Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 00 (2019) 1–13

Performance assessment of 40 Gbit/s off-the-shelf network cards for virtual network probes in 5G networks

Rafael Leira[a,b,1], Guillermo Julián-Moreno[b], Iván González[a,b], Francisco J. Gómez-Arribas[a,b], Jorge E. López de Vergara[a,b,*]

[a]Department of Electronics and Communications Technologies, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain.
[b]Naudit High Performance Computing and Networking, S.L., Spain

Abstract

Incoming 5G networks will evolve regarding how they operate due to the use of virtualization technologies. Network functions that are necessary for communication will be virtual and will run on top of commodity servers. Among these functions, it will be essential to deploy monitoring probes, which will provide information regarding how the network is behaving, which will be later analyzed for self-management purposes. However, to date, the network probes have needed to be physical to perform at link-rates in high-speed networks, and it is challenging to deploy them in virtual environments. Thus, it will be necessary to rely on bare-metal accelerators to deal with existing input/output (I/O) performance problems. Next, to control the costs of implementing these virtual network probes, our approach is to leverage the capabilities that current commercial off-the-shelf network cards provide for virtual environments. Specifically, to this end, we have implemented HPCAP40vf, which is a driver that is GPL-licensed and available for download, for network capture in virtual machines. This driver handles the communication with an Intel XL710 40 Gbit/s commercial network card to enable a network monitoring application run within a virtual machine. To store the captured traffic, we have relied on NVMe drives due to their high transference rate, as they are directly connected to the PCIe bus. We have assessed the performance of this approach and compared it with DPDK, in terms of both capturing and storing the network traffic by measuring the achieved data rates. The evaluation has taken into account two virtualization technologies, namely, *KVM* and *Docker*, and two access methods to the underlying hardware, namely, *VirtIO* and *PCI passthrough*. With this methodology, we have identified bottlenecks and determined the optimal solution in each case to reduce overheads due to virtualization. This approach can also be applied to the development of other performance-hungry virtual network functions. The obtained results demonstrate the feasibility of our proposed approach: when we correctly use the capabilities that current commercial network cards provide, our virtual network probe can monitor at 40 Gbit/s with full packet capture and storage and simultaneously track the traffic among other virtual network functions inside the host and with the external network.

Keywords: traffic capture and storage, virtual network function, virtual function, software-defined networking, HPCAP, HPCAP40vf, DPDK, PCI passthrough, VirtIO, KVM, Linux containers, Docker

1. Introduction

The incoming mobile 5G networks will be autonomic [1] and sliced [2]. In this context, these networks will have to implement an observe-analyze-act (OAA) loop [3] that enables them to be self-managed with little human intervention. The observation process involves checking the network health regularly by monitoring every network element and the traffic that is traversing the network. Monitoring can be carried out with passive or active probes; each has benefits and drawbacks. Active monitoring refers to injecting traffic into the network, which can be harmful in congested scenarios. In contrast, passive monitoring refers to capturing all the traffic and providing an exact picture of what is happening in the network at each moment, even in the worst-case scenarios when the network is heavily loaded, which are probably the most interesting scenarios from a network management point of view.

According to [4], future 5G networks will provide at least 20 Gbit/s downlink and 10 Gbit/s uplink per mobile base sta-

[*]Corresponding Author.

Email addresses: rafael.leira@naudit.es (Rafael Leira), guillermo.julian@naudit.es (Guillermo Julián-Moreno), ivan.gonzalez@uam.es (Iván González), francisco.gomez@uam.es (Francisco J. Gómez-Arribas), jorge.lopez_vergara@uam.es (Jorge E. López de Vergara)

[1]First author.

Received: 15th May 2018. Revised: 27th September 2018. Revised: 10th December 2018. Accepted: 25th January 2019.

The final publication is available at Elsevier via doi:10.1016/j.comnet.2019.01.033, to be published in *Computer Networks*.

tion. Therefore, to monitor the traffic passively in these nodes, it will be necessary to deploy at least 40 Gbit/s capture interfaces on every base station —if a standard Ethernet data rate is used—, to feed the OAA loop. Although this presents a substantial challenge, monitoring at 40 Gbit/s is only a step towards higher speeds, such as 100 Gbit/s or the recently approved 400 Gbit/s Ethernet standard [5].

Additionally, the slicing of 5G networks refers to resource virtualization, with concepts such as software-defined networking (SDN) and network function virtualization (NFV) [6]. In this context, the Broadband Forum has proposed a virtualized architecture: *Cloud Central Office* (CloudCO) [7]. In this approach, most of the communication network elements are encapsulated in CloudCO macro-nodes, which virtualize every necessary component. This proposed architecture, along with others, such as the architecture that is being developed in the H2020 METRO-HAUL project[2], tends to reduce overprovisioning and human interaction. This would lower the network costs in terms of both capital expenditures (CAPEX) and operational expenditures (OPEX), which is desirable for every network operator and Internet service provider.

To monitor future 5G networks, we will need to use such virtualization computing nodes. Thus, we introduce the concept of a virtual network probe (VNP): A VNP can be defined as a virtual network function (VNF) that is in charge of monitoring the traffic of both physical and virtual network elements. The term VNP represents a use case that aligns with in the monitoring as a service (MaaS) trend [8, 9]. Apart from all the advantages that NFV provides, VNP can solve some of the problems that are introduced by a pure virtualized network or data center, which may not appear in a classical full-bare-metal scenario of prior-generation networks.

So far, traditional monitoring deployments have had a very straightforward architecture: the physical probe is connected to a mirror port in the switch of interest for the traffic analysis and captures the network frames that are copied to that port. When virtualization enters the stage, there are additional possibilities, depending on what needs to be monitored and how.

A first example, which does not require a fully virtualized infrastructure, is a distributed environment with many servers and possible monitoring points. In this scenario, deploying physical probes to those monitoring points might be unfeasible or, at least, very expensive. However, if the infrastructure provides computing nodes, we have the option of deploying these VNPs on those nodes. Then, the traffic could be externally redirected to the computing node that contains the virtual probe to monitor the network without changing anything in the physical infrastructure.

This approach can be extended to scenarios that have an NFV infrastructure, where the traffic to monitor is transmitted among virtual machines (VM) via virtual functions (VF). Since these data do not leave the physical server that hosts the VMs, physical monitoring probes cannot monitor them. In contrast, those VFs can be configured to mirror their traffic to a virtual network probe that will be able to monitor and analyze these data.

Both cases are especially interesting for 5G mobile networks. As described above, the 5G network infrastructure will rely on virtualized computing nodes that will provide the necessary networking functions. Our VNP approach enables network operators to deploy, either manually or automatically, virtual machines that monitor the traffic of both physical and virtual network elements without any hardware modification, thereby reducing the costs and improving the reaction speeds to issues and changes in the network.

The main challenge in developing such passive VNPs is that traffic capture is a very performance-hungry function [10] because we must deal with tens of Gigabits per second and tens of millions of packets per second that must be passed to the VNP. Therefore, in this paper, we demonstrate how to deal with these input/output (I/O) performance problems by using and leveraging commercial off-the-shelf network card capabilities. Such hardware provides functionality that can be applied as bare-metal accelerators, thereby enabling the implementation of such VNPs with the performance of physical equipment.

In this work, we study a full packet capture and retention implementation at 40 Gbit/s under virtualized environments for passive network monitoring and investigate how we can reach this data rate without packet loss. To handle the specified data rate, we have developed HPCAP40vf, which is an evolved version of our custom Linux Kernel driver for network traffic capture that enables the development of VNPs that rely on available hardware capabilities. This driver is based on previous implementations [11, 12] and provides support for virtualization at 40 Gbit/s. We have also tested a VNP that is based on the Data Plane Development Kit (DPDK)[3] to compare our solution with a more generic solution. We have assessed the performance in both cases when the VNPs are deployed on a commercial server that is equipped with Xeon processors and an Intel XL710 40 Gbit/s network card and benchmarked the performance with previous physical developments.

With our approach, the use of a VNP is feasible for network traffic capture and retention, since the achieved performance is similar to that provided by physical network probes. In addition, these results are useful for other performance-hungry functions apart from network monitoring since it is possible to apply these ideas in the development of other virtual network functions. Moreover, all the developed code has been made open-source for the community and can be found on GitHub.[4]

The remainder of this article is structured as follows: First, we discuss the state-of-the-art packet capture and storage systems in Section 2. Then, we demonstrate various virtualization approaches for I/O devices in Section 3 and their associated bottlenecks. In Section 4, we explain the architecture of our driver and the techniques that are used to optimize the performance when building a custom VNP, while Section 5 provides the results of our tests. Finally, Section 6 presents this study's conclusions.

[2] `https://metro-haul.eu/`

[3] `https://www.dpdk.org/`

[4] `https://github.com/hpcn-uam/hpcap40g`

2. State of the art

One of the goals of the monitoring community over the past decade has been the development of dedicated equipment for servicing high-speed networks, where custom hardware [13] or field programmable gate arrays (FPGAs) [14] have been used to realize the required performance. However, this approach causes operators' facilities to be populated with a huge amount of heterogeneous hardware boxes, which complicates and raises the maintenance costs. The scalability of such infrastructure is limited, as it involves adding new hardware boxes when new technology appears or evolves, which also increases the physical space, cooling and power consumption requirements.

To minimize hardware costs, the first approach from the industry was to use commodity hardware. Although not necessarily inexpensive, those systems have a wide range of applications and are usually more affordable than specific-purpose systems. In this scenario, researchers have been working on the software that powers the monitoring systems and focusing on high-performance packet capture engines [15–19], subsequent analysis tools [12, 17] and solutions that address high-performance needs for network monitoring tasks, such as routing or classifying traffic on multi-Gbit/s links [20]. Furthermore, the availability of commodity hardware has enabled widespread experimentation and opened the field to new ideas. One of those ideas is the focus of this work: to virtualize the monitoring equipment.

Currently, despite its overhead, virtualization is no longer considered a low-performance option. Current virtualization engines such as Linux KVM, and XEN, combined with suitable hardware support, such as VT-d for Intel or Vi for AMD, reduce the load that is faced by the host operating system and bring these systems closer to the bare-metal level. The virtualization of hardware components also introduces new possibilities. The PCI passthrough technology assigns a PCI device to a guest VM, which gives it full and direct access to the PCI device. Although PCI passthrough opens very interesting scenarios in virtualized systems, it presents limitations, mainly because it is an exclusive assignment and only one VM can access each device at a time. For this reason, manufacturers made an extra effort to develop the concept of single-root input/output virtualization (SR-IOV), which is also known as virtual functions (VFs) —not to be confused with VNF.

A VF is a PCIe virtual device that is generated at the hardware level by the physical device. As the system considers this an almost fully independent PCIe device, it can be remapped to a VM, similar to any other physical PCIe device, thereby allowing a host to share a physical device with almost no overhead with many virtual machines at the same time. VFs are used not only in traditional VMs but also in lighter ones, such as Linux containers. Containers are a technology on which many virtualization tools, such as Docker, have appeared over the last years. The difference between them and a classical VM is that the VM has its own memory space, kernel and operating system, whereas in the container the kernel is shared between the host and the VM. This virtualization type has higher risks compared to the traditional VMs [21]; however, it performs better [22], as resources are easily shared with the host and a full operating system need not be run on top of the host. More details about virtual environments will be discussed in Section 3.

Virtualization is also changing the way we design, build and operate networks. SDN and NFV are examples of the new trend in which software can be finally decoupled from the hardware. An example is presented in [23], where authors discuss the usage scenarios for virtual switches that utilize physical and virtual network interfaces to quantify the throughput and latency of software switches. They focus on Open vSwitch (OvS) [24], which is an open-source implementation of a virtual multilayer switch that provides a switching stack for hardware virtualization environments.

Therefore, the existence of virtual networks and virtual network equipment requires new monitoring solutions. In recent years, the virtualization of monitoring network probes has attracted substantial attention. For instance, in patent [25], a *virtual* network probe is presented as a system for monitoring LTE networks, but with a slightly different meaning. It is *virtual* simply because the LTE operator does not know the details of how monitoring is taking place. ConMon [26] is an automated system for monitoring the network performance of container-based applications. In particular, it focuses on the effects of passive traffic observation on the performance of the application containers and on the system resources. EXFO's Virtual Probes[5] are solutions for deploying verifier virtualized network functions across any SDN/NFV network to perform testing and troubleshooting. A similar idea is presented in [27], where authors design a new Open vSwitch vProbe for troubleshooting performance issues based on the newly proposed IETF network service header (NSH) with monitoring extensions [28]. Although these works deal with virtual probes, they do not present a probe that can capture and store traffic, as we are presenting here.

The vProbe concept appears again in Qosmos's white paper [29] and, in this case, the authors introduce one of the key ideas that we are presenting in this article: a method for monitoring inter-NFV traffic. The white paper does not include any implementation details; however, a DPI product that is based on this technology is able to analyze networks at speeds of up 10 Gbit/s, depending on the configuration. Qosmos's vProbe architecture demonstrates that monitoring software runs at the virtualization-layer level. The main disadvantage of this approach is that the monitoring capabilities are limited by the performance of the hypervisor, as it is difficult to capture data at high speed, which must be at least 30 Gbit/s for 5G networks, as discussed earlier. A similar performance limitation in network virtualization is discussed in [30], where DPDK is used to increase the throughput of Open vSwitch. Specialized hardware, such as FPGAs [31], could also be used to increase the performance of virtual networks, although the issues regarding cost and scalability that discussed previously would remain in this scenario.

In our solution, we use the network card as a bare-metal accelerator for packet capture and monitoring and obtain the high-

[5] `https://www.exfo.com/en/products/network-performance-monitoring/virtual-probes/`

est possible performance with the best performance/price ratio. Our VNP offers the same functionality for monitoring with the additional capability of capturing and storing all network data. The concept of VNP fits into the 5G network philosophy and with the OAA loop and takes into account that the passive monitoring process may not depend on the underlying network deployment or its virtualization technology.

3. Virtualized environments and I/O processing

The use of virtual machines imposes a computational overhead on any application that is executed on top of them. The applications that can be substantially affected are the computing-intensive applications and those with high I/O data transfer, such as the NFV. For example, an application that is processing the traffic from a 40 GbE link may have to process up to 59.52 Mpps in the worst-case scenario. This implies extremely low-latency memory accesses and high memory bandwidth, where even the shortest pauses can have a significant impact on packet loss (for example, in a Skylake processor at 3.40 GHz, an L1 cache hit represents 7% of the available time for processing a packet in the worst case). There are many virtualization methods, each with advantages and drawbacks and various degrees of isolation and performance. Since 5G networks would not be limited nor restricted to a virtual environment, we are going to explain the possible virtual environments and the most popular hypervisors.

Virtualization can be divided into 3 components: the host, the hypervisor, and the guest. The host speaks with the real hardware; the hypervisor creates a virtual environment from the real hardware; and the guest uses it. This type of hypervisor has a substantial isolation between the host and the guest and may make guest functionality independent from the underlying hardware. However, this method usually has overheads, such as double copies, etc. The hypervisor concept has changed since the Linux containers appeared [32], where the host kernel acts as a lighter and simpler hypervisor. A Linux container is a closed environment inside a Linux host, where all the processes share the host kernel but each can only interact with its own filesystem and associated —possibly virtual— hardware. This implies that Linux containers can run Linux over Linux and isolation issues may be encountered, such as a critical kernel error that is caused in a container and affects every other running container [21].

Depending on the selected virtualization technology, performance tuning must take place. In terms of classical virtualization, the kernel virtual machine (KVM) [33] is the most frequently used and popular virtualization technology and slightly outperforms other solutions [34]. Several large companies, such as Google and Amazon[6], offer its virtualization services over KVM. In terms of virtualization that is based on containers, Docker[7] is probably the most well-known and flexible solution [21]. Throughout this article, we are going to focus on these two solutions, namely, KVM and Docker, as representatives of their technologies.

The two principal optimizations that must be performed in virtual environments are the CPU and memory allocations. When the physical system has more than one CPU, this must be communicated to the corresponding VM or container. If the tuning is omitted, the VM will have suboptimal resource usage, with many communication between the CPUs, which results in a memory latency increase, and performance degradation. There is also an issue regarding memory allocation that may concern only classical VMs: In classical VMs, everything is virtualized, including memory. This implies that the VM memory would be divided into pages; however, the guest would divide it again into different pages. To mitigate this duplicate memory page level, we should map the VM memory into *hugepages* [35]. Finally, if the host hardware configuration supports it, as in our case, virtualization enhancement instructions should be enabled.

Due to the importance of I/O issues for network monitoring tasks, the remainder of this section discusses diverse alternatives for mapping I/O devices into virtualized environments, along with their advantages and drawbacks. All the scenarios are illustrated in Figure 1.

3.1. Full-virtualization

The full-virtualization paradigm is produced when the guest cannot distinguish whether it is running in a virtualized environment or not, as every available hardware looks similar to the physical one. That includes its identifiers, registers, and every low-level characteristic. This paradigm cannot be applied to Linux containers because every process knows that it is running in a virtual environment and most of the available hardware is real hardware or paravirtualized hardware (see Subsection 3.2). This type of pure-virtualization is the most theoretically desirable because the guest will not be dependent on the chosen hypervisor or the hardware differences among machines, thereby enabling the uniformity among all deployed virtual machines.

In the practical case of network full virtualization, a large performance drawback is encountered. The incoming packets are received by the physical NIC driver that is running on the physical server and traverse the system host network stack before being delivered to the hypervisor network module. Once acquired by the hypervisor, packets are delivered to the target VM depending on the virtual network configuration. This data path requires at least two additional copies, namely, from the host to the hypervisor and from the hypervisor to the virtual machine, which substantially degrades the performance. The performance degradation that is experienced by network applications that are running in this configuration has motivated researchers in academia and industry to tune and optimize the hypervisor packet handling policy. By using a "paravirtualized" `e1000` network card (an Intel Gigabit Ethernet network card), the authors of [36] proposed an approach for increasing the system throughput from 300 Mbit/s using the conventional approach to nearly 1 Gbit/s using a VALE software switch in the worst-case scenario (64-byte UDP packets); the results ranged

[6] `https://aws.amazon.com/ec2/faqs/?nc1=h_ls`

[7] `https://www.docker.com/`

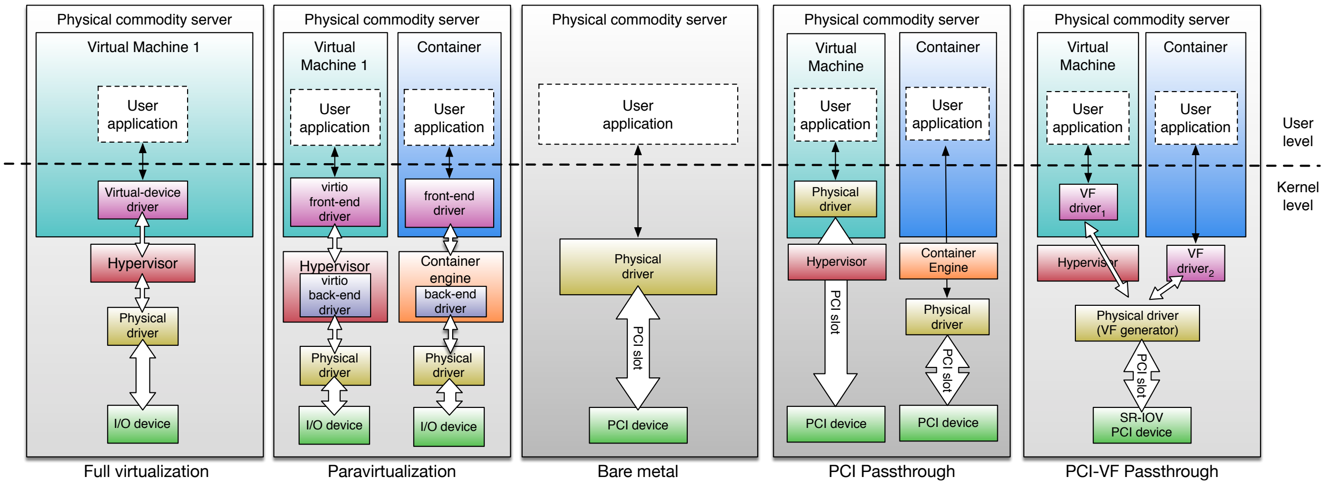

Fig. 1. Different ways of relating an I/O device with a VM or a Linux container.

from 2.7 to 4.4 Gbit/s when transmitting TCP traffic between VMs in the same physical server.

When using a fully virtualized I/O device, the hypervisor is the central communication element. Consequently, the hypervisor becomes the bottleneck and a single point of failure for network processing tasks, thereby limiting their applicability for I/O-intensive applications.

3.2. Paravirtualization and VirtIO

Paravirtualization emerged as an alternative to the full-virtualization approach, with the difference that the guest knows that the devices are virtual. Therefore, a driver that is compatible with the virtual device must be instantiated in the guest. This causes a dependency on the hypervisor technology; however, it provides a fast data-path between the host and the guest, with better performance and lower overheads that are due to simulating unnecessary low-level hardware details. VirtIO [37] has emerged as the *de facto* standard for this communication between the guest and a KVM hypervisor. It implements a flexible API for this communication via a set of queue structures and callback function handlers. VirtIO supports various types of I/O devices, such as block, console and PCI devices. In the last decade, network device support has been added [38]. Its performance has been compared in a previously cited work [36], where VirtIO could reach 4 Gbit/s with the VALE switch. A possible alternative to VirtIO is the NetVM module [39]. This work obtains promising results when performing packet forwarding between virtual machines that are instantiated in the same physical server (up to 34 Gbit/s); however, it does not deal with the problem of capturing the traffic, neither internal nor external.

The Linux containers have much simpler paravirtualized devices. Network interfaces that are inside the container are typically provided by network bridges or by the macvtap module, which are both generated from physical devices.

Devices of this type are similar to those that are created natively by Open vSwitch [24] and can provide an internal bandwidth that is closer to the memory bandwidth of the host [23]. However, when a mirroring port is active, the classic OvS will forward the packet into a user space application and copy it twice into the destination interface and the monitoring interface. In terms of efficiency, the final delivery of the packet requires up to five copies: first, from the service to the host interface; second, to the OvS user space management process; third, again to the host kernel space; and finally, fourth and fifth copies to the legacy recipient and the monitoring agent. Even if the memory bandwidth is huge, these copies degrade the overall performance. There are several similarities in how VirtIO and containers work. In the case of storage devices, VirtIO maps a Linux file into the VM space and containers carry out this mechanism for every shared device between guest and host.

3.3. PCI passthrough

Most important microprocessor vendors, such as Intel, AMD and ARM, implement I/O memory management units and a set of techniques for I/O management. The name of these techniques may vary from one vendor to other (VT-d for Intel *vs.* Vi for AMD). Those features theoretically supply the protection and support that are required for virtual machines to map safely into their memory space physical or virtual PCI Express devices. This technology is called *PCI passthrough*. By using PCI passthrough, the access from the VM to the device incurs minimal overhead as all intermediate virtualization layers disappear. In this configuration, the VM views the devices as if they were physically connected, which implies that the used driver must be the same one that manages such devices in a bare-metal configuration. Consequently, this allows network applications that are being executed in virtual machines to benefit from the high-performance packet capture solutions that have been developed for bare-metal scenarios. PCI passthrough has been successfully applied in high-performance computing scenarios [12, 40].

The concept of passthrough is simpler in the container approach than in any other case. It can be implemented by mapping the correct control character devices to the container filesystem. The driver that would use the physical device must be running in the host, as the container cannot load any kernel module for security reasons. Therefore, the performance difference between the host and the container should be small, as we will explain in detail in Section 5.

3.4. PCI virtual functions

Using PCI passthrough for mapping I/O devices to VMs has an inherent constraint: only one VM can make use of each mapped I/O device. Thus, PCI passthrough presents a scalability problem when the number of VMs that are in use is increased, which can only be solved by adding additional PCI devices, the number of which is also limited, as the number of PCIe slots is limited by hardware. With the goal of promoting virtualization performance and interoperability, the PCI Special Interest Group developed a series of standards: Single-Root I/O Virtualization (SR-IOV)[8]. These standards use the term *physical function* (PF) to refer to a PCI device that is connected to a physical PCI slot. They also introduce the concept of *virtual function* (VF) as a way for PCI devices to offer a lightweight register interface for managing the data that are interchanged with the physical PCI device. A VF is a lightweight PCI device, that is, it has an I/O memory area that is assigned to it with a set of control-related registers that allow the end system to use the VF as a regular PCI device, typically with reduced functionality. Importantly, the driver that manages this new virtual device is typically different from the one that manages the corresponding physical device, as it must be aware of the peculiarities that the virtual device presents. After creating the corresponding VF, they could be mapped to a VM via PCI passthrough as if they were purely physical devices. A relevant advantage of VF over PF is that, depending on the underlying hardware, a single PF can produce several VFs. This behavior allows the system manager to solve the scalability issue by attaching new virtual machines to new VFs without needing to increase the amount of hardware in the system.

Most current network devices support SR-IOV. As our target is at least 40 Gbit/s, we have focused on the Intel XL700 NIC family. Those NICs refer to their VFs by introducing the concept of virtual machine device queue (VMDq) and have one-to-one correspondences with the instantiated VFs. The number of virtual functions that are generated per physical device is limited by the hardware and is 128 virtual functions in the case of the Intel XL710 adapter. As discussed above, only VF-aware drivers can be used to manage the virtual NIC, which limits the number of available traffic capture engines.

DPDK has native support for working with VF and Intel also supplies a VF-aware counterpart to the i40e driver that is used by the XL710 NIC, which is named i40evf. Additionally, we developed a VF-aware version of HPCAP, which we named HPCAP40vf, by following all the design principles that have guided HPCAP conception [41]. We have tested those three drivers for use with VFs because, to the best of our knowledge, no other drivers can capture traffic from an XL710 virtual NIC. The native driver that is running on the host controls VF generation and its basic configuration. In a previous work with a 10 Gbit/s Intel card [12], we discovered a relation between the generation driver and the final performance. These differences were due to the way in which the old 10 Gbit/s card manages the packets internally, which is improved in the newer Intel cards, such as the one that is studied here. This XL710 card fully implements a switch internally; hence, we have not experienced any performance differentiation between the generation drivers, because the VF generation consists only of configuring the internal switch.

For security reasons, the VF default traffic distribution policy is based on the MAC and IP addresses that are associated with the VF. Thus, each VF only receives the traffic that is targeted to its corresponding VM. This limitation is necessary in most cases; however, it is problematic when our objective is to listen to the traffic of other VFs or all the traffic that is received by the card. Through a set of hardware registers —or with the latest i40 driver, which has an undocumented control interface in the debugfs— the system manager can insert a mirroring rule into the internal card switch to distribute or replicate the incoming traffic to each VF. Enabling this feature in the Intel XL710 NIC yields a hardware-level packet replication, which minimizes the impact on the capture process performance. Nevertheless, the total throughput is limited by both the performance of the XL710 internal switch and the PCI interface bandwidth. This is an important issue because it can affect the normal operation of the whole system since each incoming packet must be transferred twice in the PCIe bus.

4. VNP Architecture

As we stated in the introduction, traditional monitoring passive probes are connected to a mirror port of a switch to receive the selected traffic (see Figure 3a). While this approach should yield the best performance, because of the use of dedicated hardware, there are several scenarios in which it cannot be used. For example, when there are multiple monitoring points, deploying several physical probes might be unfeasible or expensive. In those cases, a more desirable alternative is to use the available hardware to execute the monitoring tasks. We propose a virtual network probe (VNP), which can be deployed onto any computing node in the data center or network infrastructure that can run virtual machines. In this section, we will describe the driver that provides the high-speed capture of network traffic and the possible configurations for the deployment of the probe, depending on the monitoring requirements.

4.1. HPCAP40vf: A 40 Gbps capture driver

In most typical environments, the approach that NICs use when receiving traffic is to write the incoming packets in the reception (RX) ring in host memory via DMA. Then, the operating system processes those packets and sends them to the corresponding application. However, the performance of this approach is far from sufficient when the only purpose of the system is to capture traffic at high speeds.

[8] `http://pcisig.com/specifications/iov/`

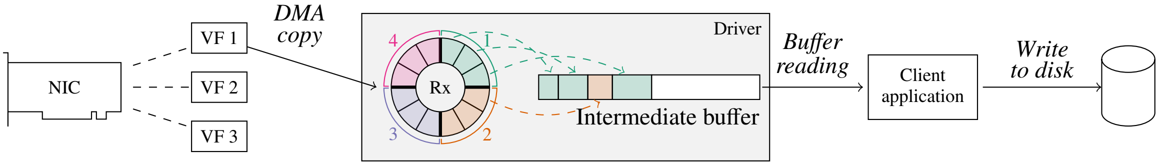

Fig. 2. Architecture of the HPCAP40vf driver.

The typical solution in high-speed network monitoring has been to develop custom capture drivers that bypass the system and provide direct access to the incoming traffic [42]. For 10 GbE networks, single-threaded capture drivers have been capable of receiving traffic at line rate; however, with faster standards, parallelism becomes necessary and the complexity of the capture systems increases. The driver that powers our VNP, namely, HPCAP40vf, is based on the same architecture that proved capable of capturing and storing traffic in 40 GbE networks (HPCAP40), which was described in detail in a previous paper [11] — here, we will only discuss the main ideas behind that architecture.

As discussed before, the main focus of our driver is to store the traffic to disk for subsequent analysis. Since the optimal write performance is achieved with fewer transferences of large blocks of data instead of many small blocks of data (as is the case with network packets), we implement a one-copy approach: the driver copies the frames to a contiguous zone of memory, which is later written to disk in large blocks, thereby achieving optimal speeds when writing to disk. While this solution has proven useful in our capture drivers for 10 GbE networks [41], it is essentially single-threaded and the translation to a parallel environment is not straightforward.

In the context of network drivers, parallelism is implemented in the reception side with multiple receive-side scaling (RSS) queues, where the NIC copies the frames to different reception (RX) rings via DMA, depending on their destination. Then, each ring is assigned to a different CPU so that incoming traffic is processed in parallel. However, for our capture driver, RSS is not the most appropriate solution. For example, the traffic can be distributed unevenly among queues (the NIC assigns packets from the same network flow —packets that share source and destination IP addresses and ports— to the same queue and imbalances can easily appear), thereby causing packet losses that depend on the traffic profile. Additionally, the frames become disordered (it is impossible to know which frame came before if we see the frames in different queues at similar times) and the precision of the timestamping is decreased, thereby causing issues in the subsequent analysis of the traffic.

In our driver, we do not use RSS queues; instead, we use a single RX ring. This ring is divided in fixed-size segments and each thread is assigned to one segment, as illustrated in Figure 2. No synchronization mechanism is needed, as every thread knows exactly its assigned descriptors, and the process becomes cache-friendly with a predictable latency and throughput. Furthermore, disorder of the frames can only occur on the boundary of each sector, when two threads are copying their corresponding frames at the same time without respecting their relative order.

Each thread polls continuously the first descriptor in its ring segment where the NIC will copy the packets. Once the NIC finishes the copy, it signals that the descriptor is available to be read. The thread timestamps the packet, copies it to the intermediate buffer, and marks the read descriptors as available for the NIC to write again. The threads for the copy to the intermediate buffer are synchronized by following the model of a multiple reader - multiple writer queue but taking into account the predictable behavior of our capture driver, which enables the use of several shortcuts and optimizations that improve the performance. After the write is completed, client applications can read from that buffer, either in a packet-by-packet fashion for analysis or, as described above, in fixed-size blocks that are written to storage media.

We also provide filtering and truncation features to reduce the amount of data that is passed to the client applications. This can be useful in scenarios where the storage is limited, in terms of either speed or capacity, or where part of the traffic is known to be unnecessary for the analysis and discarding it improves the efficiency.

4.2. Deployment of the VNP

The VNP that is proposed here consists of two components: the capture driver and a tool that processes the data. In a production environment, the tool could be similar to Naudit DetectPro [42], which generates flow records for real-time monitoring and storage of the traffic for a later and more detailed analysis of the full traffic or only intervals of interest. However, other software can be used to process the data, such as specific-purpose monitoring applications or, as in the case of the tests that are performed for this paper, a program that writes data to disk while measuring the throughput. Similarly, the capture driver can also be changed: although we propose the use of HPCAP40vf to achieve optimal performance when writing to storage media, alternatives such as DPDK could be used instead.

Both components are packaged in the virtual machine or container, which can be later deployed directly to computing nodes. Those nodes only need to assign a VF to the VNP, which will start monitoring the traffic that passes through the physical network. This scenario is illustrated in Figure 3b. If not only physical networks but also virtual ones require monitoring, the configuration would be as shown in Figure 3c. The virtual network is provided by the VFs that are generated by the NIC with an internal mirror configured so that the traffic from all virtual machines is captured by our VNP.

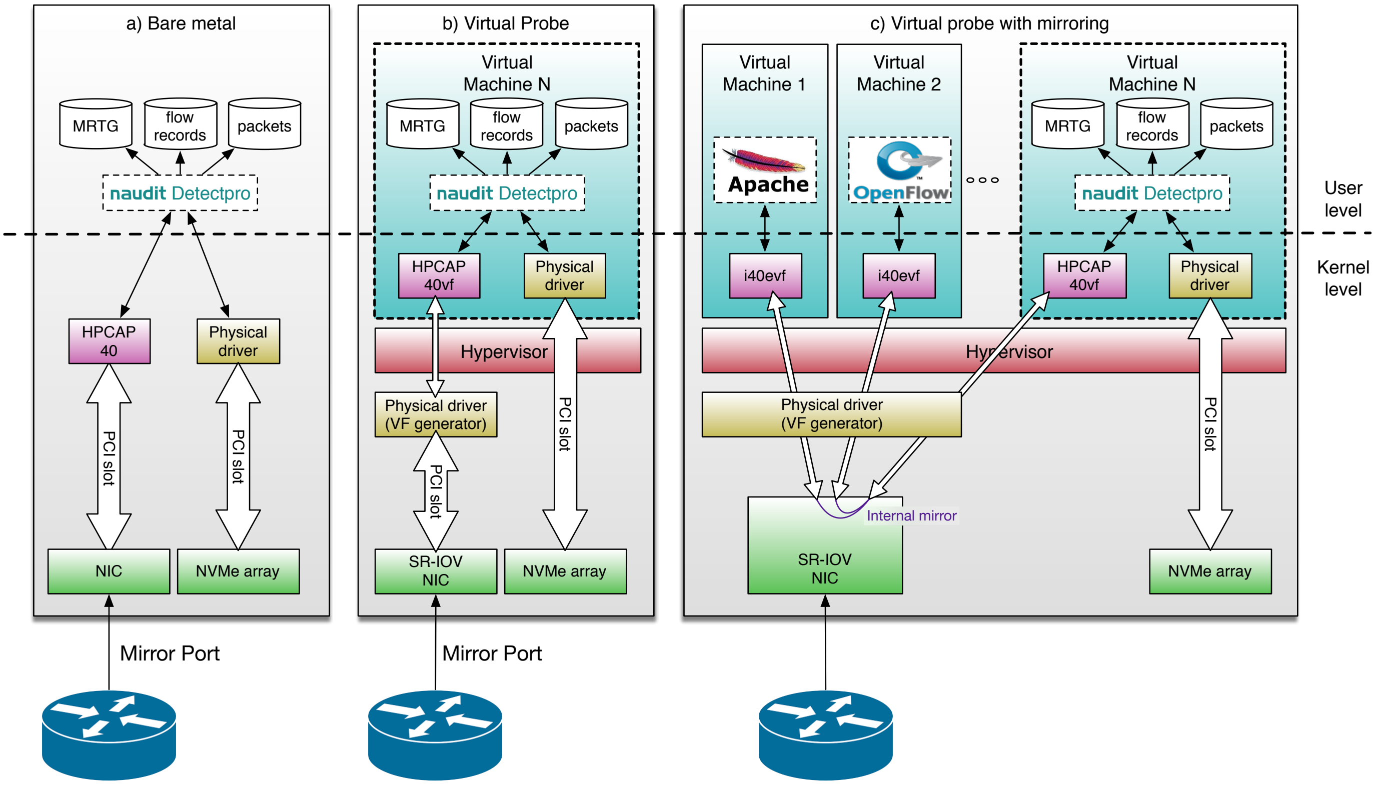

Fig. 3. Example of a deployment of network probes. a) Bare metal, b) a virtual probe and c) a virtual probe with mirroring.

The storage media is, in all cases, a physical NVMe drive array that is capable of providing both the required capacity and speed. This space can be linked to the VNP via the techniques that were described in the previous section: a bare-metal configuration in which the disks are mounted in the host machine, with PCI passthrough, or with VirtIO. The final choice will depend on the achieved throughput and on the capabilities of the computing node. For our tests and as we will discuss later, the bare-metal configuration was determined to be the optimal one in that aspect.

In our tests, we have run our VNP in KVM and Docker to assess both virtualization technologies. However, our solution is not restricted to any specific virtualization platform as we depend only on standard features, such as the capability of enabling PCI passthrough for specific devices or sharing part of the filesystem with the probe.

5. Performed tests and results

Once the architecture and the underlying technology of the VNP have been discussed, we present a series of tests and results, which demonstrate that our virtual network probes are feasible with 40 Gbit/s off-the-shelf network cards in future 5G networks.

All the tests have been performed in the same testbed, which consists of two separate physical machines that are connected by a 40 GbE direct attach cable, with the specifications that are listed in Table 1. Both machines run the same operating system, namely, a minimal Gentoo with a 4.14.7 Linux kernel, with the recent Meltdown/Spectre patches disabled for performance reasons. The VMs were all running the same system, namely, a CentOS 7.4 OS with a 3.10.0-693 Linux kernel.

Table 1
Specifications of the servers that were used for testing. HyperThreading was disabled.

	Traffic generator	**VM Host**
CPU	2 × Intel Xeon E5-2630v4	2 × Intel Xeon Gold 6126
Clock	2.20 GHz	2.60 GHz
Cores	20	24
Memory	128 GB	192 GB
NIC	Intel XL710	Intel XL710

As a baseline, we first performed a test in which we measured the capture percentages of both DPDK and HPCAP40 [11] with the physical interface in both the bare-metal and PCI passthrough configurations. In the first case, the drivers ran in the host, and in the second, inside a virtual machine. In the latter case, although it appears similar to our objective, enabling PCI passthrough for the physical NIC blocks its usage in the host and in other virtual machines; therefore, is not useful for our purposes. However, it serves as a baseline test of how much overhead the PCI passthrough configurations introduce.

Table 2
Percentage of packets that are captured for various traffic patterns in a fully saturated 40 Gbit/s link that are obtained by different solutions in bare-metal (BM) and PCI passthrough (PT) configurations.

Capture engine	**% of packets captured**			
	BM		PT	
	64 B	*CAIDA*	*64 B*	*CAIDA*
i40e	1.6	11.59	0.2	9.44
DPDK	100	100	100	100
HPCAP40	75.1	100	51.5	100

We compared the results that were obtained with the capture drivers with what we achieved with the default driver, namely, *i40e*, and a *tcpdump* program. According to Table 2, the default driver does not achieve a sufficient capture rate in any case and reaches at most an 11 % when capturing data from a CAIDA trace [43]. DPDK reaches full capture in both minimum-frame-size traffic and the CAIDA trace, whereas HPCAP40 only achieves full capture in the CAIDA trace, with noticeable problems in capturing minimum-frame-size traffic. This is due to the way in which HPCAP40 operates, namely, by copying the frames to an intermediate buffer for subsequent storage to disk. In contrast, DPDK discards the packets after capturing them.

We also tested the throughput of our NVMe disk array with three setups: a bare-metal setup with the software RAID that is mounted directly in the host machine, a software RAID mounted in a virtual machine using PCI passthrough, and a setup that uses VirtIO to mount the disks inside of the virtual machine. Then, we copied 3000 blocks of size 5632 KB each (the blocksize was chosen according to the specifications of the NVMe drives for optimal write speed) and annotated the throughput, repeating each measurement ten times.

The system that was used in the test was the VM host, where we would later run the traffic tests. We had up to 11 Intel DC P3700 NVMe drives available. Drives of this type were chosen because, at the time of writing this paper, they were the best-performing storage option in terms of read/write speeds. Achieving similar speeds with mechanical drives requires larger arrays, which might be overly expensive or impractical for our use case.

The results are shown in Figure 4. The bare-metal solution for the NVMe array consistently performs above 40 Gbit/s with 5 disks, which is the speed that is necessary for our storage requirements. The passthrough configuration can barely reach 40 Gbit/s and the maximum speed of VirtIO seems to be 25 Gbit/s.

After we confirmed that the hardware yields acceptable results, we performed tests with both HPCAP40vf and DPDK in two main scenarios, each related to a critical point in the implementation of the traffic capture with virtual functions. First, we want to ensure that we can receive the network frames in the virtual probe at a sufficiently high rate and that the capture driver performs well and is not substantially affected by the overhead of running in the virtual environment. Additionally, we want to measure the effect of the capture on the normal operation of the system and the links between virtual machines. We also want to test the storage system to ensure that we can store the data from the virtual probe to disk with sufficient bandwidth.

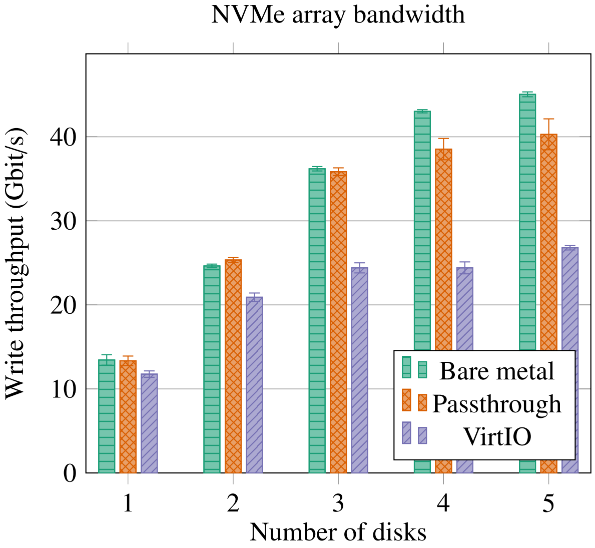

Fig. 4. Bandwidths using software RAIDs and various numbers of NVMe drives.

5.1. Network traffic capture

To measure the capacity of our system to receive the network traffic in the virtual probe and the effect of the capture on the normal operation of the VFs, we set up our testbed in the following way: One of the machines sent synthetic traffic of a fixed frame size that saturated the 40 GbE link to the XL710 card. A second machine was set with 3 VMs, each with one assigned VF. The first VM captured traffic with the driver and discarded it afterwards, while the other two ran an iperf bandwidth test[9]. The first VF was set with mirroring rules and promiscuous mode so that it received both the synthetic and the iperf-generated traffic. As a reference, iperf reported a maximum bandwidth of 25 Gbit/s with no external load on the XL710 link.

We performed several tests in which the frame size (including CRC) was varied by powers of two, as that is the variable that most affects the performance of the capture system. Table 3 lists the results of these tests. For brevity, we do not list the results for all frames sizes above the threshold at which we achieved 100 % capture in all cases. With 64-byte frames, DPDK captures 66.3 % of the frames and HPCAP40vf 57.5 %. However, at this high frame rate, the internal switch is not capable of dealing with the extra load of an iperf test, which reports a mere 66.7 Kbps bandwidth. The results improve with a frame size of 128 bytes, which is still below what is typically observed in enterprise networks. At this point, DPDK captures 100 % of the traffic and HPCAP40vf 61.9 %. This difference was expected, as HPCAP40vf is optimized for storage of the traffic and the

[9] `https://iperf.fr/`

Table 3
Packet capture performance when using PCI passthrough to a VM and capturing traffic in a fully saturated 40 GbE link, together with the bandwidth that was reported by iperf in two concurrent VMs that were running in the same host.

Synthetic frame size	Synthetic TX rate	DPDK capture %	HPCAP40vf capture %	iperf bandwidth
64 bytes	59.5 Mpps	66.3 %	57.5 %	66.7 Kbit/s
128 bytes	33.8 Mpps	100.0 %	61.9 %	2.2 Gbit/s
256 bytes	17.9 Mpps	100.0 %	100.0 %	3.5 Gbit/s
1024 bytes	4.7 Mpps	100.0 %	100.0 %	3.8 Gbit/s
1514 bytes	3.2 Mpps	100.0 %	100.0 %	3.6 Gbit/s

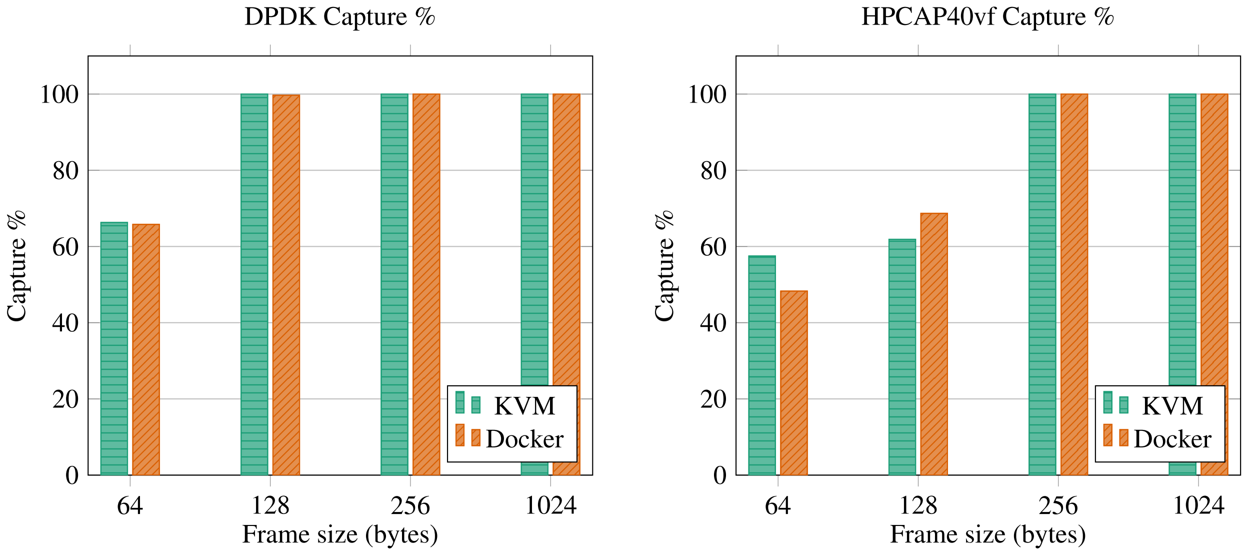

Fig. 5. Comparison of the capture percentages of DPDK and HPCAP40vf in a virtual machine and in a Docker environment.

extra copy adds an overhead that affects the performance when the traffic is not stored but discarded.

With 256-byte frames, both DPDK and HPCAP40vf capture 100 % of the traffic and the bandwidth that is reported by iperf increases to 3.5 Gbit/s since the internal switch has a lower load due to the decrease in the number of packets per second that it must process. Further increases in the frame size do not result in better iperf performance, as the bottleneck appears to be the PCI bandwidth of the system, which has an apparent limit of ~44 Gbit/s.

We also compared the performances that were obtained when running the virtual network probe as a kernel virtual machine and as a Docker container. Due to the restrictions of the containers, both DPDK and HPCAP40vf must run in the host. Mounting a set of special files inside the containers enables them to use and control the capture driver.

The results are shown in Figure 5, where the same effect is observed in DPDK and HPCAP40vf: for minimum-sized frames, both drivers perform worse in Docker than in KVM. However, their performances improve faster in Docker and in both cases, full capture without loss is realized for frames of size 254 bytes or higher.

5.2. *Traffic storage*

Now that we are sure that both drivers can receive traffic at sufficiently high rates, we test whether we can store that traffic at line rate. For both Docker and KVM, the setup is the same: a software RAID-0 is mounted with 5 NVMe drives in the virtual probe where the driver is running, as illustrated in Figure 4. Then, our traffic generator reproduces a trace with real traffic and we measure the throughput of our application.

For HPCAP40vf, the storage application is hpcapdd, which copies the data in blocks from the intermediate buffer to a file. Although in a previous work [11] we used a specialized application that worked with SPDK[10] for optimal results, we decided to not use it in these tests as SPDK adds complexity to the installation and we achieved a sufficient rate with the hpcapdd and 5 NVMe drives. This is because we used a better NVMe drive, which doubles the write speed with respect to the prior drive model (Intel DC P3600).

With DPDK, various public tools promise that every packet can be captured and appended to a pcap file at reasonable speeds. However, we could not run any of them at 40 Gbit/s for

[10] http://www.spdk.io/

Table 4
Performance when storing two traffic traces to disk.

		Capture %	
Trace	**Send rate**	**DPDK**	**HPCAP40vf**
CAIDA	39.80 Gbit/s	99.5 %	100.0 %
UAM	39.83 Gbit/s	100.0 %	100.0 %

a long period. The most promising solution is probably Flowscope [17]. The authors claim they can reach up to 100 Gbit/s. Unfortunately, this figure is not a sustained rate and is only realized over a very short time period, where the system detects that there is an anomaly. In addition to Flowscope, we can find other DPDK-based implementations, such as dpdkcap [16]; however, the code is very old and unmaintained. Therefore, we could not use it with a DPDK version that is compatible with the firmware of our card. Hence, we have developed our custom DPDK capture solution[11], which provides full pcap dump at 40 Gbit/s.

Table 4 lists the results that we obtained with two traffic traces: one was obtained at CAIDA [43] and the other is a capture of traffic in the student laboratory at our university (UAM). The average frame sizes in these files are 787 and 910 bytes, respectively. We show only the results in KVM, as the differences with the results in Docker are negligible.

DPDK only captures the UAM trace without losses. However, HPCAP40vf is capable of capturing all frames in both cases without loss, which demonstrates that it is feasible to monitor and store the traffic of a network at 40 Gbit/s using a virtual network probe.

These results and the differences between capture engines are expected. As explained previously, HPCAP40vf is oriented to the capture and storage of the traffic packets; hence, it makes an extra copy after capturing the packets to align them in memory. This affects the capture percentage when the traffic is discarded and not stored; however, it enables client applications to obtain better performance when writing the data to disk in aligned blocks.

6. Conclusions

In this paper, we proposed a virtual network probe that uses 40 Gbit/s off-the-shelf network cards and demonstrated its feasibility as part of the observe-analyze-act loop that will enable future 5G networks to be autonomic and self-managed. This virtual probe can be incorporated easily into existing deployments without any custom hardware and used as a permanent monitoring solution or launched on-demand as part of a monitoring-as-a-service platform.

After exploring the possibilities regarding probe virtualization, we demonstrated that an architecture that makes use of the SR-IOV virtual functions, which are present in commercial off-the-shelf NICs, enables the monitoring of both physical networks with virtual probes and fully virtual networks. The use of the VFs as an accelerator enables our solution to reach sufficient speed to capture all the traffic that is passing through the network in most scenarios. Moreover, via NVMe arrays and our custom drivers, we can store that traffic and analyze it at a later time if necessary. However, operators must be careful to avoid saturating existing deployments if they require full monitoring of a virtual network that uses VFs: the mirrored traffic is delivered to the virtual probe via the PCI bus and the card chipset is unable to surpass the upper bound of ∼44 Gbit/s, as demonstrated in our tests.

We have also demonstrated that our system is agnostic to the virtualization technology that is employed, as it can run in both containers and virtual machines. Although we tested our system using Docker and KVM, we did not use any features that are specific to either of those solutions; therefore, our VNP should work with other alternatives, such as LXC and VMWare.

For our system, we have used the freely available Data Plane Development Kit (DPDK) and developed HPCAP40vf, which is a 40 Gbit/s VF-aware version of our capture and storage driver, namely, HPCAP. We have made available its source code under a GPL license on GitHub so that the community can freely use under Linux both solutions that are presented in this paper.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under the project TRÁFICA (MINECO/FEDER TEC2015-69417-C2-1-R), and by the European Commission under the project H2020 METRO-HAUL (Project ID: 761727).

References

[1] S. van der Meer, J. Keeney, L. Fallon, 5G networks must be autonomic!, in: IEEE/IFIP Network Operations and Management Symposium (NOMS 2018), 2018.

[2] L. Velasco, L. Gifre, J. L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, F. Cugini, An architecture to support autonomic slice networking, Journal of Lightwave Technology 36 (1) (2018) 135–141. doi:10.1109/JLT.2017.2748233.

[3] L. Gifre, A. P. Vela, M. Ruiz, J. López de Vergara, L. Velasco, Experimental assessment of node and control architectures to support the observe-analyze-act loop, in: Optical Fiber Communication Conference, Optical Society of America, 2017, pp. Th1J–3.

[4] Minimum requirements related to technical performance for IMT-2020 radio interface(s), Tech. Rep. SG05 Contribution 40, ITU-R (Feb. 2017).

[5] IEEE Standard for Ethernet - Amendment 10: Media Access Control Parameters, Physical Layers, and Management Parameters for 200 Gb/s and 400 Gb/s Operation, IEEE Std 802.3bs-2017 (Amendment to IEEE 802.3-2015 as amended by IEEE's 802.3bw-2015, 802.3by-2016, 802.3bq-2016, 802.3bp-2016, 802.3br-2016, 802.3bn-2016, 802.3bz-2016, 802.3bu-2016, 802.3bv-2017, and IEEE 802.3-2015/Cor1-2017) (2017) 1–372doi:10.1109/IEEESTD.2017.8207825.

[6] R. Jain, S. Paul, Network virtualization and software defined networking for cloud computing: a survey, IEEE Communications Magazine 51 (11) (2013) 24–31.

[11] https://github.com/hpcn-uam/DPDK2disk

[7] G. Karagiannis, D. Hai, G. Dobrowski, Y. Hertoghs, N. Zong, Cloud central office reference architectural framework, Tech. Rep. TR-384, Broadband forum (Jan. 2018).

[8] S. C. Ugbolue, Polyolefin Fibres: Structure, Properties and Industrial Applications, Woodhead Publishing, 2017, chapter 2.4: "Monitoring-as-a-Service (MaaS)".

[9] Altnix, Monitoring as a service (MaaS) - does it work?, White Paper (Feb. 2014).

[10] M. Jarschel, T. Zinner, T. Höhn, P. Tran-Gia, On the accuracy of leveraging sdn for passive network measurements, in: 2013 Australasian Telecommunication Networks and Applications Conference (ATNAC), 2013, pp. 41–46. `doi:10.1109/ATNAC.2013.6705354`.

[11] G. Julián-Moreno, R. Leira, J. E. López de Vergara, F. J. Gomez-Arribas, I. Gonzalez, On the feasibility of 40 gbps network data capture and retention with general purpose hardware, in: 33rd ACM/SIGAPP Symposium On Applied Computing (SAC'2018).

[12] V. Moreno, R. Leira, I. Gonzalez, F. J. Gomez-Arribas, Towards high-performance network processing in virtualized environments, in: 2015 IEEE 17th International Conference on High Performance Computing and Communications, IEEE, 2015, pp. 535–540.

[13] M. Kang, D.-I. Kang, M. Yun, G.-L. Park, J. Lee, Design for run-time monitor on cloud computing, in: Security-Enriched Urban Computing and Smart Grid, Springer, 2010, pp. 279–287.

[14] M. Forconesi, G. Sutter, S. Lopez-Buedo, J. E. Lopez de Vergara, J. Aracil, Bridging the gap between hardware and software open-source network developments, IEEE Network 28 (5).

[15] V. Moreno, J. Ramos, P. M. S. del Río, J. L. García-Dorado, F. J. Gomez-Arribas, J. Aracil, Commodity packet capture engines: Tutorial, cookbook and applicability, IEEE Communications Surveys Tutorials 17 (3) (2015) 1364–1390. `doi:10.1109/COMST.2015.2424887`.

[16] W. de Vries, Scalable high-speed packet capture, in: RIPE 71, 2015.

[17] P. Emmerich, M. Pudelko, S. Gallenmüller, G. Carle, Flowscope: Efficient packet capture and storage in 100 gbit/s networks, in: Proceedings of the 16th International IFIP TC6 Networking Conference, IEEE, 2017.

[18] L. Deri, A. Cardigliano, Towards 100 Gbit flow-based network monitoring, in: FloCon 2016, 2016.

[19] R. Bolla, R. Bruschi, A. Carrega, F. Davoli, Green networking with packet processing engines: Modeling and optimization, IEEE/ACM Transactions on Networking 22 (1) (2014) 110–123. `doi:10.1109/TNET.2013.2242485`.

[20] R. Leira, P. Gómez, I. González, J. E. López de Vergara, Multimedia flow classification at 10 Gbps using acceleration techniques on commodity hardware, in: 2013 International Conference on Smart Communications in Network Technologies (SaCoNeT), Vol. 03, 2013, pp. 1–5. `doi:10.1109/SaCoNeT.2013.6654555`.

[21] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux Journal 2014 (239) (2014) 2.

[22] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2015, pp. 171–172. `doi:10.1109/ISPASS.2015.7095802`.
URL `http://domino.watson.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf`

[23] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, G. Carle, Throughput and latency of virtual switching with open vswitch: A quantitative analysis, Journal of Network and Systems Management 26 (2) (2018) 314–338. `doi:10.1007/s10922-017-9417-0`.
URL `https://doi.org/10.1007/s10922-017-9417-0`

[24] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, The design and implementation of open vswitch, in: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), USENIX Association, Oakland, CA, 2015, pp. 117–130.
URL `https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff`

[25] S. Yamany, Methods, systems, and computer program products for distributed packet traffic performance analysis in a communication network, uS Patent 9,565,073 (Feb. 7 2017).

[26] F. Moradi, C. Flinta, A. Johnsson, C. Meirosu, Conmon: an automated container based network performance monitoring system, in: Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on, IEEE, 2017, pp. 54–62.

[27] H. Mahkonen, R. Manghirmalani, M. Shirazipour, M. Xia, A. Takacs, Elastic network monitoring with virtual probes, in: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015, pp. 1–3. `doi:10.1109/NFV-SDN.2015.7387390`.

[28] E. J. Halpern, E. C. Pignataro, Service function chaining (sfc) architecture, RFC 7665 (Oct. 2015).

[29] G. Brown, Virtual probes for nfvi monitoring, White Paper (Feb. 2017).

[30] S. Shanmugalingam, A. Ksentini, P. Bertin, Dpdk open vswitch performance validation with mirroring feature, in: 2016 23rd International Conference on Telecommunications (ICT), 2016, pp. 1–6. `doi:10.1109/ICT.2016.7500387`.

[31] J. F. Zazo, S. Lopez-Buedo, Y. Audzevich, A. W. Moore, A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances, in: ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on, IEEE, 2015, pp. 1–6.

[32] M. Helsley, Lxc: Linux container tools, IBM devloperWorks Technical Library 11.

[33] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, KVM: the linux virtual machine monitor, in: Proceedings of the Linux Symposium, 2007.

[34] M. Raho, A. Spyridakis, M. Paolino, D. Raho, Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing, in: 2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), 2015, pp. 1–8. `doi:10.1109/AIEEE.2015.7367280`.

[35] A. Kudryavtsev, V. Koshelev, A. Avetisyan, Prospects for virtualization of high-performance x64 systems, Programming and Computer Software 39 (6).

[36] L. Rizzo, G. Lettieri, V. Maffione, Speeding up packet I/O in virtual machines, in: Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2013.

[37] R. Russell, Virtio: Towards a de-facto standard for virtual I/O devices, SIGOPS Operating Systems Review 42 (5).

[38] G. Motika, S. Weiss, Virtio network paravirtualization driver: Implementation and performance of a de-facto standard, Computer Standards & Interfaces 34 (1).

[39] J. Hwang, K. Ramakrishnan, T. Wood, NetVM: High performance and flexible networking using virtualization on commodity platforms, in: Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 2014.

[40] C. Yang, J. Liu, H. Wang, C. Hsu, Implementation of gpu virtualization using pci pass-through mechanism, The Journal of Supercomputing 68 (1).

[41] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gomez-Arribas, J. Aracil, Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems, International Journal of Network Management 24 (4).

[42] V. Moreno, P. Santiago del Rio, J. Ramos, J. Garcia-Dorado, I. Gonzalez, F. Gomez-Arribas, J. Aracil, Packet storage at multi-gigabit rates using off-the-shelf systems, in: Proceedings of the IEEE International Conference on High Performance and Communications (HPCC2014), 2014.

[43] C. Walsworth, E. Aben, k. Claffy, D. Andersen, The CAIDA anonymized Internet traces 2016 dataset, `http://www.caida.org/data/passive/passive_2016_dataset.xml`, [06 April 2016].

Rafael LEIRA received his B.Sc. (Hons) degree in Computer Science in 2014, and his double M.Sc. in Computer Engineering and in Information and Communication Technologies in 2016, both from Universidad Autónoma de Madrid (UAM), Spain, where he is pursuing his Ph.D. in Computer Science and Telecommunication Engineering. He has been collaborating with the High-Performance Computing and Networking research group at UAM since 2012, where he held an FPU-MECD research grant until August 2018. He is now hired at Naudit HPCN.

Guillermo JULIÁN-MORENO received his double B.Sc. (Hons) degree in Computer Engineering and Mathematics from Universidad Autónoma de Madrid, Spain in 2016, and an M.Sc. in Computational Science and Engineering at École Polytechnique Fédérale de Lausanne, Switzerland in 2018. He has been collaborating with the High-Performance Computing and Networking research group since 2013 and now works in its spin-off, namely, Naudit HPCN, on network capture drivers and network analysis.

Iván GONZÁLEZ received his M.Sc. degree in Computer Engineering in 2000 and his Ph.D. in Computer Engineering in 2006, both from UAM, Spain. From October 2002 to October 2006, he was a Teaching Assistant at the Computer Engineering Department of UAM. From November 2006 to January 2008, he was a Postdoctoral Research Scientist at the High-Performance Computing Laboratory (HPCL), Electrical & Computer Engineering Department, George Washington University (Washington, DC). He was a Faculty Member of the NSF Center of High-Performance Reconfigurable Computing (CHREC) at George Washington University. He is currently Associate Professor at UAM, where he is teaching computer-architecture-related courses. His main research interests are heterogeneous computing (with GPUs, FPGAs, etc.), parallel algorithms, and performance tuning. Other interests include FPGA-based reconfigurable computing applications, with a special focus on dynamic partial reconfiguration, embedded systems, and robotics.

Francisco J. GÓMEZ-ARRIBAS received his Ph.D. from UAM, Spain, in 1996. From October 1996 until November 2000, he was an assistant professor at the Computer Engineering Department of UAM. He is currently a professor of computer architecture and parallel computing at the same university. His research fields of interest concern reconfigurable computing applications that are based on FPGA circuits, with a special focus on the design of multiprocessor systems with reconfigurable architecture. Secondary fields of interest include network computing, cryptographic coprocessors, embedded system-on-a-chip, and experimental support of computer science and electrical engineering education on the Internet.

Jorge E. LÓPEZ DE VERGARA has been associate professor at Universidad Autónoma de Madrid (Spain) since 2007 and is a partner of Naudit HPCN, which is a spin-off company that was founded in 2009 and is devoted to high-performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1998 and 2003, respectively, where he also held an FPU-MEC research grant. During his Ph.D., he stayed for 6 months in 2000 at HP Labs in Bristol. He studies network and service management and monitoring, and has coauthored more than 100 scientific papers on this topic. He coordinates the Tráfica project at UAM and leads METRO-HAUL tasks that are assigned to Naudit.