NETRONOME

TECHNICAL PAPER

OVS Test Plan

BENCHMARKING TEST SUITE

THIS DOCUMENT GOES THROUGH A LIST OF BENCHMARK TESTS FOR OPEN VSWITCH ACCELERATED BY THE NETRONOME AGILIO-CX-4000 INTELLIGENT SERVER ADAPTER, TO COMPARE AGAINST NON-ACCELERATED OPEN VSWITCH RUNNING ENTIRELY IN SOFTWARE ON THE SERVER WITH A BASIC NIC.

CONTENTS

METHODOLOGY

This chapter describes the general methods used for testing the Agilio-CX-4000 intelligent server adapter as well as software based OVS.

Setup

The following hardware and software was used for the device under test:

Server: Dell PowerEdge R730

CPU: Intel Xeon CPU E5-2670 v3 @2.3Ghz with 64GB of DRAM

Operating System: Ubuntu 14.04.3 LTS

Kernel Version: 3.13.0-57-generic

Open vSwitch: v2.3.90

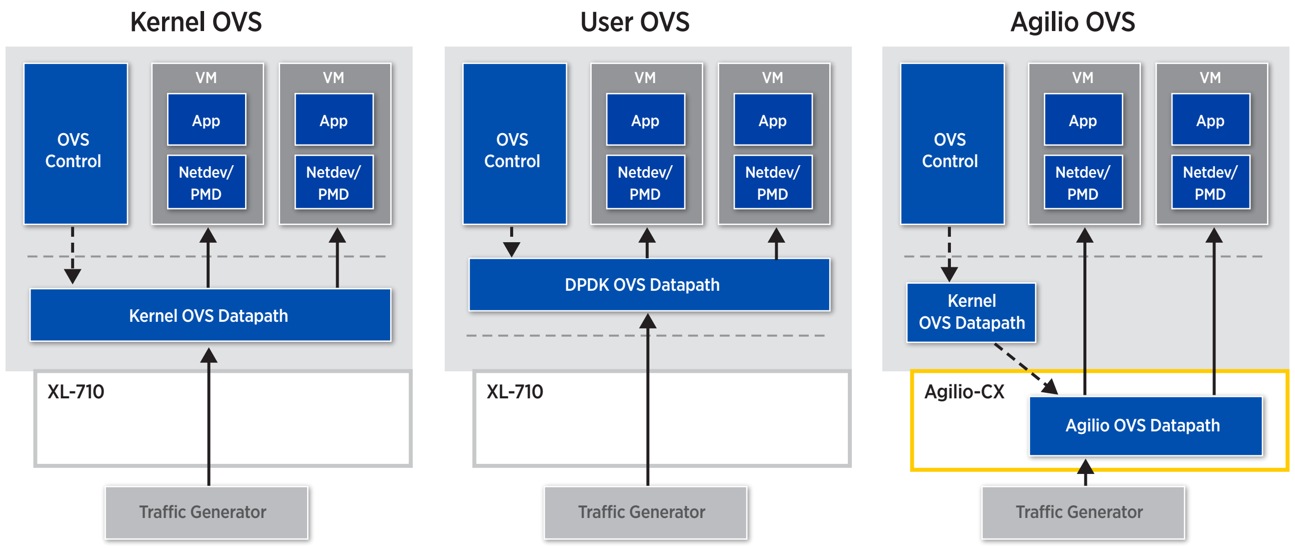
Server Adapters:

Intel XL710 for software (Kernel and User) OVS

Agilio®-CX4000-40Q-10 with LNOD2.1 for accelerated OVS

Test Generator: IXIA/Spirent

The test setups used for Kernel and User software OVS on the XL710 and accelerated OVS on the Agilio®-CX4000 are shown in the diagrams below:



In this test plan, a separate host is used for traffic generation/termination. Alternatively, an IXIA or Spirent Traffic generator may also be used, with appropriate test scripts.

Test Traffic Profile

Following is a description of the traffic patterns used for all testing described within this document. Note that when multiple traffic generators are deployed, such as when generating traffic within the VMs, the same pattern will be generated by each traffic generator instance.

A total of 64,000 flows will be generated by each instance. The generator will send one packet per flow for the first 2,000 flows and will then repeat these same 2,000 flows 10 times. Next it will in a similar fashion sequence through the second set of 2,000 flows 10 times. This will continue until all 64,000 flows have been covered at which point the sequence will start all over again.

The above described traffic pattern can be regarded as a realistic representation of real traffic where flows are very active for a short time and then disappear.

Wild Card Rules

For every test, OVS is configured with 1,000 wild card rules either matching against L2 fields or L3 fields.

L2 Wild Card Rules and Actions

The L2 rules match against the input port and a contiguous set of bits in the destination MAC address, aligned to the least significant end. The first 512 rules match against the nine least significant bits, while the remaining rules match against longer bit strings, thus overlapping with the first group.

The action is an output port. For the network-to-VM use case, the rules act as a load-balancer towards the VMs. For the VM-to-network use case, all traffic is directed towards the physical port (or the VXLAN tunnel). For the VM-to-VM use case, the 1,000 rules connect the four VMs in a full mesh. In other words, traffic from any VM will be evenly distributed (based on the destination MAC address) to the other three VMs.

L3 Wild Card Rules and Actions

The L3 rules match against input port and a contiguous set of bits in the destination IP address, aligned to the least significant bits.

The action of these rules is to modify both the MAC addresses, decrement the IP header TTL, and direct the packets to an output port in a similar fashion as the L2 rules.

Measurements

For each use case, measurements of both the packet rate and bit rate will be taken. The traffic will have a fixed packet size, where separate measurements will be taken for the frame sizes 64, 128, 256, 512, 1024, and 1514 bytes.

Further, all the above measurements will be taken for both the L2 and the L3 rule sets.

Finally, except for the VM-to-VM use case, measurements will be taken with both VXLAN enabled and disabled.

Traffic Generation Test Tool

A Netronome-developed DPDK-based test tool, named trafgen, is used for generating traffic but also for terminating traffic and for collecting statistics. This tool has a range of command line options for specifying parameters like:

- Mode (traffic generation, echo traffic, terminate traffic),
- Test Duration,
- MAC addresses,
- IP addresses
- Traffic rate
- Packet size
- Flow profile

USE CASES

Three different basic use cases are tested.

Use Case: Network to VM

In this test case, traffic will be received on the physical port of the DUT and distributed to four VMs based on the rule set. Each VM will run the Netronome-developed DPDK packet analysis tool, which will report the received bit rate and packet rate.

Use Case: VM to Network

In this test, traffic is generated within a set of four VMs and directed out via the DUT's physical port via the rule set.

Traffic is generated by the Netronome-developed packet generator tool trafgen via a DPDK port.

On the DUT host, a common OVS bridge will be used and all rules of the rule set will direct the traffic towards the physical 40 GE port.

Use Case: VM to VM

In this test, traffic is exchanged between VMs running within the DUT host.

The traffic is generated and terminated by the Netronome-developed test tool trafgen, which is connected to the DPDK port inside of the VM.

The virtual function ports will be connected to one shared OVS bridge on the host and all traffic will be matched against exactly one wild-card rule. The rules in the rule set will implement a full mesh, with traffic from one VM distributed to the other three VMs.

MEASUREMENT RESULTS

For each use case, and each traffic profile (L2, L3, VLAN, and VXLAN), packets-per-second throughput is measured and recorded for the stated packet sizes. Results are compared between Agilio OVS and software OVS (both kernel and user modes).

SCRIPTS AND METHODS

This section contains the scripts and commands needed for configuring the various use cases.

Before running any of these it is assumed that the Agilio-CX 4000 and the LNOD have been properly installed on a server, that VMs have been created, and finally, that LNOD has been installed on the VMs. Information about how to do this can be found in the LNOD User's Guide.

Preparing the VMs

As specified above, this section assumes that the LNOD software has been installed on all VMs.

Building DPDK Test Application inside a VM

Script name: build-vm-dpdk-apps.sh

The following script, which is run on the host, will copy the source code of DPDK applications and build them inside of a VM.

The steps involved:

- Copy the source code of the trafgen tool onto the target.
- Copy the source code of general DPDK example applications onto the target.
- Create a script-string of actions to take on the target:
  - Set the RTE_SDK and RTE_TARGET variables.
  - Set the RTE_OUTPUT to a build directory and create this directory.
  - Build the tool with make.
  - Copy the resulting executable to /usr/local/bin.
- Execute the script-string on the target.

```
#!/bin/bash

# IP address of VM
vmipa="$1"
shift 1
# List of DPDK applications
applist="$*"

########################################
# Usage:
```

```
#
# build-vm-dpdk-apps.sh 10.0.0.1 trafgen l2fwd
#
########################################

if [ "$1" == "" ] || [ "$1" == "--help" ]; then
  echo "Usage: <vmipa> <application> [<application> ...]"
  exit 0
fi

########################################

# The 'trafgen' application is in pkgs/dpdk/src
rsync -aq -R /opt/netronome/samples/dpdk/./trafgen $vmipa:dpdk/src

# Copy Standard DPDK example applications to VM
rsync -aq /opt/netronome/srcpkg/dpdk-ns/examples/* $vmipa:dpdk/src

scr=""
scr="$scr export RTE_SDK=/opt/netronome/srcpkg/dpdk-ns &&"
scr="$scr export RTE_TARGET=x86_64-native-linuxapp-gcc &&"
for appname in $applist ; do
  tooldir="dpdk/src/$appname"
  scr="$scr echo 'Build '$appname &&"
  scr="$scr export RTE_OUTPUT=\$HOME/.cache/dpdk/build/$appname &&"
  scr="$scr mkdir -p \$RTE_OUTPUT &&"
  scr="$scr make --no-print-directory -C \$HOME/$tooldir &&"
  scr="$scr cp \$RTE_OUTPUT/$appname /usr/local/bin &&"
done
scr="$scr echo 'Success'"

exec ssh $vmipa "$scr"
```

Setup Virtual Function as a DPDK port inside of a VM

Script name: setup-vm-vf-iface.sh

This script will attach a specific driver to the virtual function inside of a VM. For turning a VF into a DPDK port, then specify the interface driver (ifdrv) to be nfp_uio. On the other hand, if one wants to use the VF as a Linux netdev, then use the nfp_netvf driver.

The steps involved:

- Create a script-string of actions to take on the target:
  - Do a modprobe on the kernel driver to make sure that it is loaded.
  - Extract the 'bus/dev/func' information via the ethtool.
  - Remove any IP address from the interface and set it to DOWN.
  - Use the dpdk_nic_bind.py script to bind the specified driver to the VF.
  - Save the 'bus/dev/func' information to a file on the VM.
- Execute the script-string on the target.

```
#!/bin/bash

# IP address of VM
vmipa="$1"
# Interface name inside of VM of SR-IOV virtual function
vmifname="$2"
```

```
# Interface device driver to attach to the virtual function
ifdrv="$3"

# Location of DPDK bind tool
bind="/opt/netronome/srcpkg/dpdk-ns/tools/dpdk_nic_bind.py"

scr=""
# Make sure the driver is loaded into the kernel
scr="$scr modprobe $ifdrv &&"
# Extract the PCI 'bus/device/function' information of the 'NFP' Virtu-
al Function
scr="$scr bdf=\$($bind --status | sed -rn 's/^(\S+)\s.*Device\
s6003.*$/\1/p') &&"
# Make sure IPv4 is disable on interface and set it to admin-DOWN
scr="$scr grep $vmifname /proc/net/dev > /dev/null"
scr="$scr   && ifconfig $vmifname 0 down ;"
# Bind the driver to the virtual-function
scr="$scr $bind -b $ifdrv \$bdf ;"
# Save the 'bus/device/function' information for other scripts
scr="$scr echo \$bdf > /var/opt/bdf-ns-vf.txt ;"

# Remotely login to the VM and execute the above commands
ssh $vmipa "$scr"
```

Setup HugePages on a VM

Script name: setup-vm-hugepages.sh

This script will attach configure hugepages on a target VM.

The steps involved:

- Create a script-string of actions to take on the target:
  - Create the mount point /mnt/huge.
  - Mount the hugetlbfs onto this mount point.
  - Reserve 512 hugepages.
- Execute the script-string on the target.

```
#!/bin/bash

# Setup hugepages feature on VM

ipaddr="$1"

scr=""
scr="$scr mkdir -p /mnt/huge ;"
scr="$scr ( grep hugetlbfs /proc/mounts > /dev/null ||"
scr="$scr mount -t hugetlbfs huge /mnt/huge ) ;"
scr="$scr echo 512 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_huge-
pages"

ssh $ipaddr "$scr"
```

Preparing the VMs

The following script assumes that all VMs are running and accessible via SSH. It further assumes that a virtual function has been exported (PCI pass-through) into the VM and that the associated interface has been named 'int'. Further, this script makes use of the VM-specific scripts above.

```
# Example of a VM IP address list (adjust as needed)
vmipaddrlist="10.0.0.10 10.0.0.11 10.0.0.12 10.0.0.13"

for ipaddr in $vmipaddrlist ; do
  build-vm-dpdk-apps.sh $ipaddr trafgen
  setup-vm-vf-iface.sh $ipaddr int nfp_uio
  setup-vm-hugepages.sh $ipaddr
done
```

VM Traffic Generation/Termination Scripts

Traffic Analyzer Test Tool Start Script

Script name: start-vm-dpdk-sink.sh

This script, started from within a VM, starts a Netronome-provided DPDK traffic sink application. It counts received packets and does also estimate the packet rate.

```
#!/bin/bash

ipa="$1"

if [ ! -f /var/opt/bdf-ns-vf.txt ]; then
  echo "ERROR: Missing PCIe bus/device/func file"
  echo "       First use: setup-vm-vf-iface.sh <vm ipa> int nfp_uio"
  exit -1
fi

# Compose the full command line
cmd="trafgen"
cmd="$cmd -n 1 -c 3"
cmd="$cmd -d /opt/netronome/lib/librte_pmd_nfp_net.so"
cmd="$cmd -w $(cat /var/opt/bdf-ns-vf.txt)"
cmd="$cmd --"
cmd="$cmd -p 1"

# Capture the full command line into a file (for debug)
echo $cmd > /tmp/cmdline-pkt-sink

exec $cmd
```

Traffic Generator Test Tool Start Script

Script name: start-vm-dpdk-source.sh

This script, started from within a VM, starts a Netronome-provided DPDK traffic generator application. The scripts configures the traffic generator with pertinent parameters for the tests performed within the scope of this document. The only variable is the packet size which needs to be specified on the command line. In addition to generating traffic, this tool also counts received packets.

```
#!/bin/bash

pktsize="$1"

if [ ! -f /var/opt/bdf-ns-vf.txt ]; then
  echo "ERROR: Missing PCIe bus/device/func file"
  echo "       First use: setup-vm-vf-iface.sh <vm ipa> int nfp_uio"
  exit -1
fi

if [ "$pktsize" == "" ]; then
  echo "ERROR: please specify packet size"
  exit -1
fi

cmd="trafgen"
# DPDK EAL configuration
cmd="$cmd -n 4 -c 3"
cmd="$cmd --socket-mem 256"
cmd="$cmd --proc-type auto"
cmd="$cmd --file-prefix trafgen_source_"
cmd="$cmd -d /opt/netronome/lib/librte_pmd_nfp_net.so"
cmd="$cmd -w $(cat /var/opt/bdf-ns-vf.txt)"
# Delimiter between EAL arguments and application arguments
cmd="$cmd --"
# Port bit-mask
cmd="$cmd --portmask 1"

# Benchmark Mode
cmd="$cmd --benchmark"

# Count (duration of test in seconds, unspecified: indefinitely)
#cmd="$cmd --runtime 3599"
# Ethernet and IP parameters
cmd="$cmd --src-mac 00:11:22:33:44:00"
cmd="$cmd --dst-mac 00:44:33:22:11:00"
cmd="$cmd --src-ip 1.0.0.0"
cmd="$cmd --dst-ip 2.0.0.0"
# Packet Size
cmd="$cmd --packet-size $pktsize"
# Packet Rate (0: full rate)
cmd="$cmd -r 0"
# Transmit Burst Size {1..128} (DPDK: tx_burst_size, default: 32)
cmd="$cmd -t 16"

# Flows within 'Stream' (flows_per_stream, default: 65536)
cmd="$cmd --flows-per-stream 2000"
# Number of Streams (number_of_streams, default: 1)
cmd="$cmd --streams 8"
# Number of Repeats of Stream
cmd="$cmd --bursts-per-stream 10"

# Save command line to a file
echo "$cmd" > /tmp/cmdline-pkt-src

# Run command
exec $cmd
```

Agilio-Based Traffic Generator Scripts

One option for traffic generation/termination is to use a second host with an Agilio-CX 4000 card and LNOD installed.

Setup script

Script name: setup-traffic-generator-server.sh

This first script prepares the Traffic Generator Server for either traffic generation or termination. It is assumed that the test tool source code is in the $HOME/trafgen directory.

By specifying the VXLAN on the command line, the script will configure a VXLAN tunnel over the physical interface.

```
#!/bin/bash

# This script will build the Netronome traffic test tool ‘trafgen’
# and prepare OVS and DPDK for either traffic generation or termination.

if [ “$1” == “VXLAN” ]; then
  VXLAN=”yes”
fi

############################################################
# Build Test-tool application

export RTE_SDK=”/opt/netronome/srcpkg/dpdk-ns”
export RTE_TARGET=”x86_64-native-linuxapp-gcc”
export RTE_OUTPUT=”$HOME/.cache/dpdk/trafgen”
mkdir -p $RTE_OUTPUT
make -C $HOME/trafgen
cp -f $RTE_OUTPUT/trafgen /usr/local/bin

############################################################
# Clean-up all existing OVS bridges

for brname in $(ovs-vsctl list-br) ; do
  ovs-vsctl del-br $brname
  sleep 0.5
done

############################################################
# Setup ‘Huge Pages’
mkdir -p /mnt/huge
#grep hugetlbfs /proc/mounts \
#  || mount -t hugetlbfs huge /mnt/huge

# Make sure some hugepages are allocated
echo 4096 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages

############################################################
# Setup the bridge and attach the physical port

ovs-vsctl add-br br0 \
  -- set Bridge br0 protocols=OpenFlow13

# Flush default NORMAL rule
ovs-ofctl -O OpenFlow13 del-flows br0

ifconfig sdn_p0 0
ifconfig sdn_p0 down
ifconfig sdn_p0 mtu 2000

if [ "$VXLAN" != "" ]; then
  ifconfig sdn_p0 hw ether 02:11:11:00:00:02
  ifconfig sdn_p0 10.1.1.2/24
  ifconfig sdn_p0 up
  # Add Static ARP entry
  arp -i sdn_p0 -s 10.1.1.1 02:11:11:00:00:01
  ovs-vsctl \
    -- --may-exist add-port br0 vxlt \
    -- set Interface vxlt ofport_request=1 \
    -- set interface vxlt type=vxlan \
       options:local_ip=10.1.1.2 \
       options:remote_ip=10.1.1.1 \
       options:key=1
else
  ifconfig sdn_p0 up
  # Add Physical Port (=1)
  ovs-vsctl add-port br0 sdn_p0 \
    -- set Interface sdn_p0 ofport_request=1
fi

############################################################
# Attach virtual-function ports to bridge

whitelist=""
ofpgrp=""
for idx in $(seq 0 3) ; do
  iface="sdn_v0.$idx"
  # OpenFlow Port Index
  ofpidx=$(( 10 + $idx ))
  # (Domain)/Bus/Device/Function
  bdf=$(ethtool -i $iface | sed -rn 's/^bus-info:.*\s(.*)$/\1/p')
  $RTE_SDK/tools/dpdk_nic_bind.py -b nfp_uio $bdf
  ovs-vsctl add-port br0 $iface \
    -- set Interface $iface ofport_request=$ofpidx
  # Egress Rule (for generated traffic)
  ovs-ofctl -O OpenFlow13 add-flow br0 \
    "in_port=$ofpidx,action=output:1"
  # Ingress Load-balancing list (for terminating traffic)
  ofpgrp="$ofpgrp,bucket=actions=output:$ofpidx"
done

# Configure group
ovs-ofctl -O OpenFlow13 add-group $brname \
  "group_id=1,type=select$ofpgrp"

# Load-balance all traffic received on the physical port ('1')
ovs-ofctl -O OpenFlow13 add-flow $brname \
  "in_port=1,actions=group:1"
```

Generate/Terminate Traffic Script

Script name: gen-traffic.sh

The following script is used for generating or terminating traffic on the Traffic Generator Server. If operating as a generator, then the packet size must be specified on the command line. To operate as a sink (terminating traffic) then the keyword SINK needs to be specified.

```
#!/bin/bash

# This script start the Netronome traffic test tool either as a
# traffic sink (with the SINK command line argument) or as a
# traffic source (with the packet size on the command line).

if [ “$1” == “” ]; then
  echo “ERROR: Please specify packet size (or the keyword SINK)”
  exit -1
fi
if [ “$1” == “SINK” ]; then
  mode=”SINK”
else
  mode=”SOURCE”
  pktsize=”$1”
fi

############################################################
# Compose the port list (DPDK EAL white list)

whitelist=””
for idx in $(seq 0 3) ; do
  iface=”sdn_v0.$idx”
  # (Domain)/Bus/Device/Function
  bdf=$(ethtool -i $iface | sed -rn ‘s/^bus-info:.*\s(.*)$/\1/p’)
  # White list (port list) for DPDK application
  whitelist=”$whitelist -w $bdf”
done

############################################################
# Setup Command Line

cmd=”/usr/local/bin/trafgen”

cmd=”$cmd -n 1”
cmd=”$cmd -c 0xffff”
cmd=”$cmd -d /opt/netronome/lib/librte_pmd_nfp_net.so”
# Add port list (in the form of an EAL white list)
cmd=”$cmd $whitelist”
cmd=”$cmd --”
# Set the port mask to all eight ports
cmd=”$cmd -p 0xf”

case “$mode” in
  “SOURCE”)
    # ‘Benchmark’ mode (traffic generator)
    cmd=”$cmd --benchmark”
    # Duration in seconds (1..3599)
    # cmd=”$cmd -Q 3599”
    # Ethernet and IP parameters
    cmd=”$cmd --src-mac 00:11:22:33:44:00”
    cmd=”$cmd --dst-mac 00:44:33:22:11:00”
    cmd=”$cmd --src-ip 1.0.0.0”
    cmd=”$cmd --dst-ip 2.0.0.0”
    # Packet Size
    cmd=”$cmd --packet-size $pktsize”
    # Packet Rate (per thread)
    cmd=”$cmd -r 0”
    # Transmit Burst Size {1..128} (DPDK: tx_burst_size, default: 32)
    cmd=”$cmd -t 16”
    # Flows within ‘Stream’ (flows_per_stream, default: 65536)
    cmd=”$cmd --flows-per-stream 2000”
    # Number of Streams (number_of_streams, default: 1)
    cmd=”$cmd --streams 32”
    # Number of Repeats of Stream
    cmd="$cmd --bursts-per-stream 10"
    ;;
  "SINK")
    ;;
esac

# Capture the full command in a file
echo $cmd > /tmp/cmdline-$mode

# Terminate shell and execute command
exec $cmd
```

Device-under-Test Configurations

Setting up Net-to-VM and VM-to-Net

Script name: setup-net-to-vm.sh

This script creates a bridge (br0) and attaches the physical port sdn_p0. The physical port is assigned the port-id 1. It further attaches the four virtual ports sdn_v0.0 - sdn_v0.3 to the bridge and assigns them the port ids 10..13. These four ports are assumed to be connected to the four VMs and the physical port is assumed to be connected to a traffic generator.

```
#!/bin/bash

# This script configures both the Net-to-VM and VM-to-Net use case.
# It optionally also configures a VXLAN tunnel endpoint.

if [ "$1" == "VXLAN" ]; then
  VXLAN="yes"
fi

# Remove the existing br0 (if it exists)
ovs-vsctl --if-exists del-br br0

# Recreate the bridge
ovs-vsctl add-br br0

# Remove the default NORMAL rule
ovs-ofctl del-flows br0

if [ "$VXLAN" != "" ]; then
  l_ipaddr="10.1.1.1"
  r_ipaddr="10.1.1.2"
  vxifname="vxlt"
  ifconfig sdn_p0 down
  # Set MAC address of physical port
  ifconfig sdn_p0 hw ether 02:11:11:00:00:01
  ifconfig sdn_p0 mtu 2000
  ifconfig sdn_p0 up $l_ipaddr/24
  # Add Static ARP entry to peer
  arp -i sdn_p0 -s $r_ipaddr 02:11:11:00:00:02
  ovs-vsctl \
    -- --may-exist add-port br0 $vxifname \
    -- set Interface $vxifname ofport_request=1 \
    -- set interface $vxifname type=vxlan \
       options:local_ip=$l_ipaddr \
       options:remote_ip=$r_ipaddr \
       options:key=1
else
```

```
  ifconfig sdn_p0 0
  # Add Physical port (=1)
  ovs-vsctl add-port br0 sdn_p0 \
    -- set Interface sdn_p0 ofport_request=1
fi

# Add VM ports and set the port numbers to 10..13
ovs-vsctl add-port br0 sdn_v0.0 -- set Interface sdn_v0.0 ofport_re-
quest=10
ovs-vsctl add-port br0 sdn_v0.1 -- set Interface sdn_v0.1 ofport_re-
quest=11
ovs-vsctl add-port br0 sdn_v0.2 -- set Interface sdn_v0.2 ofport_re-
quest=12
ovs-vsctl add-port br0 sdn_v0.3 -- set Interface sdn_v0.3 ofport_re-
quest=13
```

Setting up VM-to-VM

Script name: setup-vm-to-vm.sh

This script will attach eight VMs (via sdn_v0.0 to sdn_v0.7) to the common bridge br0.

```
#!/bin/bash

# Bridge name
brname="br0"

# Remove the existing bridge (if it exists)
ovs-vsctl --if-exists del-br $brname

# Add the bridge back (allow for OpenFlow 1.3 features)
ovs-vsctl add-br $brname \
  -- set Bridge $brname protocols=OpenFlow13

# Remove the default NORMAL rule
ovs-ofctl -O OpenFlow13 del-flows $brname

# Attach sdn_v0.0 - sdn_v0.3 to the bridge
for idx in $(seq 0 3); do
  iface="sdn_v0.$idx"
  ofpidx=$(( 10 + $idx ))
  ovs-vsctl add-port $brname $iface \
    -- set Interface $iface ofport_request=$ofpidx
done
```

Collecting Results

The Netronome trafgen tool has been used for collecting results in the form of packet rates. The method is slightly different for each use case.

Network to VM

The traffic is received on four different VMs, so one needs to take a snapshot of the received rates on each VM. The reading from trafgen on one VM looks something like this:

```
+=======================================================+
| Timer period:                                       1 |
+------ Statistics for port   0 -----------------------+
| Packets sent:                                       0 |
| Packets received:                            49685140 |
| Packet receive rate:                          7079952 |
| Packet send rate:                                   0 |
| Bytes received:                          3179848960 |
| Byte receive rate:                        453116928 |
| Packets dropped on send:                          0 |
| Packets dropped on receive:                49685140 |
+=====================================================+
```

This was captured during a 64-byte test. Assuming that all four VMs are reporting approximately the same rates, one can deduct that the total packet rate is 4x7079952 pps, which equals 28.319 Mpps. Similarly, the total bit rate is 8x4x453116928 bps which equals 14.500 Gbps.

VM to Network

For this use case, the traffic is received by the external device, so there is no need to aggregate measurements. A read-out from trafgen may look like:

```
+================= Aggregate statistics ===============+
| Total packets sent:                               0 |
| Total packets received:                   983333538 |
| Total packet send rate:                           0 |
| Total packet receive rate:                 27689512 |
| Total bytes sent:                                 0 |
| Total bytes received:                   62933358058 |
| Total byte send rate:                             0 |
| Total byte receive rate:                 1772128768 |
| Total packets dropped on send:                    0 |
| Total packets dropped on receive:       19261435670 |
+=====================================================+
```

This was also captured during a 64-byte test. One can directly read-out the packet rate to be 27.690 Mpps. For the bit rate, the displayed value needs to be adjusted from byte rate to bit rate: 8x1772128768 bps, which equals 14.177 Gbps.

VM to VM

In this use case, traffic is generated on all four VMs and similarly received by all four VMs. The flow rules implement a full mesh (traffic generated by any specific VM is distributed to the three other VMs).

The measurement of interest is how much traffic is received by the VMs in total. In other words, just like the Network-to-VM measurement, one needs to aggregate the four measurements from the four VMs.

NETRONOME

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