# Per-Packet Load Balancing for Multi-Core Middleboxes

Hugo Sadok*, Miguel Elias M. Campista, Luís Henrique M. K. Costa
*GTA/PEE/COPPE – Universidade Federal do Rio de Janeiro*
{sadok, miguel, luish}@gta.ufrj.br

## 1 Introduction

Middleboxes are an essential part of today's networks [8]. Although middleboxes have been traditionally deployed using purpose-built hardware, there is an increasing trend towards the move to software running on commodity servers [7]. This offers several advantages such as reduced costs and simpler deployment. However, these benefits also come with significant performance overhead [9].

As a way of improving performance, software middleboxes often leverage multiple CPU cores [7, 9]. These middleboxes generally choose between two different models: *run-to-completion* and *pipeline* [9]. In the run-to-completion model, we process every individual packet using a single core, whereas in the pipeline model we process the same packet sequentially using multiple cores, one for each step of the pipeline. Comparing both models, run-to-completion performs better, presenting higher throughput and lower latency [9]. To use multiple cores in the run-to-completion model, packets must be distributed across different cores before they are processed. This is often achieved using Receive-Side Scaling (RSS) [7].

RSS is a feature of multi-queue NICs [3, 4] that distributes packets to different cores based on a hash of the "five-tuple". Ideally, with RSS, packets should be processed in all cores, with those from the same flow going to the same core. However, RSS's decisions are often unfair and inefficient. Since RSS randomly hashes flows to cores, hash collision can cause significant imbalance. The arguments against RSS are similar to those against per-flow ECMP on datacenters. In fact, per-packet ECMP has been shown to outperform per-flow ECMP, even though it causes packet reordering [6].

In this work, we show that middleboxes can benefit from per-packet load balancing and provide a design and implementation that can run in existing hardware.

*Hugo Sadok is a student author.

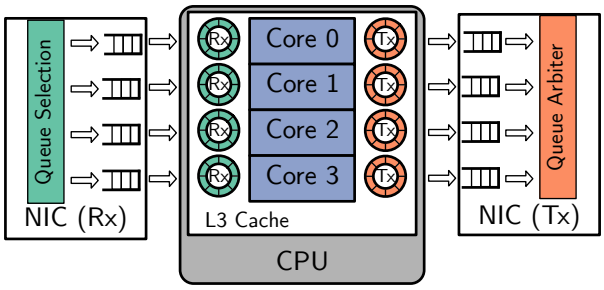


*Figure 1:* Architecture for packet processing.

## 2 Design and Implementation Overview

We now present the design of a multi-core middlebox with per-packet load balancing. Figure 1 depicts the architecture of packet processing in a 4-core CPU. Each packet that enters the NIC (Rx) is assigned to a queue. Each queue has a correspondent ring buffer that lives mostly in the CPU L3 cache. In the run-to-completion model, each ring buffer is designated to a different core. This way, when the NIC chooses a queue, it is also choosing a core. After processing the packet, a core sends it to a transmit ring buffer. Just like in the receive part, each ring buffer is associated with a different queue in the NIC (Tx). The NIC's queue arbiter then transmits packets from queues in a round-robin fashion.

To achieve per-packet load balancing, the NIC (Rx) should be able to send packets to different queues uniformly, regardless of which flow they belong to. Unfortunately, this is not directly supported by current NICs [3, 4]. However, there is a feature called Flow Director that was originally conceived to associate *specific* flows to queues. Flow Director is usually not considered for middleboxes as it limits the total number of flows that are allowed in the system. We use Flow Director in an unconventional manner, instead of matching flows, we configure it such that packets are assigned to

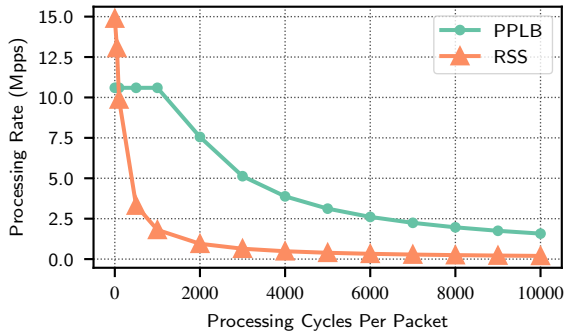


*Figure 2:* Processing rate for 64B packets with increasing number of processing cycles using PPLB and RSS.

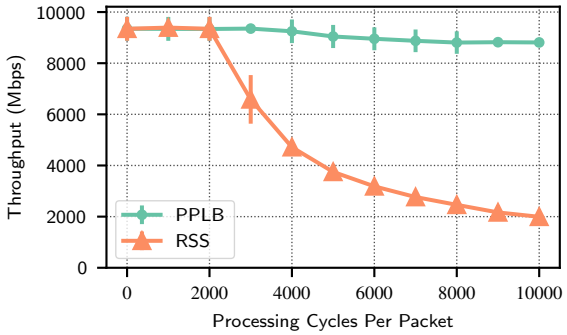


*Figure 3:* Throughput for a single TCP flow with increasing number of processing cycles using PPLB and RSS.

queues based on the checksum field of the TCP header[1]. Using this approach, TCP packets are homogeneously distributed across queues regardless of their flows, while non-TCP packets continue to be distributed using RSS.

Note that distributing packets uniformly across cores may not be enough. Packet reordering may degrade performance in some TCP implementations and legacy middleboxes may rely on receiving all packets of a flow (*e.g.*, SYN packets to identify new TCP connections). We are currently evaluating a strategy that reorders packets before they are transmitted. Unlike traditional reordering, we benefit from the knowledge of which packets arrived in the middlebox, avoiding the wait for packets that got lost. We have implemented an initial prototype using DPDK [1].

## 3 Preliminary Results

We show preliminary results using only per-packet load balancing without reordering. We run experiments on a testbed with two servers, one is used as the packet generator while the other functions as a middlebox. The middlebox server is equipped with two Intel Xeon E5-2650 CPUs, each of which has 8 cores, and 256GB of RAM (divided equally among all memory channels). The packet-generator server is equipped with a single Intel Core i7-7700 CPU and 32GB of RAM. Each server is also equipped with an Intel 82599ES 10Gb NIC [3] and runs Linux kernel 4.9.0-5. To simulate middleboxes with different complexities, we developed a simple middlebox that busy loops for a given number of cycles for every packet it receives. In all experiments we use 8 CPU cores for the middlebox.

Our first experiment measures the maximum improvement in packet processing rate that can be achieved using per-packet load balancing (PPLB). We use MoonGen [5] to generate 64B TCP packets with fixed five-tuple but variable payload – and therefore variable checksum. Since the five-tuple remains constant, RSS identifies all packets as being part of the same TCP flow and redirects them to the same CPU. This contrasts with per-packet load balancing, that distributes packets uniformly. Figure 2 shows the processing rate as a function of processing cycles per packet. Note that PPLB is limited to 10.6Mpps and, as consequence, achieves a processing rate lower than that of RSS for middleboxes with few processing cycles. This, however, is not fundamental and is in fact a hardware limitation of the 82599 when using Flow Director. For less trivial middleboxes PPLB is able to process significantly more packets than RSS.

Since packets may arrive out of order, a better processing rate may not translate into better performance for TCP. To measure the impact on a real TCP flow we conduct a second experiment using iperf3 [2] with a single flow. Figure 3 shows the throughput as a function of processing cycles per packet. Note that, even for 10,000 processing cycles per packet, PPLB is able to sustain a throughput of 8.8Gbps while RSS only achieves 2Gbps.

[1] We actually use only a certain number of LSBs of this field, depending on the number of cores. This let us circumvent the limited number of flows, defining rules that exhaust all possible matches.

## References


[1] Data Plane Development Kit. http://dpdk.org.

[2] iperf3. http://software.es.net/iperf/.

[3] Intel 82599 10 GbE Controller Datasheet, 2016.

[4] Intel Ethernet Controller X710/XXV710/XL710 Datasheet, 2018.

[5] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. Moongen: A scriptable high-speed packet generator. In *ACM IMC*, 2015.

[6] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM*, 2017.

[7] M. A. Jamshed, Y. Moon, D. Kim, D. Han, and K. Park. mOS: A reusable networking stack for flow monitoring middleboxes. In *USENIX NSDI*, 2017.

[8] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *ACM SIGCOMM*, 2012.

[9] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu. NFP: Enabling network function parallelism in NFV. In *ACM SIGCOMM*, 2017.