OVS Offload using ASAP^2 Direct on ConnectX-4 / ConnectX-4 Lx

Performance Tuning Guide

Rev 2.0

www.mellanox.com

Mellanox Technologies, Inc.
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Contents

1 Introduction

This document describes Open vSwitch (OVS) offload using Mellanox "Accelerated Switching And Packet Processing" (ASAP^2) Direct technology performance verification procedure. Additionally, it describes the proper way to bring-up a system for optimized packet processing performance. The document is concluded with expected performance results and known performance limitations.

NOTE: This release was optimized for up to 10000 flows. Future releases will further optimize current performance and address larger flow count.

2 Hardware Specification

The following hardware was used in the performance verification process. Using as similar as possible systems will increase the chance of replicating the results described below.

- Server: HP ProLiant DL380p Gen8[1]
- CPU: Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz (IvyBridge), Dual socket, 12 cores per socket[2]
- NIC: ConnectX-4 and ConnectX-4 Lx

[1] Other similar Servers such as the following can be used as well: HP Gen9, Dell, Lenovo, SuperMicro.
[2] Other similar CPUs such as the following can be used as well: v3 serious (Haswell) with at least 2.4GHz frequency, 10 cores per socket (minimum)

3 Software Specification and Requirements

The following software was verified for optimized performance. Some requirements are very strict, such as hypervisor kernel and MLNX_OFED versions.

- Hypervisor
  - OS: Ubuntu15.10[3]
  - Kernel: 4.2.3-ovs-offload-r24
  - QEMU: QEMU emulator version 2.3.0 (Debian 1:2.3+dfsg-5ubuntu9.1)[4]
- VM
  - OS: Ubuntu15.10[3]
  - Kernel: 4.2.0-23-generic (default)
  - MLNX OFED : MLNX_OFED_LINUX-3.2-2.0.3.0.7
  - DPDK: dpdk.org (v2.2.0-mr-aligne branch)

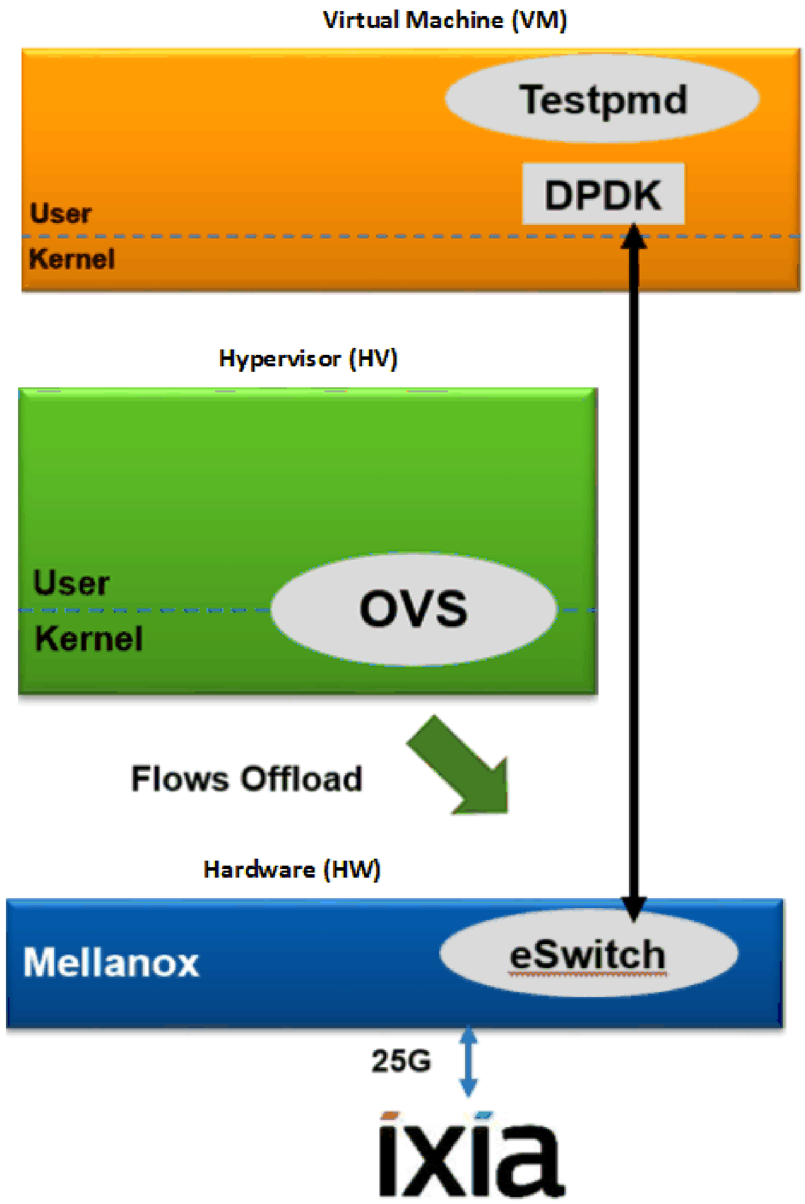
---

3 Other similar OS such as the following can be used as well: RH6.X. RH7.X is not recommended due to general SR-IOV issues.

4 If using other versions, please make sure they support the configuration detailed in "Setup Configuration"

4 Setup Description

In this setup, for optimized packet processing rate in OVS, a single VM setup was used. Packets were injected from a packet generator (Ixia) to the ConnectX-4/ConnectX-4 Lx NIC on the hypervisor and routed to the VFs connected to the VM using OVS. DPDK, running on the VM, forwarded incoming traffic on Port 1 to Port 2 and vise-versa. The packets were then routed again from the VFs to the NIC's physical port and back to the Ixia. The end results is the incoming packet rate the Ixia measures. The tests ran as dual-port and bidirectional, meaning both of the NIC's ports received and transmitted traffic from/to the Ixia.

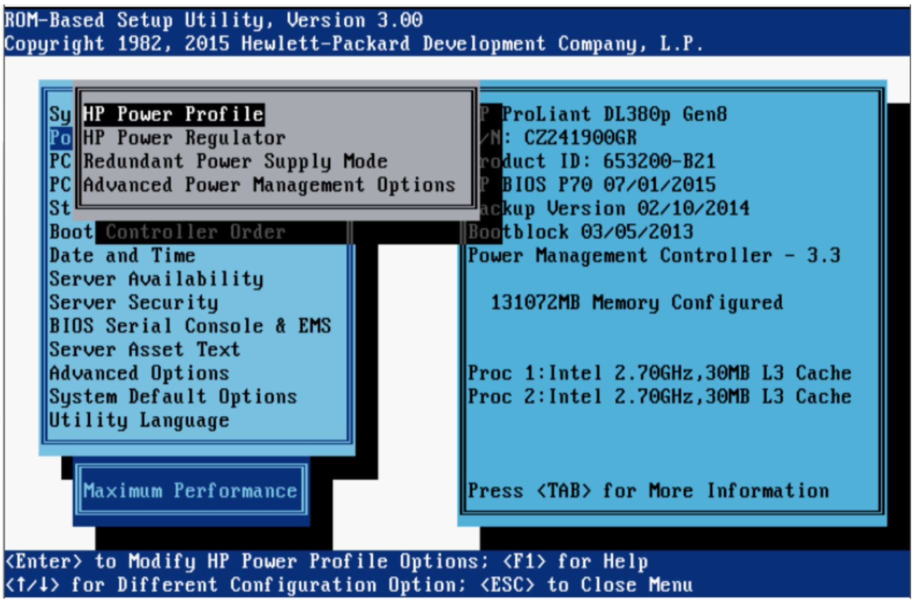


5 Setup Configuration

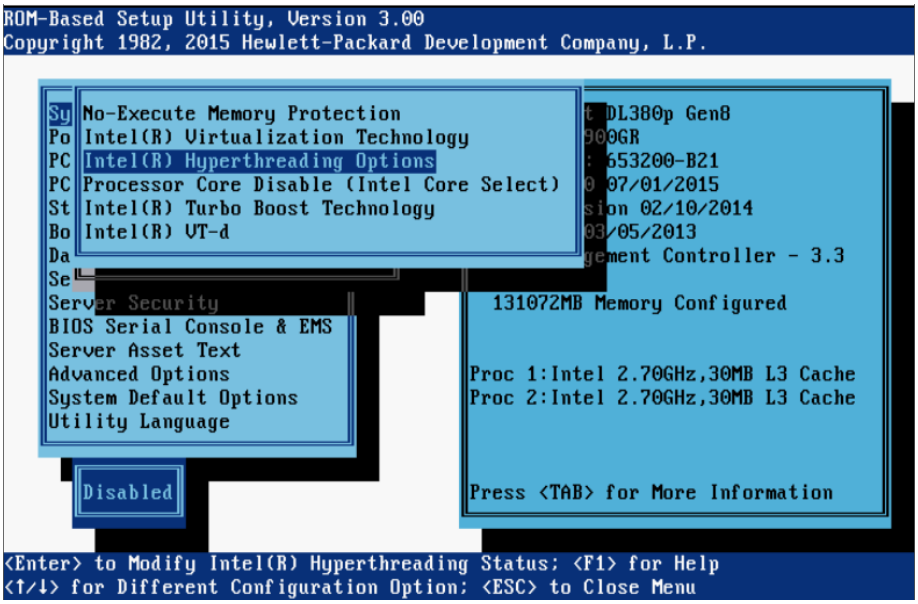
5.1 Host Configuration

Before running OVS on the host machine, the following configuration is recommended:

1. Make sure the setup is configured for ‘Max Performance’ in the BIOS power profile. (HP BIOS Example)



2. Make sure the Hyperthread is disabled on BIOS. (HP BIOS Example)



3. Find device Numa:

```
#cat /sys/class/net/<interface>/device/numa_node
1
```

4. Find the CPUs on the NUMA close to the device.

```
#lscpu
```

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    1
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 62
Model name:            Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
Stepping:              4
CPU MHz:               2055.000
CPU max MHz:           2700.0000
CPU min MHz:           1200.0000
BogoMIPS:              5392.20
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-11
NUMA node1 CPU(s):     12-23
```

5. Mark the CPUs allocated for the VM as isolated, by adding ‘isolcpus=<cpus>’ to grub file.

   We recommended choosing the CPUs related to the device numa.

   In addition, for optimal performance of OVS, intel_iommu=on iommu=pt should be added for pass-through to the device.

```
#Current Os number: 1

title Ubuntu 15.10, kernel 4.2.3+ OVS iommu NUMA 1
root (hd0,0)
kernel /vmlinuz-4.2.3-ovs-offload-r21 root=/dev/sda2 console=tty0
console=ttyS0,115200n8 ro quiet splash intel_iommu=on iommu=pt
isolcpus=12-23
initrd /initrd.img-4.2.3-ovs-offload-r21
```

Before creating the VM, we recommend to:

1. Load `mlx_core modle` with `flow_offload_group_size_log=16` (2^15 rules to config) and `flow_offload_min_inline=3` for steering on UDP ports.

```
#modprobe -v mlx5_core  flow_offload_group_size_log=16
flow_offload_min_inline=3
```

2. Turn OFF the flow control on the Physical Function interface.

```
#ethtool -A <interface> rx off tx off
```

3. Stop irq blancer.

```
/etc/init.d/irqbalance stop
```

4. Allocate at least 32G memory and make sure to enable hugepages.

```
mkdir -p /hugepages
mount -t hugetlbfs hugetlbfs /hugepages
echo 16384 > /sys/devices/system/node/node1/hugepages/hugepages-
2048kB/nr_hugepages

echo 0 > /sys/kernel/mm/transparent_hugepage/khugepaged/defrag
echo never > /sys/kernel/mm/transparent_hugepage/defrag
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

```
sysctl -w vm.zone_reclaim_mode=0
sysctl -w vm.swappiness=0
```

5. Bind the virtual CPUs thread to one of the isolated cores. Make sure to bind 1 virtual CPU per core.

   Example for a VM configuration with CPUs binding:

```
numactl --cpunodebind 1 --membind 1 -- \
echo 'info cpus' | \
/usr/bin/qemu-system-x86_64 \
-enable-kvm \
-name gen-l-vrt-178-005 \
-machine pc-i440fx-utopic,accel=kvm,usb=off \
-cpu host -m 16G \
-mem-path /hugepages -mem-prealloc  \
-realtime mlock=off -smp 9,sockets=9,cores=1,threads=1 \
-chardev socket,id=charmonitor,path=/var/lib/libvirt/qemu/gen-l-vrt-178-
005-UB15.10x64.monitor,server,nowait \
-mon chardev=charmonitor,id=monitor,mode=control \
-rtc base=utc,driftfix=slew \
-global kvm-pit.lost_tick_policy=discard -no-hpet \
-device virtio-serial-pci,id=virtio-serial0 \
-drive file=/images/gen-l-vrt-178-005/gen-l-vrt-178-005.img \
-netdev tap,id=hostnet0 -device
e1000,netdev=hostnet0,id=net0,mac=00:50:56:1b:b2:05 \
-chardev pty,id=charserial0 \
-chardev spicevmc,id=charchannel0,name=vdagent \
-device virtserialport,bus=virtio-
serial0.0,nr=1,chardev=charchannel0,id=channel0,name=com.redhat.spice.0 \
-msg timestamp=on -monitor stdio  \
-device isa-serial,chardev=charserial0,id=serial0 \
-vnc 127.0.0.1:0 \
-device cirrus-vga,id=video0,bus=pci.0,addr=0xf \
-device vfio-pci,host=24:00.2,id=hostdev0,bus=pci.0,addr=0x6 \
-device vfio-pci,host=24:01.2,id=hostdev1,bus=pci.0,addr=0x7 \
> /tmp/qemu_cpu_info.txt&


sleep 1

echo "Waiting for VM up...";
while ! ping -c1 gen-l-vrt-178-005&>/dev/null;
 do :
 sleep 1;
done

echo "VM is UP";

a=( $(cat /tmp/qemu_cpu_info.txt  | grep thread_id | cut -d '=' -f 3 | tr
-d '\r' ) )


taskset -p 0x002000  ${a[0]}
taskset -p 0x004000  ${a[1]}
taskset -p 0x008000  ${a[2]}
taskset -p 0x010000  ${a[3]}
taskset -p 0x020000  ${a[4]}
taskset -p 0x040000  ${a[5]}
taskset -p 0x080000  ${a[6]}
taskset -p 0x100000  ${a[7]}
taskset -p 0x200000  ${a[8]}
```

5.2 VM Configuration

5.2.1 DPDK Compilation

1. Get the tar DPDK file:

```
tar -xvf MLNX_DPDK_2.2.0-MR.tar.gz
```

2. Edit .the config/common_linuxapp file.

```
cd MLNX_DPDK_2.2.0-MR/
sed -i -- 's/CONFIG_RTE_LIBRTE_MLX5_PMD=n/CONFIG_RTE_LIBRTE_MLX5_PMD=y/g'
./config/common_linuxapp
sed -i --
's/CONFIG_RTE_LIBRTE_MLX5_SGE_WR_N=4/CONFIG_RTE_LIBRTE_MLX5_SGE_WR_N=1/g'
./config/common_linuxapp
sed -i --
's/CONFIG_RTE_LIBRTE_MLX5_MAX_INLINE=0/CONFIG_RTE_LIBRTE_MLX5_MAX_INLINE=
64/g' ./config/common_linuxapp
sed -i -- 's/CONFIG_RTE_EAL_IGB_UIO=n/CONFIG_RTE_EAL_IGB_UIO=n/g'
./config/common_linuxapp
sed -i --
's/CONFIG_RTE_LIBRTE_IXGBE_PMD=n/CONFIG_RTE_LIBRTE_IXGBE_PMD=n/g'
./config/common_linuxapp
sed -i -- 's/CONFIG_RTE_LIBRTE_KNI=n/CONFIG_RTE_LIBRTE_KNI=n/g'
./config/common_linuxapp
sed -i -- 's/CONFIG_RTE_KNI_KMOD=n/CONFIG_RTE_KNI_KMOD=n/g'
./config/common_linuxapp
sed -i -- 's/CONFIG_RTE_LIBRTE_EM_PMD=n/CONFIG_RTE_LIBRTE_EM_PMD=n/g'
./config/common_linuxapp
sed -i -- 's/CONFIG_RTE_LIBRTE_IGB_PMD=n/CONFIG_RTE_LIBRTE_IGB_PMD=n/g'
./config/common_linuxapp
sed -i -- 's/CONFIG_RTE_LIBRTE_I40E_PMD=n/CONFIG_RTE_LIBRTE_I40E_PMD=n/g'
./config/common_linuxapp
```

3. Apply the proper DPDK configuration.

```
make config T=x86_64-native-linuxapp-gcc
```

4. Compile dpdk

```
make -j
```

5. Allocate memory and make sure to enable hugepages for running DPDK.

```
mkdir -p /hugepages
mount -t hugetlbfs hugetlbfs /hugepages
echo 4096 > /sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
cat /proc/meminfo | grep Huge

echo 0 > /sys/kernel/mm/transparent_hugepage/khugepaged/defrag
echo never > /sys/kernel/mm/transparent_hugepage/defrag
echo never > /sys/kernel/mm/transparent_hugepage/enabled

sysctl -w vm.zone_reclaim_mode=0
sysctl -w vm.swappiness=0
```

a. Turn OFF the flow control on the physical function interface.

```
#ethtool -A <interface> rx off tx off
```

For benchmarking, DPDK IO forward on the VM was used. The recommended DPDK command is:

```
MLX5_WQE_MIN_INLINE_SIZE=64
MLX5_ENABLE_CQE_COMPRESSION=1/tmp/MLNX_DPDK_2.2.0-MR/build/app/testpmd  -c
0x1F0 -n 2 -w 00:06.0 -w 00:07.0 -- --burst=64 --txd=512 --rxd=512 --
mbcache=256 --rxq=2 --txq=2 --nb-cores=4  --rss-udp -i -a
```

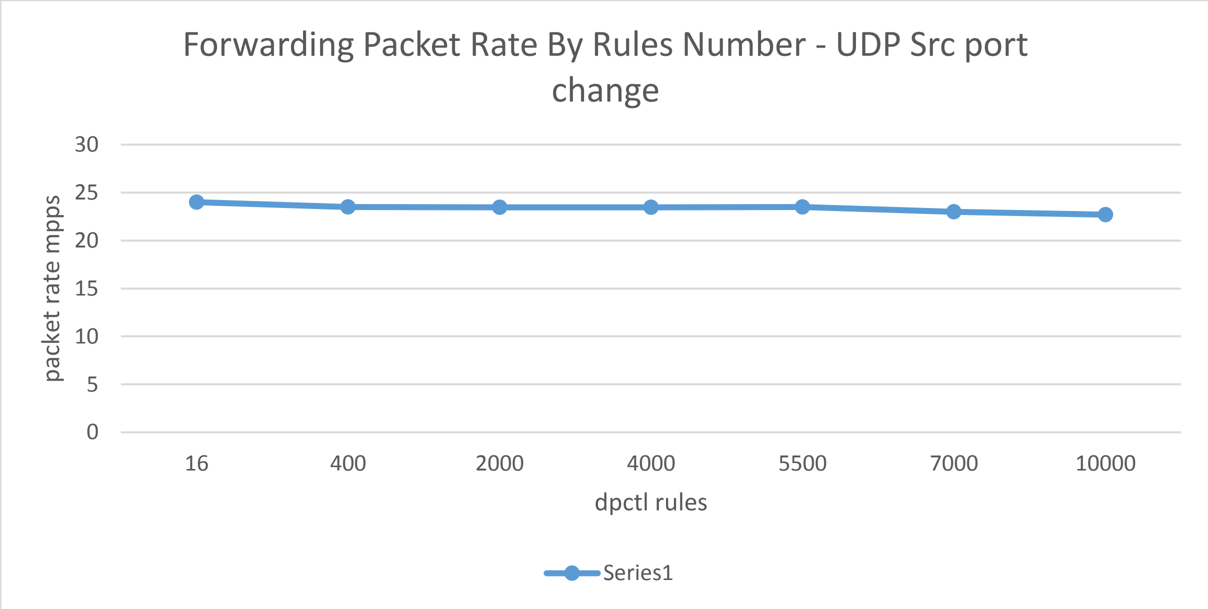
Notes:

1. Assign 1 queue per core.
2. Make sure UDP RSS is enabled.
3. Assign at least 4 queues per port.

6 Observed Performance Results

These are the performance results when using the configuration described above. The focus was DPDK IO forward message rate. This benchmark shows the OVS offload capabilities, while avoiding unrelated limitations, such as kernel message rate performance.

VXLAN performance: Message rate behavior is function of number of rules.



VXLAN	
rules	**packet rate [pps]**
16	24
400	23.5
2000	23.46
4000	23.45
5500	23.5
7000	22.98
10000	22.7

CPU utilization:

- On the host – No CPU utilization should be observed, beside on the cores that run the virtual CPUs for the VM
- On the VM – DPDK is running in polling mode, which means it utilize 100% of each CPU handling a queue