Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration

Conference Paper · November 2015

DOI: 10.1109/NFV-SDN.2015.7387409

8 authors, including:

Michail Alexandros Kourtis
National Center for Scientific Research Demokritos
61 PUBLICATIONS **478** CITATIONS
SEE PROFILE

George Xilouris
National Center for Scientific Research Demokritos
106 PUBLICATIONS **755** CITATIONS
SEE PROFILE

Vincenzo Riccobene
Intel
11 PUBLICATIONS **158** CITATIONS
SEE PROFILE

Michael J Mcgrath
Intel
54 PUBLICATIONS **1,258** CITATIONS
SEE PROFILE

Enhancing VNF Performance by Exploiting SR-IOV and DPDK Packet Processing Acceleration

Michail-Alexandros Kourtis[1,2], Georgios Xilouris[2], Vincenzo Riccobene[3], Michael J. McGrath[3], Giuseppe Petralia[3], Harilaos Koumaras[2], Georgios Gardikis[4], Fidel Liberal[1]

[1]Department of Communications Engineering, University of the Basque Country (UPV/EHU), Bilbao, Spain,

[2]Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece

[3]Intel Labs Europe, Leixlip, Co. Kildare, Ireland

[4]R&D Department, Space Hellas S.A. Athens, Greece

***Abstract*— The primary goal of Network Function Virtualization (NFV) is the migration of physical network functions to software versions running on virtual machines (VM) in cloud computing environments. Rapid developments in virtualization technologies have made high-speed network connections and line rate packet processing viable in virtualized infrastructure environments. Deep Packet Inspection (DPI) of network traffic in the form of a computational intensive virtualized network function (VNF) was selected as a representative use-case. The DPI use case was used, to demonstrate the benefits of using SR-IOV enabled devices with DPDK to support performant Virtual Network Function (VNF) deployments.. Performance evaluation of VNF versions using LibPCAP, SR-IOV and DPDK have been carried out. The results demonstrate that significantly higher packet throughput performance can be achieved when using SR-IOV and DPDK in unison in comparison to packet processing with the native Linux kernel network stack.**

Keywords—SR-IOV, DPDK,NFV,VNF,LibPCAP, DPI

I. INTRODUCTION

The proliferation of software defined networking (SDN) and the rollout of network function virtualization (NFV) has attracted significant attention from the Telecommunications domain with the promise of a new network design approach. NFV's network model is focused on the implementation of software based network functions that run on virtualized infrastructure. This concept enables multiple virtualized network functions to operate on the same high-volume server simultaneously. As a result, the operator gains significant flexibility and agility, by having the ability to dynamically start, reconfigure and terminate VNFs. However, in order to deliver these expectations a significant number of considerations have to be taken into account. In conventional network devices the software component is highly optimized for its custom hardware platform. It is this close connection between the software and hardware element which provides the expected performance and reliability for carrier-grade network functions.

When VNFs are deployed on Virtual Machines (VMs) hosted in a cloud infrastructure, for example OpenStack, the amount of network traffic that a single server appliance is expected to handle is increased significantly. When multiple VNFs run simultaneously and each one of them processes a significant number of packets per second, significant performance challenges may be generated. The system's architecture features such as memory access, task and resource allocations, network I/O etc. can have significant influence on performance.

As I/O performance is critical in cloud infrastructures, virtualization optimizations are required in order to maximize the utilization of computer system resources. The Single Root I/O Virtualization (SR-IOV) [1] is a specification released by the PCI-SIG, which defines hardware enhancements that reduce hypervisor's interactions with a VM, in order to improve its data processing performance. An SR-IOV enabled device is capable of spawning various "light" instances of PCI functions, named Virtual Functions (VFs). Each VF can be pinned to a VM, granting direct access to its physical resources. Exploiting this feature, multiple VMs can share the same physical PCI device resources, thus achieving high performance with no significant additional CPU overhead. As SR-IOV provides direct access and control to the system's hardware resources, it enables the efficient allocation of low-level network resources.

In order to fully exploit the system's resources, both in the network and computational domains, and at the same time enhance and facilitate the implementation of intensive network applications, Intel has developed the Data Plane Development Kit (Intel® DPDK) [2]. DPDK comprises of a set of libraries that support efficient implementations of network functions through access to the system's network interface card (NIC). DPDK offers network function developers a set of tools to build high speed data plane applications. DPDK operates in polling mode for packet processing, instead of the default interrupt mode. The polling mode operation adopts the busy-wait technique, continuously checking for state changes in the network interface. This mitigates interruption in packet processing, as it bypasses the kernel, efficiently consuming CPU cycles, which leads to increased packet throughput [3]. Using DPDK network packet ingress and egress is faster in comparison to the standard Linux kernel network stack as applications are supported in userspace, thus bypassing kernel network stack bottlenecks.

In this paper, a network and computational intensive use-case has been selected, in order to demonstrate the benefits of using SR-IOV enabled devices with DPDK to support performant VNF deployments. Deep Packet Inspection (DPI) of network traffic was selected as a viable and suitable use case. The required packet analysis and flow information processes consume a significant amount of computing resources, as all

network flows and most of their packets need to be processed deeper than protocol headers, in many cases at payload level, to provide accurate and precise packet identification of network traffic. DPI is primarily used to identify network traffic profiles for various security, or network management purposes. DPI analyzes IP traffic from Layers 2 to 7 including headers and data protocol structures together with the payload of the packet's message. This information is used to identify the various application protocols and traffic flows. Collection of this information can be used to provide a traffic classification of the network being monitored. Various commercial DPI solutions exist from companies such as WindRiver [4], PACE [5] and Qosmos [6]. The Qosmos DPI offers a virtualized solution built on Qosmos' flagship product ixEngine, which has established itself as the de facto industry-standard DPI engine for developers of telecoms and enterprise solutions. An alternative free open-source solution is nDPI [7]. NDPI is an ntop-maintained superset of the popular OpenDPI library. NDPI has performed better, compared to other open source DPI libraries and Cisco's NBAR, in extensive traffic recognition experiments performed in [8].

The DPI solution described in this paper is built using the nDPI library. This approach uses only an indicative, small number of initial packets from each flow in order to identify the payload content and does not inspect each packet. In this respect it follows a Packet Based per Flow State (PBFS). This method uses a table to track each session based on the 5-tuples (source address, destination address, source port, destination port, and the transport protocol) that is maintained for each flow.

The paper is organized as follows: Section II outlines the problem statement, Section III presents the detailed architecture and solution implementation, Section IV evaluates and presents the performance results of the proposed solution. Section V extends the discussion on automatic VNF deployment and enhanced platform awareness in current virtualized infrastructures. Finally, Section VI concludes the paper and describes the next steps for this research work.

II. Problem Statement

Cloud infrastructure solutions such as OpenStack, run on top of a broad range of network components offered by current operating systems, such as routing, filtering, flow reconstruction, etc. These capabilities generate significant overhead, resulting in performance penalties, as each packet passes through the OS kernel and is not directly accessible to userspace. The Linux network stack, which is commonly used as a basis for cloud networking solutions, sets as its primary goal the provision of a general purpose network stack for a fully functional operating system. Therefore, a standard Linux network stack cannot scale to the performance level required for a software router application. In general, OpenStack's networking service Neutron is essentially built upon an Open vSwitch (OVS) instance, which consists of various native OVS virtual ports, TAP virtual network kernel interfaces and kernel bridges connecting them, for each VM instance. This multi-bridge virtual interface setup, introduces substantial delays in packet forwarding performance as every packet must pass through and be processed by the kernel multiple times [9], before it reaches its final destination. With this form of architectural design the operating system's kernel performance quickly becomes a bottleneck.

III. Description and Implementation

Two setup configurations were utilized for performance testing of the traffic classifier VNF. Both test deployments were performed using servers with Intel® Xeon® E5-2620 v3 @ 2.40GHz CPUs. Each server had a dual port 10Gbit Intel® Ethernet Converged Network Adapter X520-T2. In both configurations one server was used as a traffic generator and the second server hosted the DPI application. The first test was performed at the physical layer meaning the DPI application was executed directly using the physical NICs, whereas in the second test the DPI application was deployed as VM through a KVM hypervisor with the packets arriving through an SR-IOV fast path. In both performance tests two versions of the DPI application were evaluated one using LibPCAP and one using DPDK. The purpose of the first configuration was to benchmark and set a baseline performance for the DPDK enhanced DPI application. Additionally the test evaluated the cost overhead of virtualization on performance. Amongst various packet acceleration candidate frameworks Netmap [10], PF_RING [11], and DPDK that provide zero-copy, kernel bypass, and support for multi-queuing capabilities, Intel® DPDK was selected to implement the accelerated version of the VNF, as it provides advanced user-level functionalities, such as a multi-core framework with enhanced NUMA-awareness, and libraries for packet manipulation across different cores. The DPDK libraries also provide two execution models: a pipeline model where one core has the capability of transferring packets from the receiving NIC to a second core for processing, and the run-to-completion model where packets are distributed among all cores for accelerated processing. The features of packet scheduling and the different execution models make the DPDK framework more than just and I/O library. In the current VNF implementation only a baseline set of features have been used from the DPDK libraries, as more complex packet processing algorithms using DPDK capabilities will be investigated in future work.

A. VM testbed with SR-IOV and DPDK

In figures 1 and 2 a detailed overview of the two configuration setups are shown. Figure 1 shows the baseline setup, where no additional modifications were made to enhance the performance of the DPI application. In both the physical NIC and virtual NIC driver layers the packets are handled by the Linux kernel network stack. The DPI application in both tests (physical and virtualized), uses the LibPCAP in order to read and analyze the network traffic received.

Figure 2 shows the enhanced architecture where SR-IOV has been enabled at the physical NIC of the host server, the corresponding VF driver has been attached to the DPI virtual machine, and finally the virtual NIC is loaded with the DPDK driver for faster NIC-userspace communication. In this case the DPI application reads and processes the packets received using the DPDK framework, in both the physical and virtualized experiment.

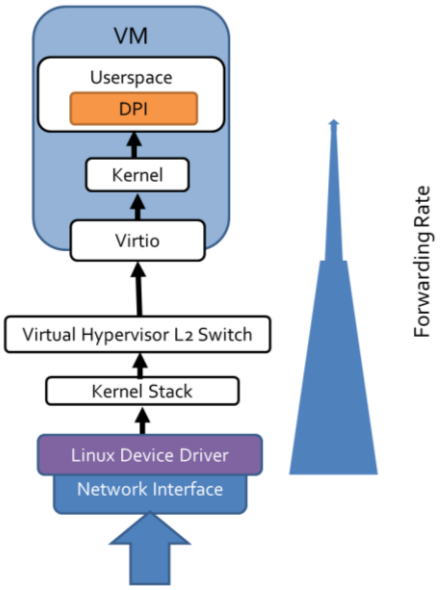

Fig. 1. Standard Setup

Figure 2 shows SR-IOV enablement at the physical NIC of the host server. A corresponding VF driver is attached to the DPI virtual machine. The virtual NIC is loaded with the DPDK driver for faster NIC-userspace communication. In this configuration the DPI application reads and processes the packets received using the DPDK framework, in both the physical and virtualized experiments.

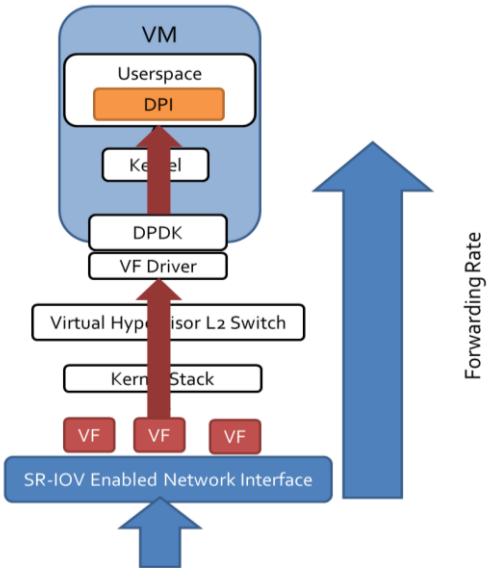

Fig. 2. Setup with SR-IOV enabled ports and DPDK on the VM.

The primary goal of these experiments was to evaluate and quantify the performance improvements in a VNF enabled by SR-IOV and DPDK compared to the standard Linux network stack. Additionally, the purpose of the evaluation tests was to quantify the cost of virtualization in a computational intensive network function, such as a DPI.

For the evaluation and benchmarking of the DPI VNF the DPDK PktGen [12] traffic generator was used, which is built on top of the DPDK fast packet processing framework in order to achieve line-rate traffic generation. The PktGen is capable of generating 10Gbps rate of traffic even with 64 byte frames. For the purpose of the experiments PCAP traffic replay based, on a data file, was used and retransmitted at a 10Gbps rate. Real traffic traces captured in NCSRD were used to generate the PCAP file.

Figure 3 shows the distribution of the packet sizes. As it can be clearly seen, most packets belong to the size group of 1280-1518 bytes, and the second most popular size group is the 40-79 bytes category. Analysis of the PCAP file indicates that the DPI is exposed to a rich set of traffic types and protocols which ensured a realistic test scenario. In total the captured PCAP traffic file consists of 2343 unique flows and 28 different application protocols.

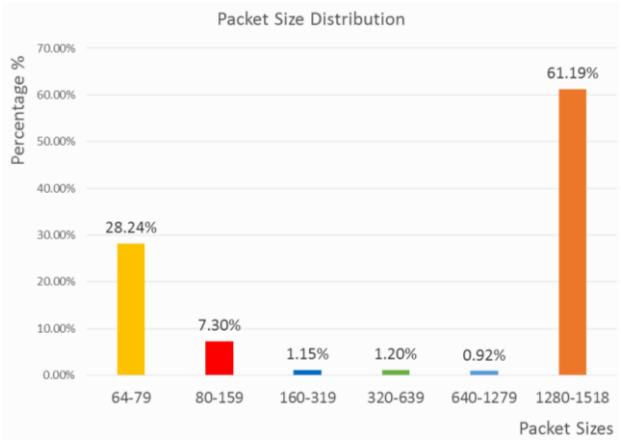

Fig. 3. Packet Size Distribution Histogram.

B. *nDPI integration with DPDK*

The standard implementation of nDPI is built on top of the packet processing library LibPCAP. As the interface receives packets, nDPI can extract the packet's timestamp from the pcap header. The pcap header is then removed in order to access and read the content of the packet. DPDK uses its own primitive types to store and access data, however conversion is required in order for the received packet to be DPDK-compatible. Conversions included network types, e.g. IP header types, as well as primitive types, e.g. unsigned char types, in order to fully exploit the user level capabilities of the DPDK framework.

Apart from the type conversion and porting to DPDK, a top-down redesign of the nDPI packet processing has been carried out, in order to bring the DPDK plane to a layer that nDPI recognizes. The standard nDPI library is mainly designed to function properly with LibPCAP, in terms of accessing network packet data. Various modifications on the nDPI end were made in order to facilitate the integration with the DPDK framework.

C. *VM Interconnection*

Service Function Chaining (SFC), traffic forwarding and inter-VM communications are required for the VNF to function correctly. However functional integration of the VNF into OpenStack's networking environment, and more specifically with the Neutron service is non-trivial and remains to be implemented as Neutron currently offers limited abilities to

support arbitrary traffic steering. Future enhancements to this work will include the integration of the current work in an automated, flexible and efficient network appliance for virtualized network infrastructures, as both DPDK and SR-IOV currently require various complex system configurations to function properly and provide improved performance.

In order to support direct traffic forwarding, i.e. the virtual network interface of one Virtual Network Function Component (VNFC) is directly connected to the virtual network interface of another VNFC's, modification of Neutron's OVS is required. Each virtual network interface of a VNFC is reflected upon one TAP-virtual network kernel device, a virtual port on Neutron's OVS, and a virtual bridge connecting them. Packets travel from the VNFC to Neutron's OVS through the Linux kernel. The virtual kernel bridges of the two VNFCs need to be shut down and removed, then an OVSDB rule needs to be applied at the Neutron OVS. This rule defines an all-forwarding policy between the OVS ports of the corresponding VNFCs.

IV. Experimental Results

This section presents the results of comparison tests between DPDK and LibPCAP versions of the DPI in physical and virtualized environments. The traffic for both experimental evaluations was generated by a traffic generator in a linear manner from 1 to 100%, with 1% increments per second. Traffic statistics were collected from the VNF every second and were post processed for performance evaluation.

A comparison of the packet processing performance of the LibPCAP based deployment of the DPI versus the DPDK accelerated version of the VNF under a bare metal scenario is shown in Figure 4. The results clearly show that the DPI's performance is significantly improved when DPDK is used to accelerate packet processing. The LibPCAP version exhibited saturation at approximate 1Gbps. This throughput compares poorly in comparison to the approximately line rate performance of the DPDK accelerated version.

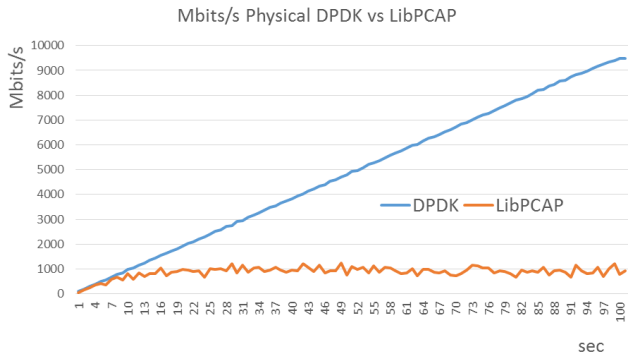

Fig. 4. Physical Testbed DPDK vs LibPcap.

A second set of experiments focused on identifying the effect of combining SR-IOV and DPDK on DPI performance when deployed as a VM. A linear scaling network traffic load up to 10Gbps was used to stress test both the LibPCAP and SR-IOV/DPDK versions of the DPI VM.

As shown in Figure 5 the SR-IOV/DPDK version achieves approximately 81% of the physical DPDK testbed packet transmission performance. The LibPCAP version displayed saturation effect at 1Gbps with an 87.5% throughput reduction in comparison to the DPDK version. The results also indicate that DPDK's performance in the virtualized scenario is degraded, approximately 19% in comparison to the corresponding physical test. This performance degradation can be explained as DPDK runs on a virtualized environment, and does not really bind a CPU core, so thread/CPU isolation is not really performed.

The results indicate the promising performance of the virtualized DPI solution, and a clear improvement when DPDK is utilizing. The gap in performance between the physical and the virtualized solution shows further optimization is required in order for the VNF to achieve performance close to the corresponding physical DPI solution i.e. line rate. Additionally, it is clear that despite the use of SR-IOV the network kernel stack remains the bottleneck in the packet processing path.

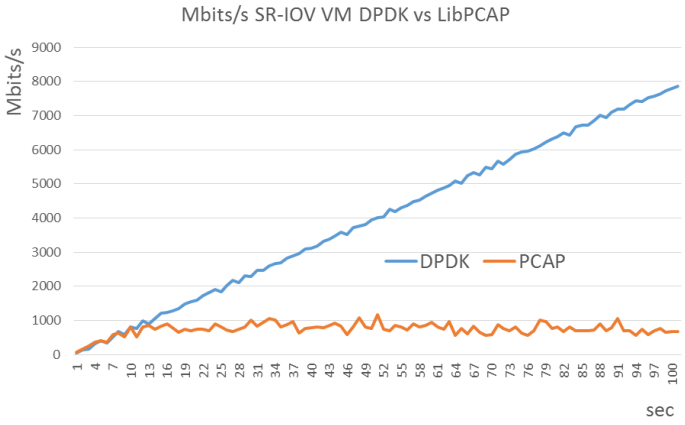

Fig. 5. Virtual Machine over SR-IOV Testbed DPDK vs LibPcap.

V. Considerations on Automatic Deployment

The previous section provides evidence that packet acceleration technology solutions and their appropriate integration can provide comparable performance for virtualized instantiations of real physical equipment running network functions (e.g. middle boxes). However, current virtualized infrastructure environments for hosting VNFs pose a number of management and configuration challenges. . For example with OpenStack which has established itself as the de-facto open source solution for the implementation and management of cloud infrastructures, significant manual configuration is required in order to create the required service chain links between VNFs correctly and in a performant manner.

However, assuming that all the pre and post configurations considerations have been addressed, the VNF implementation should not be agnostic to the hosting platform capabilities. Essentially, features such as DPDK need to be considered carefully during the development of a VNF as well during testing before deployment in a real world environment. Additionally efforts to enhance the platform awareness of OpenStack and other cloud frameworks are gaining momentum. Increased platform awareness will support appropriate and automated mapping of VNFs to the most appropriate compute nodes that have required platform features.

Due to DPDK's mode of operation within a VM, or on bare metal, scaling mechanisms that depend on CPU usage monitoring are incapable of triggering correct decisions. This is due to the fact that when DPDK is used in a VNF, utilization for a CPU core reaches the cap of 100% regardless of the traffic load on the network interface (virtual or not). Additionally, when DPDK is used the NIC is no longer recognized by the Linux kernel stack. Therefore, the NIC has no TCP/IP stack, as the kernel is bypassed. Also, a DPDK-enabled NIC cannot respond to ICMP requests, and is therefore not discoverable from a Layer 3 network perspective. This adds some additional overhead to the development of a DPDK-enabled VNF. These are acceptable drawbacks in order to achieve significantly improved performance, however auto-scaling mechanisms inherent in the OpenStack framework or even manual ones will fail to scale correctly. The solution is to use VNF specific monitoring information, internal to the VNF and to create new rules and methods for scaling the VNF when required.

Based on the previous considerations appropriate selection of data-path acceleration features for VNFs is a trade-off with respect to infrastructure and management flexibility. This selection affects various actors to varying extents in the NFV chain. In summary:

- The Function Developer
  - VNF code development (code should be factored appropriately to support DPDK). This requirement has limited risk as DPDK offers a large library of methods in order to serve integration and porting of a number of network functions.
  - VNF development environment: consideration of enhanced platform features during the development process can be challenging due to the heterogeneity of the operational environments where the VNF will run.
- The Infrastructure Provider
  - Vendor lock-in and cost. Procurement of equipment that provide the enhanced capabilities. For example VNFs which require specialized PCIe co-processing cards (e.g. FPGAs) to provide specific packet processing capabilities can potentially result in a degree of vendor dependency. However, as the cloud ecosystem is primarily based around technology standards vendor lock-in is less of an issue in comparison to other technology areas.
  - Provision of platform information to an Orchestrator, in order to achieve the best mapping of application types to resource allocation among an NFVI-Point of Presence (PoP) compute nodes.
- The Service Provider
  - Achieving support at an Orchestration level of the enhanced platform features (i.e. infrastructure repository, service mapping modules) in order to map VNF required resources and capabilities to the underlying infrastructure.

VI. Conclusions and Future Work

A virtualized DPI which supports high-speed network packet processing has been described. The VNF in various instantiations has been benchmarked to evaluate its performance. The DPI VNF was implemented both using LibPCAP and Intel's DPDK framework. The SR-IOV framework was also used, in order to maximize packet throughput in a virtualized environment. Additionally, various considerations and discussions have been presented regarding the automatic deployment of enhanced VNFs in cloud environments. Various limitations and development restraints have been described and analyzed. The results of the performance evaluation results showed that with SR-IOV and DPDK a significantly higher performance can be achieved compared to packet processing with the Linux kernel network stack. Additional work is required to close the remaining gap with the SR-IOV/DPDK enabled version of the DPI and 10Gbps line rate performance. The next steps in this work will focus on NUMA and core pinning and optimization of huge page sizes in an effort to achieve near line rate performance for the DPI.

Acknowledgment

This work was undertaken under the Information Communication Technologies, EU FP7 T-NOVA project, which is partially funded by the European Commission under the grant 619520.

References

[1] SR-IOV. PCI Special Interest Group, http://www.pcisig.com/home

[2] DPDK.org. (2015). DPDK: Data Plane Development Kit. Available: http://dpdk.org/

[3] K. Salah, A. Qahtan, "Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme," Computer Communications, vol..32 no..1, pp.179-188, January, 2009

[4] Wind River, "Wind River Content Inspection Engine" on-line: http://www.windriver.com/products/product-overviews/PO_Wind-River-Content-Inspection-Engine.pdf

[5] IPOQUE, "Protocol and Application Classification with Metadata Extraction", on-line: http://www.ipoque.com/en/products/pace

[6] http://www.qosmos.com

[7] L. Deri,,M.Martinelli, T. Bujlow, A. Cardigliano, "nDPI: Open-Source High-Speed Deep Packet Inspection", International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 617-622, 4-8th Aug. 2014.

[8] T. Bujlow, V. Carela-Español, P. Barlet-Ros, (2014). Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification. Universitat Politècnica de Catalunya.

[9] https://www.hastexo.com/system/files/neutron_packet_flows-notes-handout.pdf

[10] L. Rizzo, "netmap: a novel framework for fast packet I/O," in USENIX Annual Technical Conference, April 2012

[11] "PF_RING ZC," http://www.ntop.org/products/pf_ring/pf_ring-zc-zero-copy/,.

[12] PktGen, Pktgen version 2.7.7 using DPDK-1.7.1, available https://github.com/Pktgen/Pktgen-DPDK/