Jumbo frames Performance Analysis Over Red Hat OpenStack 16.1 Infrastructure

July, 2021

Whitepaper

Abstract

This whitepaper describes how to manage Jumbo frames throughput for Zero Packet Loss (ZPL) using Intel® Ethernet Network Adapters XXV710 over Dell EMC Ready Architecture Red Hat OpenStack infrastructure.

Dell Technologies Solutions

Copyright

Contents

List of Figures

List of Tables

List of Acronyms/Abbreviations

Acronym	Definition
CSP	Communication Service Provider
NFV	Network Function Virtualization
OVS	Open virtual Switch
PMD	Poll Mode Driver
RA	Ready Architecture
RSS	Receive Side Scaling
RHOSP	Red Hat OpenStack Platform
SDN	Software Defined Networking
SUT	System Under Test
ZPL	Zero Packet Loss

Introduction

Software-Defined Network (SDN) and network function virtualization (NFV) have been driving the network transformation for many years, resulting in the majority of networking solutions virtualized. There are still challenges in getting optimum performance for services running on underlying virtual infrastructures.

Communications Service Providers (CSPs) support many services, for example, broadband access services, Voice over IP (VoIP) services, call center, IPTV and cable services. One of the most important measurements is the network capability to support Zero Packet Loss (ZPL). Any packet loss means disruption to services, especially for emergency call centers. For example, a 911 call drop is unacceptable. Traditionally, fixed network appliances can deliver the performance needed, and they do so with assurance. With SDN and NFV transformation, the Zero-Packet Loss capability to show expected performance over the infrastructure becomes extremely important.

Various laboratory measurement observations are not all published. Part of the problem is knowing whether the measured performance is, in fact, the expected performance. Typically, the DPDK open-source community published performance data for Physical Function and VirtIO for the Network Interface[1]. This is a great reference point to baseline expected performance. However, the DPDK open-source community doesn't cover the test cases for jumbo packet size[2].

This collaborative whitepaper between Dell and Intel aims to provide guidance on jumbo packet size for Zero Packet Loss (ZPL) test cases over Dell EMC Ready Architecture (RA) Red Hat OpenStack Infrastructure. In addition, Red Hat engineering team also helped in finalizing result section and verified the best practices applied in this activity. The key learnings and best-known practices documented in this paper can help your testing and analysis.

Problem statement

The network throughput testing measured in Red Hat OpenStack Platform (RHOSP) 16.1 revealed consistent packet drop for jumbo frames across DPDK physical NICs. Packet drops were observed ranging 5K - 38K for different jumbo frames sizes that is, 4096, 8192 and 9000 bytes[3]. The occurrence of packet drops limited the ZPL to converge at 99.2% line rate. The network performance testing was carried out on RHOSP 16.1 with OVS-DPDK[4] feature enabled, providing fast packet processing capability for network intensive workloads. Synthetic network load was generated using Spirent's proprietary network testing tool, Spirent MethodologyCenter. Unidirectional traffic was sent from the

---

[1] https://core.dpdk.org/perf-reports/

[2] **Jumbo frame(s)/Jumbo packet size(s)**: Ethernet frames with more than 1500 bytes payload are called jumbo frames. They are to be enabled on both hardware (switches, routers) and software levels to ensure network devices can process larger packet sizes without fragmentation.

[3] The network performance is evaluated using RFC2544 benchmarking methodology. It measures the throughput for SUT. MethodologyCenter calculates the difference between transmitted and received packets to measure how much packets have dropped.

[4] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/network_functions_virtualization_planning_and_configuration_guide/assembly_OVSdpdk_parameters

provisioned VM on one compute node as traffic generator to the traffic receiver VM on another node. Refer to *Appendix A* for hardware and software details. The Figure 1 depicts the scenario mentioned above and the marked red cross indicates packet drop observed during testing.

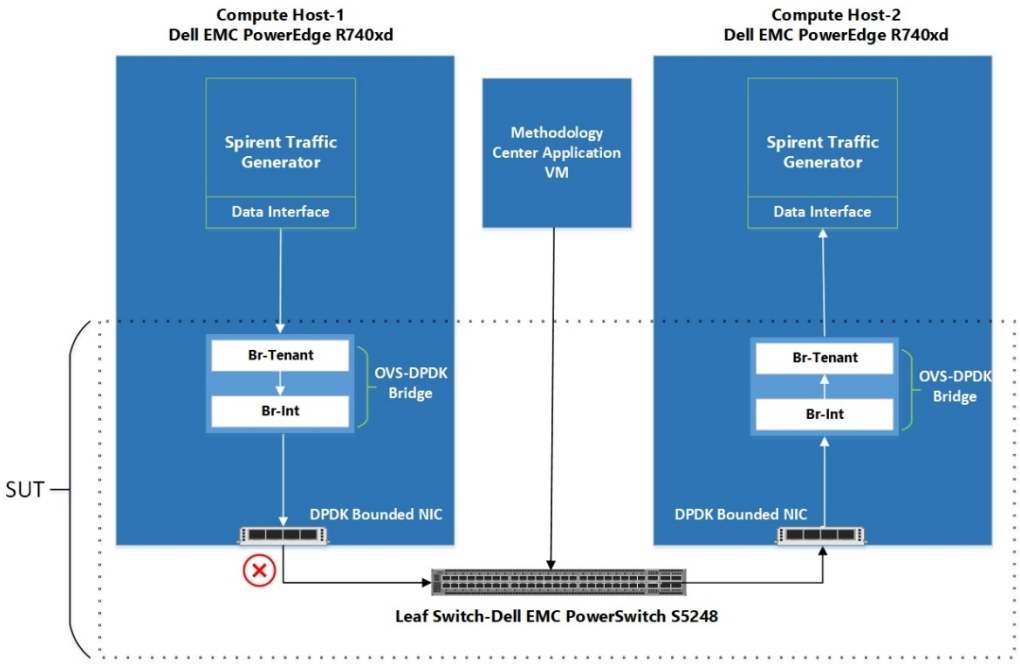

Figure 1. Test environment highlighted with the packet drop area

Various methods were employed to eliminate the packet drops. Different parameters were incrementally tweaked to help eliminate the packet drop with enhanced network throughput. This guide highlights the best practices to consider for better throughput targeting for jumbo frames use-cases.

Troubleshooting

To troubleshoot and analyze the reasons for packet loss in Dell EMC RA based RHOSP 16.1, standalone KVM based environment was setup with intent to replicate the *Problem statement*. In isolated setup, we replicated similar software versions and configurations supported by Red Hat OpenStack 16.1.

The debug analysis was done in collaboration with Intel covering multiple test cases that were run incrementally in a standalone server setup to reproduce the issue for packet loss.

Refer to *Appendix B* for the test results of the aforementioned test cases.

The variations implemented in test cases include use of single and dual OVS bridges, modifying TX and RX queues, bonding on NICs and VLAN transitions. Isolated and combinations of these parameters were incrementally applied. Each parameter played a vital role in minimizing the packet drops but scaling the RX queues was the major optimization that eliminated the packet drops for jumbo frames.

The following section highlights the test results obtained on Dell EMC PowerEdge R740xd in RHOSP 16.1 before and after scaling the RX queues. Implementing the findings obtained from debug analysis, RX queues were scaled in the OpenStack environment and 100% line rate was achieved. The RX queues were scaled on both physical and virtual ports of traffic sender and receiver nodes. Details for the scaling of RX queues and their impact on network throughput are given below.

Figure 2 highlights the virtual and physical ports where multiple RX queues are enabled in an OpenStack environment.

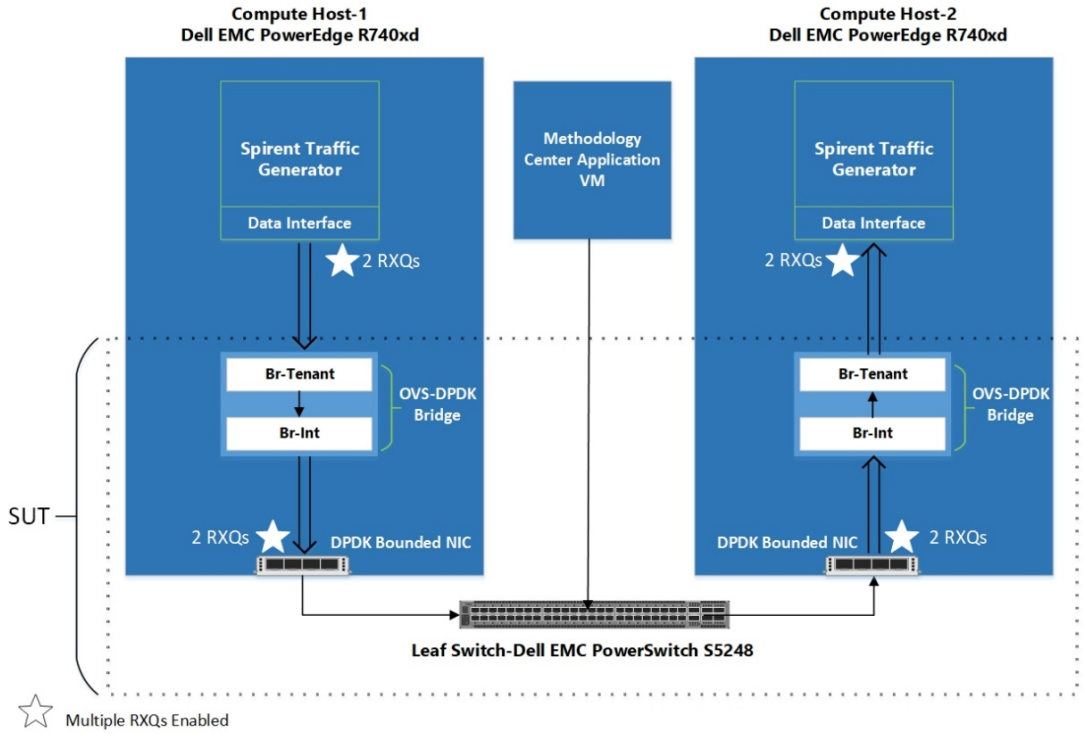

Figure 2. Scaled RX queues on Dell Technologies OpenStack Solution

The detailed implementation of scaling the RX queues has been discussed in *Appendix C*.

Table 1 shows the benchmarked throughput results for jumbo frames with single and multiple RXQs in RHOSP 16.1. In addition to scaled RX queues, other optimization considerations are mentioned *in Learnings and Best-Known Practices.*

Table 1. Line rate % and PPS performance comparison for jumbo frames

Packet Size (B)	1 RXQ		2 RXQs	
	Line Rate (%)	Packet Per Second	Line Rate (%)	Packet Per Second
4096	99.23	758570	100	759828
8129	99.22	380206	100	380845
9000	99.23	346724	100	346953

The following figure is a graphical representation of the results in Table 1.

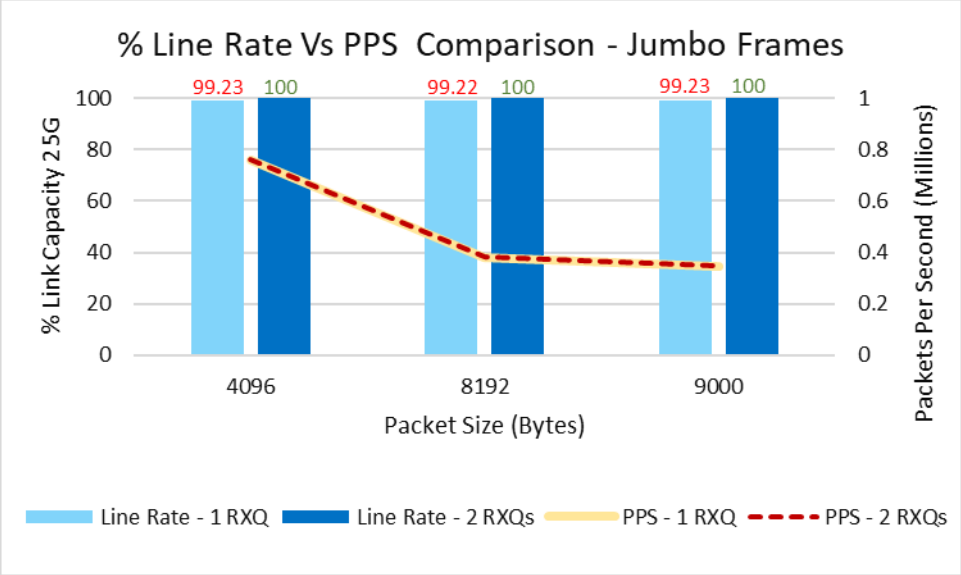

Figure 3. Line rate % vs PPS comparison results for jumbo frames

Note: The recommended configurations are suggested based on 25 GbE Intel® Ethernet Network Adapter XXV710-DA2 results. Higher speed NICs would require additional parameters to consider for best performance numbers for jumbo frames.

Learnings and Best-Known Practices

This section highlights a set of key observations in addition to multiple RX queues regarding best-known practices when handling jumbo packets for a Red Hat OpenStack 16.1 environment.

Red Hat OpenStack template recommendation for jumbo packets

The Red Hat OpenStack (RHOSP) deployment for an OVS-DPDK network environment should address the following factors for efficient network throughput:

- **Hypervisor level NUMA alignment:** The relationship between the NIC and NUMA socket.
- **NUMA balanced architecture:** The compute node should have an equal, symmetrical set of resources (CPUs, memory, NICs) allocated to each socket. For the discussed scenario in the paper, setup was dual socket.
- **Guest level NUMA alignment:** The VM profile should align memory (hugepages), vCPUs, and vNICs locally to the appropriate NUMA node to avoid cross UPI traffic.
- **DPDK socket memory:** Should be allocated, ideally, symmetrically across sockets, and calculated based on both the number of DPDK ports and the MTU size handling the data traffic.
- **DPDK memory channels:** To align with the number of physical memory channels available for each socket.
- **Multiple RX queues on DPDK physical ports:** To avoid potential packet drops when scaling up to handle multiple simultaneous flows.
- **Hugepage support:** The VM profile should include hugepages for backing DPDK packet processing.
- **DPDK TX and RX descriptor size:** Increases the TX/RX buffer of the NIC reducing packet loss.
- **Emulator thread policy:** Emulator threads should run on a set of vCPUs independent from the threads dedicated to the guest VM.
- **Multiple RX queues on VirtIO ports:** The VM profile should include multi-queue support and align with the number of RX/TX queues assigned to the DPDK physical ports.
- **Isolcpus[5]:** PMD threads and vCPUs for the guest VM should reside within the isolcpus, rcu_nocbs, and nohz_full lists to avoid potential context switches due to kernel thread interrupts.

Refer to *Appendix C* for further implementation details.

Summary and conclusion

A good understanding of the underlying infrastructure for any cloud deployment is critical to learning how its network(s) work to identify and eliminate possible bottlenecks for reliant and optimal performance. The objective of this paper was to identify and eliminate the causes of packet drops for jumbo frames. A set of key considerations that enhances performance for jumbo frame are highlighted as best practices. The tunings applied through various parameters resulted in ZPL throughput to reach 100% line rate. The optimizations are recommended to be accounted for to get maximum network performance.

[5] The IsolCPUs are list of CPUs which are isolated from host processes.

Generally, standard Ethernet frames are used for network traffic over the cloud environments. However, in performance specific use-cases jumbo frames are required to enhance payload capacity by reducing the overheads and CPU cycles for packet processing, thus efficiently utilizing the given bandwidth channel.

The occurrence of packet drops for jumbo frames in the RHOSP 16.1 couldn't achieve ZPL at 100% line rate. The issue was reproduced in a standalone KVM based environment with different variations in test environment. The variations such as Tx and Rx queues, single OVS bridge, dual OVS bridges, VLAN transitions and bonding were implemented both in isolation and combinations with other parameters. Every variation improved performance but scaling the Rx queues was major optimization that eliminated packet drops increasing ZPL throughput to reach 100% line rate.

This paper summarizes key learnings that took a lot of time and effort and we hope that this paper is helpful for tuning the network performance of your infrastructure. Please refer to *Appendix C* for details of implementation with respect to manual and JetPack[6] powered OpenStack deployment methods separately for general performance practice undertaken for jumbo frames. We encourage you to implement the list of best practices presented here to get the maximum performance for jumbo frames.

Lastly, this paper confirmed that recommendations for best practices from Red Hat are vital to achieve optimal performance for all packet sizes including jumbo frames over OpenStack environment.

[6] **JetPack**: Dell Technologies powered JetPack automation toolkit is an open-source offering, which is an innovative automation package that is used to configure and deploy the Ready Architecture Dell Technologies infrastructure hardware with Red Hat OpenStack in a fully automated fashion. For further details, please visit:

https://www.delltechnologies.com/en-au/solutions/cloud/openstack/ready-bundle-for-openstack.html

https://github.com/dsp-jetpack/JetPack/tree/JS-16.1

Appendix A

Following are the details for hardware and software specifications for Red Hat OpenStack Platform infrastructure consisting of Dell EMC PowerEdge R740xd servers.

Compute Node specification

Table 2. Compute Node specification

Component	Specification
Platform	Dell EMC PowerEdge R740xd
CPU	Intel® Xeon® Gold 6140 CPU @ 2.30GHz (18C per socket)
CPU microcode	0x5003003
RAM	192GB - 12 x 16GB Dual Rank DDR4 2666 MT/s
Bonded NICs	4 x Intel® Ethernet Network Adapters XXV710-DA2 (25GbE)

Note: The issue was observed by Dell during testing in November 2020 on RHOSP 16.1 test setup and resolved later in April 2021 in collaboration with Intel.

Software specifications

Table 3. Software specifications

Parameter	Specification
Kernel version	4.18.0-193.14.3.el8_2.x86_64
PMD cores	1 PMD core used (8 PMD cores/compute node)
DPDK version	19.11.1
Libvirt version	6.0.0
Hypervisor QEMU	4.2.0
NIC firmware version	driver: i40e version: 2.8.20-k firmware-version: 7.10 0x800075e6 19.5.12
BIOS version	2.8.2
OpenStack Platform	16.1
Operating System	Red Hat Enterprise Linux (RHEL) 8.2

Other configurations

Table 4. Other configurations

Parameter	Configuration
DPDK socket memory	8192
Grub command line parameters	BOOT_IMAGE=(hd0,gpt3)/boot/vmlinuz-4.18.0-193.14.3.el8_2.x86_64 root=UUID=99a3522b-0b91-43d5-9c37-47911cb1682b ro console=ttyS0 console=ttyS0,115200n81 no_timer_check crashkernel=auto rhgb quiet iommu=pt intel_iommu=on intel_pstate=disable nosoftlockup default_hugepagesz=1GB hugepagesz=1G hugepages=164.0 skew_tick=1 nohz=on nohz_full=2-35,38-71 rcu_nocbs=2-35,38-71 isolcpus=2-35,38-71
Spirent VM flavor	4 vCPUs, 4096 MB RAM, 20 GB HDD

Note: Node restart is required after updating the grub command line parameters.

Appendix B

This section highlights the troubleshooting steps carried out in the Standalone environment in collaboration with Intel.

Table 5 represents the results for the test cases mentioned in *Troubleshooting* section. The details of the test cases are given below.

Table 5. RFC2544 ZPL benchmark results

Test Case ID	Test Case	VLAN Translation	Bonding	RX Qs, TX Qs	Line Rate (%) / PPS for 4096 Byte Packets	Line Rate (%) / PPS for 8192 Byte Packets	Line Rate (%) / PPS for 9000 Byte Packets
1	Baremetal PF	#N/A	Disabled	2	100 / 759,231	100 / 380,540	100 / 346,452
2	VM PF	#N/A	Disabled	2	100 / 759,231	100 / 380,540	100 / 346,452
3	OVS-DPDK Single Bridge	Disabled	Disabled	2	100 / 759,230	100 / 380,539	100 / 346,451
4	OVS-DPDK Dual Bridge	Disabled	Disabled	2	100 / 759,228	100 / 380,539	100 / 346,451
5	OVS-DPDK Dual Bridge	Enabled	Disabled	1	86.852 / 659,404	99.227 / 377,596	100 / 346,450
6	OVS-DPDK Dual Bridge	Enabled	Enabled	1	96.906 / 735,740	100 / 380,540	100 / 346,450
7	OVS-DPDK Dual Bridge	Enabled	Disabled	2	100 / 759,224	100 / 380,539	100 / 346,451
8	OVS-DPDK Dual Bridge	Enabled	Enabled	2	100 / 759,232	100 / 380,537	100 / 346,452

Note: Unless otherwise stated, all configurations use DPDK 19.11.6 on the host/guest, with OVS 2.13.3, along with Intel® Ethernet Network Adapters XXV710-DA2, testpmd, with a 4K descriptor size, benchmarked with RFC2544 with Zero Packet Loss (ZPL).

The following test cases were run in the isolated KVM based environment (listed in order of priority) with increasingly complexity step by step to reproduce the issue.

1. Bare Metal Physical Function (PF).
2. Single Root IO Virtualization (SRIOV).
3. VirtIO with OVS and DPDK with single bridge topology.
4. VirtIO with OVS and DPDK with dual bridge topology, VLAN translation disabled.
5. VirtIO with OVS and DPDK with single queue, dual bridge topology, VLAN translation enabled.
6. VirtIO with OVS and DPDK with single queue, dual bridge topology, VLAN translation enabled with bonding.

The following run test cases define the impact of single queue for transmit and receive:

7. VirtIO with OVS and DPDK with dual bridge topology, VLAN translation enabled.

8. VirtIO with OVS and DPDK with dual bridge topology, VLAN translation enabled with bonding.

By implementing different optimizations, line rate network throughput was ensured for each test case with increasing level of virtual overhead. Test cases 1-4 with minimal infrastructure complexity reported no packet loss.

Identical test setup was simulated in test cases 5-8 with the OpenStack environment. Configuring single RX queue resulted in packet loss for all jumbo frames. However, re-running the same test setup with multiple RX queues, i.e. test case 7 and 8 eliminated the packet loss and 100% line-rate throughput was achieved.

The following figure represents isolated test setup used in test cases 5-8.

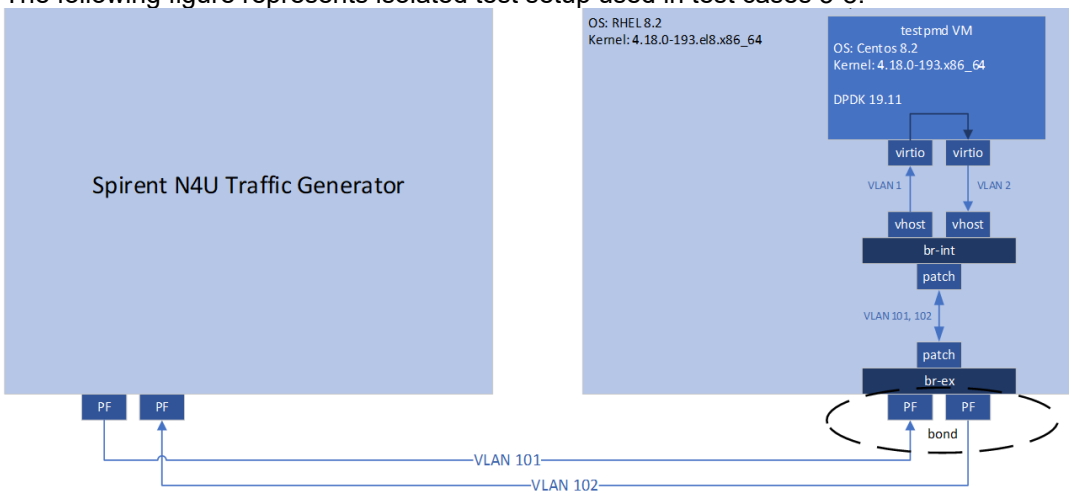

Figure 4. Isolated test setup replicating OpenStack environment

Appendix C

Implementation of best practices

This section provides insights on implementing best practices concerning Dell EMC powered JetPack OpenStack and manual Red Hat OpenStack Platform deployments.

Red Hat OpenStack Platform can either be deployed manually or via automation scripts provided under JetPack automation toolkit. Manual deployment is typical but time-consuming. Installation and configuration need to be performed manually[7]. Contrarily, automation scripts ease the deployment with JetPack automation toolkit. Separate instructions have been written for the suggested tunings, where applicable, for manual and automated modes of deployment in the below sections.

The best practices discussed below have been validated and implemented on Dell PowerEdge R740xd servers provided under Dell EMC Ready Architecture RHOSP 16.1 [8].

Relation between NIC and NUMA socket

NUMA-awareness is hardware architectural design dividing processor cores into single or multiple NUMA nodes. Each NUMA node has its own set of CPUs and memory allocation. Memory access to the processor has performance significance. A processor accessing memory from neighboring NUMA node would be slower than utilizing its own local memory. Performance could be improved when NIC and CPUs allocated to VMs are from the same NUMA node. The relationship between the two components can be verified through the steps discussed below.

Logical cores are divided equally among the two NUMA nodes on each compute host. The following snapshot indicates the vCPUs distribution on each compute node for the current setup.

```
[root@r85perfv-dell-compute-0 ~]# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70
node 0 size: 95111 MB
node 0 free: 4833 MB
node 1 cpus: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71
node 1 size: 96732 MB
node 1 free: 140 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

Figure 5. Distribution of cores across NUMA sockets

After RHOSP 16.1 installation, the NUMA socket of DPDK enabled interfaces can be verified using the commands below.

List all the interfaces on compute node with PCIe addresses:

```
# lshw -c network -businfo
```

[7] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/director_installation_and_usage/index

[8] https://www.delltechnologies.com/asset/en-us/products/ready-solutions/technical-support/dell-emc-ready-architecture-guide-red-hat-v16-1.pdf

```
[root@r85perfv-dell-compute-0 ~]# lshw -c network -businfo
Bus info          Device      Class          Description
========================================================
pci@0000:01:00.0  eno3        network        I350 Gigabit Network Connection
pci@0000:01:00.1  eno4        network        I350 Gigabit Network Connection
pci@0000:18:00.0  eno1        network        Ethernet Controller X710 for 10GbE SFP+
pci@0000:18:00.1  eno2        network        Ethernet Controller X710 for 10GbE SFP+
pci@0000:3b:00.0  ens1f0      network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:3b:00.1  ens1f1      network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:5e:00.0              network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:5e:00.1              network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:60:00.0              network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:60:00.1              network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:af:00.0  ens4f0      network        Ethernet Controller XXV710 for 25GbE SFP28
pci@0000:af:00.1  ens4f1      network        Ethernet Controller XXV710 for 25GbE SFP28
```

Figure 6. List of interfaces with PCIe Addresses

Find the NUMA socket for DPDK interfaces using their PCIe addresses.

```
# lspci -vmms 01:00.0 | grep NUMANode
```

```
[root@r85perfv-dell-compute-0 ~]# lspci -vmms 5e:00.0 | grep NUMANode
NUMANode:       0
[root@r85perfv-dell-compute-0 ~]# lspci -vmms 60:00.0 | grep NUMANode
NUMANode:       0
```

Above mentioned snapshot verifies that DPDK ports are from the same NUMA socket 0.

Note: This step is valid for both manual and automated deployments.

Isolated CPU allocation for PMD and guest VMs

The current setup i.e. *Appendix A* includes the isolation of CPUs for DPDK Poll Mode Drivers (PMDs) and guest VMs from CPUs allocated to host processes. Out of 72 cores on a host, four are dedicated to host processes with parameter **HostCpusList**, whereas ten are assigned as PMD cores (eight from NUMA socket-0, two from NUMA socket-1) with parameter **NeutronDpdkCoreList**. The default convention is to allocate one PMD thread per NUMA node, whereas guest VMs utilize the remaining cores with the parameter **NovaVcpuPinSet**.

These updated parameters are in a template file for automated (**neutron-OVS-dpdk.yaml**) and manual (**network-environment.yaml**) deployments. The reference for CPU parameters [9], DPDK parameters for network environment files[10] and OVS-DPDK parameters template [11] are given for detailed understanding.

DPDK socket memory

The socket memory is a buffer memory for packets in OVS-DPDK. Allocating optimal size for socket memory on compute hosts reduces packet drops.

---

[9] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.0/html/network_functions_virtualization_planning_and_configuration_guide/assembly_OVSdpdk_parameters#concept_OVSdpdk-cpu-parameters

[10] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html-single/network_functions_virtualization_planning_and_configuration_guide/index#proc_derive-dpdk

[11] https://github.com/dsp-jetpack/JetPack/blob/JS-16.1/src/pilot/templates/neutron-ovs-dpdk.yaml

Note: This is also pre-deployment optimization that needs to be verified for compute host(s).

Based on the following guidance[12] for a quad-port Intel Ethernet Network Adapter XXV710 NIC, MTU of 9,000 bytes for each port, the following set of socket memory calculations apply:

1. Round-up 9,000 bytes to the nearest multiple of 1,024 bytes, specifically 9,216 bytes.
2. Add padding of 800 bytes to the previous result and multiply by (4,096 X 64) to achieve 2,625,634,304 bytes.
    a. **Note** that 800 bytes are the overhead value, and 4,096 X 64 is the number of packets in the mempool.
3. Repeat steps one and two for each DPDK port connected to socket 0.
4. Calculate the total amount of memory required and include a 512 MB buffer. In this case the total memory required would be (4 X 2,625,634,304 bytes) + 536,870,912 bytes or 11,039,408,128 bytes.
5. Convert from bytes to megabytes, in this case resulting in 10,528 MB.
6. Round the previous result to the nearest multiple of 1,024 MB, specifically 11,264 MB.
7. Repeat steps one through six for socket one. Thus, for a NUMA balanced architecture with 2x quad-port Intel Ethernet Network Adapter XXV710 NIC, MTU of 9,000 bytes, the result would be 11,264 MB for socket 0 and 11,264 MB for socket one.

Note: Allocate at least 1024 MB to the unused socket if there are no DPDK ports attached to socket one.

Manual deployment

The compute hosts assign hugepage memory allocation as DPDK socket memory on each NUMA node. The **network-environment.yaml** file needs to be updated and passed as a template file when deploying OVS-DPDK. Socket memory parameter is set as **OVSDpdkSocketMemory** parameter in the file. The size of socket memory for each NIC is dependent on the size of MTU. The greater the size of MTU, the larger the socket memory. The previous section describes the detailed calculations for socket memory.

The reference template **network-environment.yaml** populates the DPDK socket memory with other parameters mentioned.[13]

[12] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/network_functions_virtualization_planning_and_configuration_guide/assembly_OVSdpdk_parameters#c_OVSdpdk-memory-params

[13] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/network_functions_virtualization_planning_and_configuration_guide/part-dpdk-configure#proc_derive-dpdk

Automated deployment

Update the socket memory for RHOSP 16 deployment using JetPack automation toolkit as follows:

- Manually calculate the DPDK socket memory for MTU 9000 bytes as calculated in the previous section. Name that value X. This value is for a single NIC. Similarly, calculate for other NICs, as this setup uses four NICs. Use the aggregated value for 4 NICs in the **get_socket_memory** formula's packets pool to produce 12288 MB of resultant socket memory.
- Clone the JetPack automation toolkit.
- Go to directory: Jetpack/src/pilot
- Edit the file **nfv_parameters.py**, update the function **get_socket_memory** by replacing 4096*64 with calculated value 'X' in previous steps as (4X*64). It will populate the **neutron-ovs-dpdk.yaml** file parameters on the runtime.

```
def get_socket_memory(self, mtu, dpdk_nics):
    nodes = self.data['nics'].keys()
    numa_mem = {el: 1024 for el in nodes}
    mtu = self.round_to_nearest(mtu, 1024)
    for n in nodes:
        mem = 536870912
        for nic in dpdk_nics:
            if nic in self.data['nics'][n]:
                mem += (mtu + 800) * (11264*64)
                break
        mem = mem / (1024*1024)
        numa_mem[n] = int(self.round_to_nearest(mem, 1024))
    self.socket_mem = ','.join(map(str, numa_mem.values()))
```

Figure 7. Socket memory function

Multiple RX queues on DPDK physical ports

Usually, the NIC has single Rx queue processing packets between the hardware and the kernel. However, the performance with single Rx queue is confined to larger packet sizes. The NIC redirects traffic to the Rx queue by hashing source/destination IP or source/destination port. Better performance is obtained with multiple Rx queues because each Rx queue has a separate CPU pinned to it.

Initially, the current setup utilized a single Rx queue and experienced packet drops. Afterward, we enabled multiple Rx queues on physical DPDK ports of compute hosts. The IP stepping was enabled in the traffic generator to hash the traffic on multiple Rx queues, basically employing Receive Side Scaling (RSS). Significant optimization was achieved that ultimately helped to gain ZPL with a 100% line rate.

We now discuss how to configure these parameters in a manual and automated RHOSP deployment.

Manual deployment

Enable multiple Rx queues in the **compute.yaml** template file. Reference for compute template is included to scale Rx queues[14]. Make sure to enable multi-Rx queues before deployment. The parameter that sets the number of Rx queues is **rx_queue** in Red Hat documentation.

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
        - type: ovs_dpdk_port
          name: dpdk1
          mtu: 9000
          members:
            - type: interface
              name: nic5
```

Figure 8. Rx Queue parameter in compute template

Automated deployment

In the JetPack toolkit, edit the file **dellcompute_dpdk.yaml** containing detailed parameters of compute host, located at the path:

JetPack/src/pilot/templates/nic-configs/OVS-dpdk_9_port/dellcompute_dpdk.yaml

To scale the Rx queues, update this parameter in the file: **rx_queue**. This parameter takes positive integer values.

```
NumDpdkInterfaceRxQueues:
  description: Number of Rx Queues required for DPDK bond or DPDK ports
  default: 2
  type: number
```

[14] https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/network_functions_virtualization_planning_and_configuration_guide/part-dpdk-configure#proc_OVSdpdk-multiqueu

```
----------

- type: OVS_dpdk_bond

              name: dpdkbond0

              rx_queue:

                get_param: NumDpdkInterfaceRxQueues
```

Note: The latest deployment was done using two Rx queues per DPDK port with zero ZPL.

Hugepages backing for VM profile

Hugepages are to be enabled on compute hosts for OVS-DPDK deployment in Red Hat OpenStack Platform16.1; this is a pre-deployment requirement. The recommended size for a hugepage is 1GiB. Virtual machines spun on compute hosts are allocated the desired number of hugepages from the host node.

DPDK memory channels

The multi-channel memory architecture technology enables support for multiple communication channels, thus, increasing the data transfer rates between memory modules, DRAM, and the memory controller. Hardware should be capable of supporting multiple memory channels.

Memory channels are wire traces between the memory unit and CPU for data movement. The number of memory channels acts as paths for data transfer at faster rates. For OVS-DPDK, the parameter **OVSDpdkMemoryChannels** holds the number of actively used channels. Mostly, memory channels per NUMA node are four (default) but may vary with platform. Configure the number of memory channels pre or post-deployment.

Before deployment, expose parameter "**OvsDpdkMemoryChannels**" in **network-environment.yaml** file. After deployment on the compute node, scale the number of memory channels on the runtime. The following commands scale the memory channels from default to the desired number.

List the default values:

```
# ovs-vsctl get Open_vSwitch . other_config
```

Scale the memory channels to six.

```
# ovs-vsctl set Open_vSwitch . other_config:dpdk-extra="-n 6"

{dpdk-extra="-n 6", dpdk-init="true", dpdk-lcore-
mask="3000000003", dpdk-socket-mem="8977,1024", pmd-cpu-
mask="1006010041006010004"}
```

DPDK Tx and Rx descriptor size

The descriptor size defines the Tx/Rx queue size for each port. On each compute node, both Tx and Rx descriptor sizes change from default 2048 to 4096. The steps to configure this are as follows:

Check the default statistics for the DPDK port.

```
# ovs-appctl dpctl/show --statistics | grep dpdk0
```

Scale the Tx, Rx descriptors size for each DPDK port, respectively.

```
# ovs-vsctl set Interface dpdk0 options:n_rxq_desc=4096
# ovs-vsctl set Interface dpdk0 options:n_txq_desc=4096
```

```
[root@r85perfv-dell-compute-0 ~]# ovs-appctl dpctl/show --statistics | grep dpdk0
  port 5: dpdk0 (dpdk: configured_rx_queues=2, configured_rxq_descriptors=2048, configured_tx_queues=11, configured_txq_descriptors=
2048, lsc_interrupt_mode=false, mtu=9000, requested_rx_queues=2, requested_rxq_descriptors=2048, requested_tx_queues=11, requested_t
xq_descriptors=2048, rx_csum_offload=true, tx_tso_offload=false)
[root@r85perfv-dell-compute-0 ~]#
[root@r85perfv-dell-compute-0 ~]#
[root@r85perfv-dell-compute-0 ~]#
[root@r85perfv-dell-compute-0 ~]# ovs-vsctl set Interface dpdk0 options:n_rxq_desc=4096
[root@r85perfv-dell-compute-0 ~]# ovs-vsctl set Interface dpdk0 options:n_txq_desc=4096
[root@r85perfv-dell-compute-0 ~]# ovs-appctl dpctl/show --statistics | grep dpdk0
  port 5: dpdk0 (dpdk: configured_rx_queues=2, configured_rxq_descriptors=4096, configured_tx_queues=11, configured_txq_descriptors=
4096, lsc_interrupt_mode=false, mtu=9000, requested_rx_queues=2, requested_rxq_descriptors=4096, requested_tx_queues=11, requested_t
xq_descriptors=4096, rx_csum_offload=true, tx_tso_offload=false)
```

Figure 9. DPDK port statistics

Emulator thread policy, multiple Rx queues on VirtIO

Achieve Emulator thread pinning by assigning an extra flavor spec, the default policy for this spec is **dedicated.** It's recommended to use the **isolate** policy in addition to the **dedicated** property with the flavor[15].

Enabling multiple Rx queues on virtual interfaces requires assigning an extra flavor spec. The following command shows how to set both properties flavors during instance creation[16].

```
# openstack flavor set <flavor name> --property
hw:emulator_threads_policy=isolate --property
hw:cpu_policy=dedicated --property hw_vif_multiqueue_enabled=true
```

```
| extra_specs               | {'hw:cpu_policy': 'dedicated', 'hw:cpu_thread_policy': 'require', 'hw:emulator_threads_policy': 'isol
ate', 'hw:mem_page_size': 'large', 'hw:numa_mempolicy': 'preferred', 'hw:numa_nodes': '1', 'hw_vif_multiqueue_enabled': 'true'} |
| id                        | a200c05b-1f9d-435d-9a38-0a8b1d2e81ee                                                                                          |
| name                      | mc-flavor                                                                                                                     |
| os-flavor-access:is_public | True                                                                                                                          |
| properties                | hw:cpu_policy='dedicated', hw:cpu_thread_policy='require', hw:emulator_threads_policy='isolate', hw:m
em_page_size='large', hw:numa_mempolicy='preferred', hw:numa_nodes='1', hw_vif_multiqueue_enabled='true'                       |
| ram                       | 8192                                                                                                                          |
```

Figure 10. VM flavor properties

IsolCPUs

The kernel thread interrupts are run in isolation from CPUs dedicated for guest VMs. This parameter is configured in grub command line parameters. List of CPUs for parameters nohz_full and rcu_nocbs are identical to isolCPUs. The configuration is mentioned in Appendix A under "*Other configurations*" section.

[15] https://access.redhat.com/solutions/3384881

[16] https://docs.openstack.org/glance/rocky/admin/useful-image-properties.html