|  |
| --- |
| Juniper Networks |
| Contrail DPDK logs Analysis |
| How to read Read the Contrail DPDK vrouter logs |

|  |
| --- |
| Laurent Durand  31/03/2020 |

Table of Contents

[1 Understand vHost User protocol 2](#_Toc36587034)

[1.1 Contrail vRouter and vHost protocol interaction 2](#_Toc36587035)

[1.2 Vhost-user Protocol 4](#_Toc36587036)

[2 Read the vRouter DPDK logs 7](#_Toc36587037)

[2.1 vRouter DPDK and control connectivity setup 7](#_Toc36587038)

[2.1.1 User vHost connectivity 7](#_Toc36587039)

[2.1.2 Netlink connectivity 8](#_Toc36587040)

[2.1.3 Packet0 (vif 2) connectivity 10](#_Toc36587041)

[2.1.4 Forwarding cores 11](#_Toc36587042)

[2.2 VRouter vNIC interface creation: 12](#_Toc36587043)

[2.3 VRouter hostuser interface creation: 12](#_Toc36587044)

[2.3.1 Vhost user socket creation 12](#_Toc36587045)

[2.3.2 Vhost user vrings creation (vnic queues) 15](#_Toc36587046)

[2.3.3 Vhost user vrings memory mapping 17](#_Toc36587047)

# Understand vHost User protocol

## Contrail vRouter and vHost protocol interaction



The vhost user protocol consists of a control path and a data path.

* All control information is exchanged via a Unix socket. This includes information for exchanging memory mappings for direct memory access, as well as kicking / interrupting the other side if data is put into the virtio queue. The Unix socket, in neutron, is named vhuxxxxxxxx-xx.
* The actual dataplane is implemented via DMA (direct memory access). The virtio-net driver within the guest allocates part of the instance memory for the virtio queue. The structure of this queue is standardized in the virtio standard. Qemu shares this memory section’s address with Contrail vRouter over the control channel. DPDK itself then maps the same standardized virtio queue structure onto this memory section and can thus directly read from and write to the virtio queue within the instance’s hugepage memory. This direct memory access is one of the reasons why both Contrail vRouter and qemu need to use hugepage memory. If qemu is otherwise set up correctly, but lacks configuration for huge page memory, then Contrail vRouter will not be able to access qemu’s memory and hence no packets can be exchanged. Users will notice this if they forget to request instance hugepages via nova’s metadata.

When Contrail vRouter transmits towards the instance, these packets will show up within Contrail vRouter DPDK’s statistics as TX port packets on vif 0/n. Within the instance, these packets show up as RX packets.

When the instance transmits packets to Contrail vRouter, then on the instance, these packets show up as TX packets, and on Contrail vRouter vif 0/N port, they show up as RX port packets :

[root@jnprctdpdk02 qemu]# vif --get 7

vif0/7 PMD: tapbd77b8fc-70 NH: 47

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:120.0.0.4

Vrf:4 Mcast Vrf:4 Flags:PL3L2DEr QOS:-1 Ref:14

RX port packets:361 errors:0 syscalls:1

RX queue packets:8 errors:0

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

RX packets:361 bytes:15970 errors:0

TX packets:383 bytes:17080 errors:0

ISID: 0 Bmac: 02:bd:77:b8:fc:70

Drops:9

TX port packets:367 errors:16 syscalls:367

$ ifconfig eth0

eth0 Link encap:Ethernet HWaddr 02:BD:77:B8:FC:70

inet addr:120.0.0.4 Bcast:120.0.0.255 Mask:255.255.255.0

inet6 addr: fe80::bd:77ff:feb8:fc70/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:367 errors:0 dropped:0 overruns:0 frame:0

TX packets:361 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:16408 (16.0 KiB) TX bytes:15970 (15.5 KiB)

Although packets can be directly transmitted via shared memory, either side needs a means to tell the opposite side that a packet was copied into the virtio queue. This happens by kicking the other side over the control plane which is implemented with the vhost user socket vhuxxxxxxxx-xx.

Kicking the other side comes at a cost.

* Firstly, a system call is needed to write to the socket.
* Secondly, an interrupt will have to be processed by the other side. Hence both sender and receiver spend costly extra time within the control channel.

These “kick” calls are also displayed on each contrail vif interface statistics in syscalls counter.

In order to avoid costly kicks via the control plane, both contrail vRouter and qemu can set specific flags to signal to the other side that they do not wish to receive an interrupt. However, they can only do so if they either temporarily or constantly poll the virtio queue.

For virtual machine network performance this means that the optimal means of packet processing is DPDK within the instance itself. While Linux kernel networking (NAPI) uses a mix of interrupt and poll mode processing, it is still exposed to a high number of interrupts.

Contrail DPDK sends packets towards the instance at very high rates. At the same time, the RX and TX buffers of qemu’s virtio queue are limited to a default of 256 and a maximum of 1024 entries. As a consequence, the instance itself needs to process packets very quickly. This is ideally achieved by constantly polling with a DPDK PMD on the instance’s interface.

## Vhost-user Protocol

This protocol is aiming to complement the ioctl interface used to control the vhost implementation in the Linux kernel. It implements the control plane needed to establish virtqueue sharing with a user space process on the same host. It

uses communication over a Unix domain socket to share file descriptors in the ancillary data of the message.

The protocol defines 2 sides of the communication, master and slave. Master is the application that shares its virtqueues, in our case QEMU. Slave is the consumer of the virtqueues.

In the current implementation QEMU is the Master, and the Slave is intended to be a software Ethernet switch running in user space, such as Contrail vRouter. Master and slave can be either a client (i.e. connecting) or server (listening)

vhost user has since 2 sides:

* Master: QEMU
* Slave: Contrail vRouter

vhost user can run in 2 modes:

* vhostuser-client: qemu is the server, the software switch (like Contrail vRouter) is the client
* vhostuser: the software switch (like Contrail vRouter) is the server, qemu is the client

Contrail vRouter is using "vhostuser" mode.

2020-03-28 15:59:18,598 UVHOST: vif (server) 7 socket tapbd77b8fc-70 FD is 58

vhost user is based on the vhost architecture and implements all features in user space. When a qemu instance boots, it will allocate all of the instance memory as shared hugepages. The OS' virtio paravirtualized driver will reserve part of this hugepage memory for holding the virtio ring buffer. This allows OVS DPDK to directly read from and write into the instance's virtio ring. Both Contrail vRouter and qemu can directly exchange packets across this reserved memory section.

"The user space application will receive file descriptors for the pre-allocated shared guest RAM. It will directly access the related vrings in the guest's memory space" (<http://www.virtualopensystems.com/en/solutions/guides/snabbswitch-qemu/>).

Qemu is instructed to allocate memory from the huge page pool and to make it shared memory (share=yes):

-object memory-backend-file,id=ram-node0,prealloc=yes,mem-path=/dev/hugepages2M/libvirt/qemu/2-instance-00000008,share=yes,size=67108864,host-nodes=0,policy=bind \

Simply copying packets into the other party's buffer is not enough, however. Additionally, vhost user uses a Unix domain socket (vhu[a-f0-9-]) for communication between the Contrail vRouter and qemu, both during initialization and to kick the other side when packets were copied into the virtio ring in shared memory. Interaction hence consists of a control path (vhu socket) for setup and notification and a datapath (direct memory access) for moving the actual payload.

For the described Virtio mechanism to work, we need a setup interface to initialize the shared memory regions and exchange the event file descriptors. A Unix domain socket implements an API which allows us to do that. This straightforward socket interface can be used to initialize the userspace Virtio transport (vhost-user), in particular:

* Vrings are determined at initialization and are placed in shared memory between the two processed.
* Virtio events (Vring kicks) we shall use eventfds that map to Vring events. This allows us compatibility with the QEMU/KVM implementation described in the next chapter, since KVM allows us to match events coming from virtio\_pci in the guest with eventfds (ioeventfd and irqfd).

Sharing file descriptors between two processes differs than sharing them between a process and the kernel. One needs to use sendmsg over a Unix domain socket with SCM\_RIGHTS set.

(http://www.virtualopensystems.com/en/solutions/guides/snabbswitch-qemu/)

In vhostuser mode, Contrail vRouter creates the vhu socket and qemu connects to it.

In the Virtual Instance XML definition, qemu is instructed to connect a netdev of type vhost-user to /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70:

-chardev socket,id=charnet0,path=/var/run/vrouter/uvh\_vif\_tapbd77b8fc-70,server \

-netdev vhost-user,chardev=charnet0,id=hostnet0 \

-device virtio-net-pci,rx\_queue\_size=512,tx\_queue\_size=512,netdev=hostnet0,id=net0,mac=02:bd:77:b8:fc:70,bus=pci.0,addr=0x3 \

# lsof /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

qemu-kvm 238022 qemu 13u unix 0xffff9ea0aceb3000 0t0 285651613 /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70

qemu-kvm 238022 qemu 15u unix 0xffff9e9dc9a7f800 0t0 285648356 /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70

In DPDK log we have following info:

2020-03-28 15:59:18,598 UVHOST: Adding vif 7 virtual device tapbd77b8fc-70

2020-03-28 15:59:18,598 UVHOST: vif (server) 7 socket tapbd77b8fc-70 FD is 58

2020-03-28 15:59:18,598 UVHOST: error connecting uvhost socket FD 58 to /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70: No such file or directory (2)

...

2020-03-28 15:59:23,598 UVHOST: connected to /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70 for uvhost socket FD 58

We can see the processes using a given socket:

# ss -x -p | grep 285654089

u\_str ESTAB 0 0 \* 285654089 \* 285648356 users:((**"contrail-vroute",pid=6663,fd=58**))

u\_str ESTAB 0 0 /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70 285648356 \* 285654089 users:(("qemu-kvm",pid=238022,fd=15))

Cf: <https://www.cyberciti.biz/tips/linux-investigate-sockets-network-connections.html>

<https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/10/html-single/ovs-dpdk_end_to_end_troubleshooting_guide/index>

# Read the vRouter DPDK logs

## vRouter DPDK and control connectivity setup

### User vHost connectivity

In DPDK log file, we can see vRouter DPDK dataplane to configure a socket with uvhost and an event FD hander:

2020-01-21 17:56:23,542 UVHOST: Starting uvhost server...

2020-01-21 17:56:23,542 UVHOST: server event FD is 46

2020-01-21 17:56:23,542 UVHOST: server socket FD is 47

Using following commands, we can inspect both File descritors:

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/46

285931258 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/46 -> anon\_inode:[eventfd]

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/47

285931259 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/47 -> socket:[64070]

Using following commands, we can check this is the vrouter which is listening on this socket, and we can retrieve the socket filename:

# ss -xpa | grep "vr\_nl\_uvh\|vr\_uvh\_nl"

u\_seq ESTAB 0 0 /var/run/vrouter/vr\_nl\_uvh 56909 \* 64072 users:(("contrail-vroute",pid=6663,fd=48))

u\_seq LISTEN 0 1 /var/run/vrouter/vr\_uvh\_nl 64070 \* 0 users:(("contrail-vroute",pid=6663,fd=47))

u\_seq ESTAB 0 0 /var/run/vrouter/vr\_uvh\_nl 64072 \* 56909 users:(("contrail-vroute",pid=6663,fd=49))

# ls -li /var/run/vrouter/vr\_uvh\_nl

64071 srwxr-xr-x. 1 root root 0 Jan 21 17:56 /var/run/vrouter/vr\_uvh\_nl

# ls -li /var/run/vrouter/vr\_nl\_uvh

59140 srwxr-xr-x. 1 root root 0 Jan 21 17:56 /var/run/vrouter/vr\_nl\_uvh

This is the vrouter process that is listening on this socket:

# lsof -nn 2> /dev/null | grep vr\_uvh\_nl | awk '{ print $2, " ", $3, " ", $1 }' | uniq

6663 root contrail-

6663 7265 rte\_mp\_ha

6663 7266 rte\_mp\_as

6663 8346 eal-intr-

6663 8347 lcore-sla

6663 8348 lcore-sla

6663 8349 lcore-sla

6663 8350 lcore-sla

6663 8351 lcore-sla

6663 8352 lcore-sla

6663 8353 lcore-sla

6663 11867 lcore-sla

### Netlink connectivity

In DPDK log file, we can see vRouter DPDK dataplane to configure a socket for Netlink:

2020-01-21 17:56:23,541 VROUTER: Starting NetLink...

2020-01-21 17:56:23,542 USOCK: usock\_alloc[7fcb05946700]: new socket FD 45

2020-01-21 17:56:23,542 USOCK: usock\_alloc[7fcb05946700]: setting socket FD 45 send buff size.

Buffer size set to 18320000 (requested 9216000)

Using following commands, we can inspect this File descritor:

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/45

285931257 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/45 -> socket:[56907]

Using following commands, we can check this is the vrouter which is listening on this socket, and we can retrieve the socket filename:

# ss -xap | head -1 ; ss -xap |grep 56907

Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port

u\_str LISTEN 0 1 /var/run/vrouter/dpdk\_netlink 56907 \* 0 users:(("contrail-vroute",pid=6663,fd=45))

# ls -li /var/run/vrouter/dpdk\_netlink

56908 srwxr-xr-x. 1 root root 0 Jan 21 17:56 /var/run/vrouter/dpdk\_netlink

This is the vrouter process that is listening on this socket:

# lsof -nn 2> /dev/null | grep dpdk\_netlink | awk '{ print $2, " ", $3, " ", $1 }' | uniq

| grep dpdk\_netlink | awk '{ print $2, " ", $3, " ", $1 }' | uniq

6663 root contrail-

6663 7265 rte\_mp\_ha

6663 7266 rte\_mp\_as

6663 8346 eal-intr-

6663 8347 lcore-sla

6663 8348 lcore-sla

6663 8349 lcore-sla

6663 8350 lcore-sla

6663 8351 lcore-sla

6663 8352 lcore-sla

6663 8353 lcore-sla

6663 11867 lcore-sla

Then we are looking for the process connected onto the second leg of this socket:

# ss -xap | grep dpdk\_netlink

u\_str LISTEN 0 1 /var/run/vrouter/dpdk\_netlink 56907 \* 0 users:(("contrail-vroute",pid=6663,fd=45))

u\_str ESTAB 0 0 /var/run/vrouter/dpdk\_netlink 69888 \* **70669** users:(("contrail-vroute",pid=6663,fd=51))

# ss -x -p -e | grep **70669**

u\_str ESTAB 0 0 /var/run/vrouter/dpdk\_netlink 69888 \* 70669 users:(("contrail-vroute",pid=6663,fd=51)) <->

u\_str ESTAB 0 0 \* 70669 \* 69888 users:(("contrail-vroute",pid=11679,fd=15)) <->

The vrouter agent is connected on the other end of the socket:

# ps 6663 11679

PID TTY STAT TIME COMMAND

6663 ? Sl 298780:44 /usr/bin/contrail-vrouter-dpdk --no-daemon --socket-mem 1024

11679 ? Sl 983:13 /usr/bin/contrail-vrouter-agent

# ls -li /proc/$(pidof contrail-vrouter-agent)/fd/15

285932129 lrwx------. 1 root root 64 Mar 28 17:20 /proc/11679/fd/15 -> socket:[70669]

### Packet0 (vif 2) connectivity

In DPDK log file, we can see vRouter DPDK dataplane to configure a socket for Packet0:

2020-01-21 17:56:36,007 VROUTER: Adding vif 2 (gen. 5) packet device unix

2020-01-21 17:56:36,007 USOCK: usock\_alloc[7fcb05946700]: new socket FD 54

2020-01-21 17:56:36,007 USOCK: usock\_alloc[7fcb05946700]: setting socket FD 54 send buff size.

Buffer size set to 18320000 (requested 9216000)

Using following commands, we can inspect this File descriptor:

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/54

285931266 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/54 -> socket:[69934]

Using following commands, we can check that both vrouter agent and vrouter DPDK dataplane are listening on this socket, and we can retrieve the socket filename:

# ss -xap | head -1 ; ss -xap |grep 69934

Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port

u\_dgr UNCONN 0 0 /var/run/vrouter/agent\_pkt0 70662 \* 69934 users:(("contrail-vroute",pid=11679,fd=10))

u\_dgr UNCONN 0 0 /var/run/vrouter/dpdk\_pkt0 69934 \* 70662 users:(("contrail-vroute",pid=6663,fd=54))

# ps 6663 11679

PID TTY STAT TIME COMMAND

6663 ? Sl 298780:44 /usr/bin/contrail-vrouter-dpdk --no-daemon --socket-mem 1024

11679 ? Sl 983:13 /usr/bin/contrail-vrouter-agent

# ls -li /var/run/vrouter/dpdk\_pkt0

69935 srwxr-xr-x. 1 root root 0 Jan 21 17:56 /var/run/vrouter/dpdk\_pkt0

# ls -li /var/run/vrouter/agent\_pkt0

70663 srwxr-xr-x. 1 root root 0 Jan 21 17:56 /var/run/vrouter/agent\_pkt0

### Forwarding cores

In Contrail DPDK logs, we can see assigned CPU to vRouter:

2019-12-20 05:58:18,210 VROUTER: --lcores "(0-2)@(0-9),(8-9)@(0-9),**10@1,11@2,12@3**"

2019-12-20 05:58:18,214 EAL: Detected 10 lcore(s)

2019-12-20 05:58:18,214 EAL: Detected 1 NUMA nodes

2019-12-20 05:58:18,984 VROUTER: Using 3 forwarding lcore(s)

2019-12-20 05:58:18,984 VROUTER: Using 0 IO lcore(s)

2019-12-20 05:58:18,984 VROUTER: Using 5 service lcores

Using following command, we can check the forwarding threads are well running on assigned CPU:

# ps -eT -o psr,tid,comm,pid,ppid,cmd,pcpu,stat | head -1 ; \

ps -eT -o psr,tid,comm,pid,ppid,cmd,pcpu,stat | grep $(pidof contrail-vrouter-dpdk)

PSR TID COMMAND PID PPID CMD %CPU STAT

7 6663 contrail-vroute 6663 4172 /usr/bin/contrail-vrouter-d 2.1 Sl

3 7265 rte\_mp\_handle 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

3 7266 rte\_mp\_async 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

3 8346 eal-intr-thread 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

6 8347 lcore-slave-1 6663 4172 /usr/bin/contrail-vrouter-d 3.0 Sl

9 8348 lcore-slave-2 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

3 8349 lcore-slave-8 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

4 8350 lcore-slave-9 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

**1** 8351 lcore-slave-10 6663 4172 /usr/bin/contrail-vrouter-d 99.9 Rl

**2**  8352 lcore-slave-11 6663 4172 /usr/bin/contrail-vrouter-d 99.9 Rl

**3**  8353 lcore-slave-12 6663 4172 /usr/bin/contrail-vrouter-d 99.9 Rl

0 11867 lcore-slave-9 6663 4172 /usr/bin/contrail-vrouter-d 0.0 Sl

0 630676 grep 630676 519716 grep --color=auto 6663 0.0 S+

Using taskset command, we can display CPU pinning rule set on each thread:

# taskset -cap $(pidof contrail-vrouter-dpdk)

pid 6663's current affinity list: 0-9

pid 7265's current affinity list: 3-7

pid 7266's current affinity list: 3-7

pid 8346's current affinity list: 3-7

pid 8347's current affinity list: 0-9

pid 8348's current affinity list: 0-9

pid 8349's current affinity list: 0-9

pid 8350's current affinity list: 0-9

pid 8351's current affinity list: 1

pid 8352's current affinity list: 2

pid 8353's current affinity list: 3

pid 11867's current affinity list: 0-9

Polling threads are will pinned to a single CPU.

## VRouter vNIC interface creation:

Here is the VIF interface creation by the vRouter:

* VirtIO device name is given: *tapbd77b8fc-70*
* Queue mapping is given (which lcore will poll the queue):
  + TX Queue mapping: *lcore N TX to HW queue M*
  + RX Queue mapping: *lcore N RX from HW queue M*

2020-03-28 15:59:18,598 VROUTER: Adding vif 7 (gen. 10) virtual device tapbd77b8fc-70

2020-03-28 15:59:18,598 VROUTER: lcore 10 TX to HW queue 0

2020-03-28 15:59:18,598 VROUTER: lcore 11 TX to HW queue 1

2020-03-28 15:59:18,598 VROUTER: lcore 12 TX to HW queue 2

2020-03-28 15:59:18,598 VROUTER: lcore 8 TX to HW queue 3

2020-03-28 15:59:18,598 VROUTER: lcore 9 TX to HW queue 4

2020-03-28 15:59:18,598 VROUTER: lcore 10 RX from HW queue 0

2020-03-28 15:59:18,598 VROUTER: lcore 11 RX from HW queue 1

2020-03-28 15:59:18,598 VROUTER: lcore 12 RX from HW queue 2

We can notice that we are seeing as many RX Queue that the number of polling cores configured onto the vRouter.

We have 2 TX queues more than RX queues. These are queues bound to lcore 8 and lcore 9 (service threads)

## VRouter hostuser interface creation:

### Vhost user socket creation

Here is the vhostuser socket creation by the vRouter:

* Socket file name is given: */var/run/vrouter/uvh\_vif\_tapbd77b8fc-70*
* Socket file descriptor is given: *FD is 58*

2020-03-28 15:59:18,598 UVHOST: Adding vif 7 virtual device tapbd77b8fc-70

2020-03-28 15:59:18,598 UVHOST: vif (server) 7 socket tapbd77b8fc-70 FD is 58

2020-03-28 15:59:18,598 UVHOST: error connecting uvhost socket FD 58 to /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70: No such file or directory (2)

...

2020-03-28 15:59:23,598 UVHOST: connected to /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70 for uvhost socket FD 58

We can have a look onto this created socket using provided File Descriptor:

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/58

285931270 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/58 -> socket:[285654089]

We can see than the socket is well established, and we are displaying both process connected to it using this command:

# ss -x -p | grep 285654089

Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port

u\_str ESTAB 0 0 \* 285654089 \* 285648356 users:(("contrail-vroute",pid=6663,fd=58))

u\_str ESTAB 0 0 /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70 285648356 \* 285654089 users:(("qemu-kvm",pid=238022,fd=15))

With:

* Netid; u\_str = Stream Socket. These sockets are used for interprocess communication.
* Recv-Q The count of bytes not copied by the user program connected to this socket.
* Send-Q The count of bytes not acknowledged by the remote host.

Then using this last command, we can have a look on remote file descriptor:

# ls -li /proc/238022/fd/15

285919052 lrwx------. 1 qemu qemu 64 Mar 28 17:20 /proc/238022/fd/15 -> socket:[285648356]

We can also collect similar information using following commands:

# netstat -nxp | head -1 ; netstat -nxp | grep tapbd77b8fc-70

Active UNIX domain sockets (w/o servers)

Proto RefCnt Flags Type State I-Node PID/Program name Path

unix 3 [ ] STREAM CONNECTED 285648356 238022/qemu-kvm /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70

Raw info about Unix sockets are given in /proc/net/unix:

# head -n1 < /proc/net/unix ; grep 285648356 /proc/net/unix

Num RefCount Protocol Flags Type St Inode Path

ffff9e9dc9a7f800: 00000003 00000000 00000000 0001 03 285648356 /var/run/vrouter/uvh\_vif\_tapbd77b8fc-70

We can also collect information about QEMU process which is listening on this Socket:

# lsof -nn 2> /dev/null | grep uvh\_vif\_tapbd77b8fc-70 | awk '{ print $2, " ", $3, " ", $1 }' | uniq

238022 qemu qemu-kvm

238022 238044 qemu-kvm

238022 238045 qemu-kvm

238022 238087 CPU

238022 238089 vnc\_worke

# ps -Tp `pgrep -f instance-00000008`

PID SPID TTY TIME CMD

238022 238022 ? 00:03:44 qemu-kvm

238022 238044 ? 00:00:00 qemu-kvm

238022 238045 ? 00:00:00 qemu-kvm

238022 238087 ? 00:00:41 CPU 0/KVM

238022 238089 ? 00:00:00 vnc\_worker

# ps -eT -o psr,tid,comm,pid,ppid,cmd,pcpu,stat | grep 238087

9 238087 CPU 0/KVM 238022 5082 /usr/libexec/qemu-kvm -name 0.5 Sl

To get more info about /proc/<PID>/fd directory, have a look into:

<http://man7.org/linux/man-pages/man5/proc.5.html>

### Vhost user vrings creation (vnic queues)

vRings are the Virtual interface queues used to transfer packets between virtual instances virtual NIC and the vRouter. We can see vRing activation into vrouter DPDK logs:

2020-03-28 15:59:24,578 UVHOST: Client \_tapbd77b8fc-70: handling message 1

2020-03-28 15:59:24,578 UVHOST: GET FEATURES: returns 0x44429983

2020-03-28 15:59:24,578 UVHOST: Client \_tapbd77b8fc-70: handling message 15

2020-03-28 15:59:24,578 UVHOST: GET PROTOCOL FEATURES: returns 0x3

2020-03-28 15:59:24,578 UVHOST: Client \_tapbd77b8fc-70: handling message 16

2020-03-28 15:59:24,578 UVHOST: SET PROTOCOL FEATURES: 0x3

2020-03-28 15:59:24,578 UVHOST: Client \_tapbd77b8fc-70: **handling message 17**

2020-03-28 15:59:24,578 UVHOST: **GET QUEUE NUM**: returns 0x10

2020-03-28 15:59:24,578 UVHOST: Client \_tapbd77b8fc-70: ***no handler defined for message 3***

2020-03-28 15:59:24,578 UVHOST: Client \_tapbd77b8fc-70: handling message 1

2020-03-28 15:59:24,578 UVHOST: GET FEATURES: returns 0x44429983

Here are the User space VHOST message bound for each ID :

|  |  |
| --- | --- |
| **Message** | **ID** |
| VHOST\_USER\_GET\_FEATURES | 1 |
| **VHOST\_USER\_SET\_FEATURES** | **2** |
| **VHOST\_USER\_SET\_OWNER** | **3** |
| VHOST\_USER\_SET\_VRING\_CALL | 13 |
| VHOST\_USER\_GET\_PROTOCOL\_FEATURES | 15 |
| **VHOST\_USER\_SET\_PROTOCOL\_FEATURES** | **16** |
| **VHOST\_USER\_GET\_QUEUE\_NUM** | **17** |
| **VHOST\_USER\_SET\_VRING\_ENABLE** | **18** |

Then 2 VRings are started to be configured. The vrings correspond to both transmit and receive queues. The receive queues are the odd numbers, transmit queues are the even numbers. Here a single Q is started to be configured:

2020-03-28 15:59:24,579 UVHOST: Client \_tapbd77b8fc-70: handling message 13

2020-03-28 15:59:24,579 UVHOST: SET VRING CALL: vring 0 FD 65

2020-03-28 15:59:24,579 UVHOST: Client \_tapbd77b8fc-70: handling message 13

2020-03-28 15:59:24,579 UVHOST: SET VRING CALL: vring 1 FD 67

These are eventfd sockets (IRQ management):

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/65

285931277 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/65 -> anon\_inode:[eventfd]

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/67

285931279 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/67 -> anon\_inode:[eventfd]

Then vrings are enabled:

2020-03-28 15:59:33,792 UVHOST: Client \_tapbd77b8fc-70: handling message 18

2020-03-28 15:59:33,792 UVHOST: Client \_tapbd77b8fc-70: setting vring 0 ready state 1

2020-03-28 15:59:33,792 UVHOST: Client \_tapbd77b8fc-70: handling message 18

2020-03-28 15:59:33,792 UVHOST: Client \_tapbd77b8fc-70: setting vring 1 ready state 1

vRouter is enabling or disabling queues according to QEMU VirtIO requests.

* ready state 1 = enabled.
* ready state 0 = disabled

If the VirtIO interface is created with more Queues than the vrouter is able to manage (number of vrouter polling core), following error message will be issued:

*2019-09-24 16:37:46,693 UVHOST: Client* \_tapbd77b8fc-70*: handling message 18*

*2019-09-24 16:37:46,693 UVHOST: vr\_uvhm\_set\_vring\_enable: Can not disable TX queue* ***N*** *(only* ***M*** *queues)*

*2019-09-24 16:37:46,693 UVHOST: Client* \_tapbd77b8fc-70*: handling message 18*

*2019-09-24 16:37:46,693 UVHOST: vr\_uvhm\_set\_vring\_enable: Can not disable RX queue* ***N*** *(only* ***M*** *queues)*

2020-03-28 15:59:33,798 UVHOST: Client \_tapbd77b8fc-70: **handling message 2**

2020-03-28 15:59:33,798 UVHOST: **SET FEATURES**(original): 0x40009983

2020-03-28 15:59:33,799 UVHOST: **SET FEATURES**( updated): 0x40009983

### Vhost user vrings memory mapping

Next the vRouter need to get infor about vRing memory.

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: handling message 5

2020-03-28 15:59:33,799 UVHOST: SET MEM TABLE:

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: unmapping 0 memory regions:

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: mapping 1 memory regions:

2020-03-28 15:59:33,799 UVHOST: 0: FD 82 addr 0x0 size 0x4000000 off 0x0

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/82

285931293 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/82 -> anon\_inode:[eventfd]

|  |  |
| --- | --- |
| Message | ID |
| VHOST\_USER\_SET\_MEM\_TABLE | 5 |
| VHOST\_USER\_SET\_VRING\_NUM | 8 |
| **VHOST\_USER\_SET\_VRING\_ADDR** | **9** |
| VHOST\_USER\_SET\_VRING\_BASE | 10 |
| **VHOST\_USER\_SET\_VRING\_KICK** | **12** |
| **VHOST\_USER\_SET\_VRING\_CALL** | **13** |
| **VHOST\_USER\_SET\_VRING\_ENABLE** | **18** |

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: handling message 8

2020-03-28 15:59:33,799 UVHOST: SET VRING NUM: vring 0 **num 512**

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: handling message 10

2020-03-28 15:59:33,799 UVHOST: SET VRING BASE: vring 0 base 0

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: handling message 9

2020-03-28 15:59:33,799 UVHOST: SET VRING ADDR: vring 0 flags 0x0 desc 0x7f8383d78000 used 0x7f8383d7b000 avail 0x7f8383d7a000

2020-03-28 15:59:33,799 UVHOST: Client \_tapbd77b8fc-70: vring 0 is ready

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: no handler defined **for message 12**

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: **handling message 13**

2020-03-28 15:59:33,800 UVHOST: **SET VRING CALL**: vring 0 FD 82

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/82

285931293 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/82 -> anon\_inode:[eventfd]

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: handling message 8

2020-03-28 15:59:33,800 UVHOST: SET VRING NUM: vring 1 **num 512**

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: handling message 10

2020-03-28 15:59:33,800 UVHOST: SET VRING BASE: vring 1 base 0

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: handling message 9

2020-03-28 15:59:33,800 UVHOST: SET VRING ADDR: vring 1 flags 0x0 desc 0x7f8383c40000 used 0x7f8383c43000 avail 0x7f8383c42000

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: vring 1 is ready

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: no handler defined for **message 12**

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: **handling message 13**

2020-03-28 15:59:33,800 UVHOST: **SET VRING CALL**: vring 1 FD 65

# ls -li /proc/$(pidof contrail-vrouter-dpdk)/fd/65

285931277 lrwx------. 1 root root 64 Mar 28 17:20 /proc/6663/fd/65 -> anon\_inode:[eventfd]

Then vrings are enabled:

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: handling message 18

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: setting vring 0 ready state 1

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: handling message 18

2020-03-28 15:59:33,800 UVHOST: Client \_tapbd77b8fc-70: setting vring 1 ready state 1

vRouter is enabling or disabling queues according to QEMU VirtIO requests.

* ready state 1 = enabled.
* ready state 0 = disabled

Queue size are defined into Virtual Machine network interface configuration:

-chardev socket,id=charnet0,path=/var/run/vrouter/uvh\_vif\_tapbd77b8fc-70,server \

-netdev vhost-user,chardev=charnet0,id=hostnet0 \

-device virtio-net-pci,rx\_queue\_size=512,tx\_queue\_size=512,netdev=hostnet0,id=net0,mac=02:bd:77:b8:fc:70,bus=pci.0,addr=0x3 \