution [Soares and Stumm, 2010]. Such phenomenon describes the fact that every structure in the CPU, like L1, L2, and L3 data cache and Translation Look-aside Buffers, is populated with the current state (e.g., kernel mode state), and every time a CPU switches to another state, it must change all its data and buffer structures as well, which consumes time and CPU cycles. Further properties of the Linux packet processing that contribute towards the bottleneck issue include the execution of multiple packet copying processes [Bruijn and Dumazet, 2017], the presence of large **sk_buff** buffers often with data not needed by all protocols [Wu et al., 2007], and also the full implementation of layer 3 and 4 protocols, which are sometimes unnecessary to specific user space applications [Jeong et al., 2014].

All of these issues prevent Linux from achieving a high throughput performance. Studies conducted in [Høiland-Jørgensen et al., 2018] show that when operating Linux on a 100GbE network card with one core, it can only achieve a maximum of 3.5Gbps throughput. This shows how suboptimal Linux can be regarding network processing. The overall picture is not promising as every data center server that runs Linux subjects itself to these processing issues, potentially causing low network performance.

Midst this scenario, a diversity of so-called packet processors or packet processing frameworks emerged in an attempt to solve such kernel problems and enable fast packet processing inside servers. They mainly are software solutions seeking to achieve line-rate processing and fast packet delivery to user space applications. Typical applications that require line-rate packet processing are firewall applications, Deep Packet Inspection (DPI), Virtual Network Functions (VNFs) or Distributed Denial-of-Service (DDoS) protection. Note that these solutions are not designed as general-purpose ones but performance-driven instead. In other words, they often make processing choices favoring their scope instead of concerning themselves with compatibility with multiple use-cases.

Among the diverse and sometimes competing proposals, currently two main packet processors, namely, eXpress Data Path (XDP) [Høiland-Jørgensen et al., 2018] and Data Plane Development Kit (DPDK) [Foundation, 2018] stand out. Other packet processing frameworks such as SR-IOV and the Vector Packet Processor (VPP) are not considered in this research as the authors focus on comparing the XDP and DPDK frameworks. Both are widely adopted in the academy and by different companies in the networking field. However, they achieve fast packet processing in rather opposite ways. DPDK uses the kernel bypass approach, whereas XDP uses the in-kernel processing. Both frameworks and their approaches are detailed in future sections.

According to [IDG, 2020], in 2020, approximately 81% of IT organizations uses cloud computing in their data centers or have the application deployed in the cloud, which helps understand how cloud computing is predominant in data centers. The widespread use of the Cloud Computing paradigm in current data centers introduces new performance requirements while adding complexity to packet processing. A key technology that acts as an enabler to Cloud Computing is Virtualization, putting virtual environments at the core of all these Cloud Computing Data Centers. The virtualization of operating systems includes the design of virtual network drivers and the introduction of new processing layers spanning between host and virtual machines. These software layers contribute to the packet processing overhead and impact its performance. In addition, even emerging and more modern packet processor frameworks are not limited to deployment inside the virtual environment in a Virtual Machine (VM). These frameworks can operate over a host OS concurrently with the virtual environment, which creates an unpredictable resource competition between the VMs and the frameworks.

Virtual environments are known to create overhead for network applications, degrading their performance

[de Oliveira Filho et al., 2022] [Liu, 2010] [Dantas et al., 2015]. Because of this, one must be careful in evaluating the interplay between the different existing virtual environments (for example, NIC virtualization technologies) and a given packet processing framework. It may be the case that a given virtualization mechanism may interfere with and degrade the performance of these packet processing frameworks, preventing them from achieving the optimal performance they are designed for.

In this work, we perform multiple experiments to measure and estimate how the virtual environment from typical cloud computing data centers can interfere with the performance of DPDK and XDP frameworks, considering the metrics of packet throughput and packet loss.

This article is organized as follows. Section 2 explains how XDP and DPDK works. Section 3 details some related works and where this paper stands in the literature. Section 4 details the experimental setup used to perform the evaluation. Sections 5, 6 and 7 discuss the results from the experiments and Section 8 concludes the work with final thoughts and insights.

This article is based on the master thesis in [Freitas, 2021] that details additional experiments with different discussions. We offer a closer look at the experiments and bring more insights into the results.

2. Background

XDP and DPDK approach the packet processing bottleneck problem in different opposing ways. Whereas DPDK uses kernel bypass and places network processing in user space, XDP maintains it within the kernel while using a special built-in packet filtering mechanism that is visible to applications.

2.1 DPDK

As already mentioned, DPDK uses the kernel bypass technology to achieve high processing performance. First, DPDK takes control of the network card by loading a special user space NIC driver developed by the DPDK community that prioritizes performance. After loading and binding it to the network card, the NIC becomes inaccessible to the rest of the system since the system's kernel no longer manages it [Group, 2017a]. In addition, the network card now interacts directly with DPDK. As a result, incoming network packets are delivered directly to user space.

DPDK uses multiple approaches for the efficient handling of packet processing. First, it preallocates packet buffers during initialization, avoiding costly memory allocation procedures during processing execution [Group, 2017c]. Also, the number of packet copying is reduced [Group, 2017b] since DPDK does not need to copy data between kernel and user memory, as it runs in user space. Moreover, DPDK has a simplified network stack, one that does not include unnecessary modules such as the implementation of layer 3 and 4 protocols. DPDK also retrieves packets from the network card in batches, meaning it can pull multiple packets from the NIC at once. In addition, as the network processing does not rely on the kernel [Group, 2017d], it moves away from complex operations like frequent mode switching and system calls.

However, removing the kernel also brings some concerns [Høiland-Jørgensen et al., 2018]. Without kernel involvement, applications must implement features already provided by the kernel, such as device drivers, layer 3 and 4 protocols, or routing tables. Also, DPDK, and kernel bypass solutions in general, tend to become isolated from the OS, preventing the use of standard tools like tcpdump, iproute2 or iptables. This happens because the network card and DPDK environment are separated from the operating system, as DPDK takes complete control of the NIC. Also, the OS cannot use the NIC, further separating the OS from DPDK.

Furthermore, the kernel provides security mechanisms to protect the system, like process isolation, protected memory region, or controlled hardware I/O. Bypassing the kernel means losing these mechanisms and imposing their reimplementation, bringing other concerns such as the reliability of such implementations.

2.2 XDP

XDP emerges in an attempt to create a high packet processing performance solution without sacrificing the kernel and therefore solving the kernel bypass concerns previously discussed in the case of DPDK. The leveraged in-kernel processing uses the extended Berkeley Packet Filter (eBPF) to implement packet processing without the need to modify kernel code [Vieira et al., 2020] [Monnet, 2016]. XDP creates a kernel hook that enables the user to create applications that run inside the kernel via the protected eBPF environment, that counts with security measures to protect the kernel, like program bounds, memory access, verified loops, and supported instruction calls [Rybczyńska, 2019]. This environment provides fast processing by running in an early stage of the kernel packet processing, right after the network card receives the packet even and before buffer allocation takes place [Høiland-Jørgensen et al., 2018] [Vieira et al., 2020].

As XDP runs inside the kernel, it maintains all previously mentioned standard security measures provided by Linux, such as protected memory region or controlled hardware I/O. Also, an XDP application can still interact with the kernel and use its internal functions like routing tables or TCP stack.

XDP, in its standard behavior explained above, is called the native or driver mode. However, XDP counts with two other additional modes, the generic and the offloaded modes [Cilium, 2022]. The generic mode performs a stack bypass on the kernel and is intended for experimental/testing environments where the network card driver does not support the native XDP mode. The offloaded mode enables the XDP application to execute in a programmable NIC, often called SmartNICs, enabling the host to free resources while the application runs inside the network card. In this mode, it is mandatory that the device driver support XDP and that the application does not rely on any kernel function or library since it will not be available when executed inside the NIC.

3. Related Word

Multiple works describe experiments to evaluate packet processor performance. This section introduces their main insights and obtained results. This step is necessary to position the current contribution midst of existing efforts.

The work described in [Gallenmüller et al., 2015] evaluates the performance of three packet processors that adopt the kernel bypass technique, DPDK, PF RING, and netmap. The experiments measure the achievable throughput by the frameworks using the packets per second (PPS) metric. The authors initially evaluate each framework's throughput when executing tasks with variable CPU load for each processed packet. Next, they examine the influence of batch sizes on throughput performance and per-packet latency. Results show that increasing CPU cycles decreases the throughput for all three considered frameworks. The netmap framework is the first to suffer from throughput loss. The authors also discover that the size of a batch of processed packets only leads to a gain in throughput when the processing task uses higher CPU cycles. However, if the CPU cycles per task become too high (in their experiments, around 400 cycles), the throughput drops regardless of the used batch size.

Regarding latency, batch size influence exhibits a trade-off. The larger a batch size, the lower packet latency is. This tendency remains valid up to the packet processing batch size of 256, beyond which latency increases. In other words, when processing packets in large batches, latency increases as the time a packet spends queued is higher since it has to wait for new packets to arrive and fill the batch size.

The research paper in [Høiland-Jørgensen et al., 2018], written by the developers of XDP, describes the framework's details, followed by a performance evaluation. The evaluation performs experiments to directly compare XDP and DPDK. The work compares the throughput, in terms of the PPS metric, of both frameworks without the presence of any processing task and an increasing number of CPU cores. DPDK overcomes XDP in all cases, even though XDP exhibits an increase in

performance with the increase of CPU cores. Note that this first set of conducted measurements does not include any host resource usage like CPU or memory. However, the authors measure host CPU usage when each framework actually processes packets. Results show that DPDK always consumes 100% of CPU while XDP scales accordingly to the offered packet load. DPDK CPU abuse comes from its constant polling for new packet queues as it avoids the performance overhead of interrupt processing. The work also measures latency, where XDP achieves a lower one than that of DPDK when forwarding to the same NIC, whereas it suffers a higher latency when forwarding to a different NIC.

The research study described in [Kourtis et al., 2015] focuses on the Network Function Virtualization (NFV) context. The experiments compare the performance of a DPI VNF with a normal DPI program executing in user space. The experiments alternate between the use of DPDK and a standard Linux kernel network stack. In other words, the evaluation focuses on comparing virtual network functions' performance with bare-metal network programs. Authors use DPDK to investigate if it can improve the VNF's performance as much as it improves the bare-metal based application. Results show that DPDK can achieve line-rate performance in a bare-metal environment and that when deployed in a virtual environment, it suffers approximately 19% throughput performance degradation.

The work in [Hohlfeld et al., 2019] focuses on evaluating XDP performance at different execution points, including at the user space with AF XDP, the XDP execution at the device driver, and during the special offloading to a programmable NICs. First, they evaluate processing tasks with no resource usage, similarly to [Høiland-Jørgensen et al., 2018]. The results show that in order to achieve line-rate throughput, XDP has to run across multiple cores or be offloaded to a programmable NICs. To put it more simply, user space XDP cannot achieve line-rate performance. They also perform experiments on a VM while setting performance observation points inside the virtual machine itself. These additional execution points are located: at the VM user space with AF XDP, at the VM device driver, at the host device driver, and the physical NIC. Results show that running XDP inside a VM presents similar results to when it is running in user space or over a device driver.

The work in [Scholz et al., 2018] evaluates packet filtering capabilities using XDP. The first evaluation uses a sample XDP firewall application that drops unwanted packets. Results show that XDP cannot achieve line-rate performance, reaching about half of the desired line-rate in their best-case scenario. They also measure CPU utilization, showing that when offering a 10Mpps incoming packet rate, XDP uses around 60% of CPU load only for packet processing with eBPF-related functions.

Recent research acknowledges the importance of modern packet processing frameworks. Our review of existing experiments identifies a lack of XDP and DPDK studies regarding the impact of virtualization in the context of a cloud computing scenario. Although the contribution in [Hohlfeld et al., 2019] conducts experiments on virtual machines, its results cannot be taken as typical of a working cloud computing environment, as they represent those for a single VM and present a point of view from inside the VM. Such experiments do not create an environment with multiple VMs, running multiple resource usage tasks as usually encountered in a real cloud computing environment. Also, most of the reviewed works do not use nor do they compare XDP and DPDK in the same experimental scenario.

4. Methodology

In order to perform the experiments with XDP and DPDK, it is first necessary to create a user program that interacts with the evaluated frameworks. Then, the experimental design can be built and the experiments executed. The present section details the way the experiments were setup and conducted.

4.1 The PPVE Architecture

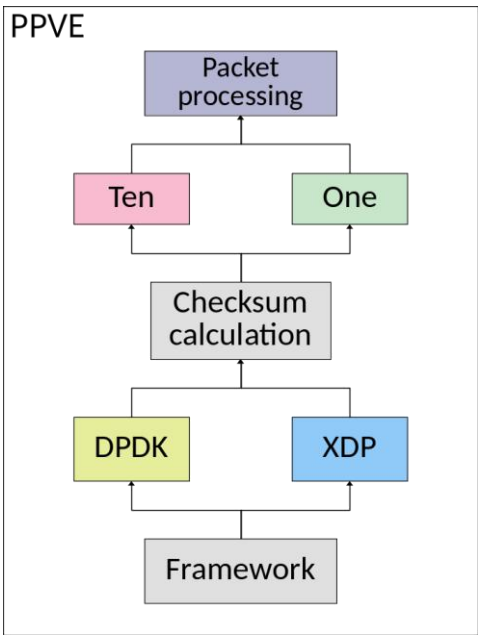
Given that this work focuses on a cloud computing data center scenario, the experiments should faithfully represent

typical applications encountered in such scenarios. Packet processors are packet I/O frameworks that exchange packets with the network card. A user program is necessary to interact with these packets and process them in a significant way. In a cloud computing scenario, this processing may occur in different ways. Typical packet processing involves, for example, DDoS protection [Miano et al., 2019] [Fabre, 2018], load balancing [Incubator, 2022], network management [Cilium, 2022b] or deep packet inspection [Van Tu et al., 2019]. In these cases, it is common to encounter packet processing frameworks running directly on the host, in a bare-metal fashion, while a virtual environment runs on the host, with multiple VMs performing different types of tasks. This is the setup selected for our evaluations.

Because of this, the first step taken to build the testbed was to develop a user space program that would interact with the two packet processing frameworks, namely, XDP and DPDK. This interaction should represent a task that typically occurs in both data center servers and standard packet processing applications. However, as this research aims to evaluate the behavior of these frameworks when inserted in such scenarios, the complexity of the application itself is not the main design goal as opposed to the impact it will have on a server's resources. For example, in a real data center environment, it is reasonable to assume that packet processors would perform resource-intense tasks like the ones already mentioned in [Miano et al., 2019] [Van Tu et al., 2019]. In other words, the application should also perform heavy-load tasks when performing the experiments in order to emulate such scenarios faithfully. One primary hardware resource is CPU, since it is present in almost every packet processing task [Hohlfeld et al., 2019] [Miano et al., 2019] [Van Tu et al., 2019] [Gallenmüller et al., 2015] and is also a vital resource used by any other task running on the operating system. In addition, tasks like DDoS protection, load balancing, or deep packet inspection usually do not interact directly with the virtual machines on the host. They instead act as "outside" network nodes that filter the packets that will be delivered to the VMs.

As a result, the development of a special architecture named the Packet Processing under Virtual Environments (PPVE) Architecture took place. It is a packet processing architecture that runs directly on the host and interacts with both DPDK and XDP to receive packets and perform simple packet filtering IPv4 checksum verification, simulating a firewall to the guest's VMs inside the host. Although checksum calculation does not model all types of CPU processing, it serves as a representative measure for CPU processing power usage and complexity [Hohlfeld et al., 2019] [Gallenmüller et al., 2015]. In essence, PPVE will calculate the IP checksum for every packet it receives. To further assess our heavy-loaded tasks scenario, we also configured PPVE to repeat the checksum calculation ten times. This is useful for representing different resource usage levels by the packet processing tasks and, consequently, providing better insights into how it may influence the framework's performance. Figure 1 illustrates the PPVE architecture.

Figure 1: The PPVE architecture.



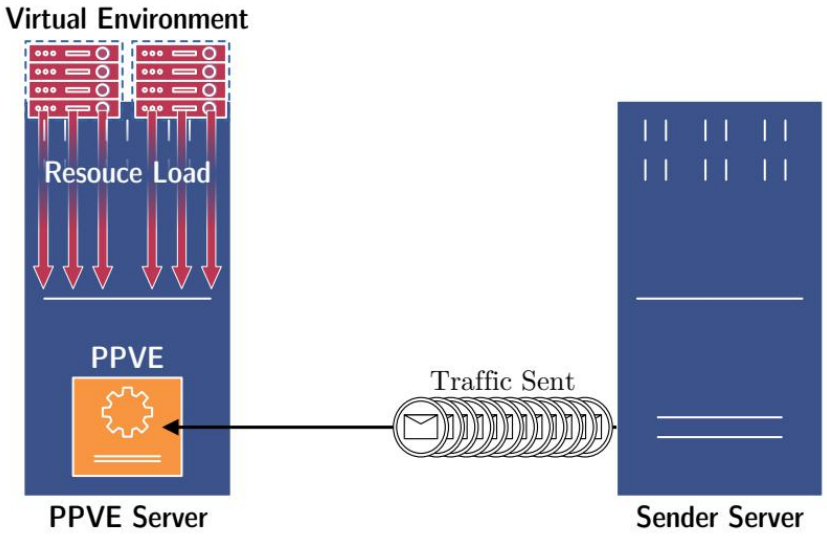
Source: Authors (2022).

4.2 Testbed and Experimental Design

The adopted testbed consists of two servers connected in a Peer-to-Peer (P2P) topology. The first server is called PPVE Server (PS) and represents the Device under Test (DuT). It is responsible for the execution of PPVE. The second server is the Sender Server (SS). It is responsible for sending network traffic to PS, which will be processed by PPVE. The testbed setup is based on previous well-established testbeds in cloud computing emulation [de Oliveira Filho et al., 2022] [Dantas et al., 2015].

The experiment initiates with the PPVE Server starting the virtual environment, with Kernel-based Virtual Machine (KVM) VMs. This environment simulates different data center contexts in-line with the scope of this work, such as database, storage, and VNF-hosting servers. Thus, this virtual environment executes a variable number of VMs, and each VM executes applications that generate some workload on the server. The applied workload varies between disk I/O, CPU, and network loads, selected to reflect the aforementioned data center context. The I/O and CPU workload is performed by the stress tool, while the network load consists of the ping tool sending echo packets of 1500 bytes at a 1Kpps rate through the virtual network card that connects to the NIC where PPVE is running. It is worth pointing out that, initially, the vNIC uses a standard VirtIO device driver connection to interact with the NIC. We intend to use other types of enhanced connections such as SR-IOV and PCI Passthrough in future evaluations. Figure 2 illustrates the adopted testbed.

Figure 2: Experiment's testbed.



Source: Authors (2022).

After setting up the virtual environment along with the workload inside each VM, PPVE starts and waits for packets to process. Initially, we execute PPVE in a single core since it is understandable that both XDP and DPDK should not scale too much on the host resources in such a cloud data center environment. Otherwise, they could compromise resources that are mainly directed to the VMs [Høiland-Jørgensen et al., 2018]. Next, the Sender Server starts sending network traffic to PPVE. To measure the number of packets processed by PPVE, we artificially alter the checksum field to force PPVE to drop every packet received and therefore measure the packet drop rate. This is because to drop a packet, PPVE must have finished calculating the checksum header and performing the comparison, meaning the packet has been entirely processed. The network traffic sent lasts 30s, sending IP packets at NIC line-rate (10Gbps), generated and sent by MoonGen [Emmerich et al., 2015]. Table 1 details the software versions used and the hardware configuration of the servers of the testbed.

Table 1: Testbed Hardware and Software Specification.

Component	Specification
Processor	Intel Xeon Bronze 3204 1.9GHz, 12 cores
RAM	64 GiB
HD	Dell BOSS VD
NIC	Intel Ethernet Controller X540-AT2 10GbE
OS	GNU/Linux Ubuntu 20.04.2 LTS
Kernel	Linux 5.4.0-70-generic
KVM	Version 2.11.1
DPDK	Version 20.11.1
MoonGen	git commit 525d991

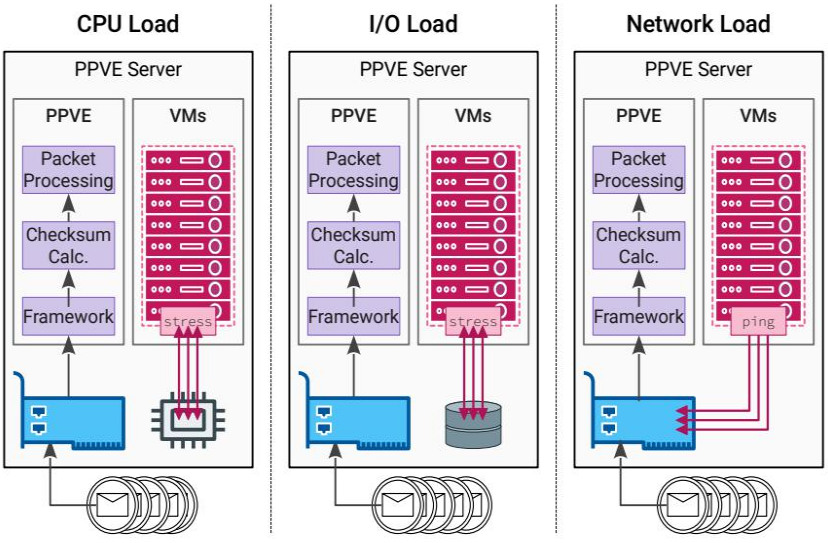
Source: Authors (2022).

The experimental design follows an experiment that compares DPDK and XDP's performance against adverse conditions. The experiment's factors are:

- Framework, which is the packet processing framework used, either XDP or DPDK. It is worth mentioning that XDP runs in native mode since our NIC does not support offloading. Also, we aim to measure the frameworks at their most standard and efficient operation.
- Checksum calculation, this is the number of checksum calculations performed by PPVE, being either one or ten.
- VM Load, the workload that the VMs executes, varying between CPU, I/O, and Network.
- VM Number, meaning the number of VMs running in the virtual environment and performing the workload. The values are either 0, 8, 32, or 128.
- Packet Size, the size of the packets sent by MoonGen to PPVE. This ranges during the experiments between 64 and 1500 Bytes. When evaluating packet processors, it is common practice in the literature to use minimum-sized packets in order to achieve the highest amount of processed packets possible by the framework, as seen in [Rizzo, 2012], [Høiland-Jørgensen et al., 2018], [Han et al., 2010] or [Jeong et al., 2014]. However, the goal of this work is to simulate a real cloud computing environment. Nonetheless, minimum-sized packets do not dominate such environments, so we also inserted larger size packets with 1500 Bytes, also typical of data centers.

Figure 3 summarizes the experiments with the multiple VM load scenarios.

Figure 3: Experiment's flow in its different scenarios.



Source: Authors (2022).

5. Results

In this Section, we present the results obtained from the experiments and divide them based on the response from each experiment: achieved throughput and packet loss.

5.1 Throughput Performance Evaluation

Following the experiment design described in the previous section, we conduct a performance evaluation of XDP and DPDK in our emulated cloud computing virtual environment. We start the evaluation by first measuring the throughput of

packets processed by each framework, according to the two factors: VM load type and packet size.

In order to obtain a bearing for the efficiency of the obtained results, one must keep in mind the line-rate metrics that our 10GbE network card can process. A 10GbE NIC refers to one that may theoretically process 10Gb per second. Nonetheless, the actually achieved line rate in packets per second highly depends on the incoming packet size. When testing with larger packet sizes, fewer packets per second will "fit" in our NIC. It is understandable that most NIC manufacturers state in the datasheets the achieved packet rate when actually using 64 bytes packets. We summarize the line-rate metrics in Table 2.

Table 2: Description of 10GbE line-rate in pps.

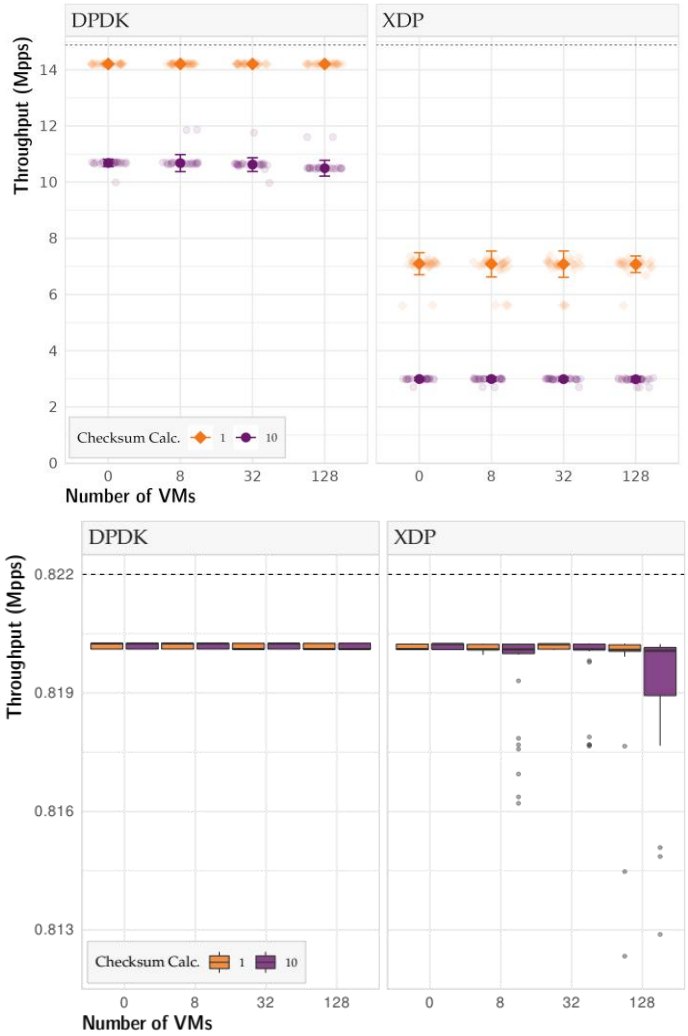
Packet size	Line-rate in packets per second
64 Bytes	14.880Mpps
1500 Bytes	0.822Mpps

Source: Authors (2022).

5.1.1 Influence of Cloud CPU usage

The CPU workload performed by the cloud computing VMs poses some minor influence on the throughput performance of XDP, when the framework has to process large packets of 1500 bytes, as seen in Figure 4b. In this scenario and with no VM running, the throughput is close to line-rate topping at around 0.820Mpps. As soon as the VM number starts to increase, initiating with 8VMs, the throughput begins to drop, while also exhibiting outliers. Nonetheless, this experienced throughput degradation is insignificant, as the median stays constant at 0.820Mpps. The worst-case scenario is when XDP processes ten checksum calculations and the virtual environment is concurrently running 128VMs. Here we observe a minimum throughput value of 0.807Mpps. However, even in this case, the variation is still minimal, with a standard deviation of only 0.004Mpps. Observe also that there is a different case with DPDK, that maintains network performance close to line-rate, regardless of the number of VMs and checksum calculations, presenting a standard deviation of 0.00007Mpps. The achieved optimal performance takes place mainly because 1500B packets create a low incoming packet rate, allowing the frameworks to process nearly every packet the NIC receives.

Figure 4: Graphics displays the scenarios when the VMs perform CPU load on the hosts. The dashed line indicates the NIC line-rate. Left: (a) Median of throughput with standard deviation and scatter plot in the background. Scenarios when the frameworks process 64B packets. Right: (b) Box plot of the throughput. Scenarios when the frameworks process 1500B packets.



Source: Authors (2022).

However, this behavior is different when the frameworks handle much smaller 64B packets, as displayed in Figure 4a. First, we can observe that the overall throughput performance of DPDK is higher than that of XDP. In the best-case scenario, in the presence of no virtual environment and while performing a single checksum calculation, DPDK presents a throughput median of 14.205Mpps, while XDP presents a lower 7.096Mpps. This represents around 50% less throughput performance than that of DPDK.

In addition, when processing minimum-sized packets, a combination of factors leads to intense performance degradation. First, the number of checksum calculations directly impacts the throughput of both frameworks. Increasing from 1 to 10 checksum calculation drops 24.8% of DPDK's performance, and 57.84% of XDP's. Combining this with the intense CPU usage from the cloud environment causes DPDK to further decrease its performance, reaching a throughput median of 10.496Mpps when running 128VMs in the virtual environment. To explain this effect, we recall that the main way that DPDK

works is by pinning a CPU core with busy polling and processing packets in this core [Group, 2017a]. When there is competition for CPU cycles and DPDK is especially performing CPU intense tasks, intuitively, this will cause performance degradation.

XDP does not suffer from this issue. As its performance when executing ten checksum calculations is already as low as 2.70Mpps, XDP does not seem to be affected by the presence of the cloud environment VMs. Its throughput median decreases from 2.991Mpps when having no VM to 2.984Mpps when competing with as much as 128VMs.

Takeaway: A cloud computing environment with predominantly high CPU usage that receives mainly big-sized packets causes little interference in the throughput performance of XDP and no interference on DPDK. Also, if the cloud environment receives mainly small packets, the framework's performance varies according to how resource-dependent the framework's processing task is, more than it depends on the amount of VMs the cloud environment is running.

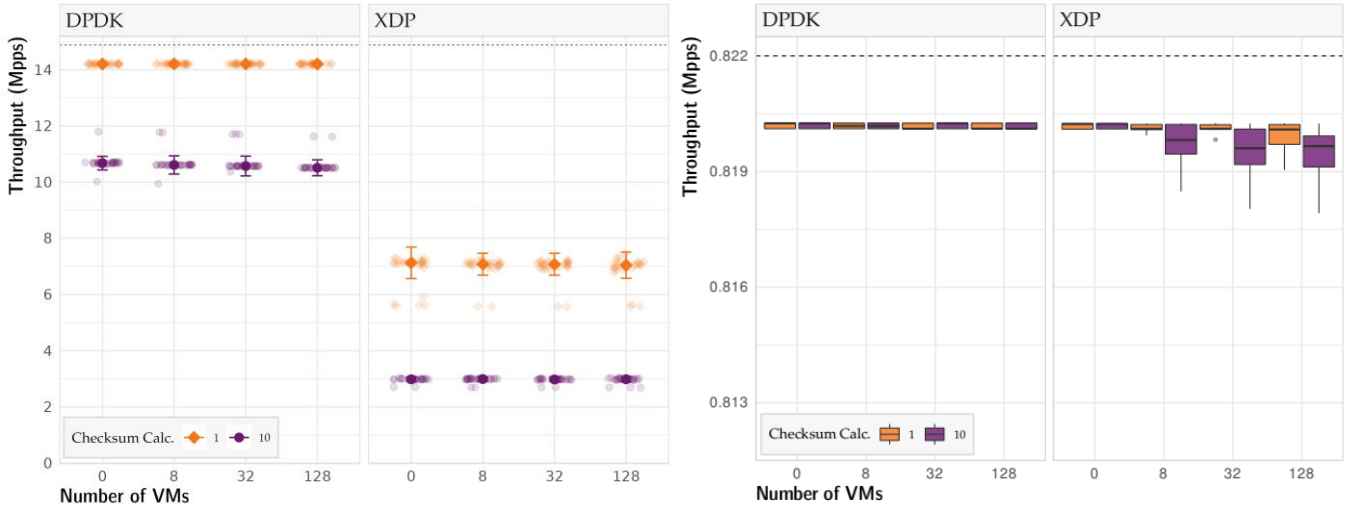
5.1.2 Influence of Cloud I/O usage

When the virtual environment executes I/O usage on the host, there is even less interference in the framework's performance. When receiving 64 bytes packets, see Figure 5a, DPDK still suffers a considerable performance degradation due to the number of checksum calculations, causing the throughput to drop and present higher variation. When performing ten checksum calculations, the throughput median drops from 10.676Mpps with no VM to 10.511Mpps when running as much as 128VMs. This represents approximately a mere 1.55% performance loss, which turns out to be not as significant as the one caused by the checksum calculation. This can be explained because the frameworks do not rely on disk I/O operations to process packets since they mainly work directly with the network card.

When processing big packets of 1500B, DPDK maintains its packet processing performance level with nearly no variation and near line-rate performance, as shown in Figure 5b. XDP indicates minimal interference from the virtual environment when performing ten checksum calculations. There is slightly more variation observed and a decrease from a throughput median of 0.8202Mpps with a standard deviation of 0.0001Mpps to a median of 0.8197Mpps with a standard deviation of 0.0006Mpps, when stepping from running 0 to 128VMs, respectively. This performance degradation is not significant in the overall scenario and processing performance, as it can be expressed as 0.06% throughput decay.

Takeaway: A cloud computing host with a virtual environment executing I/O intense tasks will cause minimal interference on the throughput performance of DPDK or XDP. In such an environment, the processing task that the framework executes is once again a more decisive aspect of the throughput performance. Also, DPDK with processing tasks with small CPU usage has near line-rate performance whereas XDP does not.

Figure 5: Graphics displays the scenarios when the VMs perform I/O load onthe hosts. The dashed line indicates the NIC line-rate. Left: (a) Median of throughput with standard deviation and scatter plot in the background. Scenarios when the frameworks process 64B packets. Right: (b) Box plot of the throughput. Scenarios when the frameworks process 1500B packets.



Source: Authors (2022).

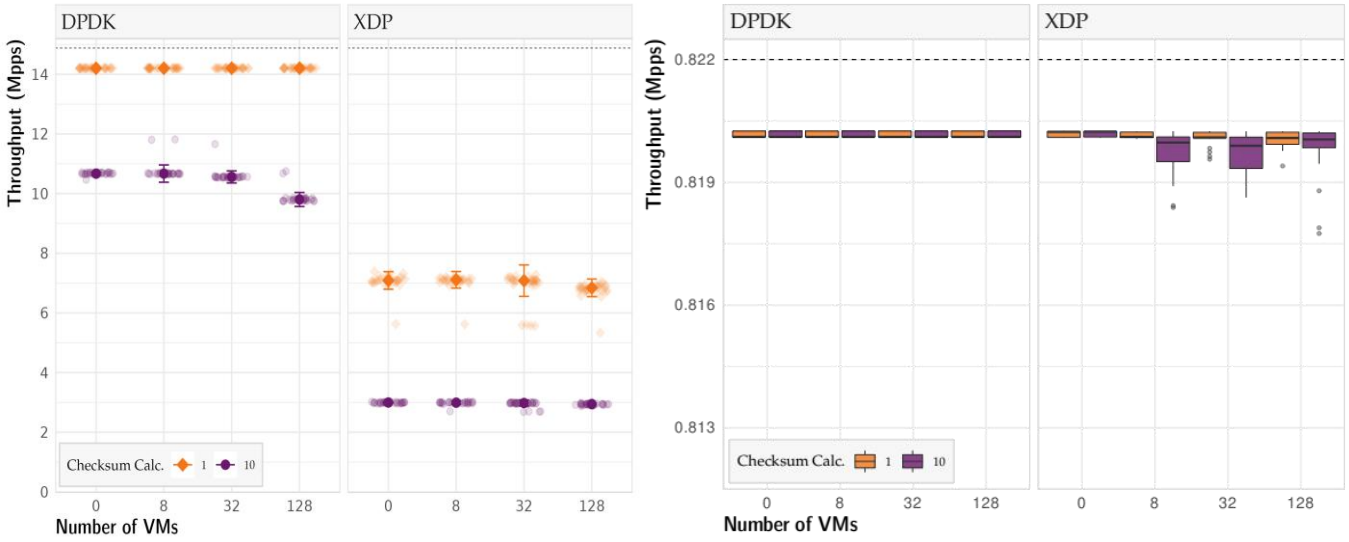
5.1.3 Influence of Cloud network usage

The intense execution of network load on the cloud computing environment affects the throughput performance of both frameworks when processing 64B packets. Figure 6a shows this scenario. When processing one checksum calculation, XDP already presents a non-optimal throughput performance with a median of 7.09Mpps. When introducing the cloud environment VMs, this throughput median drops to 6.84Mpps with 128VMs. When DPDK performs ten checksum calculations, this performance drop is more significant, with a decrease of 8.15% accumulated by the throughput median. This result is intuitive since the cloud computing environment is exchanging packets, and the frameworks are also receiving packets to process. The competition can overload the network card and create an overhead of queued packets to be processed, which delays its processing by the two tested technologies.

When the framework processes 1500B packets, DPDK still shows almost no variation, and the virtual environment does not affect its performance. As for the XDP framework, the VMs are responsible for a small throughput variation, making the throughput unstable and spreading the results when a higher number of VMs are present. Figure 6b demonstrates these results.

Takeaway: When cloud computing hosts are dominated by network usage from its VMs, and it receives mainly small packets, the throughput performance of the frameworks is compromised, causing its performance to drop. DPDK's performance still resists more when its processing tasks rely on a small CPU usage, but it fails to do so if tasks are CPU dependent. When the host receives mainly big-size packets, the virtual environment makes the throughput of XDP unstable, presenting more variation.

Figure 6: Graphics displays the scenarios when the VMs perform network load on the hosts. The dashed line indicates the NIC line-rate. Left: (a) Median of throughput with standard deviation and scatter plot in the background. Scenarios when the frameworks process 64B packets. Right: (b) Box plot of the throughput. Scenarios when the frameworks process 1500B packets.



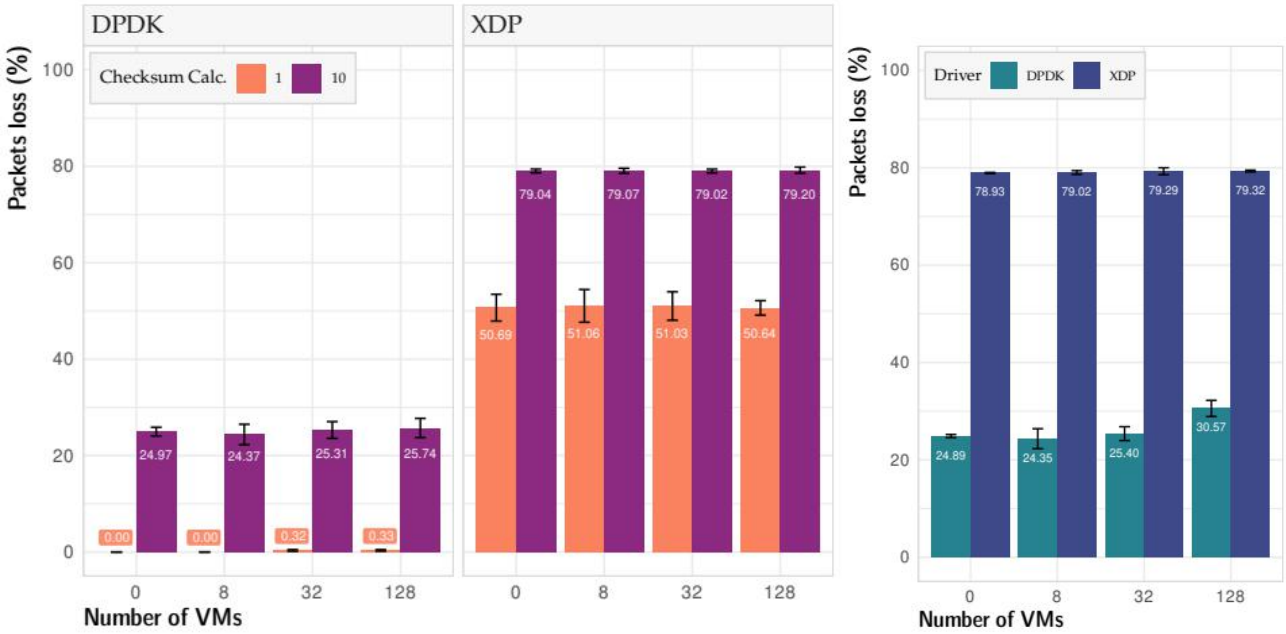
Source: Authors (2022).

5.2 Packet Loss Evaluation

We continue the performance evaluation by executing our testbed and measuring packet loss. The driver used by each framework has a buffer queue where it saves every new incoming packet to enter the processing stack later. This queue has limited space, and whenever it is full and has no empty slots left, it discards every new incoming packet, causing packet loss until new queue space is available again. We measured the number of packets that the NIC discards to evaluate the packet loss that each framework suffers due to its driver implementation.

Once again, we observe that the number of checksum calculations has a more substantial influence on the performance of both frameworks than the virtual environment itself, especially when the VMs execute CPU or I/O workload. Figure 7a displays the results when the VMs execute only CPU load. The results for I/O load are similar. It is possible to see that DPDK outperforms XDP, with almost 0% of packet loss while XDP has approximately half of its packets discarded when both run one checksum calculation. The difference is more significant when analyzing the cases with ten checksum calculations, where DPDK has a peek mean of 25.74% of packet loss and XDP has a much higher one at 79.20%.

Figure 7: Graphics displays the mean with the standard deviation range. Allcases when the frameworks process packets of 64 bytes. Left: (a) Scenarios when the VMs perform CPU Load on the Host. Right: (b) Scenarios when the VMs perform network Load on the Host.



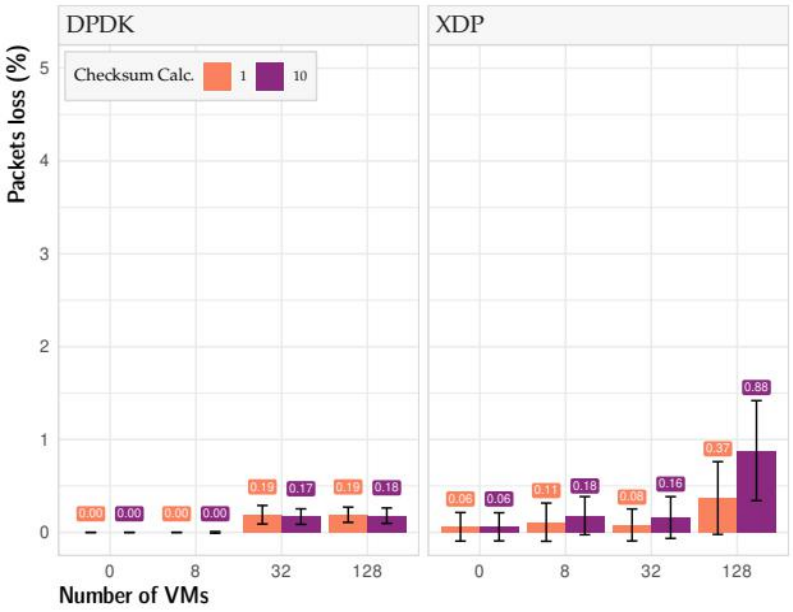
Source: Authors (2022).

This significant difference happens especially because XDP uses the standard Linux driver to process packets. These drivers have no optimization for packet processing, run inside limited kernel space, and can only retrieve one packet at a time from the NIC. On the other hand, DP-DK has a special driver that runs in user space with better memory management and can also rely on the support of the batch processing feature explained in Section 2.1. This feature allows DPDK to retrieve multiple packets at once from the NIC, thus preventing the network card from discarding more packets.

One particular case is when the virtual environment executes network load, illustrated in Figure 7b. In such a scenario, the cloud computing environment has a direct impact on the packet loss of DPDK, causing a packet loss increase of ≈6% when moving from running 0 to 128VMs. This result complements the ones seen in Section 5.3, where the throughput of DPDK would decrease when using as much as 128 competing VMs. As explained there, the NIC is overloaded with packets from the cloud environment and frameworks, queuing more packets and creating delays to process them – which decreases throughput – and leads to more packet losses.

When incoming packets are of size 1500B, the suffered packet loss is smaller. Intuitively, the use of larger packets results in a lower packet rate. With fewer packets per second, the driver's queue tends to fill less often. However, the frameworks are not yet free from packet loss. With big packets, the virtual environment has a direct – although smaller – impact on packet loss, as displayed in Figure 8. DPDK suffers no packet loss until a total of 32VMs is present in the virtual environment. A similar result is observed with XDP, going from 0.06% of packet loss with 0VM to 0.88% when running 128VMs.

Figure 8: Mean with the standard deviation range. Scenarios when the frameworks process 1500B packets and VMs perform CPU load on the Host.



Source: Authors (2022).

Takeaway: Packet size directly affects the packet loss on both frameworks, creating a higher loss rate when the packets are small. Also, XDP exhibits a significant amount of packet loss, mainly because of the standard Linux driver it uses. In addition, DPDK is directly impacted by the cloud computing environment when it performs network load. Finally, the VMs also has a direct impact on packet loss of both frameworks when receiving big packets.

6. Discussion

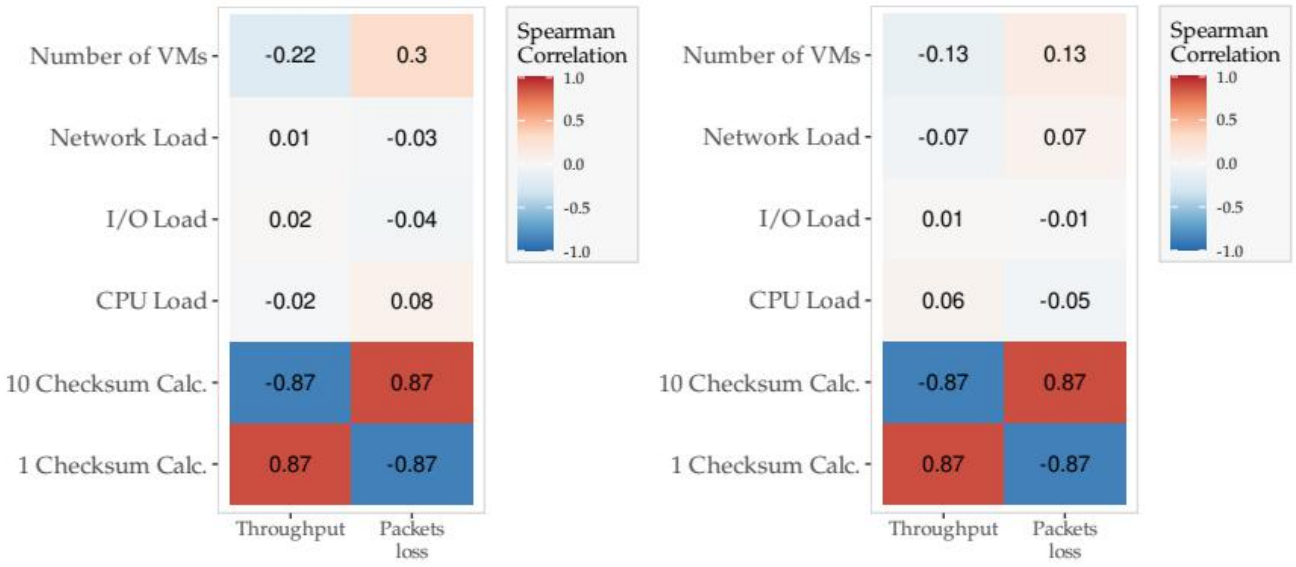
In order to strengthen the interpretation of the obtained results, this section discuss the results with some additional statistical results derived from the analysis of the established metrics.

The analysis was performed using the Spearman correlation rank, a non-parametric measure of the monotonicity of the relationship between two variables. This technique can be defined as follows: “Spearman rank-order correlation is a statistical procedure that is designed to measure the relationship between two variables on an ordinal measurement scale” [Corder and Foreman, 2011]. The correlation coefficient of Spearman ranges from -1 to 1, where 0 does not reflect a relationship, while -1 and +1 indicate a perfect relationship, positive correlations indicate a directly proportional relationship between the variables, and a negative correlation indicates an inversely proportional relationship between the variables. We use this technique as a statistical validation of the observed results seen in Sections 5. We perform the Spearman correlation rank for different scenarios, separating the scenarios based on the used framework and packet size. This separation is important to remove biased coefficients since the frameworks are two separate experiments and the packet size directly affects the achievable throughput of each framework.

Figure 9a and 9b shows the correlation when processing 64B packets using DPDK and XDP, respectively. We confirm the results seen before stating that the checksum calculation has a straight impact on the throughput and packet loss of both frameworks, with a Spearman coefficient of -0.87 and 0.87 for throughput and packet loss, respectively, when using ten checksum calculations. It also shows that the increasing number of VMs influences the throughput and packet loss, even

though the impact is not as great as the number of checksum calculations.

Figure 9: Spearman rank correlation coefficient heatmap. Left: (a) Scenarios with DPDK framework processing 64B packets. Left: (b) Scenarios with XDP framework processing 64B packets.

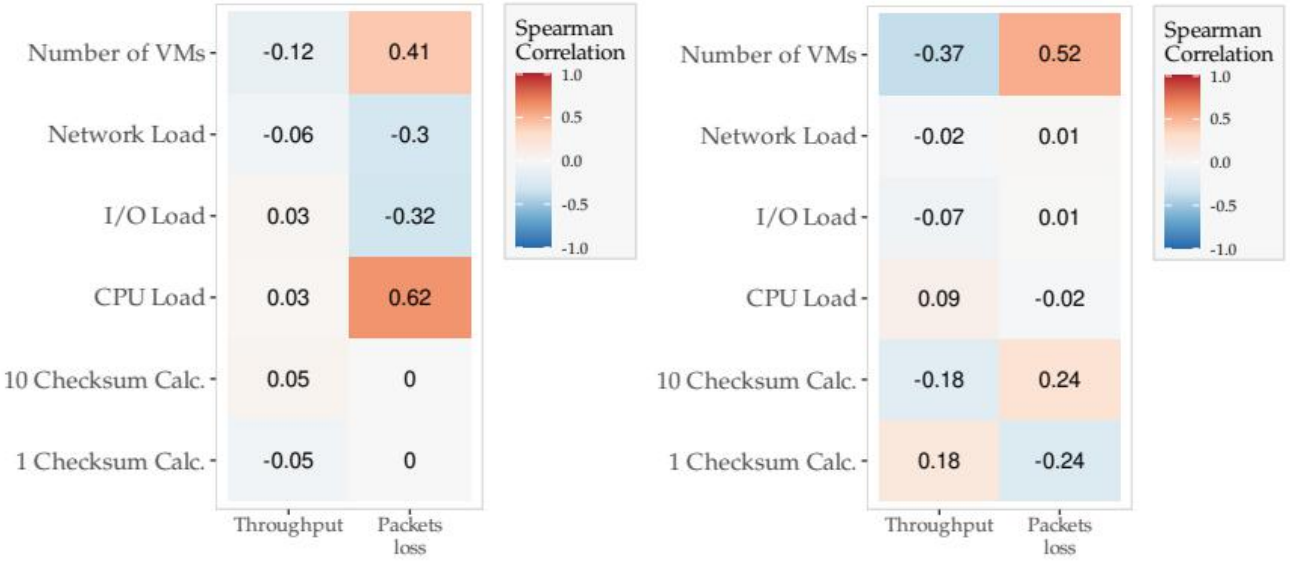


Source: Authors (2022).

Figure 10a and 10b shows the correlation when processing 1500B packets with DPDK and XDP, respectively. We see that for DPDK, there is no correlation between the number of checksum calculations and the packet loss, and almost no correlation is observed with throughput. This case confirms the conclusion we have seen before, but it brings an interesting outcome. It demonstrates the amount by which the virtual environment affects packet loss. Especially when the environment performs CPU usage, there is a 0.62 correlation with packet loss.

XDP on the other hand, has a slight correlation between the checksum calculation and the throughput and packet loss, showing that the number of checksum calculations has an influence on XDP's performance with big packets. However, the number of VMs has a more significant influence, especially when analyzing the packet loss.

Figure 10: Spearman rank correlation coefficient heatmap. Left: (a) Scenarios with DPDK framework processing 1500B packets. Left: (b) Scenarios with XDP framework processing 1500B packets.



Source: Authors (2022).

7. Conclusion

The presented framework gives the interested reader several takeaway ideas that the current literature lacks. One may decide to invest in a packet processing framework such as XDP or DPDK within a data center or cloud computing environment, thinking it is the right thing to do given both are state-of-the-art technologies. Our evaluations show that such a policy may lead to network performance loss both in throughput and packet loss. A more appropriate strategy should rely on understanding the behavior of both frameworks when operating under different conditions. We discovered through the conducted tests that opting for using a given framework depends on the operating environment for a server. To make an informed decision, one must consider important factors such as dominant packet sizes and VMs competing for CPU, I/O load, and network resources.

To summarize, there is no single configuration that fits all servers well. Administrators must first identify the kind of environment a server operates in and decide according to the takeaway lessons presented in this work. We plan to consider other packet processing frameworks that support NIC programmability and offloading and look at the impact of virtualization techniques on Linux packet processing.

In future works, we intend to explore the framework's performance with different response metrics, such as latency and resource usage. It also is intended to create different network functions besides packet filtering, such as forwarding.

Acknowledgments

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

References

Cilium. (2022). Bpf and xdp reference guide. https://docs.cilium.io/en/latest/bpf/

Incubator, F. (2022). Katran - a high performance layer 4 load balancer. https://github.com/facebookincubator/katran

Cilium. (2022). What is cilium? https://cilium.io/get-started/

Belay, A., Prekas, G., Klimovic, A., Grossman, S., Kozyrakis, C., & Bugnion, E. (2014). Ix: A protected dataplane operating system for high throughput and low latency. In Proceedings of the 11th USENIX Conference on Operating Systems Design and Imple-mentation, OSDI'14, page 49–65, USA. USENIX Association.

Bruijn, W. D. & Dumazet, E. (2017). sendmsg copy avoidance with msg zerocopy. In NetDev, The Technical Conference on Linux Networking, Netdev 2.1, Montreal, Canada. Netdev.

Corder, G. W. & Foreman, D. I. (2011). Nonparametric statistics for non-statisticians.

Dantas, R., Sadok, D., Flinta, C., & Johnsson, A. (2015). Kvm virtualization impact on active round-trip time measurements. In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 810–813.

Oliveira Filho, A. T., Freitas, E., do Carmo, P. R., Sadok, D. H., & Kelner, J. (2022). An experimental investigation of round-trip time and virtualization. Computer Communications, 184:73–85.

Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., & Carle, G. (2015). Moongen: A scriptable high-speed packet generator. In Proceedings of the 2015 Internet Measurement Conference, IMC '15, page 275–287, New York, NY, USA. Association for Computing Machinery.

Fabre, A. (2018). L4drop: Xdp ddos mitigations. https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations/

Foundation, L. (2018). Data plane development kit. https://www.dpdk.org/

Freitas, E. (2021). Experimental evaluation on packet processing frameworks under virtual environments. Master's thesis, Universidade Federal de Pernambuco, Recife.

Gallenmüller, S., Emmerich, P., Wohlfart, F., Raumer, D., and Carle, G. (2015). Comparison of frameworks for high-performance packet io. In Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '15, page 29–38, USA. IEEE Computer Society.

Group, D. P. D. K. (2017a). Dpdk docs: Poll mode driver. http://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html

Group, D. P. D. K. (2017b). Dpdk docs: Programmer's guide, mempool library. https://doc.dpdk.org/guides/prog_guide/mempool_lib.html

Group, D. P. D. K. (2017c). Dpdk docs: Programmer's guide, overview. http://doc.dpdk.org/guides/prog_guide/overview.html

Group, D. P. D. K. (2017d). Dpdk docs: Programmer's guide, ring library. https://doc.dpdk.org/guides/prog_guide/ring_lib.html

Han, S., Jang, K., Park, K., & Moon, S. (2010). Packetshader: A gpu-accelerated software router. In Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM '10, page 195–206, New York, NY, USA. Association for Computing Machinery.

Hohlfeld, O., Krude, J., Reelfs, J. H., R¨uth, J., and Wehrle, K. (2019). Demystifying the performance of xdp bpf. In 2019 IEEE Conference on Network Softwarization (NetSoft), pages 208–212.

-Jørgensen, T., Brouer, J. D., Borkmann, D., Fastabend, J., Herbert, T., Ahern, D., & Miller, D. (2018). The express data path: Fast programmable packet processing in the operating system kernel. In Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '18, page 54–66, New York, NY, USA. Association for Com-puting Machinery.

IDG (2020). Cloud computing survey. Executive summary, IDG. https://resources.foundryco.com/download/cloud-computing-executive-summary

Jeong, E., Wood, S., Jamshed, M., Jeong, H., Ihm, S., Han, D., & Park, K. (2014). mtcp: a highly scalable user-level TCP stack for multicore systems. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pages 489– 502, Seattle, WA. USENIX Association.

Kourtis, M.-A., Xilouris, G., Riccobene, V., Mc-Grath, M. J., Petralia, G., Koumaras, H., Gardikis, G., & Liberal, F. (2015). 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), pages 74–78.

Liu, J. (2010). 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), pages 1–12.

Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S., & Gupta, K. (2019). Offloading distributed applications onto smartnics using ipipe. In Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM '19, page 318–333, New York, NY, USA. Association for Computing Machinery.

Miano, S., Doriguzzi-Corin, R., Risso, F., Siracusa, D., & Sommese, R. (2019). Introducing smartnics in server-based data plane processing: The ddos mitigation use case. IEEE Access, 7:107161–107170.

Monnet, Q. (2016). Dive into bpf: a list of reading material. https://qmonnet.github.io/whirl-offload/2016/09/01/dive-into-bpf/#what-is-bpf

Rizzo, L. (2012). netmap: A novel framework for fast packet i/o. In 2012 USENIX Annual Technical Conference (USENIX ATC 12), pages 101–112, Boston, MA. USENIX Association.

Rybczyńska, M. (2019). Bounded loops in bpf for the 5.3 kernel. https://lwn.net/Articles/794934/

Scholz, D., Raumer, D., Emmerich, P., Kurtz, A., Lesiak, K., & Carle, G. (2018). Performance implications of packet filtering with linux ebpf. In2018 30th International Teletraffic Congress (ITC 30), volume 01, pages 209–217.

Soares, L. & Stumm, M. (2010). Flexsc: Flexible system call scheduling with exception-less system calls. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, page 33–46, USA. USENIX Association.

Surveys, W. W. W. W. T. (2021). Usage of operating systems broken down by data center providers. https://w3techs.com/technologies/cross/operating_system/data_center

Tsuna (2010). How long does it take to make a context switch? https://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html

Van Tu, N., Yoo, J.-H., & Hong, J. W.-K. (2019). evnf - hybrid virtual network functions with linux express data path. In 2019 20th Asia-Pacific Network Operations and Management Sym-posium (APNOMS), pages 1–6.

Vieira, M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Júnior, E. P. M. C., & Vieira, L. F. M. (2020). Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. ACM Comput. Surv., 53(1).

Wu, W., Crawford, M., & Bowden, M. (2007). The performance analysis of linux networking – packet receiving. Computer Communications, 30(5):1044 – 1057. Advances in Computer Commu-nications Networks.