Batched packet processing for high-speed software data plane functions

David Barach[1], Leonardo Linguaglossa[2], Damjan Marion[1],
Pierre Pfister[1], Salvatore Pontarelli[2,3], Dario Rossi[2], Jerome Tollet[1]
[1]Cisco Systems, [2]Telecom ParisTech, [3]CNIT - Consorzio Nazionale Interuniversitario per le Telecomunicazioni
{first.last}@cisco.com, {first.last}@telecom-paristech.fr

***Abstract*—In the last decade, a number of frameworks started to appear that implement, directly in user-space with kernel-bypass mode, high-speed software data plane functionalities on commodity hardware. Vector Packet Processor (VPP) is one of such frameworks, representing an interesting point in the design space in that it offers: (i) in user-space networking, (ii) the flexibility of a modular router (Click and variants) with (iii) the benefits brought by techniques such as batch processing that have become commonplace in lower-level building blocks of high-speed networking stacks (such as netmap or DPDK). Similarly to Click, VPP lets users arrange functions as a processing graph, providing a full-blown stack of network functions. However, unlike Click where the whole tree is traversed for each packet, in VPP each traversed node *processes all packets in the batch* before moving to the next node. This design choice enables several code optimizations that greatly improve the achievable processing throughput: the purpose of this demonstration is to introduce the main VPP concepts and architecture, as well as experimentally showing the impact of design choices –and especially of batch packet processing–, on the achievable packet forwarding performance.**

I. Context

To help escaping network ossification, software implementation of network function on Common Off The Shelves (COTS) hardware has started to gradually replace dedicated network hardware equipment. A recent tendence has further emerged to implement high-speed network stacks bypassing the OS kernel and bringing the hardware abstraction directly to the user-space. A number of efforts in this space aim at providing useful low-level building blocks for high-speed packet capture (such as netmap [10] and DPDK [2]) whereas others target the design of full-blown modular frameworks for high-speed packet processing (such as RouteBricks [9] and FastClick [4]).

Initially proposed in [5], Vector Packet Processor (VPP) is one such full-blown framework, that was recently released as open source software in the context of the Fast Data IO (FD.io) Linux Foundation project [3]. VPP exploits low-level building blocks (such as DPDK) to provide a complete set of layer-2 and 3 functionalities: in contrast with frameworks whose first aim is performance on a limited set of functionalities, VPP is feature-rich. As illustrated in Fig.1, the VPP processing graphs comprises hundreds of network functions: in particular, the picture highlights (in green and red) two of the functions (respectively, Ethernet switching and IP forwarding) that will be used to illustrate VPP architecture and to show performance benefits of its design in the demonstration.

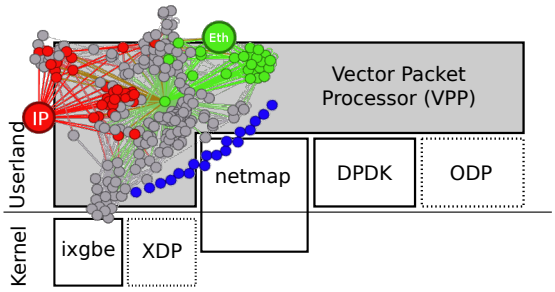

Fig. 1. VPP scope and processing tree (the nodes used in the demonstration are highlighted in red and green and blue).

II. VPP Architecture

Due to space limits, we provide only a very brief overview of VPP here, and refer the interested reader to a technical report for further details [8]. In a nutshell, VPP offers the flexibility of a modular router, retaining a programming model similar to that of Click [7] and variants [9, 4]. Additionally, VPP does so in a very effective way, by extending benefits brought by techniques such as *batch processing* to the whole packet processing path. Whereas DPDK uses *low-level batching* to efficiently fetch packets from the Network Interface Card (NIC) avoiding per-packet interrupt pressure (that kills the peformance of traditional kernel-based network stacks), VPP employs *batched processing* to increase as much as possible the number of instructions per clock cycle (IPC) executed by the microprocessor (making packet processing as efficient as possible). In a sense, VPP raises and extends the batching techniques used from low-level building blocks to high-level packet processing functions.

The additional original idea in VPP is to rearrange the order in which functions are executed: instead of letting each packet of the batch to traverse the full forwarding-graph before processing the next packet (as in the "run-to-completion" model of Click/FastClick [7, 4]), VPP let a single node of the forwarding-graph *process all packets in the batch* before moving to the next node. Otherwise stated, VPP offers a systematic way to efficiently process packets in a "vectorized" fashion, that as we will show in this demonstration, provides sizeable performance benefits. In particular, this peculiar VPP design is explicitly tailored to (i) minimize the data cache misses using data prefetching, (ii) minimize the instruction cache misses, (iii) increase the instructions per clock-cycle

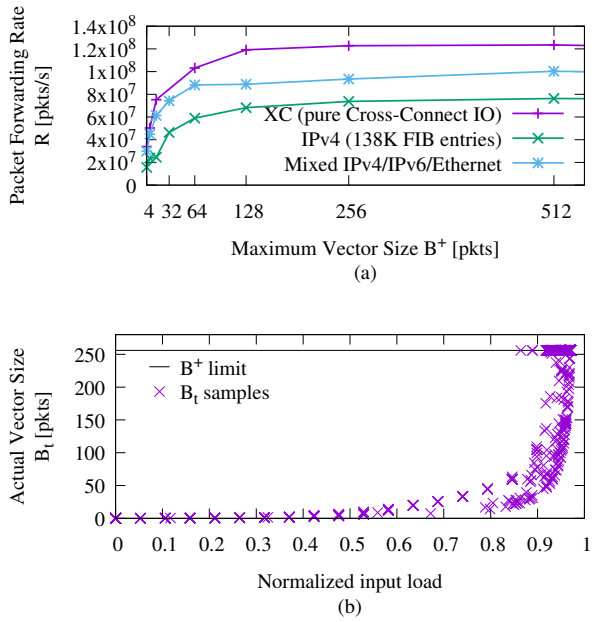

Fig. 2. Demonstrating VPP ability to handle 10Gbps traffic rate on a single-core: (a) packet processing rate as a function of the maximum vector size B^+ in a stress-test scenario and (b) actual vector size as a function of the traffic rate for $B^+ = 256$ in realistic scenarios

of the CPU front-end. These software architectural decisions all concur in efficiently exploiting the parallelism offered by modern pipelined CPU of standard COTS hardware.

It is worth pointing out that tools such as G-opt [6] and FastClick [4] do offer some form of "compute batching", which however barely resembles to batching in VPP only from a very high-level viewpoint. In paricular G-opt batching just avoids CPU stalls due to memory latency, whereas FastClick implicitly process packets individually, and only specific nodes have been augmented to *also* accept batched input (which pays the price of linked-list implementation and thus provide limited benefits as in "*the forwarding test case [...] the batching couldn't improve the performance*" [4]).

For reason of space, we cannot report on a number of other important architectural details of VPP, such as lock-free multi-threading, systematic function flattening, dual and quad-loop processing, support for multi-architecture and Direct Cache Access (DCA), that are covered in [8].

III. Demonstration setup and workflow

We plan to demonstrate benefits of systematic vectorized processing in both stress test and realistic scenarios. Our hardware setup consists in a COTS server with 2× Intel Xeon Processor E52690, each with 12 physical cores running at 2.60 GHz in hyper-threading and 576KB (30MB) L1 (L3) cache. The server runs a vanilla Linux kernel 4.8.0-41 and is equipped with 2× Intel X520 dual-port 10Gbps NICs, that are directly connected with SFP+ interfaces.

Due to space limits, this extented abstract illustrates two main features, although in the demonstration session we plan to additionally show benefits of multi-loop programming practice due to the instruction parallelization (improvement in terms of IPC) and pre-fetching (avoiding stalls due to data cache misses). We plan to release a video of the demo at [1].

A. Stress test: Throughput gain due to vectorized processing

VPP processing model continuously polls vectors of size up to B^+ packets from the NIC for processing. Changing the size of the vector affects the number of packets that can be processed in parallel, and in particular increasing the vector size allow to amortize the framework overhead. Fig.2-(a) depicts for simple cross-connect IO (XC), or for more complex cases including forwarding (IPv4) and mixed Ethernet switching and IPv4+IPv6 forwarding (MIX) the VPP packet processing throughput achieved on a single-CPU core for worst-case input traffic of 64B packets sent at 10Gbps. It can be seen that processing throughput rapidly increase with B^+, and saturates for $B^+ \geq 256$, which confirms the appel of vectorized processing.

B. Realistic scenario: Delay due to vectorized processing

Clearly, vectorized processing tradeoffs with per-packet delay experienced by packets that arrive at the NIC while a batch is being processed. VPP controls the *maximum* number of packets to be processed by varying the maximum batch size B^+, controlling the delay due to batching. In particular, batching is especially useful for high traffic rates, where batched processing allow to increase the instruction per clock-cycle efficiency. Conversely, at low traffic rates the NIC queues seldom fills, so that the vectors polled from the NIC at time t typically have a size $B_t < B^+$. Fig. 2-(b) reports actual batch sizes polled from the NIC for increasing input traffic rates, showing that the vector size is generally small, and that only as the input load approaches the processing capacity, the vector size grows to $B_t \approx B^+$. Otherwise stated, vectors grow only when this is actually needed, as per Fig.2-(a), to increase the efficiency of the processing to sustain the incoming traffic. This additionally confirms that the expected delay will be negligible in practical non-overloaded scenarios.

Acknowledgments

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" at Telecom ParisTech (https://newnet.telecom-paristech.fr).

References

[1] https://newnet.telecom-paristech.fr/index.php/vpp-bench.
[2] Data plane development kit. http://dpdk.org.
[3] Fast Data Project (FD.io). https://fd.io.
[4] T. Barbette, C. Soldani, and L. Mathy. Fast userspace packet processing. In *ACM/IEEE ANCS*, 2015.
[5] D. Barach and E. Dresselhaus. Vectorized software packet forwarding, June 2011. US Patent 7,961,636.
[6] A. Kalia, D. Zhou, M. Kaminsky, and D. G. Andersen. Raising the bar for using gpus in software packet processing. In *USENIX NSDI*, 2015.
[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The Click Modular Router. *Operating Systems Review*, 34(5):217–231, 1999.
[8] L. Linguaglossa et al. High-speed Software Data Plane via Vectorized Packet Processing (Extended Version). In *Tech.Rep.*, 2017.
[9] M. Dobrescu et al. Routebricks: exploiting parallelism to scale software routers. In *SIGOPS*, 2009.
[10] L. Rizzo. netmap: a novel framework for fast packet I/O. In *USENIX ATC*, 2012.