PERFORMANCE BENCHMARK REPORT

Intel Corporation
Software Defined Datacenter Solutions Group

Enhanced Platform Awareness in Kubernetes

Intel® Xeon® Scalable Processors

1.0 Executive Summary

Enhanced Platform Awareness (EPA) represents a methodology targeting intelligent platform capability, configuration and capacity consumption. EPA delivers improved and deterministic application performance, and input/output throughput.

EPA underpins a three-fold objective of the discovery, scheduling and isolation of server hardware capabilities. Intel® and partners have worked together to make the following technologies available in Kubernetes*, the leading container orchestration engine (COE) for production-grade container scheduling and management:

- Node Feature Discovery (NFD) enables Intel Xeon® Processor-based platform capability discovery in Kubernetes
- CPU Manager for Kubernetes (CMK) provides a mechanism for CPU core pinning and CPU core isolation of containerized workloads
- Huge page support (a native feature in Kubernetes v 1.8) enables the discovery, scheduling and allocation of huge pages as a native first-class resource
- Single Root I/O Virtualization (SR-IOV) for networking

This performance benchmarking report demonstrates how using the above technologies can enhance container application performance. The aim of the benchmarking was two-fold:

- To demonstrate data plane performance for containerized DPDK enabled application (testpmd*) and non-DPDK-enabled applications (using qperf*) using the following EPA features: CPU Pinning and Isolation, SR-IOV; Huge Pages.
- To show how CPU core pinning and isolation prevent application impact from "noisy neighbor" applications (using stress-ng*) that consume many CPU cycles for both DPDK (testpmd) and kernel TCP/IP (qperf) applications.

To conduct the benchmark tests, a Kubernetes environment was setup on servers powered by Intel Xeon Gold Processors 6138T with 20 physical cores (40 hardware threads). A detailed list of software and hardware ingredients is available in Section 4.0

Contents

Highlights from the benchmark tests include:

- EPA enables DPDK applications to achieve 96% line-rate of a 25 GbE link for packet sizes larger than 512 bytes. Performance results were similar for DPDK applications running in containers versus running in the host.
  - Using SR-IOV for networking, huge pages and core pinning, the DPDK (testpmd) application in a container passed data at more than 20Gbit/s (40% line rate) of the 50 Gbps (dual 25 GbE NICs) network throughput for 64-byte packets (See Section 5.3.1). These results scale to more than 48Gbit/s (96% line rate) for 512-byte and larger packets for all container use cases. EPA thus enables DPDK applications to get similar performance in a container as compared to running in the host.
- Core pinning and core isolation improves predictability of the target workloads in both DPDK-based applications and non-DPDK applications in the presence of a noisy neighbor workload, i.e. stress-ng.
  - DPDK-based applications: When the DPDK testpmd application is run with stress-ng in a container without core isolation, the network throughput fluctuates significantly and drops more than 75% and packet latency increases more than 10 times for most packet sizes. (See Section 5.3.2)
  - Non DPDK-based applications: When the kernel network-based qperf runs inside a container with stress-ng without core pinning and core isolation features, network throughput and packet latency vary widely. Network throughput drops by more than 60%, while packet latency increases by more than 40 times for most message sizes for both TCP and UDP traffic types. (See Section 6.2)

Note: The system used for this performance benchmarking report was based on the Intel Xeon Gold Processor 6138T CPU running at 2.00 GHz with 20 physical cores (40 hardware threads). Intel also offers CPUs with a higher number of cores, including the Intel Xeon Platinum Processor 8180 with 28 cores (56 hardware threads) running at 2.50 GHz. The aggregated system throughput in this test report is limited by the number of NIC ports used (2x25G). Xeon Scalable Processor-based systems, like the one used in this report, are capable of scaling to much higher network throughput as shown in a number of DPDK performance benchmarking reports available at http://dpdk.org/doc. Higher performance should be achievable when using more NIC ports and available cores in the system.

2.0 Introduction

For high-performance workloads that require particular hardware capabilities to achieve their target performance, the container orchestration layer needs to discover and match platform capability with workload requirements. EPA for Kubernetes allows these workloads to run on the optimal available platform and achieve the required service level objectives and key performance indicators (KPIs).

This document will describe the tested benefits of the following technologies:

- CPU Manager for Kubernetes (CMK) provides a mechanism for CPU pinning and isolation of containerized workloads
- Node Feature Discovery (NFD) enables Intel Xeon Processor server hardware capability discovery in Kubernetes
- Huge page support is native in Kubernetes v1.8 and enables the discovery, scheduling and allocation of huge pages as a native first-class resource

To simulate real application performance for these tests, the following software tools were used:

1. testpmd, a Data Plane Development Kit (DPDK)-based application, configured in I/O forwarding mode.

   Note: CPU pinning and huge pages are required in order to run DPDK applications like testpmd in a container (or VM).
2. qperf, a non-DPDK Linux kernel network-based traffic generation application, configured for TCP and UDP traffic.
3. Stress-ng, an application used to simulate a noisy neighbor workload. Stress-ng is designed to exercise various physical subsystems of a computer as well as various operating system interfaces. For these tests, stress-ng is used to generate CPU load on all the cores available to the stress-ng application.

This document is written for software architects and developers who are implementing and optimizing container-based applications on bare metal hosts using Kubernetes and Docker. It is part of the Container Experience Kits for EPA. Container Experience Kits are collections of user guides, application notes, feature briefs and other collateral that provide a library of best-practice documents for engineers who are developing container-based applications. Other documents in this Experience Kit can be found online at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

An additional list of resources is located in Appendix D: along with links for downloading. The appendix also lists links to GitHub repositories for the software required to enable EPA for Kubernetes.

3.0 Performance Test Scenarios

A total of eight performance test scenarios (see summary in Table 3-1) were designed in order to demonstrate how applications using EPA can achieve optimal performance in a container environment running on Intel's Xeon Scalable Processors. Furthermore, these test scenarios show that using core pinning and core isolation can negate the noisy neighbor impact and achieve consistent results for a target application.

The following software applications were used for these test scenarios:

- testpmd DPDK user-mode application. DPDK is a set of libraries providing a programming framework to enable high-speed data packet networking applications. Applications using DPDK libraries and interfaces run in user mode and directly interface with NIC functions, skipping slow, kernel layer components to boost packet processing performance and throughput. These applications process raw network packets without relying on protocol stack functionality provided by kernel. For more information on DPDK go to http://www.dpdk.org.
- Linux qperf kernel network application. Applications using the kernel network stack are designed to utilize protocol and driver stack functionality built into the kernel.

Figure 3-1 shows the container environment, including application stacks running inside containers. The figure shows stacks that are using DPDK libraries in addition to the Linux kernel network stack. In addition, the image shows the stress-ng application that does not need to use the networking stack to generate the stress load on system cores.

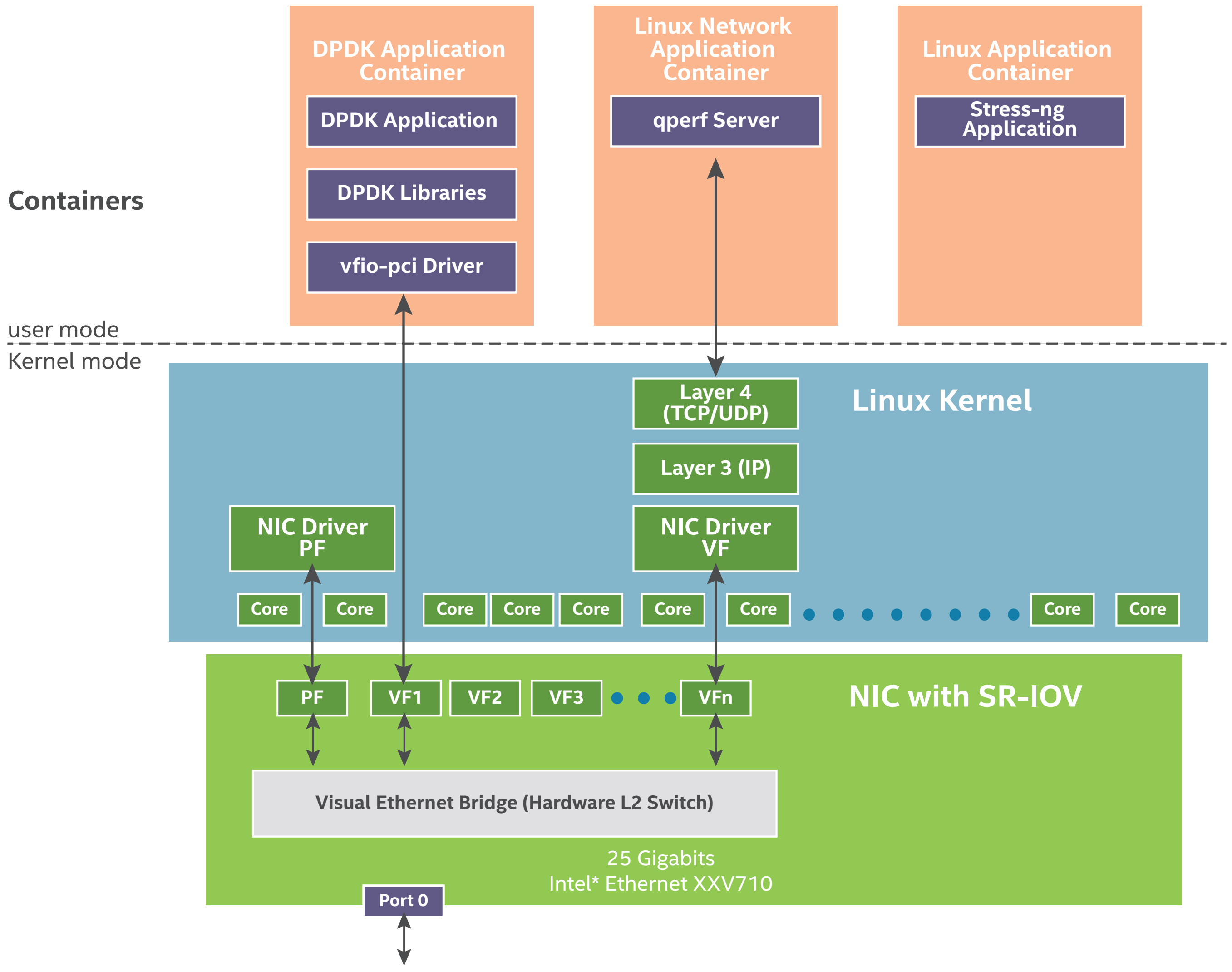

Figure 3-1 Layered stack for DPDK application container and kernel network application containers

Without CPU core pinning and CPU core isolation, Kubernetes may place the noisy neighbor container on the same physical core as the container hosting the target application, thus impacting application performance. The performance impact will vary depending on the CPU processing required by the noisy neighbor container on the assigned cores. The stress-ng application generates a workload equal to 50% of the processing available in each core, thus reducing the processing available to the application under test.

Table 3-1 summarizes the eight test case scenarios performed, the platform capabilities used in each scenario and the test configurations. A detailed list of software and hardware ingredients are listed in Section 4.0.

Table 3-1 Performance Test Scenarios

Test Application	DPDK user mode application (testpmd)				Kernel network driver application (qperf)			
Test Scenarios	**No-CMK**	**CMK**	**No-CMK w/ Noisy Neighbor**	**CMK w/ Noisy Neighbor**	**No-CMK**	**CMK**	**No-CMK w/ Noisy Neighbor**	**CMK w/ Noisy Neighbor**
SR-IOV	✓	✓	✓	✓	✓	✓	✓	✓
Huge Pages	✓	✓	✓	✓				
Core pinning	✓	✓	✓	✓		✓		✓
Core isolation		✓		✓		✓		✓
PF driver (Host)	i40e v2.0.30							
VF driver	vfio-pci				i40evf v2.0.30.			
DPDK (container)	v17.05							
Number of flow	256 bidirectional flows per container				1 uni-directional flow per container.			
Traffic type	IPv4 Traffic				UDP and TCP			
Host OS	Ubuntu* 16.04.2 x86_64 (Server) Kernel: 4.4.0-62-generic							
No of containers	1, 2, 4, 8 & 16							

4.0 Platform Specifications

Table 4-1 & Table 4-2 list the hardware and software components used for the performance tests.

4.1 Hardware ingredients

Table 4-1 Hardware ingredients used in performance tests

Item	Description	Notes
Platform	Intel Server Board S2600WFQ	Intel Xeon processor-based dual-processor server board with 2 x 10 GbE integrated LAN ports
Processor	2x Intel Xeon Gold Processor 6138T (formerly Skylake)	2.0 GHz; 125 W; 27.5 MB cache per processor 20 cores, 40 hyper-threaded cores per processor
Memory	192GB Total; Micron* MTA36ASF2G72PZ	12x16GB DDR4 2133MHz 16GB per channel, 6 Channels per socket
NIC	Intel Ethernet Network Adapter XXV710-DA2 (2x25G) (formerly Fortville)	2 x 1/10/25 GbE ports Firmware version 5.50
Storage	Intel DC P3700 SSDPE2MD800G4	SSDPE2MD800G4 800 GB SSD 2.5in NVMe/PCIe
BIOS	Intel Corporation SE5C620.86B.0X.01.0007.060920171037 Release Date: 06/09/2017	Hyper-Threading - Enable Boot performance Mode – Max Performance Energy Efficient Turbo – Disabled Turbo Mode - Disabled C State - Disabled P State - Disabled Intel VT-x Enabled Intel VT-d Enabled

4.2 Software ingredients

Table 4-2 Software ingredients used in performance tests

Software Component	Description	References
Host Operating System	Ubuntu 16.04.2 x86_64 (Server) Kernel: 4.4.0-62-generic	https://www.ubuntu.com/download/server
NIC Kernel Drivers	i40e v2.0.30 i40evf v2.0.30	https://sourceforge.net/projects/e1000/files/i40e%20stable
DPDK	DPDK 17.05	http://fast.dpdk.org/rel/dpdk-17.05.tar.xz
CMK	V1.0.1	https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes
Ansible*	Ansible 2.3.1.0	https://github.com/ansible/ansible/releases
Bare Metal Container RA scripts	Includes Ansible* scripts to deploy Kubernetes v1.6.4	https://github.com/intel-onp/onp
Docker*	v1.13.1	https://docs.docker.com/engine/installation/
SR-IOV-CNI	v0.2-alpha. commit ID: a2b6a7e03d8da456f3848a96c6832e6aefc968a6	https://www.ubuntu.com/download/server

5.0 Setting up the DPDK application performance test in containers using SR-IOV virtual functions

5.1 Test setup

The test setup for running testpmd as a workload inside a container is shown in Figure 5-1. The traffic is generated by Ixia IxNetwork test system (version 8.10.1046.6 EA; Protocols: 8.10.1105.9, IxOS 8.10.1250.8 EA-Patch1) running RFC 2544.

Up to 16 containers, each running the testpmd application, are instantiated using Kubernetes. Each container pod is assigned one virtual function (VF) instance from each physical port of the dual-port 25 GbE NIC for a total of two VFs per container pod. The maximum aggregated theoretical system throughput is thus 50Gbps for bidirectional traffic. Two ports are paired, one as ingress and other as egress in each direction (i.e., one 25 Gbps bidirectional flow consumes two ports), and traffic with 256 bidirectional flows is run through the system under test (SUT). All results are measured for 0% packet loss. A separate container running the stress-ng application is used to simulate a noisy neighbor application.

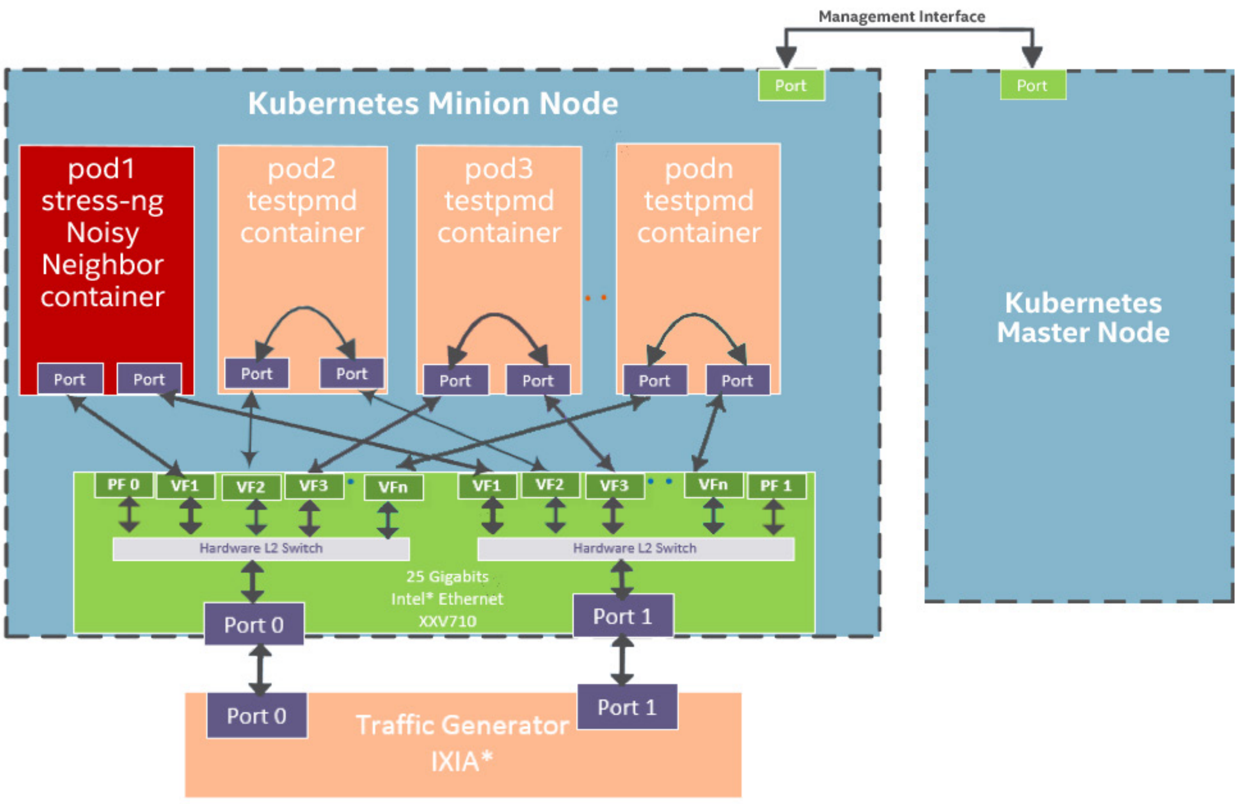

Figure 5-1 High-Level Overview of DPDK performance setup with SR-IOV VF using testpmd.

5.2 Traffic profiles

The IP traffic profiles used in these tests conform to RFC 2544:

- Packet sizes (bytes): 64, 128, 256, 512, 1024 and 1518
- L3 protocol: IPv4
- 256 bidirectional flows per container. Each flow has a different source and destination IP address.
- Bidirectional traffic with the same data rate being offered in each direction for 60 seconds.

5.3 Test results

5.3.1 Results of DPDK application performance in containers with EPA

The test results in Figure 5-2 compare the DPDK performance using testpmd in both a container and a host. Tests were run in each of these environments of the performance of physical functions (PF) and SR-IOV VFs. Tests are run in the host for PF-PF and VF-VF traffic using 2 x25G ports and testpmd that is assigned two logical sibling cores with hyper threading enabled. These results are compared to testpmd performance in container for VF-VF traffic. The results show that Kubernetes can run DPDK applications inside a container and get almost similar performance to when it is run inside the host, providing the benefit of EPA features SR-IOV, core pinning and huge pages to container-based environments.

Testpmd is assigned two hyper threaded sibling cores in each case. Results show the performance as system throughput in millions of packets per second (Mpps) and packet latency when running RFC 2544 tests with 0% frame loss for 2 25G ports.

The following is key to understanding the test codes:

- 2P_1C_2T_HOST_PF (gray bar) indicates the test configuration run with 2x25G ports and are assigned 1Core/2Threads with hyper thread enabled. The test is run inside host without container between PF-PF.
- 2P_1C_2T_HOST_VF (light blue bar) indicates the test configuration run with 2x25G ports and are assigned 1Core/2Threads with hyper thread enabled. The test is run inside a host without container between VF-VF.
- 2P_1C_2T_HOST_Container (dark grey bar) indicates the test configuration where the test is run with 2x25G ports and are assigned 1Core/2Threads with hyper thread enabled. The test is run inside container between VF-VF.

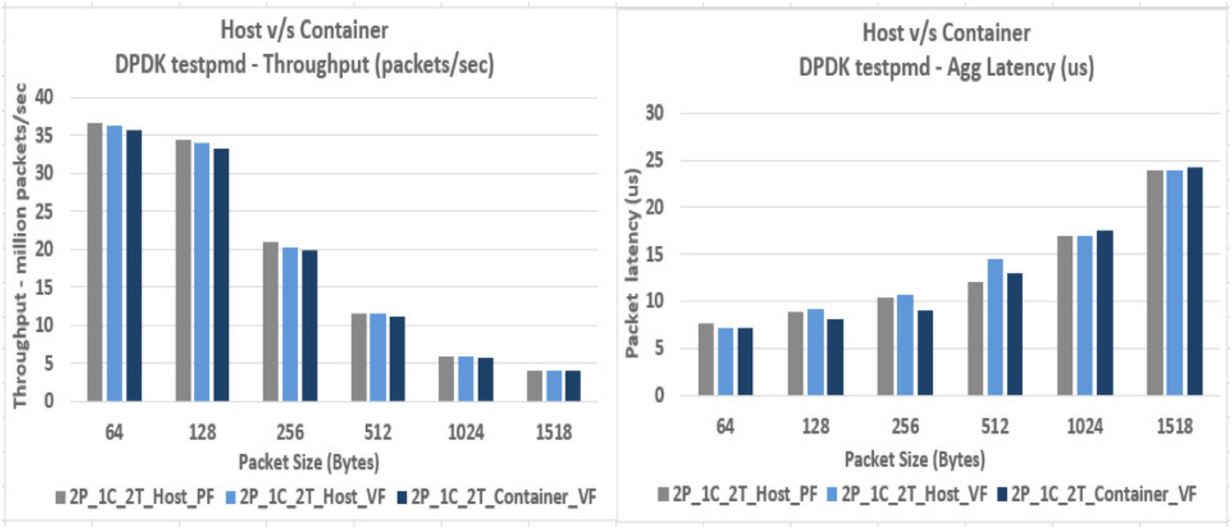

Figure 5-2 DPDK testpmd performance comparison for host versus container with EPA using 2 25G ports.

The test results in Figure 5-3 & Figure 5-4 below show DPDK performance running testpmd application in containers with up to 16 containers running concurrently in the same physical host and sharing the SR-IOV VFs from same 2x25 physical NIC ports. The results show that using SR-IOV, huge pages, core pinning and core isolation, provides more than 20Gbits/sec performance for 64-byte packets that scales to 48Gbits/sec (96% line rate) for packet sizes of 512 bytes and above for all container cases.

Testpmd in each container is assigned two separate hyper threaded sibling cores. Results show the performance as system throughput in packets/sec and Gbits/sec when running RFC 2544 test with 0% frame loss.

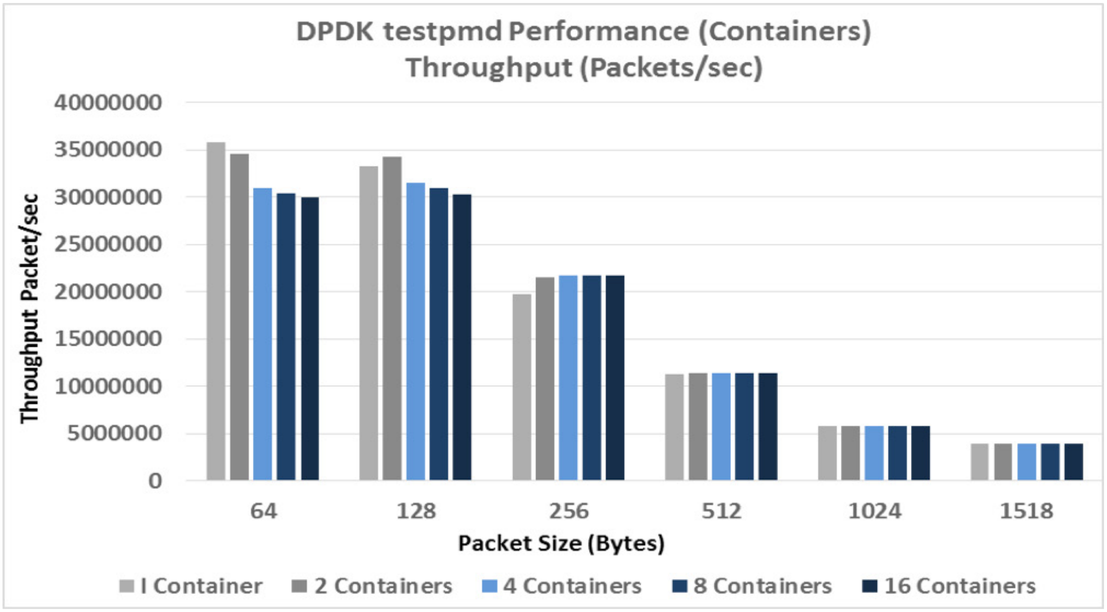

Figure 5-3 DPDK testpmd performance shown as packets/sec with multiple containers using EPA.

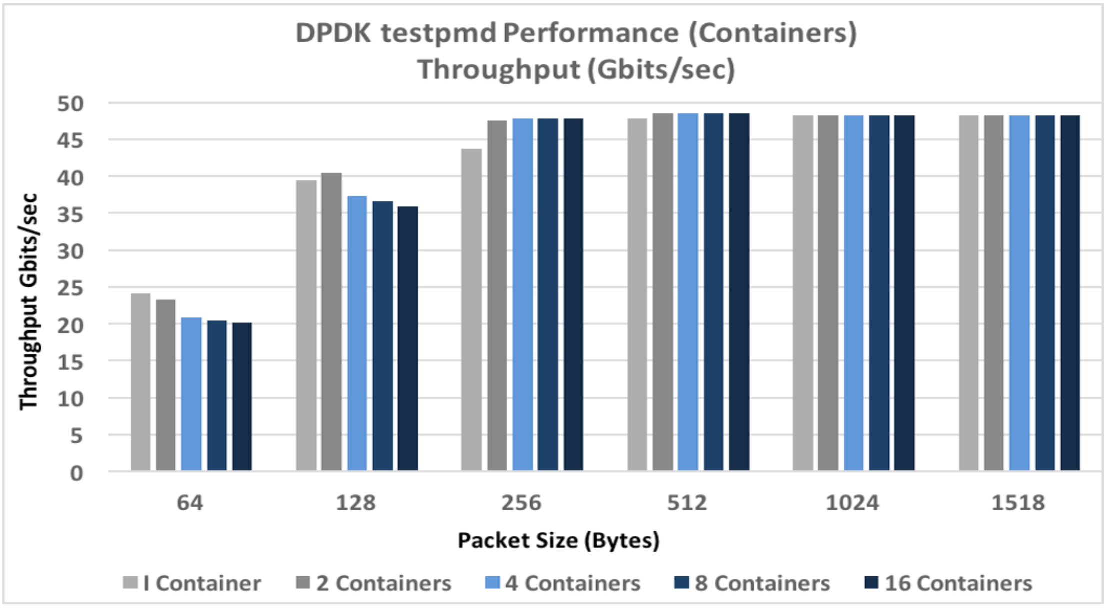

Figure 5-4 DPDK testpmd performance as Gbits/sec with multiple containers using EPA.

Note: The system used for this performance benchmarking report was based on the Intel Xeon Gold Processor 6138T CPU running at 2.00 GHz with 20 physical cores (40 hardware threads). Intel also offers CPUs with a higher number of cores, including the Intel Xeon Platinum Processor 8180 with 28 cores (56 hardware threads) running at 2.50 GHz. The aggregated system throughput in this test report is limited by the number of NIC ports used (2x25G). Xeon Scalable Processor-based systems, like the one used in this report, are capable of scaling to much higher network throughput as shown in a number of DPDK performance benchmarking reports available at http://dpdk.org/doc. Higher performance should be achievable when using more NIC ports and available cores in the system.

Detailed results for all container test cases are provided in Appendix B.1 & B.2. DPDK test results for all packet sizes for host tests are available in Appendix B.1

5.3.2 Test results of DPDK application performance in containers with and without CMK

The test results in this section show network throughput and packet latency for 16 containers running the testpmd application with and without a noisy neighbor container present and also when using CPU core pinning and CPU core isolation and when not using CPU core pinning and CPU core isolation.

The application containers are deployed using Kubernetes. CMK assigns two hyper-threaded sibling cores to each container application from its dataplane core pool. When running testpmd with CMK, the cores that are isolated and assigned via CMK are used to run the application. When running testpmd without CMK, two separate hyper-thread sibling cores are assigned to each testpmd instance manually.

Without CMK, Kubernetes may place the noisy neighbor container on the same physical core where the container under test is running. In this scenario, the noisy application may share the cores assigned to the application under test, thus impacting target application performance. The performance impact will vary depending on the load placed by the noisy container on the application assigned cores. In these tests, a load of 50% is generated on all available cores using stress-ng.

Tests data is collected and compared for the following use cases:

1. Without CMK and no noisy neighbor
2. With CMK and no noisy neighbor
3. Without CMK in presence of noisy neighbor
4. With CMK in presence of noisy neighbor

The results show a detrimental impact of having a noisy neighbor container when no CMK functionality is available compared to when CPU core isolation and CPU core pinning are available. This demonstrates how this technology alleviates the impact of noisy neighbors on application performance.

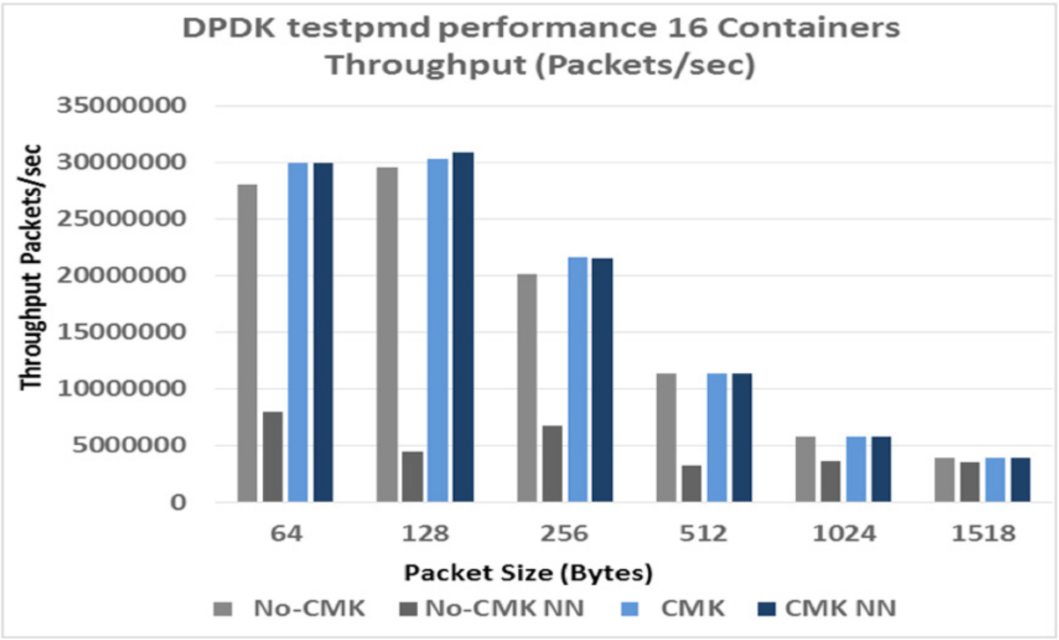

Figure 5-5 testpmd packets/sec with and without CMK and noisy neighbor for 16 containers.

As shown in Figure 5-6 & Figure 5-7:

- When running testpmd without CMK, the presence of a noisy neighbor container caused network throughput to degrade by more than 70% for packet sizes 512 bytes and smaller while the throughput is ~25% less for larger packet sizes.
- Similarly, packet latency increased by more than 20 times for most packet sizes.
- When running the testpmd using CMK, the performance is not impacted by having a noisy neighbor container in the system due the cores being isolated. As a result, running testpmd with CMK gets consistent performance. Detailed results for all container test cases are provided in appendices B.1 & B.2.

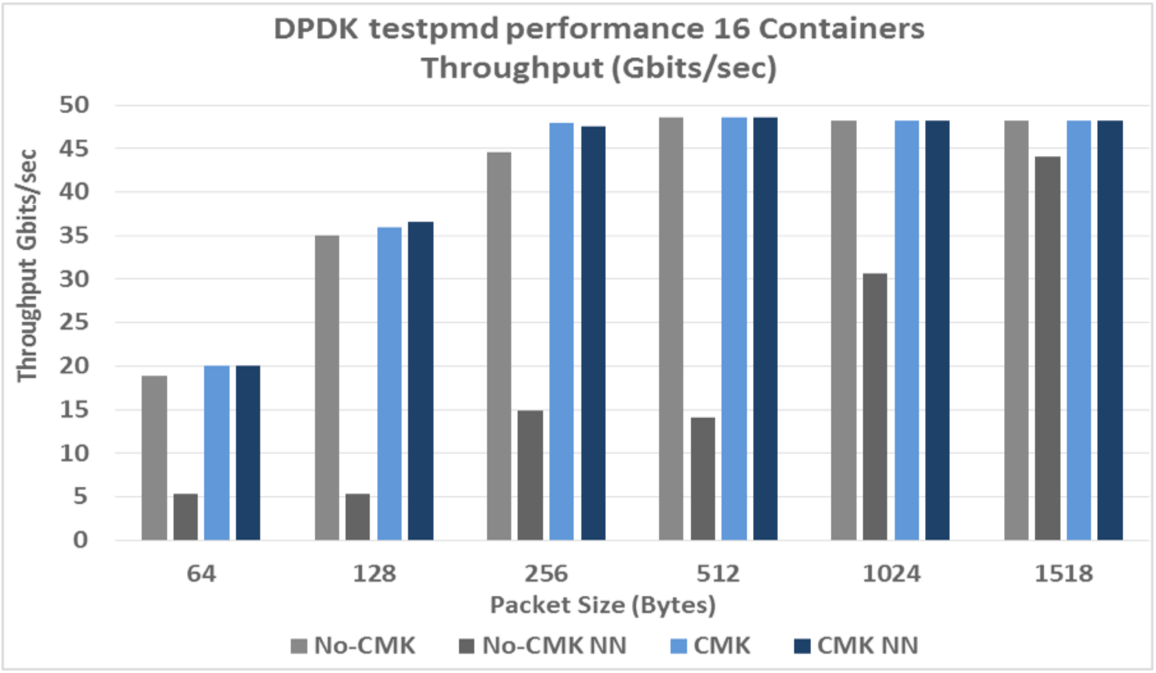

Figure 5-6 testpmd throughput with and without CMK and noisy neighbor for 16 containers.

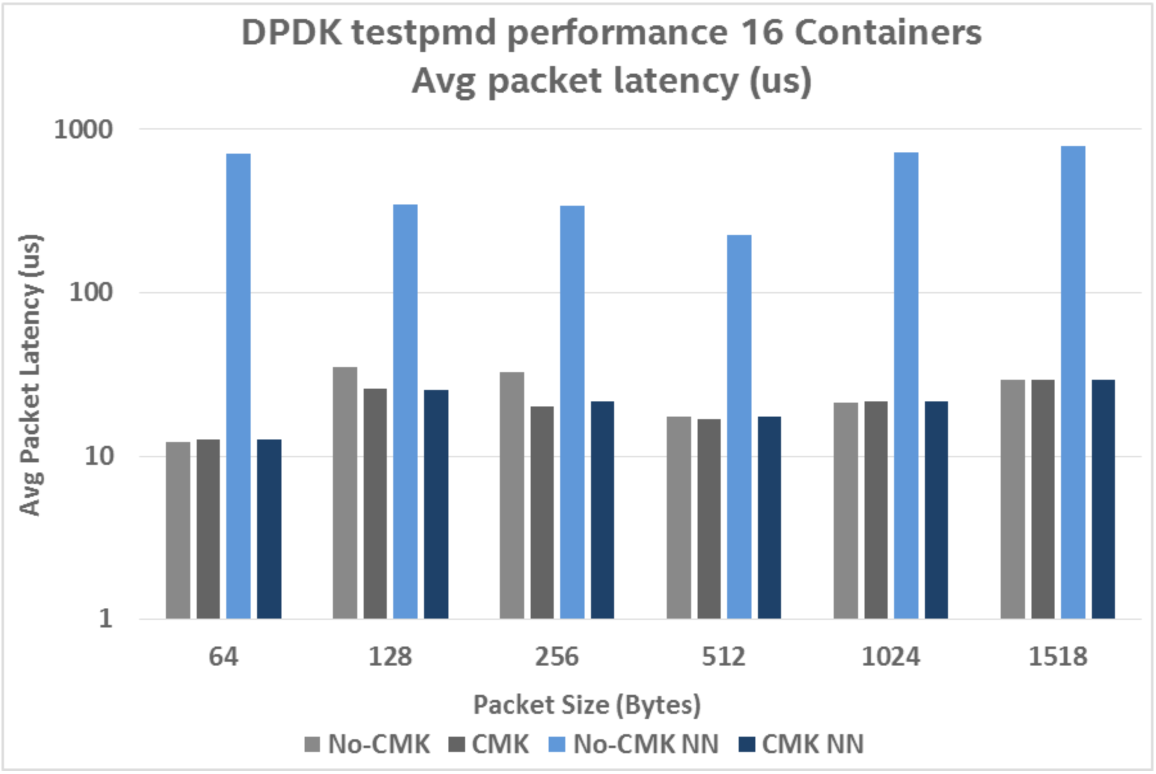

Figure 5-7 testpmd average packet latency with and without CMK and noisy neighbor for 16 containers.

6.0 Setting Up the Test of Kernel Network Application Performance in Containers Using SR-IOV Virtual Functions

6.1.1 Test setup

The test setup for running qperf server workload is shown in Figure 6-1. The qperf clients run on a separate physical server connected to SUT using a single 25 GbE NIC port. Both client and server processes run on Intel Xeon Gold Processor 6138T-based servers. Up to 16 containers, each running qperf server, are instantiated and connected to qperf clients. There is one qperf client instance for each qperf server and one flow between client and server. Each container pod is assigned one VF instance from the same 25Gbe NIC port. The maximum theoretical system throughput is thus 25Gbps bidirectional. The tests are run with unidirectional traffic where the client is sending and the server is receiving for a maximum of 25Gbps network throughput. A container running stress-ng is used to simulate a noisy neighbor scenario.

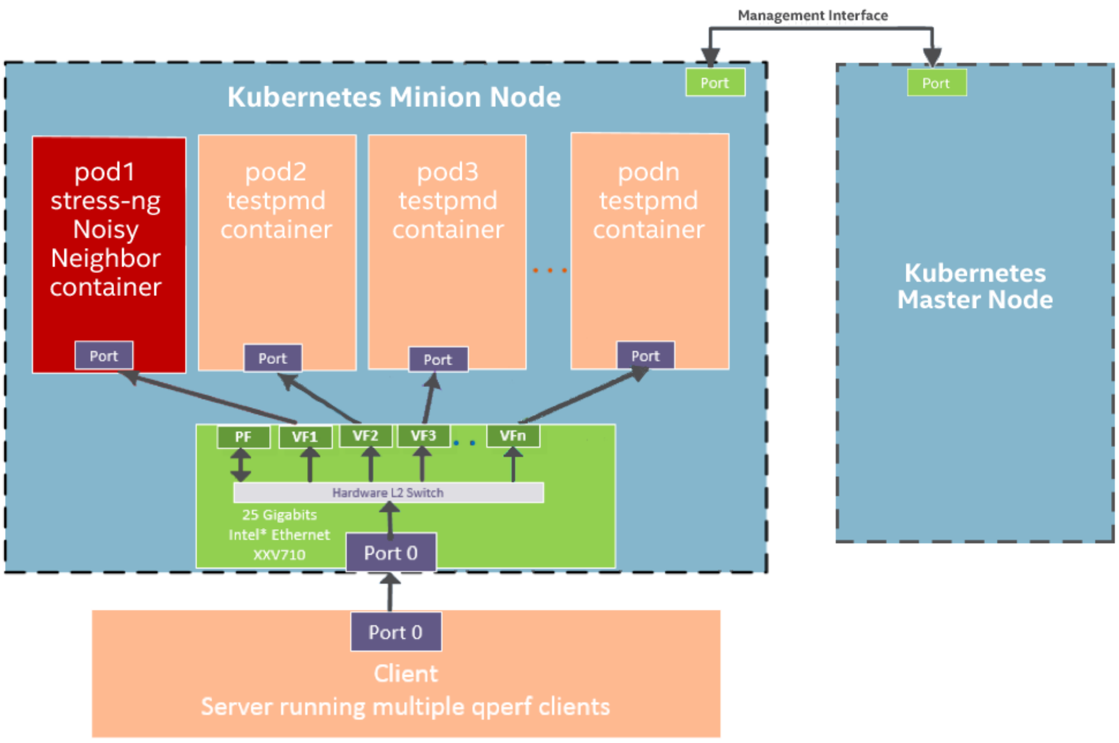

Figure 6-1 High-level overview of kernel driver performance setup with SR-IOV VF using qperf.

6.1.2 Traffic profiles

The traffic profile used for qperf tests are as follows:

- Packet sizes (bytes): 64, 128, 256, 512, 1024 and 1472
- L3 protocol: IPv4
- L4: UDP & TCP
- 1 flow per container in one direction where client is sending the data to the qperf server

6.2 Test results

The performance test results in this section show the network throughput and packet latency for 16 containers running qperf server with and without noisy neighbor container present. The qperf containers are deployed using Kubernetes* and qperf application is run with and without CMK. When qperf is run using CMK, CMK isolates and assigns two hyper threaded sibling cores to a qperf server instance inside a container from its dataplane core pool.

Dataplane cores are exclusive and only one workload can acquire a pair of hyper threaded cores. When qperf is run without CMK, it is not pinned to any specific cores and thus is free to use any available cores in the system. Tests are run for both TCP and UDP traffic types. Each test iteration is run for a duration of five minutes.

Without CMK, Kubernetes may place the noisy neighbor container on the same physical system where the container under test is running. In this scenario, the noisy application may share the cores assigned to the application under test, thus impacting the target application's performance. Performance impact will vary depending on the load placed by the noisy container on the application assigned cores. In these tests, a load of 50% is generated on all available cores using stress-ng application.

Test data is collected and benchmarked for the following test cases:

1. Without CMK and no noisy neighbor
2. With CMK and no noisy neighbor
3. Without CMK in presence of noisy neighbor
4. With CMK in presence of noisy neighbor

The results show a detrimental impact of having a noisy neighbor container when no CMK functionality is available compared when CPU core isolation and CPU core pinning are available. This demonstrates how this technology alleviates the impact of noisy neighbors on application performance.

6.2.1 Qperf container TCP throughput performance with and without CMK

The test results in this section show the system performance for TCP traffic for a 16-container test case. There is one connection per container which means there are a total 16 TCP connections altogether.

The test results are described below and also shown in Figure 6-2 & Figure 6-3:

- With SR-IOV enabled for the qperf container, more than 23Gbits/sec throughput is achieved for both CMK and non-CMK test cases as reported by qperf clients. Note: The throughput reported by qperf clients does not account for TCP header (32 bytes), IP header (20 bytes) and Ethernet header (14 bytes) for each packet, thus reducing the effective line rate.
- When running qperf without CMK, the presence of a noisy neighbor container caused network throughput to degrade by more than 70% for 64 and 128-byte size packets and ~20% lower for packet sizes greater than 512 bytes. The latency increased more than 70 times for most packet sizes.

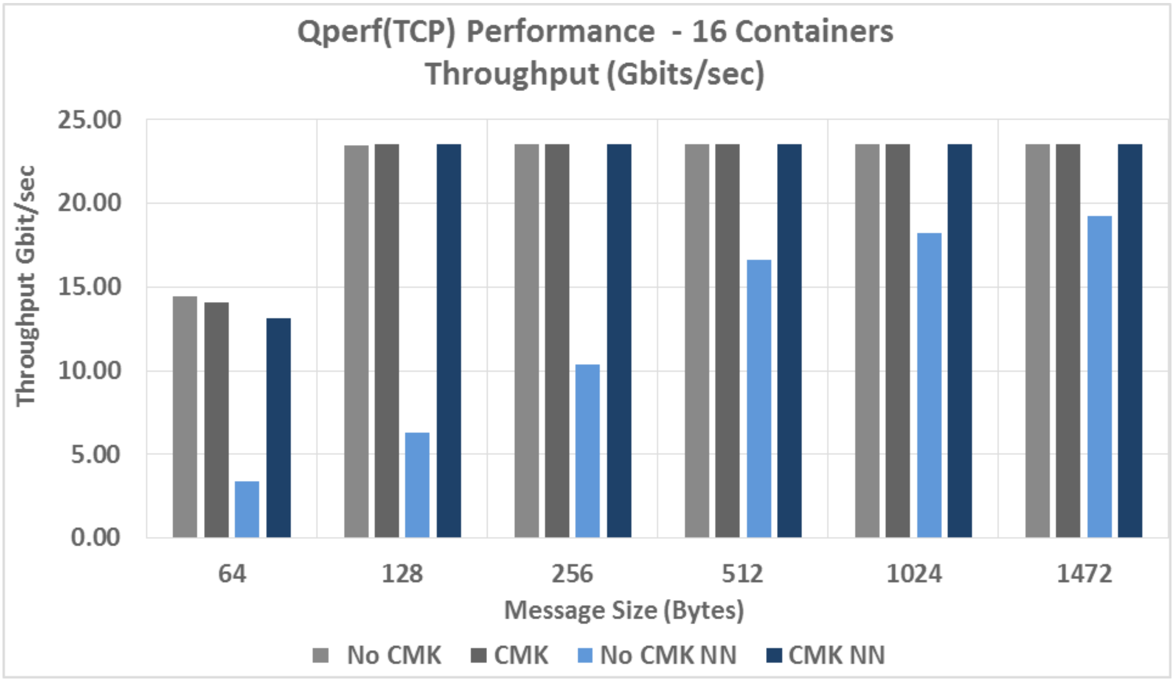

Figure 6-2 qperf TCP throughput comparison with and without CMK and noisy neighbor for 16 containers.

- When running the qperf server using CMK, the performance is not impacted by having a noisy neighbor container running in the system, as the cores are now isolated and assigned to the qperf server and are not available to other containers.
- Detailed results for all container test cases for qperf TCP are presented in Appendices B.3 & B.4

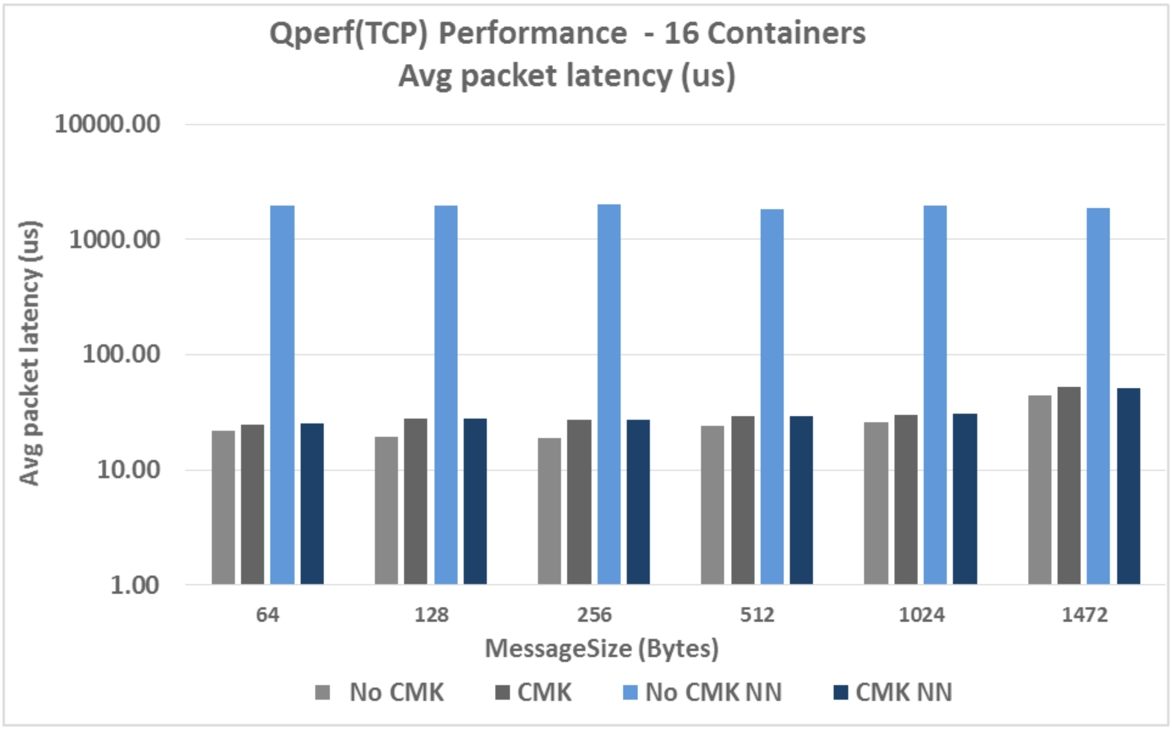

Figure 6-3 qperf TCP latency comparison with and without CMK and noisy neighbor for 16 containers.

6.2.2 Qperf container UDP throughput performance measured with and without CMK

The test results in this section show the system performance for UDP traffic for the 16-container test case. There is one flow per container, which means there are a total of 16 UDP flows altogether.

The test results are described below and also shown in Figure 6-4 & Figure 6-5:

- With SR-IOV enabled for the qperf container, more than 20Gbits/sec throughput is achieved for both CMK and non-CMK test cases as reported by qperf clients. Note: The throughput reported by qperf clients does not account for UDP header (20 bytes), IP header (20 bytes) and Ethernet header (14 bytes) for each packet thus reducing the effective line rate of 25Gbits/sec.
- When running qperf without CMK, the presence of a noisy neighbor container caused network throughput to drop more than 50% for 64-byte packet size and more than 70% for all other packet sizes and latency increased more than 70 times for most packet sizes.
- When running the qperf server using CMK, the performance is not significantly impacted by having a noisy neighbor container running in the system. For certain packet sizes and container cases, non-CMK tests seems to perform better than CMK test case. This is due to the current limitation of CMK where only two hyper threaded sibling cores can be assigned to the container application. When not using CMK, the application is free to use any available cores. This limitation is expected to be addressed in future releases of CMK.

- UDP performance for 64-byte packet sizes is lower compared to TCP. This is because TCP/IP improves network efficiency by reducing the number of packets that need to be sent over the network by combining a number of small outgoing messages and sending them all at once (Nagle's algorithm) thus reducing the packet headers overhead on the wire as well server processing overhead.
- Detailed results for all container cases for qperf UDP tests are available in Appendices B.5 & B.6.

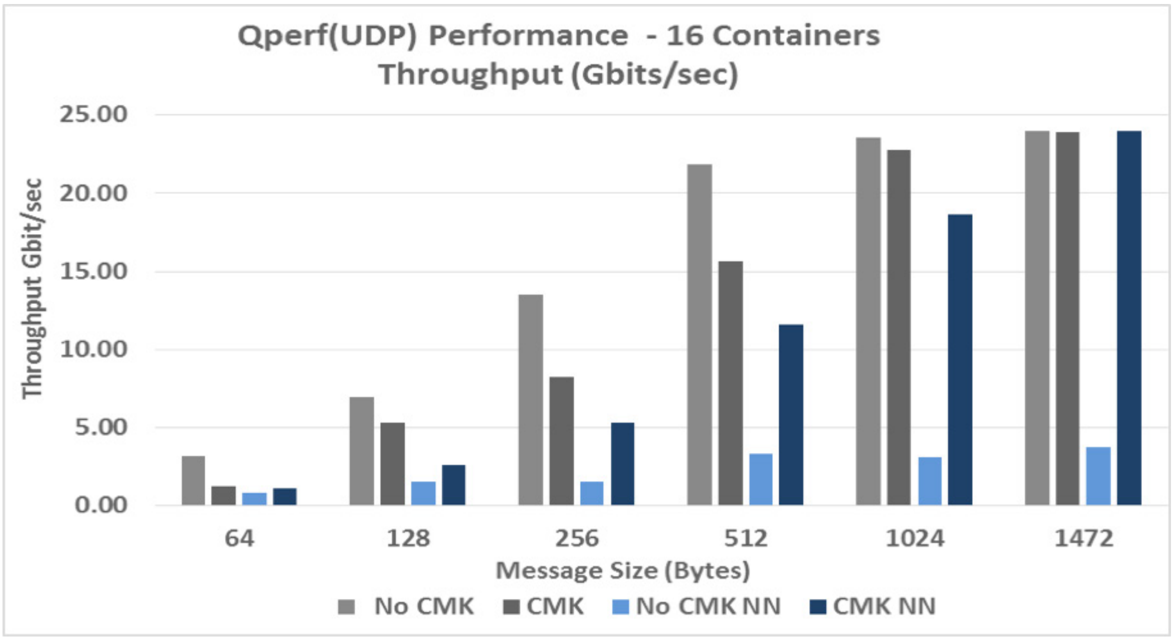

Figure 6-4 qperf UDP throughput comparison with and without CMK and noisy neighbor for 16 containers.

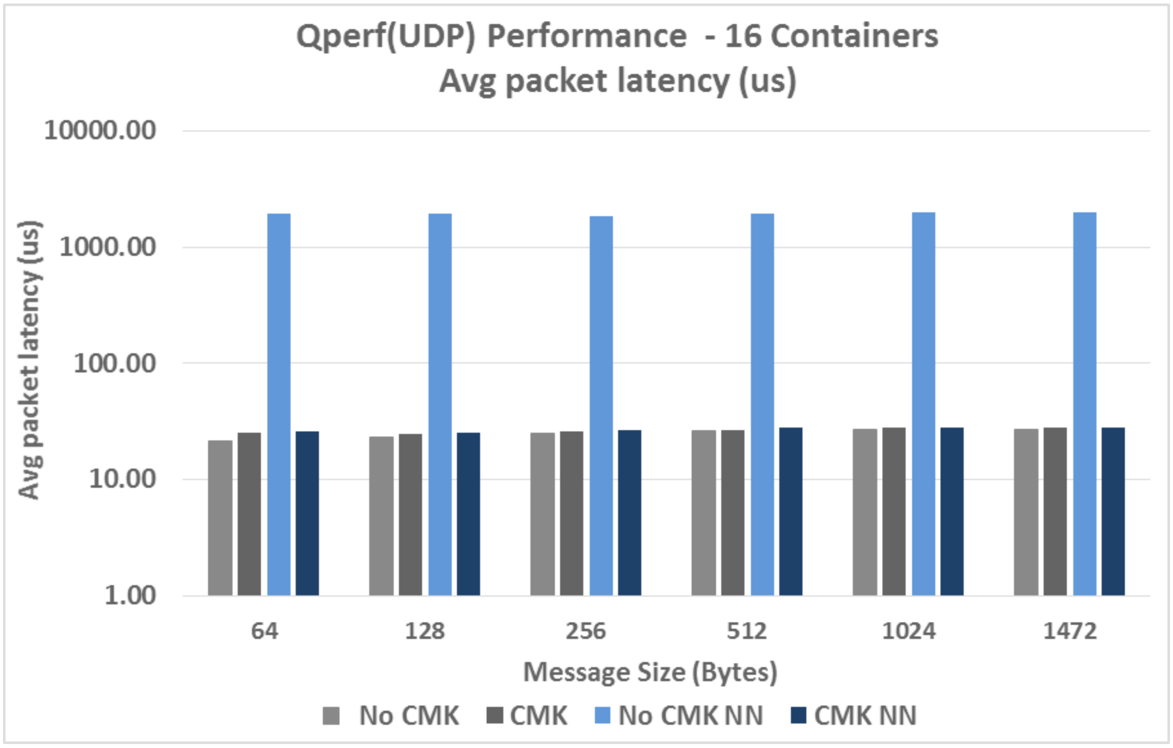

Figure 6-5 qperf UDP latency comparison with and without CMK and noisy neighbor for 16 containers.

7.0 Kubernetes Cluster Deployment

The test setup and methodology follows the user guide titled: *Installation and Configuration Guide for Kubernetes and Container Bare Metal Platform*. This document is also part of the Container Experience Kit and provides instructions on how to deploy a Kubernetes cluster including one master node and one minion node. This document can be downloaded from the link found in Appendix D.

Note: The SR-IOV CNI plugin for Kubernetes needs to be installed in the minion node as per the user guide instructions as VFs are used for networking for the containers. All container workloads run on the minion node that is referred to in this document as the system under test (SUT).

After the instructions in the user guide are complete, three container images will be created: one for DPDK testpmd, one for the qperf server and another one for stress-ng.

8.0 Test Execution

In this section, detailed steps are provided for conducting a series of tests to demonstrate the positive impact of huge pages and CPU core pinning and CPU core isolation. The first series of tests use testpmd to demonstrate EPA benefits for the throughput of DPDK-enabled applications.

The second series of tests uses qperf to generate the traffic for throughput and latency tests for non-DPDK applications. In the last series of tests, stress-ng is used to represent a noisy neighbor application in order to show how CPU core pinning and CPU core isolation can provide deterministic application performance for a target application.

8.1 DPDK application container test execution

8.1.1 Running testpmd without CMK

The following are the necessary steps to take in order to run testpmd without CMK.

Deploy DPDK pods and connect to it using a terminal window.

```
# kubectl create –f no-cmk-dpdk-pod<x>.yaml
# kubectl exec no-cmk-dpdk-pod<x> -ti – bash
```

1. Each pod is assigned two VFs, one from each physical port from 2x25Gbe NIC.
2. Use container ID (CID) to get the PCI address of each VF assigned to the container.

```
# kubectl exec dpdk-pod-c1-m1 –ti – bash
# export cid=”$(sed –ne ‘/hostname/p’ /proc/1/task/1/mountinfo | awk –F ‘/’ ‘{print
$6}’)-north0”
# export PCIADDR1=”$(awk –F ‘”’ ‘{print $4}’ /sriov-cni/$cid)”
# export cid=”$(sed –ne ‘/hostname/p’ /proc/1/task/1/mountinfo | awk –F ‘/’ ‘{print
$6}’)-south0”
# export PCIADDR2=”$(awk –F ‘”’ ‘{print $4}’ /sriov-cni/$cid)”
```

3. Run the DPDK testpmd app in each container.

```
# x86 _ 64-native-linuxapp-gcc/app/testpmd –file-prefix=<name>--socket-mem=1024,1024 –l
<core1, core2> -w $PCIADDR1 –w $PCIADDR2 –n 4 -- -I –txqflags=0xf01 –txd=2048 – rxd=2048
# testpmd> start
```

   Note: To run testpmd, at least two logical cores must be assigned to the application. One core for control plane and one for data plane. These cores should be separate cores for each testpmd instance. Two hyper threaded sibling cores are used in the above command.

4. Start RFC2544 test on Ixnetwork with 256 flows for each container running testpmd. Flows are specified by DMAC address matching to the virtual function’s MAC address assigned to the container.

8.1.2 Running testpmd with CMK

The following are the necessary steps to take to run testpmd with CMK.

1. Deploy DPDK pods and connect to it using a terminal window.

```
# kubectl create -f cmk-dpdk-pod<x>.yaml
# kubectl exec cmk-dpdk-pod<x> -ti – bash
```

2. Each pod is assigned two VFs, one from each physical port from 2x25G NIC.
3. Create /etc/kcm/use_cores.sh file with the following content:

```
#!/bin/bash
export CORES=`printenv KCM _ CPUS _ ASSIGNED`
COMMAND=${@//'$CORES'/$CORES}
$COMMAND
```

Note: The above script uses CMK to assign the cores from temporary environment variable 'KCM_CPUS_ASSIGNED' to its local variable CORES. Then, this variable substitutes $CORES phrase in command provided below as argument to this script and executes it with the correct cores selected.

4. Make this an executable script:

```
# chmod +x /etc/kcm/use _ cores.sh
```

5. Use container ID (CID) to get the PCI address of each VF assigned to the container.

```
# kubectl exec dpdk-pod-c1-m1 -ti – bash
# export cid="$(sed -ne '/hostname/p' /proc/1/task/1/mountinfo | awk -F '/' '{print
$6}')-north0"
# export PCIADDR1="$(awk -F '"' '{print $4}' /sriov-cni/$cid)"
# export cid="$(sed -ne '/hostname/p' /proc/1/task/1/mountinfo | awk -F '/' '{print
$6}')-south0"
# export PCIADDR2="$(awk -F '"' '{print $4}' /sriov-cni/$cid)"
```

6. Start testpmd using use_cores.sh script:

```
# /opt/bin/kcm isolate --conf-dir=/etc/kcm --pool=dataplane /etc/kcm/use _ cores.sh 'testpmd
--file-prefix=<name> --socket-mem=1024,1024 -l \$CORES – -w $PCIADDR1 -w $PCIADDR2 -n 4 -- -i
--txqflags=0xf01 --txd=2048 --rxd=2048'
# testpmd> start
```

7. Start RFC2544 test on Ixnetwork with 256 flows for each container running testpmd. Flows are specified by DMAC address matching to the VF's MAC address assigned to the container.

8.2 Non-DPDK application container test execution

When i40evf kernel mode driver is loaded in the container for a VF, the driver doesn't set the MAC address filter correctly. This issue is expected to be addressed in a future driver release. The following workaround is needed with the current version of driver before VF can start to receive traffic.

1. Find MAC addresses assigned to the VF in dmesg:

```
#dmesg | grep "MAC Address:"
[   54.297588] i40evf 0000:18:02.0: MAC address: 52:54:00:10:6d:64
```

2. Set VF MAC to the MAC address seen above:

```
#ip link set dev virtual-1 vf n <mac>
```

8.2.1 Running qperf tests without CMK

The following are the necessary steps to take to run qperf without CMK.

1. Deploy qperf pods and connect to it using a terminal window.

```
# kubectl create -f no-cmk-qperf-pod<x>.yaml
# kubectl exec no-cmk-qperf-pod<x> -ti – bash
```

2. Each container is assigned 1 VF from the same physical port of the 2x25Gbe NIC.
3. Turning off adaptive interrupts for VF driver and adjust ring size.

```
# ethtool -G south0 rx 256
# ethtool -G south0 tx 256
# ethtool -C south0 adaptive-rx off
# ethtool -C south0 adaptive-tx off
```

4. Run the qperf server in each container.

```
#  qperf
```

5. Start qperf TCP tests on qperf client system one client per qperf server instantiated.

```
#  qperf <server _ ip> tcp _ bw tcp _ lat ud _ lat ud _ bw
```

8.2.2 Running qperf tests with CMK

For kernel network application performance tests using SR-IOV VF driver, CMK assigns an isolated core to the container application. However, the kernel VF driver runs inside the host and its interrupt affinity is not managed by CMK. As a result, the VF driver uses cores that may be different than the ones assigned to container application. Each VF driver has four queues and interrupts for these queues, by default, use cores 0-3. CMK does not isolate these cores for VF driver. A workaround is to manually add these cores to the list of isolated cores in the file /boot/grub/grub.cfg after deploying cluster on the minion node.

1. To implement the workaround, update /boot/grub/grub.cfg file to add VF driver interrupt cores to the list of isolated cores as below.

```
GRUB _ CMDLINE _ LINUX="$GRUB _ CMDLINE _ LINUX intel _ iommu=on" # added by onp sriov role
GRUB _ CMDLINE _ LINUX="$GRUB _ CMDLINE _ LINUX
isolcpus=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60" # added by onp isolcpus role
GRUB _ CMDLINE _ LINUX="$GRUB _ CMDLINE _ LINUX default _ hugepagesz=1G hugepagesz=1G hugepages=16"
# added by onp hugepages role
```

2. Save /boot/grub/grub.cfg and run grub-update and reboot the system.
3. Deploy qperf pods and connect to it using a terminal window.

```
# kubectl create -f cmk-qperf-pod<x>.yaml
# kubectl exec cmk-qperf-pod<x> -ti – bash
```

4. Each container is assigned one VF from the same physical port of the 2x25Gbe NIC.
5. Turn off adaptive interrupts for VF driver and adjust ring size.

```
# ethtool -G south0 rx 256
# ethtool -G south0 tx 256
# ethtool -C south0 adaptive-rx off
# ethtool -C south0 adaptive-tx off
```

6. Run the qperf server in each container using use_cores.sh script:

```
# /opt/bin/kcm isolate --conf-dir=/etc/kcm --pool=dataplane qperf
```

7. Start qperf TCP tests on qperf client system one client per qperf server instantiated.

```
#  qperf <server _ ip> tcp _ bw tcp _ lat ud _ lat ud _ bw
```

9.0 Summary

The results of performance benchmarks detailed in this report demonstrate the improved data plane and application performance that comes from utilizing EPA (CPU pinning and isolation, SR-IOV and huge pages) with DPDK on servers based on Intel Xeon Gold Processor 6138T.

As shown in the executive summary, using SR-IOV for networking, huge pages, core pinning and DPDK allowed for improved data throughput in a containerized application (testpmd).

Application performance predictability was also achieved utilizing core pinning and isolation, which negated the impact of a noisy neighbor application (stress-ng). This performance was significant in non-DPDK applications; but the performance when DPDK applications were used was close to the performance delivered when the applications are running in the host.

Network performance and application performance predictability are critical performance metrics for containerized applications. This benchmark performance report gives developers the tools to maximize both metrics for their applications.

To access more information that is part of the Intel Container Experience Kits (user guides, application notes, feature briefs and other collateral) go to: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

Appendix A: Configuration files

A.1 Configuration file to create a pod without CMK

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/tolerations:
  name: <pod-name>
spec:
  containers:
  - name: <pod-name>
    image: <containerImage>
    volumeMounts:
    - mountPath: /sriov-cni
      name: cni-volume
    - mountPath: /mnt/huge
      name: hugepage-volume
    command: ["/bin/sleep","infinity"]
 ports:
    - containerPort: 81
      protocol: TCP
    securityContext:
        privileged: true
        runAsUser: 0
  volumes:
  - name: cni-volume
    hostPath:
      path: /var/lib/cni/sriov/
  - name: hugepage-volume
    hostPath:
      path: /mnt/huge
  securityContext:
    runAsUser: 0
  restartPolicy: Never
  nodeSelector: kubernetes.io/<hostname>
```

A.2 Configuration file to create a pod with CMK

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: <app-name>
  annotations:
    "scheduler.alpha.kubernetes.io/tolerations": '[{"key":"cmk", "value":"true"}]'
  name: <pod-name>
spec:
  containers:
 - command:
  - "sleep"
  - "infinity"
    env:
    - name: CMK_PROC_FS
      value: "/host/proc"
    image: <container_image>
    name: <app-name>
    resources:
      requests:
        pod.alpha.kubernetes.io/opaque-int-resource-cmk: '1'
    volumeMounts:
    - mountPath: "/sriov-cni"
      name: cni-volume
    - mountPath: "/host/proc"
      name: host-proc
      readOnly: true
    - mountPath: "/opt/bin"
      name: cmk-install-dir
    - mountPath: "/etc/cmk"
      name: cmk-conf-dir
    - mountPath: /dev/hugepages
      name: hugepage-volume
  securityContext:
    privileged: true
    runAsUser: 0
  volumes:
  - hostPath:
      path: "/var/lib/cni/sriov/"
    name: cni-volume
  - hostPath:
      path: "/opt/bin"
    name: cmk-install-dir
  - hostPath:
      path: "/proc"
    name: host-proc
  - hostPath:
      path: "/etc/cmk"
    name: cmk-conf-dir
  - hostPath:
      path: /dev/hugepages
    name: hugepage-volume
```

A.3 Configuration file to create a stress-ng pod

```
kind: Pod
apiVersion: v1
metadata:
name: stress-ng
labels:
pod-1: true
spec:
containers:
- name: stress-ng
image: lorel/docker-stress-ng:latest
imagePullPolicy: IfNotPresent
args:
- "--cpu 0"
- "-p 50"
- "-t 800m"
restartPolicy: Never
nodeSelector: kubernetes.io/<hostname>
```

A.4 Multus configuration file (pre-requisite for SR-IOV)

```
# cat /etc/cni/net.d/10-multus.conf
{
  "name": "multus-demo-network",
  "type": "multus",
  "delegates": [
    {
      "type": "sriov",
      "if0": "enp134s0f0",
      "if0name": "south0",
      "dpdk": {
      "kernel _ driver":"i40evf",
      "dpdk _ driver":"vfio-pci",
      "dpdk _ tool":"/opt/dpdk/install/share/dpdk/usertools/dpdk-devbind.py"
     }
    },
    {
      "type": "sriov",
      "if0": "enp134s0f1",
      "if0name": "north0",
      "dpdk": {
      "kernel _ driver":"i40evf",
      "dpdk _ driver":"vfio-pci",
      "dpdk _ tool":"/opt/dpdk/install/share/dpdk/usertools/dpdk-devbind.py"
     }
    },
    {
      "name": "cbr0",
      "type": "flannel",
      "masterplugin": true,
      "delegate": {
        "isDefaultGateway": true
      }
    }
  ]
}
```

A.5 ops_config.yml configuration file changes

```
# Num of hugepages:
ovs_num_hugepages: 32
# select one of the network types:
ovs_type: multus
# Enable sriov: true or false
use_sriov: true
num_virtual_funcions:20

# CMK - below 3 configurations required only when using CMK
Enable cmk: true
num_dp_cores = 17
num_cp_cores = 1

use_udev: false
use_cmk: false
cmk_img: "quay.io/charliekang/cmk:v1.0.1"
num_dp_cores: 16
num_cp_cores: 1

use_udev: true
proxy_env:
http_proxy: <http proxy configurations>
https_proxy: <https proxy configurations>
#  socks_proxy: http://proxy.example.com:1080
no_proxy: "localhost,{{ inventory_hostname }}"
```

Appendix B: Test results for all container cases

B.1 DPDK application results: Host versus container

i. Network throughput

Framesize	DPDK testpmd Host v/s Container - Throughput (Gbits/sec)		
	2P_1C_2T_Host_PF	2P_1C_2T_Host_VF	2P_1C_2T_Container_VF
64	24.64842821	24.32977668	24.05972522
128	40.62500346	40.1951145	39.44886463
256	46.17185735	44.58343785	43.66716512
512	48.82810354	48.80729427	47.87857407
1024	48.55467972	48.54364922	48.24191225
1518	48.39811681	48.40570593	48.24191311

ii. Frames per second

Framesize	DPDK testpmd Host v/s Container - Throughput (Packets/sec)		
	2P_1C_2T_Host_PF	2P_1C_2T_Host_VF	2P_1C_2T_Container_VF
64	36679208.64	36205024.82	35803162.52
128	34311658.33	33948576.44	33318297.83
256	20911167.28	20191774.39	19776795.8
512	11472768.69	11467879.29	11249664.96
1024	5813539.239	5812218.537	5776091.026
1518	3933527.049	3934143.85	3920831.69

iii. Packet latency

Framesize	DPDK testpmd Host v/s Container - Avg Latency (us)		
	2P_1C_2T_Host_PF	2P_1C_2T_Host_VF	2P_1C_2T_Container_VF
64	7.58	7.192	7.242
128	8.9045	9.189	8.143
256	10.421	10.7235	9.0815
512	12.111	14.4315	12.9585
1024	16.9575	16.984	17.515
1518	23.8845	23.9235	24.29

B.2 DPDK application test results without CMK

i. Network throughput

Framesize	Testpmd - No CMK Agg Throughput (Gbits/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	23.9864789	2.956358	23.2811492	4.497602968	20.82020865	4.671146	20.46867	5.219799	18.8597816	5.351472821
128	39.8000917	3.170548	40.5075808	4.600321255	37.34359043	5.1826	35.93734	5.333913	35.0096055	5.351532942
256	44.0183599	4.831747	47.5389932	4.917708682	47.89048544	5.316495	47.89042	5.351523	44.5676855	14.84368586
512	47.1873714	5.248079	48.5934734	5.331246433	48.59354134	5.351303	48.59352	14.14057	48.5935432	14.14055012
1024	48.2420463	5.291707	48.2421166	5.351491758	48.24198052	5.859356	48.24198	33.12474	48.2419861	30.66392079
1518	48.2419144	5.350782	48.2419811	5.351478493	48.2419798	8.496041	48.24196	35.58582	48.2419794	44.02327597

ii. Frames per second

Framesize	Testpmd - No CMK Agg Throughput (Packets/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	35694165.1	4399343	34644567.2	6692861.56	30982453.35	6951110	30459328	7767557	28065151.3	7963501.221
128	33614942.4	2677828	34212483.8	3885406.465	31540194.62	4377196	30352483	4504994	29568923.6	4519875.796
256	19935851.4	2188291	21530341.1	2227223.135	21689531.45	2407833	21689501	2423697	20184640.2	6722683.815
512	11087258.3	1233101	11417639.4	1252642.489	11417655.39	1257355	11417651	3322503	11417655.8	3322497.678
1024	5776107.07	633585.7	5776115.49	640743.745	5776099.2	701551.3	5776099	3966085	5776099.87	3671446.455
1518	3920831.8	434881.5	3920837.21	434938.109	3920837.11	690510.5	3920836	2892215	3920837.08	3577964.562

iii. Packet latency

Framesize	Tesppmd - NO CMK Avg Latency (us)									
	1 Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	I6 Container w/ NN
64	7.4305	449.943	17.7	281.7515	18.11325	502.4273	14.22288	633.7204	12.213233	707.409813
128	8.643	705.71	17.421	507.52625	23.24475	474.3229	14.89829	888.4254	35.367533	345.893406
256	9.927	240.672	11.48	701.109	31.347	648.8365	74.86669	731.5077	32.558267	338.452438
512	13.774	144.039	14.712	131.732	12.15375	384.4215	13.65444	297.7683	17.381375	226.432344
1024	17.1575	1249.679	13.345	854.34825	12.41225	146.629	14.93881	229.7188	21.373313	725.870875
1518	24.3795	1090.054	14.37725	222.583	14.676125	495.6538	19.74025	530.6776	29.4615	790.713406

B.3 DPDK test results with CMK

i. Network throughput

Framesize	Testpmd - CMK - Agg Throughput (Gbits/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	24.0597252	24.05973	23.2811448	23.28100723	20.82019483	20.82022	20.46864	20.46865	20.1171019	20.11709444
128	39.4488646	39.44886	40.5076415	40.50763991	37.34356512	37.69514	36.64047	36.64047	35.9373399	36.64045408
256	43.6671651	43.66717	47.5388593	47.53879068	47.89041649	47.89042	47.8904	47.89045	47.8904382	47.53887255
512	47.8785741	46.83568	48.5936118	48.59354296	48.59357334	48.5935	48.59356	48.59352	48.5935627	48.59353304
1024	48.2419122	48.24192	48.2419833	48.24211707	48.24194863	48.24195	48.24199	48.24203	48.2419266	48.24198114
1518	48.2419131	48.24192	48.2420503	48.24191423	48.2419867	48.24198	48.24202	48.24195	48.2419764	48.24200544

ii. Frames per second

Framesize	Testpmd - CMK Agg Throughput Packets/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	35803162.5	35803163	34644560.8	34644355.99	30982432.78	30982467	30459290	30459299	29936163.5	29936152.44
128	33318297.8	33318298	34212535	34212533.71	31540173.24	31837112	30946343	30946345	30352483	30946329.46
256	19776795.8	19776796	21530280.5	21530249.4	21689500.22	21689503	21689495	21689516	21689510.1	21530286.48
512	11249665	11004623	11417671.9	11417655.77	11417662.91	11417645	11417661	11417651	11417660.4	11417653.44
1024	5776091.03	5776091	5776099.53	5776115.55	5776095.382	5776095	5776100	5776105	5776092.75	5776099.274
1518	3920831.69	3920832	3920842.84	3920831.781	3920837.671	3920837	3920840	3920834	3920836.83	3920839.194

iii. Packet latency

Framesize	Testpmd - CMK - Avg Latency (us)									
	1 Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	I6 Container w/ NN
64	7.242	7.2985	15.84725	14.78275	17.836	16.823	13.56663	12.44631	12.579156	12.754469
128	8.143	8.2545	17.5305	17.41525	23.788125	23.4625	14.88494	14.73481	26.077688	25.511469
256	9.0815	9.5795	11.561	11.989	30.598375	29.95588	40.06263	39.96363	20.082781	21.660906
512	12.9585	13.9295	14.8605	15.0185	12.11325	12.12513	13.62781	13.62588	16.962094	17.400375
1024	17.515	17.492	13.35525	13.50825	12.52625	12.688	14.99931	15.0505	21.497781	21.803938
1518	24.29	24.309	14.42975	14.45075	14.53875	14.679	19.62313	19.77031	29.346406	29.550031

B.4 Non-DPDK (TCP) test results without CMK

i. Network throughput as reported by qperf client

Framesize	Qperf(TCP) - No CMK Agg Throughput (Gbits/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	1.01	0.303	2.05	0.689	3.80	1.002	7.46	2.786	14.44	3.402
128	1.95	0.612	3.71	1.174	7.01	2.834	13.84	3.391	23.49	6.322
256	3.11	1.176	6.41	2.288	11.54	3.400	22.68	3.970	23.54	10.362
512	6.20	1.992	12.33	2.936	19.96	2.800	23.52	7.438	23.55	16.617
1024	10.01	1.816	18.84	3.416	22.51	6.368	23.54	11.272	23.55	18.256
1472	12.64	2.032	19.66	3.896	23.53	8.392	23.54	11.416	23.56	19.264

ii. Packet latency as reported by qperf client

Framesize	Qperf(TCP) - NO CMK Avg Latency (us)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	I Container w/o NN	I Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	I6 Container w/ NN
64	12.32	2000	14.88	1985	16.64	1998	16.29	1988	22.09	1987
128	12.34	2000	15.88	1740	16.08	1918	18.20	2000	19.24	1993
256	14.89	2000	15.70	1910	14.22	1998	18.93	1998	19.08	1994
512	22.40	2000	20.75	1975	20.26	2000	21.67	2000	24.08	1821
1024	25.38	2000	25.50	2000	25.68	2000	25.74	2000	26.20	1949
1472	38.96	2000	38.94	2000	40.94	2000	41.44	2000	44.80	1883

B.5 Non-DPDK (TCP) test results with CMK

i. Network throughput as reported by qperf client

Framesize	Qperf(TCP) - CMK Agg Throughput (Gbits/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	1.065	1.064	2.029	2.037	3.867	3.951	7.473	7.556	14.116	13.112
128	1.976	1.996	3.836	3.834	7.369	7.337	13.798	13.554	23.524	23.510
256	3.591	3.629	6.820	6.767	12.902	13.021	21.951	21.525	23.544	23.532
512	6.205	6.291	10.835	11.825	22.156	19.713	23.543	23.532	23.550	23.550
1024	10.178	10.421	18.560	13.493	23.533	23.534	23.538	23.518	23.561	23.537
1472	12.928	12.939	23.509	19.040	23.538	23.532	23.545	23.538	23.546	23.538

ii. Packet latency as reported by qperf client

Framesize	Qperf(TCP) - CMK Avg Latency (us)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	I Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	22.432	19.007	23.677	18.915	18.803	20.200	21.853	22.599	25.007	25.652
128	16.674	20.071	20.980	20.341	21.181	20.133	22.062	23.641	27.719	27.916
256	16.197	20.076	20.560	18.077	20.925	21.051	24.776	24.886	26.995	27.185
512	24.552	25.141	22.846	24.020	24.407	24.440	26.271	26.164	29.318	29.230
1024	26.138	26.217	25.704	25.978	26.235	26.775	27.563	27.673	30.386	30.593
1518	41.730	43.808	42.317	44.781	43.559	46.019	47.521	49.794	53.191	51.097

B.6 Non-DPDK (UDP) test results without CMK

i. Network throughput as reported by qperf client

Framesize	Qperf(UDP) - No CMK Agg Throughput (Gbits/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	0.311	0.005	0.309	0.096	0.860	0.086	1.413	0.426	3.178	0.862
128	0.624	0.007	0.598	0.181	2.200	0.586	3.334	0.799	6.963	1.512
256	1.199	0.006	1.189	0.190	4.387	0.487	8.402	0.960	13.516	1.547
512	2.352	0.014	4.206	0.365	6.657	0.974	14.975	1.792	21.853	3.318
1024	4.779	0.045	4.900	1.073	16.703	1.787	22.284	6.958	23.520	3.119
1472	6.979	0.110	11.351	0.034	18.110	0.174	23.709	0.709	23.996	3.724

ii. Packet latency as reported by qperf client

Framesize	Qperf(UDP) - No CMK Avg Latency (us)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	I Container w/o NN	I Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	I6 Container w/ NN
64	12.08	1940	19.91	2000	14.58	1960	22.32	1898	21.47	1961
128	12.05	1970	12.15	2000	21.46	1903	17.77	1960	23.17	1934
256	24.96	1990	26.94	1975	23.61	1993	24.58	1989	24.77	1845
512	25.11	1980	26.04	2000	26.13	1993	25.21	1983	26.02	1954
1024	26.97	2000	26.95	2000	26.83	1988	26.87	2000	27.07	1988
1472	27.10	1990	27.10	1955	27.13	1813	27.22	1921	27.28	2001

B.7 Non-DPDK (UDP) test results with CMK

i. Network throughput as reported by qperf client

Framesize	Qperf(UDP) - CMK Agg Throughput (Gbits/sec)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	0.289	0.303	0.507	0.557	0.981	1.041	0.902	0.984	1.271	1.131
128	0.563	0.575	1.179	0.792	1.812	1.318	2.132	1.964	5.310	2.618
256	1.136	1.108	1.516	1.403	3.434	2.415	5.945	4.814	8.233	5.347
512	2.334	2.469	4.302	2.814	6.878	6.729	10.805	9.080	15.634	11.578
1024	4.632	4.968	7.320	9.136	12.652	9.732	16.326	15.225	22.788	18.616
1472	6.952	7.061	12.729	7.085	18.579	19.169	20.175	20.521	23.926	23.948

ii. Packet latency results as reported by qperf client

Framesize	Qperf(UDP) - CMK Avg Latency (us)									
	I Container		2 Containers		4 Containers		8 Containers		16 Containers	
	1 Container w/o NN	1 Container w/ NN	2 Container w/o NN	2 Container w/ NN	4 Container w/o NN	4 Container w/ NN	8 Container w/o NN	8 Container w/ NN	16 Container w/o NN	16 Container w/ NN
64	12.521	24.977	19.728	26.961	13.639	26.972	21.079	24.215	25.497	26.225
128	19.131	24.968	17.009	19.836	18.051	21.001	23.966	25.165	24.974	25.327
256	25.006	26.976	24.996	26.973	25.489	24.361	25.438	25.697	26.064	26.958
512	26.984	26.972	26.969	27.002	26.532	26.545	25.452	26.208	26.947	27.716
1024	26.967	26.964	26.974	26.977	26.983	27.080	27.151	27.592	28.314	28.196
1472	27.100	27.113	27.112	27.103	27.108	27.125	27.228	27.355	27.886	28.060

Appendix C: Abbreviations

Abbreviation	Description
CMK	CPU Manager for Kubernetes
COE	Container orchestration engine
CPU	Central Processing Unit
DPDK	Data Plane Development Kit
DUT	Device Under Test
EPA	Enhanced Platform Awareness
NFD	Node Feature Discovery
NFV	Network Functions Virtualization
PF	Physical Function
PMD	DPDK Poll Mode Driver
p-state	CPU performance state
SDI	Software Defined Infrastructure
SDN	Software Defined Networking
SKU	Stock Keeping Unit
SLA	Service Level Agreement
SR-IOV	single root input/output virtualization
SUT	System Under Test
VF	Virtual Function
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function

Appendix D: Reference Documents

#	Title	Reference
1	Kubernetes Overview	https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
2	Kubernetes API Server	https://kubernetes.io/docs/admin/kube-apiserver/
3	Kubernetes Pod Overview	https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/
4	Multus CNI Plugin	https://github.com/Intel-Corp/multus-cni
5	SR-IOV	https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf
6	SR-IOV CNI Plugin	https://github.com/Intel-Corp/sriov-cni
7	Enhanced Platform Awareness	https://builders.intel.com/docs/networkbuilders/EPA_Enablement_Guide_V2.pdf
8	Node Feature Discovery	https://github.com/Intel-Corp/node-feature-discovery
9	CPU Manager for Kubernetes	https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes
10	Use cases for Kubernetes	https://thenewstack.io/dls/ebooks/TheNewStack_UseCasesForKubernetes.pdf
11	Kubernetes Components	https://kubernetes.io/docs/concepts/overview/components/
12	Containers vs Virtual Machines	https://docs.docker.com/get-started/ - containers-vs-virtual-machines
13	Intel Ethernet Converged Network Adapter X710-DA2	http://ark.intel.com/products/83964/Intel-Ethernet-Converged-Network-Adapter-X710-DA2
14	Intel Ethernet Network Adapter XXV710-DA2	http://ark.intel.com/products/95260/Intel-Ethernet-Network-Adapter-XXV710-DA2
15	Intel Server Board S2600WT2	http://ark.intel.com/products/82155/Intel-Server-Board-S2600WT2
17	Intel Xeon GOLD 6138T Processor	http://ark.intel.com/products/123542/Intel-Xeon-Gold-6138T-Processor-27_5M-Cache-2_00-GHz
18	RFC 2544 Benchmarking Methodology	https://tools.ietf.org/html/rfc2544
19	Installation and Configuration Guide for Kubernetes and Container Bare Metal Platform	https://networkbuilders.intel.com/network-technologies/container-experience-kits

Legal Information