NFPA: Network Function Performance Analyzer

Levente Csikor, Márk Szalay, Balázs Sonkoly, and László Toka
High Speed Networks Laboratory, Budapest University of Technology and Economics
Email: {csikor, szalay, sonkoly, toka}@tmit.bme.hu

***Abstract*—With the soar of Software Defined Networking planning a network service becomes harder of a task than ever before. Selecting traditional network elements that provide the best value for money given the performance requirements and the allocated budget is not the only option today: one might also take the software solution on generic hardware alternative. The problem is that the set of available solutions and the possible combinations of software and hardware components in this nowadays' alternative is frustratingly vast while the decision maker lacks any clear benchmarking comparison between the existing options. Our solution presented in this paper provides an answer to this critical need: we propose a benchmarking tool that allows the user to measure the important performance metrics of any network function realized on any hardware and software combination, and then to compare the results on a web interface with those of all the setups collected in our database.**

Keywords*—SDN, NFV, VNF, benchmarking tool, performance analyzer

I. Introduction

It is commonly accepted that decoupling network functions (NFs) from the underlying physical infrastructure using virtualization and cloud technologies enables service innovation. By means of Software Defined Networking (SDN), the programmability of the Network Function Virtualization (NFV) infrastructure has gained a huge potential for supporting the deployment of NFs in a variety of (virtualized) environments, including Internet and cloud service providers, campus and enterprise networks, and over-the-top applications. However, the widely spread usage of SDN is still to come: this promising new approach of networking has not yet been deployed ubiquitously for several plausible reasons. First, most commercial off-the-shelf devices are not fully compliant with OpenFlow, therefore deploying services over OpenFlow in carrier-grade networks is not always feasible. Second, although relying on (inexpensive) generic platforms and software-based solutions (as in data centers) first looks promising for faster adaptation of the latest standards, it is not obvious how a certain service chain should be embedded or mapped to a set of heterogeneous networking nodes [1], [2], [3]. Finally, the main fundamental question that always arises is: *are the software-based solutions (ever going to be) able to cope with the forever changing and increasing traffic demands?*

Traditional black box networking assigned the responsibility for performance to the box vendors with SLAs. Now, potential adopters of the SDN paradigm worry about the fact that in a virtualized system the responsibility for quality is not so clear. What is more, there is limited information available about the performance of software-based NF implementations. Although each solution promises to outperform others, the details of their performance evaluation, e.g., the testing environment of hardware and software components and properly installed drivers, remain unrevealed, e.g., in [4]. Only a limited number of independent benchmarking tools have appeared so far and even those lack the necessary generality, e.g., project Yardstick [5] aims at describing Virtual Network Function (VNF) performance with a number of metrics, but it is limited to Openstack-based clouds. We believe that a NF benchmarking tool should not be limited to only a subset of available technologies, it is crucial to explore possibly the whole range of all the component levels: evaluate hardware accelerators, e.g., programmable NICs, Intel DPDK [6], Netmap [7], Open DataPlane [8], pinpoint software bottlenecks, e.g., packet I/O, OS kernel, test the effects of virtualization, etc. Furthermore, it is also important to analyze the packet processing performance of the data plane under realistic traffic (captured or emulated traces). Thus, we propose our Network Function Performance Analyzer (NFPA), which is not only in accordance with standardized methodologies (RFC 2544, [9]), but also makes possible to comprehensively compare performance metrics of NFs in an exhaustive range of dimensions. More precisely, NFPA answers

- how a *software-based NF*
- implemented in a *generic language* (e.g., C/C++)
- running on a *generic platform* (e.g., Intel Xeon)
- over a *generic operating system* (e.g., Linux)
- in *different environments* (e.g., virtual machine)
- using *different drivers* performs
- under *different traffic patterns*.

The rest of this paper is organized as follows. In Sec. II, we give a detailed description of the proposed framework, while in Sec. III we show some results of various measurement setups. Finally, in Sec. IV we conclude our work.

II. Architecture

Complying with RFC 2544, our NFPA is a standalone benchmarking tool, connected to the Device Under Test (DUT), as depicted in Fig. 1: the NF to be tested runs in the desired environment (NF node), e.g., in a KVM virtual machine, while NFPA runs on a separate machine. First, the user determines the measurement target on the NFPA node by all its relevant parameters, e.g., details of the hardware and software components, number of repeated measurements and their duration, and selects the traffic traces to be used. Afterwards, NFPA sends packets on port 0 (anwithd also on port 1 in case of bidirectional measurements), while receives (a portion of those) packets on port 1 (and also on port 0 respectively), then it calculates the throughput of the DUT in terms of packet/s and bit/s. Once the measurement is finished, NFPA analyzes the results and plots them in the preset units, and in parallel saves the measurement data in a local database. In addition, we provide a central server with a web-based API, whereby the NFPA user can upload and share the results for

978-1-4673-6884-1/15/$31.00 ©2015 IEEE

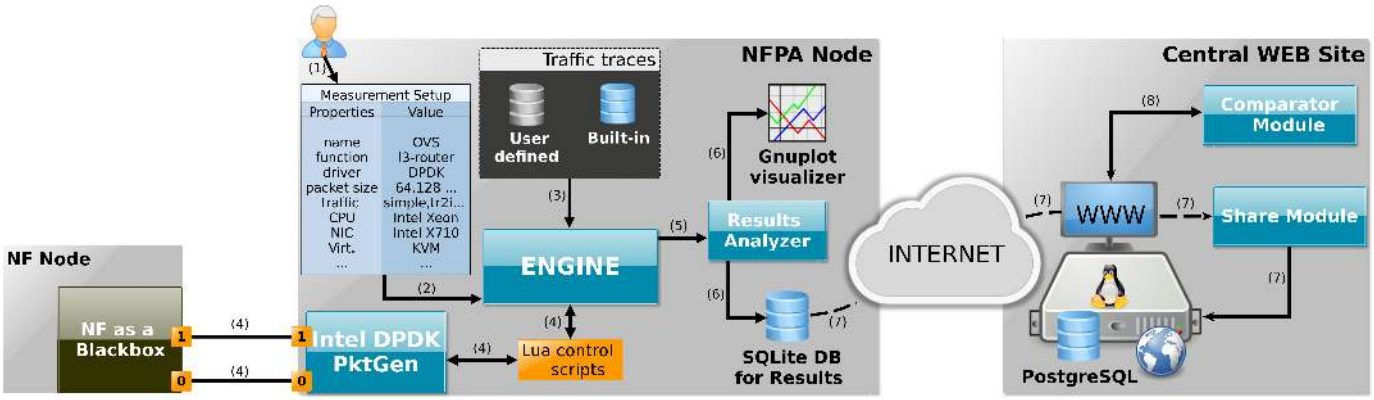

Fig. 1. The system architecture and the applied method. Note the numbers in parentheses that mark the sequence of the measurement procedure.

further comparison, and in return the user can browse the set of results measured by others.

A. Platform and Applied Technologies

NFPA's engine is implemented in Python and relies on standard libraries. It can be configured both by config files and via a lightweight web GUI. In order to avoid the limitation of kernel space network card drivers, NFPA's network interface is built on Intel's DataPlane Development Kit (DPDK, [6]) for fast packet processing: in particular, for sending and receiving traffic, NFPA uses PktGen[1] with custom Lua scripts for parameterizing, automating and controlling the measurements. In order to support portability, the results are stored in a local SQLite database. Result charts are created using Gnuplot[2]. The overall set of measurements at our central node is stored in PostgreSQL database.

B. Traffic Traces

The characteristics of the traffic traces that we use as inputs influence the measurement results on a NF's performance. Naturally, packet sizes and the number of flows in the trace play an important role, e.g., the smaller the packet size the lower the throughput assuming a quasi-constant packet/s performance. Specific NFs, however, require even higher caution when crafting the input trace. For instance, when performance testing an Open vSwitch (OVS[3]) programmed to function as a VxLAN gateway[4], not only the packet headers must be heterogeneous, e.g., the source and destination IP addresses, L2 informations, VLAN ids, but also the characteristics of the traffic patterns on the sending and receiving ports need to be different. NFPA, therefore, provides a wide selection of synthetic traffic traces with different packet headers and sizes in order to approximate realistic scenarios as closely as possible. A brief overview of those is the following:

- *simple*: PktGen's default settings where each flow has the same header information.
- *tr2e*: 100 different flows with each flow having different destination MAC address.
- *tr2i*: 100 different flows with each flow having different destination IP address.
- *tr3e*: 10.000 different flows with each flow having different destination MAC address.
- *tr3i*: 10.000 different flows with each flow having different destination IP address.
- *tr4e*: 500.000 different flows with each flow having different destination MAC address.
- *tr5ul*: 64.500 different flows with each flow having different source IP and 6.450 of them have different destination IP addresses.
- *tr5ul*: 64.500 different flows with each flow having 6450 different source IP addresses and a certain destination IP address.

Traffic traces ending with "e" and "i" can be used for measuring the throughput of L2-switches and L3-routers, respectively. Traffic traces are available for the following packet sizes: 64, 128, 256, 512, 1024, 1280, 1500 bytes.

Besides the more than 100 different pre-generated synthetic traffic traces, NFPA is capable of using any *pcap* trace the user has or obtains from, e.g., Caida [10].

III. Preliminary Findings

We have set up some scenarios and carried out several measurements with NFPA. Here, we show how the well-known OVS performs in different virtualized environments compared to another emerging software-based switch, called xDPd (extensible DataPath daemon)[5].

First, let us demonstrate how the performance of OVS is affected by different drivers (kernel, DPDK) and environments (native, Kernel-based Virtual Machine (KVM), Linux Container (LXC)) assuming a simple port forward flow-rule installed. In this case, OVS was running on an Intel Xeon E5-2620 (2 GHz) server with an Intel X710 10Gb Ethernet interface. The measurement details are depicted in Fig. 2, where the red solid line shows the theoretical maximum number of packets that can be transferred in one second on a 10Gbit/s Ethernet port. The other lines are denoted by *X_Y*, where *X* yields the used driver and *Y* indicates the type of virtualization. One can observe that the performance of OVS running natively or in a lightweight Linux container (LXC)

[1] http://dpdk.org/browse/apps/pktgen-dpdk/refs/
[2] http://www.gnuplot.info/
[3] http://openvswitch.org/
[4] A VxLAN gateway allows a virtual extensible LAN to communicate with another network, particularly a virtual LAN.
[5] http://www.xdpd.org/

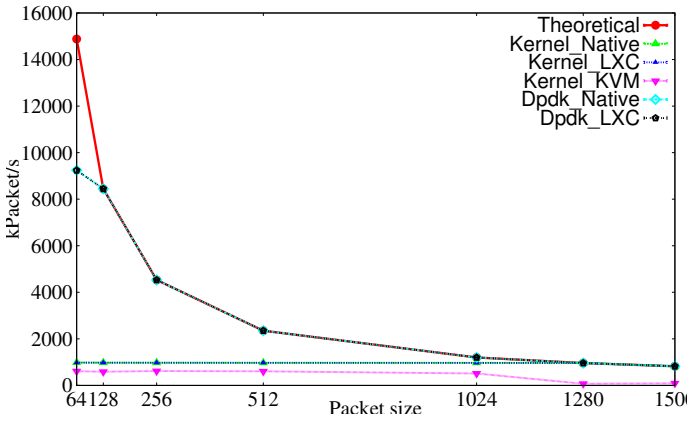

Fig. 2. Open vSwitch's performance over different drivers and environments

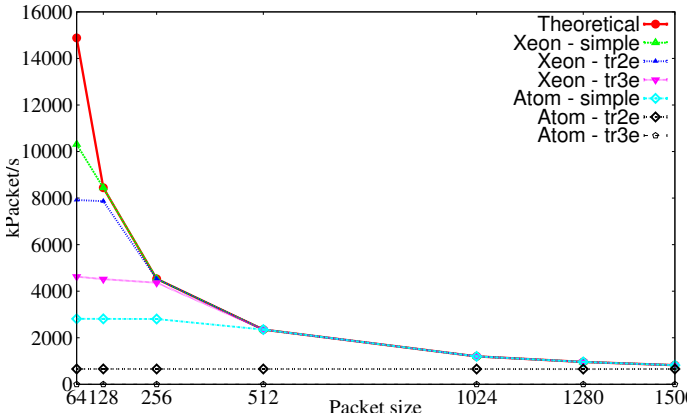

Fig. 3. xDPd's performance on different CPUs and with various traffic traces

hardly differ, while KVM degrades the throughput[6]. More importantly, it is noticeable that if OVS uses Intel's DPDK driver for forwarding then maximal throughput can be achieved already with 128-byte packets, but in case of kernel drivers it is reached when the packet size is greater than 1280 bytes. Although not shown in this particular example, we have also learned that due to the flow cache method of OVS, if different flows are sent, i.e., addresses in packet headers differ, then the performance significantly reduces, even if a simple port forward rule is installed.

Second, we show to what extent the performance of a DPDK enabled xDPd is affected by different platforms and traffic traces. We used *simple*, *tr2e*, and *tr3e* traffic traces mentioned above. xDPd was first running on the Intel Xeon server as in the above-mentioned OVS scenario, and then on an Intel Atom C2758 (2.4 GHz) with an Intel 82599 10Gb Ethernet interface, respectively. Proper flow rules were installed in the flow table according to the traces. The results are depicted in Fig. 3, where again the red solid line shows the theoretically maximal throughput. One can observe that in case of the Xeon platform and simple port forwarding scenario the DPDK enabled xDPd performs similarly to OVS, i.e., it achieves the maximal throughput immediately as the packet size reaches 128 bytes. On the other hand, the results indicate that in case of an Atom-based server the average throughput reaches the theoretical values only when the packet size is greater than 512 bytes. Furthermore, one can easily see that the smaller the packet is, the more the performance is decreased due to the number of different flows, e.g., in case of 64-byte packets, the throughput from ~ 10 million packet/s reduces to ~ 8 Mpps if 100 different packets are sent instead of sending the same packet every time.

[6]Note that in case of KVM the performance significantly depends on the way the virtual interfaces are connected to physical ones by e.g., linux bridge, Nat, macvtap-passthrough, ivshmem, userspace vhost.

IV. Conclusion

The importance of fully understanding the performance range of each software-based NF is essential for creating and operating reliable networks and services. Additional metrics beyond link speed, such as packets per second, connections per second, flow modifications per second, one-way delay will be crucial to develop and deploy new network services. With this demo, we make the first step in this direction and propose a publicly available measurement tool that not only measures the most important properties, but also provides an API for analyzing and comparing measurements carried out by others. We believe that such a huge set of measurement data could provide a very basis for further more complex benchmarking, e.g., the ones mentioned in [11]. Since NFPA is open-source, it can be easily extended with additional features to examine other performance metrics too.

During the demo, we showcase[7] each important part of the NFPA architecture with the steps depicted in Fig. 1. The audience can access our cloud (1), start a NF (2) and measure the throughput via NFPA (3). After the measurements are done, results will be accessible through our central web page and comparisons could be made in a straightforward manner.

Acknowledgements This work has been supported by Ericsson. L. Toka was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

[1] M. Rost, S. Schmid, and A. Feldmann, "It's about time: On optimal virtual network embeddings under temporal flexibilities," in *IEEE IPDPS*, 2014, pp. 17–26.

[2] S. Zhang, Z. Qian, B. Tang, J. Wu, and S. Lu, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *IEEE INFOCOM*, 2012.

[3] Y. Y. M. Yu, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," in *ACM SIGCOMM CCR*, vol. 38, no. 2, Apr. 2008, pp. 17–29.

[4] Cloud Networking Performance Lab, "A blazingly fast software switch, an overview in our high-performance switch with our extensions." http://cnp.neclab.eu/vale, 2014.

[5] H. Feldt, "Yardstick: Prototype architecture and status," presentation: https://wiki.opnfv.org/yardstick, May 2015.

[6] Intel, "Guide: Data plane development kit for linux," Guide, April 2015.

[7] L. Rizzo, "netmap: A novel framework for fast packet i/o," in *USENIX Security Symposium*, Aug. 2012, pp. 101–112.

[8] Linaro Networking Group, "Opendataplane introduction and overview," January 2014.

[9] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices," RFC 2544, March 1999.

[10] CAIDA, "Caida data - overview of datasets, monitors, and reports," http://www.caida.org/data/overview/, Last access: Aug 2015.

[11] A. Morton, "Considerations for benchmarking virtual network functions and their infrastructure," Internet Draft, May 2015.

[7]demo showcase: http://goo.gl/zQ5XsI