PMDlink – Bridging the Gap Between DPDK and Hardware Packet Processing

Rubens Figueiredo, Hagen Woesner, Nic Hart
BISDN GmbH
Berlin, Germany
rubens.figueiredo@bisdn.de
hagen.woesner@bisdn.de
nic.hart@bisdn.de

Andreas Kassler
Deggendorf Institute of Technology
Deggendorf, Germany
andreas.kassler@th-deg.de
Karlstads Universitet
Karlstad, Sweden
andreas.kassler@kau.se

Holger Karl
Hasso Plattner Institute
Potsdam, Germany
holger.karl@hpi.de

***Abstract*—Software networking functions flexibly implement packet processing, but performance is lacking compared to purpose built hardware. To address this, commodity NICs execute packet processing in hardware, improving performance, however their fixed-function design restricts the features which can be offloaded to hardware. One particular challenge is to implement HQoS, which manages traffic by enforcing QoS requirements at multiple levels, maintaining thousands of queues and applying rate limits. In this work, we combine DPDK based user-space packet processing with hardware offloaded packet processing on commodity NICs, by integrating DPDK with Linux switchdev, thus combining software flexibility and hardware acceleration. We propose PMDlink, a side-channel API enabling direct hardware control via DPDK's PMD, exposing advanced packet processing features without major driver modifications. Our preliminary results highlight the benefits of hardware offloading by showing the combined hardware-offloaded HQoS functions and DPDK. Future work will refine PMDlink as a modular interface, to better expose NIC capabilities to DPDK, and to program more complex scheduling profiles.**

I. Introduction

Software-Defined Networking (SDN) proposes to implement network functions such as firewalls or routers, on commodity, multi-core, servers and Network Interface Cards (NICs), aiming to be more flexible and to deploy new functions, faster. SDN virtualizes these functions, turning them into software entities known as *Virtual Network Functions (VNFs)*. Typically, VNFs bypass the Linux kernel, rather being executed directly in user space, using frameworks such as DPDK, for maximum throughput at low latency.

Commodity NICs can offload specific packet processing functions from the Central Processing Unit (CPU), reducing server load [1]. Possible offloads depend on hardware features, such as: Receive-Side Scaling (RSS), segmentation offloading, and transmit packet steering (XPS). Such offloading is, however, rather limited in scope. In contrast to smartNICs, which incorporate programmable hardware engines for operating on packets directly on the data path, commodity NICs include fixed-function hardware packet processing capabilities, with typically limited configuration options. This limitation means that fine-grained control over hardware functions, such as implementing hardware flow monitoring [2] or tuning per-queue shaping parameters, has traditionally required complex integration schemes between hardware and software. However, their low cost and close integration with standard hardware make these attractive candidates for integration into high-performance packet processing operations.

Coresident latency-critical and high data-rate services require high-performance traffic management, including queuing, shaping and scheduling [3] to prevent impairment that may lead to unacceptable latency or packet loss. At speeds of hundreds of Gbps, traffic management is increasingly challenging, as traffic needs to be classified, assigned to queues, rate shaped according to provider and user contracts, and scheduled for transmission, often within a few nanoseconds, to meet the Quality of Service (QoS) requirements of multiple service classes and different users. Managing these multiple constraints requires a scheduling algorithm capable of grouping different queues according to their QoS requirements. Hierarchical Quality-of-Service (HQoS) addresses this by structuring packet schedulers into multiple layers, creating hierarchical relationships between scheduling nodes. These nodes correspond to physical network layers, such as subscribers or network aggregation, enabling fine-grained traffic prioritization and shaping [4].

In previous work [5], we have shown the performance and scalability challenges for software HQoS. In particular, latency increases with background load at the scheduler, leading to high latency and packet loss. The demand for high-performance HQoS enforcement has led to some important work on the execution of HQoS in commodity hardware. For example, TONIC [6] approximates hierarchical scheduling by serving multiple transmission queues as Weighted Round Robin (WRR), and enforcing strict priority through packet enqueueing. This approach modifies only the enqueue mechanisms, but does not address traffic shaping. Kuperman et al. [7] offload Linux's HTB scheduler directly on Mellanox NICs via Linux traffic control `tc`, moving entire HQoS enforcement to hardware. However, the primary limitation of this approach is the limited number of hardware queues, which limits the scalability of the solution. Moreover, an evaluation of commodity NIC hardware HQoS and DPDK packet processing has not

yet been done.

This work proposes a solution to the problem of interconnecting software and hardware packet processing. We focus on two main aspects. First, we integrate DPDK with Linux `switchdev`, which combines software packet processing with hardware-accelerated packet forwarding and QoS. By leveraging commodity NIC to execute HQoS, we integrate standard components into DPDK, combining the flexibility of software packet processing with the performance of hardware. Moreover, we benefit from well established software and hardware components widely accepted by the community. We also present a small benchmarking experiment that measures latency of a traffic stream while background traffic is being shaped, showing that NIC QoS can be used with DPDK to protect delay critical traffic at high data rates and low latency. Second, we address driver limitations to configure the available hardware packet processing features, restricting full configuration of the HQoS and other functionality. To this end, we propose a novel interface by exposing the NIC hardware functionality to DPDK applications via the PMD, requiring changes in the kernel driver to allow access to low-level messages and passing messages directly to the NIC.

II. Hardware Offloading HQoS onto Intel NICs through DPDK

A NIC can be logically partitioned into multiple virtual interfaces, known as Virtual Functions (VF), using Single Root Input/Output Virtualization (SR-IOV). Each VF can be directly bound to a DPDK Poll Mode Driver (PMD), with its own exclusive queues, resources and control structures. Communication between the Physical Function (PF) and the VFs occurs through a pair of mailbox receive and transmit queues, known as the `virtchnl` [8]. The channel is used to configure VF features, such as queue settings, RSS, and offloads such as TSO and checksum calculation.

QoS offloading on the Intel E810 is configured using a set of `virtchnl` messages that set up the Tx scheduler. This scheduler implements HQoS for fine-grained enforcement of rate limits and transmit priorities. The Tx scheduler is made up of scheduler nodes, each defined by parameters such as its position in the hierarchy, parent-child relationships, priority and shaper configuration. The Intel E810 establishes the HQoS scheduler hierarchy with the outgoing interface as the root node, the VFs as inner nodes, and the Tx queues as the leaf nodes, with their positions determined by the tree height set via firmware. Figure 1 illustrates the transmit path of the Intel E810, highlighting both the scheduler and message passing mechanisms [9].

Access to these advanced offloading features on the Intel E810 can be programmed through `switchdev` mode, accessible through the `devlink` tool[1]. This enables two key features on the NIC. First, it provides the control of the integrated switch, allowing integration with a control plane such as OVS[2] or Linux bridge. Second, it enables direct management of the transmit scheduler.

But `switchdev` has limited compatibility with DPDK. As of DPDK 24.11[3], support for HQoS offload in the IAVF PMD is restricted. For this driver, QoS configuration is available through RTE_TM[4], but is limited to DCB functionality [10]. Furthermore, configuration of the entire HQoS tree is not possible through `devlink`. While `devlink` can configure nodes above the VF, DPDK control of the VF removes the interface from Linux, making the scheduler parameters of leaf nodes, or other nodes below the VF, inaccessible through this interface.

To overcome these limitations in the current driver, we have implemented a side-channel API to directly control advanced NIC features, what we call `PMDlink`. The side channel facilitates communication between the DPDK userspace and PF via `virtchnl`. The main addition to implement the side-channel is a new API method `rte_eth_dev_send_vf_msg` in the DPDK ethernet device API and implemented on the IAVF PMD, which ensures that other PMDs could implement similar functionality. Further additions are the following:

- modify the ice [5] driver to accept the HQoS configuration messages. This assumes no DCB/ ETS configuration, as this mode and HQoS operation are mutually exclusive.
- extend the message structure to expose the mapping between `tx queue id` and `scheduler node id` to userspace/IAVF driver.

III. Preliminary Results

We showcase the capabilities of DPDK packet forwarding and NIC-accelerated HQoS with a small benchmarking experiment on a real test-bed. The testbed is the same as the one documented in our previous work [5]. Four packet flow groups are created, based on the TOS field in the IP header. The first flow group carries high-priority traffic (178-byte packets, 1 Gbps) to the first VF, while the other three groups represent background traffic (1518-byte packets) distributed across the remaining VFs, with loads varying between 0-95 Gbps. We measure the latency of the traffic stream directed to the first VF.

The NIC Tx scheduler is configured to prioritize incoming traffic from the first VF, while the other three are served in a WRR, with all weights equal and set to default 1. Furthermore, a 50 Gbps peak shaper is added on the parent node of all VFs. This illustrates the latency of a high-priority stream when combining software packet processing and with hardware-accelerated HQoS.

The result of the experiment is visible in Figure 2, showing latency under 25 µs at the worst case scenario.

Key Takeaway Software packet processing and hardware HQoS can process traffic at low latency and high data rates.

[1]https://wiki.linuxfoundation.org/networking/iproute2

[2]https://www.openvswitch.org/

[3]https://github.com/DPDK/dpdk/tree/v24.11

[4]https://doc.dpdk.org/guides/prog_guide/ethdev/traffic_management.html#traffic-management-api

[5]https://github.com/intel/ethernet-linux-ice/

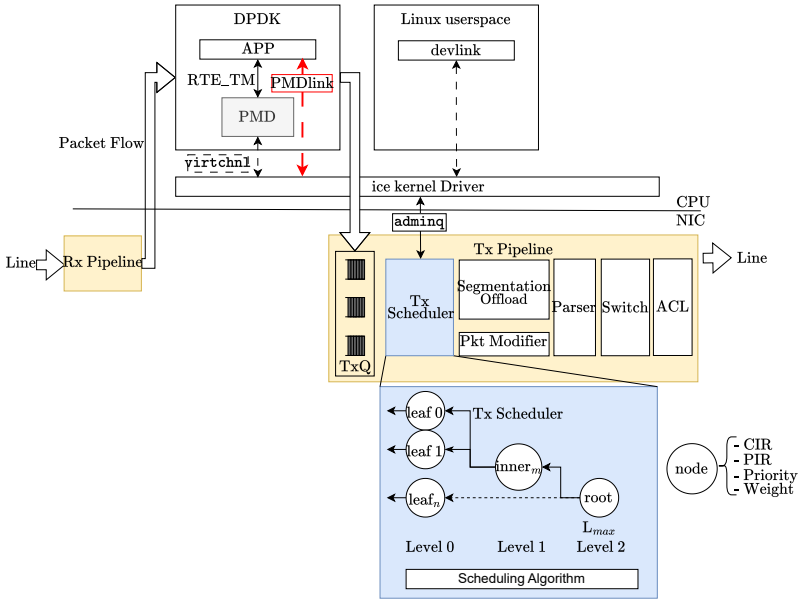

Fig. 1: Intel E810 pipeline overview [9].

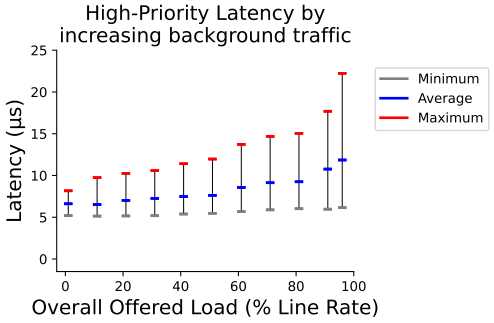

Fig. 2: The plot shows the latency of high-priority traffic as the total offered load increases, with minimum, average, and maximum latency values. The first value is plotted at 1 Gbps, reflecting no background load. The last value is plotted at 95 Gbps. The overall shaper for all VFs is configured for 50% of Line Rate, for a 100 Gbps interface.

IV. Conclusion and Future Work

In this work we introduce `PMDlink`, an extension to DPDK `eth_dev` API that enables direct control of low level HQoS of the NIC. Specifically, `PMDlink` provides a generic interface to hardware packet processing functions, while retaining the flexibility of software-defined processing. This ensures that critical networking tasks, such as latency-sensitive packet scheduling, can now be executed at hardware speeds without sacrificing flexibility.

But the need for a side-channel mechanism highlights a broader gap between hardware capabilities and their accessibility through standard driver interfaces. Generic API implementations such as DPDK are useful, but modular interfaces are needed to integrate functionality that may not be supported across all platforms. In future work, we plan to evaluate the capabilities of hardware offloading features in greater detail, including the impact of number of queues on latency or using different scheduling algorithms on different commodity NICs. While the focus of this article is in HQoS, the flexible nature of `PMDlink` can be used as a configuration interface to configure other functionality in the NIC, such as the Flow Director or integrated switch.

Acknowledgements

Parts of this work has been funded by the Bavarian State Ministry of Education and Culture, Science and Art through the High-Tech Agenda (HTA).

References

[1] P. Shinde, A. Kaufmann, T. Roscoe, and S. Kaestle, "We need to talk about NICs,"

[2] L. Fu, H. Zhang, Z. Zhang, J. Li, and H. Guan, "FlowLever: Leverage Flow Director for Packet Dispatch Acceleration in NFV," *IEEE Access*, vol. 12, pp. 36122–36134, 2024. Conference Name: IEEE Access.

[3] B. Constantine and R. Krishnan, "Traffic Management Benchmarking," Request for Comments RFC 7640, Internet Engineering Task Force, Sept. 2015. Num Pages: 51.

[4] F. Fejes, S. Nadas, G. Gombos, and S. Laki, "DeepQoS: Core-Stateless Hierarchical QoS in Programmable Switches," *IEEE Transactions on Network and Service Management*, vol. 19, pp. 1842–1861, June 2022.

[5] R. Figueiredo, H. Woesner, A. Kassler, and H. Karl, "Quality of Service Performance of Multi-Core Broadband Network Gateways," in *2024 8th Network Traffic Measurement and Analysis Conference (TMA)*, (Dresden, Germany), pp. 1–10, IEEE, May 2024.

[6] G. Kim and W. Lee, "Network Policy Enforcement With Commodity Multiqueue NICs for Multitenant Data Centers," *IEEE Internet of Things Journal*, vol. 9, pp. 6252–6263, Apr. 2022.

[7] Y. Kuperman, M. Mikityanskiy, and R. Efraim, "Hierarchical QoS Hardware Offload (HTB)," https://netdevconf.info/0x14/pub/papers/44/0x14-paper44-talk-paper.pdf.

[8] "Intel® Ethernet Adaptive Virtual Function (AVF) Hardware Architecture Specification (HAS)," 2018.

[9] "Intel® Ethernet Controller E810 Datasheet_rev2_8-1," Sept. 2024.

[10] "IEEE 802.1 Data Center Bridging Task Group."