A benchmarking methodology for evaluating software switch performance for NFV

Original

Availability:
This version is available at: 11583/2730203 since: 2019-09-11T10:29:28Z

Publisher:
IEEE

Published
DOI:10.1109/NETSOFT.2019.8806695

Terms of use:
openAccess
This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)
A benchmarking methodology for evaluating software switch performance for NFV

Tianzhu Zhang*, Leonardo Linguaglossa*, James Roberts*, Luigi Iannone*, Massimo Gallo†, Paolo Giaccone‡

*Network and Computer Science Department (INFRES), Telecom ParisTech, France
†Nokia Bell Labs, France
‡Department of Electronics and Telecommunications (DET), Politecnico di Torino, Italy

Abstract—Interest in software networking has grown significantly since the introduction of Network Function Virtualization (NFV). Software switches are used in NFV to steer traffic between different virtualized network functions and physical Network Interface Cards (NICs). It is becoming more and more important to objectively evaluate and compare the performance of the multiple alternative implementations that have recently been proposed. A comprehensive performance analysis is still missing for two main reasons: (i) the amount of time required to configure and compare all such tools is enormous; (ii) it is very difficult to define a proper methodology to compare different solutions in a fair manner. In this paper we propose a methodology based on four simple yet representative test scenarios used to evaluate the performance of software switches. We apply this methodology to measure throughput and latency metrics for 6 state-of-the-art software switches namely, OVS-DPDK, snabb, BESS, FastClick, VPP and netmap VALE. Our work constitutes a first step to building a better understanding of design tradeoffs and identifying performance bottlenecks.

I. CONTEXT

Network Function Virtualization (NFV) is expected to have a significant impact on the networking infrastructure, considerably reducing CapEx and OpEx by replacing proprietary, expensive and inflexible hardware middleboxes with virtualized network functions (VNFs) implemented on commodity servers. Software switches are widely adopted by NFV platforms as the dataplane to steer traffic and deliver services. For example, E2 [1] and ParaBox [2] use BESS [3] as their dataplane while ClickOS [4] and HyperNF [5] adopt netmap’s VALE switch [6]. Other implementations such as OVS-DPDK [7], snabb [8], and FD.io VPP [9] also aim to deliver high performance NFV solutions. It is thus of paramount importance for network operators and service providers to understand the performance baselines and to identify the bottlenecks of software switches.

In agreement with the recent observations by Fang et al. [10], we believe that evaluating the performance of software switches is in practice a very difficult task. As noted in [10], most works consider the evaluation of a single design at a time [11, 12, 13], or compare just of a small subset of designs in a narrow test scenario [14, 15, 16].

We have additionally observed that the metrics chosen to compare designs are not always sufficient to fully determine which one is “best”. For instance, we reproduced a very simple scenario with a software switch deployed as a L2 forwarder between two NICs, and measured overall throughput (bidirectional), latency (round-trip time in µs), and the standard deviation of the latter. We tried to understand if these metrics are correlated, or otherwise stated: is the design with the highest throughput also offering the smallest latency? We plotted the measured latency against the standard deviation of the latter. We tried to understand if these metrics are correlated, or otherwise stated: is the design with the highest throughput also offering the smallest latency? We then plotted the measured latency against the standard deviation: this time no pattern could be detected, meaning that there is no correlation between the packet processing rate and performance stability. For instance, while OVS-DPDK can offer a high processing throughput and low latency, it does so with a very high variance in the latency measurements, thus showing that performance can be rather unstable.

We conclude that it is necessary to provide: (i) a fair comparison for all the state-of-the-art software switches, since existing works consider only one or a small subset of them; (ii) simple yet representative test scenarios that can help us to develop a better understanding of the performance baseline.

Fig. 1: Scatter plot of latency/throughput and latency/std. deviation. The throughput results shown in the left plot are obtained with bidirectional tests between two 10Gbps NICs using 64B packets.
and identify the bottleneck. The latter is especially important given the fact that network functions are deployed in virtual environments such as virtual machines (VM) and containers.

**II. METHODOLOGY**

In our work, we consider 6 state-of-the-art software switch implementations, namely OVS-DPDK [7], BESS [3], snabb [8], netmap VALE [6], FastClick [17] and FD.io VPP [9]. All the selected designs are open-source projects with high performance on our testbed. In particular, OVS-DPDK is an OpenFlow (OF) switch performing match-action instructions on a per-packet basis. It can also forward packets to a remote OF controller for further processing. BESS, FastClick, Snabb and FD.io VPP are modular tools. They are capable of composing complex services and interconnecting VNFs with the network. Netmap’s VALE is a pure layer-2 switch. Other open-source software switch implementations are excluded either because of very low performance (eg., Lagopus [18]) or high degree of similarity (eg., ClickNF [19]).

We assume VNFs can run either inside virtual machines or containers and we consider throughput and latency (in terms of round-trip time) as the main performance indicators. Each switch implementation, the device under test (DUT), is evaluated with 4 different test scenarios: physical NIC to physical NIC (p2p), physical NIC to VNF (p2v), VNF to VNF (v2v) and physical NIC - VNF - physical NIC or loopback (lo), as illustrated in Fig. 2. The data flows of the 4 scenarios are explained as follows:

**(p2p) physical-to-physical:** In this scenario, packets arrive via a physical NIC, are fetched by the DUT and then forwarded out via the other NIC. Measurements are performed on both source and sink of the packet flow, outside the DUT. This is the most basic test scenario with no VNFs running in the virtual environment and provides the baseline results for all software switches.

**(p2v) physical-to-virtual:** As for the p2p scenario, packets arrive via a physical NIC and fetched by the software switch under test. They are then forwarded into a simple VNF, where measurements are performed. This scenario demonstrates the typical hop of an NFV data flow and reflects how efficiently software switches can relay packets from NIC to VNFs.

**(v2v) virtual-to-virtual:** In the v2v scenario, packets are generated by a source VNF and injected into the DUT. The DUT then forwards the packets to a sink VNF for measurement. This scenario demonstrates a DUT’s efficiency in terms of traffic forwarding between VNFs.

**(lo) loopback:** In the loopback scenario, packets arrive via a physical NIC and are fetched by the DUT, which forwards them to a VNF. The VNF is a lightweight cross-connect function that forwards the received packets from one virtual interface to the DUT via another virtual interface. The DUT then proceeds to forward the packets to the other NIC. This is a typical NFV scenario with a single VNF on the service chain and provides baseline reference for scenarios of multi-VNF chains.

We believe that these four test scenarios provide a baseline performance characterisation for the software switches. The p2p scenario provides the baseline result showing how efficiently software switches can process packets between physical NICs. The p2v and v2v scenarios, representing two typical dissected hops of NFV data flows, allow us to analyse and identify potential performance bottlenecks. The loopback scenario provides results for a service chain with a single VNF and can be used as a basis to predict the performance of multi-VNF chains.

**III. EVALUATION PLATFORM**

Experiments are conducted on a commodity server running Linux 4.8.0-41-generic distribution. The server is equipped with 2 Intel Xeon E5-2690 v3 @ 2.60GHz CPUs (each with 24 physical cores), using 32k/256k/30720K L1-3 caches, and 2 Intel 82599ES dual-port 10-Gbps NICs. All the VNFs are hosted by QEMU/KVM virtual machines. We use MoonGen [20] as traffic generator, FlowWatcher-DPDK [21] as VNF for p2v and v2v scenarios to measure the throughput. DPDK l2fwd [22] is the VNF used to forward packets in the loopback scenario. For the packet exchange between VMs and software switches, we adopt the netmap passsthrough [23] exclusively for the netmap VALE switch, and the vhost-user protocol [24] for the other considered software switches. Both enable zero-copy packet delivery through shared memory. For all the experiments, we use identical settings for the virtual machines and software switches to ensure a fair comparison. We pin threads of the software switches to isolated CPU cores and disable Turbo Boost to reduce performance variance.
Fig. 3: preliminary throughput test results for all scenarios. Note that these results are obtained with unidirectional tests, thus limited up to 14.88 Mpps for 64B packets.

IV. EXPERIMENTAL RESULTS

In this section, we provide some preliminary results for the throughput test. Fig. 3 illustrates the unidirectional throughput for the considered software switches in the scenarios described in Fig.2. As we observe, no single software switch prevails in the considered scenarios. For example, BESS outperforms the others in the p2v case thanks to its efficient implementation of packet delivery to virtual machines. Snabb prevails in the v2v scenario because its host-user implementation is more efficient for packet delivery between virtual machines.

ACKNOWLEDGMENTS

This work has been carried out at LINCS (http://www.linics.fr) and benefited from support of NewNet@Paris, Cisco’s Chair “NETWORKS FOR THE FUTURE” at Telecom ParisTech (https://newnet.telecom-paristech.fr).

REFERENCES