NETRONOME

Installation verification guide for

Agilio OVS 2.6A

Ubuntu 16.04

Revision: 1.1
July 2017

1 Document Overview and Goals **3**
1.1 Overview 3
1.2 Goals 4

2 Agilio OVS 2.6A Installation on Ubuntu 16.04 **5**
2.1 Installing Agilio OVS 2.6A 5
2.1.1 Install dependencies for Agilio OVS 2.6A 5
2.1.2 Package Installation 6
2.2 Installing and configuring KVM and virt-manager 7
2.3 Virtual Machine Setup 8
2.3.1 Create base image 8
2.3.2 Create additional virtual machines using base image 8

3 Performance Tuning **10**
3.1 Operating System Settings 10
3.2 BIOS settings 10
3.3 Host boot settings 11
3.2.1 Proper IOMMU setting 11
3.4 IRQ balancing 13
3.5 VirtIO relay tuning 14
3.6 VM CPU Pinning 14

4 Setup of prerequisites for Test Cases **15**
4.1 Infrastructure 15
4.1.1 Hardware 15
4.1.2 Software 16
4.2 General Host configuration 17
4.2.1 Configure hugepages 17

5 Executing Installation Verification Tests **19**
5.1 Test Case 1: Simple VF netdev back-2-back configuration 19
5.1.1 Test Setup 19
5.1.2 Bind Netronome nfp_netvf driver and assign IPv4 address 19
5.1.3 Configure simple bridge 20
5.1.4 Configure flows 21
5.1.5 Test Results 21
5.2 Test Case 2: VM-VM - SR-IOV Configuration 22
5.2.1 Prerequisites 22
5.2.2 Test Setup 22

5.2.3 Host: Bind VFIO-PCI driver 22
5.2.4 Host: Configure AOVS 23
5.2.5 Host: Configure OVS rules 24
5.2.6 Host: Configure Guest Domain Configuration 25
5.2.7 Host: Boot Guest Machine 25
5.2.8 Host: SSH into Guest Machine 25
5.2.9 Guest: Prepare Guest Machine 26
5.2.10 Guest: Setup Hugepages 26
5.2.11 Guest: Bind IGB-UIO driver 26
5.2.12 Guest: Using dpdk-pktgen 27
5.2.13 Test Results 30
5.3 Test Case 3: VM-VM - Virtio Configuration 33
5.3.1 Prerequisites 33
5.3.2 Test Setup 33
5.3.3 Host: Bind IGB-UIO driver 33
5.3.4 Host: Configure AOVS 34
5.3.5 Host: Configure OVS rules 35
5.3.6 Host: Configure Apparmor 35
5.3.7 Host: Configure Guest Domain Configuration 36
5.3.8 Host: Boot Guest Machine 37
5.3.9 Host: SSH into Guest Machine 38
5.3.10 Guest: Prepare Guest Machine 38
5.3.11 Guest: Setup Hugepages 38
5.3.12 Guest: Bind IGB-UIO driver 38
5.3.13 Guest: Using dpdk-pktgen 39
5.3.14 Test Results 42
5.4 Test Case 4: External Traffic Generator - SR-IOV Configuration 45
5.4.1 Prerequisites 45
5.4.2 Test Setup 45
5.4.3 Host: Bind VFIO-PCI driver 46
5.4.4 Host: Configure AOVS 46
5.4.5 Host: Configure OVS rules 47
5.4.6 Host: Configure Guest domain configuration 48
5.4.6 Host: Boot Guest Machine 49
5.4.7 Host: SSH into Guest Machine 49
5.4.8 Guest: Prepare Guest Machine 49
5.4.9 Guest: Bind IGB-UIO driver 49
5.4.10 Guest: Configure and run L2FWD 50

5.4.11 Test Results 51
5.5 Test Case 5: External Traffic Generator - Virtio Configuration 52
5.5.1 Prerequisites 52
5.5.2 Configuration setup 52
5.5.3 Host: Bind IGB-UIO driver 53
5.5.4 Host: Configure AOVS 53
5.5.5 Host: Configure OVS rules 54
5.5.6 Host: Configure Apparmor 55
5.5.7 Host: Configure Guest XML 56
5.5.8 Host: Boot Guest Machine 57
5.5.9 Host: SSH into Guest Machine 57
5.5.10 Guest: Prepare Guest Machine 57
5.5.11 Guest: Bind IGB-UIO driver 57
5.5.12 Guest: Configure and run L2FWD 58
9.5.13 Test Results 58

Appendix A : Quick Reference - Troubleshooting 60

Appendix B : Quick Reference - General Ubuntu Commands 61

Appendix C : BIOS Settings Explained 62
Intel Virtualization Technology 62
Hyper-threading 62
Turbo Boost 62
C-state 63
P-state 63

1 Document Overview and Goals

1.1 Overview

The Agilio platform is the first software- and hardware-based solution designed from the ground up to completely and transparently offload open source server-based networking data paths such as the widely deployed Open vSwitch (OVS). Agilio OVS software and the Agilio SmartNICs aim are a drop-in dataplane accelerator for Open vSwitch. Use cases include compute nodes for IaaS or SaaS, Network Functions Virtualization (NFV), and non-virtualized service nodes, among others.

In these use cases it is common to have a large number of network overlays and/or security policies that are enforced on the server. Agilio software focuses on OVS offload because it is the defacto standard for implementing overlays and network-level security policies among many data center operators.

This document includes the following benchmarking results:

1. Software: Agilio OVS 2.6A
2. Hardware: Netronome Agilio SmartNICs:
    a. Agilio CX 2x25GbE
    b. Agilio CX 1x40GbE
3. Benchmarking Topology: Use cases evaluated
    a. host-to-host (back-to-back; 2 servers)
    b. VM-to-VM (back-to-back; 2 servers)
    c. port-to-VM-to-port (single server; External Traffic Generator)

1.2 Goals

Repeatability is essential when testing complex systems. This can be difficult to achieve especially when testing complex virtualization solutions that include Open vSwitch. The complexity comes from the fact that there are lots of different components each with their own installation and configurations such as:

- Server/DUT Hardware -- Installation, drivers, configuration, firmware load, BIOS settings
- Test Equipment/Traffic Generator -- Installation, configuration, traffic characteristics, number of flows etc.

- DUT Software Optimization -- Host Operating System, KVM/QEMU Optimization, CPU Pinning, NUMA affinity, VM Operating System Optimization, OvS configuration etc.
- Theory of Operations for SmartNIC and Agilio Platform in the context of the above components.

The goals of this document are as follows:

1. Provide details on the validation of Agilio OVS 2.6A configuration
2. Provide well-documented Reference Architectures to replicate the configuration environments used within this document

2 Agilio OVS 2.6A Installation on Ubuntu 16.04

This installation guide serves to supplement the Agilio OvS Getting Started Guide and the Agilio OvS User Guide. Please refer to these documents for additional information.

Note: All commands and scripts contained in this document must be executed with **root** privileges.

2.1 Installing Agilio OVS 2.6A

When installing Agilio OVS 2.6A it is important to completely remove any previous installations of Agilio OVS prior to starting the install procedure (2.1.3 Package Installation). The uninstall can be explicitly executed by running the uninstall script

```
/opt/netronome/bin/agilio-ovs-uninstall.sh
```

2.1.1 Install dependencies for Agilio OVS 2.6A

The following dependencies must be installed before installing Agilio OVS 2.6A.
To install the packages the following command must be executed.

Script: **install_dependencies.sh**

```
#!/bin/bash
#install_dependencies.sh

apt-get install make autoconf automake libtool gcc g++ bison \
    flex hwloc-nox libreadline-dev libpcap-dev dkms libftdi1 \
    libjansson4 libjansson-dev guilt pkg-config libevent-dev \
    ethtool libssl-dev libnl-3-200 libnl-3-dev libnl-genl-3-200 \
    libnl-genl-3-dev psmisc gawk libzmq3-dev protobuf-c-compiler \
    protobuf-compiler python-protobuf libnuma1 libnuma-dev \
    python-six python-ethtool libcap-ng0 uuid-runtime
```

2.1.2 Package Installation

Download and extract the latest Agilio OVS release from the Netronome Support Site. For the purposes of this document **agilio-ovs-2.6.A-r5662-2017-05-16_dpkg** will be used as an example.

```
tar -xf agilio-ovs-2.6.A-r5662-2017-06-15_dpkg.tar.gz
cd agilio-ovs-2.6.A-r5662-2017-05-16_dpkg/xenial
```

Install Agilio OVS by issuing the following command
Script: **package_install.sh**

```
#!/bin/bash
#package_install.sh

cd agilio-ovs-2.6.A-r5662-2017-06-15_dpkg/xenial

dpkg -i agilio-ovs*_2.6.A.*_amd64.deb \
agilio-ovs*-common_2.6.A.*_amd64.deb \
agilio-ovs-trivial_2.6.A.*_amd64.deb \
igb-uio-dkms_*_amd64.deb \
nfp-bsp-6000-b0-2017.02_*_amd64.deb \
nfp-bsp-6000-b0-2017.02-dkms_*_all.deb \
nfp-cmsg-dkms_*_amd64.deb nfp-fallback-dkms_*_amd64.deb \
nfp-conntrack-dkms_*_amd64.deb nfp-offloads-dkms_*_amd64.deb \
openvswitch-common_2.6.1*agilio2.6.A.*_amd64.deb \
openvswitch-datapath-dkms_2.6.1*agilio2.6.A.*_all.deb \
```



```
openvswitch-switch_2.6.1*agilio2.6.A.*_amd64.deb \
netronome-dpdk_16.11.*_amd64.deb \
virtiorelayd_2.6.A.*_amd64.deb
```

Once all the packages have been installed issue the following command to check and/or update the NIC flash. If the NIC flash was updated, the system will require a reboot.

```
/opt/netronome/bin/nfp-update-flash.sh
```

Configure physical interfaces if a specific mode is required

To view the current port configuration:

```
nfp-media
```

Refer to the Agilio User Guide for valid port configurations. To change the port configuration, run the follow command:

```
nfp-media phy0=4x10G phy1=40G #etc
```

If the port configuration has been changed, a reboot will be required:

```
reboot
```

2.2 Installing and configuring KVM and virt-manager

KVM must be installed on the host machine as a hypervisor. To ensure that the latest version of KVM is installed, this command should be run regardless whether KVM is installed on the system or not. To install KVM, run the following command:

```
apt-get install kvm libvirt-bin virtinst bridge-utils \
cpu-checker cloud-image-utils
```


virt-manager can also be installed to provision and manage virtual machines with a GUI. This package may be installed on any system that has ssh access to the host running the virtual machines. To install virt-manager in a Ubuntu environment, run the following command:

```
apt-get install virt-manager
```

2.3 Virtual Machine Setup

Note: If you reach the internet by means of a proxy, the following commands may be issued to successfully **`wget`** a file from the internet.

```
export http_proxy=http://proxy_host:proxy_port

#For authenticated proxy:
export http_proxy=http://username:password@proxy_host:proxy_port

#Replace "http" with "https" if you are using https
```

2.3.1 Create base image

To quickly provision new virtual machines, the scripts located in vm_creator may be used to rapidly create new virtual machines.

The contents of **`vm_creator/ubuntu/vm_scripts/samples`** will be copied into the base image. If you need any additional files in your VM, they may be placed inside this folder.

Running **`x_create_backing_image.sh`** will run VM creation scripts in sequence to create a base image that may be used to provision new virtual machines. This script can only be run once and the resulting modified image is used to create more virtual machines.

The image is saved in **`/var/lib/libvirt/images`** as **`ubuntu-16.04-server-cloudimg-amd64-disk1.img`** with a approximate size of 1.2Gb

```
#!/bin/bash

./0_download_cloud_image.sh
./1_cloud_init.sh
./2_install_vm.sh
./3_copy_vm_scripts.sh
./4_run_vm_scripts.sh
```


2.3.2 Create additional virtual machines using base image

After creating the base image, **`y_create_vm_from_backing.sh`** may be used to create new VM's using the modified image. Changes to VCPU count and RAM can be made to suit a specific use-case. This script will not reboot the new VM. Changes to the VM's xml file can now be made before the VM boots up for the first time.

```
#!/bin/bash

LIBVIRT_DIR=/var/lib/libvirt/images
basefile=$LIBVIRT_DIR/ubuntu-16.04-server-cloudimg-amd64-disk1.img
read -p "Enter a name for VM: " VM_NAME

cat <<- EOF > /tmp/ifcfg-eth0
DEVICE="eth0"
ONBOOT="yes"
IPV6INIT="no"
BOOTPROTO="dhcp"
TYPE="Ethernet"
EOF

echo "create overlay image"
overlay=$LIBVIRT_DIR/$VM_NAME.qcow2
qemu-img create -b $basefile -f qcow2 $overlay
sleep 5
guestfish --rw -i -a $overlay write /etc/hostname $VM_NAME
echo "create domain"

  cpu_model=$(virsh capabilities | grep -o '<model>.*</model>' | head -1 | sed
's/\(<model>\|<\/model>\)//g')
  name=$VM_NAME
  virt-install \
       --name $name \
       --disk path=${overlay},format=qcow2,bus=virtio,cache=none \
       --ram 4096 \
       --vcpus 4 \
       --cpu $cpu_model \
       --network network=default \
       --nographics \
       --debug \
       --accelerate \
       --os-type=linux \
       --os-variant=ubuntu16.04 \
```



```
      --noautoconsole \
      --noreboot \
      --import

echo "VM has been created!"
```

3 Performance Tuning

3.1 Operating System Settings

- Linux OS Services Settings
    - Disable NetworkManager
    - Disable firewalld
    - Disable iptables
    - Disable irqbalance

3.2 BIOS settings

There are several BIOS settings that influence the processor's control of the CPU frequency and power consumption. The following BIOS settings has the largest impact on performance:

- Functional:
    - Intel Virtualization Technology
    - Hyper-Threading
- Performance:
    - CPU C State Control
    - CPU P State Control
    - Turbo Boost

The desired BIOS settings are listed in the table below. For a more detailed explanation on the effect each BIOS setting has on performance, see BIOS Settings Explained

BIOS Setting	Recommended Value/Setting
Intel Virtualization Technology	Enabled
Hyper-threading	Disabled
Turbo Boost	Disabled
C-State	C0 (Fully Operational)
P-State	Disabled

3.3 Host boot settings

The following parameters must be added to **`GRUB_CMDLINE_LINUX_DEFAULT=`** located in **`/etc/default/grub`** An example of a boot line configuration is provided at the end of this section.

3.2.1 Proper IOMMU setting

```
intel_iommu=on iommu=pt intremap=on
```

These setting are REQUIRED.
Incorrect iommu settings will result in improper operation, e.g. traffic fails to flow to VMs.

- intel_iommu=on: Activate Intel VT-d (enable SR-IOV) in the kernel. The Intel VT-d extensions provides hardware support for directly assigning a physical devices to guest. The main benefit of the feature is to improve the performance as native for device access.
- iommu=pt: the adapter does not need to use DMA translation to the memory, and this improves the performance which improves hypervisor performance
- intremap=on: enable Interrupt Remapping. Interrupt remapping provides hardware support for remapping and routing of interrupt requests from I/O devices (generated directly or through I/O interrupt controllers). The indirection achieved through remapping enables isolation of interrupts across partitions.

3.2.1 Turn off CPU power throttling:

```
intel_idle.max_cstate=0 processor.max_cstate=0 idle=mwait intel_pstate=disable
```


These settings are DESIRED. If set incorrectly, processor power state changes will cause “blips” in the traffic. Throughput results will therefore not be optimal. For more details on the C-state and P-state see Section 1.1 BIOS preparation

3.2.2 Enable hugepages:

```
default_hugepagesz=2M hugepagesz=2M hugepages=26624
```

Note: the amount of hugepages allocated is depended on your design

3.2.3 Isolate CPUs for OS:

CPU’s that are used by VirtIO_relay and VM’s should be isolated to ensure optimal performance.

Further, it is desired that the CPU’s chosen for VirtIO_relay and VM’s belong to the same NUMA node where the Agilio NIC is installed.

Run the following script to determine which NUMA node each Agilio NIC is installed on and what CPUs are associated with that NUMA node:
Script: **`identify_NIC_numa_node.sh`**

```
#!/bin/bash
#identify_NIC_numa_node.sh

for card in /sys/bus/pci/drivers/nfp/0*; do
    address=`basename $card`
    echo "Agilio address: $address"
    echo -n "NUMA node: "; cat $card/numa_node
    echo -n "Local CPUs: "; cat $card/local_cpulist
done
```

Example output:

```
Agilio address: 0000:04:00.0
NUMA node: 0
Local 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
```

Finally the isolcpus parameter in the **`GRUB_CMDLINE_LINUX_DEFAULT`** may be modified to isolate the correct CPU’s


```
isolcpus=2-27,30-55
```

The above setting will allow the OS (and Agillio OVS) to run on CPU 0,1,28, and 29 only
Example line in **`/etc/default/grub`**:

```
GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on iommu=pt intremap=on
default_hugepagesz=2M hugepagesz=2M hugepages=26624 isolcpus=2-27,30-55
intel_idle.max_cstate=0 processor.max_cstate=0 idle=mwait intel_pstate=disable"
```

An example script is supplied that will modify your grub file. Changes may be made to this script as needed:

Script: **`1_configure_grub.sh`**

```
#!/bin/bash
#1_configure_grub.sh

OUT="$(grep -n "GRUB_CMDLINE_LINUX_DEFAULT" /etc/default/grub | cut -d: -f1)"
sed -i "$OUT s/^/#/" /etc/default/grub

sed -i "$OUT a GRUB_CMDLINE_LINUX_DEFAULT=\"intel_iommu=on iommu=pt intremap=on
isolcpus=10-15 intel_idle.max_cstate=0 processor.max_cstate=0 idle=mwait
intel_pstate=disable\"" /etc/default/grub
```

Activate grub changes

```
sudo update-grub
reboot
```

To verify that grub changes have been made, the following command can be executed:

```
cat /proc/cmdline
```

3.4 IRQ balancing

The following script will turn IRQ balancing off permanently:

Script: **2_disable_irq_balancing.sh**

```
#!/bin/bash
#2_disable_irq_balancing.sh

sed -i 's/ENABLED="1"/ENABLED="0"/' /etc/default/irqbalance

service irqbalance stop
```

3.5 VirtIO relay tuning

Edit **SDN_VIRTIORELAY_PARAM** in **/etc/netronome.conf** to configure VirtIO-relay. Isolate which CPUs VirtIO-relay will run on, by specifying a list of physical CPU's. These CPU's will run with DPDK poll-mode-driver so it is expected that CPU load will be at 100% for these CPU's

```
--cpus=2,4
```

Choose CPUs that are in the grub "isolcpus" list. Be sure to use CPUs from the same NUMA node that the Agilio NIC is on.

Note: Care must be taking in insuring that the hyperthread partners of the chosen VirtIO-relay cores are idle. VirtIO-relay performance will be adversely affected when using their respective virtIO-relay partners for other tasks

A restart of A-OVS is required after making changes to netronome.conf.

3.6 VM CPU Pinning

The following example will pin 3 physical CPUs to a VM with 3 virtual CPU's running on the example system:

```
virsh list

 Id    Name                           State
----------------------------------------------------
 2     vm1                             running
```


List default CPU affinity for the VM :

```
virsh vcpupin vm1  # List CPU Affinity for vm1

VCPU: CPU Affinity
----------------------------------
   0: 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
   1: 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
   2: 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
```

Pin CPU's:

```
virsh vcpupin 2 0 2 # Pin VCPU 0 to CPU 2
virsh vcpupin 2 1 4 # Pin VCPU 1 to CPU 4
virsh vcpupin 2 2 6 # Pin VCPU 2 to CPU 6
```

Use the command below to confirm that the VM is pinned correctly.

```
virsh vcpupin vm1
```

Sample of expected result:

```
VCPU: CPU Affinity
----------------------------------
   0: 2
   1: 4
   2: 6
```

4 Setup of prerequisites for Test Cases

4.1 Infrastructure

4.1.1 Hardware

The benchmarking setup was performed using the following servers:

dual_DUT1 and dual_DUT2 were connected back to back.
dual_DUT2 and dual_DUT3 were connected back to back

- single_DUT
  - System: Supermicro (X10DRI-T)
  - CPU: 2x Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz (10 cores; HT enabled)
  - Memory: 8x 8192 MB; DDR4; 2133 MHz (Samsung)
  - Agilio CX 2x40GbE
- dual_DUT1
  - System: Dell PowerEdge R730
  - CPU: Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
  - Memory: 8x 8192 MB; DDR4; 2133 Mhz
  - Agilio CX 2x40GbE
- dual_DUT2
  - System: Dell PowerEdge R730
  - CPU: Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
  - Memory: 8x 8192 MB; DDR4; 2133 Mhz
  - Agilio CX 2x40GbE

- dual_DUT3
  - System: Dell PowerEdge R730
  - CPU: Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
  - Memory: 8x 8192 MB; DDR4; 2133 Mhz
  - Agilio CX 1x25GbE
- dual_DUT4
  - System: Dell PowerEdge R730
  - CPU: Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
  - Memory: 8x 8192 MB; DDR4; 2133 Mhz
  - Agilio CX 1x25GbE

4.1.2 Software

- single_DUT
  - Host OS: Ubuntu 16.04.2 LTS kernel: 4.4.0-75-generic
  - libvirt version: libvirt 1.3.1; QEMU 2.5.0
  - Agilio OVS 2.6A [r5559] (OVS: 2.6.1)
- dual_DUT1
  - Host OS: Ubuntu 16.04.2 LTS kernel 4.4.0-77-generic
  - libvirt version: libvirt 1.3.1; QEMU 2.5.0
  - Agilio OVS 2.6A [r5559] (OVS: 2.6.1)
- dual_DUT2

    - Host OS: Ubuntu 16.04.2 LTS kernel 4.4.0-77-generic
    - libvirt version: libvirt 1.3.1; QEMU 2.5.0
    - Agilio OVS 2.6A [r5559] (OVS: 2.6.1)
- dual_DUT3
    - Host OS: Ubuntu 16.04.2 LTS kernel 4.4.0-77-generic
    - libvirt version: libvirt 1.3.1; QEMU 2.5.0
    - Agilio OVS 2.6A [r5559] (OVS: 2.6.1)
- dual_DUT4
    - Host OS: Ubuntu 16.04.2 LTS kernel 4.4.0-77-generic
    - libvirt version: libvirt 1.3.1; QEMU 2.5.0
    - Agilio OVS 2.6A [r5559] (OVS: 2.6.1)

4.2 General Host configuration

4.2.1 Configure hugepages

Script: **`configure_hugepages.sh`**

```
#!/bin/bash
#configure_hugepages.sh

function printCol {
  # Usage: printCol <COLOR> <MESSAGE>
  # 0 - Black. 1 - Red.     2 - Green. 3 - Yellow.
  # 4 - Blue.  5 - Magenta. 6 - Cyan.  7 - White.
  echo "$(tput bold)$(tput setaf $1)$2$(tput sgr0)"
}

echo "START"

cat /proc/mounts | grep hugetlbfs

umount /mnt/aovs-huge-2M ## should we stick to default mount or create new?

printCol 7 "Setting 2M"
grep hugetlbfs /proc/mounts | grep -q "pagesize=2M" || \
( mkdir -p /mnt/huge && mount nodev -t hugetlbfs -o rw,pagesize=2M /mnt/huge/ )

printCol 7 "Setting 1G"
grep hugetlbfs /proc/mounts | grep -q "pagesize=1G" || \
( mkdir -p /mnt/huge-1G && mount nodev -t hugetlbfs -o rw,pagesize=1G /mnt/huge-1G/
)
```

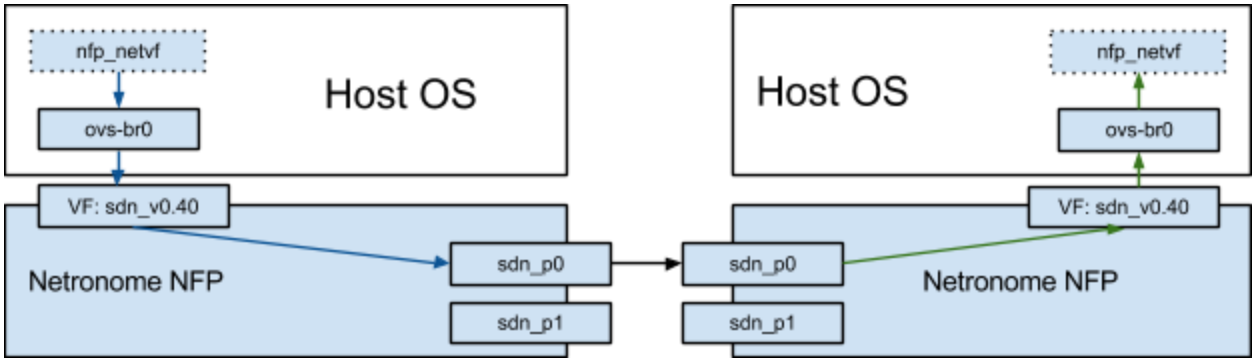


```
printCol 7 "/proc/mounts | grep hugetlbfs"
cat /proc/mounts | grep hugetlbfs

printCol 7 "libvirt folders"
mkdir -p /mnt/huge-1G/libvirt
mkdir -p /mnt/huge/libvirt
chown libvirt-qemu:kvm -R /mnt/huge-1G/libvirt
chown libvirt-qemu:kvm -R /mnt/huge/libvirt

service libvirt-bin restart

echo 4096 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages

echo "END"
```

5 Executing Installation Verification Tests

5.1 Test Case 1: Simple VF netdev back-2-back configuration

5.1.1 Test Setup



5.1.2 Bind Netronome nfp_netvf driver and assign IPv4 address

The following script will bind the basic nfp_netvf driver to VF sdn_v0.40 and assign an IPv4 address to the spawned ethx interface.

Execute on both hosts - change the IP address before executing on the second host. The IP address can be passed into this script as the first argument. If no argument is supplied the script will use 14.0.0.1 as the default IP.

For example: **`./1_bind_netronome_nfp_netvf_driver.sh 20.0.0.1`**

Script: **`1_bind_netronome_nfp_netvf_driver.sh`**

```
#!/bin/bash
#1_bind_netronome_nfp_netvf_driver.sh

if [ -z “$1” ]; then
   IP=14.0.0.1
else
   IP=$1
fi

#Configure Interface

DPDK_DEVBIND=$(locate dpdk-devbind.py | head -1)
PCIA="$(ethtool -i sdn_v0.40 | grep bus | cut -d ' ' -f 5)"

  echo $DPDK_DEVBIND --bind nfp_netvf $PCIA
  $DPDK_DEVBIND --bind nfp_netvf $PCIA

echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status

#Assign IP

ETH=$($DPDK_DEVBIND --status | grep $PCIA | cut -d ' ' -f 4 | cut -d '=' -f 2)

ip addr add $IP/24 dev $ETH
ip link set dev $ETH up
```

5.1.3 Configure simple bridge

The following script will create a bridge containing the physical interface(sdn_p0) and VF(sdn_v0.40).

Script: **2_configure_bridge.sh**

```
#!/bin/bash
#2_configure_bridge.sh

BRIDGE=br0

# Delete all bridges
for br in $(ovs-vsctl list-br);
do
  ovs-vsctl --if-exists del-br $br
done

# Create a new bridge
ovs-vsctl add-br $BRIDGE

# Add physical ports
ovs-vsctl add-port $BRIDGE sdn_p0 -- set interface sdn_p0 ofport_request=1

# Add VF ports
ovs-vsctl add-port $BRIDGE sdn_v0.40 -- set interface sdn_v0.40 ofport_request=40

ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
ovs-vsctl set Open_vSwitch . other_config:flow-limit=1000000
ovs-appctl upcall/set-flow-limit 1000000

ovs-vsctl show
ovs-ofctl show $BRIDGE
ovs-ofctl dump-flows $BRIDGE
```

5.1.4 Configure flows

The following script creates basic flow rules to allow all traffic through the bridge.

Script: **3_configure_ovs_rules.sh**

```
#!/bin/bash
#3_configure_ovs_rules.sh

BRIDGE=br0

ovs-ofctl del-flows $BRIDGE
ovs-ofctl add-flow $BRIDGE arp,actions=normal
ovs-ofctl add-flow $BRIDGE ip,actions=normal
```


5.1.5 Test Results

Ping result (ping 14.0.0.1 from 14.0.0.2)

```
PING 14.0.0.1 (14.0.0.1) 56(84) bytes of data.
64 bytes from 14.0.0.1: icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from 14.0.0.1: icmp_seq=2 ttl=64 time=0.013 ms
64 bytes from 14.0.0.1: icmp_seq=3 ttl=64 time=0.012 ms
64 bytes from 14.0.0.1: icmp_seq=4 ttl=64 time=0.012 ms
64 bytes from 14.0.0.1: icmp_seq=5 ttl=64 time=0.012 ms
64 bytes from 14.0.0.1: icmp_seq=6 ttl=64 time=0.012 ms
```

5.2 Test Case 2: VM-VM - SR-IOV Configuration

5.2.1 Prerequisites

This test assumes that the following tasks have already been completed:

- Agilio OVS 2.6A Installation on Ubuntu 16.04
- Installing and configuring KVM and virt-manager
- Virtual Machine Setup
- Performance Tuning

5.2.2 Test Setup

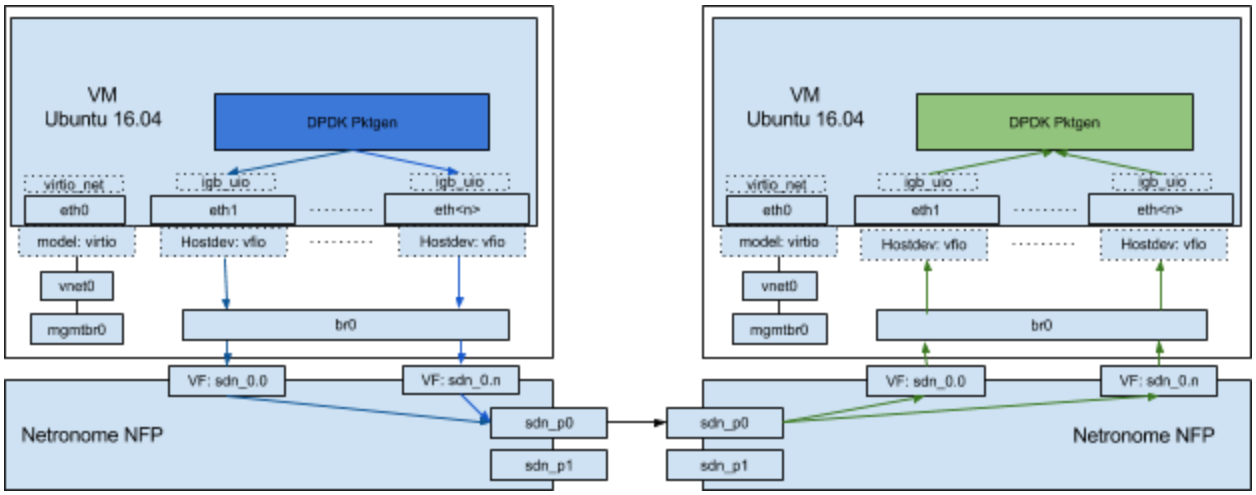


Figure X. SR-IOV Use Case

5.2.3 Host: Bind VFIO-PCI driver

This script will bind two VF's (sdn_v0.41 and sdn_v0.42) to the vfio-pci driver. vfio-pci is available in kernels >4.1 and is prefered over the UIO framework, which has no notion of IOMMU protection and has limited interrupt support.

Script: **1_bind_VFIO-PCI_driver.sh**

```
#!/bin/bash
#1_bind_VFIO-PCI_driver.sh

PCIA="$(ethtool -i sdn_v0.41 | grep bus | cut -d ' ' -f 5)"
PCIB="$(ethtool -i sdn_v0.42 | grep bus | cut -d ' ' -f 5)"
driver=vfio-pci
# updatedb
DPDK_DEVBIND=$(find /opt/netronome -iname dpdk-devbind.py | head -1)
echo "loading driver"
modprobe $driver
echo "DPDK_DEVBIND: $DPDK_DEVBIND"
echo $DPDK_DEVBIND --bind $driver $PCIA
$DPDK_DEVBIND --bind $driver $PCIA

echo $DPDK_DEVBIND --bind $driver $PCIB
$DPDK_DEVBIND --bind $driver $PCIB

echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status
```

5.2.4 Host: Configure AOVS

Script: **2_configure_AVOS.sh**

```
#!/bin/bash
#2_configure_AVOS.sh

echo "CURRENT configuration"
cat /etc/netronome.conf

cat << 'EOF' > /etc/netronome.conf
SDN_VIRTIORELAY_ENABLE=n
SDN_FIREWALL=n
EOF
```



```
echo "NEW configuration"
cat /etc/netronome.conf

ovs-ctl status
ovs-ctl stop
ovs-ctl start
ovs-ctl status
```

5.2.5 Host: Configure OVS rules

Script: **3_configure_AOVS_rules.sh**

```
#!/bin/bash
#3_configure_AOVS_rules.sh

BRIDGE=br0

# Delete all bridges
for br in $(ovs-vsctl list-br);
do
  ovs-vsctl --if-exists del-br $br
done

# Create a new bridge
ovs-vsctl add-br $BRIDGE

# Add physical ports
ovs-vsctl add-port $BRIDGE sdn_p0 -- set interface sdn_p0 ofport_request=1

# Add VF ports
ovs-vsctl add-port $BRIDGE sdn_v0.41 -- set interface sdn_v0.41 ofport_request=41
ovs-vsctl add-port $BRIDGE sdn_v0.42 -- set interface sdn_v0.42 ofport_request=42

#Add NORMAL RULE
ovs-ofctl del-flows br0
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE actions=NORMAL


ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
ovs-vsctl set Open_vSwitch . other_config:flow-limit=1000000
ovs-appctl upcall/set-flow-limit 1000000

ovs-vsctl show
```



```
ovs-ofctl show $BRIDGE
ovs-ofctl dump-flows $BRIDGE
```

5.2.6 Host: Configure Guest Domain Configuration

This script will modify a VM's XML file and add two SR-IOV interfaces to the VM. In this case, sdn_v0.41 and sdn_v0.42.

Note: Change the VM_NAME variable to the name of the VM you want to add the interfaces to.

Script: **`4_guest_xml_configure.sh`**

```
#!/bin/bash
#4_guest_xml_configure.sh

# Remove vhostuser interface
EDITOR='sed -i "/<interface type=.vhostuser.>/,/<\/interface>/d"' virsh edit
$VM_NAME
EDITOR='sed -i "/<hostdev mode=.subsystem. type=.pci./,/<\/hostdev>/d"' virsh edit
$VM_NAME

# Add vhostuser interfaces
# sdn_v0.41 --> 0000:81:0d.1
# sdn_v0.42 --> 0000:81:0d.2

bus=$(ethtool -i sdn_v0.42 | grep bus-info | awk '{print $5}' | awk -F ':' '{print
$2}')
EDITOR='sed -i "/<devices/a \<hostdev mode=\"subsystem\" type=\"pci\"
managed=\"yes\">  <source> <address domain=\"0x0000\" bus=\"0x'${bus}'\"
slot=\"0x0d\" function=\"0x1\"\/> <\/source>  <address type=\"pci\"
domain=\"0x0000\" bus=\"0x00\" slot=\"0x06\" function=\"0x0\"\/> <\/hostdev>"'
virsh edit $VM_NAME

EDITOR='sed -i "/<devices/a \<hostdev mode=\"subsystem\" type=\"pci\"
managed=\"yes\">  <source> <address domain=\"0x0000\" bus=\"0x'${bus}'\"
slot=\"0x0d\" function=\"0x2\"\/> <\/source>  <address type=\"pci\"
domain=\"0x0000\" bus=\"0x00\" slot=\"0x07\" function=\"0x0\"\/> <\/hostdev>"'
virsh edit $VM_NAME
```


5.2.7 Host: Boot Guest Machine

```
VM_NAME=vm1
virsh start $VM_NAME
```

5.2.8 Host: SSH into Guest Machine

```
VM_NAME=vm1
ssh root@$(virsh net-dhcp-leases default | awk '/'"$VM_NAME"'/ {print $5}' | cut
-d"/" -f1)

#Default password: changeme
```

5.2.9 Guest: Prepare Guest Machine

The newly created VM will have a folder named **`vm_scripts/samples`**. The following two scripts in this folder must executed to successfully run DPDK-pktgen tests.

5.2.10 Guest: Setup Hugepages

Script: **`1_configure_hugepages.sh`**

```
#!/bin/bash
#1_configure_hugepages.sh

echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

5.2.11 Guest: Bind IGB-UIO driver

This script will automatically bind all available SR-IOV interfaces presented in the VM to the IGB-UIO driver.

Script: **`2_auto_bind_igb_uio.sh`**

```
#!/bin/bash
#2_auto_bind_igb_uio.sh
```



```
DRIVER=igb_uio
mp=uio
whitelist=""

lspci | grep 01:
if [ $? = 1 ]; then
  NETRONOME_VF_LIST=$(lspci -d 19ee: | awk '{print $1}')
else
  NETRONOME_VF_LIST=$(lspci | grep 01: | awk '{print $1}')
fi

DPDK_DEVBIND=$(readlink -f $(find /root/ -name "dpdk-devbind.py" | head -1))
DRKO=$(find ~ -iname 'igb_uio.ko' | head -1 )

echo $NETRONOME_VF_LIST
modprobe $mp
insmod $DRKO

for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  $DPDK_DEVBIND --bind $DRIVER $netronome_vf
  whitelist="$whitelist $netronome_vf"
done

echo "whitelist: $whitelist"
exit 0
```

5.2.12 Guest: Using dpdk-pktgen

The following steps may be followed to perform a **uni-directional** test between two VM's located on different compute nodes.

To log results that are generated with DPDK-pktgen, the following scripts located in **`/vm_scripts/samples/3_dpdk_pktgen_lua_capture`** may be used.

The DPDK-Pktgen scripts located in this folder automatically assigns one CPU to a Pktgen port and therefore one CPU to every VF that is presented in the VM.

Start **`0_run_dpdk-pktgen_uni-tx.sh`** on VM1 and then start **`1_run_dpdk-pktgen_uni-rx.sh`** on VM2 to capture traffic within 60 seconds of starting the transmitter script.

0_run_dpdk-pktgen_uni-tx.sh sends traffic to the second VM. The script iterates through different packet sizes. Modifications may be made to **unidirectional_transmitter.lua** to change packet sizes that are being sent.

Script: **0_run_dpdk-pktgen_uni-tx.sh**

```
#!/bin/bash
#0_run_dpdk-pktgen_uni-tx.sh

export DPDK_BASE_DIR=/root
export PKTGEN=/root/pktgen-dpdk-pktgen-3.3.2
script_dir="$(dirname $(readlink -f $0))"
cd $PKTGEN

CPU_COUNT=$(cat /proc/cpuinfo | grep processor | wc -l)

#Check for virtIO-relay interfaces on bus 1, otherwise, it will be SR-IOV
interfaces
lspci | grep 01:
if [ $? == 1 ]; then
NETRONOME_VF_LIST=$(lspci -d 19ee: | awk '{print $1}')
else
NETRONOME_VF_LIST=$(lspci | grep 01: | awk '{print $1}')
fi

memory="--socket-mem 1024"
lcores="-l 0-$((CPU_COUNT-1))"

# whitelist
whitelist=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  whitelist="$whitelist $netronome_vf"
done

# cpumapping
cpu_counter=0
port_counter=0
mapping=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  mapping="${mapping}-m "
```



```
        cpu_counter=$((cpu_counter+1))
        echo "cpu_counter: $cpu_counter"
        mapping="${mapping}${cpu_counter}"

  mapping="${mapping}.${port_counter} "
  port_counter=$((port_counter+1))
done

echo "whitelist: $whitelist"
echo "mapping: $mapping"

/root/dpdk-pktgen $lcores --proc-type auto $memory -n 4 --log-level=7 $whitelist
--file-prefix=dpdk0_ -- $mapping -N -f $script_dir/unidirectional_transmitter.lua

reset
```

1_run_dpdk-pktgen_uni-rx.sh is run on the second VM to sync traffic that is being transmitted from the first VM.

Script: **1_run_dpdk-pktgen_uni-rx.sh**

```
#!/bin/bash
#1_run_dpdk-pktgen_uni-rx.sh

export DPDK_BASE_DIR=/root
export PKTGEN=/root/pktgen-dpdk-pktgen-3.3.2
script_dir="$(dirname $(readlink -f $0))"
cd $PKTGEN

CPU_COUNT=$(cat /proc/cpuinfo | grep processor | wc -l)

#Check for virtIO-relay interfaces on bus 1, otherwise, it will be SR-IOV
interfaces
lspci | grep 01:
if [ $? == 1 ]; then
NETRONOME_VF_LIST=$(lspci -d 19ee: | awk '{print $1}')
else
NETRONOME_VF_LIST=$(lspci | grep 01: | awk '{print $1}')
fi

memory="--socket-mem 1024"
lcores="-l 0-$((CPU_COUNT-1))"

# whitelist
```



```
whitelist=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  whitelist="$whitelist $netronome_vf"
done

# cpumapping
cpu_counter=0
port_counter=0
mapping=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  mapping="${mapping}-m "

       cpu_counter=$((cpu_counter+1))
       echo "cpu_counter: $cpu_counter"
       mapping="${mapping}${cpu_counter}"

  mapping="${mapping}.${port_counter} "
  port_counter=$((port_counter+1))
done

echo "whitelist: $whitelist"
echo "mapping: $mapping"

/root/dpdk-pktgen $lcores --proc-type auto $memory -n 4 --log-level=7 $whitelist
--file-prefix=dpdk0_ -- $mapping -N -f $script_dir/unidirectional_receiver.lua

reset
```

5.2.13 Test Results

Below is a sample output of a DPDK-pktgen instance running on the Tx VM.

- **`Pkts/s`** - Indicates packets being sent and received
- **`MBits/s`** - Indicates throughput, sending and receiving.
- **`PktSize/Tx Burst`** - Indicates the current packet size
- **`TotalRate`** - Indicates the summation of all traffic being sent and/or received (In this example case, only one port is sending traffic)

```
Ports 0-1 of 2   <Main Page>  Copyright (c) <2010-2016>, Intel Corporation
```



```
  Flags:Port        :   ---------------:0   ---------------:1
Link State          :      <UP-10000-FD>       <UP-10000-FD>       ----TotalRate----
Pkts/s Max/Rx       :      5651840/2347680     5690069/2347670     11306279/4695350
       Max/Tx       :      12985312/2347616    12750400/2347648    25505728/4695264
MBits/s Rx/Tx       :            9991/9991           9991/9991           19983/19983
Broadcast           :                   0                   0
Multicast           :                   0                   0
  64 Bytes          :                   0                   0
  65-127            :                   0                   0
  128-255           :                   0                   0
  256-511           :                   0                   0
  512-1023          :           250570331           250571294
  1024-1518         :                   0                   0
Runts/Jumbos        :                   0/0                 0/0
Errors Rx/Tx        :                   0/0                 0/0
Total Rx Pkts       :           411537636           348013420
       Tx Pkts      :           601034880           410701472
       Rx MBs       :           1166181             1123496
       Tx MBs       :           1268441             1140547
ARP/ICMP Pkts       :                 0/0                 0/0
                    :
Pattern Type        :           abcd...             abcd...
Tx Count/% Rate     :       Forever / 100%      Forever / 100%
PktSize/Tx Burst    :       512 /   32          512 /   32
Src/Dest Port       :       1234 / 5678         1234 / 5678
Pkt Type:VLAN ID    :             IPv4 / TCP:0001     IPv4 / TCP:0001
Dst  IP Address     :       192.168.2.1         192.168.3.1
Src  IP Address     :       192.168.2.2/24      192.168.3.2/24
Dst MAC Address     :   52:54:00:d0:2b:87   52:54:00:ce:b9:23
Src MAC Address     :   52:54:00:e2:d3:01   52:54:00:04:c1:18
VendID/PCI Addr     :   1af4:1000/00:06.0   1af4:1000/00:07.0

-- Pktgen Ver: 3.0.17 (DPDK 16.11.0)  Powered by Intel® DPDK ------------------
```

The results of the test run will be saved on the receiving VM as /root/capture.txt. The results are comma separated and may easily be pasted into a spreadsheet file.

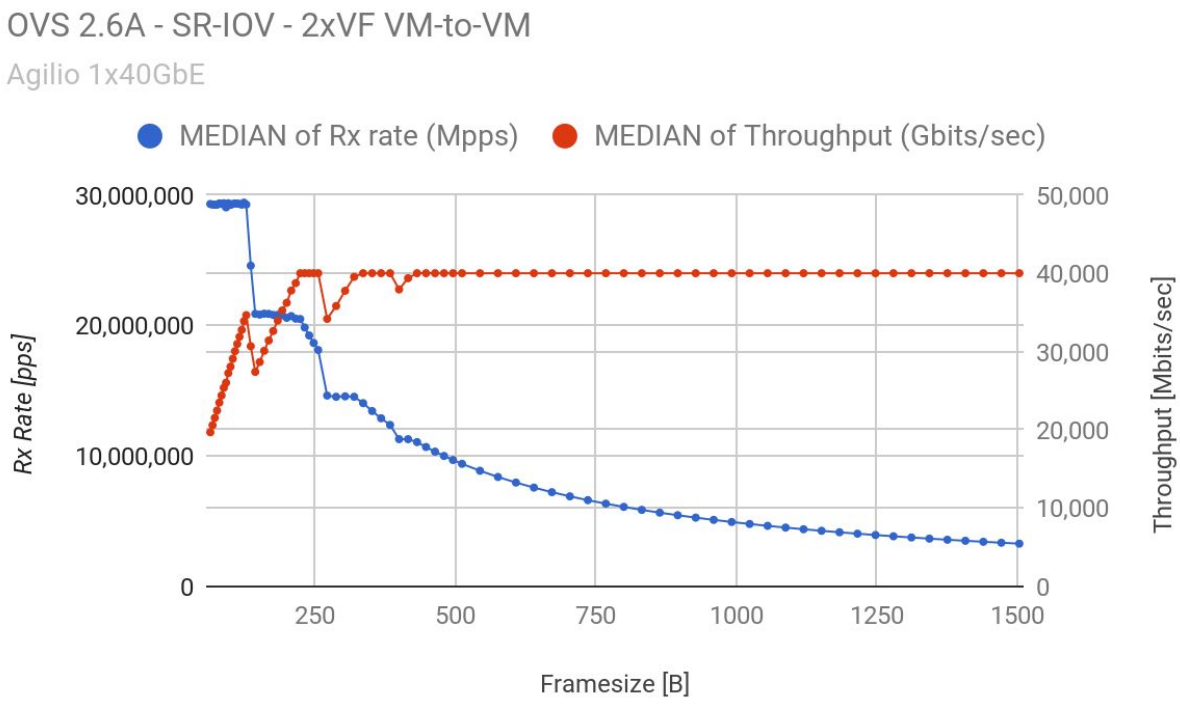
Example output of comma separated file of the data that is logged by DPDK-pktgen on the Rx VM.

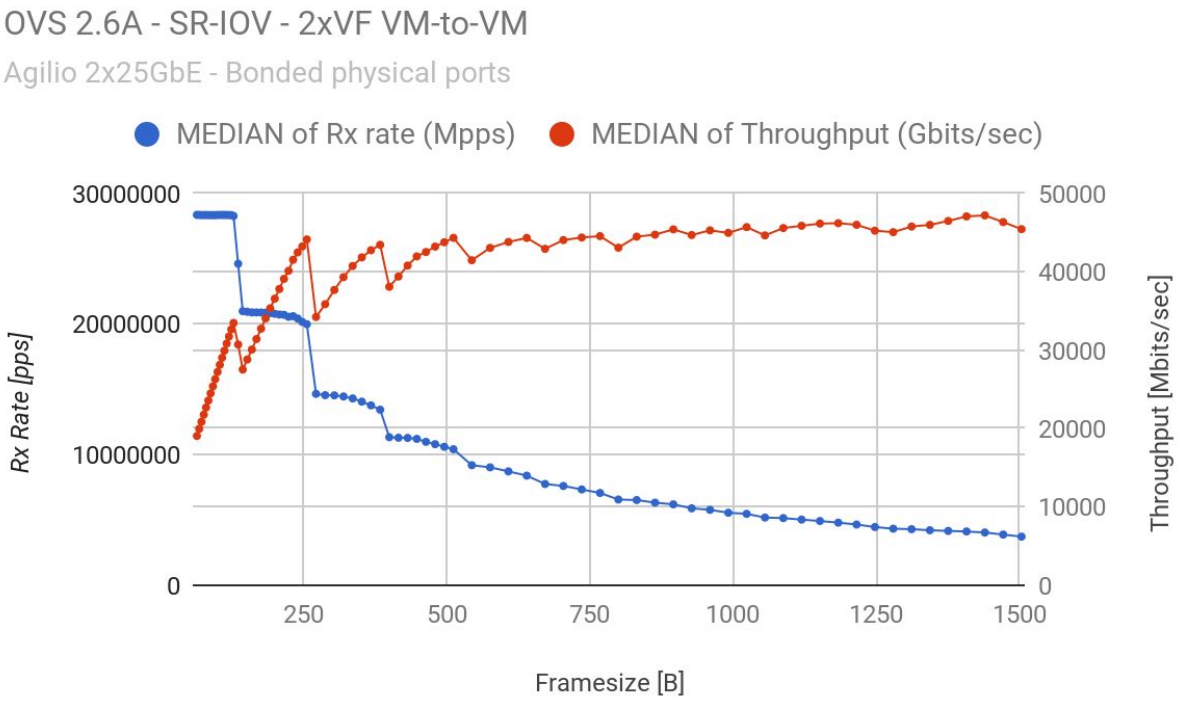
```
      Time,  Ports, Framesize,  total_pkts_rx, total_mbits_rx
```



```
7.0,        1,    64,        20198371,        13573
8.0,        1,    64,        20042486,        13468
9.0,        1,    64,        20177209,        13559
10.0,       1,    64,        20218240,        13586
11.0,       1,    64,        20185959,        13564
12.0,       1,    64,        20200468,        13574
13.0,       1,    64,        20328945,        13661
14.0,       1,    64,        20220310,        13588
15.0,       1,    64,        20212942,        13583
16.0,       1,    64,        20219968,        13587
17.0,       1,    64,        20198027,        13573
18.0,       1,    64,        20083556,        13496
19.0,       1,    64,        20214888,        13584
22.0,       1,    128,       20213392,        23932
```

The resulting comma separated file was imported into google sheets (or similar) and a pivot table was created using framesize as rows and rx rate and throughput as columns. The following graphs were generated using this pivot table.





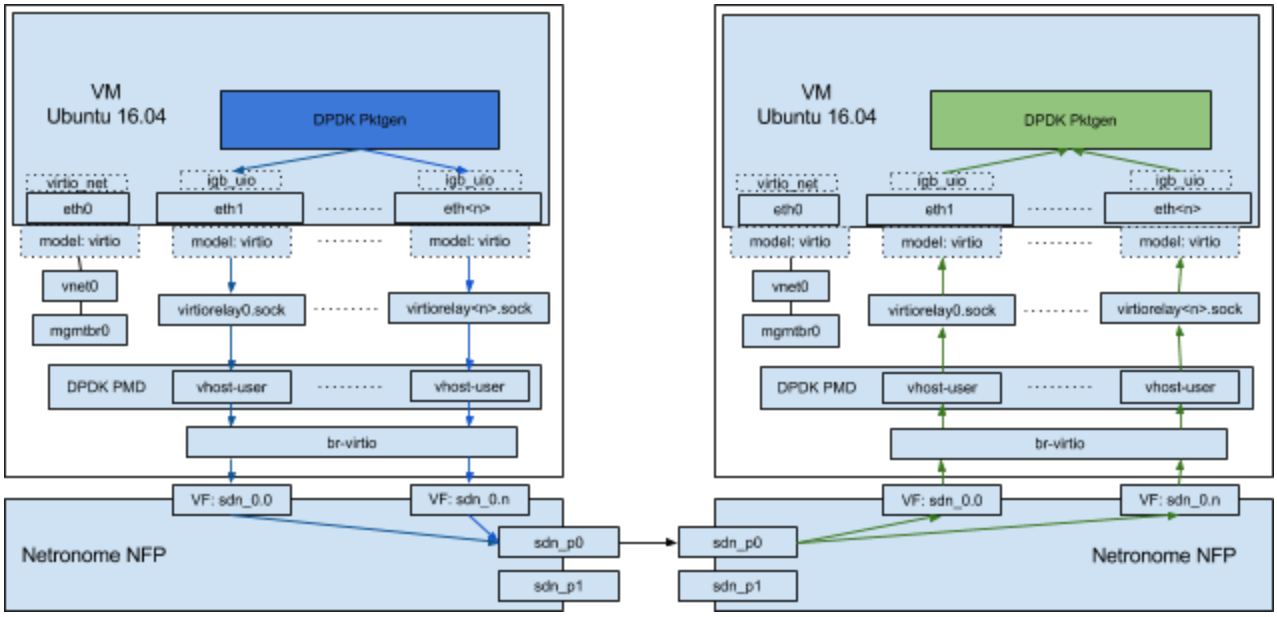
5.3 Test Case 3: VM-VM - Virtio Configuration

5.3.1 Prerequisites

This test assumes that the following tasks have already been completed:

- Agilio OVS 2.6A Installation on Ubuntu 16.04
- Installing and configuring KVM and virt-manager
- Virtual Machine Setup
- Performance Tuning

5.3.2 Test Setup



5.3.3 Host: Bind IGB-UIO driver

Script: **`1_bind_IGB-UIO_driver.sh`**

```
#!/bin/bash
#1_bind_IGB-UIO_driver.sh

PCIA1="$(ethtool -i sdn_v0.40 | grep bus | cut -d ' ' -f 5)"
PCIA2="$(ethtool -i sdn_v0.41 | grep bus | cut -d ' ' -f 5)"

interface_list=($PCIA1 $PCIA2)
driver=igb_uio
mp=uio
# updatedb
DPDK_DEVBIND=$(find /opt/ -iname dpdk-devbind.py | head -1)
DRKO=$(find /opt/ -iname 'igb_uio.ko' | head -1 )
echo "loading driver"
modprobe $mp
insmod $DRKO
echo "DPDK_DEVBIND: $DPDK_DEVBIND"
for interface in ${interface_list[@]};
do
  echo $DPDK_DEVBIND --bind $driver $interface
```



```
  $DPDK_DEVBIND --bind $driver $interface
done
echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status
```

5.3.4 Host: Configure AOVS

Script: **2_configure_ovs.sh**

```
#!/bin/bash
#2_configure_ovs.sh

echo "CURRENT configuration"
cat /etc/netronome.conf

cat << 'EOF' > /etc/netronome.conf
SDN_VIRTIORELAY_ENABLE=y
SDN_VIRTIORELAY_PARAM="--cpus=1,2 --enable-tso --enable-mrgbuf
--vhost-username=libvirt-qemu --vhost-groupname=kvm --huge-dir=/mnt/huge
--ovsdb-sock=/var/run/openvswitch/db.sock"
SDN_FIREWALL=n
EOF

echo "NEW configuration"
cat /etc/netronome.conf

ovs-ctl status
ovs-ctl stop
ovs-ctl start
ovs-ctl status
```

5.3.5 Host: Configure OVS rules

Script: **3_configure_ovs_rules.sh**

```
#!/bin/bash
#configure_ovs_rules.sh

BRIDGE=br0

# Delete all bridges
for br in $(ovs-vsctl list-br);
do
  ovs-vsctl --if-exists del-br $br
done
```



```
# Create a new bridge
ovs-vsctl add-br $BRIDGE

ovs-vsctl add-port $BRIDGE sdn_p0 -- set interface sdn_p0 ofport_request=1

ovs-vsctl add-port $BRIDGE sdn_v0.40 -- set interface sdn_v0.40 ofport_request=40
external_ids:virtio_relay=40
ovs-vsctl add-port $BRIDGE sdn_v0.41 -- set interface sdn_v0.41 ofport_request=41
external_ids:virtio_relay=41

ovs-ofctl del-flows $BRIDGE
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE actions=NORMAL

ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
ovs-vsctl set Open_vSwitch . other_config:flow-limit=1000000
ovs-appctl upcall/set-flow-limit 1000000

ovs-vsctl show
ovs-ofctl show $BRIDGE
ovs-ofctl dump-flows $BRIDGE
```

5.3.6 Host: Configure Apparmor

Script: **`4_configure_apparmor.sh`**

```
#!/bin/bash
#4_configure_apparmor.sh

ADD="$(grep -n 'libxl-save-helper' /etc/apparmor.d/usr.sbin.libvirtd | cut -d:
-f1)"
OUT="$(grep -n '/usr/local/bin/qemu* PUx,' /etc/apparmor.d/usr.sbin.libvirtd | cut
-d: -f1)"
[ -z "$OUT" ] && sed -i "$ADD a /usr/local/bin/qemu* PUx, #ADD OVS"
/etc/apparmor.d/usr.sbin.libvirtd
OUT="$(grep -n '/usr/local/bin/ovs-vsctl rmix,' /etc/apparmor.d/usr.sbin.libvirtd |
cut -d: -f1)"
[ -z "$OUT" ] && sed -i "$ADD a /usr/local/bin/ovs-vsctl rmix, #ADD OVS"
/etc/apparmor.d/usr.sbin.libvirtd

OUT="$(grep -n "deny /tmp/" /etc/apparmor.d/abstractions/libvirt-qemu | cut -d:
-f1)"
sed -i "$OUT s/^/#/" /etc/apparmor.d/abstractions/libvirt-qemu
```



```
OUT="$(grep -n "deny /var/tmp/" /etc/apparmor.d/abstractions/libvirt-qemu | cut -d:
-f1)"
sed -i "$OUT s/^/#/" /etc/apparmor.d/abstractions/libvirt-qemu

sudo service apparmor reload
```

5.3.7 Host: Configure Guest Domain Configuration

This script will modify a VM's XML file and add two VHost-user interfaces to the VM. In this case, sdn_v0.40 and sdn_v0.41.

Note: Change the VM_NAME variable to the name of the VM you want to add the interfaces to.

Script: **`5_configure_guest_xml.sh`**

```
#!/bin/bash
#5_configure_guest_xml.sh

VM_NAME="vm1"
VM_CPU=6
max_memory=$(virsh dominfo $VM_NAME | grep 'Max memory:' | awk '{print $3}')

# Remove vhostuser interface
EDITOR='sed -i "/<interface type=.vhostuser.>/,/<\/interface>/d"' virsh edit
$VM_NAME
EDITOR='sed -i "/<hostdev mode=.subsystem. type=.pci./,/<\/hostdev>/d"' virsh edit
$VM_NAME

# Add vhostuser interfaces
# sdn_v0.40 --> 0000:81:0d.0
# sdn_v0.41 --> 0000:81:0d.1

bus=$(ethtool -i sdn_v0.42 | grep bus-info | awk '{print $5}' | awk -F ':' '{print
$2}')
# Add vhostuser interfaces
EDITOR='sed -i "/<devices/a \<interface type=\"vhostuser\">  <source type=\"unix\"
path=\"/tmp/virtiorelay40.sock\" mode=\"client\"\/>  <model type=\"virtio\"/>
<driver name=\"vhost\" queues=\"1\"\/>  <address type=\"pci\" domain=\"0x0000\"
bus=\"0x01\" slot=\"0x06\" function=\"0x0\"\/><\/interface>"' virsh edit $VM_NAME
EDITOR='sed -i "/<devices/a \<interface type=\"vhostuser\">  <source type=\"unix\"
path=\"/tmp/virtiorelay41.sock\" mode=\"client\"\/>  <model type=\"virtio\"/>
<driver name=\"vhost\" queues=\"1\"\/>  <address type=\"pci\" domain=\"0x0000\"
bus=\"0x01\" slot=\"0x07\" function=\"0x0\"\/><\/interface>"' virsh edit $VM_NAME
```



```
EDITOR='sed -i "/<numa>/,/<\/numa>/d"' virsh edit $VM_NAME
EDITOR='sed -i "/vcpu/d"' virsh edit $VM_NAME
EDITOR='sed -i "/<cpu/,/<\/cpu>/d"' virsh edit $VM_NAME
EDITOR='sed -i "/<memoryBacking>/,/<\/memoryBacking>/d"' virsh edit $VM_NAME

virsh setvcpus $VM_NAME $VM_CPU --config --maximum
virsh setvcpus $VM_NAME $VM_CPU --config

# MemoryBacking
EDITOR='sed -i "/<domain/a \<memoryBacking><hugepages><page size=\"2048\"
unit=\"KiB\" nodeset=\"0\"\/><\/hugepages><\/memoryBacking>"' virsh edit $VM_NAME

# CPU
echo max_memory: $max_memory
echo VM_CPU: $VM_CPU
EDITOR='sed -i "/<domain/a \<cpu mode=\"host-model\"><model
fallback=\"allow\"\/><numa><cell id=\"0\" cpus=\"0-'$((VM_CPU-1))'\"
memory=\"'${max_memory}'\" unit=\"KiB\" memAccess=\"shared\"\/><\/numa><\/cpu>"'
virsh edit $VM_NAME
```

5.3.8 Host: Boot Guest Machine

```
VM_NAME=vm1
virsh start $VM_NAME
```

5.3.9 Host: SSH into Guest Machine

```
VM_NAME=vm1
ssh root@$(virsh net-dhcp-leases default | awk '/'"$VM_NAME"'/ {print $5}' | cut
-d"/" -f1)

#Default password: changeme
```

5.3.10 Guest: Prepare Guest Machine

The newly created VM will have a folder named **`vm_scripts/samples`**. The following two scripts in this folder must executed to successfully run DPDK-pktgen tests.

5.3.11 Guest: Setup Hugepages

Script: **`1_configure_hugepages.sh`**

```
#!/bin/bash
#1_configure_hugepages.sh

echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

5.3.12 Guest: Bind IGB-UIO driver

This script will automatically bind all available SR-IOV interfaces presented in the VM to the IGB-UIO driver.

Script: **`2_auto_bind_igb_uio.sh`**

```
#!/bin/bash
#2_auto_bind_igb_uio.sh

DRIVER=igb_uio
mp=uio
whitelist=""

lspci | grep 01:
if [ $? = 1 ]; then
  NETRONOME_VF_LIST=$(lspci -d 19ee: | awk '{print $1}')
else
  NETRONOME_VF_LIST=$(lspci | grep 01: | awk '{print $1}')
fi

DPDK_DEVBIND=$(readlink -f $(find /root/ -name "dpdk-devbind.py" | head -1))
DRKO=$(find ~ -iname 'igb_uio.ko' | head -1 )

echo $NETRONOME_VF_LIST
modprobe $mp
insmod $DRKO

for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  $DPDK_DEVBIND --bind $DRIVER $netronome_vf
```



```
  whitelist="$whitelist $netronome_vf"
done

echo "whitelist: $whitelist"
exit 0
```

5.3.13 Guest: Using dpdk-pktgen

The following steps may be followed to perform a **uni-directional** test between two VM's located on different compute nodes.

To log results that are generated with DPDK-pktgen, the following scripts located in **/vm_scripts/samples/3_dpdk_pktgen_lua_capture** may be used.

The DPDK-Pktgen scripts located in this folder automatically assigns one CPU to a Pktgen port and therefore one CPU to every VF that is presented in the VM.

Start **0_run_dpdk-pktgen_uni-tx.sh** on VM1 and then start **1_run_dpdk-pktgen_uni-rx.sh** on VM2 to capture traffic within 60 seconds of starting the transmitter script.

0_run_dpdk-pktgen_uni-tx.sh sends traffic to the second VM. The script iterates through different packet sizes. Modifications may be made to **unidirectional_transmitter.lua** to change packet sizes that are being sent.

Script: **0_run_dpdk-pktgen_uni-tx.sh**

```
#!/bin/bash
#0_run_dpdk-pktgen_uni-tx.sh

export DPDK_BASE_DIR=/root
export PKTGEN=/root/pktgen-dpdk-pktgen-3.3.2
script_dir="$(dirname $(readlink -f $0))"
cd $PKTGEN

CPU_COUNT=$(cat /proc/cpuinfo | grep processor | wc -l)

#Check for virtIO-relay interfaces on bus 1, otherwise, it will be SR-IOV
interfaces
lspci | grep 01:
if [ $? == 1 ]; then
```



```
NETRONOME_VF_LIST=$(lspci -d 19ee: | awk '{print $1}')
else
NETRONOME_VF_LIST=$(lspci | grep 01: | awk '{print $1}')
fi

memory="--socket-mem 1024"
lcores="-l 0-$((CPU_COUNT-1))"

# whitelist
whitelist=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  whitelist="$whitelist $netronome_vf"
done

# cpumapping
cpu_counter=0
port_counter=0
mapping=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  mapping="${mapping}-m "

       cpu_counter=$((cpu_counter+1))
       echo "cpu_counter: $cpu_counter"
       mapping="${mapping}${cpu_counter}"

  mapping="${mapping}.${port_counter} "
  port_counter=$((port_counter+1))
done

echo "whitelist: $whitelist"
echo "mapping: $mapping"

/root/dpdk-pktgen $lcores --proc-type auto $memory -n 4 --log-level=7 $whitelist
--file-prefix=dpdk0_ -- $mapping -N -f $script_dir/unidirectional_transmitter.lua

reset
```

`1_run_dpdk-pktgen_uni-rx.sh` is run on the second VM to sync traffic that is being transmitted from the first VM.

Script: **1_run_dpdk-pktgen_uni-rx.sh**

```
#!/bin/bash
#1_run_dpdk-pktgen_uni-rx.sh

export DPDK_BASE_DIR=/root
export PKTGEN=/root/pktgen-dpdk-pktgen-3.3.2
script_dir="$(dirname $(readlink -f $0))"
cd $PKTGEN

CPU_COUNT=$(cat /proc/cpuinfo | grep processor | wc -l)

#Check for virtIO-relay interfaces on bus 1, otherwise, it will be SR-IOV
interfaces
lspci | grep 01:
if [ $? == 1 ]; then
NETRONOME_VF_LIST=$(lspci -d 19ee: | awk '{print $1}')
else
NETRONOME_VF_LIST=$(lspci | grep 01: | awk '{print $1}')
fi

memory="--socket-mem 1024"
lcores="-l 0-$((CPU_COUNT-1))"

# whitelist
whitelist=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  whitelist="$whitelist $netronome_vf"
done

# cpumapping
cpu_counter=0
port_counter=0
mapping=""
for netronome_vf in ${NETRONOME_VF_LIST[@]};
do
  echo "netronome_vf: $netronome_vf"
  mapping="${mapping}-m "

       cpu_counter=$((cpu_counter+1))
       echo "cpu_counter: $cpu_counter"
```



```
      mapping="${mapping}${cpu_counter}"

  mapping="${mapping}.${port_counter} "
  port_counter=$((port_counter+1))
done

echo "whitelist: $whitelist"
echo "mapping: $mapping"

/root/dpdk-pktgen $lcores --proc-type auto $memory -n 4 --log-level=7 $whitelist
--file-prefix=dpdk0_ -- $mapping -N -f $script_dir/unidirectional_receiver.lua

reset
```

5.3.14 Test Results

Below is a sample output of a DPDK-pktgen instance running on the Tx VM.

- **`Pkts/s`** - Indicates packets being sent and received
- **`MBits/s`** - Indicates throughput, sending and receiving.
- **`PktSize/Tx Burst`** - Indicates the current packet size
- **`TotalRate`** - Indicates the summation of all traffic being sent and/or received (In this example case, only one port is sending traffic)

```
Ports 0-1 of 2   <Main Page>  Copyright (c) <2010-2016>, Intel Corporation
  Flags:Port        :   ---------------:0   ---------------:1
Link State          :      <UP-10000-FD>       <UP-10000-FD>       ----TotalRate----
Pkts/s Max/Rx       :      5651840/2347680     5690069/2347670     11306279/4695350
       Max/Tx       :      12985312/2347616    12750400/2347648    25505728/4695264
MBits/s Rx/Tx       :             9991/9991           9991/9991           19983/19983
Broadcast           :                     0                   0
Multicast           :                     0                   0
  64 Bytes          :                     0                   0
  65-127            :                     0                   0
  128-255           :                     0                   0
  256-511           :                     0                   0
  512-1023          :             250570331           250571294
  1024-1518         :                     0                   0
Runts/Jumbos        :                   0/0                 0/0
Errors Rx/Tx        :                   0/0                 0/0
Total Rx Pkts       :             411537636           348013420
       Tx Pkts      :             601034880           410701472
       Rx MBs       :             1166181             1123496
```



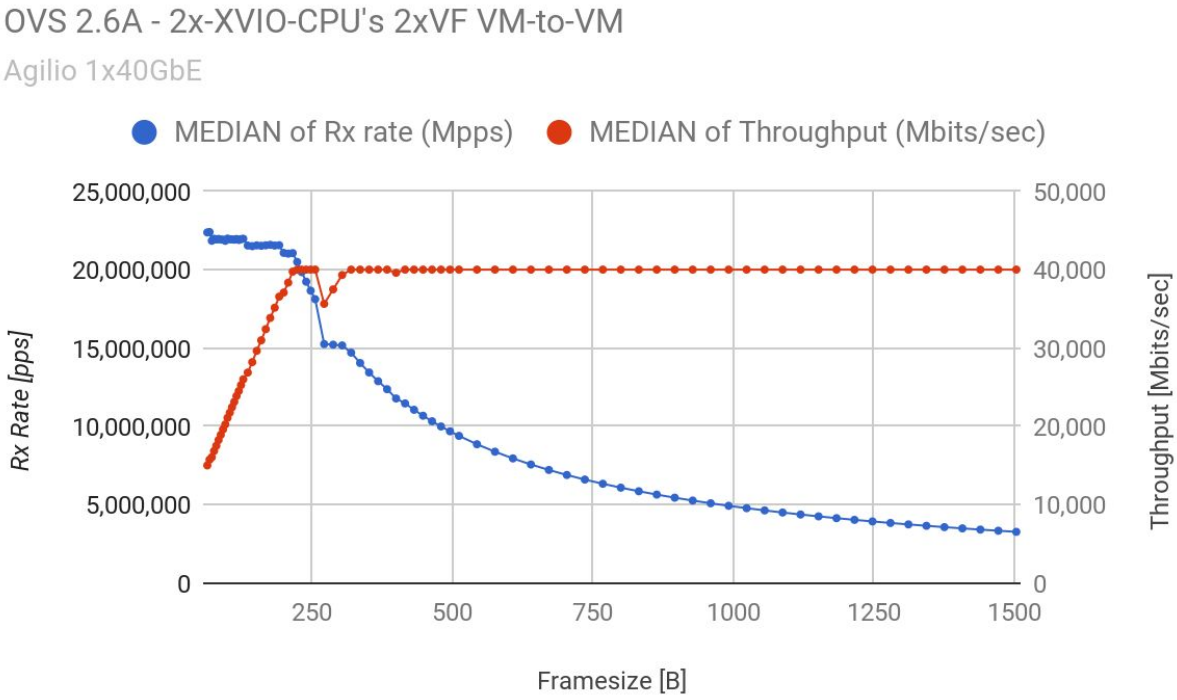
```
      Tx MBs       :            1268441            1140547
ARP/ICMP Pkts      :                  0/0                0/0
                   :
Pattern Type       :            abcd...            abcd...
Tx Count/% Rate    :       Forever / 100%     Forever / 100%
PktSize/Tx Burst   :       512 /   32         512 /   32
Src/Dest Port      :       1234 / 5678        1234 / 5678
Pkt Type:VLAN ID   :              IPv4 / TCP:0001    IPv4 / TCP:0001
Dst  IP Address    :       192.168.2.1        192.168.3.1
Src  IP Address    :       192.168.2.2/24     192.168.3.2/24
Dst MAC Address    :   52:54:00:d0:2b:87   52:54:00:ce:b9:23
Src MAC Address    :   52:54:00:e2:d3:01   52:54:00:04:c1:18
VendID/PCI Addr    :   1af4:1000/00:06.0   1af4:1000/00:07.0

-- Pktgen Ver: 3.0.17 (DPDK 16.11.0)  Powered by Intel® DPDK -----------------
```

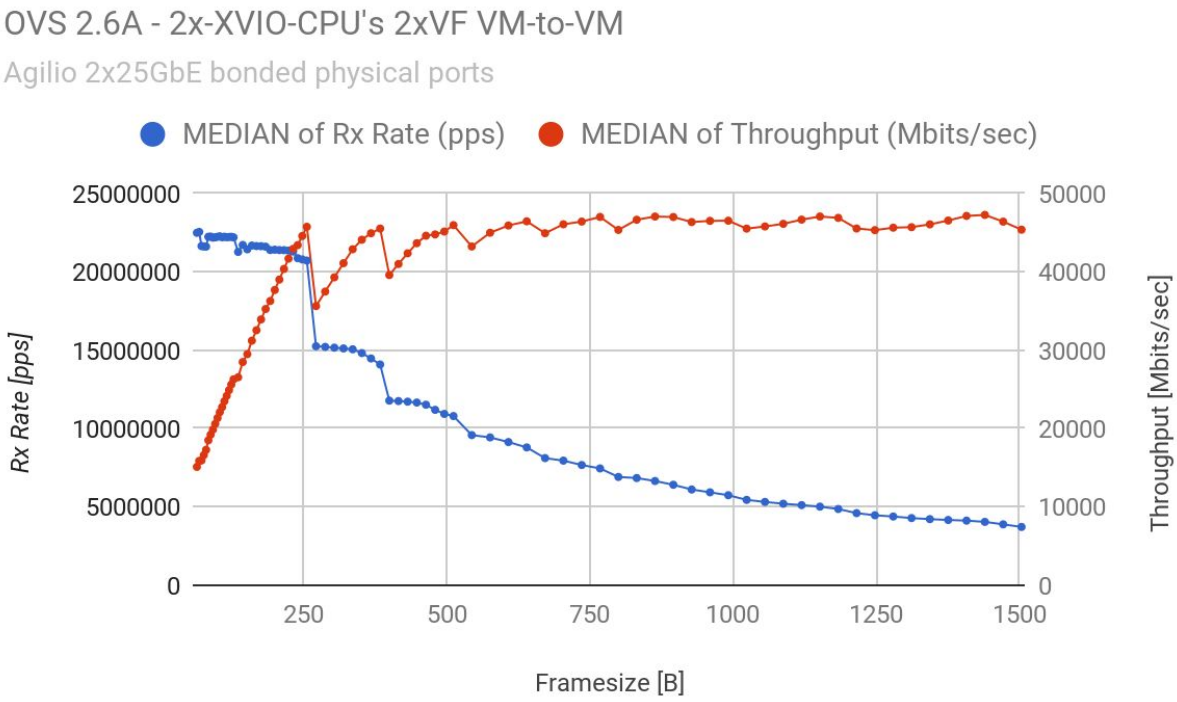
The results of the test run will be saved on VM2 in /root/pktgen-dpdk-pktgen-3.3.2. The results are comma separated and may easily be pasted into a spreadsheet file.

Example output of comma separated file of the data that is logged by DPDK-pktgen on the Rx VM.

```
      Time,  Ports, Framesize,  total_pkts_rx, total_mbits_rx

      7.0,         1,     64,          20198371,          13573
      8.0,         1,     64,          20042486,          13468
      9.0,         1,     64,          20177209,          13559
      10.0,        1,     64,          20218240,          13586
      11.0,        1,     64,          20185959,          13564
      12.0,        1,     64,          20200468,          13574
      13.0,        1,     64,          20328945,          13661
      14.0,        1,     64,          20220310,          13588
      15.0,        1,     64,          20212942,          13583
      16.0,        1,     64,          20219968,          13587
      17.0,        1,     64,          20198027,          13573
      18.0,        1,     64,          20083556,          13496
```

The resulting comma separated file was imported into google sheets (or similar) and a pivot table was created using framesize as rows and rx rate and throughput as columns. The following graphs were generated using this pivot table.





5.4 Test Case 4: External Traffic Generator - SR-IOV Configuration

5.4.1 Prerequisites

This test assumes that the following tasks have already been completed:

- Agilio OVS 2.6A Installation on Ubuntu 16.04
- Installing and configuring KVM and virt-manager
- Virtual Machine Setup
- Performance Tuning

5.4.2 Test Setup

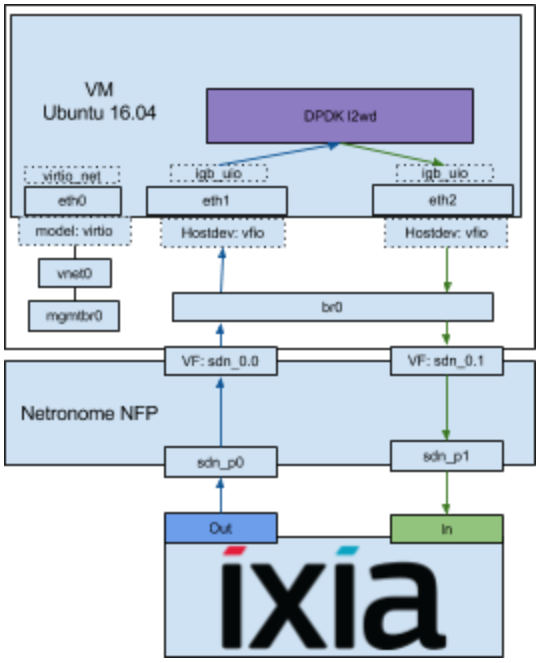


Figure X. RFC2544 SR-IOV

5.4.3 Host: Bind VFIO-PCI driver

Script: **`1_bind_VFIO-PCI_driver.sh`**

```
#!/bin/bash
#1_bind_VFIO-PCI_driver.sh

PCIA="$(ethtool -i sdn_v0.42 | grep bus | cut -d ' ' -f 5)"
PCIA2="$(ethtool -i sdn_v0.44 | grep bus | cut -d ' ' -f 5)"
interface_list=($PCIA $PCIA2)
driver=vfio-pci
# updatedb
DPDK_DEVBIND=$(find /opt/ -iname dpdk-devbind.py | head -1)
echo "loading driver"
modprobe $driver
echo "DPDK_DEVBIND: $DPDK_DEVBIND"
for interface in ${interface_list[@]};
do
  echo $DPDK_DEVBIND --bind $driver $interface
```



```
  $DPDK_DEVBIND --bind $driver $interface
done
echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status
```

5.4.4 Host: Configure AOVS

Script: **2_configure_AVOS.sh**

```
#!/bin/bash
#2_configure_AVOS.sh

echo "CURRENT configuration"
cat /etc/netronome.conf

cat << 'EOF' > /etc/netronome.conf
SDN_VIRTIORELAY_ENABLE=n
SDN_FIREWALL=n
EOF

echo "NEW configuration"
cat /etc/netronome.conf

ovs-ctl status
ovs-ctl stop
ovs-ctl start
ovs-ctl status
```

5.4.5 Host: Configure OVS rules

Script: **3_configure_OVS_rules.sh**

```
#!/bin/bash
#3_configure_OVS_rules.sh

BRIDGE=br0

# Delete all bridges
for br in $(ovs-vsctl list-br);
do
  ovs-vsctl --if-exists del-br $br
done
```



```
# Create a new bridge
ovs-vsctl add-br $BRIDGE

# Add physical ports
for i in $(seq 0 1);
do
  ovs-vsctl add-port $BRIDGE sdn_p${i} -- set interface sdn_p${i}
ofport_request=$((i+1))
done

# Add VF ports
for i in 42 44;
do
  ovs-vsctl add-port $BRIDGE sdn_v0.${i} -- set interface sdn_v0.${i}
ofport_request=$((i))
done

ovs-ofctl del-flows $BRIDGE
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=1,actions=output:42
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=42,actions=output:1
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=2,actions=output:44
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=44,actions=output:2

ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
ovs-vsctl set Open_vSwitch . other_config:flow-limit=1000000
ovs-appctl upcall/set-flow-limit 1000000

ovs-vsctl show
ovs-ofctl show $BRIDGE
ovs-ofctl dump-flows $BRIDGE
```

5.4.6 Host: Configure Guest domain configuration

VM's have to be shutdown before editing their XML files.

```
virsh shutdown <vm name>
```

Add the PCI address of sdn_v0.42 and sdn_v0.44 to the VM

Script: **`4_configure_guest_xml.sh`**


```
#!/bin/bash
#4_configure_guest_xml.sh

VM_NAME="vm1"

# Remove vhostuser interface
EDITOR='sed -i "/<interface type=.vhostuser.>/,/<\/interface>/d"' virsh edit
$VM_NAME
EDITOR='sed -i "/<hostdev mode=.subsystem. type=.pci./,/<\/hostdev>/d"' virsh edit
$VM_NAME

# Add vhostuser interfaces
# sdn_v0.42 --> 0000:81:0d.2
# sdn_v0.44 --> 0000:81:0d.4
bus=$(ethtool -i sdn_v0.42 | grep bus-info | awk '{print $5}' | awk -F ':' '{print
$2}')
EDITOR='sed -i "/<devices/a \<hostdev mode=\"subsystem\" type=\"pci\"
managed=\"yes\">  <source> <address domain=\"0x0000\" bus=\"0x'${bus}'\"
slot=\"0x0d\" function=\"0x2\"\/> <\/source>  <address type=\"pci\"
domain=\"0x0000\" bus=\"0x00\" slot=\"0x06\" function=\"0x0\"\/> <\/hostdev>"'
virsh edit $VM_NAME
EDITOR='sed -i "/<devices/a \<hostdev mode=\"subsystem\" type=\"pci\"
managed=\"yes\">  <source> <address domain=\"0x0000\" bus=\"0x'${bus}'\"
slot=\"0x0d\" function=\"0x4\"\/> <\/source>  <address type=\"pci\"
domain=\"0x0000\" bus=\"0x00\" slot=\"0x09\" function=\"0x0\"\/> <\/hostdev>"'
virsh edit $VM_NAME
```

5.4.6 Host: Boot Guest Machine

```
VM_NAME=vm1
virsh start $VM_NAME
```

5.4.7 Host: SSH into Guest Machine

```
VM_NAME=vm1
ssh root@$(virsh net-dhcp-leases default | awk '/'"$VM_NAME"'/ {print $5}' | cut
-d"/" -f1)

#Default password: changeme
```

5.4.8 Guest: Prepare Guest Machine

Prepare the Guest Machine by following the steps specified in General Guest configuration.
Note: `run 1_configure_hugepages.sh` after a VM restart.

5.4.9 Guest: Bind IGB-UIO driver

Script: **`2_bind_IGB-UIO_driver.sh`**

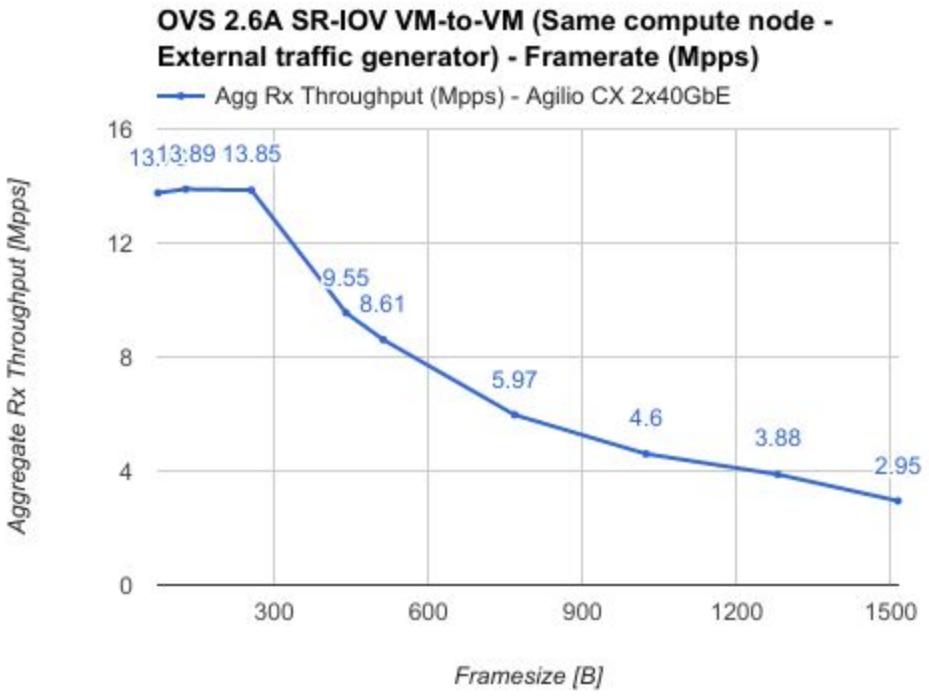
```
#!/bin/bash
#2_bind_IGB-UIO_Driver.sh

interface_list=(00:06.0 00:09.0)
driver=igb_uio
mp=uio
# updatedb
DPDK_DEVBIND=$(find -iname dpdk-devbind.py | head -1)
DRKO=$(find -iname 'igb_uio.ko' | head -1 )
echo "loading driver"
modprobe $mp
insmod $DRKO
echo "DPDK_DEVBIND: $DPDK_DEVBIND"
for interface in ${interface_list[@]};
do
  echo $DPDK_DEVBIND --bind $driver $interface
  $DPDK_DEVBIND --bind $driver $interface
done
echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status
```

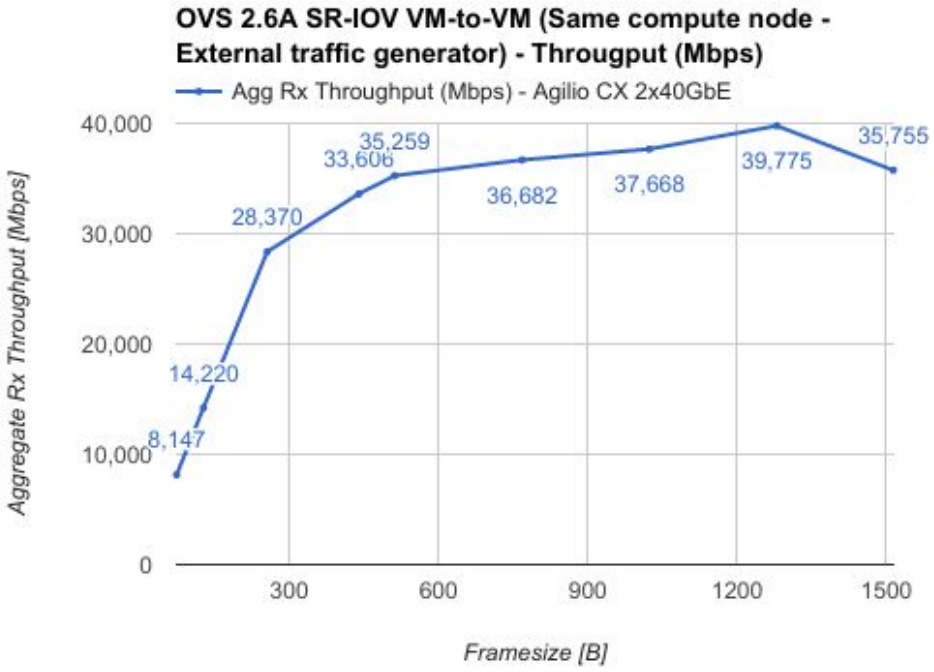
5.4.10 Guest: Configure and run L2FWD

Script: **`3_configure_L2FWD.sh`**

```
#!/bin/bash
#3_configure_L2FWD.sh

#allocate hugepages
echo 512 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages

export DPDK_BASE_DIR=/root

#run l2fwd
$DPDK_BASE_DIR/dpdk-l2fwd -c 0x06 -n 4 --socket-mem 1024 --proc-type auto -- -p 0x3
--no-mac-updating
```


5.4.11 Test Results





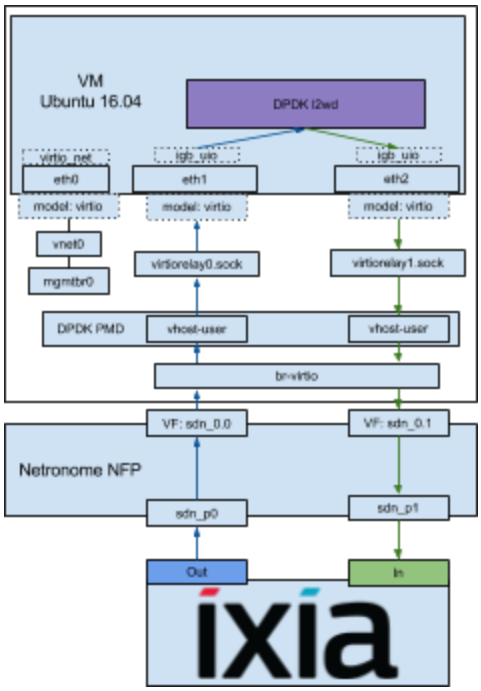
5.5 Test Case 5: External Traffic Generator - Virtio Configuration

5.5.1 Prerequisites

This test assumes that the following tasks have already been completed:

- Agilio OVS 2.6A Installation on Ubuntu 16.04
- Installing and configuring KVM and virt-manager
- Virtual Machine Setup
- Performance Tuning
- General Guest configuration

5.5.2 Configuration setup



5.5.3 Host: Bind IGB-UIO driver

Script: **`1_bind_IGB-UIO_driver.sh`**

```
#!/bin/bash
#1_bind_IGB-UIO_driver.sh

PCIA="$(ethtool -i sdn_v0.42 | grep bus | cut -d ' ' -f 5)"
PCIA2="$(ethtool -i sdn_v0.44 | grep bus | cut -d ' ' -f 5)"
interface_list=($PCIA $PCIA2)
driver=igb_uio
mp=uio
# updatedb
DPDK_DEVBIND=$(find /opt/ -iname dpdk-devbind.py | head -1)
DRKO=$(find /opt/ -iname 'igb_uio.ko' | head -1 )
```



```
echo "loading driver"
modprobe $mp
insmod $DRKO
echo "DPDK_DEVBIND: $DPDK_DEVBIND"
for interface in ${interface_list[@]};
do
  echo $DPDK_DEVBIND --bind $driver $interface
  $DPDK_DEVBIND --bind $driver $interface
done
echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status | grep "$PCIA\|$PCIA2"
```

5.5.4 Host: Configure AOVS

Script: **`2_configure_AOVS.sh`**

```
#!/bin/bash

echo "CURRENT configuration"
cat /etc/netronome.conf

cat << 'EOF' > /etc/netronome.conf
SDN_VIRTIORELAY_ENABLE=y
SDN_VIRTIORELAY_PARAM="--cpus=18,19 --enable-tso --enable-mrgbuf
--vhost-username=libvirt-qemu --vhost-groupname=kvm --huge-dir=/mnt/huge
--ovsdb-sock=/var/run/openvswitch/db.sock"
SDN_FIREWALL=n
EOF

echo "NEW configuration"
cat /etc/netronome.conf

ovs-ctl status
ovs-ctl stop
ovs-ctl start
ovs-ctl status
```


5.5.5 Host: Configure OVS rules

Script: **3_configure_OVS_rules.sh**

```
#!/bin/bash

BRIDGE=br0

# Delete all bridges
for br in $(ovs-vsctl list-br);
do
  ovs-vsctl --if-exists del-br $br
done

# Create a new bridge
ovs-vsctl add-br $BRIDGE

# Add physical ports
for i in $(seq 0 1);
do
  ovs-vsctl add-port $BRIDGE sdn_p${i} -- set interface sdn_p${i}
ofport_request=$((i+1))
done

# Add VF ports
for i in 42 44;
do
  ovs-vsctl add-port $BRIDGE sdn_v0.${i} -- set interface sdn_v0.${i}
ofport_request=$((i)) external_ids:virtio_relay=$((i))
done

#d=$(locate dpdk-devbind.py | head -1)
#echo "Using $d"
#modprobe igb_uio
#for i in 42 44;
#do
#  echo "sdn_v0.${i} -> $(ethtool -i sdn_v0.${i} | grep bus-info | awk '{print
$5}')"
#  $d --bind igb_uio $(ethtool -i sdn_v0.${i} | grep bus-info | awk '{print $5}')
#done

ovs-ofctl del-flows $BRIDGE
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=1,actions=output:42
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=42,actions=output:1
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=2,actions=output:44
```



```
ovs-ofctl -O OpenFlow13 add-flow $BRIDGE in_port=44,actions=output:2
ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
ovs-vsctl set Open_vSwitch . other_config:flow-limit=1000000
ovs-appctl upcall/set-flow-limit 1000000

ovs-vsctl show
ovs-ofctl show $BRIDGE
ovs-ofctl dump-flows $BRIDGE
```

5.5.6 Host: Configure Apparmor

Script: **`4_configure_apparmor.sh`**

```
#!/bin/bash

ADD="$(grep -n 'libxl-save-helper' /etc/apparmor.d/usr.sbin.libvirtd | cut -d:
-f1)"
OUT="$(grep -n '/usr/local/bin/qemu* PUx,' /etc/apparmor.d/usr.sbin.libvirtd | cut
-d: -f1)"
[ -z "$OUT" ] && sed -i "$ADD a /usr/local/bin/qemu* PUx, #ADD OVS"
/etc/apparmor.d/usr.sbin.libvirtd
OUT="$(grep -n '/usr/local/bin/ovs-vsctl rmix,' /etc/apparmor.d/usr.sbin.libvirtd |
cut -d: -f1)"
[ -z "$OUT" ] && sed -i "$ADD a /usr/local/bin/ovs-vsctl rmix, #ADD OVS"
/etc/apparmor.d/usr.sbin.libvirtd

OUT="$(grep -n "deny /tmp/" /etc/apparmor.d/abstractions/libvirt-qemu | cut -d:
-f1)"
sed -i "$OUT s/^/#/" /etc/apparmor.d/abstractions/libvirt-qemu
OUT="$(grep -n "deny /var/tmp/" /etc/apparmor.d/abstractions/libvirt-qemu | cut -d:
-f1)"
sed -i "$OUT s/^/#/" /etc/apparmor.d/abstractions/libvirt-qemu

sudo service apparmor reload
```

5.5.7 Host: Configure Guest XML

Script: **`5_configure_guest_XML.sh`**

```
#!/bin/bash
#5_configure_guest_XML.sh
```



```
VM_NAME="vm1"
VM_CPU=6
max_memory=$(virsh dominfo $VM_NAME | grep 'Max memory:' | awk '{print $3}')

# Remove vhostuser interface
EDITOR='sed -i "/<interface type=.vhostuser.>/,/<\/interface>/d"' virsh edit
$VM_NAME
EDITOR='sed -i "/<hostdev mode=.subsystem. type=.pci./,/<\/hostdev>/d"' virsh edit
$VM_NAME

# Add vhostuser interfaces
# sdn_v0.42 --> 0000:81:0d.2
# sdn_v0.44 --> 0000:81:0d.4
bus=$(ethtool -i sdn_v0.42 | grep bus-info | awk '{print $5}' | awk -F ':' '{print
$2}')
# Add vhostuser interfaces
EDITOR='sed -i "/<devices/a \<interface type=\"vhostuser\">  <source type=\"unix\"
path=\"/tmp/virtiorelay42.sock\" mode=\"client\"\/>  <model type=\"virtio\"/>
<driver name=\"vhost\" queues=\"1\"\/>  <address type=\"pci\" domain=\"0x0000\"
bus=\"0x00\" slot=\"0x06\" function=\"0x0\"\/><\/interface>"' virsh edit $VM_NAME
EDITOR='sed -i "/<devices/a \<interface type=\"vhostuser\">  <source type=\"unix\"
path=\"/tmp/virtiorelay44.sock\" mode=\"client\"\/>  <model type=\"virtio\"/>
<driver name=\"vhost\" queues=\"1\"\/>  <address type=\"pci\" domain=\"0x0000\"
bus=\"0x00\" slot=\"0x09\" function=\"0x0\"\/><\/interface>"' virsh edit $VM_NAME

EDITOR='sed -i "/<numa>/,/<\/numa>/d"' virsh edit $VM_NAME
EDITOR='sed -i "/vcpu/d"' virsh edit $VM_NAME
EDITOR='sed -i "/<cpu/,/<\/cpu>/d"' virsh edit $VM_NAME
EDITOR='sed -i "/<memoryBacking>/,/<\/memoryBacking>/d"' virsh edit $VM_NAME

virsh setvcpus $VM_NAME $VM_CPU --config --maximum
virsh setvcpus $VM_NAME $VM_CPU --config
# MemoryBacking
EDITOR='sed -i "/<domain/a \<memoryBacking><hugepages><page size=\"2048\"
unit=\"KiB\" nodeset=\"0\"\/><\/hugepages><\/memoryBacking>"' virsh edit $VM_NAME
# CPU
echo max_memory: $max_memory
echo VM_CPU: $VM_CPU
EDITOR='sed -i "/<domain/a \<cpu mode=\"host-model\"><model
fallback=\"allow\"\/><numa><cell id=\"0\" cpus=\"0-'$((VM_CPU-1))'\"
memory=\"'${max_memory}'\" unit=\"KiB\" memAccess=\"shared\"\/><\/numa><\/cpu>"'
virsh edit $VM_NAME
```


5.5.8 Host: Boot Guest Machine

```
VM_NAME=vm1
virsh start $VM_NAME
```

5.5.9 Host: SSH into Guest Machine

```
VM_NAME=vm1
ssh root@$(virsh net-dhcp-leases default | awk '/'"$VM_NAME"'/ {print $5}' | cut
-d"/" -f1)

#Default password: changeme
```

5.5.10 Guest: Prepare Guest Machine

Prepare the Guest Machine by following the steps specified in General Guest configuration.
Note: **`run 1_configure_hugepages.sh`** after a VM restart.

5.5.11 Guest: Bind IGB-UIO driver

Script: **`2_bind_IGB-UIO_driver.sh`**

```
#!/bin/bash
#2_bind_IGB-UIO_driver.sh

interface_list=(00:06.0 00:09.0)
driver=igb_uio
mp=uio
# updatedb
DPDK_DEVBIND=$(find ~ -iname dpdk-devbind.py | head -1)
DRKO=$(find ~ -iname 'igb_uio.ko' | grep dpdk-16.11 | head -1)
echo "loading driver"
modprobe $mp
insmod $DRKO
echo "DPDK_DEVBIND: $DPDK_DEVBIND"
for interface in ${interface_list[@]};
do
```



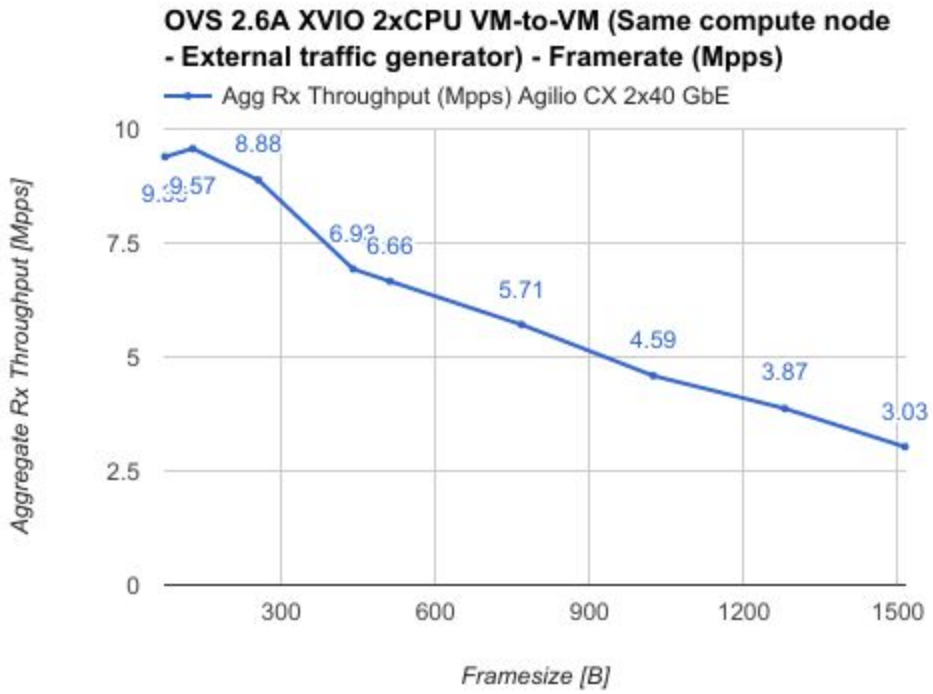
```
  echo $DPDK_DEVBIND --bind $driver $interface
  $DPDK_DEVBIND --bind $driver $interface
done
echo $DPDK_DEVBIND --status
$DPDK_DEVBIND --status
```

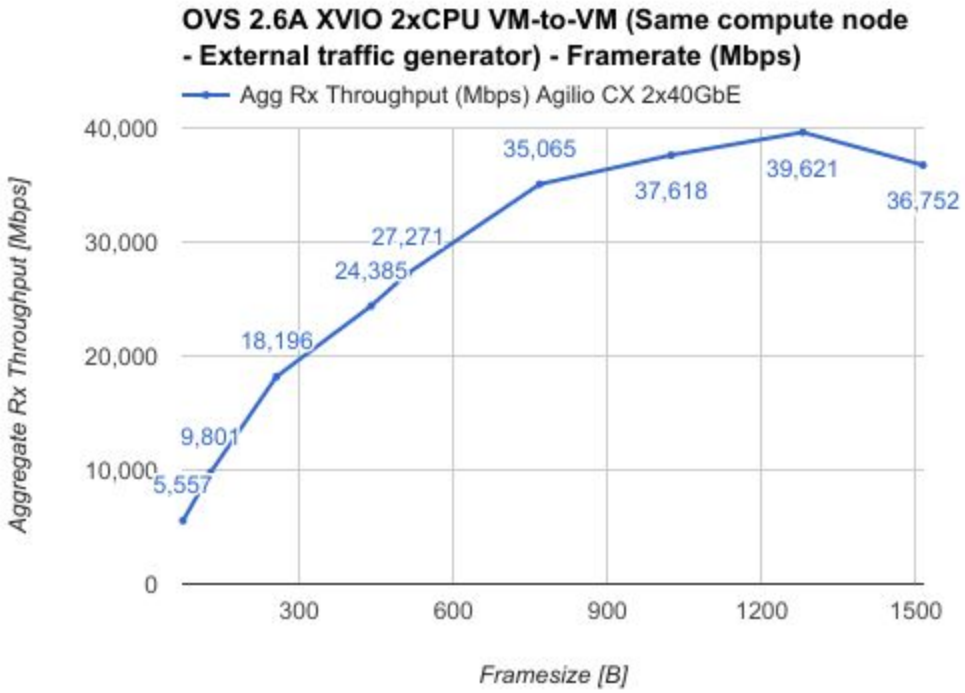
5.5.12 Guest: Configure and run L2FWD

Script: **`3_configure_l2fwd.sh`**

```
#!/bin/bash
#3_configure_l2fwd.sh

#run l2fwd
./dpdk-l2fwd  -c 0x03 -n 4 --socket-mem 1024 --proc-type auto -- -p 0x3
--no-mac-updating
```

9.5.13 Test Results





Appendix A : Quick Reference - Troubleshooting

Command	Function	Options
Resources		
htop	View core and memory utilization	
lscpu	Display core/Numa information	
cat /sys/bus/pci/drivers/nfp/0*/numa_node	View NUMA node associated with NFP	
AOVS		
ovs-ctl status	Display OVS status	
/opt/netronome/bin/virtio_relay_stats	Display XVIO statistics	
cat /sys/module/nfp_offloads/control/rh_entries	Display offloaded flow cache rules	
ovs-ctl status troubleshoot <option>	Display OVS/NFP statistics	-no option (Generate full report) -i (view deltas)
ovs-ofctl dump-flows <bridge>	View OF rules	
ovs-dpctl dump-flows <bridge>	View DP rules	
Interface		
ethtool <option> <interface>	Set/view port parameters	-i (port information) -S (traffic statistics) -k (view port configuration) -K (set port configuration)
Logs		
journalctl -an 10000		
tail -1000 /var/log/openvswitch/ovs-vswitchd.log (package-based install)		

tail -1000 /usr/local/var/log/openvswitch/ovs-vswitchd.log (source-based install)		

Appendix B : Quick Reference - General Ubuntu Commands

Command	Function	Options
General		
chmod +x <script.sh>	Set script's execution flag	
Interface management		
ip link show <interface>	View status of link	
ip link set device <interface> up/down	Change status of link	
ip address show <interface>	View interface addresses	
ip address add <IP>/<mask> device <interface>	Add IP address to interface	
ip address del <IP>/<mask> device <interface>	Remove IP address from interface	
OVS general		
ovs-ctl start/stop/restart	General OVS	
ovs-vsctl add-br <bridge>	Create new OVS bridge	
ovs-vsctl del-br <bridge>	Delete OVS bridge	
ovs-vsctl add-pr <bridge> <port>	Add port to OVS bridge	
ovs-vsctl del-pr <bridge> <port>	Remove port from OVS bridge	
ovs-ofctl show <bridge>	View port indexes	
ovs-dpctl show		
ovs-appctl show <bridge>	View bond status	
VM		
virsh list	List VMs	-no option (List running VMs) --all (List all defined VMs)

`virsh vcpupin <vm_id>`	`List vcpu pinning`	
`virsh vcpupin <vm_id> <vcpu> <hcpu>`	`Pin vcpus`	

Appendix C : BIOS Settings Explained

Intel Virtualization Technology

Intel Virtualization Technology (Intel VT) represents a growing portfolio of technologies and features that make virtualization practical by eliminating performance overheads and improving security. Intel VT provides hardware assist to the virtualization software, reducing its size, cost, and complexity.

CPU virtualization features enable faithful abstraction of the full prowess of Intel CPU to a virtual machine (VM). All software in the VM can run without any performance or compatibility hit, as if it was running natively on a dedicated CPU. Live migration from one Intel CPU generation to another, as well as nested virtualization, is possible.

The recommended setting is: **enabled**.

Hyper-threading

Intel Hyper-Threading Technology uses processor resources more efficiently, enabling multiple threads to run on each core. As a performance feature, it also increases processor throughput, improving overall performance on threaded software. Intensive workloads competing for processor time allocation on hyperthread siblings can result in severe latency penalties.

The recommended setting is: **disabled.**

Turbo Boost

Turbo Boost has a series of dynamic frequency scaling technologies enabling the processor to run above its base operating frequency via dynamic control. Although Turbo Boost (overclocking) would increase the performance, not all cores would be running at the turbo frequency and it would impact the consistency of the performance tests, thus it would not be a real representation of the platform performance.

The recommended setting is: **disabled.**

C-state

Each CPU has several power modes, and they are collectively called “C-states” or “C-modes”, in order to save energy when the CPU is idle, the CPU can be commanded to enter a low-power mode. Ensure that the CPUs are always in the C0 (fully operational) state - disable CPU C3 Report, CPU C6 Report, and Enhanced Halt State C1E).

The recommended setting is: **C0 (fully operational)**.

P-state

The processor P-state is the capability of running the processor at different voltage and/or frequency levels. Generally, P0 is the highest state resulting in maximum performance, while P1, P2, and so on, will save power but at some penalty to CPU performance. Disabling EIST (Enhanced SpeedStep) will also disable TurboBoost.

The recommended setting is: **disabled.**

Contact us

Netronome Systems, Inc.
2903 Bunker Hill Lane, Suite 150 Santa Clara, CA 95054
Tel: 408.496.0022 | Fax: 408.586.0002
www.netronome.com