High-speed per-flow software monitoring with limited resources

Tianzhu Zhang
Telecom ParisTech

Leonardo Linguaglossa
Telecom ParisTech

Massimo Gallo
Nokia Bell Labs

Paolo Giaccone
Politecnico di Torino

Dario Rossi
Telecom ParisTech

1 INTRODUCTION

Software packet processing has always been a relevant solution for both industry and academia. Because of its unparalleled flexibility with respect to proprietary hardware solutions, software packet processing is widely used for the prototyping and debugging of new protocols. While software packet processing has usually been several orders of magnitude slower than its hardware counterpart, the situation started to change with the emergence of fast packet I/O libraries such as netmap [8] and Data Plane Development Kit (DPDK) [6]. Of particular significance, we now assist to the advent of flow-level high-speed applications that either provide per-flow fairness [3], or propose a high-performance user-space flow-level network function framework [5].

The emergence of such applications requires not only to generate but also to monitor traffic flows in real-time, which is particularly relevant for stress-tests during the development phase. However, few tools are capable of high-speed flow-level monitoring in the worst-case scenarios (i.e., 64B packets at line rate) without sampling and by using a limited amount of resources; the latter property is of paramount importance when the monitor is co-located either with the traffic generator (to allow more complex traffic patterns) or with the device under test (DUT). As an example, DPDK-Stat [9] focuses on advanced traffic analysis (e.g., including full-payload TCP flow reconstruction and deep packet inspection) at 40 Gbps line rate using commodity hardware, it does however consumes all the available resources, and geared toward post-processing analysis. Similarly, other tools exists in the literature for monitoring incoming traffic but rely on heavy sampling [2], occupy too many resources [4, 7] or do not focus on worst-case scenario [1, 9].

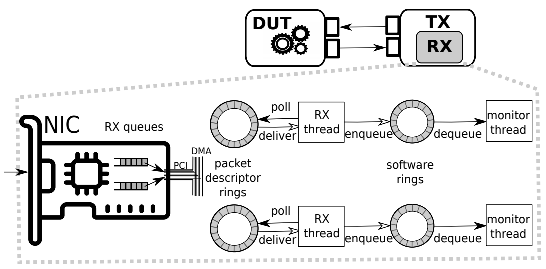

Figure 1: The configuration of FlowMon-DPDK

In this demo, we showcase *FlowMon-DPDK* [11], our software traffic monitor capable of both packet- and flow-level statistics by using a limited amount of resources, and that we make available as open-source project [10].

2 FLOWMON-DPDK DESIGN

The typical usage scenario for stress-testing a network device is shown in Fig. 1: a traffic generator (TX) transmits packets at line rate to the device under test (DUT), which forwards packets to a traffic monitor (RX). In order to minimize the amount of resources, in [11] we carefully analyzed the design space and adopted the solutions yielding to the best performance.

In essence, for a 10 Gbps link, FlowMon-DPDK uses 2 hardware queues to split the traffic load over 2 cores. Note that we expect to easily extend the maximum sustainable capacity beyond 10 Gbps by using for example 20 cores (i.e., a second CPU on a different socket) for capturing 100 Gbps, provided that the PCI Express bus does not become the bottleneck. RX thread polls packets from the NIC ring buffer, transfers them via a software ring to the monitor thread. This thread continuously polls packets from the dedicated software ring and implements FlowMon-DPDK main processing functionalities. FlowMon-DPDK is capable of both per-packet or per-flow monitoring. While packet counting can be done directly by the hardware, per-flow counting is only minimally facilitated: particularly, to avoid the overhead of computing a hash over the packet header, we re-use the 32-bit hash computed by the NIC for flow identification.

FlowMon-DPDK uses a double-hash table with a variable number of entries (2^{16} by default). Each entry contains two static flow buckets, and each bucket contains a packet counter (plus additional statistics if properly configured) which is updated when packets belonging to the corresponding flow arrive. If there are more than two flows indexed to the same entry, a linked list is used to store additional flows. FlowMon-DPDK supports also advanced per-flow operations, such as computing the flow interleaving degree (i.e., the number of packets of *other* flows in between two packets of the current flow) as well as first, second and higher-order moments of the variables of interest (e.g., flow rates, interleaving degree distribution, etc.).

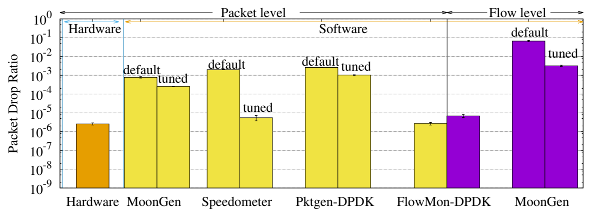

Figure 2: Hardware vs software solutions for packet- vs flow-level monitoring: Average PDR of different monitoring tools with 95% confidence interval.

3 FLOWMON-DPDK DEMONSTRATION

3.1 Setup

The testbed comprises one server, equipped with Xeon E5-2660 v3 2.60 GHz CPUs (with L1-L3 caches 32/256/25600 kB) and 2 Intel® 82599ES 10 Gbps NICs. Since we want to stress-test the monitoring applications with the maximum line rate, we directly connect the TX and RX without any DUT (i.e., the DUT is a lossless fiber cable), thus the two NICs are directly connected through an optical fiber. As TX, we deploy the MoonGen traffic generator while FlowMon-DPDK as well as the other monitoring tools are installed as RX. We configure MoonGen to generate 5 billion minimum size packets (64 bytes) at line rate (14.88 Mpps) from 65536 flows. Whereas traffic generation employs 4 cores, FlowMon-DPDK uses the 2 physical (Fig. 1) cores, and the RX server is configured with optimal tunings (e.g., set the CPU frequency scaling governor to "performance", disable Turbo-boost, pin processes to dedicated cores, etc.).

3.2 Demo scenarios

In particular, we consider applications of increasing complexity, notably: (i) hardware-based packet-level counting: native DPDK application, (ii) software-based packet-level counting: MoonGen and pktgen-DPDK (both programmed through lua scripts) and Speedometer [7] (DPDK application), with default configuration from GitHub or our own tuned version, (iii) software-based flow-level packet counting: FlowMon-DPDK and MoonGen (flow-level version).

All the tools are tested under the same scenario as FlowMon-DPDK (256 packet batches, 4096 rx/tx descriptors, etc.), and the primary performance metric we consider is the Packet Drop Ratio (PDR). While all applications have different outputs, their ability to keep-up with the traffic is the first indication of the scenario they can be used with, as a large PDR testifies inaccuracy in the reported results.

Fig. 2 shows the expected performance according to [11]. In particular, from left to right: (i) already accessing hardware registers yields to 10^{-6} drops, so that (ii) packet-loss of software tools is already higher for packet-level operations, (iii) where an (opportunely tune) Speedometer and FlowMon-DPDK have similar performance. Finally, it can be shown that (iv) FlowMon-DPDK packet loss rate only minimally increase with flow-level operations.

3.3 Demo workflow

The demonstration will allow users to interact with both the TX and RX, notably: (i) altering the sending process (e.g., number of flows, flow skew, etc.), (ii) changing the enabled FlowMon-DPDK counters (e.g., packet-level vs flow-level; flow-rate vs flow-burstiness; instantaneous values vs cumulated vs mean vs high orders vs percentiles), (iii) interacting with the monitoring CLI (quiet mode vs periodic writing vs ncurses based terminal with sorted flows statistics). A video showing the demonstration is available at https://youtu.be/B8uaw9UgMm0.

ACKNOWLEDGMENT

This work has been carried out at LINCS (http://www.lincs.fr) and benefited from support of NewNet@Paris, Cisco's Chair "Networks for the Future".

REFERENCES

[1] 2018. ntop. https://www.ntop.org/. (2018).
[2] 2018. sFlow. https://sflow.org/. (2018).
[3] Vamsi Addanki, Leonardo Linguaglossa, James Roberts, and Dario Rossi. 2018. Controlling software router resource sharing by fair packet dropping. In *IFIP Networking*.
[4] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*. ACM, 275–287.
[5] Massimo Gallo and Rafael Laufer. 2018. ClickNF: a Modular Stack for Custom Network Functions. In *2018 USENIX Annual Technical Conference (ATC)*.
[6] Intel. 2010. Data Plane Development Kit. http://dpdk.org/. (2010).
[7] Rafael Leira. 2014. iDPDK-Speedometer. https://github.com/hpcn-uam/. (2014).
[8] Luigi Rizzo, Marta Carbone, and Gaetano Catalli. 2012. Transparent acceleration of software packet forwarding using netmap. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2471–2479.
[9] Martino Trevisan, F Alessandro, Marco Mellia, Maurizio Munafò, and Dario Rossi. 2016. DPDK-Stat: 40Gbps Statistical Traffic Analysis with Off-the-Shelf Hardware. *Technical report* (2016).
[10] Tianzhu Zhang. 2017. FlowMon-DPDK. https://github.com/ztz1989/FlowMon-DPDK. (2017).
[11] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. 2018. FlowMon-DPDK: Parsimonious per-flow software monitoring at line rate. In *TMA Conference 2018*.

TECHNICAL REQUIREMENTS

Equipment to be used: The main components of the demo, including the traffic generator and FlowMon-DPDK, run on remote servers. So we only require one PC/laptop (with SSH terminal applications) with a large monitor (e.g., 24 or 27 inches), so as to demonstrate the experimental results. If needed, we can use our own laptop. We also need a big desk to host the laptop and the monitor. To give the audience a clear picture of the demo, we will prepare a poster (size A1), in which the setup is described in detail, along with the ideas and results of our work. So we also need a support for the poster.

Space needed: For the demo, we need enough space to host the desk (with the laptop and the monitor) and the poster support.

Setup time required: Few minutes are needed to setup the demo, including the time to log into the remote server, plus the time to initialize both the traffic generator (MoonGen) and FlowMon-DPDK.

Additional facilities needed: Since the testbed runs remotely, we need reliable Internet access to enable SSH sessions.