Technology Guide

Intel® Data Streaming Accelerator (DSA) - Packet Copy Offload in OVS with Intel® DSA

Authors

Sunil Pai G

Bruce Richardson

Cian Ferriter

Harry Van Haaren

Ian Stokes

1 Introduction

Open vSwitch (OVS) is a software production quality, multilayer virtual switch licensed under the open-source Apache 2.0 license and focuses on the need for automated and dynamic network control in large-scale Linux-based virtualization environments.

OVS provides support for both kernel as well as user space packet switching. The latter can make use of Data Plane Development Kit (DPDK), which is a set of user space libraries that enable a user to create optimized performant packet processing applications and supports both physical and virtual interfaces.

Many deployments today utilize virtual interfaces like vHost/virtio extensively to steer packets to and from the Virtual Machines (VM). These virtual interfaces utilize packet copy to transfer messages between the VM and OVS running on the host. In such scenarios, the computational cost of performing those copies can be significant, especially for larger packet sizes. This guide describes how such packet copies can be offloaded to copy or Direct Memory Access (DMA) accelerators[1], such as the Intel® Data Streaming Accelerator, by utilizing the new DPDK DMA-offload library, "dmadev", through the DPDK vHost library.

This document is part of the Network Transformation Experience Kits.

[1] Note that DMA and DSA are used interchangeably in this document.

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	December 2022	Initial release.
002	January 2023	Revised for public distribution on Intel Network Builders.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
AVX-512	Intel® Advanced Vector Extensions 512 (Intel® AVX-512)
DPDK	Data Plane Development Kit
DPIF	DataPath InterFace, OVS datapath component that represents the software datapath as a whole
DPCLS	Datapath Classifier, OVS software datapath component that performs wildcard packet matching
DSA	Intel® Data Streaming Accelerator (Intel® DSA)
DMA	Direct Memory Access
DWQ	Dedicated Work Queue
D2K	Direct-to-UPI
SWQ	Shared Work Queue
EMC	Exact Match Cache, OVS software datapath component that performs exact packet matching
IA	Intel Architecture
IP	Internet Protocol
MFEX	Miniflow Extract
NIC	Network Interface Card
NUMA	Non-Uniform Memory Access
NDR	No Drop Rate
OVS	Open vSwitch
PMD	Poll Mode Driver
SMC	Signature Match Cache, OVS software datapath component that performs wildcard packet matching
VXLAN	Virtual Extensible LAN
VLAN	Virtual local area network

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
Intel® DSA specification	https://software.intel.com/content/www/us/en/develop/articles/intel-data-streaming-accelerator-architecture-specification.html
Open Source blog at 01.org	https://01.org/blogs/2019/introducing-intel-data-streaming-accelerator
Intel® DSA driver GitHub* repository	https://github.com/intel/idxd
Intel® Data Mover Library (DML)	https://github.com/intel/DML
Intel® DSA perf micros	https://github.com/intel/dsa-perf-micros
Accel-config	https://github.com/intel/idxd-config
DPDK dmadev documentation	https://doc.dpdk.org/guides/prog_guide/dmadev.html
DPDK dmadev IDXD driver documentation	http://doc.dpdk.org/guides/dmadevs/idxd.html
Open vSwitch - Optimized Deployment Benchmark	https://networkbuilders.intel.com/solutionslibrary/open-vswitch-optimized-deployment-benchmark-technology-guide
OVS documentation	https://docs.openvswitch.org/
DPDK documentation	https://doc.dpdk.org/guides/index.html
Intel® Architecture Instruction Set Extensions Programming Reference	https://www.intel.com/content/www/us/en/content-details/671368/intel-architecture-instruction-set-extensions-programming-reference.html
Intel® Data Streaming Accelerator (DSA) - Packet Copy Offload in DPDK with Intel® DSA Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-data-streaming-accelerator-packet-copy-offload-dpdk-technology-guide
Intel® Data Streaming Accelerator (DSA) - Accelerating DPDK Vhost Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-data-streaming-accelerating-dpdk-vhost-technology-guide

2 Overview

For high-performance packet processing applications like OVS, the packet pipeline is written in such a way as to minimize or eliminate packet data copies. Where necessary, these copies can take up a significant number of processor cycles and can become a significant performance bottleneck.

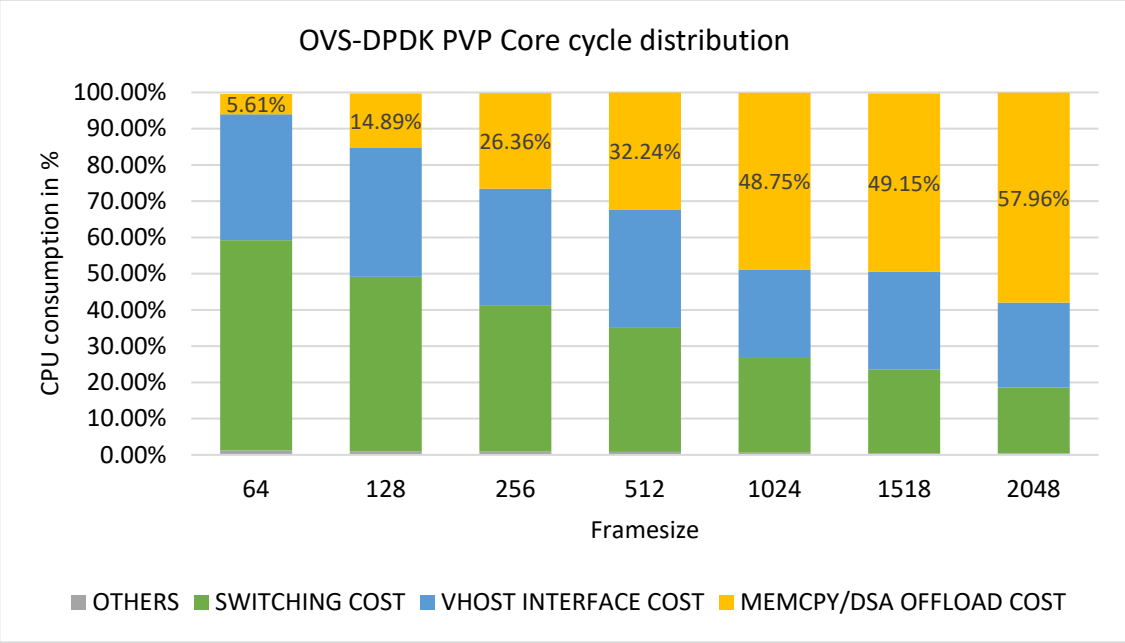

Figure 1. OVS-DPDK Core cycle distribution[2]

3 Intel® Data Streaming Accelerator (Intel® DSA) Hardware Accelerator

Modern hardware platforms, such as those using the latest Intel® Xeon® processors, often include a variety of accelerators to increase performance of selected workloads. Copying data is no exception to this, and DMA accelerator devices, as they are called, can be used on many platforms to have copy, fill, and other operations performed by the accelerator, thereby saving CPU core cycles for other parts of the application workload.

One such DMA acceleration device is the Intel® Data Streaming Accelerator (Intel® DSA), found on the 4th Gen Intel® Xeon® Scalable processor systems. This accelerator supports both streaming data movement, that is, copy and data transformation operations, applicable to a wide variety of applications. More details on this device can be found here.

4 DPDK DMA Device Support

Support for DMAdev devices has been added since DPDK 21.11 and the relevant vHost library changes, which consumes the DMAdev are in DPDK 22.11.

More details about the DMAdev capabilities and its configuration can be found here.

4.1 Device Assignment

The capability to use DMAdev for offloading vHost packet copy is exposed in OVS through a simple command:

```
# ovs-vsctl set Open_vSwitch . other_config:vhost-async-support=true[3]
```

The default value is set to false.

[2] As measured on Sapphire Rapids XCC (pre-production part) using Linux perf top: Test by Intel as of Sep 30, 2022, 1-node, 2x Intel(R) Xeon(R) Platinum 8490H, 60 cores, HT On, Turbo Off, Total Memory 512GB (16x32GB 4800 MT/s [4800 MT/s]), BIOS EGSDCRB1.86B.8901.P01.2209200239, microcode 0xab0000c0, 1x Intel Ethernet Network Adapter I225-LM, 4x Intel Ethernet Network Adapter E810-CQDA2 for QSFP, 1x 745.2G INTEL SSDSC2BB800G7, Ubuntu 22.04 LTS, 5.15.0-48-generic, gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, DPDK 22.07 (1c9a7fba5c net: accept unaligned data in checksum routines), OVS 3.0 (7eee450f8 datapath-windows: Fix icmp related error code), built with cflags "-g -Ofast -march=native", 1 core OVS-DPDK PVP setup with Ether/IP/UDP profile, guest application running dpdk-testpmd in checksum forwarding mode, lossy test.

[3] Note: Issuing this requires restarting the vswitchd daemon.

When this global configuration is set, each data-plane thread chooses and configures a DMAdev device based on its NUMA by iterating over the pool of available devices. This way the thread effectively owns the DMAdev device and is released back to the pool when the thread gets detached/destroyed. This per thread assignment model is chosen to best utilize the limited DMA resources available on the platform effectively and to avoid any contention among data-plane threads over the DMA device. When there are not enough DMA devices available for each data-plane thread, the threads without DMA companion will fall back to perform CPU copies, but in such cases performance may vary. Hence, it is recommended to ensure there are enough DMAdev devices available before starting OVS.

Other assignment schemes like mapping DMAdev device to each vHost queue and dynamic assignment were also considered but they introduce another layer of lock contention on DMA over the existing vHost queue locks causing traffic dependent scaling and hence these assignment schemes were not chosen.

4.2 Device Usage

Each DMA device owned by the data-plane threads is viewed merely as a copy offload resource. The device id of this resource is passed to the DPDK vHost library through their APIs, which then do the actual offloading of the copy and hence the device usage is abstracted from OVS. A logical view of how DSA is used in OVS is shown below:

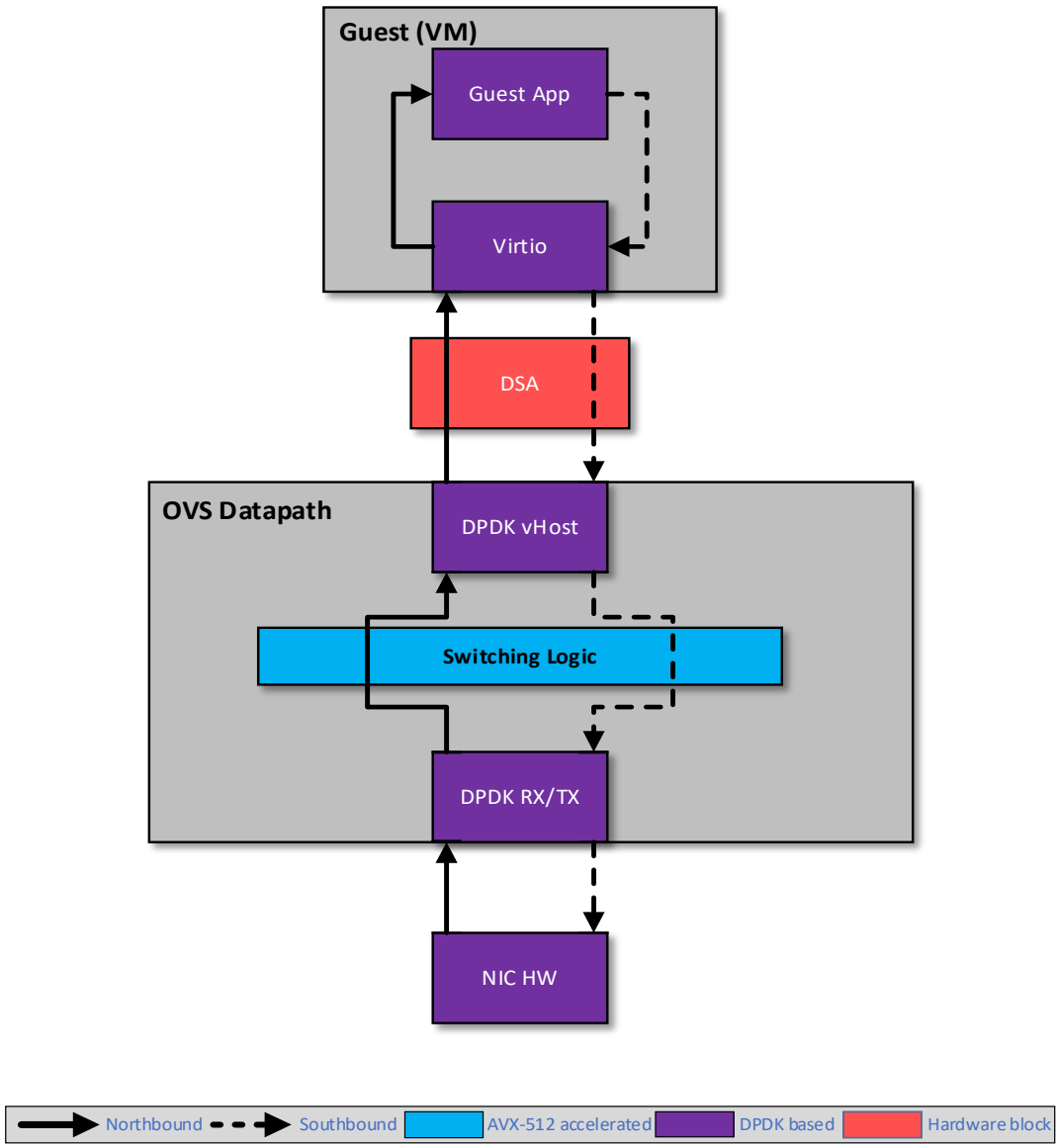

Figure 2: Logical view of DSA usage in OVS with 1 data plane thread.

More details on the device usage by the DPDK vHost library can be found here.

4.3 Submitting and Handling DMA Operations

DMA devices are asynchronous by nature and hence the copies once submitted must be checked for completion sometime later in the pipeline. This asynchronous utilization is very advantageous since it enables the CPU to perform other useful tasks like fetching the next set of packets and switching them while the copies are being carried out by the DMA, in parallel.

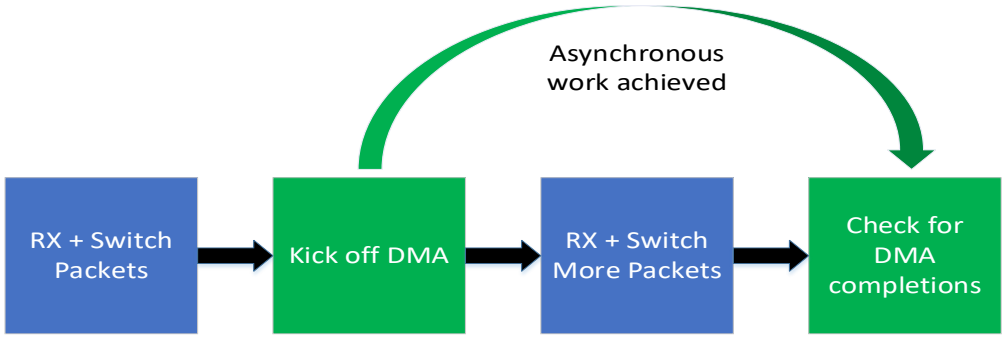

Figure 3: Simplified Asynchronous pipeline.

The vHost library provides a set of async API's for both enqueue (vHost Tx) and dequeue (vHost Rx) paths, which utilize the DMAdev APIs to offload the packet copy. Since the DMA utilization is asynchronous, for enqueue path, vHost library offers two separate APIs – one to submit the copies to DMA, another to check for completions and notify the VM of new packets. For the dequeue path on the other hand, vHost library offers a single API that performs both DMA submit as well as checks for completions. This is possible because the dequeue interface is always polled and hence the copies that were submitted in the current iteration could be checked for completions in the next one.

OVS design today is quite synchronous in nature, and it becomes very tricky to enable such asynchronous DMA acceleration without a proper framework to do so. To address this, the deferral of work framework is introduced in OVS. This framework allows the packet completion checks to be deferred sometime later in the pipeline.

4.3.1 Deferral of work

As the name suggests, this framework provides a way to defer or delay an operation to be executed sometime in future. The framework uses work rings per data-plane thread, which serve as a FIFO queue to keep track of the asynchronous work items that are deferred. The work item is added to the work ring when a Tx operation on a netdev returns -EINPROGRESS indicating it kicked off an asynchronous Tx. The work ring is then dequeued sometime later and the work items are executed in two cases:

a) In the main PMD loop after processing every RXQ assigned to the data-plane thread
b) When the work ring is full while trying to queue more work items

OVS today provides a common abstraction of devices called netdev and operations for any device type (both physical and virtual) is through this common interface. To meet this design requirement in OVS, the work item to be deferred also needs to be part of the netdev interface like the netdev_send, and hence a new function pointer is introduced for this purpose per netdev called netdev_process_async. Figure 4 shows how the packet switching pipeline operates including deferral of work.

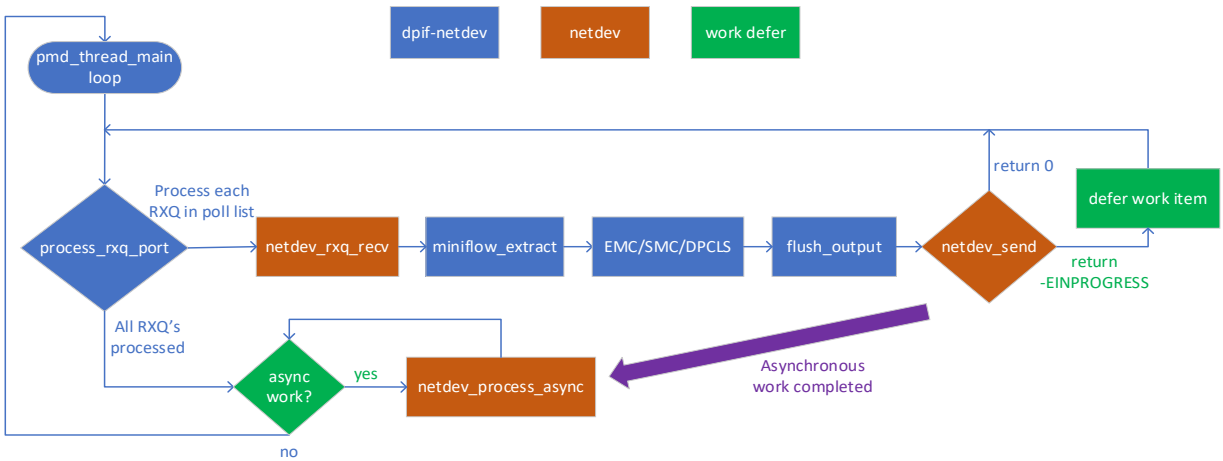

Figure 4: Deferral of work pipeline

For vHost interfaces, the netdev_process_async is initialized to a function that calls the required vHost Tx API, which checks the DMA packet completions and notifies the VM when new packets are available for its consumption.

Other alternate architectures were also explored like hiding the vHost completions for each Tx queue under its corresponding sibling Rx queue wrapper and having a lockless ring in between the parts of the pipeline – netdev_send and the vHost enqueue (DMA enqueue), to avoid the contentions on the Tx queue being operated on. It turned out that this approach was not as performant as the deferral of work, as well as, architecturally deemed incorrect as it mixes Rx and Tx operations under one wrapper leading to a potential inconvenience in terms of debugging and maintainability.

5 Performance

The CPU cycle distribution with such a design (as described in 4.3.1) is shown in Figure 5.

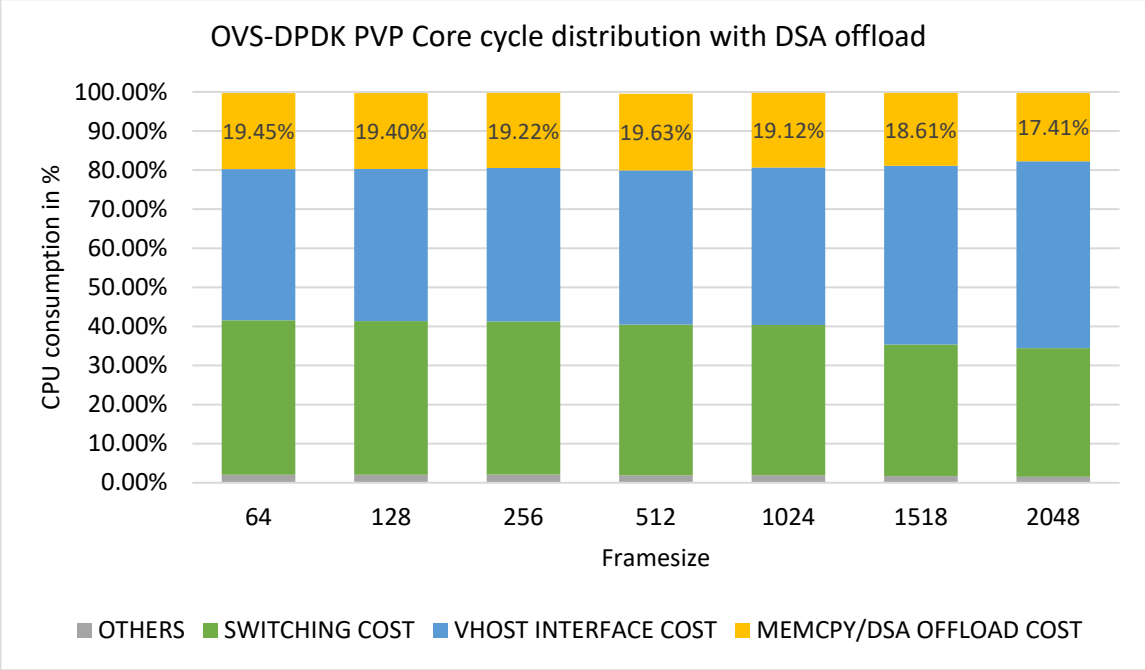

Figure 5: Core cycle distribution with DSA offload[4]

As clearly shown in the above figure, compared to Figure 1, the packet copy is eliminated and is substituted by the DMA offload cost, which remains fixed across packet sizes depending on the number of packets per burst. As a result of this asynchronous offload, the switching cost and the interface cost take a higher proportion of CPU time because the CPU can process or switch more packets. Because of these two reasons, the throughput or bandwidth remains stable across packet sizes up to 2048 byte packets. The figure also evidently indicates that DMA offload can yield better throughput for packet sizes > 256 bytes[5].

5.1 Performance Tunings

5.1.1 CPU copy optimization for small packets

DSA copy is observed to be better performant for large packets compared to CPU, but for small packets, the cost of creating the descriptor and awaiting the completion is significant compared to the time to do the copy on the CPU. This can lead to scenarios where small copies are faster when performed on the CPU. So, users can get the best of both modes if DSA is used for large packets and CPU for small packets while preserving the packet ordering.

[4] As measured on Sapphire Rapids XCC (pre-production part) using Linux perf top: Test by Intel as of Sep 30, 2022, 1-node, 2x Intel(R) Xeon(R) Platinum 8490H, 60 cores, HT On, Turbo Off, Total Memory 512GB (16x32GB 4800 MT/s [4800 MT/s]), BIOS EGSDCRB1.86B.8901.P01.2209200239, microcode 0xab0000c0, 1x Intel Ethernet Network Adapter I225-LM, 4x Intel Ethernet Network Adapter E810-CQDA2 for QSFP, 1x 745.2G INTEL SSDSC2BB800G7, Ubuntu 22.04 LTS, 5.15.0-48-generic, gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, DPDK 22.07 (7bdd32de18a8 vhost: add special DMA ID (-1) support for cpu fallback), OVS 3.0 (70816755a943 netdev-dpdk: Enable vhost async API's in OvS.), built with cflags "-g -Ofast -march=native", 1 core OVS-DPDK PVP setup with Ether/IP/UDP profile, guest application running dpdk-testpmd in checksum forwarding mode, lossy test, 1 DSA instance bound to vfio-pci, vhost async threshold set to 0.

[5] Given packet size is for the specified configuration as per figure 5. The exact packet size and the throughput where OVS-DSA may be better compared to CPU case may vary depending on the workload and system configuration.

Hence, users are given the flexibility to choose a threshold (in bytes) at runtime beyond which packets must be offloaded to DMAdev by the vHost library using the below command:

```
# ovs-appctl netdev-dpdk/set-vhost-async-thresh <threshold_size>
```

The default threshold size is set to 128 bytes.

Similarly, users can also fetch the current threshold set using the command:

```
# ovs-appctl netdev-dpdk/get-vhost-async-thresh
```

5.1.2 VFIO-PCI / kernel IDXD bound DSA

Based on our experiments, for DSA 1.0 on SPR systems, it's recommended to use vfio-pci driver bound DSA devices over the kernel idxd driver bound devices for this workload.

5.1.3 Intel® Advanced Vector Extensions 512 (Intel® AVX-512)

Most parts of the packet processing pipeline in OVS like Miniflow Extract, DPCLS, DPIF, and others have been optimized with Intel® Advanced Vector Extensions 512 (Intel® AVX-512). Since it drastically reduces the processing cycles of these precursor steps to vHost DMAdev offload, it increases the rate of DMAdev enqueue and therefore elevates the overall OVS throughput. Hence, enabling Intel AVX-512 along with DMAdev offload is highly recommended to make the most out of the features offered by the CPU/platform.

For more information on how to enable Intel AVX-512 in OVS, refer to the official OVS documentation here.

5.1.4 Output Packet Batching

This feature in OVS is found to be quite beneficial for DMAdev offload as it increases the enqueue size into DSA and hence the throughput. This also reduces the possibility of the packets being copied by the CPU as a fallback option at the vHost interface because of DSA ring full scenarios where traffic pattern is quite scattered, as it increases the effective utilization of batch descriptors. It's also important to note that the additional throughput seen might come at the cost of increased latency. For more details, refer to the official documentation on output batching in OVS.

5.1.5 DSA WQ queue-depth

You may see a minor drop in throughput when DSA runs out of available batch descriptors/slots in its DWQ, and CPU copy is used as a fallback option. This behavior is possible for cases where the configured DSA WQ queue-depth is low and/or the number of packets per batch is low, leading to sub-optimal use of DSA batch descriptors. Hence, it is recommended to configure the WQ queue-depth to a sufficient value, such as 16/32 per WQ. For vfio-pci bound DSA device, DPDK by default allocates the WQ queue-depth based on the total WQ-size available per device (128 in 4th Gen Intel Xeon Scalable processor) by splitting them uniformly (8 DWQ in 4th Gen Intel Xeon Scalable processor, hence 16 queue-depth per DWQ).

5.2 Latency Impact

The maximum and average latency at a fixed packet rate for a given packet size with DMA offload has been observed to be better for packet sizes greater than 256 bytes compared to CPU. Also, at peak NDR bandwidth, for a given packet size, DMA offload may result in higher latencies compared to CPU but it's important to note that the bandwidth difference is vast (DMA offload bandwidth being higher) in such cases.

5.3 Other Observations

5.3.1 Packet segmentation

Offloading costs to Intel DSA may increase with segmented/chained packets, for example, packets whose size is beyond the mbuf size (default size of 2KB) being split into multiple mbuf. This implies more DSA enqueues per packet and therefore increases the offload cost and impacts performance.

6 Summary

In summary, it is noted that using Intel DSA offload along with Intel AVX-512 optimizations in OVS for packet sizes beyond 256 bytes[6], compared to vanilla OVS, saves considerable CPU cycles, which can be spent on switching, to significantly increase throughput.

[6] Given packet size is for the specified configuration as per figure 5. The exact packet size and the throughput where OVS-DSA may be better compared to CPU case may vary depending on the workload and system configuration.

0123/DN/WIT/PDF 758697-002US