Isolated vRouter Setup and Testing with Trex

Test Methodology

2.4.2019 (1.1)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Comment |
| 1.0 | 8.20.2018 | Joseph Gasparakis [joseph.gasparakis@intel.com](mailto:joseph.gasparakis@intel.com)  Subarna Kar [subarna.kar@intel.com](mailto:subarna.kar@intel.com)  Savannah Loberger [savannah.loberger@intel.com](mailto:savannah.loberger@intel.com)  Wang, Yipeng [yipeng1.wang@intel.com](mailto:yipeng1.wang@intel.com) | Initial Version |
| 1.1 | 2.4.2019 | Savannah Loberger [savannah.loberger@intel.com](mailto:savannah.loberger@intel.com)  Wang, Yipeng [yipeng1.wang@intel.com](mailto:yipeng1.wang@intel.com) | Added updates with commit IDs and information for DPDK setup required before running vRouter. Update to using vRouter 5.0 version. |
| 1.2 | 7.2.2019 | Matthew Davis [Matthew.Davis.2@team.telstra.com](mailto:Matthew.Davis.2@team.telstra.com) | Typo fixes  Additional commands (wget, tar etc)  sandeshy.hh workaround added |

Table of Contents

[Overview 3](#_Toc187241)

[**Goals** 3](#_Toc187242)

[**Setup Information** 4](#_Toc187243)

[Installation 4](#_Toc187244)

[**Machine 1 (vRouter and Qemu)** 4](#_Toc187245)

[Install vRouter 4](#_Toc187246)

[Install Qemu 6](#_Toc187247)

[**Machine 2 (Trex and DPDK)** 6](#_Toc187248)

[Install DPDK 6](#_Toc187249)

[Install Trex 7](#_Toc187250)

[Running Isolated vRouter 8](#_Toc187251)

[**Overview** 8](#_Toc187252)

[**Start vRouter** 8](#_Toc187253)

[Setup DPDK 8](#_Toc187254)

[Run vRouter 9](#_Toc187255)

[**Setup vRouter** 9](#_Toc187256)

[Add Interfaces to vRouter 9](#_Toc187257)

[Using vTest 9](#_Toc187258)

[Example xml File 10](#_Toc187259)

[Run vTest 12](#_Toc187260)

[**Start and Setup VM** 13](#_Toc187261)

[Start VM 13](#_Toc187262)

[Configure VM 13](#_Toc187263)

[Install L2FWD in VM 13](#_Toc187264)

[**Setup Trex to send MPLS/UDP Packets** 14](#_Toc187265)

[DPDK Nic Setup 14](#_Toc187266)

[Trex configuration 14](#_Toc187267)

[Sample Packet File for Trex 16](#_Toc187268)

[Running Traffic 17](#_Toc187269)

[**Forwarding Packet(s) in VM** 17](#_Toc187270)

[Running L2FWD 17](#_Toc187271)

[**Running Trex** 17](#_Toc187272)

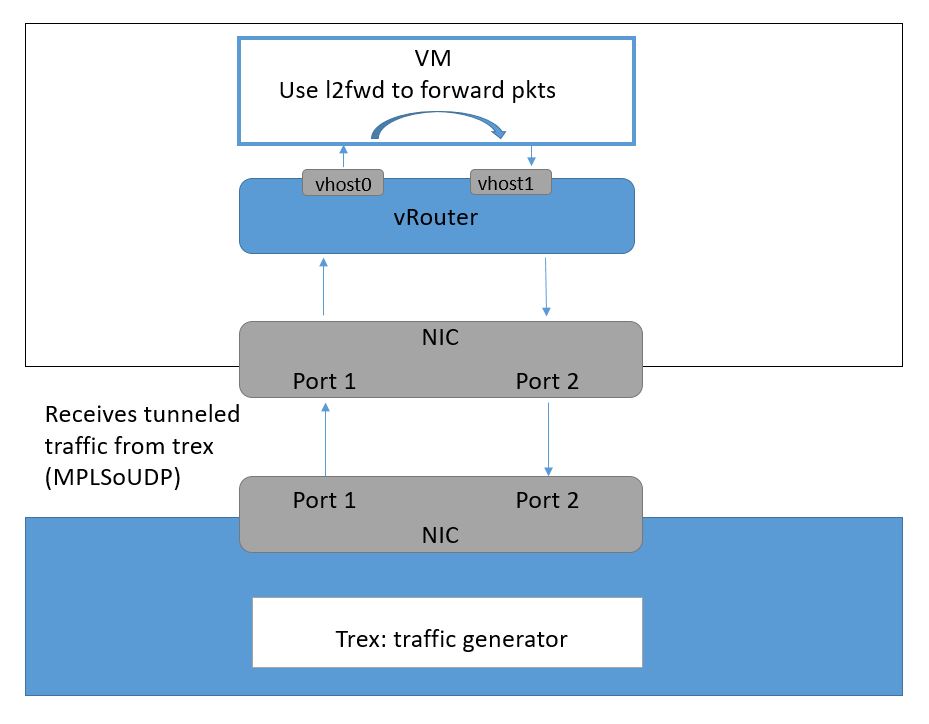
[Trex Stateless Mode 17](#_Toc187273)

[References 18](#_Toc187274)

**Overview**

## **Goals**

* This setup was originally created in order to find the limitations/bottlenecks of vRouter version 4.0 to then implement optimizations of the virtual router (vRouter) that is used in Tungsten Fabric (tungstenfabric.io).
* Isolated vRouter is the vRouter running outside of the usual Contrail environment. There is no agent or other software controlling the flows or paths. Programming of the datapath of the vRouter is done by the user through vrouter utilities such as vtest tool – the vrouter unit test framework, vif – vrouter interface tool, etc.
* The eventual goal is to run isolated vRouter from the master branch. That would provide a good opportunity for posting patches to the community and continue to improve Tungsten Fabric.



Machine 2

Machine 1

## **Setup Information**

* Ubuntu 16.04 LTS
* X710-DA4 NICs
  + 2 ports connected on each machine

**Installation**

## **Machine 1 (vRouter and Qemu)**

### Install vRouter

1. As root, Download the code:

# cd /root

# git clone https://github.com/Juniper/contrail-vrouter.git vrouter

# git clone https://github.com/Juniper/contrail-build.git tools/build

# git clone https://github.com/Juniper/contrail-sandesh.git tools/sandesh

# git clone https://github.com/Juniper/contrail-dpdk.git third\_party/dpdk

# git clone https://github.com/Juniper/contrail-common/ src/contrail-common

Now, the directory tree should look like below:

root

|

---vrouter (this is the contrail-vrouter)

|

---tools

| |

| ----build (this is the contrail-build)

| |

| ----sandesh (this is should be the contrail-sandesh)

|

---third\_party

| |

| ----dpdk (this should be the contrail-dpdk)

|

---src

| |

| ----contrail-common (this is another library needed for sandesh to work)

|

---SConstruct (this should be the file you copy from contrail-build)

1. Enter all the folders and checkout the following commit-ids. This is ensure that all the project is on the same page and to work together successfully.

# cd <folder>

# git checkout <commit-id>

|  |  |
| --- | --- |
| Folder | Commit-id |
| vrouter | bdf961e447ecada259548905e1582a8696878443 |
| third\_party/dpdk | c5841c5284bca2f6f1afe077131489674324db1c |
| tools/sandesh | b5d5c1ee1117f59d8f00de620c8b9db236f6cb1e |
| tools/build | a99cefda8c8b22174347e276ad97b85155086874 |
| src/contrail-common | c93ef4b32cb64faac8267e2b7cd58c5c1ecb4f87 |

1. Copy the SConstruct file to the root folder:

# cd /root

# cp tools/build/SConstruct ./

1. To compile you may need the following packages (you may need to install each package individually as we have found that the package manager may skip some):

# apt install -y libboost-all-dev libnl-genl-3-dev libxml2-dev liburcu-dev byacc flex libpcap-dev scons python python-pip pkg-config zlib1g-dev libglib2.0-dev libfdt-dev libpixman-1-dev cloud-image-utils bison binfmt-support

For the setup, the following packages were also needed

# wget http://ubuntu-cloud.archive.canonical.com/ubuntu/pool/main/libu/liburcu/liburcu2\_0.8.5-1ubuntu1~cloud0\_amd64.deb

# wget http://ubuntu-cloud.archive.canonical.com/ubuntu/pool/main/libu/liburcu/liburcu-dev\_0.8.5-1ubuntu1~cloud0\_amd64.deb

The package name may vary. The build system should warn you if you lack some packages.

To install these packages run

# dpkg –i liburcu-dev\_0.8.5-1ubuntu1~cloud0\_amd64.deb

# dpkg –i liburcu2\_0.8.5-1ubuntu1~cloud0\_amd64.deb

You may also have to also install the following .deb files (wget then dpkg -i):

* https://master.dl.sourceforge.net/project/libipfix/RELEASES/libipfix-dev\_0.8.1-1ubuntu1\_amd64.deb
* http://downloads.datastax.com/cpp-driver/ubuntu/16.04/cassandra/v2.11.0/cassandra-cpp-driver\_2.11.0-1\_amd64.deb
* http://downloads.datastax.com/cpp-driver/ubuntu/16.04/cassandra/v2.11.0/cassandra-cpp-driver-dev\_2.11.0-1\_amd64.deb
* http://downloads.datastax.com/cpp-driver/ubuntu/16.04/cassandra/v2.11.0/cassandra-cpp-driver-dbg\_2.11.0-1\_amd64.deb

1. Compile vrouter, if no option is selected the default build is debug, however the production has better performance: (-jn may not work, if error, remove the -j option)

# cd /root

# scons vrouter -j2 --opt=production/debug/...

If you get an error about sandeshy.hh, try running:

# grep -r sandeshy\.hh -l | xargs sed -i 's/sandeshy\.hh/sandeshy\.h/g'

1. After a successful build the DPDK-Vrouter binary is located at:

/root/build/debug/vrouter/dpdk/contrail-vrouter-dpdk

Or

/root/build/production/vrouter/dpdk/contrail-vrouter-dpdk

### Install Qemu

1. In root directory, download qemu source code and install (we used qemu-2.11.1 in our setup)
2. Download the zip from this site: <https://github.com/qemu/qemu/tree/stable-2.5>

# wget https://github.com/qemu/qemu/archive/v2.11.1.tar.gz

# tar -xzf v2.11.1.tar.gz

# cd qemu-2.11.1

# ./configure

# make

1. The build for qemu can take a couple hours so better use the –j option of make (see your distro’s man pages), depending the number of cores in your system

## **Machine 2 (Trex and DPDK)**

### Install DPDK

1. We used version 18.05.1, you can find all dpdk source code from [www.dpdk.org](http://www.dpdk.org)

$ wget https://fast.dpdk.org/rel/dpdk-18.05.1.tar.xz

$ tar xf dpdk-18.05.1.tar.xz

$ cd dpdk-18.05.1

1. Build the DPDK environment (x86\_64-native-linuxapp-gcc).

$ sudo apt install libnuma-dev

$ make config T=x86\_64-native-linuxapp-gcc

$ make # use –j

1. Become superuser (if not already)

# sudo su

1. Insert the igb\_uio module

# modprobe uio

# insmod ./build/kmod/igb\_uio.ko

Setup hugepages and enter the amount of hugepages you want (64 is a good number to start with)

# mkdir -p /mnt/huge

# mount -t hugetlbfs nodev /mnt/huge

# echo 64 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr\_hugepages

# # repeat for other nodes if you have them

# echo 64 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr\_hugepages

### Install Trex

* The setup of DPDK, explained above, must be done before running [Trex](https://trex-tgn.cisco.com/trex/doc/trex_manual.html" \l "_download_and_installation).
* Install a dependency

# sudo apt-get install python3-distutils

* We used Trex version 2.53

# mkdir trex

# cd trex

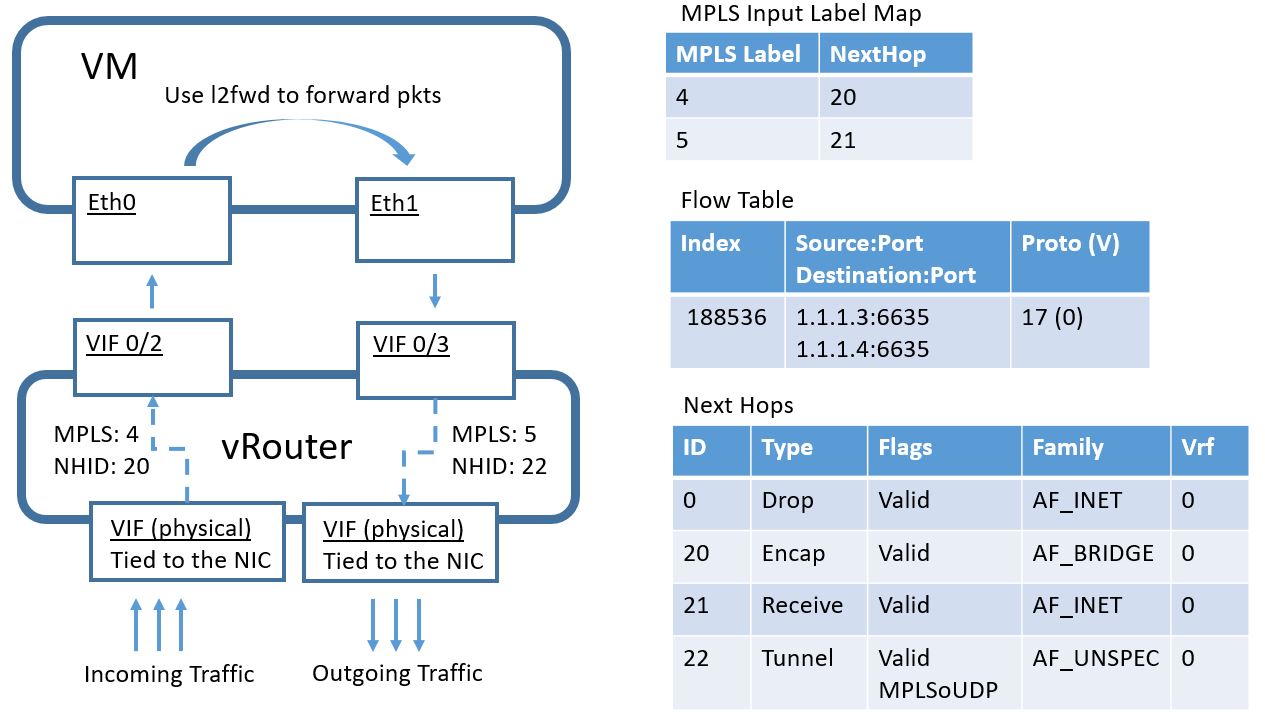
# wget --no-cache <http://trex-tgn.cisco.com/trex/release/v2.09.tar.gz>

# tar –xzf v2.53.tar.gz

**Running Isolated vRouter**

## **Overview**

* Diagram of our configuration of vRouter:



VIF 0/0 is the bottom left one. VIF 0/1 is the bottom right.

Here are the MAC addresses for this setup.

|  |  |  |  |
| --- | --- | --- | --- |
| Interface | Description | MAC | Different for you |
| Eth0 | Inside the VM | 02:e9:ee:49:c3:bc | No. Use this MAC value |
| Eth1 | Inside the VM | 02:e9:ee:49:c3:bd | No. Use this MAC value |
| VIF 0/0 | Physical NIC | 3c:fd:fe:9c:5b:19 | Yes, lookup with `ip a` |
| VIF 0/1 | Physical NIC | 3c:fd:fe:9c:5b:18 | Yes, lookup with `ip a` |
| VIF 0/2 | vRouter interface, looking up into VM | 00:00:5e:00:01:00 | No. Use this MAC value |
| VIF 0/3 | vRouter interface, looking up into VM | 00:00:5e:00:01:?? | No. Use this MAC value |

## **Start vRouter**

### Setup DPDK

Return to the first machine. Before running vRouter, setup hugepages and insert the igb\_uio modules. The following commands have to be ran before running vRouter after a reboot of the machine.

1. Setup hugepages.

# mkdir /dev/hugepages1G /mnt/huge -p

# mount -t hugetlbfs -o pagesize=1G none /mnt/huge

1. For Ubuntu, the vRouter references the hugepage information from the “/proc/mount” file. Verify that the entry was made in the file.

# grep /mnt/huge /proc/mounts

1. Insert igb\_uio module.

# modprobe uio

# cd /root

# insmod build/production/vrouter/dpdk/x86\_64-native-linuxapp-gcc/kmod/igb\_uio.ko

1. Choose the 2 physical interfaces you want to use. Write down their MAC address, PCI address and linux interface name.

# ./third\_party/dpdk/usertools/dpdk-devbind.py --status # for pci and linux interface name

# ip a # to get mac

1. Bind devices to igb\_uio. Replace the <interfaces> with your devices names.

# ifconfig <eth device> down

# ./third\_party/dpdk/usertools/dpdk-devbind.py --bind=igb\_uio <pcie address of nic>

# ./third\_party/dpdk/usertools/dpdk-devbind.py --bind=igb\_uio <pcie address of nic>

### Run vRouter

1. If you are restarting vrouter
2. Command to start vRouter:

# taskset 0x3f ./build/production/vrouter/dpdk/contrail-vrouter-dpdk --no-daemon --socket-mem 1024,1024

(or perhaps debug instead of production)

## **Setup vRouter**

**Setup config file**

sudo mkdir –p /etc/contrail

sudo vim /etc/contrail/contrail-vrouter-agent.conf

Add the following to the file

[DEFAULT]  
platform=dpdk

### Add Interfaces to vRouter

You can add physical and virtual devices to the vRouter. You can run vRouter with only one interface of each. However, for this setup you need:

* 2 physical interfaces
* 2 virtual interfaces

1. Add 2 physical interfaces to vRouter. The MAC addresses should match the ones for these interfaces which you wrote down earlier:

# ./build/production/vrouter/utils/vif --add 0 --mac 3c:fd:fe:9c:5b:19 --vrf 0 --type physical --pmd

# ./build/production/vrouter/utils/vif --add 1 --mac 3c:fd:fe:9c:5b:18 --vrf 0 --type physical --pmd

1. Add 2 virtual interfaces to vRouter: 1

# ./build/production/vrouter/utils/vif --add 2 --mac 00:00:5e:00:01:00 --vrf 0 --type virtual --transport pmd --pmd --policy

# ./build/production/vrouter/utils/vif --add 3 --mac 00:00:5e:00:01:01 --vrf 0 --type virtual --transport pmd --pmd --policy

(“--policy” sets the policy flag to on, so that packets from the VM will go through the flow lookup.)

Check to see what has been set up.

# ./build/production/vrouter/utils/vif --list

It should look like this:

Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror

Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2

D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged

Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is Monitored

Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning Enabled

Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, Ig=Igmp Trap Enabled

vif0/0 PMD: 0 (Speed 10000, Duplex 1)

Type:Physical HWaddr:f8:f2:1e:05:ff:a4 IPaddr:0.0.0.0

Vrf:0 Mcast Vrf:65535 Flags:TcL3L2Dpdk QOS:0 Ref:15

RX device packets:103 bytes:33470 errors:0

RX port packets:101 errors:0

RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

RX packets:101 bytes:33330 errors:101

TX packets:0 bytes:0 errors:0

Drops:101

vif0/1 PMD: 1 (Speed 10000, Duplex 1)

Type:Physical HWaddr:f8:f2:1e:05:ff:a6 IPaddr:0.0.0.0

Vrf:0 Mcast Vrf:65535 Flags:TcL3L2Dpdk QOS:0 Ref:15

RX device packets:102 bytes:33140 errors:0

RX port packets:90 errors:0

RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

RX packets:90 bytes:29700 errors:90

TX packets:0 bytes:0 errors:0

Drops:90

vif0/2 PMD: 2

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0

Vrf:0 Mcast Vrf:65535 Flags:PL3L2Dpdk QOS:0 Ref:15

RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

RX packets:0 bytes:0 errors:0

TX packets:0 bytes:0 errors:0

vif0/3 PMD: 3

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0

Vrf:0 Mcast Vrf:65535 Flags:PL3L2Dpdk QOS:0 Ref:15

RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

RX packets:0 bytes:0 errors:0

TX packets:0 bytes:0 errors:0

Drops:0

### Using vTest

* vTest is the vrouter test tool that allows you to control the vRouter through xml file. Using vTest we are able to run vRouter without the agent present.

### Example xml File

For our setup we created 2 nexthops, 1 route lookup, 1 flow request, and added an MPLS label to the VM.

Some of the MAC addresses in the following file must be changed. Read the XML comments.

Contents of mpls\_add.xml: (When trying to comprehend this file, please note that the endianness is not always what you expect.)

|  |
| --- |
| <?xml version="1.0"?>  <test>  <test\_name>Adds nexthop, </test\_name>  <message>  <vr\_nexthop\_req>  <!--This nexthop is to receive the tunneled packet and is obtained by route lookup of outer dst IP -->  <h\_op>Add</h\_op>  <nhr\_type>1</nhr\_type> <!-- RCV type -->  <nhr\_id>21</nhr\_id> <!-- you can choose any number, other than 0 which is the default nexthop to drop the pkt -->  <nhr\_family>2</nhr\_family> <!-- type AF\_INET -->  <nhr\_encap\_oif\_id>1</nhr\_encap\_oif\_id>  <!-- In case of a non tunneled packet, it will be sent out of the other NIC -->  <nhr\_vrf>0</nhr\_vrf>  <nhr\_flags>1</nhr\_flags>  </vr\_nexthop\_req>  <return>0</return>  </message>  <message>  <vr\_nexthop\_req>  <!--This nexthop is for inner packet, obtained from mpls tag and points to the corresponding VM the pkt should be sent to -->  <h\_op>Add</h\_op>  <nhr\_type>2</nhr\_type> <!-- encap type -->  <nhr\_id>20</nhr\_id>  <nhr\_family>7</nhr\_family> <!-- type AF\_BRIDGE -->  <nhr\_encap\_oif\_id>2</nhr\_encap\_oif\_id>  <!--pointing to the VM which is interface 2 in our setup -->  <nhr\_encap>02:e9:ee:49:c3:bc:00:00:5e:00:01:00:08:00</nhr\_encap> <!--MAC address of interface in VM and MAC address of virtual interface in vrouter, then 08:00 -->  <nhr\_vrf>0</nhr\_vrf>  <nhr\_flags>7</nhr\_flags>  </vr\_nexthop\_req>  <return>0</return>  </message>  <message>  <vr\_nexthop\_req>  <!--This nh is for pkts coming from 3rd vif from VM to go out of the 1st physical vif to go to trex -->  <h\_op>Add</h\_op>  <nhr\_type>3</nhr\_type> <!-- Tunnel type -->  <nhr\_id>22</nhr\_id>  <nhr\_encap\_oif\_id>1</nhr\_encap\_oif\_id> <!--go out of NIC 1 -->  <nhr\_encap>68:05:ca:04:4a:91:3c:fd:fe:9c:5b:18:08:00</nhr\_encap>  <nhr\_tun\_sip>33686019</nhr\_tun\_sip> <!-- 2.2.2.3 -->  <nhr\_tun\_dip>33686021</nhr\_tun\_dip> <!-- 2.2.2.5 -->  <nhr\_tun\_sport>0</nhr\_tun\_sport>  <nhr\_tun\_dport>0</nhr\_tun\_dport>  <nhr\_vrf>0</nhr\_vrf>  <nhr\_flags>65</nhr\_flags> <!--NH\_FLAG\_TUNNEL\_UDP\_MPLS, NH\_FLAG\_VALID -->  </vr\_nexthop\_req>  <return>0</return>  </message>  <message>  <vr\_route\_req>  <!--This route lookup is to connect pkts with outer dst IP 2.2.2.2 to RCV NH of id 21. In full setup, the IP shd be same as vhost0's IP -->  <h\_op>Add</h\_op>  <rtr\_family>2</rtr\_family>  <rtr\_nh\_id>21</rtr\_nh\_id> <!--ID of the RCV nexthop defined earlier -->  <rtr\_prefix>2.2.2.2</rtr\_prefix> <!--this can be any IP address you mention in outer dst IP in TREX -->  <rtr\_prefix\_len>8</rtr\_prefix\_len>  <rtr\_vrf\_id>0</rtr\_vrf\_id>  </vr\_route\_req>  <return>0</return>  </message>  <message>  <vr\_route\_req>  <!--This is to create bridge entry for pkts coming out of the VM to reach trex port through NIC -->  <!--Naturally it shd search for TREx dst MAC/RX VM's MAC in bridge, in this case, l2fwd, no mac updating, so pkt has same dst MAC as being sent by TREX, as set in udp\_2pkt\_simple.py -->  <h\_op>Add</h\_op>  <rtr\_family>7</rtr\_family>  <rtr\_nh\_id>22</rtr\_nh\_id>  <rtr\_label\_flags>3</rtr\_label\_flags> <!-- BR\_BE\_VALID\_FLAG -->  <rtr\_mac>02:e9:ee:49:c3:bc</rtr\_mac>  <rtr\_label>5</rtr\_label> <!-- does this become MPLS label? -->  <rtr\_vrf\_id>0</rtr\_vrf\_id>  </vr\_route\_req>  <return>0</return>  </message>  <message>  <vr\_flow\_req>  <!--This flow is for the inner packet lookup after mpls header decapsulation, it looks at the 5 tuples -->  <fr\_op>flow\_set</fr\_op>  <fr\_flow\_sip\_l>50397441</fr\_flow\_sip\_l> <!--decimal version of src IP 3.1.1.1 -->  <fr\_flow\_sip\_u>0</fr\_flow\_sip\_u>  <fr\_flow\_dip\_l>67174657</fr\_flow\_dip\_l> <!-- decimal version 4.1.1.1 -->  <fr\_flow\_dip\_u>0</fr\_flow\_dip\_u>  <fr\_family>2</fr\_family>  <fr\_index>-1</fr\_index>  <fr\_flags>1</fr\_flags>  <fr\_flow\_proto>17</fr\_flow\_proto> <!-- UDP -->  <fr\_flow\_sport>60185</fr\_flow\_sport><!-- this port is needed for it to know this is MPLS pkt, this value is hardcoded in vrouter code -->  <fr\_flow\_nh\_id>0</fr\_flow\_nh\_id>  <fr\_action>2</fr\_action> <!-- FORWARD -->  <fr\_flow\_dport>60185</fr\_flow\_dport>  </vr\_flow\_req>  <return>0</return>  </message>  <message>  <vr\_mpls\_req>  <!--Adding mpls tag, which should be done by agent when VM comes up -->  <h\_op>Add</h\_op>  <mr\_label>4</mr\_label> <!--Randomly chosen, shd be mentioned in TREX pkt -->  <mr\_nhid>20</mr\_nhid> <!--points to the ENCAP nexthop defined earlier -->  </vr\_mpls\_req>  <return>0</return>  </message>  <message>  <vr\_mpls\_req>  <h\_op>Add</h\_op>  <mr\_label>5</mr\_label>  <mr\_nhid>21</mr\_nhid>  </vr\_mpls\_req>  <return>0</return>  </message>  </test> |

### Run vTest

1. Add mpls tags, next hops, and routes to vRouter, create the new file and store mpls\_add.xml at /root :

# cd /root

# ./build/production/vrouter/utils/vtest/vtest mpls\_add.xml

You can inspect the hops and flows with:

# ./build/production/vrouter/utils/nh --list

#./build/production/vrouter/utils/flow -l

## **Start and Setup VM**

### Prep VM

1. Download and prep images (Ubuntu 16)

IMG\_DATA="xenial-server-cloudimg-amd64-disk1.img"

IMG\_UEFI="xenial-server-cloudimg-amd64-uefi1.img"

URL\_PREFIX=<https://cloud-images.ubuntu.com/xenial/current/>

USER\_DATA="user-data.img"

wget ${URL\_PREFIX}${IMG\_DATA} --show-progress

wget ${URL\_PREFIX}${IMG\_UEFI} --show-progress

/root/qemu-2.11.1/qemu-img resize "${IMG\_DATA}" +128G

1. Prep user data for cloud init. Edit file user-data.txt

#cloud-config

password: myPass

chpasswd: { expire: False }

ssh\_pwauth: True

1. Create user data image

cloud-localds "$USER\_DATA" user-data.txt

### Start VM

1. Qemu command to start VM (using Qemu built from source),

“path” is to the socket that was created after adding virtual interfaces to the

isolated vrouter:

# taskset 0xf00 ./qemu-2.11.1/x86\_64-softmmu/qemu-system-x86\_64 \

-m 3G \

-drive "file=${IMG\_UEFI},format=qcow2" \

-drive "file=${IMG\_DATA},format=qcow2" \

-drive "file=${USER\_DATA},format=raw" \

-cpu host \

-object memory-backend-file,id=mem,size=3072M,mem-path=/mnt/huge,share=on \

-numa node,memdev=mem \

-mem-prealloc \

-mem-path /mnt/huge,prealloc=on,share=on \

-smp cores=4,threads=1,sockets=1 \

--enable-kvm \

-chardev socket,id=chr0,path=/var/run/vrouter/uvh\_vif\_2 \

-netdev type=vhost-user,id=net0,chardev=chr0 \

-device virtio-net-pci,netdev=net0,mac=02:e9:ee:49:c3:bc \

-chardev socket,id=chr1,path=/var/run/vrouter/uvh\_vif\_3 \

-netdev type=vhost-user,id=net1,chardev=chr1 \

-device virtio-net-pci,netdev=net1,mac=02:e9:ee:49:c3:bd \

-nographic \

-device virtio-net-pci,netdev=net2 \

-netdev user,id=net2 \

;

If you’re doing this over a ssh connection (without a graphic window) you may need to remove the vnc argument and add

-serial mon:stdio

The login is user:ubuntu password: myPass

To quit qemu, use Ctrl-A X

You will be able to run `apt update` from inside the VM, but not ping, because the internet interface blocks icmp (and inbound ssh).

### Configure VM

1. Open VM and stop network manager
2. Grow the root partition because the original cloud image has no room

# apt-get install cloud-initramfs-growroot

# reboot

### Install L2FWD in VM

1. Inside the same VM, install and compile DPDK to use L2FWD to forward the incoming packets. Below are the list of commands that we ran to setup DPDK for L2FWD

# wget https://fast.dpdk.org/rel/dpdk-18.08.tar.xz

# tar xf dpdk-18.08.tar.xz

# cd dpdk-18.08

# export RTE\_SDK=/home/$USER/dpdk-18.08

# export RTE\_TARGET=x86\_64-native-linuxapp-gcc

First build the DPDK environment (x86\_64-native-linuxapp-gcc).

# apt install build-essential libnuma-dev python pkg-config

# make config T=$RTE\_TARGET

# make

Second insert the igb\_uio module

# sudo modprobe uio

# sudo insmod build/kmod/igb\_uio.ko

Setup hugepages and enter the amount of hugepages you want (64 is a good number to start with)

# mkdir -p /mnt/huge

# mount -t hugetlbfs nodev /mnt/huge

# echo 64 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr\_hugepages

1. Connect the two eth interfaces to DPDK

# ./usertools/dpdk-devbind.py --bind=igb\_uio eth0

# ./usertools/dpdk-devbind.py --bind=igb\_uio eth1

1. Build L2FWD (RTE\_SDK is the path to the DPDK folder)

# make -C examples RTE\_SDK=$(pwd) RTE\_TARGET=build O=$(pwd)/build/examples

## **Setup Trex to send MPLS/UDP Packets**

### DPDK Nic Setup

1. Write down the interface name, MAC and PCI addresses of your interfaces

# cd trex/v2.53

# ./dpdk\_nic\_bind.py --status

# ip a

Or, you can use ./dpdk\_nic\_bind.py –t to display a table of information about your NICs

Using the displayed information, bind a minimum of 2 ports to igb\_uio driver

# ./dpdk\_nic\_bind.py -b igb\_uio 08:00.0 08:00.1

### Trex configuration

1. Sample Trex Configuration File, stored at /etc/trex\_cfg.yaml, add in your devices MAC addresses and port information

If that file does not exist, run through `sudo ./dpdk\_setup\_ports.py -i`. Say yes to MAC based, and use MAC of DUT. When prompted for the MAC, use what you originally saw with `ip a` on the other machine.

Example contents of trex\_cfg.yaml:

|  |
| --- |
| ### Config file generated by dpdk\_setup\_ports.py ###  - version: 2  interfaces: ['63:00.0', '63:00.1']  port\_info:  - dest\_mac: f8:f2:1e:05:ff:a4  src\_mac: 3c:fd:fe:3b:45:00  - dest\_mac: f8:f2:1e:05:ff:a6  src\_mac: 3c:fd:fe:3b:45:02  platform:  master\_thread\_id: 0  latency\_thread\_id: 1  dual\_if:  - socket: 0  threads: [2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62] |

The src\_mac values should match the MAC addresses of the interfaces on the generator NICs.

The dest\_mac values should match the MAC addresses of the physical NICs on the unit under test.

### Sample Packet File for Trex

* Based on [RFC7510](https://tools.ietf.org/html/rfc7510) for the vRouter to accept the MPLSoUDP packets the UDP destination and source ports must be set to 6635
* Also, there must be time-to-live (TTL) set to a high enough value (ex. we used 63) otherwise the packet is dropped

Save the following as mpls\_udp\_1pkt\_simple.py:

|  |
| --- |
| from trex\_stl\_lib.api import \*  from scapy.contrib.mpls import \*  class STLS1(object):  def create\_stream (self):  return STLStream(  packet =  STLPktBuilder(  pkt = Ether()/IP(src="2.2.2.1",dst="2.2.2.2")/UDP(dport=6635,sport=6635)/MPLS(label=0x04,ttl=63)/Ether(dst="02:e9:ee:49:c3:bc")/IP(src="1.1.1.3",dst="1.1.1.4")/UDP(dport=6635,sport=6635)/(10\*'x')  ),  mode = STLTXCont())  def get\_streams (self, direction = 0, \*\*kwargs):  # create 1 stream  return [ self.create\_stream() ]  # dynamic load - used for trex console or simulator  def register():  return STLS1() |

**Running Traffic**

## **Forwarding Packet(s) in VM**

### Running L2FWD

1. Forward the incoming traffic inside the VM on eth0 to eth1

# ./examples/l2fwd/build/app/l2fwd -l 0-3 -n 4 -- -p 0x03 --no-mac-updating

## **Running Trex**

### Trex Stateless Mode

1. In one terminal of Machine 2 start trex in interactive mode (at this point there should be available hugepages for trex to run that was setup in a previous step with DPDK)

# cd trex/v2.53

# ./t-rex-64 -i -c 4

1. In another terminal start the console for trex and start sending traffic, below configures the receiving port (port 1) to promiscuous mode to receive the incoming packets and sends MPLSoUDP packets from port 0 for 60 seconds

# cd trex/v2.53

# ./trex-console

trex> stats

trex> portattr --port 1 --prom on

trex> start -p 0 -d 60 -f stl/mpls\_udp\_1pkt\_simple.py -m 100%

trex> stats

trex> quit

**References**

* For more information about setting up and running DPDK, refer here:

<https://doc.dpdk.org/guides/index.html>

* For more information about trex, refer here:

<https://trex-tgn.cisco.com/trex/doc/trex_manual.html#_download_and_installation>

* For more information about running the trex console refer here:

<https://trex-tgn.cisco.com/trex/doc/trex_console.html>

* For more information about the vRouter tool, vif:

<https://www.juniper.net/documentation/en_US/contrail2.0/topics/task/configuration/vrouter-cli-utilities-vnc.html>

* For more information about the vRouter tool, vtest:

<https://github.com/Juniper/contrail-vrouter/tree/master/utils/vtest>