WHITE PAPER

Deterministic Network Functions Virtualization with Intel® Resource Director Technology

Authors

Joseph Gasparakis
Sunku Ranganath
Edwin Verplanke
Priya Autee

Contributors

Adrian Hoban
Mark D. Gray
Ciara Loftus
Yunhong Jiang
Michelle Smekal

Table of Contents

1 Introduction

1.1 Audience and Purpose

Intel delivers engineering guidance and ecosystem-enablement support to encourage widespread adoption of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) solutions in telco*, enterprise, and cloud applications.

The primary audience for this white paper are architects and engineers interested in the Intel® Resource Director Technology (Intel® RDT) and how the technology can be used to achieve service assurance, security, predictable latency and improve throughput in their NFV deployments with key software ingredients such as Open vSwitch (OvS) with Data Plane Development Kit (DPDK) and Kernel-Based Virtual Machine (KVM) on a Fedora* platform.

1.2 Abstract

Service assurance, security, and deterministic performance (predictable latency and throughput) are system qualities that are important for communications service providers, cloud service providers and enterprise NFV deployments. This document provides a brief description on how they can be greatly improved by using the Intel® RDT. Full software enablement in open source projects such as the Linux* Kernel, DPDK, OpenStack* are in progress. However, Intel is taking the opportunity to provide a preview of the Intel® RDT technology using the pqos utility in deployments with DPDK, OvS and KVM.

Elements of the Intel® RDT, Cache Allocation Technology (CAT), Cache Monitoring Technology (CMT), Code and Data Prioritization (CDP) and Memory Bandwidth Management (MBM) are introduced. The important concept of VNF profiling using Cache Allocation Technology is detailed showcasing the key performance tuning aspects of the Class of Service (COS) in order to have the Virtual Network Functions (VNFs) function successfully with optimal performance. Based on the deductions from this profiling, two commonly deployed NFV environments are characterized, PHY-VM-PHY and PHY-VM-VM-PHY, under noisy neighbor conditions. The benefits of using CAT for the above two NFV deployments under noisy neighbor situations are demonstrated by preserving throughput, latency, and predictability in performance. The paper is concluded by suggesting the NFV workload profiling methods using CAT to help achieve deterministic performance.

1.3 Motivation

As market adoption of NFV is taking place, it is crucial that during the migration from fixed functions to consolidated virtual functions, both service delivery and service assurance continues to be guaranteed. This paper documents how Intel® RDT can be used to meet these goals. The last level cache is an example of a shared resource that is important for packet processing applications to maintain throughput and latency requirements, while at the same time it can affect the performance of VMs and potentially create issues with service assurance in enterprise and cloud environments. Aggressors or noisy neighbors causing evictions of packet processing code and data while executing in parallel can affect service delivery and service assurance. Both cache monitoring technology and cache allocation technology are used to monitor and control respectively the last level cache occupancy to create a deterministic environment while consolidating a variety of workloads.

2 NFV Related Technology Overview

This section describes some key technologies (both hardware and software) that are crucial for packet processing in an NFV environment.

Network Functions such as firewalls, routers, Intrusion Prevention Systems (IPS), Intrusion Detection Systems (IDS), Deep Packet Inspection (DPI) systems, WAN acceleration and so on, all require some form of packet inspection/movement/processing. Traditionally fixed function physical appliances are designed to meet peak performance, even at a time frames where this is not required. They are complex to scale back the resource requirements. This is one of key advantages of NFV that helps customize the deployment on the general purpose processor based hardware. However, after consolidation on a general purpose processor, it is crucial that both the performance and deterministic requirements of Service Level Agreements (SLAs) are met. Therefore, data plane performance is important and hence Data Plane Development Kit (DPDK) is crucial for achieving high performance.

For service deployment agility and flexibility a soft switch that allows switching of the packets between virtual ports (for VMs) and/or physical ports (physical network interfaces) is an important element of the Commercial off-the-shelf (COTS) server platform. However, for some use cases (mainly telco), it needs to meet the performance of network functions, specifically the small packet performance throughput and latency. For this reason various soft switch developments with enhanced packet processing capabilities are under development and/or available in the open source communities such as OpenvSwitch (OvS), BESS*, Snab, Lagopus*, etc. This paper focuses on the usage of OvS with the DPDK data path.

Lastly, Intel® RDT is covered in some depth. After meeting the performance using DPDK and OvS-DPDK, it is crucial that to meet the latency and deterministic needs as well. Intel® RDT is briefly mentioned to help with these needs and what hardware features are available to both monitor and enforce policies to meet SLAs.

2.1 Data Plane Development Kit (DPDK)

DPDK is an open source project and its webpage is http://www.dpdk.org. It is a set of libraries and drivers (mostly in user space) that allow more efficient packet processing.

Figure 1 shows, in high level, two networking applications and the software stacks involved. On the left, a typical socket-based Linux* networking application is shown. The packets are being received by the physical networking interface, then through the driver and New API (NAPI), with a combination of interrupt (expensive context switching in terms of performance) and polling mode driven interface. Then the packets are bubbled up to the Linux kernel networking stack and finally through to the socket interface. The packets will be copied over to the user space from where the application will receive the data for the specific protocol it opened the socket for. It should be noted that this copy of the packets from kernel into user space is expensive in terms of CPU cycles, and this negative effect is being magnified on higher data speeds and smaller packet sizes.

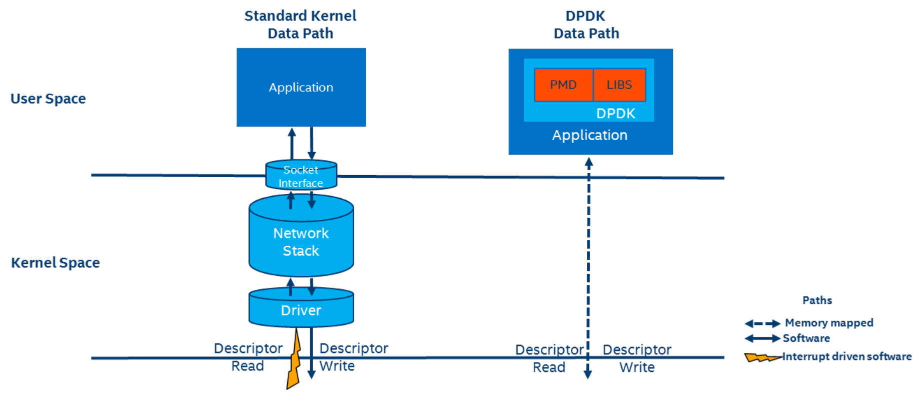

Figure 1. Comparison of the Kernel Data Path vs. User Space Data Path

Transmitting packets is similar, where the application is sending the packet data down into the kernel through the socket (again, performing an expensive buffer copy), where the data will be encapsulated by the right headers on the different layers of the kernel network stack, and through the driver the fully structured packet will be written to the write descriptor of the networking interface. Finally having the hardware push it out on the wire.

On the right side, a typical DPDK based networking application is shown. The memory space of the networking interface is being mapped in the user space through the user space I/O (UIO) facility of the kernel (other options also available such as Virtual Function I/O* (VFIO*). The user space Poll Mode Driver (PMD) is reading the packets from the mapped descriptor (no interrupt) and the application will receive the raw packet. Then using the optimized packet processing libraries, it will modify the packet and write it out to the memory mapped write descriptor, letting the hardware push it out to the physical network.

2.2 Open vSwitch (OvS) and its Caching Mechanisms

OvS is an open source project that is hosted in http://openvswitch.org/.

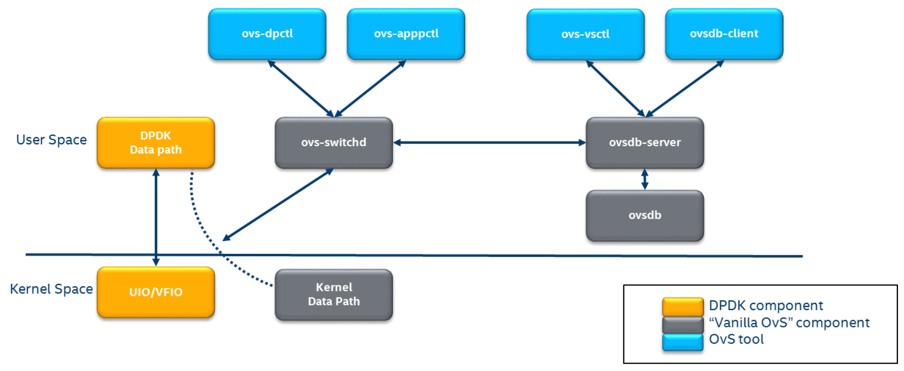

Figure 2. High-Level Architecture of Open vSwitch

The above figure is showing the high-level architecture of OvS with its main software components shown as blocks.

The data path component is the component that is forwarding frames from one port to another. These ports can be physical or virtual. There are two options in terms of data path (shown as a dotted arrow), one is kernel based (grey block called Kernel Data Path) and the other is DPDK based, shown in yellow boxes on the left. For the UIO/VFIO yellow block, refer to section Data Plane Development Kit (DPDK), noting that it is only used to map the address space of the device to user space, and no software involvement is required.

When the data path, either implemented in kernel or `DPDK`, receives a frame from a new flow that has not been seen before, it sends it to `ovs-vswitchd`. The `vSwitch` daemon inspects the frame and programs the data path to forward it to the right port (again, virtual or physical). From that point on the data path will be able to handle the rest of the packets of the flow without having to push them to the daemon using the available flow cache.

The daemon has a backend server, `ovsdb-server` that accesses the database `ovsdb` that holds forwarding (just in terms of persistency, this is not part of the hot-path) and configuration data. The blue components are tools that allow configuration, debugging and programming of the data paths (mainly using `ovsdb` and `OpenFlow` protocols).

2.2.1 Open vSwitch (OvS) Flow Caching

When a packet is received by the OvS data path its cache of flows is consulted to perform associated actions. For example, to modify the headers or forward the packets, if the relevant entry is found. Otherwise, the packet is passed to `ovs-vswtichd` to execute the `OpenFlow` pipeline on the packet and calculate the associated action. Then, it is sent back to fast path for forwarding, and a flow cache entry is installed so similar packets in future can use this rule and avoid expensive flow rule calculations.

This is accomplished using the concept of `Microflow` cache and `Megaflow` caches. Interior details of each of these `OvS` caches is outside the scope of this paper. However, the implementation of these caches using a set of flow tables and its hierarchy is briefly discussed. It is important to understand the `OvS` table hierarchy while considering CPU cache segmentation and allocation for `ovs-vswitchd` process.

OvS Table Hierarchy:

1. Extra Match Cache (EMC):

- This is the first table consulted for a packet to determine its fate of actions.
- The comparisons are done for exact match of parameters.
- The EMC is allocated as a single table per data path thread.
- There are 8192 entries per thread with each entry of size

2. Data Path Classifier:

- If the matching action is not found in the EMC, the next comparison is done in data path classifier.
- Dynamically created based on flow rule setup and entries do not have a fixed structure
- Entries can have wildcard matches.
- Single table per data path thread
- There can be maximum of 65536 entries.
- Lookup cost increases from the EMC with data path classifier

3. `ofproto` Classifier:

- If the matching action is not found in the data path classifier, the `ofproto` classifier is used.
- Up to 255 OpenFlow tables are in pipeline per the OvS bridge.
- Wildcard matches are used.
- Cost of lookup is highest with the `ofproto` classifier.

2.3 Real-Time Kernel and KVM4NFV

The NFV hypervisors provide crucial functionality in the NFV Infrastructure (NFVI). The existing hypervisors, however, are not necessarily designed or targeted to meet the requirements for the NFVI. Open Platform for NFV (OPNFV) is a carrier-grade, integrated, open source platform to accelerate the introduction of new NFV products and services, the details of which are beyond the scope of this paper. The KVM4NFV project, under OPNFV, is a collaborative development project in OPNFV making efforts toward enabling the existing hypervisors for NFV features to provide crucial functionality in the NFV Infrastructure.

In the KVM4NFV project, the focus is on the KVM hypervisor to enhance it for NFV, by looking at the following areas:

1. Minimal Interrupt latency variation for data plane VNFs that includes:
   a. Minimal timing variation for timing correctness of real-time VNFs and
   b. Minimal packet latency variation for data-plane VNFs.
2. Fast live migration (outside of current scope)

2.3.1 Minimal Interrupt Latency

Processing performance and latencies depend on a number of factors, including CPUs (frequency, power management features, etc.), micro-architectural resources, the cache hierarchy and sizes, memory (and hierarchy, such as NUMA) and speed, inter-connects, I/O and I/O NUMA, devices and other factors. There are two separate types of latencies to minimize: 1) minimal timing variation for timing correctness of real-time VNFs – timing correctness for scheduling operations (such as radio scheduling), and 2) minimal packet latency variation for data-plane VNFs – packet delay variation, which applies to packet processing. For a VM, interrupt latency (time between arrival of H/W interrupt and invocation of the interrupt handler in the VM), for example, can be either of the above or both, depending on the type of the device. Interrupt latency with a (virtual) timer can cause timing correctness issues with real-time VNFs even if they only use polling for packet processing.

The assumption is that the VNFs are implemented properly to minimize interrupt latency variation within the VMs, but there are additional causes of latency variation on KVM:

1. Asynchronous (e.g., external interrupts) and synchronous (e.g., instructions) VM exits and handling in KVM (and kernel routines called), which may have loops and spin locks.
2. Interrupt handling in the host Linux and KVM, scheduling and virtual interrupt delivery to VNFs.
3. Potential VM exit in the interrupt service routines in VNFs.
4. Exit to the user-level (e.g., QEMU).

While these items require software development and/or specific hardware features there are also some adjustments that need to be made to system configuration information, like hardware, Basic Input/Output (BIOS), Operating System (OS), etc. Achieving low latency with the KVM4NFV project requires setting up a special test environment. This environment includes the BIOS settings, kernel configuration, kernel parameters and the run-time environment. Not only are special kernel parameters needed but a special run-time environment is also required. The correct configuration is critical for improving the NFV performance/latency. Even working on the same codebase, configurations can cause wildly different performance/latency results. There are many combinations of configurations, from hardware configuration to operating system configuration and application level configuration. Also, there is no one simple configuration that works for every case. To tune a specific scenario, it is important to know the behaviors of different configurations and their impact.

Platform Configuration: Some hardware features can be configured through firmware interface (like BIOS), but others may not be configurable (e.g., SMI on most platforms).

- **Power Management:** Most power management related features save power at the expensive of latency. These features include: Intel® Turbo Boost Technology, Enhanced Intel SpeedStep® Technology, processor C states and P states. Normally, they should be disabled, but depending on the real-time application design and latency requirements, some features can be enabled if the impact on deterministic execution of the workload is small.
- **Hyper-Threading:** The logical cores that share resource with other logical cores can introduce latency, so the recommendation is to disable this feature for real-time use cases.
- **Legacy USB Support/Port 60/64 Emulation:** These features involve some emulation in firmware and can introduce random latency. It is recommended that they are disabled.
- **System Management Interrupt (SMI):** SMI runs outside of the kernel code and can potentially cause latency. There is a global SMI enable/disable bit that can be used to disable all SMI events, but it is not recommended to do so as the BIOS/UEFI is responsible for handling a number of critical tasks that help to avoid serious harm to the processor. Instead there are usually BIOS/UEFI dependent options that can minimize the SMI impact such as disable USB mouse and keyboard, use ACPI, disable the TCO watchdog timer, lockdown GPIOs, and disable periodic SMIs. Configuring these items should only be done after consulting the BIOS/UEFI vendor.

2.3.2 Operating System Configuration

- **CPU isolation:** To achieve deterministic latency, dedicated CPUs should be allocated for real-time application. This can be achieved by isolating CPUs from kernel scheduler. Refer to http://lxr.free-electrons.com/source/Documentation/kernel-parameters.txt#L1608 for more information.
- **Memory allocation:** Memory should be reserved for real-time applications and usually huge pages should be used to reduce page faults/TLB misses.
- **IRQ affinity:** All the non-real-time IRQs should be affinitized to non-real-time CPUs to reduce the impact on real-time CPUs. Some OS distributions contain an `irqbalance` daemon which balances the IRQs among all the cores dynamically. It should be disabled as well.
- **Device assignment for VM:** If a device is used in a VM, then device pass through is desirable. In this case, the IOMMU should be enabled.
- **Tick less:** Frequent clock ticks cause latency. `CONFIG_NOHZ_FULL` should be enabled in the Linux kernel. With `CONFIG_NOHZ_FULL`, the physical CPU will trigger many fewer clock tick interrupts (currently, 1 tick per second). This can reduce latency because each host timer interrupt triggers a VM exit from guest to host which causes performance/latency impacts.
- **Time Stamp Counter (TSC):** Mark TSC clock source as reliable. A TSC clock source that seems to be unreliable causes the kernel to continuously enable the clock source watchdog to check if TSC frequency is still correct. On recent Intel platforms with constant TSC/Invariant TSC/Synchronized TSC, the TSC is reliable so the watchdog is useless but cause latency.
- **Idle:** The poll option forces a polling idle loop that can slightly improve the performance of waking up an idle CPU.
- **`RCU_NOCB`:** Read-Copy-Update (RCU) is a kernel synchronization mechanism. Refer to http://lxr.free-electrons.com/source/Documentation/RCU/ what is `RCU.txt` for more information. With `RCU_NOCB`, the impact from RCU to the VNF will be reduced.
- **Disable the RT throttling:** RT throttling is a Linux kernel mechanism that occurs when a process or thread uses 100% of the core, leaving no resources for the Linux scheduler to execute the kernel/housekeeping tasks. RT throttling increases the latency so should be disabled.
- **NUMA Configuration:** To achieve the best latency. CPU/memory and device allocated for real-time application/VM should be in the same NUMA node.

2.4 Intel® RDT Overview

Intel® RDT is a set of Intel technologies, namely CMT, CAT, Code, and CDP and MBM that provide the hardware framework to monitor and control the utilization of shared resources, like Last Level Cache (LLC) and main (DRAM) memory bandwidth. As multithreaded and multicore platform architectures continue to evolve, running workloads in single-threaded, multithreaded, or complex virtual machine environment such as in NFV, the last level cache and memory bandwidth are key resources to manage and utilize based on the nature of workloads. Intel introduces CMT, MBM, CAT and CDP to manage these various workloads across shared resources. Although this document strictly focuses on CAT and CMT, more details on all of the aforementioned technologies and Intel® RDT in general are in Appendix C.1: Test Setup.

2.4.1 Cache Monitoring Technology and Memory Bandwidth Management

CMT and Memory MBM are features that allows an Operating System (OS) or hypervisor or Virtual Machine Monitor (VMM) to determine the usage of cache and memory bandwidth by applications running on the platform. Use CMT and MBM to do the following:

- To detect if the platform supports these monitoring capabilities (via CPUID).
- For an OS or VMM to assign an ID for each of applications or VMs that are scheduled to run on a core. This ID is called the Resource Monitoring ID (RMID).
- To monitor cache occupancy and memory bandwidth on a per-RMID basis.
- For an OS or VMM to read LLC occupancy and memory bandwidth for a given RMID at any time.

2.4.2 Cache Allocation Technology and Code and Data Prioritization

CAT and CDP are features that allows an OS, hypervisor, or VMM to control allocation of a CPU's shared LLC. Once CAT or CDP is configured, the processor allows access to portions of the cache according to the established COS. The processor obeys the COS rules when it runs an application thread or application process. This can be accomplished by performing these steps:

- Determine if the CPU supports the CAT and CDP feature.
- Configure the COS to define the amount of resources (cache space) available. This configuration is at the processor level and is common to all logical processors.
- Associate each logical processor with an available COS.
- Run the application on the logical processor that uses the desired COS.

Following SKUs of Intel® Xeon® processors support both CAT and CMT:

- Intel® Xeon® Processor E5 2658 v3
- Intel® Xeon® Processor E5 2658A v3
- Intel® Xeon® Processor E5 2648L v3
- Intel® Xeon® Processor E5 2628L v3
- Intel® Xeon® Processor E5 2618L v3
- Intel Xeon Processor E5 2608L v3
- All SKUs of Intel® Xeon® Processor D Product Family
- All SKUs of Intel® Xeon® Processor E5-2600 v4 Product Family

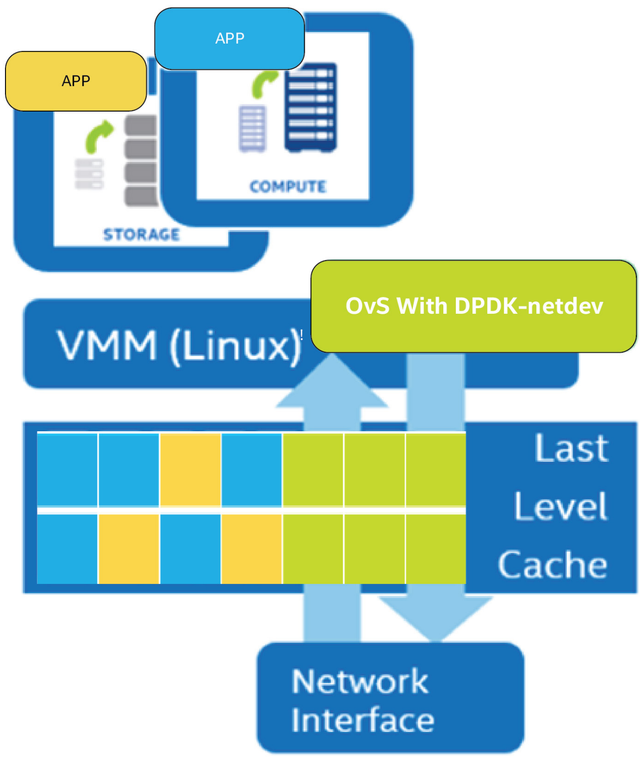

Figure 3. Allocating and Associating Cache Segments per Process Using CAT

2.4.3 The `intel-cmt-cat` Software Package

The `intel-cmt-cat` is a software package that provides basic support for CMT, MBM, CAT, and CDP Technology, and includes the Platform Quality of Service (PQoS) utility. Refer to https://github.com/01org/intel-cmt-cat for specific information with regard to CMT, MBM, CAT and CDP software package details.

After compilation, the PQoS executable can be used to configure the last level cache allocation feature and monitor the last level cache occupancy as well as memory bandwidth. The compilation and execution details are provided in the README file of the package.

The PQoS utility can be used by typing the following commands:`./pqos -h`

This option will display extensive help page. Refer to "-h" option for usage details.

`./pqos -s` Shows current CAT, CMT and MBM configuration.

`./pqos -T` Provides top like monitoring output.

`./pqos -f FILE` Loads the commands from selected file.

2.4.4 Intel® RDT Enabling in OpenStack

OpenStack is a leading open-source software suite for creating private and public clouds. The functionality provided can be, for the most part, classified under similarly intentioned names such as Cloud Operating System or Virtualized Infrastructure Manager (VIM). OpenStack is used to manage pools of computer, networking, and storage infrastructure. These infrastructure resources are typically based on industry-standard, high volume servers and can be partitioned and provisioned for the user in an on-demand style by means of a Command Line Interface (CLI), RESTful API, or a web interface. OpenStack was released to the open-source community in 2010 and has since grown in popularity with an active community of users and contributors. The code is released under an Apache 2.0 license.

The OpenStack compute service is called Nova*. It is responsible for managing all compute infrastructure in an OpenStack managed cloud. Multiple hypervisor drivers are supported including QEMU*/KVM* (by means of libvirt), Xen*, and VMware vSphere* Hypervisor (VMware ESXi*). Nova contains the scheduling functionality that is used to select which compute host runs a particular workload. It filters all available platforms to a suitable subset of platforms based on the input requirements, and then selects a platform from the subset based on a weighting routine.

Intel is collaborating with others in the OpenStack community to enable Intel® RDT. It is anticipated that CMT will be enabled in the OpenStack Newton release. This will enable OpenStack users to access to CMT data via Ceilometer, the OpenStack data collection service. Ceilometer will receive the data from Nova when the QEMU/KVM hypervisor is configured. The OpenStack Ocata release is the target intersect for enabling CAT, CDP and MBM. Details on how CAT, CDP and MBM will be leveraged with OpenStack Ocata will be published in conjunction with the Ocata release.

Note: The anticipated releases of OpenStack that will contain Intel® RDT are subject to change.

3 VNF Profiling

In order to provide a performance guarantee it is important to profile the workload/VNF before deployment. To find the required last level cache for optimum performance, developers or system administrator would have to monitor the last level cache occupancy before deployment. Subsequently, in production cache allocation can be used to isolate to appropriate amount of last level cache for guarantee performance. The next two sections will describe this process. Refer to Appendix A: and Appendix B: for the hardware and software configuration of the setup.

3.1 Last Level Cache Occupancy

As mentioned earlier, the performance of the VNFs, and generally speaking all the other software components, on the platform can be affected by the size of LLC. Logic needs to be applied in order to determine the conditions as to when a VNF will require the maximum cache size, and how much that maximum size would be for optimal performance. In other words, one needs to be able to determine the worst case conditions in order to push the VNF or software component to require as much LLC as possible while not affecting the overall performance. Most certainly it will be required to stress the platform with a networking workload at highest throughput rate possible and with the type of traffic that would be expected to be seen in the production environment. While the platform is under the maximum sustainable stress, without any cache allocations, CMT can be used to measure/monitor the profile of these components.

In the Intel case, 64, 256, 1024 and 1518 byte size packets were injected at a 10 Gb/s rate. Using the RFC2544 methodology, the throughput was measured for the acceptable rate loss (0.1%) and measured the maximum LLC occupancy using CMT for the software infrastructure components under two different test cases of PHY-VM-PHY and PHY-VM-VM-PHY. The software infrastructure components are OVS vSwitched daemon, two DPDK Poll Mode Drivers (PMD) running on the host and VMs (QEMU threads). Table 1 below shows the results, and The LLC occupancy information provides profile of extent of cache occupied by each SW component used. Out of the total of 30 MB of LLC cache available (for Intel® Xeon® E5-2658 processor), Table 1 indicates maximum occupancy at optimal throughput conditions by various packet sizes. This information is later used to calculate the cache class of service associations.

Figure 4 shows the graphs of the LLC occupancy –vs- each component. The different colors of the bars denote the different packet sizes as per the legend at the bottom of the graph. Refer to Appendix C for more details about the setup and software infrastructure details.

The LLC occupancy information provides profile of extent of cache occupied by each SW component used. Out of the total of 30 MB of LLC cache available (for Intel® Xeon® E5-2658 processor), Table 1 indicates maximum occupancy at optimal throughput conditions by various packet sizes. This information is later used to calculate the cache class of service associations.

Packet Size in Bytes	Test Case	vswitchd	PMD1	PMD2	VM1 OS	VM1 PMD2	VM1 PMD2	VM2 OS	VM2 PMD1	VM2 PMD2
64	**Single VM**	1200	7104.00	6768.00	384.00	1824.00	816.00	-	-	-
	Two VM	1824	8160.00	7536.00	144.00	192.00	1296.00	624.00	192.00	192.00
256	**Single VM**	1248	7392.00	7296.00	144.00	576.00	336.00	-	-	-
	Tw0 VM	1536	7440.00	6720.00	96.00	288.00	864.00	480.00	192.00	288.00
1024	**Single VM**	720.00	11856.00	12144.00	144.00	720.00	48.00	-	-	-
	Tw0 VM	1008.00	9360.00	9600.00	48.00	96.00	144.00	192.00	144.00	240.00
1518	**Single VM**	672.00	11952.00	11376.00	240.00	336.00	96.00	-	-	-
	Tw0 VM	1200.00	11424.00	11616.00	48.00	336.00	240.00	192.00	144.00	144.00

Table 1. LLC Occupancy in KB by Processes at Maximum Throughput as per RFC2544

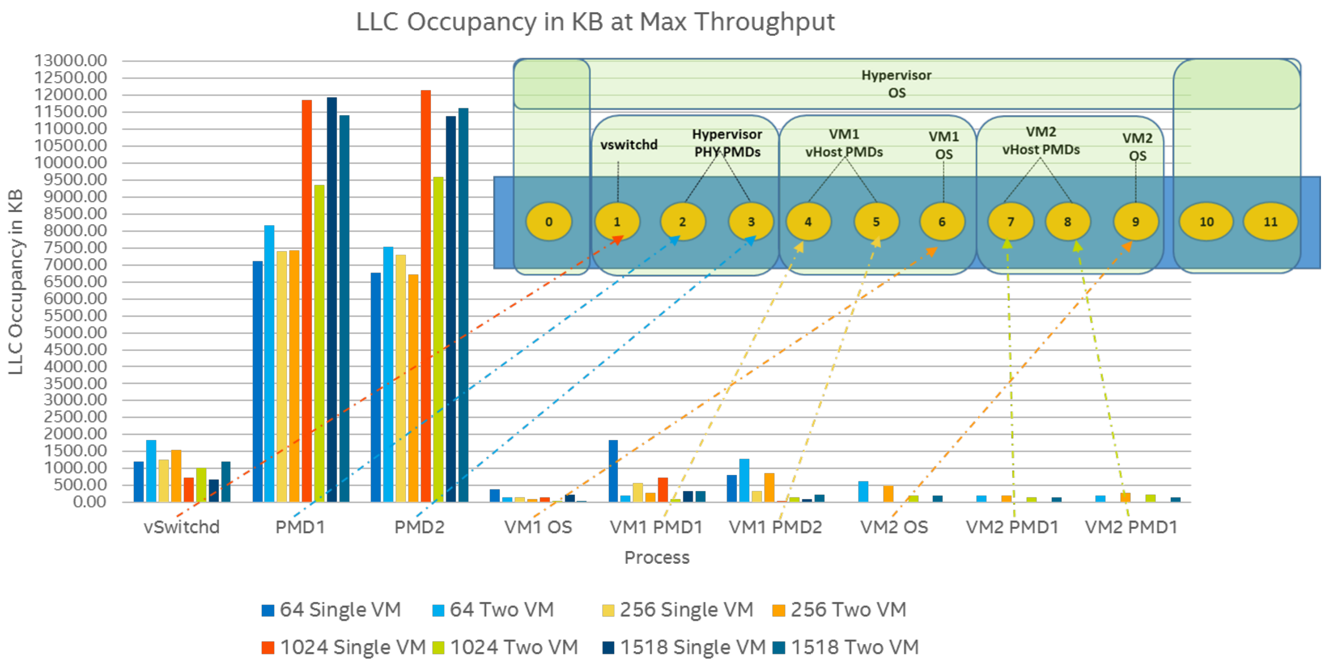

Figure 4. LLC Occupancy by Processes at Maximum Throughput as per RFC2544

Based on Intel profile, the following software infrastructure components were found to require the following maximum LLC occupancies for each of the SW infrastructure components under maximum sustainable throughput. This data is further used to divide the COS classes that allocates LLC usage per component.

Component	Maximum LLC Footprint
Vswitchd	2 MB
DPDK PMDs in the hypervisor	12 MB
VM (L2 forwarding)	2 MB

Table 2. (Maximum) LLC Profiling Required per Component

3.2 Class of Services

The term Class of Services is used widely in Quality of Service (QoS), where it denotes logical pipes with different priorities. In a similar fashion for CAT, class of service denotes a segment of LLC that is assigned for the sole use of one or more cores of the same CPU die. In other words, when this particular core or cores are executing memory reads or writes, their data get cached in these cache segments. This also means that the other cores of the same CPU die, when executing read or write transactions, they read or write outside of the cache segments belonging to the class of service.

The size of classes of service is measured by number of cache ways. For the specific CPU SKU that was used, the cache way has size of 1.5 MB. These cache ways can be overlapped or isolated as discussed in the next section.

3.3 Overlapped vs. Isolated Classes of Service

Knowing the required COS per software component is one part of the characterization. The next part requires identifying the type of allocation schema that will yield a better performance. Usually, consider if the software components will work better sharing or isolating their caches. Although considering these interactions might allow better prediction, experimental analysis is required to verify. Figure 5 gives an example of how the default COS case differs from overlapped and isolated COS cases. More details can be found in *Intel® Architecture Software Developer Systems Programming Manual* at: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-system-programming-manual-325384.pdf

Note: COS distribution provided in Figure 5 are independent of the current platform and provided *only* as an example.

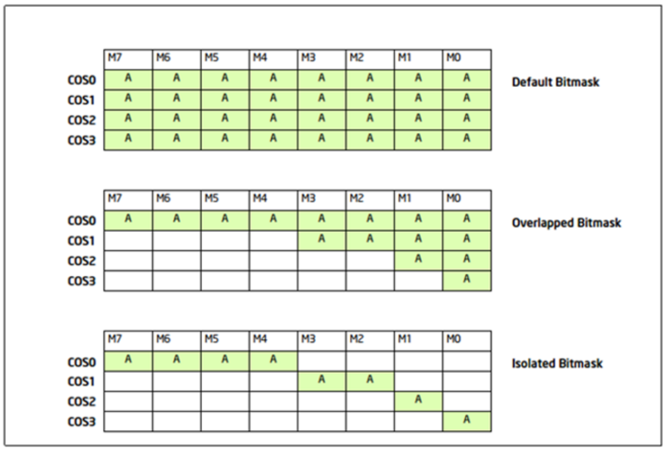

Figure 5. Example of Cache Allocation with Overlapped and Isolated COS

For the processor used in current study, Intel® Xeon® E5-2658 v3, has 20 cache ways in LLC. The Table 3 shows the COS distribution of cache ways for each SW component used. The size in MB signifies the amount of cache available for corresponding SW component. The bitmask value in hex signifies the corresponding cache size made available for the SW component.

	ovs-vswitchd/PMDs	VM Under Test	Noisy Neighbor VM	Operating System
Overlapping Case 1	24 MB (0xFFFF0)	6 MB (0x00F00)	6 MB (0x000F0)	6 MB (0x0000F)
Isolated Case 1	12 MB (0xFF000)	6 MB (0x00F00)	6 MB (0x000F0)	6 MB (0x0000F)
Overlapping Case 2	24 MB (0xFFFF0)	3 MB (0x000C0)	3 MB (0x00030)	6 MB (0x0000F)
Isolated Case 2	18 MB (0xFFF00)	3 MB (0x000C0)	3 MB (0x00030)	6 MB (0x0000F)

Table 3. Division of Cache Ways for Isolated and Overlapped COS Test Cases

Based on the above schemas, a DPDK-OvS based platform was deployed and VMs were spawned as shown in Figure 15. The throughput and latency are obtained at maximum acceptable packet loss (0.1%) with VM under test configured as a L2 forwarder and a second VM is acting as noisy neighbor with memtester. The results are detailed below.

Note: Having non zero packet loss might introduce some inaccuracies in latency measurement by the tools used.

The naming scheme used below is `<Type of COS> Case#` (PMD's Cache size in MB – VM1's Cache size in MB – Noisy Neighbor VM's Cache size in MB – Operating System's cache size in MB). For example, "Overlapping Case 1 (24-6-6-6)" can be deconstructed as:

- Overlapping: It is an overlapping class of service allocation type.
- Case 1: First case of its type
- 24: PMDs in this case have 24-MB of cache allocated.
- 6: VM1 (or VM under test) has 6-MB of cache allocated.
- 6: Noisy neighbor VM (with memtester) has 6-MB of cache allocated.
- 6: Operating System (and rest of the processes) has 6-MB of cache allocated.

This can be expanded for rest of the cases used below.

3.3.1 Difference in Throughput

Packet Sizes in Bytes	Overlapping Case 1 (24-6-6-6)	Isolated Case 1 (12-6-6-6)	Overlapping Case 2 (24-3-3-6)	Isolated Case 2 (18-3-3-6)
64	3,980,656	3,980,658	3,980,655	3,980,656
256	3,873,691	3,803,639	3,873,695	3,873,691
1024	2,394,630	2,394,617	2,394,626	2,394,614
1518	1,625,485	1,625,484	1,625,484	1,625,484

Table 4. Throughput in Packets per Second Comparison in PPS with Varying COSes

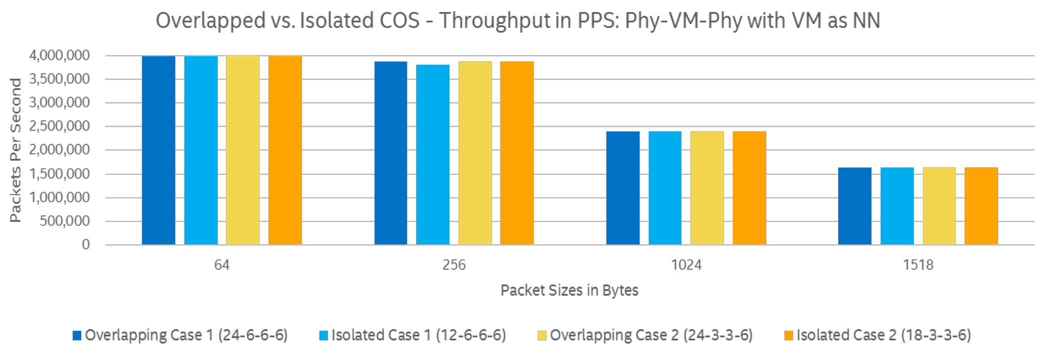

Figure 6. Throughput Comparison in PPS with Varying COSes

Based on Figure 6 results, it is apparent that throughput is not affected by the different allocation schemas. This stems from mainly the fact that the profile of the L2 forwarding VM is only 2-MB and the allocated cache is sufficient that throughput is not impacted. The only exception seems to be the isolated case 1 for 256B packets which exhibited about 70 kpps lower performance. This performance degradation however is 1.8% which in many cases would be considered negligible.

3.3.2 Difference in Latency

Packet Sizes in Bytes	Overlapping Case 1 (24-6-6-6)	Isolated Case 1 (12-6-6-6)	Overlapping Case 2 (24-3-3-6)	Isolated Case 2 (18-3-3-6)
64	61	80	59	51
256	175	202	165	160
1024	224	327	256	633
1518	104	141	108	119

Table 5. Latency Comparison in μsecs with Varying COSes

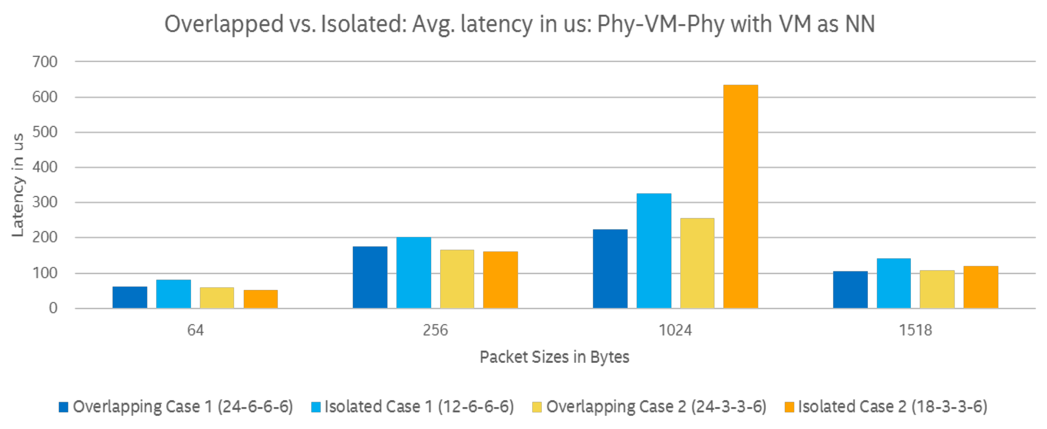

Figure 7. Latency Comparison in µsecs with Varying COS'es

In terms of latency, it becomes apparent that the Isolated Case 2 (yellow) can be problematic for 1024B packets. Also, the Isolated Case 1 seems to be not favorable as it exhibits the highest latency for 256B and 1518B packets while it is the second worst case for 1024B packet sizes. Both Overlapping Cases 1 and 2 seem to be equivalent, with small average differences between them, one being best option for smaller packet sizes (case 1), and the other being the best option for bigger packet sizes (case 2).

3.3.3 Determinism in Latency

After looking at the performance that each schema yields, the deterministic latency predictability was looked into of the platform based on the same schemas. Latency bins was used that show concentration of the packet latencies. Apparently the narrower the concentration is, the more deterministic the performance is and the more these concentrations are towards the left, the lower (better) latency obtained. Latency bin information is shown for 64B and 1518B packets to cover typical telco or cloud/enterprise type traffic's packet sizes.

3.3.3.1 COS for 64B Packets

Latency Bins in µsecs	Overlapping Case 1 (24-6-6-6)	Isolated Case 1 (12-6-6-6)	Overlapping Case 2 (24-3-3-6)	Isolated Case 2 (18-3-3-6)
5 µs – 10 µs	0	78	0	26
10 µs – 25 µs	2232945	1785229	1980879	1731249
25 µs – 50 us	54615309	9982799	68762669	61899614
50 µs – 75 µs	148829151	74247866	152705307	157655376
75 µs – 100 µs	31391425	134770032	15198439	17524017
100 µs – 150 µs	1733897	18042127	191992	29004
150 µs – 200 µs	4896	9586	0	0
200 µs – 250 µs	5653	1569	0	0
250 µs – 500 µs	26010	0	0	0

Table 6. 64B Latency Bins Comparison in µsecs with Varying COSes

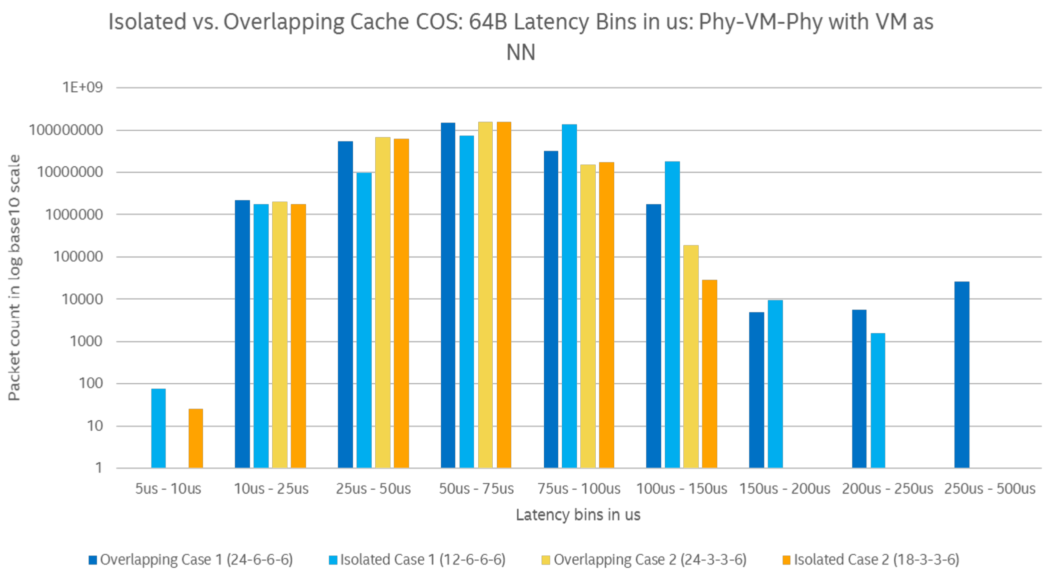

Figure 8. 64-B Latency Bins Comparison in µsecs with Varying COSes

Figure 8 shows Overlapping Case 1 and Isolated Case 1 span into more latency buckets, whereas Overlapping Case 2 is the best in terms of predictability (spans into the least number of low latency buckets) and Isolated Case 2 is the close second option.

3.3.3.2 COS for 1518B Packets

Latency Bins in µsecs	Overlapping Case 1 (24-6-6-6)	Isolated Case 1 (12-6-6-6)	Overlapping Case 2 (24-3-3-6)	Isolated Case 2 (18-3-3-6)
5 µs – 10 µs	0	0	0	0
10 µs – 25 µs	11	27	1	18
25 µs – 50 us	111	95	4	226
50 µs – 75 µs	9400	267	1379	308
75 µs – 100 µs	37598684	937	34177318	20034386
100 µs – 150 µs	59920862	71371436	63348353	77437736
150 µs – 200 µs	190	26156345	2203	56584
200 µs – 250 µs	0	151	0	0
250 µs – 500 µs	0	0	0	0

Table 7. 1518B Latency Bins Comparison in µsecs with Varying COSes

For the 1518B packet sizes in Figure 9, Isolated Case 1 seems the least predictable option and Overlapping Case 2 is the best (spans into least number of buckets). Isolated Case 2 and Overlapping Case 1 exhibit similar results in terms of predictability.

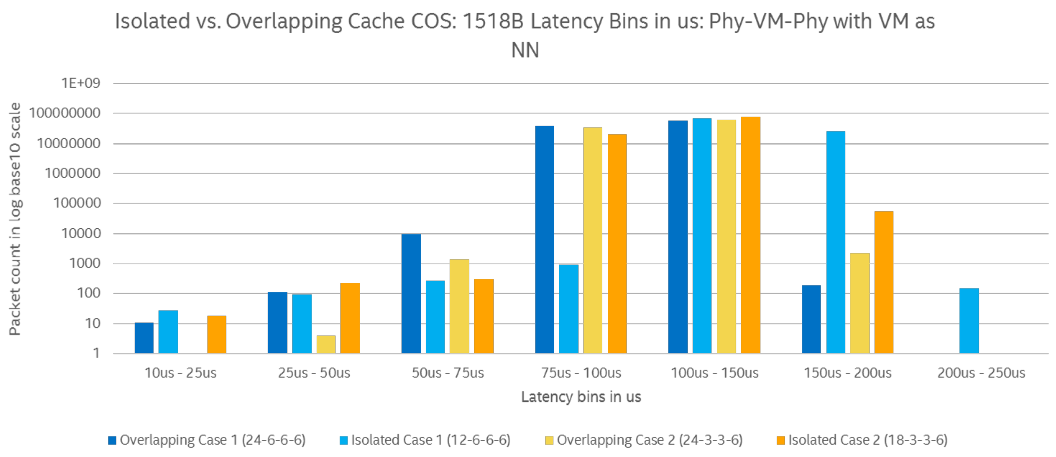

Figure 9. 1518B Latency Bins Comparison in μsecs with Varying COSes

3.4 Analysis of Determinism

Based on the performance analysis the two best (and similar in terms of results) schemas were the two overlapping cases. From the deterministic predictability analysis, it is made clear that Overlapping Case 2 is the one that yields the most predictable results. Therefore, Overlapping Case 2 (24-3-3-6) was chosen to move forward and execute the set of tests with a noisy software component (either noisy VM or noisy application in the hypervisor) using this schema.

4 Test Results

This section describes how the technology can be used to achieve considerably lower and much more predictable latency in a VM with a (simulated) noisy neighbor. It might be worth mentioning that the noisy neighbor can be either another VM/VNF that just happens to generate a lot of traffic, or it can be a malicious neighbor trying to starve the forwarding VM from resources such as cache and affect its performance. In this case, it is shown below that CAT can protect and isolate VMs/VNFs from each other avoiding the negative effects of an intentionally malicious (or not) noisy neighbor. The noisy neighbor in our study is simulated using memtester software tool. Memtester is a userspace utility for stress testing memory subsystem. We use memtester to stream synthetic traffic upto 12 Gbytes/s per core from memory there by quickly polluting LLC.

Having performed the tests the same way as previously discussed (of finding the throughput rate that yields acceptable percentage loss, and then measuring the average latency and latency buckets at that rate) with a L2 forwarding VM acting as the VNF under test, the following results were obtained. It should be noted that three configurations are used for each test case:

1. **NoCAT-NoMemtester:** No cache allocations (using CAT) were set and no memtester was running to act as noisy application (to yield the performance impact in the standard default configuration). This is the default setup of the platform.
2. **NoCAT-Memtester:** No cache allocations (using CAT) were set but memtster was running to act as noisy application either in hypervisor or in the VM, based on the case (to yield the performance impact in the standard configuration with noisy application).
3. **WithCAT-Memtester:** CAT was used with specific COS and memtester was running to act as noisy application either in hypervisor or in the VM, based on the case (to showcase the impact of CAT with a noisy application).

Refer to Appendix A: and Appendix B: for the hardware and software configuration of the setup.

4.1 PHY-VM-PHY with Noisy Process on the Hypervisor

This test setup is being used to emulate an application in the hypervisor that is being aggressive with cache to the point where it has the potential to make the VNF(s) or VM(s) starve in terms of cache. For this purpose, one COS for all the processes of the operating system, including memtester was created. The other infrastructure and forwarding VM components would have a COS on their own. The following figure shows the COS allocations and core affinities on the platform.

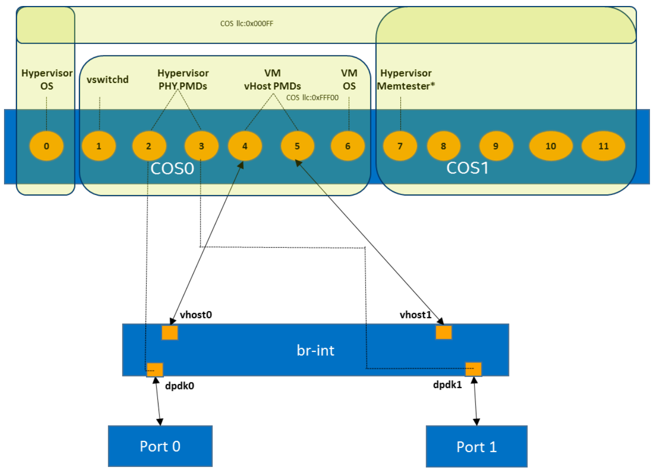

Figure 10. Core Affinities and Cache Association for the PHY-VM-PHY Setup with Memtester in Hypervisor as a Noisy Neighbor (NN)

Table 8 shows the COS with its size in software component level.

Physical Core(s)	Process	COS Bit Mask	Cache Restored
1	ovs-vswitchd	0xFFF00	18 MB
2	PMD1	0xFFF00	18 MB
3	PMD2	0xFFF00	18 MB
7, 8, 9	Qemu	0xFFF00	18 MB
11	Memtester	0x000FF	12 MB
0, 4-6, 10	OS	0x000FF	12 MB

Table 8. Core Associations and Overlapping Cache Allocations for Hypervisor Processes

4.1.1 Throughput

Packet Size in Bytes	NoCAT-No Memtester	NoCAT-Memtester	WithCAT-Memtester
64	3,980,656	3,980,649	3,750,465
256	3,803,620	3,663,501	3,663,526
1024	2,394,611	2,394,584	2,394,605
1518	1,625,478	1,625,468	1,625,484

Table 9. Throughput Comparison in Packets per Second with Memtester as Noisy Neighbor in Hypervisor

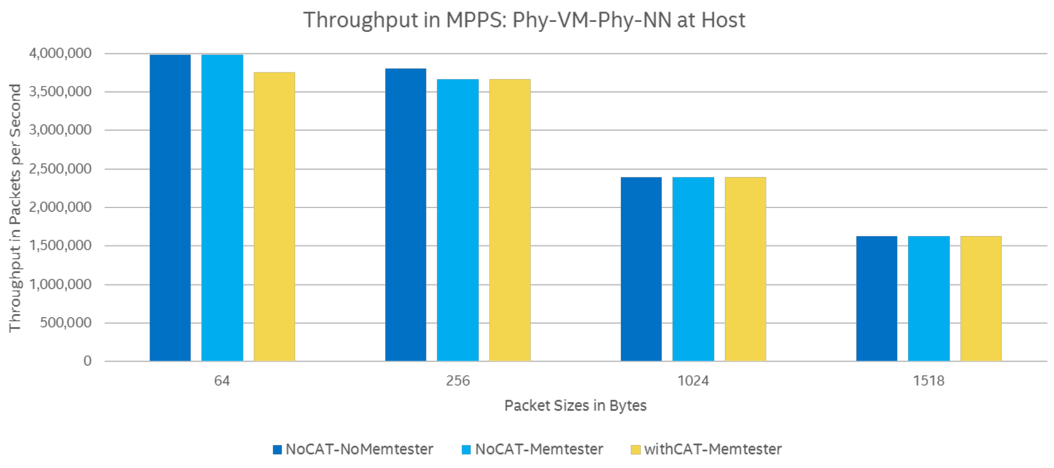

Figure 11. Throughput comparison in Packets per Second with Memtester as Noisy Neighbor in Hypervisor

4.1.2 Latency

Packet Size in Bytes	NoCAT-No Memtester	NoCAT-Memtester	withCAT-Memtester
64	48	56	30
256	119	161	117
1024	688	1388	809
1518	421	739	121

Table 10. Average Latency comparison in µsecs with Noisy Neighbor in Hypervisor

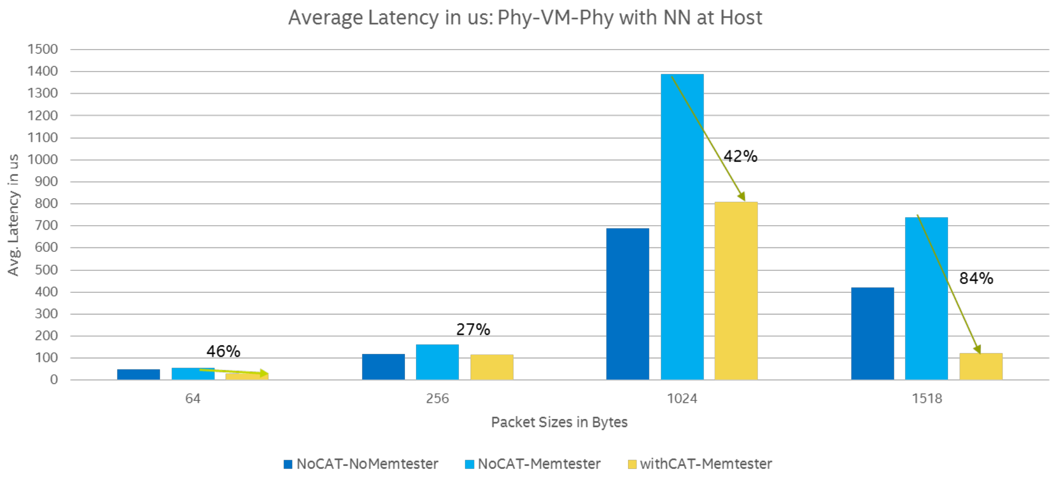

Figure 12. Average Latency Comparison with Noisy Neighbor in Hypervisor

The above results leads to conclude that the use of CAT is highly beneficial with or without the presence of a noisy application running in the hypervisor. Without a noisy application, the results demonstrate a good level of latency reduction, which is magnified in the presence of a running noisy application. In the case where a noisy application is running in parallel, there is an improvement of 27% in worst case and a huge improvement of 84% in the best case. It is worth noting that in some cases, the latency with CAT present is even lower than the latency without CAT and without a noisy neighbor.

In terms of throughput, the performance stayed the same in all cases with the only exception where the noisy application reduced the throughput by less than 6% in presence of a noisy application for the 64B packets. Also for the 256B packets, the presence of noisy neighbor had a negative effect in the throughput of all the other cases by less than 4%.

4.1.3 Deterministic Predictability Analysis

Latency Bins in µsecs	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
5 µs – 10 µs	0	1664	9842
10 µs – 25 µs	1830	16019176	42170855
25 µs – 50 µs	142808621	54466586	181468399
50 µs – 75 µs	95908820	146951007	1377672
75 µs – 100 µs	84378	19872346	1134
100 µs – 150 µs	6822	893301	0
150 µs – 200 µs	6648	173521	0
200 µs – 250 µs	7035	198978	0
250 µs – 500 µs	15132	222275	0
500 µs – 750 µs	0	14499	0
750 µs – 1000 µs	0	9531	0
1000 µs - max	0	15803	0

Table 11. 64B Packets Latency Bins Comparison in µsecs with Noisy Neighbor in Hypervisor

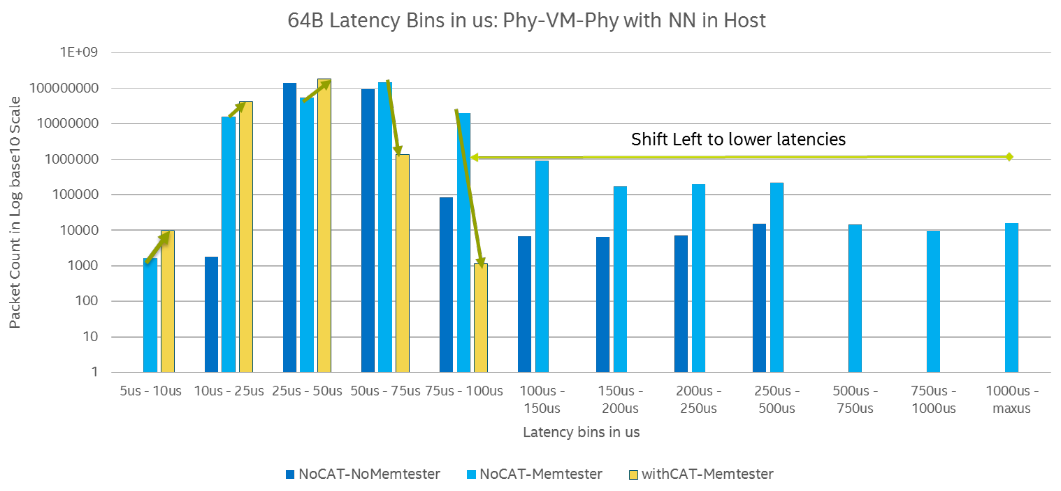

Figure 13. 64B Packets Latency Bins Comparison in µsecs with Noisy Neighbor in Hypervisor

Latency Bins in µsecs	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
10 µs – 25 µs	0	0	23
25 µs – 50 µs	1	0	100
50 µs – 75 µs	0	2	270
75 µs – 100 µs	1	0	1833590
100 µs – 150 µs	189	0	95675501
150 µs – 200 µs	256	128	19774
200 µs – 250 µs	416	128	0
250 µs – 500 µs	97528395	1024	0
500 µs – 750 µs	0	80741326	0
750 µs – 1000 µs	0	16786631	0

Table 12. 1518B Packets Latency Bins Comparison in µsecs with Noisy Neighbor in Hypervisor

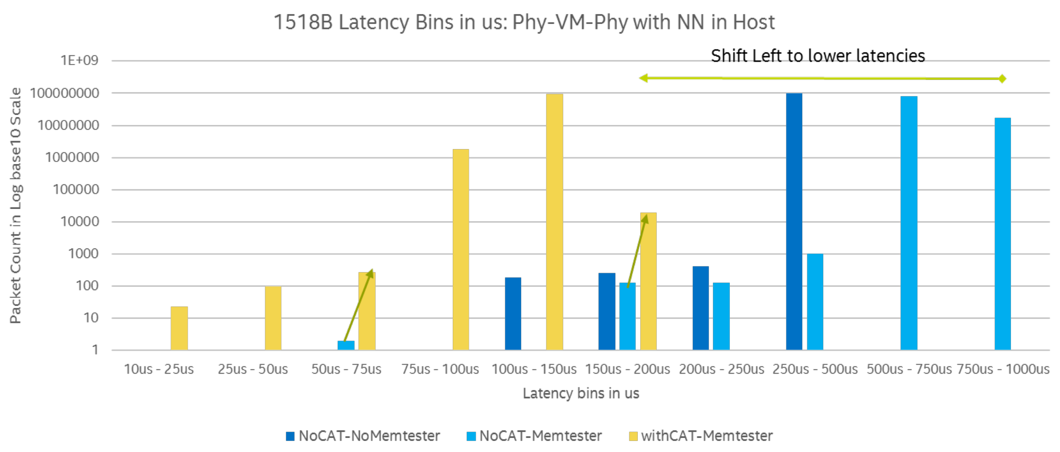

Figure 14. 1518B Packets Latency Bins Comparison with Noisy Neighbor in Hypervisor

In terms of determinism, the presence of CAT, without a question, improved the predictability:

- For 64B packets the latency variation became 5 to 100 µsecs –vs- 5 µsecs to above 1 msec in the presence of a noisy neighbor.
- For 1518B packets the latency variation became 10 to 200 µsecs –vs- 50 µsecs to 1 msec in the presence of noisy neighbor

4.2 PHY-VM-PHY with Noisy Process inside a VM (Noisy Neighbor)

This setup emulates the case where a VM is acting as a noisy neighbor in the platform. Figure 15 shows running the memtester inside a VM. There is one COS for the Operating System, one for the vSwitch including the PMDs and one for each of the forwarding VM and the noisy neighbor that runs memtester.

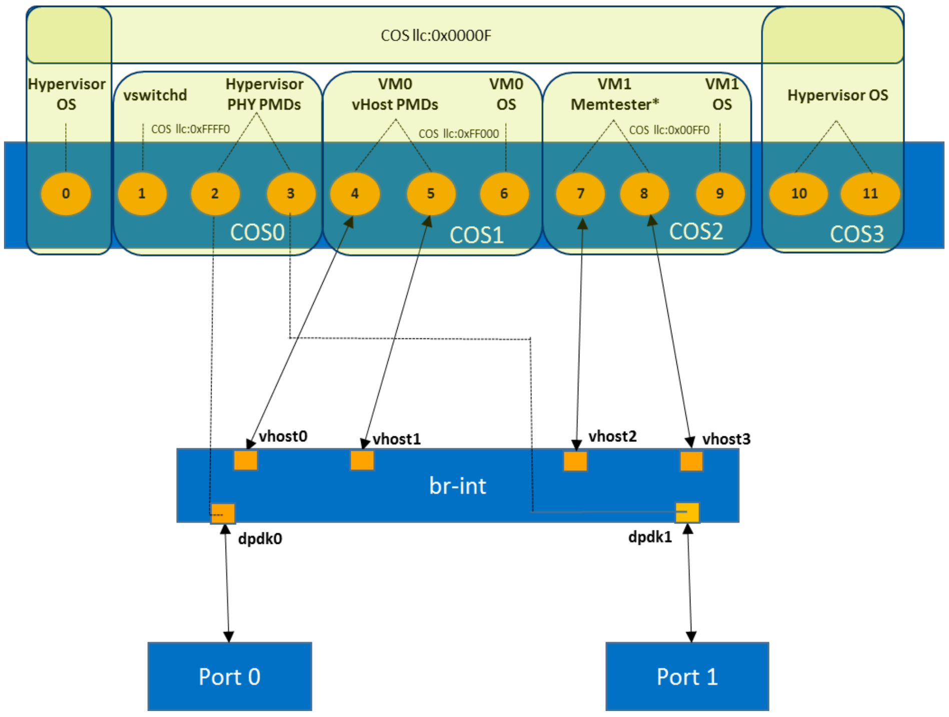

Figure 15. Core Affinities and Cache Association for the PHY-VM-PHY Setup with VM as a Noisy Neighbor (NN)

The COS test example has been allocated using overlapping COS as shown in Table 13 below and based on Section 3.4, Analysis of Determinism detailed above.

Physical Core	Process	COS Bit Mask	Cache Restored
1	ovs-vswitchd	COS0:0xFFFF0	24 MB
2	PMD1	COS0:0xFFFF0	24 MB
3	PMD2	COS0:0xFFFF0	24 MB
4,5,6	VM1 (Qemu)	COS1:0x00F00	3 MB
7,8,9	VM2 (Qemu)	COS2:0x000F0	3 MB
0	OS	COS3:0x0000F	6 MB

Table 13. Core Affinities and Overlapping Cache Associations

The results below show the impact of CAT and noisy neighbor in the networking performance.

4.2.1 Throughput

Packet Sizes in Bytes	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
64	3,980,660	3,980,620	3,980,655
256	3,943,753	3,661,738	3,873,695
1024	2,394,625	2,394,582	2,394,626
1518	1,625,483	1,625,470	1,625,484

Table 14. Throughput in Packets per Second for Best CAT Association for PHY-VM-PHY with VM as Noisy Neighbor

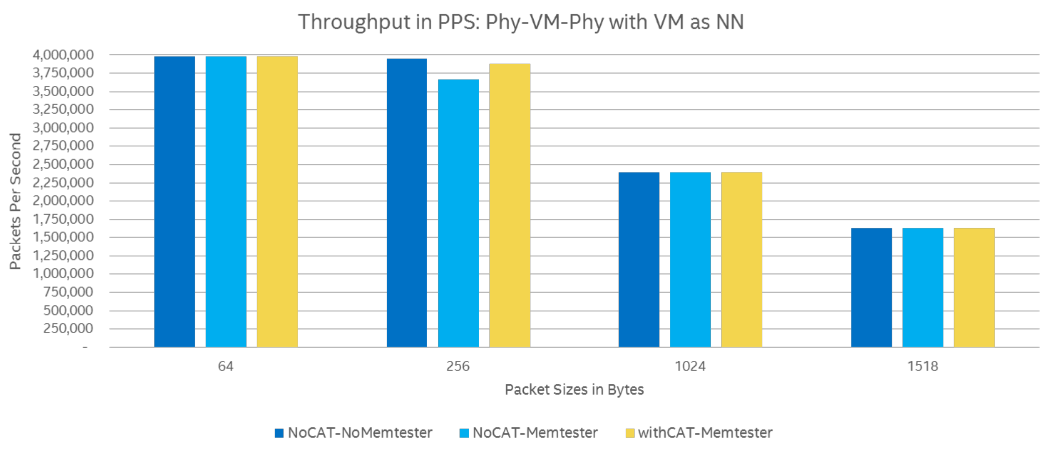

Figure 16. Throughput in Packets per Second for Best CAT Association for PHY-VM-PHY with VM as Noisy Neighbor

4.2.2 Latency

Packet Sizes in Bytes	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
64	42	85	55
256	214	458	166
1024	277	1,323	257
1518	151	649	121

Table 15. Average Latency in μsecs for Best CAT COS for PHY-VM-PHY with VM as Noisy Neighbor

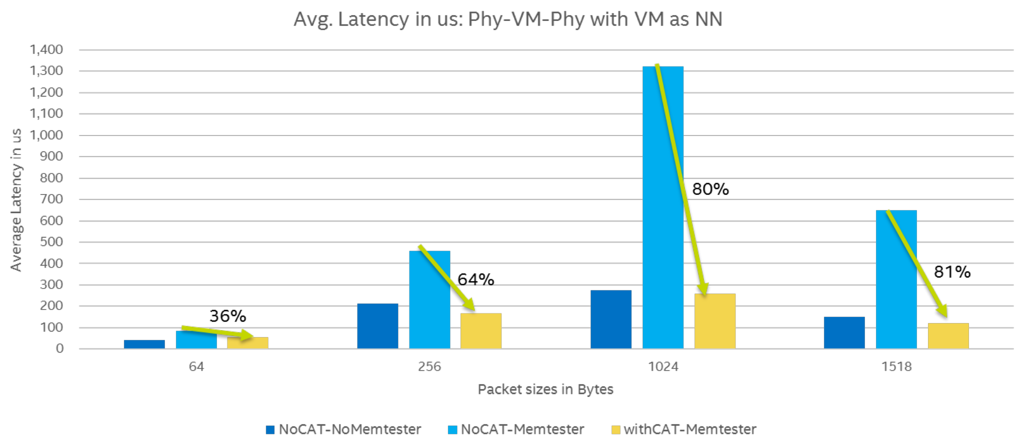

Figure 17. Average Latency in μsecs for Best CAT Association for PHY-VM-PHY with VM as Noisy Neighbor

From the above results, it can be seen that in this case too, throughput stays constant with small variations, while in the 256B packets with noisy neighbor and without CAT exhibits the greatest performance degradation where the throughput has been reduced by slightly over 7% (which CAT reduces this negative impact to less than 2%).

In terms of average latency, there is a huge positive impact in the presence of a noisy neighbor which CAT reduces by 36% in the worst case (64B packets) or 81% in the best case (1518B packets). The average latency with CAT was improved even compared to the non-CAT and non-noisy neighbor case for all packet sizes apart from 64B (in which case the improvement of CAT was 36% in the presence of noisy neighbor).

4.2.3 Deterministic Predictability Analysis

Latency bins in µsecs	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
5 µs – 10 µs	0	0	72
10 µs – 25 µs	382782	0	1998304
25 µs – 50 µs	209446984	3051813	85616436
50 µs – 75 µs	28635855	47704660	137741020
75 µs – 100 µs	298295	167458985	13153634
100 µs – 150 µs	25244	19306825	329564
150 µs – 200 µs	4509	225009	256
200 µs – 250 µs	4245	240335	0
250 µs – 500 µs	24614	784480	0
500 µs – 750 µs	16758	38641	0
750 µs – 1000 µs	0	26424	0

Table 16. 64B Packets Latency Bins in µsecs for Best CAT COS for PHY-VM-PHY with VM as Noisy Neighbor

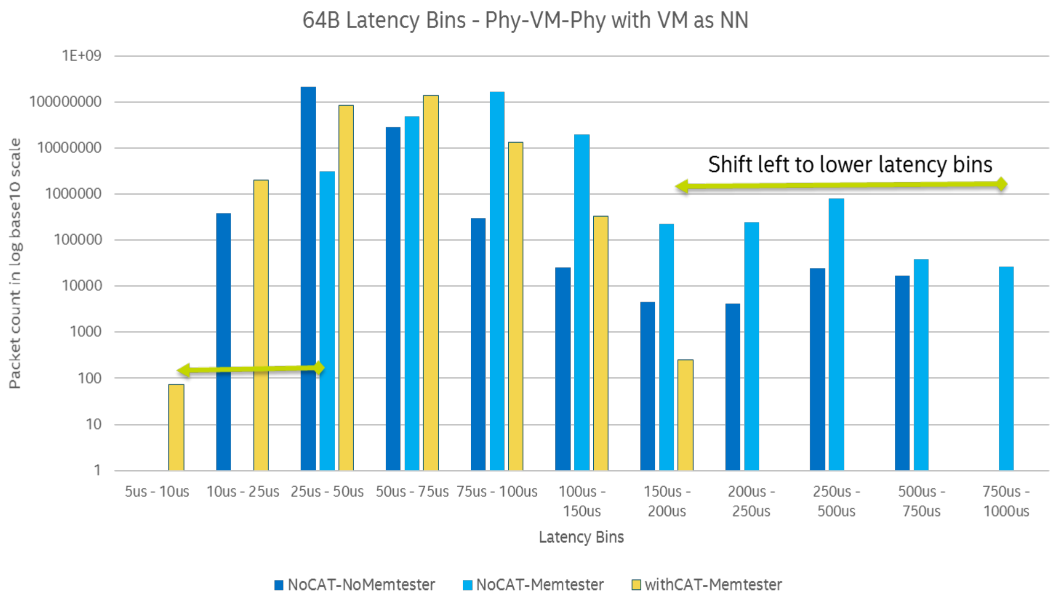

Figure 18. 64B Packets Latency Bins for Best CAT COS for PHY-VM-PHY with VM as Noisy Neighbor

Latency bins in µsecs	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
10 µs – 25 µs	7	1	19
25 µs – 50 µs	88	24	110
50 µs – 75 µs	125	26	450
75 µs – 100 µs	164	19	11943917
100 µs – 150 µs	47069316	137	84649818
150 µs – 200 µs	50458887	203	934944
200 µs – 250 µs	671	189	0
250 µs – 500 µs	0	1734	0
500 µs – 750 µs	0	97526925	0

Table 17. 1518B Packets Latency Bins in µsecs for Best CAT COS for PHY-VM-PHY with VM as Noisy Neighbor

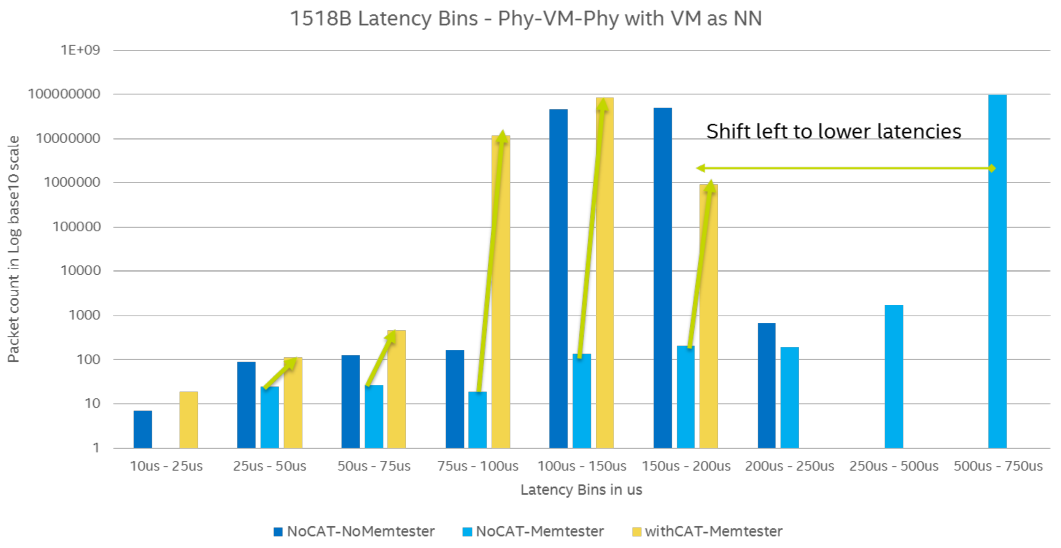

Figure 19. 1518B Packets Latency Bins for Best CAT association for PHY-VM-PHY with VM as Noisy Neighbor

In terms of determinism for networking workloads, the presence of CAT reduced the latency variation in the presence of a noisy neighbor from 25 µsecs to 1 msec, to just 5 to 200 µsecs for 64B packets and from 25 to 750 µsecs, to 10 to 200 µsecs for 1518B packets. This is a huge improvement.

4.3 PHY-VM-VM-PHY with Noisy Process inside a VM (Noisy Neighbor)

This setup emulates the case where a VM is acting as a noisy neighbor in the platform while there are two VMs acting as L2 forwarding VMs. Figure 20 shows a running memtester application inside a VM. There is one COS for the operating system, one for the vSwitch including the PMDs and one for each of the VMs under test, the two forwarding VMs. However, for the noisy neighbor VM that runs memtester only two cores were allotted due to lack of sufficient core count and due to limitation of only four COSes for this processor, it had to share the COS with hypervisor operating system.

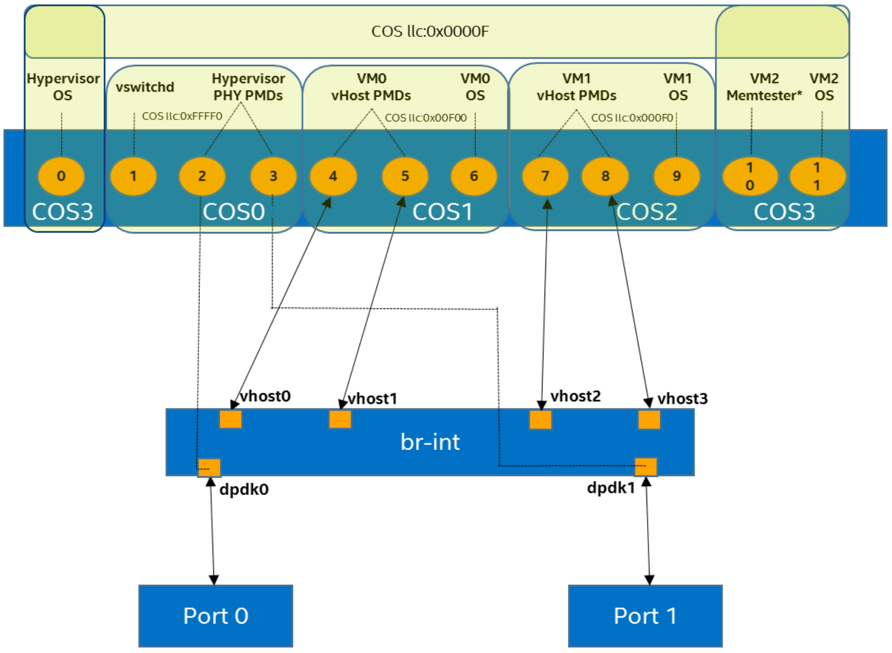

Figure 20. Core Affinities and Cache Association for the PHY-VM-VM-PHY Setup with VM as a Noisy Neighbor

The COS have been allocated using overlapping COS as shown in Table 18 below and based on the Overall Analysis Results detailed above.

Process	Physical Cores	COS Bit Mask	Cache Restored
ovs-vswitchd	1	COS0: 0xFFFF0	24 MB
PMD 1	2	COS0: 0xFFFF0	24 MB
PMD 2	3	COS0: 0xFFFF0	24 MB
VM1 (Qemu)	4,5,6	COS1: 0x00F00	3 MB
VM2 (Qemu)	7,8,9	COS2: 0x000F0	3 MB
Operating System	0	COS3: 0x0000F	6 MB

Table 18. Core Associations and Overlapping Cache Allocations for Hypervisor Processes

4.3.1 Throughput

Packet Sizes in Bytes	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
64	1,908,947	1,908,938	1,908,947
256	1,912,078	1,912,059	1,912,081
1024	1,209,252	801,778	1,172,220
1518	984,105	682,523	908,846

Table 19. Throughput Comparison in Packets per Second with PHY-VM-VM-PHY Case with VM as Noisy Neighbor

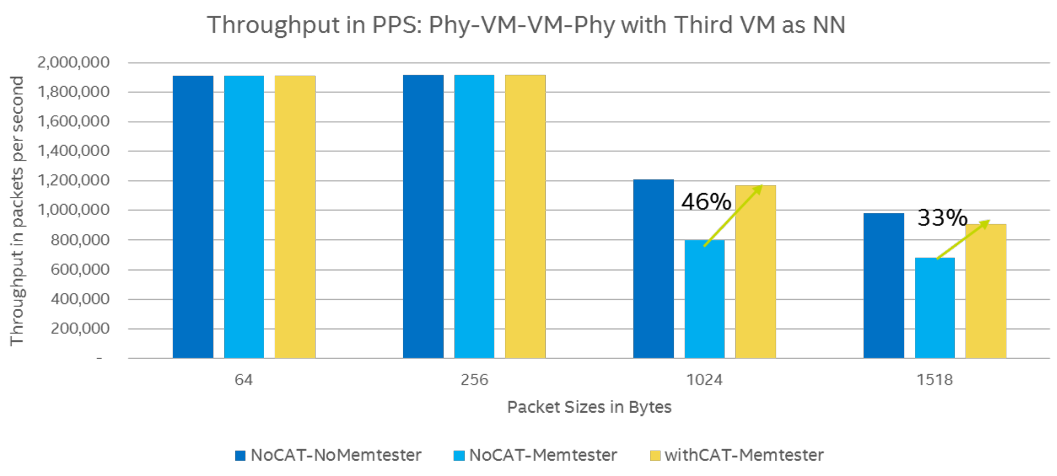

Figure 21. Throughput Comparison in Packets per Second with PHY-VM-VM-PHY Case with VM as Noisy Neighbor

In this case, where the VM density is higher, it can be seen that the presence of the noisy neighbor creates issues in the throughput, dropping the rate significantly for the larger packet sizes (approx. 33% and 31% decrease in the throughput of the VM chain for 1024B and 1518B packets respectively). Using CAT improved the performance and almost completely restored the throughput for these packet sizes.

4.3.2 Latency

Packet Sizes in Bytes	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
64	46	49	49
256	78	98	104
1024	148	34	168
1518	2,305	32	688

Table 20. Average latency comparison in µsecs for PHY-VM-PHY case with VM as Noisy Neighbor

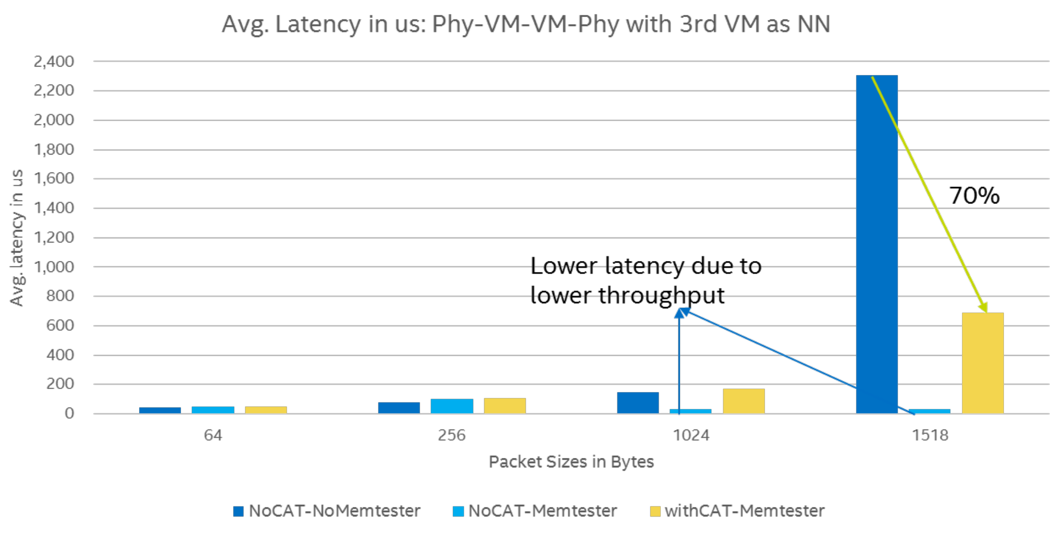

Figure 22. Average latency comparison in µsecs for PHY-VM-PHY case with VM as Noisy Neighbor

One important aspect to be noted in these latency figures is that the fact that the throughput of 1024B and 1518B packets was reduced in presence of a noisy neighbor had a knock-on effect on the average latencies for these packets lowering the values to a high degree. However, the fact that they are lower should not be interpreted as an improvement. If anything, the improvement in these results is the drop of average latency for the 1518B packets by 70% in the presence of CAT, even if there was a noisy neighbor in the platform, compared to default set up without noisy neighbor and without CAT.

4.3.3 Deterministic Predictability Analysis

Packet Sizes	NoCAT-NoMemtester	NoCAT-Memtester	withCAT-Memtester
10 µs – 25 µs	4929	221	219
25 µs – 50 µs	82971821	60402251	65317469
50 µs – 75 µs	31370646	53034755	48750717
75 µs – 100 µs	186442	1078451	467902
100 µs – 150 µs	2731	4971	523
150 µs – 200 µs	261	1080	0
200 µs – 250 µs	0	1249	0
250 µs – 500 µs	0	5005	0
500 µs – 750 µs	0	3662	0
750 µs – 1000 µs	0	3593	0
1000 µs – max µs	0	961	0

Table 21. 64B Packets Latency Bins Distribution Comparison for PHY-VM-VM-PHY Case with VM as Noisy Neighbor

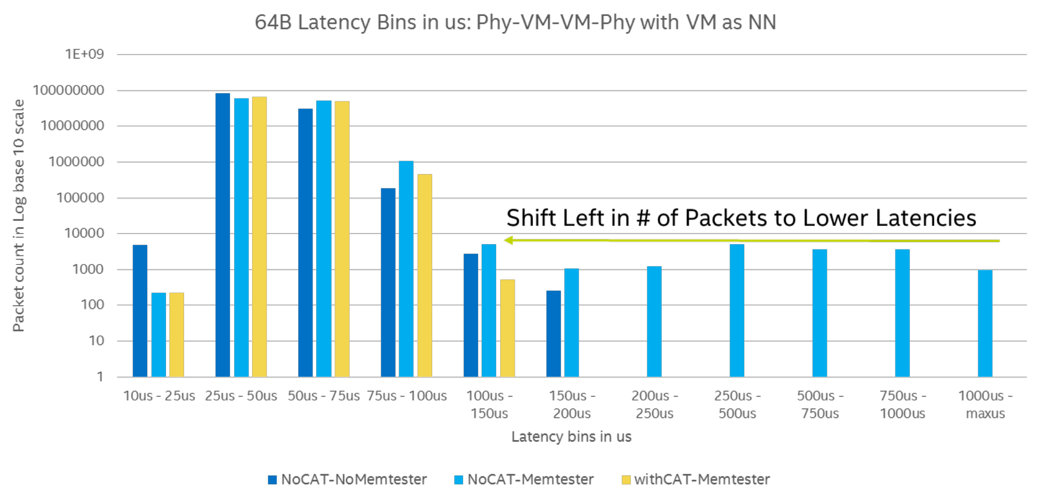

Figure 23. 64B Packets Latency Bins Distribution Comparison for PHY-VM-VM-PHY Case with VM as Noisy Neighbor

In terms of determinism, the figure above shows again the same scenario: the presence of memtester is spreading the concentration into multiple buckets from 10 µs all the way up to 1 ms and beyond, and the presence of CAT is significantly reducing the spread from 10 µs to 200 µs. There is no point showing the latency buckets for 1518B as for that packet size the throughput was negatively affected.

5 Conclusions and Deployment Considerations

Intel Resource Director Technology, in this example, cache allocation technology can provide a better platform service assurance. Using the technology helps cloud service providers or communication service providers meet their SLAs with varied set of VNFs that can be deployed general purpose COTS server platforms. It helps optimize the precious hardware resources allocation while helping achieve a predictable deterministic performance. Extending its usage to a dynamically orchestrated environment can further help the deployment to be resource (like LLC) aware and enable VNF schedulers to be intelligent enough to achieve service assurance. With the overall data detailed in earlier sections it is safe to say Intel® RDT helps achieve determinism in NFV environments that are sensitive to performance considerations.

Using CAT networking data access latency is reduced and becomes more deterministic, especially in conditions where noisy neighbors exist in the same platform. Intel Resource Director Technology also provides a strong security aspect as VMs can be safeguarded from malicious noisy neighbors which deploy Denial of Service (DoS) attack techniques and starve legitimate VMs from their cache resources.

It is important to determine the profile (size of cache segments and allocation schema) of the software infrastructure and VNFs with workloads as close as possible to production in a test setup and then deploy with this in mind. In case of strict performance SLAs, it might be worth doing a static deployment, meaning obtain the profile in a test bed, and deploy based on that profile in production and not reconfigure the platform in terms of Intel® RDT again. However, if VM density is more important while the performance SLAs are more relaxed, it might be worth considering monitoring LLC utilization with CMT and in conjunction with some other metrics (like LLC misses, or measuring local network latency) provide a more "elastic" approach where the size of the COSes varies (but most probably not to be allowed to get bigger than the profile).

A Appendix

A.1 Hardware Components

Item	Description
Server Platform	Intel® Server Board S2600WTT, Formerly Wildcat Pass 2 x PCIe* 3.0 x16 slot, 1 x PCIe 3.0 x8 slot
Processor	Intel® Xeon® Processor E5-2658 v3 2.20 GHz 2 Sockets (NUMA Nodes), 12 cores/Socket, 12 threads, 2.2 GHz, 30 MB Last Level Cache
Memory	64 GB 1600 MHZ DDR3L ECC CL11 SODIMM 1.35V
BIOS	SE5C610.86B.01.01.0011.081020151200 Hyper-Threading technology: Disabled Intel® Virtualization Technology (Intel® VT-x): Enabled Intel® Virtualization Technology for Directed I/O (Intel® VT-d): Disabled CPU C-State: Disabled CPU P-State Control Enhanced Intel SpeedStep® Technology: Disabled Fan PWM Offset: 100 CPU Power and Performance Policy: Performance QPI/DMI: Auto Legacy USB Support: Disabled Port 60/64 Emulation: Disabled
Network Interfaces	1 x Intel® Ethernet X710-DA4 Adapter (Total: 4 Ports) http://ark.intel.com/products/83965/Intel-Ethernet-Converged-Network-Adapter-X710-DA4 Tested with Intel® FTLX8571D3BCV-IT transceivers
Local Storage	Intel® SSD DC S3500 Series Formerly Wolfsville SSDSC2BB120G4 120 GB SSD 2.5in SATA 6 Gb/s
Security options	UEFI Secure Boot

Table 22. Intel® Xeon® Processor E5-2658 v3 Platform Hardware Ingredients

B Appendix

B.1 Software Components

Table 23 describes functions of the software ingredients along with their version or configuration. For open source components, a specific commit ID set is used for this deployment. Note that the commit IDs detailed in the table are used as they are the latest working set at the time of this release

Software Component	Function	Version/Configuration
Fedora* 23	Host Operating System	Fedora 23 Server x86_64
KVM4NFV Kernel	Host Operating System Kernel	4.1.10-rt10, KVM4NFV kernel, Bramhaputra.1.0 tag
Fedora 20	Guest Operating System Kernel	Real Time kernel: 3.14.16-rt34-M_L_E_X2-VM
QEMU-KVM*	Virtualization technology	QEMU-KVM version: 2.3.1-7.fc22.x86_64 libvirt version: 1.2.13.1-3.fc22.x86_64
DPDK	Network stack bypass and libraries for packet processing; includes user space vhost drivers	DPDK Release 2.2.0 commit id: a38e5ec15e3fe615b94f3cc5edca5974dab325ab
Open vSwitch	vSwitch	OvS Release 2.5.0, commit id: at61c4e39460a7db3be7262a3b2af767a84167a9d8 Used for Open vSwitch 2.5 with DPDK
Cache Allocation Technology (CAT)/Cache Monitoring Technology (CMT)	Resource Director Technology components	intel-cmt-cat commit id: 1c473f93a2639f9d564ed7869bd1ef7a725bd513
Intel® Ethernet Drivers	Ethernet drivers	Driver Version: i40e 1.4.25 Firmware Version: 5.02 0x80002284 0.0.0

Table 23. Software Versions

C Appendix

C.1 Test Setup

C.1.1 Test Procedures

Benchmark methodology in both profiling and test runs:

- Three trials for each test case
- RFC2544 settings with 2000 bi-directional flows
- Maximum throughput at 0.1% packet loss
- 60 sec traffic duration for latency related measurements
- Average latency and latency bins with varying granularity levels were measured. For example: 5 µs-10 µs, 10 µs-25 µs, 25 µs-50 µs, 50 µs-75 µs, 75 µs-100 µs, 100 µs-150 µs, 150 µs-200 µs, 200 µs-250 µs, 250 µs-500 µs, 500 µs-750 µs, 750 µs-1000 µs, and 1000 µs-maximum
- L2 Forwarding in the VMs

Execute following test cases with above methodology and based on findings in profiling

- PHY-VM-PHY with Memtester in hypervisor
- PHY-VM-PHY with VM as a noisy neighbor
- PHY-VM-VM-PHY with VM as a noisy neighbor

C.1.2 Physical Setup

The physical setup included connecting the Ixia traffic generator with 10 GbE ports back to back with the compute node under test, as shown in Figure 24. IxNetwork* software was used to configure the test scenario with 2000 bi-directional flows. UDP packets of sizes 64B, 256B, 1024B, and 1512B were used for the test.

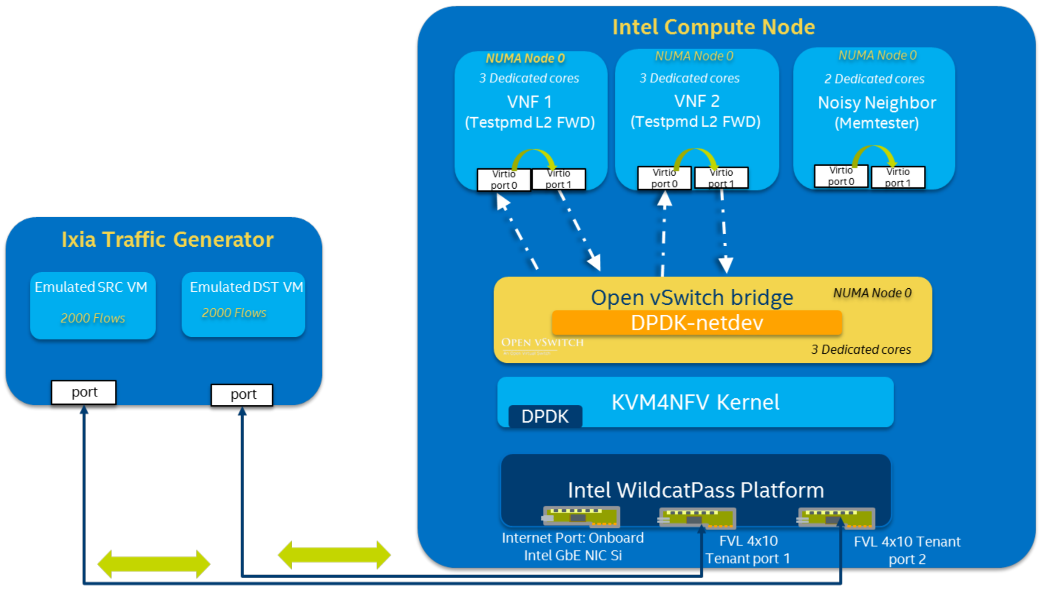

Figure 24. Example Setup of PHY-VM-VM-PHY with VM as Noisy Neighbor Case with Ixia* Traffic Generator

C.2 Software Setup

C.2.1 Hypervisor Setup

Follow the instructions below to configure the hypervisor that has Fedora23:

1. Download the KVM4NFV kernel from github and reset to Bramhaputra.1.0 release tag

```
$ cd /usr/src/kernels
$ git clone https://gerrit.opnfv.org/gerrit/p/kvmfornfv.git
$ cd kvmfornfv
$ git reset --hard branhmaputra.1.0
```

2. Compile the real time kernel using instructions from below and reboot:
   https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

3. Install all the required packages:

```
# dnf install -y gdb glibc-devel libtool make pkgconfig strace byacc ccache cscope ctags
elfutils  indent ltrace libffi-devel fuse-devel glusterfs bridge-utils ebtables libffi-devel
openssl-devel virt-install libvirt-daemon-config-network libvirt-daemon-kvm qemu-kvm virt-
manager virt-viewer libguestfs-tools virt-top python-libguestfs autoconf automake binutils
bison flex gcc-c++ gcc deltarpm libselinux-python psacct socat ntp libxslt-devel vim qemu
msr-tools
# dnf group install with-optional virtualization
```

4. Copy the existing kernel config file to real time kernel folder:

```
# cp /usr/src/kernel/ 4.2.3-300.fc23.x86_64/.config /usr/src/kernel/kvmfornfv/
# cd /usr/src/kernel/kvmfornfv
# make menuconfig
```

5. Enable the following:

   a. **Enable the high resolution timer:** `General Setup > Timers Subsystem > High Resolution Timer Support` (This option is selected by default.)

   b. **Enable the max number SMP:** Processor type and `features > Enable Maximum Number of SMP Processor and NUMA Nodes`

   c. **Enable PREEMPT-RT:** Processor type and `features > Preemption Model > Fully Pre-emptible Kernel (RT)`

   d. **Set the high-timer frequency:** Processor type and `features > Timer frequency > 1000 HZ` (This option is selected by default.)

   e. Exit and save.

6. Compile the kernel: `# make -j `grep -c processor /proc/cpuinfo` &&` make modules_install `&&` make install

7. Make changes to boot sequence, reboot and login to real time kernel:

```
# grep ^menuentry /boot/grub2/grub.cfg
# grub2-set-default "the desired default menu entry"
# grub2-editenv list
```

8. Use the following script to setup real time kernel variables:

   a. `#!/bin/bash`

   b. `#RT` Throttling is to be disabled

   c. echo -1 > `/proc/sys/kernel/sched_rt_runtime_us`

   d. echo -1 > `/proc/sys/kernel/sched_rt_period_us`

   e. #Interrupt binding is done to ensure no interrupts will be routed to isolated CPUs. This is achieved by pinning IRQs to cores note being used by vSwitch/vCPUs.

```
   for irq in /proc/irq/* ; do
       if [ -d ${irq} ] && ! grep - ${irq}/smp_affinity_list > /dev/null ; then
        al=`cat ${irq}/smp_affinity_list`
        if [[ ${cpu[*]} =~ ${al} ]] ; then
            echo 0 > ${irq}/smp_affinity_list
            cat ${irq}/smp_affinity_list
        fi
     fi
done

   #Minimise System stats collection
   echo 10 > /proc/sys/vm/stat_interval
   #Disable Softlockup detection
   echo 0 > /proc/sys/kernel/watchdog_thresh
   echo 0 > /proc/sys/kernel/watchdog
   echo 0 > /proc/sys/kernel/nmi_watchdog
```

```
# Choose the CPUs that are isolated in grub config
host _ isolcpus=1,2,3,4,5,6,7,8

i=0
for c in `echo $host _ isolcpus | sed 's/,/ /g'` ; do
    tid=`pgrep -a ksoftirq | grep "ksoftirqd/${c}$" | cut -d ' ' -f 1`
    echo "Chaning chrt value of 2 for ksoftirq tid" ${tid}
    chrt -fp 2 ${tid}
    tid=`pgrep -a rcuc | grep "rcuc/${c}$" | cut -d ' ' -f 1`
    echo "Changing chrt value of 3 for rcuc tid" ${tid}
    chrt -fp 3 ${tid}
    cpu[$i]=${c}
    i=`expr $i + 1`
done
# Change RT priority of rcub kernel threads
     for tid in `pgrep -a rcub | cut -d ' ' -f 1` ; do
         chrt -fp 3 ${tid}
         echo "changing chrt value of rcub for tid" ${tid}
     done
```

9. Stop and disable the interrupt requests (IRQ) balance, firewall, `iptables`, `SELinux*`, address space layout randomization and `IPv4` forwarding:

```
# killall irqbalance
# systemctl stop irqbalance.service
# systemctl disable irqbalance.service
# systemctl stop firewalld.service
# systemctl disable firewalld.service
# systemctl stop iptables.service
# sed -i 's/SELINUX=enabled/SELINUX=disabled/g' /etc/selinux/config
# echo "kernel.randomize _ va _ space=0" >> /etc/sysctl.d/aslr.conf
# echo "# Enable IPv4 Forwarding" > /etc/sysctl.d/ip _ forward.conf
# echo "net.ipv4.ip _ forward=0" >> /etc/sysctl.d/ip _ forward.conf
# systemctl restart systemd-sysctl.service
# echo "randomize _ va _ space value:"
# cat /proc/sys/kernel/randomize _ va _ space
# echo "ip _ forward value:"
# cat /proc/sys/net/ipv4/ip _ forward
```

10. Remove the following modules:

```
# rmmod ipmi _ si
# rmmod ipmi _ devintf
# rmmod ipmi _ ssif
# rmmod ipmi _ msghandler
# rmmod eventfd _ link
# rmmod ioeventfd
```

11. Set the kernel boot parameters with huge pages and isolate the required set of CPUs in the /etc/default/grub file:

```
GRUB _ CMDLINE _ LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv-fedora-server/swap default _
hugepagesz=1G hugepagesz=1G hugepages=16 hugepagesz=2M hugepages=2048 intel _ iommu=off
isolcpus=1-11 nohz _ full=1-11 rcu _ nocbs=1-11 rcu _ nocb _ poll=1 tsc=reliable idle=poll
irqaffinity=0 selinux=0 enforcing=0 rhgb quiet"
```

Note: The number of CPUs to be isolated will depend on the test case with the number of VMs under test. All the cores used in the report only use CPUs from Socket0. Also CPU0 is not used in the list of isolated cores so OS can use it. Purpose of the real time options used are detailed below:

```
nohz _ full=<…>; Mark cpus as adaptive-ticks cpus
rcu _ nocbs=<…>; Mark cpus to offload RCU callbacks to kthreads
rcu _ nocb _ poll=1; Kthreads perform polling so offloaded CPUs don't need to do wakeups.
tsc=reliable; Disable clocksource stability checks for TSC
idle=poll; Turn off C States
irqaffinity=0; Assuming core 0 is an OS core and not being used by the vSwitch/vCPUs
Selinux=0 enforcing=0; Disable SE Linux to improve performance
```

The range indicated by <…> is the comma-separated list of CPUs being used in the particular test by both the vSwitch and the vCPUs in the guest.

12. Save the file and update the GRUB config file:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

13. Reboot the host machine and check to make sure 1GB and 2MB hugepage sizes are created:

```
# ls /sys/devices/system/node/node0/hugepages/hugepages-*
```

14. Mount 1-GB and 2-MB hugepages and verify:

```
# mkdir -p /mnt/huge
# mkdir -p /mnt/huge _ 2mb
# mount -t hugetlbfs nodev /mnt/huge
# mount -t hugetlbfs nodev /mnt/huge _ 2mb -o pagesize=2MB
# mount
```

C.2.2 DPDK and OvS Setup

Instructions below will help to setup DPDK and OvS based on the Intel test case:

1. Download and compile DPDK with following configuration:

```
# git clone  http://dpdk.org/git/dpdk
# cd dpdk
# git reset --hard a38e5ec15e3fe615b94f3cc5edca5974dab325ab
# make install T=x86 _ 64-native-linuxapp-gcc
# cd x86 _ 64-native-linuxapp-gcc
# sed -i 's/CONFIG _ RTE _ BUILD _ COMBINE _ LIBS=n/CONFIG _ RTE _ BUILD _ COMBINE _ LIBS=y/g'
.config
# sed -i 's/CONFIG _ RTE _ LIBRTE _ VHOST=n/CONFIG _ RTE _ LIBRTE _ VHOST=y/g' .config
# sed -i 's/CONFIG _ RTE _ LIBRTE _ VHOST _ USER=n/CONFIG _ RTE _ LIBRTE _ VHOST _ USER=y/g' .config
# make -j20 -Ofast
# export $DPDK _ BUILD=/root/dpdk/x86 _ 64-native-linuxapp-gcc/
```

2. Insert the required modules:

```
# modprobe msr
# modprobe uio
# insmod /$DPDK _ BUILD/kmod/igb _ uio.ko
```

3. Check the PCI ID for the 10-GbE NIC ports, for example:

```
# lspci | grep Ethernet
06:00.0 Ethernet controller: Intel Corporation Ethernet
Controller X710 for 10GbE SFP+ (rev 01)
06:00.1 Ethernet controller: Intel Corporation Ethernet
Controller X710 for 10GbE SFP+ (rev 01)
06:00.2 Ethernet controller: Intel Corporation Ethernet
Controller X710 for 10GbE SFP+ (rev 01)
06:00.3 Ethernet controller: Intel Corporation Ethernet
Controller X710 for 10GbE SFP+ (rev 01)
```

4. Bind the two ports connected to traffic generator to DPDK.

```
# cd $DPDK_BUILD
# python tools/dpdk_nic_bind.py --bind=igb_uio 0000:06:00.0
# python tools/dpdk_nic_bind.py --bind=igb_uio 0000:06:00.1
# python tools/dpdk_nic_bind.py --status
```

5. Download, compile and start OvS.

```
# rm -rf /usr/local/var/run/openvswitch
# rm -rf /usr/local/etc/openvswitch/
# rm -f /tmp/conf.db
# mkdir -p /usr/local/etc/openvswitch
# mkdir -p /usr/local/var/run/openvswitch
# git clone https://github.com/openvswitch/ovs.git
# cd ovs
# git reset --hard 61c4e39460a7db3be7262a3b2af767a84167a9d8
# cd /root/ovs/
# ./boot.sh
# ./configure --with-dpdk=/root/dpdk/x86_64-native-linuxapp-gcc
# make 'CFLAGS=-Ofast -march=native'
# ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db ./vswitchd/vswitch.ovsschema
# ./ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock
--remote=db:Open_vSwitch,Open_vSwitch,manager_options --pidfile --detach
# ./utilities/ovs-vsctl --no-wait init
```

6. Set real time priority for `ovs-vswitchd` process with `SCHED_FIFO` as policy. Start `vSwitchd` process on CPU1, hex equivalent being `0x2`.

```
# chrt -f 95 ./vswitchd/ovs-vswitchd --dpdk -c 0x2 -n 4 --socket-mem 2048,0 --
unix:/usr/local/var/run/openvswitch/db.sock –pidfile
```

Note: The value for "–c" option is specific to platform and should be used based on CPU enumeration

7. Tune the OvS to set 2 PMD cores, on CPU2 and CPU3 using `pmd-cpu-mask` option:

```
# cd ovs/
# ./utilities/ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=C
#./utilities/ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

8. Add the bridges and required ports.
   a. For PHY-VM-PHY case:

   ```
   # cd ovs/
   #./utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
   ```

```
# ./utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
# ./utilities/ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
# ./utilities/ovs-vsctl add-port br0 vhost-user0 -- set
Interface vhost-user0 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user1 -- set
Interface vhost-user1 type=dpdkvhostuser
# ./utilities/ovs-vsctl show
```

b. For PHY-VM-VM-PHY case:

```
# cd ovs/
# ./utilities/ovs-vsctl add-br br0 -- set bridge br0
datapath _ type=netdev
# ./utilities/ovs-vsctl add-port br0 dpdk0 -- set
Interface dpdk0 type=dpdk
# ./utilities/ovs-vsctl add-port br0 dpdk1 -- set
Interface dpdk1 type=dpdk
# ./utilities/ovs-vsctl add-port br0 vhost-user0 -- set
Interface vhost-user0 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user1 -- set
Interface vhost-user1 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user2 -- set
Interface vhost-user2 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user3 -- set
Interface vhost-user3 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user4 -- set
Interface vhost-user4 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user5 -- set
Interface vhost-user5 type=dpdkvhostuser
# ./utilities/ovs-vsctl show
```

9. Add the openflow rules based on the test case:

a. For PHY-VM-PHY case:

```
# cd ovs/
# ./utilities/ovs-ofctl add-flow br0
in _ port=1,dl _ type=0x800,idle _ timeout=0,action=output:3
# ./utilities/ovs-ofctl add-flow br0
in _ port=2,dl _ type=0x800,idle _ timeout=0,action=output:4
# ./utilities/ovs-ofctl add-flow br0
in _ port=3,dl _ type=0x800,idle _ timeout=0,action=output:1
# ./utilities/ovs-ofctl add-flow br0
in _ port=4,dl _ type=0x800,idle _ timeout=0,action=output:2
# ./utilities/ovs-ofctl dump-flows br0
```

a. For PHY-VM-PHY case:

```
# cd ovs/
# ./utilities/ovs-ofctl add-flow br0
in _ port=1,dl _ type=0x800,idle _ timeout=0,action=output:3
# ./utilities/ovs-ofctl add-flow br0
in _ port=3,dl _ type=0x800,idle _ timeout=0,action=output:1
# ./utilities/ovs-ofctl add-flow br0
in _ port=4,dl _ type=0x800,idle _ timeout=0,action=output:5
# ./utilities/ovs-ofctl add-flow br0
in _ port=5,dl _ type=0x800,idle _ timeout=0,action=output:4
```

```
# ./utilities/ovs-ofctl add-flow br0
in _ port=2,dl _ type=0x800,idle _ timeout=0,action=output:6

# ./utilities/ovs-ofctl add-flow br0
in _ port=6,dl _ type=0x800,idle _ timeout=0,action=output:2

# ./utilities/ovs-ofctl dump-flows br0
```

C.2.3 PQoS Setup

Basic details of the PQoS tool and its usage are provided in Section 2.4.3, The intel-cmt-cat Software Package. The class of service configurations used in the report are detailed out in this section. PQoS tools uses a .cfg file to allocated cache based on user specified class of service configuration. Examples configuration files are found under "configs" directory in the pqos package. Following configurations were used based on their corresponding test cases:

Configuration used for Overlapping case 1 24-6-6-6:

```
alloc-class-set: llc:0=0xffff0;llc:1=0x00f00;llc:2=0x000f0;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=5-7
alloc-assoc-set: llc:2=8-10
alloc-assoc-set: llc:3=0,4,11
```

Configuration used for Isolated case 1 24-6-6-6:

```
alloc-class-set: llc:0=0xff000;llc:1=0x00f00;llc:2=0x000f0;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=4-6
alloc-assoc-set: llc:2=7-9
alloc-assoc-set: llc:3=0,10,11
```

Configuration used for Overlapping case 2 24-3-3-6:

```
alloc-class-set: llc:0=0xffff0;llc:1=0x000C0;llc:2=0x00030;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=4-6
alloc-assoc-set: llc:2=7-9
alloc-assoc-set: llc:3=0,10,11
```

Configuration used for Isolated case 2 24-3-3-6:

```
alloc-class-set: llc:0=0xfff00;llc:1=0x000C0;llc:2=0x00030;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=4-6
alloc-assoc-set: llc:2=7-9
alloc-assoc-set: llc:3=0,10,11
```

Configuration used for Phy-VM-Phy with noisy neighbor in the hypervisor:

```
alloc-class-set: llc:0=0xffff0;llc:1=0xff000;llc:2=0x00ff0;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=5-7
alloc-assoc-set: llc:2=8-10
alloc-assoc-set: llc:3=0,4,11
```

Configuration used for Phy-VM-Phy with noisy neighbor VM:

```
alloc-class-set: llc:0=0xffff0;llc:1=0x000C0;llc:2=0x00030;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=4-6
```

```
alloc-assoc-set: llc:2=7-9
alloc-assoc-set: llc:3=0,10,11
```

Configuration used for Phy-VM-VM-Phy with noisy neighbor VM:

```
alloc-class-set: llc:0=0xffff0;llc:1=0x000C0;llc:2=0x00030;llc:3=0x0000f
alloc-assoc-set: llc:0=1-3
alloc-assoc-set: llc:1=4-6
alloc-assoc-set: llc:2=7-9
alloc-assoc-set: llc:3=0,10,11
```

C.2.4 Guest (VM Under Test) Setup

The guest is configured with following options:

- Assign 3 vCPUs to the guest, two for the guest application (test-pmd) and one for the OS
- Affinitise the threads of test-pmd with taskset, ensuring the cores assigned to the application are those isolated in the kernel boot parameters
- Assign a scheduling policy and high priority to the test-pmd process and threads.

An example of related options of VMs under test booted with its corresponding real time priority is given below:

```
# $HUGE _ DIR=/dev/hugepages
# $SOCK _ DIR=/usr/local/var/run/openvswitch
# $PORT0 _ NAME=vhost-user0
# $PORT1 _ NAME=vhost-user1
# chrt -f 5 taskset 0xE0 qemu-system-x86 _ 64 -name us-vhost-vm1 -cpu host -enable-kvm -m
4096 -vnc :12 -object memory-backend-file,id=mem,size=4096M,mem-path=$HUGE _ DIR,share=on
-numa node,memdev=mem -mem-prealloc -smp 3 -drive file=/root/RT _ FC20.qcow2 -chardev
socket,id=char0,path=$SOCK _ DIR/$PORT0 _ NAME -netdev type=vhost-user,id=mynet1,chardev=char0
,vhostforce -device virtio-net-pci,mac=00:00:00:00:00:01,netdev=mynet1,mrg _ rxbuf=off -chardev
socket,id=char1,path=$SOCK _ DIR/$PORT1 _ NAME -netdev type=vhost-user,id=mynet2,chardev=ch
ar1,vhostforce -device virtio-net-pci,mac=00:00:00:00:00:02,netdev=mynet2,mrg _ rxbuf=off –
nographic
```

The CPU affinity values for each of the VMs are set using the taskset option and the values can be found in the figures detailed in earlier sections corresponding to the test case.

After booting into the VM install and boot into real time kernel 3.14.16-rt34-M_L_E_X2-VM. Then follow the steps detailed in Sections 12.4, 12.5, 12.6, 12.7 and 12.8 to bootstrap the VM and deploy DPDK from the ONP2.0 performance test report available at: https://download.01.org/packet-processing/ONPS2.0/Intel_ONP_Release_2.0_Performance_Test_Report_Rev1.0-1.pdf.

Then tune the Qemu processes of the VM under test using the process provided in Section10.5 of the Intel® ONP2.0 Performance Test Report.

Note: The number of cores for the VMs under test have always been three cores per VM.

C.3 Noisy Neighbor Setup

C.3.1 Hypervisor as Noisy Neighbor

Memtester application was used as a hypervisor application to act as a noisy neighbor. Either insert the memtester module or download and build the memtester package from http://linux.softpedia.com/get/Utilities/memtester-27174.shtml.

Use the memtester application to access 100 MB of memory using:

```
# ./memtester 100M > /dev/tmp
```

C.3.2 VM as a Noisy Neighbor

The noisy neighbor VM is booted with similar options detailed in Appendix C.3.4. Either insert the memtester module or download and build the memtester package from http://linux.softpedia.com/get/Utilities/memtester-27174.shtml.

1. Use the memtester application to access 100 MB of memory using:

```
   # ./memtester 100M > /dev/tmp
```

2. Then tune the Qemu processes of the VM under test using the process provided in Section10.5 of the ONP 2.0 Performance Test Report.

Note: The number of cores used for the noisy neighbor VM were dependent on the test case:

- In PHY-VM-PHY case: 3 cores
- In PHY-VM-VM-PHY case: 2 cores

D Acronyms and Abbreviations

D.1 Acronyms and Abbreviations

Term	Description
BIOS	Basic Input/Output System
CAT	Cache Allocation Technology
CDP	Code and Data Prioritization
CLI	Command Line Interface
CMT	Cache Monitoring Technology
COS	Class of Service
COTS	Commercial off-the-shelf
DoS	Denial Of Service
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
EMC	Extra Match Cache
EOI	End of Interrupt
IDS	Intrusion Detection Systems
Intel® ONP	Intel® Open Network Platform
IPS	Intrusion Prevention Systems
KVM	Kernel-Based Virtual Machine
LLC	Last Level Cache
MBM	Memory Bandwidth Management
NAPI	New API
NFV	Network Functions Virtualization
NN	Noisy Neighbor
NUMA	Uniform Memory Access
OS	Operating System
OvS	OpenVSwitch
PMD	Poll Mode Driver
PQoS	Platform Quality of Service
QEMU	Quick EMUlator
QoS	Quality of Service
RCU	Read-Copy-Update
Intel® RDT	Intel® Resource Director Technology
RMID	Resource Monitoring ID

SDN	Software-Defined Networking
SLA	Service Level Agreement
SMI	System Management Interrupt
TSC	Time Stamp Counter
UIO	I/O
VFIO*	Virtual Function I/O*
VIM	Virtualized Infrastructure Manager
VNFs	Virtual Network Functions
VM	Virtual Machine
VMware ESXi*	VMware vSphere* Hypervisor
VMM	Virtual Machine Monitor

E References

E.1 References

Reference	Location
Intel® 64 and IA-32 Architectures Software Developer's Manuals	http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html v055, Vol 3b. Chapter 17.15 and 17.16, covers CMT, CAT, MBM and CDP
Intel, Cache Monitoring and Cache Allocation Technologies landing page	http://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html
CMT, MBM, CAT and CDP Public Software Library/Utility	https://01.org/packet-processing/cache-monitoring-technology-memory-bandwidth-monitoring-cache-allocation-technology-code-and-data
CMT, MBM, CAT and CDP Public Software Library/Utility GitHub Project	https://github.com/01org/intel-cmt-cat
Intel, "Enabling NFV to Deliver on its Promise"	http://www.intel.com/content/www/us/en/communications/nfv-packet-processing-brief.html
CAT cgroup Kernel Patches	http://marc.info/?l=linux-kernel&m=142620227328406&w=2
Christos Kozyrakis et al, "Heracles: Improving Resource Efficiency at Scale"	http://csl.stanford.edu/~christos/publications/2015.heracles.isca.pdf
Introduction to CMT Blog	https://software.intel.com/en-us/blogs/2014/06/18/benefit-of-cache-monitoring
Discussion of RMIDs and CMT Software Interfaces Blog	https://software.intel.com/en-us/blogs/2014/12/11/intel-s-cache-monitoring-technology-software-visible-interfaces
Use Models and Example Data using CMT Blog	https://software.intel.com/en-us/blogs/2014/12/11/intels-cache-monitoring-technology-use-models-and-data
Software Supports and Tools: Intel's CMT: Software Support and Tools	https://software.intel.com/en-us/blogs/2014/12/11/intels-cache-monitoring-technology-software-support-and-tools
Intel Platform Shared Resource Monitoring and CAT	http://smackerelofopinion.blogspot.com/2015/11/intel-platform-shared-resource.html
Intel, "Increasing Platform Determinism with Platform Quality of Service for the Data Plane Development Kit"	http://www.intel.com/content/www/us/en/communications/increasing-platform-determinism-pqos-dpdk-white-paper.html

Intel® ONP 2.0 Performance Test Report	https://download.01.org/packet-processing/ONPS2.0/Intel_ONP_Release_2.0_Performance_Test_Report_Rev1.0-1.pdf
Intel ONP 2.0 Reference Architecture Guide	https://download.01.org/packet-processing/ONPS2.0/Intel_ONP_Release_2.0_Reference_Architecture_Guide_Rev1.1.pdf
Intel Open Network Platform Program	https://01.org/packet-processing/intel%C2%AE-onp
Intel® ONP 2.1 Application Note on Intel® RDT	https://download.01.org/packet-processing/ONPS2.1/Intel_ONP_Release_2.1_Application_Note_on_RDT_Rev1.0.pdf
OpenStack*	http://www.openstack.org/

Figures

Tables