Assessing the performance of XDP and AF-XDP based NFs in edge data center scenarios

Poster: Assessing the Performance of XDP and AF_XDP Based NFs in Edge Data Center Scenarios

Federico Parola
Politecnico di Torino
federico.parola@polito.it

Roberto Procopio
TIM S.p.A.
roberto.procopio@telecomitalia.it

Fulvio Risso
Politecnico di Torino
fulvio.risso@polito.it

ABSTRACT

While servers in traditional data centers can be specialized to run either CPU-intensive or network-intensive workloads, edge data centers need to consolidate both on the same machine(s) due to the reduced number of servers. This paper presents some preliminary experiments to determine how to improve the overall throughput of the above servers, being XDP and AF_XDP the two main technologies into play.

CCS CONCEPTS

• **Networks** → *Middle boxes / network appliances*; *Network servers*; **Programmable networks**; • **Software and its engineering** → *Communications management*.

KEYWORDS

Network Function Virtualization, XDP, AF_XDP

1 INTRODUCTION

Telco operators are increasingly building micro data centers at the edge of their network, in which the few available servers are expected to execute both telco-oriented workloads (e.g., 5G user-plane functions) and general-purpose applications (e.g., content caches, object recognition software, 5G control plane, etc.). While in traditional data centers each server is dedicated to either one of the workloads, the reduced number of servers (usually on the order of 5-10 units) suggest that they would host a variable mix of network-oriented and general-purpose workloads.

Technologies such as eXpress Data Path (XDP)[2] and AF_XDP, could provide an ideal packet processing infrastructure for the above "integrated" scenario, while other kernel-bypass technologies (e.g. DPDK) are more suitable for servers dedicated to dataplane-oriented tasks due to the necessity of dedicated CPU cores and huge memory pages, and the difficulties to support applications that require the standard TCP/IP stack. On the other side, XDP allows to process packets in the NIC driver, retaining the possibility to yield a packet to the Linux network stack, while AF_XDP sockets can be used to bypass limitations of eBPF programs.

This paper presents our preliminary analysis of the interactions between the above-mentioned packet processing mechanisms, the underlying hardware primitives (e.g., DDIO, caches), and the running workloads, deriving insights on how to optimize the throughput in the above target scenario.

2 EXPERIMENTS

This section presents our preliminary findings when running mixed workloads on the same server.

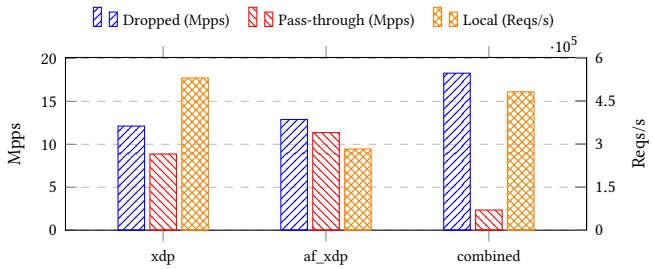

Figure 1: Throughput comparison for different technologies and different traffic destinations.

The choice between XDP and AF_XDP depends on the destination of the involved traffic. We identified three different classes of traffic that can be handled by our server, each one characterized by a different type of IO processing. **Dropped traffic** is the traffic that is discarded by the NF, typically a Firewall, a DDoS Mitigator or a traffic shaper. **Pass-through traffic** is the traffic that is forwarded by the machine after being processed by one or more network functions. This can be the case of a load balancer redirecting packets towards backends running on a different server. **Local traffic** is the traffic that has to be processed by an application running on same server as the NF, e.g., traffic that is processed by a firewall and terminated on a local pod. Even though this three scenarios are usually combined in a common deployment, we decided to study them in isolation in order to profile the behavior of the technologies under test and then define the best way to combine them in a real use case. Our experiments leverage the `xdp_rxq_info` and the `xdpsock` samples from the Linux kernel tree[1] to perform some simple packet IO in XDP and AF_XDP. For AF_XDP we ran the user space packet processing thread on the same core in charge of handling interrupts of the NIC and enabled the `SO_PREFER_BUSY_POLLING` flag, that resulted in better performance. We also evaluated a combined mode, in which AF_XDP sockets are enabled but the verdict on the final destination of the packet is taken at the XDP level. For local traffic we executed an instance of the *memcached* server and measured the maximum number of requests per second that it was able to handle.

Results in fig. 1 show that the technology achieving the best throughput depends on the destination of the traffic: while the combined mode achieves the best results for dropped traffic, pure AF_XDP yields better performance when the traffic is redirected outside the machine. The reasons for the above results can be found in the following considerations: 1) all packets processed at the AF_XDP level pass through an XDP program, hence dropping the packet at this early stage can therefore save CPU cycles; 2) packets follow a different path in the kernel if redirected at the AF_XDP or

[1] https://github.com/torvalds/linux/tree/v5.14/samples/bpf

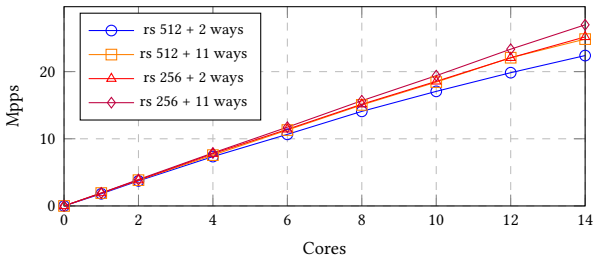

Figure 2: Multi-core scalabilty of router NF with different values of DDIO LLC ways and different RX ring sizes (rs).

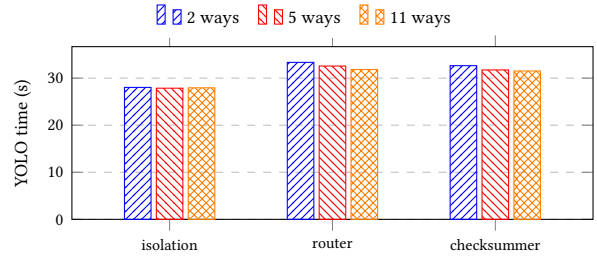

Figure 3: YOLO execution time for different DDIO ways values, when executed in isolation and in parallel with the router and the checksummer running 12 cores.

XDP level, and the AF_XDP path seems to be more optimized; 3) when working in combined mode, the buffer management model of AF_XDP is also applied to XDP; this brings an improvement when dropping packets but drastically reduces performance when redirecting traffic.

Multi-core scalability of XDP and AF_XDP is heavily affected by DDIO cache occupancy. We defined two test network functions, a static router and a checksummer, recomputing the L4 checksum of the packet multiple times, and tried to scale them on multiple cores. The blue-circle line in fig. 2 shows the throughput of the XDP router (similar results apply to the checksummer) and highlights how the NF scalability is far from linear when exceeding 4/6 cores, even if there is no apparent contention among cores. Monitoring the number of cache accesses shows an increase in the percentage of LLC misses as the number of cores grows. We suspect this behaviour is due to the problem of *leaky DMA* presented in [3] and extensively studied in [1]. We repeated the tests above applying the solutions suggested in the two papers: increasing the number of LLC ways at disposal of DDIO (from 2 to 11) and reducing the size of the RX rings used by the NIC (from 512 to 256 packets). Figure 2 shows that both solutions improve the throughput, and allow to achieve linear scalability if applied together. One interesting point we want to further investigate is why the impact of the leaky DMA is already visible for small packets in XDP/AF_XDP, while in the two papers mentioned above and experimenting with DPDK, this problem only arises for big packets.

Increasing DDIO cache occupancy does not impact on the performance of CPU-bounded workloads. Reducing the NIC's RX ring size produces known drawbacks, like the inability to handle bursts of traffic, or to buffer packets while the CPU core is busy in other tasks. In fact, given the default values suggested by Intel (2-ways DDIO LLC, equivalent to the 18% of the cache, and 512 packet buffers), we expect to pay our changes with a negative impact in other applications. Therefore, we performed a set of experiments to possibly validate our guess. In the first experiment, shown in fig. 3, we selected the YOLO image recognition application as an example of CPU-bounded program and run it in parallel with an XDP-based network function to see whether changing the DDIO cache occupancy affected the time needed by YOLO to process a sample image. In the second experiment, we measured the maximum number of requests served by an `nginx` web server when undergoing a 10 Mpps DDoS attack, verifying if increasing the number of DDIO LLC ways amplified the effect of the attack. In both cases our results show no negative effects of increasing the DDIO cache occupancy on CPU-bounded applications, while the benefits on network functions are clearly shown in the former section.

We plan to further investigate this phenomenon extending the set of experiments in order to cover more scenarios possibly affected by DDIO LLC share.

3 CONCLUSIONS

This paper presents the preliminary experiments carried out to assess and compare the performance of XDP and AF_XDP, the two main technologies provided by the Linux kernel for fast packet processing. Our results show that the choice of the best technology depends on the destination of the traffic (e.g., local vs remote), and that the interaction with the underlying hardware mechanisms of the CPU (DMA and DDIO) has to be tuned to achieve optimal performance.

This preliminary results lay the foundations of our future work that aims at providing a complete set of best practices to design VNFs leveraging the kernel provided infrastructure.

ACKNOWLEDGMENTS

Federico Parola acknowledges support from TIM S.p.A. through the PhD scholarship.

REFERENCES

[1] Farshin, A., et al. Reexamining direct cache access to optimize i/o intensive applications for multi-hundred-gigabit networks. In *2020 USENIX Annual Technical Conference* (2020), pp. 673–689.

[2] Høiland-Jørgensen, T., et al. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies* (2018), pp. 54–66.

[3] Tootoonchian, A., et al. Resq: Enabling slos in network function virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 283–297.