Towards high-performance network processing in virtualized environments

Victor Moreno, Rafael Leira, Ivan Gonzalez and Francisco J. Gomez-Arribas
High Performance Computing and Networking research group,
Universidad Autónoma de Madrid, Spain
{victor.moreno, rafael.leira, ivan.gonzalez, francisco.gomez}@uam.es

***Abstract*—Nowadays, network operators' amenities are populated with a huge amount of proprietary hardware devices for carrying out their core tasks. Moreover, those networks must include additional hardware appliances as they host more and more services and applications offered by third-party actors. Thus, such an infrastructure reduces the profitability, as including new hardware boxes in the network becomes increasingly harder in terms of space, cooling and power consumption. In a context in which virtualization has become a ubiquitous technique, industry and academia has turned an eye to Network Function Virtualization (NFV) to mitigate those effects and maximize potential earnings. NFV aims to transform future network architectures by exploiting standard IT virtualization technology to consolidate a variety of network processing elements onto standard commodity servers. On this work, we assess the feasibility of moving high-performance network processing tasks to a virtualized environment. For such purpose, we analyse the possible configurations that allow feeding the network traffic to applications running inside virtual machines. For each configuration, we compare the usage of different high-performance packet capture engines on virtual machines, namely PF_RING, Intel DPDK and HPCAP. Specifically, we obtain the performance bounds for the primary task, packet sniffing, for physical, virtual and mixed configurations. We also developed HPCAPvf, a counterpart of HPCAP for virtual environments, and made it available under a GPL license.**

Keywords*—*Virtual network functions, packet capture, virtual machines.

I. Introduction

Over the past decades, the use of the Internet has rapidly grown due to the emergence of new services and applications. The amount of services and applications available to end-users makes it necessary for those services' providers to deploy quality-assessment policies in order to distinguish their product among the rest. In this scenario, network processing and analysis becomes a central task that has to deal with humongous amounts of data at high-speed rates. Obviously, service providers must be able to accomplish such a challenging task using processing elements capable of reaching the required rates while keeping the cost as low as possible for the sake of profitability.

To deal with such amount data, specialized hardware (HW) solutions have been developed, which are traditionally based on the used of Field-Programmable Gate Arrays (FPGAs) or Network Processors. These solutions answer the high-performance needs for specific network monitoring tasks, e.g. routing or classifying traffic on multi-Gb/s links [1], [2]. However, those solutions imply high investments: such hardware's elevated cost rises capital expenditures (CAPEX), while operational expenditures (OPEX) are increased due to the difficulty of their operation, maintenance and evolution. Furthermore, HW life cycles become shorter as technology and services evolve, which prevents new earnings and limits innovation. Those drawbacks turn specialized HW solutions into a non-desirable option for large-scale network processing.

As an alternative to mitigate those negative effects, academia and industry turned an eye to the use of commodity hardware in conjunction with open source software [3]. Such combination is referred as an off-the-shelf system in the literature. The advantages of those systems lay in the ubiquity of those components, which makes it easy and affordable to acquire and replace them and consequently reduces CAPEX. Those systems are not necessarily cheap, but their wide-range of application allows their price to benefit from large-scale economies and makes it possible to achieve great degrees of experimentation. Additionally, such systems offer extensive and high-quality support, thus reducing OPEX. However, the increased demands for network processing capacity would be translated into a big number of machines even though if off-the-shelf systems were used. Such an amount of machines means high expenses in terms of power consumption and physical space. Moreover, the presence of commodity servers from different vendors empowers the appearance of interoperability issues. All those drawbacks damage the profitability that networked service providers may experience.

On the other hand, there has been an increasing trend regarding the use of virtualization for computational purposes. Such trend has been empowered by the inherent advantages provided by virtualization solutions [4]. In this light, network operators and service providers have been working during the last years on the development of the concept of Network Function Virtualization (NFV). This new paradigm aims to unify the environments where network applications shall run by means of adding a virtualization layer on top of which network applications may run. This novel philosophy also allows merging independent network applications using unique hardware equipment. Consequently, the application on NFV can increase the benefits obtained by network service providers by (i) reducing their equipment investment by acquiring large-scale manufacturers' products and by reducing the amount of physical machines required, which also entails cuts in power consumption; (ii) speeding up network applications maturation cycle as all applications are developed in an unified environment; (iii) easing maintenance procedures and expenditures as testability is radically enhanced; (iv) opening the network applications' market for small companies and academia by minimizing risk and thus encouraging innovation; (v) the possibility to rapidly adjust network applications and resources based on specific clients requirements.

The development of this novel NFV philosophy has been favoured by other trending technologies such as Cloud Computing and Software Defined Networking (SDN). In the case of Cloud Computing, NFV can easily benefit from all the research carried out on virtualization management [5]. Furthermore, NFV is to reside inside Cloud providers' networks in order to carry out all the network-related management tasks. On the other hand, NFV is complementary to SDN but not dependent on it and vice-versa [6]. However, NFV enhances SDN as it provides the infrastructure on top of which SDN software can run.

However, in order to make the most of NFV, mechanisms that allow obtaining maximum network processing throughput in such virtual environments must be developed. In a bare-metal scenario, researchers have recently focused on developing high-performance packet capture engines [3]. This work evaluates the feasibility of applying such capture engines in NFV-based environments for the primal network task: packet capture. Specifically, we evaluate different virtualization alternatives, namely PCI-passthrough and Network Virtual Functions (NVF) available on contemporary systems. For each capture engine and environment, we measure the performance bounds, so researchers and practitioners may benefit from our results in order to build their high-performance NFV-based applications.

II. Network processing on bare-metal scenarios

In the recent years both the research community and industry have paid attention to the use of off-the-shelf for network processing purposes [7]. Those systems offer interesting features that allow reducing CAPEX and OPEX when building a system on top of them. In terms of network processing, off-the-shelf systems have traditionally relied on the use of the corresponding NIC vendor's driver plus a standardized network stack. Such approach is characterized by a great degree of flexibility, as the network stack provides independent layers that allow distributing the traffic to the corresponding final applications. Nevertheless, the performance obtained by such approaches is poor: every incoming packet must traverse a set of layers, which is translated into additional copies, resource re-allocations at processing time. Thus, this flexibility the standard solution offers limits its applicability in high-speed scenarios.

Consequently, if off-the-shelf systems are to be applied in high-speed networked scenarios, they have to be carefully planned and tuned. In this light is how high-performance packet capture engines were born [3]. Solutions such as PF_RING, PacketShader, netmap, PFQ, Intel DPDK or HPCAP were created as high-performance counterparts for the traditional network driver-plus-stack alternative. All those solution are based on some of the following ideas (see [3], [8] for a detailed discussion):

- *Pre-allocation and re-use of memory*: traditional solutions allocate a set of structures and buffers for each received packet. Those resources are also released once the packet is delivered to upper layers. Such technique is a very demanding task and may be optimized by pre-allocating pools of resources for re-using them along time.
- *Memory mapping*: this technique makes it possible for high-level applications to map buffers and data structures allocated at driver-level. Consequently, the number of copies experimented by incoming packets is reduced.
- *Use of parallel direct paths*: modern NICs support have multiple parallel reception queues and to distribute incoming traffic among such queues using RSS (Receive Side Scaling) mechanisms. However, the use of contemporary network stacks becomes a bottleneck serializing all the traffic at one single point for delivering it to upper layers. In this light, high-performance network solution must avoid serialization point for really exploiting NIC's parallel nature. Note that the use of parallel paths may cause incoming packet reordering [9], so it must be carefully planned.
- *Batch processing*: traditional end-user network applications retrieves packets individually by means of system calls, which in Linux systems involves at least two context switches. In order to mitigate this effect, some capture engines pretend processing several packets with a single system call. Solutions such as PacketShader, netmap or Intel DPDK group packets into batches, and so all packets conforming the batch are handled in the same system call. Note that applying such technique, in spite of its performance improve-

ment, may entail side effects such as latency increase and inaccurate timestamping [10]. For this reason, solutions such as HPCAP propose a byte-stream oriented approach.

- *Prefetching*: this technique consists in preloading memory locations in processors' caches in a predictive way so that it can be quickly accessed in a near future. Its application reduces the number of cache misses in capture process and thus leverages performance.
- *Affinity planning*: contemporary server feature NUMA architectures, where applications performance is highly influenced by the NUMA node or processor it is executed on. Thus, the affinity of all threads and processes involved in a capture system must be carefully scheduled. Such schedule must also take into account the connectivity of the processors to the PCI slot the NICs are connected to.

All the above-mentioned solutions enable network managers to deploy high-performance network applications on top of them. Table I shows the performance level obtained for packet capture in a full-saturated 10 Gb/s link for the worst-case scenario (64 byte packets) and an average scenario (a CAIDA from a ISP's backbone link between Chicago and Seattle obtained the 19^{th} June 2014, with an average packet size of 965 bytes [11]). We have compared the performance of the standard `ixgbe` plus network stack solution compared to PF_RING, Intel DPDK and HPCAP. We chose PF_RING as an archetype for the previously mentioned capture engines, and Intel DPDK and HPCAP for being the ones supporting virtual environments. Importantly, HPCAP has been developed by the authors as capture engine focused not only on high-performance packet capture rates, but also on accurate packet timestamping [10] and on building a multi-granular multi-purpose monitoring framework [12].

The tests have been carried out using a server with two Xeon E5-2630 processors running at 2.6 Ghz and 32 GB of DDR3 RAM. The NIC used was an Intel 82599 card connected to a PCIe Gen3 slot. The operating system in use is a Linux Fedora 20 with a 3.14.7 kernel. The results obtained show that all four capture engines are capable of capturing 100% of the packets when replying the CAIDA trace at top-speed. In the worst-case scenario, `ixgbe` captures only 2.7% of the incoming packets, while PF_RING and Intel DPDK capture 100% of them, and HPCAP captures 97.9% of the packets. Note that the performance degradation experienced by HPCAP is due to the driver timestamping each incoming packet.

Nevertheless, the application of the existing packet capture engines has been limited in the literature to the bare-metal case. That is, a physical server to which the NIC is connected through a PCIe slot.

Configuration	Bare-metal		PCI pass-through	
Traffic	64 byte	CAIDA	64 byte	CAIDA
ixgbe	2.7 %	100 %	1.9 %	62.7 %
PF_RING	100 %	100 %	100 %	100 %
DPDK	100 %	100 %	100 %	100 %
HPCAP	97.9 %	100 %	82.5 %	100 %

TABLE I: Percentage of packets captured for different traffic patterns in a fully-saturated 10 Gb/s link obtained by different solutions in a bare-metal and PCI pass-through configuration

This configuration corresponds to the leftmost NIC shown on Fig. 1. Consequently, the application of the high-performance packet capture engines has not been evaluated in NFV environments yet.

III. Network processing in virtualized environments

When using virtual machines (VMs) for computing intensive applications, the performance obtained by the target application is usually damaged. For this reason, if we desire to obtain maximum performance, the creation and schedule of each VM must be carefully made. For example, the amount of cores of the VM should be such that allows the VM to be executed inside a single NUMA node in the physical server, and those virtual cores should be configured to be mapped to independent physical cores. Another determinant factor for VM's performance is optimizing how the VM accesses the physical system's memory. Contemporary VM managers allow attaching the VM's memory to certain NUMA node, which will increase overall memory access latency. However, studies such as [13] point out the importance of using Linux's huge pages for allocating the VM's memory chunk so the amount of page misses is reduced and performance is enhanced.

By using Linux huge pages both the packet capture engines and the network applications built on top of them running on a VM would experience a performance increase. To confirm this effect, we empirically tested the impact on the percentage of captured packets when using different capture approaches from VMs running over huge pages and not (those experiments were made using the PCI pass-through technique, see subsection III-B). We experienced a slight performance improvement when running them on top of huge pages. Note that those were simple experiments in which only packet capture were made, but by using Linux huge pages network applications built on top of those VMs would also see their performance boosted regardless the capture procedure used. All the VMs used along the performance experiments presented in this paper were created taking these facts into account. Regarding the operating system running inside the VMs, we used the same version as in the physical server, a Linux Fedora 20 with a 3.14.7 kernel. The choice of the operating systems to be used

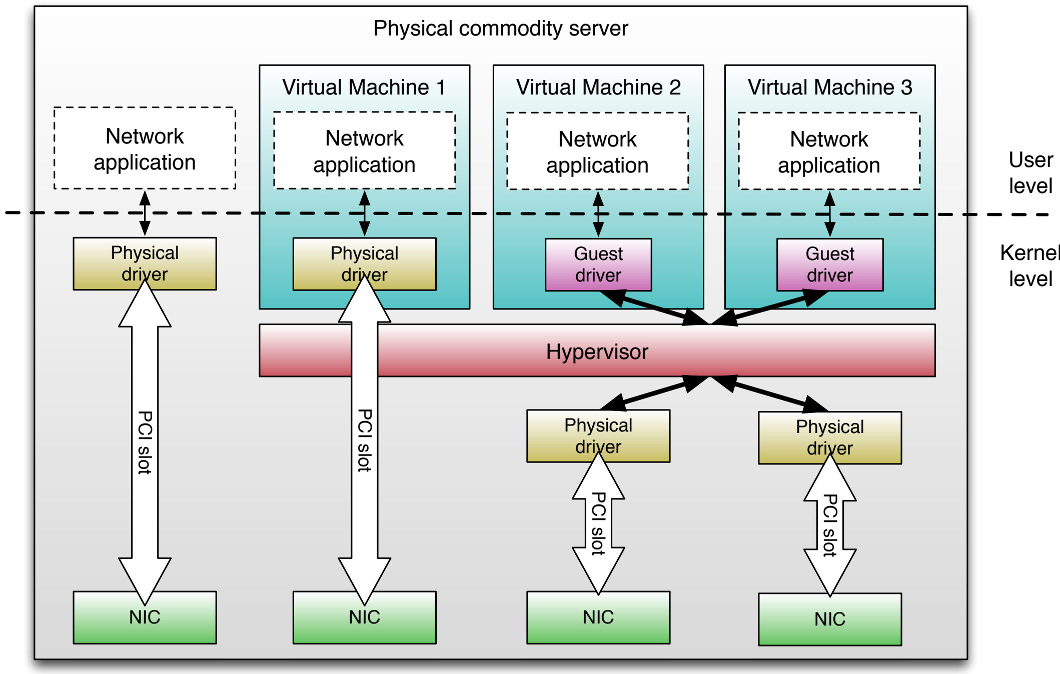

Fig. 1: Using a network device in bare-metal (leftmost), PCI pass-through (VM1) and emulated NIC (VM2, VM3)

both in the physical and virtual machines is relevant, as not all combinations support the use of Linux huge pages both in the physical server for creating the VM as mentioned and inside the VM. Note that some of the capture engines used such as PF_RING and Intel DPDK make use of those huge pages, so this requirement is nontrivial.

A. *VMs with emulated NICs*

In order to enable network applications running on top of NFV environments, we must understand the way a network device can be connected to a virtual machine (VM). The traditional scenario is composed by a set of virtual machines where an emulated network device is instantiated. In this configuration, incoming packets are captured by the NIC driver on the physical machine and traverse the system's network stack, and then delivered to the hypervisor's network module. Once acquired by the hypervisor, packets are delivered to the target VM depending on the virtual network configuration. This scenario is the one exhibited by VMs 2 and 3 on Fig. 1. Note that in this case the network driver used in the VM would be the one corresponding to the emulated device, and not the physical device that lies below. Such configuration implies at least two additional copies: from the physical server to the hypervisor and from the hypervisor to the virtual machine, which obviously degrades the performance achieved by the capture system.

Furthermore, the performance degradation experienced by network applications running in this configuration has pushed academia and industry to tune and optimize the hypervisor's packet handling policy. In this line, authors of [14] introduce VALE, which proposes a set of modification both on physical drivers and hypervisors in order to improve the network processing performance. By applying their proposals they leverage the system throughput: from 300 Mb/s using the conventional approach to nearly 1 Gb/s using VALE in the worst-case scenario (64 byte UDP packets); and from about 2.7 Gb/s to 4.4 Gb/s for TCP traffic. Although the results obtained by this approach are promising, the authors focus their attention on data exchange between VMs on the same physical server.

Note that both the traditional approach and VALE are base on the hypervisor as the central communication element. Consequently, the hypervisor becomes the bottleneck and a single point-of-failure for network processing tasks, limiting their applicability for network monitoring purposes. The use of an emulated NIC approach forces incoming packets to be processed twice: once when they arrive to the hypervisor, and again when they are transferred to their corresponding VM.

B. *VMs and PCI pass-through*

As an alternative to a hypervisor-centric approach, different hardware manufacturers developed a set of mechanism which enable direct connectivity from the

PCI adapter, referred as physical function, to the virtual machines providing near-native performance. The name given to those mechanisms depends of the underlying manufacturer: VT-d for Intel and ARM, Vi for AMD, but the technique is usually referred as PCI pass-through. This technique has been applied in several computational scenarios [15]. Applied to the network capture problem, this technology allows the VMs to map physical specific PCIe memory regions. Using this feature, VMs can operate as if they had the NIC physically connected to them, as shown by VM1 on Fig. 1. This implies that the drivers managing the NICs in the VMs are the same used to manage those NICs in the bare-metal scenario. Consequently, this allows network applications being executed in virtual machines to benefit from the high-performance packet capture solutions developed for bare-metal scenarios.

Table I shows the performance results in a fully-saturated link for the default `ixgbe` driver, PF_RING, Intel DPDK and HPCAP when executed in a VM with PCI pass-through compared to the previously mentioned bare-metal scenario. Those tests were made using the same hardware as previously mentioned and using KVM for creating and managing the VMs, and accordingly to the bare-metal scenario, the table depicts each capture engine's performance for the worst-case scenario and in an average scenario. The amount of packets captured by `ixgbe` falls from 2.7% under the bare-metal configuration to 1.9% using PCI pass-through in the worst-case scenario, and from 100% to 37.3% when replaying the previously mentioned CAIDA trace. On the other hand, PF_RING and Intel DPDK show no performance degradation when used via PCI pass-through, as they capture 100% of the packets on all scenarios. When it comes to HPCAP, packet capture performance is damaged when using PCI pass-through in the worst-case scenario, in which the amount of packets captured falls from 97.9% to 85.2%. The performance penalty experienced when introducing PCI pass-though can be blamed on the execution of the capture system in a virtual machine.

However, using PCI pass-through for instantiating network applications in different VMs has an inherent constraint: only one VM can make use of each network interface. That means that the scalability problem of instantiating more VMs for network processing purposes must be solved by adding additional NICs and probably more physical servers, as the amount of PCIe slots a server has is limited.

C. *VMs attached to virtual functions*

With the goal of promoting virtualization performance and interoperability the PCI Special Interest Group developed a series of standards, which they called Single-Root I/O Virtualization (SR-IOV). Those standards define the concept of virtual function (VF) as a lightweight image of the underlying physical PCI resource. Modern NIC such as the Intel 82599 use the concept of Virtual Machine Device Queues (VMDq) to refere to the NIC's virtual functions. By using VMDqs, the traffic arriving to the physical network device can be distributed among a set of queues based on a set rules that can be configured at hardware. Each of those queues is attached to a virtual PCI device, or virtual function (VF), that can be mapped via PCI pass-through by a certain VM. This configuration is represented in Fig. 4. Note that this configuration allows connecting an arbitrary number of VMs to a single physical device. The amount of virtual functions generated per physical device is limited by the hardware device, being 32 for the Intel 82599 adapter.

It is worth remarking that if this approach is used, only VF-aware drivers can be used in the VMs, so they handle the peculiarities those devices have. This requirement limits the amount of capture engines avail-

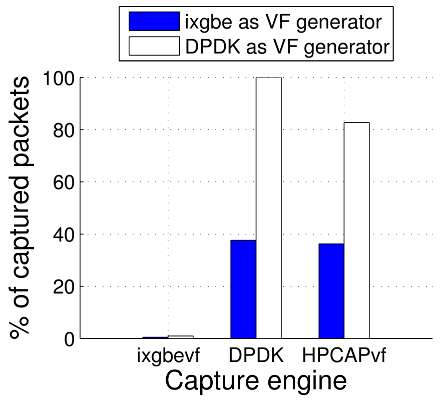

Fig. 2: Packet capture performance obtained when capturing from a 10 Gb/s link fully-saturated with 64-byte packets using virtualized alternatives for different virtual-function generators

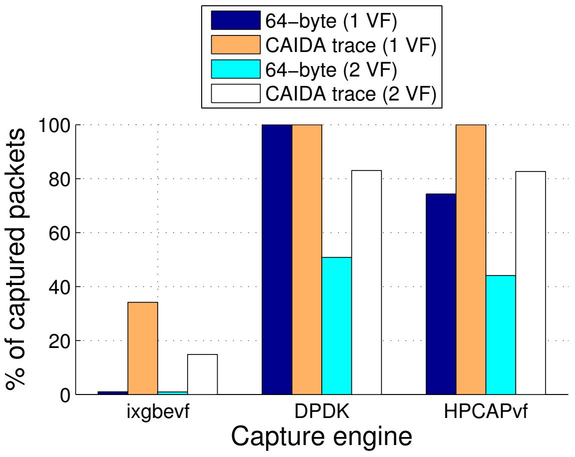

Fig. 3: Packet capture performance obtained by different virtualized alternatives when capturing different traffic in a fully-saturated 10 Gb/s link

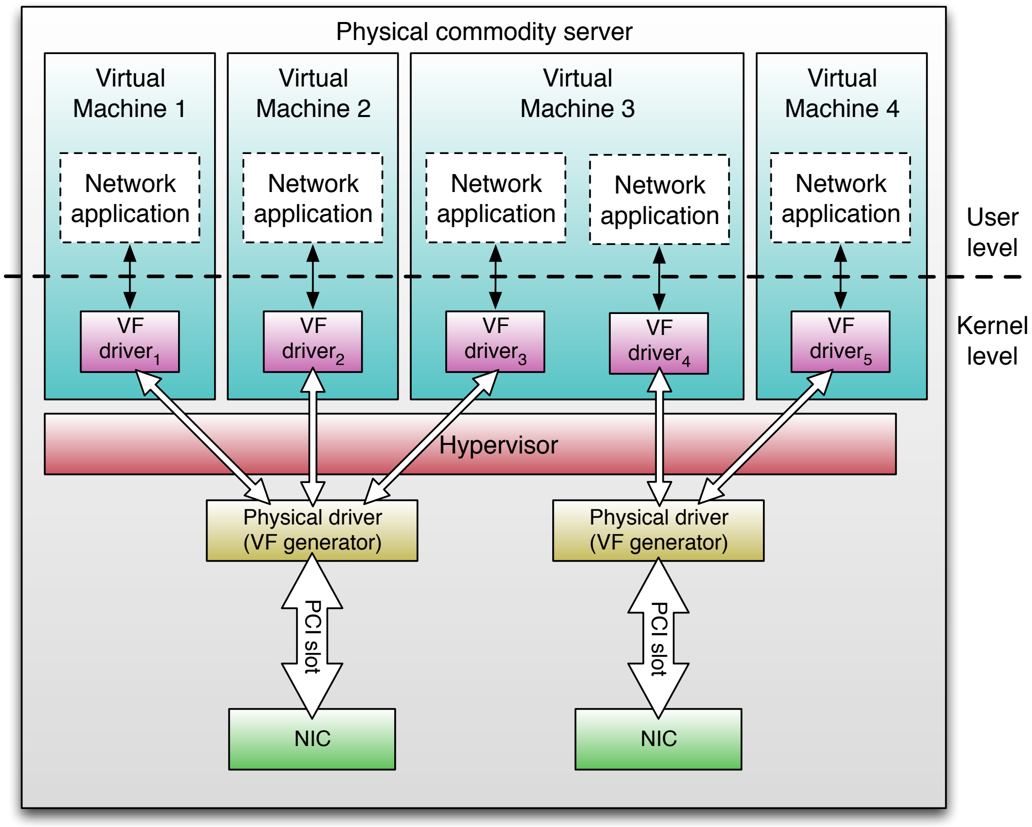

Fig. 4: Using a network device in a VM via virtual network functions

able. Specifically, Intel's DPDK has native support for working with VF, and they also supply a VF-aware counterpart to the `ixgbe` driver, named `ixgbevf`. Additionally, we developed a VF-aware version of HPCAP, that we named HPCAPvf, following all the design principles that guided HPCAP's design. Those three drivers have consequently been the only ones we have been able to test under this configuration.

On the other hand, the task of creating and managing the mentioned VF belong to the driver used in the physical server for managing the physical NIC. Above all the drivers previously mentioned capable of managing a physical NIC, only Intel's offer this feature. Consequently, when using VF only `ixgbe` and Intel DPDK are eligible. Importantly, the choice among those two drivers for generating the VF has an impact on the performance obtained by possible network applications running inside the VMs. Fig. 2 shows the effect of such choice when using different VF-aware for packet capturing in the worst-case scenario, that is a fully-saturated 10 Gb/s with 64-byte packets. Results show that using DPDK as VF generator improves the packet capture performance obtained from the VM side compared to the performance when using `ixgbe` for generating those VF. Specifically, when capturing packets in a VM using the `ixgbevf` driver using DPDK as VF generator raises the amount of packets captured from 0.5% to 1%. In a similar manner, when using DPDK and HPCAPvf in the VM side with `ixgbe` as VF generator, only 37.6% and 36.3% are respectively captured, but those ciphers are increased to 100% and 82.7% respectively when DPDK is used as VF generator.

When instantiating several VF through a single physical interface, the default traffic distribution policy is based on the MAC and IP addresses of each VM's interface the VF are connected to. Thus, each VF would only receive the traffic targeted to its corresponding VM. This would limit the use of this VF approach in scenarios where different network applications running on independent VMs need to be fed the same input traffic, which is the desirable case for scalable network monitoring. However, NICs such as Intel's 82599 supply a Mirroring and Multicast Promiscuous Enable (MPE) flags, which can be activated for each VF. By enabling those options, any VF could receive all the traffic traversing the physical device, or the traffic corresponding to a different VM, regardless it is targeted to its VM or not. Note that enabling those features in the Intel 82599 NIC implies a hardware-level packet replication, which minimizes the impact on the capture process' performance. Depending on the physical driver used to generate the NIC's VFs, activating the MPE may be done by tuning the driver's source code (that is the case when using `ixgbe` for generating the VFs) or by using a user-level application giving access to the NIC's registers (such as the `testpmd` application offered by Intel's DPDK).

Obviously, if several VMDqs are to be fed the same incoming packets, the physical driver will have to issue additional copies for each additional VMDq, and packet capture performance may be degraded. This effect is shown in the yellow and red bars of Fig. 3: adding a second VF to each physical device reduces the overall packet capture throughput obtained. When using `ixgbe` the amount of packets captured is 10% when using either one or two VFs in the worst-case scenario, but falls from 34.2% with one VF to 14.8% with two when replaying the CAIDA trace. Intel DPDK also suffer performance loss as it is capable of capturing all of the packets in both scenarios when only one VF is instantiated, but it captures 50.8% of the 64-byte packets and 83.0% of the CAIDA ones when two VFs are used. Finally, HPCAP's performance falls in both cases: from 74.3% to 44.1% in the worst case and from 100% to 82.7% for the CAIDA trace.

IV. Application scenarios

In the light of the discussion and results presented along section III, we strongly recommend the usage of the VF-based approach for processing network-data. However, if there is a primal need for performance, users may find useful to connect their NICs and their VMs via PCI pass-through. We discourage the use of an emulated NIC configuration, as this would imply unnecessary redundant computation and limits the capture performance being the hypervisor the system's bottleneck. By using the VF approach, not only a reasonably high performance is achieved but also the traffic targeted to each VM is isolated and we acquire the ability of redirecting which may be interesting in a number of scenarios. Note that the migration process from a traditional (emulated NIC) configuration to any of the other two approaches only implies driver modifications in the host server and changing the VMs' default network interface for the new one (be it a physical one via pass-through or a virtual one).

Finally, we have identified three different usage scenarios in which the VF-based alternative would apply, which are also depicted in Fig. 5:

1) Users may run on their VM one of the network-processing applications provided by Intel DPDK or HPCAP, as in the Nth VM in Fig. 5.
2) Users may create their own network processing application based on the existing APIs, just as in VM3 in Fig. 5.
3) Users already running a set of VMs with a set of legacy network-related applications needing to monitor the traffic directed to those legacy applications. In this scenario a new VM could be created, adding the proper HW rules for re-directing the desired traffic to the new VF. This new VM could use a high-performance VF-aware driver with a monitoring application. This is depicted by VMs 1,2 and N in Fig. 5. The dashed lines are used to remark that this new VM could be dynamically created or destroyed without affecting the rest of VMs already running in the same host server. Importantly, this scenario would not require any changes in the legacy network applications.

V. Conclusions

The results obtained along the experiments presented in this work assess the feasibility of migrating the usage of high-performance packet capture engines in virtualized environments. We have discussed the different configurations by which a network application can be used inside a virtual machine, and obtained the performance bounds for each of those configurations. Moreover, we have given a set of guidelines that allow exploiting the functionality of generating virtual network functions, enabling a set of interesting scenarios. Differently from PCI pass-through, the use of VF allows end-users to scale in the amount of network applications running on a single hardware, with the consequent saving in terms of space, cooling and power consumption.

We have presented a set of solutions available for use under GNU Linux and, in the case of HPCAP and Intel DPDK, accessible as free software. As an additional contribution, we have developed HPCAPvf, a VF-aware version of HPCAP, offering a set of interesting features and capabilities. HPCAPvf may be used in any Linux distribution with kernel version newer than or equal to 2.6.32 without modifying nor kernel nor hardware configuration. Furthermore, we have made available the source code of HPCAPvf under a GPL license[1]. This study paves the way for future works involving advanced network actions in the VM side, such as packet storage, at high-speed rates.

Acknowledgements

This work has been carried out with the support of the Spanish National I+D *Packtrack* project (TEC2012-33754) and Universidad Autonoma de Madrid's multidisciplinary project *Implementacion de Modelos Computacionales Masivamente Paralelos* (CEMU-2013-14).

References

[1] F. Yu, R.H. Katz, and T.V. Lakshman. Gigabit rate packet pattern-matching using TCAM. In *Proceedings of IEEE Conference on Network Protocols*, 2004.

[1]https://github.com/hpcn-uam/HPCAP

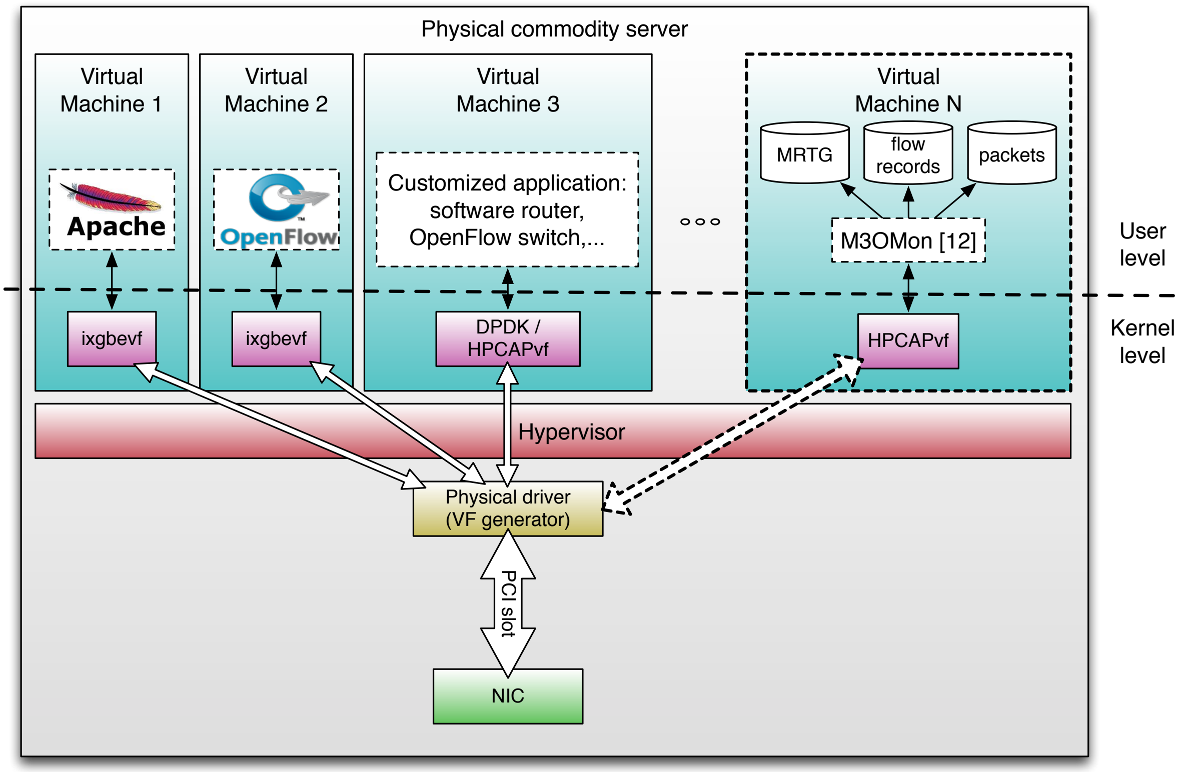

Fig. 5: Example usage cases

[2] M. Forconesi, G. Sutter, S. Lopez-Buedo, J.E. Lopez de Vergara, and J. Aracil. Bridging the gap between hardware and software open-source network developments. *IEEE Network*, 28(5), 2014.

[3] J.L. García-Dorado, F. Mata, J. Ramos, P.M. Santiago del Río, V. Moreno, and J. Aracil. High-performance network traffic processing systems using commodity hardware. In *Data Traffic Monitoring and Analysis*, volume 7754 of *Lecture Notes in Computer Science*, pages 3–27. Springer Berlin Heidelberg, 2013.

[4] AJ. Younge, R. Henschel, J.T. Brown, G. von Laszewski, J. Qiu, and G.C. Fox. Analysis of virtualization technologies for high performance computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 9–16, July 2011.

[5] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[6] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[7] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 206–217, New York, NY, USA, 2010. ACM.

[8] Intel. Intel Data Plane Development Kit (Intel DPDK) Release Notes, 2014. http://www.intel.com/content/dam/www/public/us/en/documents/release-notes/intel-dpdk-release-notes.pdf, [1 October 2014].

[9] W. Wenji, P. DeMar, and M. Crawford. Why can some advanced Ethernet NICs cause packet reordering? *IEEE Communications Letters*, 15(2):253–255, 2011.

[10] V. Moreno, P.M. Santiago del Rio, J. Ramos, J. Garnica, and J.L. Garcia-Dorado. Batch to the Future: Analyzing Timestamp Accuracy of High-Performance Packet I/O Engines. *Communications Letters, IEEE*, 16(11):1888 –1891, november 2012.

[11] C. Walsworth, E. Aben, k.c. Claffy, and D. Andersen. The CAIDA anonymized Internet traces 2014 dataset. http://www.caida.org/data/passive/passive_2014_dataset.xml, [1 October 2014].

[12] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gomez-Arribas, and J. Aracil. Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems. *International Journal of Network Management*, 24(4):221–234, 2014.

[13] A.O. Kudryavtsev, V.K. Koshelev, and A.I. Avetisyan. Prospects for virtualization of high-performance x64 systems. *Programming and Computer Software*, 39(6):285–294, 2013.

[14] L. Rizzo, G. Lettieri, and 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*, ANCS '13, pages 47–58, Piscataway, NJ, USA, 2013. IEEE Press.

[15] C.T. Yang, J.C. Liu, H.Y. Wang, and C.H. Hsu. Implementation of gpu virtualization using pci pass-through mechanism. *The Journal of Supercomputing*, 68(1):183–213, 2014.