Table des matières

[chapter 5: Contrail DPDK vrouter troubleshooting 2](#_Toc51175613)

[DPDK dataplane log files 2](#_Toc51175614)

[DPDK vrouter main parameters 2](#_Toc51175615)

[Polling core allocation 3](#_Toc51175616)

[Internal Load Balancing 3](#_Toc51175617)

[Virtual Interface queues 4](#_Toc51175618)

[DPDK physical interface list 7](#_Toc51175619)

[DPDK vrouter packet capture 8](#_Toc51175620)

[DPDK drop statistics tool 8](#_Toc51175621)

[DPDK info tool 9](#_Toc51175622)

[Hugepages Memory 10](#_Toc51175623)

# chapter 5: Contrail DPDK vrouter troubleshooting

In this section we will describe the specific tools to be used to troubleshoot Contrail DPDK vrouter. Generic tools that are common to both Kernel and DPDK dataplane won’t be described in this section (contrail-status, vif; nh, rt, mpls commands for instance)

We are inviting our readers interested to get more details about generic troubleshooting Contrail vrouter; to have a look onto Juniper Contrail or Tungsten Fabric web sites.

In this section we will describe:

* DPDK physical interface list
* DPDK dataplane log files
* DPDK vrouter packet capture
* DPDK drop statistics tool
* DPDK info tool
* Hugepages memory

## DPDK dataplane log files

Contrails DPDK vrouter dataplane log file is located at /var/log/containers/contrail/dpdk/contrail-vrouter-dpdk.log. Lots of interesting information can be collected into this log file.

### DPDK vrouter main parameters

Each time the vrouter is started, main configuration enforced parameters are listed in the log file during the vrouter initialization. We can see the DPDK library version that has be use to build the DPDK vrouter binary program, but also the different values that are used for the different setup parameters.

Here for instance, a vrouter Contrail release 2008 is running DPDK Version 19.11. Nexthops limit parameter has been decreased to 32768 instead of default value (65536).

2020-09-15 20:27:22,381 VROUTER: vRouter version: {"build-info": [{"build-time": "2020-09-15 01:07:25.101398", "build-hostname": "contrail-build-r2008-rhel-115-generic-20200914170527.novalocal", "build-user": "contrail-builder", "build-version": "2008"}]}

2020-09-15 20:27:22,382 VROUTER: DPDK version: DPDK 19.11.0

2020-09-15 20:27:23,046 VROUTER: Log file : /var/log/contrail/contrail-vrouter-dpdk.log

2020-09-15 20:27:23,046 VROUTER: Bridge Table limit: 262144

2020-09-15 20:27:23,046 VROUTER: Bridge Table overflow limit: 53248

2020-09-15 20:27:23,046 VROUTER: Flow Table limit: 524288

2020-09-15 20:27:23,046 VROUTER: Flow Table overflow limit: 105472

2020-09-15 20:27:23,046 VROUTER: MPLS labels limit: 5120

2020-09-15 20:27:23,046 VROUTER: Nexthops limit: 32768

2020-09-15 20:27:23,046 VROUTER: VRF tables limit: 4096

2020-09-15 20:27:23,046 VROUTER: Packet pool size: 16384

2020-09-15 20:27:23,046 VROUTER: PMD Tx Descriptor size: 128

2020-09-15 20:27:23,046 VROUTER: PMD Rx Descriptor size: 128

2020-09-15 20:27:23,046 VROUTER: Maximum packet size: 9216

2020-09-15 20:27:23,046 VROUTER: Maximum log buffer size: 200

2020-09-15 20:27:23,046 VROUTER: VR\_DPDK\_RX\_RING\_SZ: 2048

2020-09-15 20:27:23,046 VROUTER: VR\_DPDK\_TX\_RING\_SZ: 2048

2020-09-15 20:27:23,046 VROUTER: VR\_DPDK\_YIELD\_OPTION: 0

2020-09-15 20:27:23,046 VROUTER: VR\_SERVICE\_CORE\_MASK: 0x10

2020-09-15 20:27:23,046 VROUTER: VR\_DPDK\_CTRL\_THREAD\_MASK: 0x10

2020-09-15 20:27:23,046 VROUTER: Unconditional Close Flow on TCP RST: 0

2020-09-15 20:27:23,046 VROUTER: EAL arguments:

2020-09-15 20:27:23,046 VROUTER: -n "4"

2020-09-15 20:27:23,046 VROUTER: --socket-mem "1024"

### Polling core allocation

CPU core used by the different DPDK vrouter threads are written into the Contrail vrouter DPDK log file. They are described with a specific syntax:

* 3 first logical numbers (0 to 2) are used for “service lcores”
* 5 next ones (3 to 7) are booked for IO lcores
* 2 next ones (8 to 9) are “lcores with TX queues”.
* Remaining logical cores (with number 10 and above) are polling and processing logical cores.

DPDK vrouter threads lcore are pinned onto the compute node CPU according following DPDK vrouter parameters:

* lcore 0 to 2 and above are defined with ***CPU\_LIST***
* lcore 8 to 9 and above are defined with ***SERVICE\_CORE\_MASK***
* lcore numbers 10 and above are defined with ***DPDK\_CTRL\_THREAD\_MASK***

In the example below, polling core are CPU 2, 4, 6 and 8 (used by lcore 10 to 13):

2020-09-16 09:06:50,886 VROUTER: --lcores  "(0-2)@(10,34),(8-9)@(10,34),**10@2,11@4,12@6,13@8**

We can also see, that service threads and DPDK control threads have been pinned on CPU 10 and 34.

### Internal Load Balancing

In some situation the polling core performs a new hash calculation to distribute the polled packet to another processing core. This is a DPDK pipeline model implemented into the vrouter. In this situation, the polling core will never select itself to process a polled packet.

The selection of available processing cores for each polling core can be seen in the vrouter DPDK log file.

2020-01-07 13:08:01,403 VROUTER: Lcore 10: distributing MPLSoGRE packets to [11,12,13]

2020-01-07 13:08:01,403 VROUTER: Lcore 11: distributing MPLSoGRE packets to [10,12,13]

2020-01-07 13:08:01,403 VROUTER: Lcore 12: distributing MPLSoGRE packets to [10,11,13]

2020-01-07 13:08:01,404 VROUTER: Lcore 13: distributing MPLSoGRE packets to [10,11,12]

It is written MPLSoGRE, but it applies to all packets that are distributed via hashing by the polling core.



### Virtual Interface queues

Each time a new virtual interface is connected to the vrouter, a vif port is created on the vrouter with the same number of queues as the number of polling CPU (specified in CPU\_LIST). Each queue created is handled by a only one of the vrouter polling core. So, for each created vif, we have a one to one mapping between vrouter polling cores and RX queues. This mapping can be seen in DPDK vrouter logs:

2019-09-24 16:36:50,011 VROUTER: Adding vif 8 (gen. 37) virtual device tap66e68bc1-a9

…

2019-09-24 16:36:50,012 VROUTER: lcore 12 RX from HW queue 0

2019-09-24 16:36:50,012 VROUTER: lcore 13 RX from HW queue 1

2019-09-24 16:36:50,012 VROUTER: lcore 10 RX from HW queue 2

2019-09-24 16:36:50,012 VROUTER: lcore 11 RX from HW queue 3

Here the vif interface 0/8 is created in order to connect the virtual NIC tap66e68bc1-a9 to the vrouter. This vif is created with 4 queues, q0,q1,q2 and q3 which are respectively handled by polling cores 12,13,10 and 11.

The connected virtual machine interface can have a different number of queues than the number configured onto the vrouter vif interface it is connected to. Several case have to be taken into consideration:

* The connected virtual machine network interface has less queues than the number of queues created on the vrouter interface. Some vrouter interface queues won’t be enabled.
* The connected virtual machine network interface has the same number of queues than the number of queues created on the vrouter interface. All vrouter interface queues will be enabled.
* The connected virtual machine network interface has more queues than the number of queues created on the vrouter interface. This is an unsupported case. Some virtual machine interface queues won’t be polled as there is no matching queue on the vif it is connected to. Some queues have to be disabled on the virtual machine NIC network interface.

When a polling queue is enabled on the vrouter, a ring activation message is generated in the Contrail DPDK log file.

The vrings correspond to both transmit and receive queues:

* the transmit queues are the even numbers. Divide them by 2 to get the queue number. i.e. vring 0 is TX queue 0, vring 2 is TX queue 1, …
* the receive queues are the odd numbers. Divide them by 2 (discard the remainder) to get the queue number. i.e. vring 1 is RX queue 0, vring 3 is RX queue 1, …
* ready state 1 = enabled. ready state 0 = disabled

In the example below, only 1 RX (and TX) queue is enabled on the vrouter vif interface. A single queue virtual machine interface is connected to the vrouter port:

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 0 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 1 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 2 ready state 0

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 3 ready state 0

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 4 ready state 0

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 5 ready state 0

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 6 ready state 0

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 7 ready state 0



In the example hereafter, 4 RX (and TX) queues are enabled on the vrouter vif interface. But a virtual machine interface having more than 4 queues is connected to the vrouter port:

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 0 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 1 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 2 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 3 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 4 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 5 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 6 ready state 1

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: setting vring 7 ready state 1

2019-09-24 16:37:46,693 UVHOST: vr\_uvhm\_set\_vring\_enable: Can not disable TX queue 4 (only 4 queues)

2019-09-24 16:37:46,693 UVHOST: Client \_tap66e68bc1-a9: handling message 18

2019-09-24 16:37:46,693 UVHOST: vr\_uvhm\_set\_vring\_enable: Can not disable RX queue 4 (only 4 queues)

….

As there are more than 4 queues on the virtual machine interface, some queues must not be enabled on the virtual machine NIC. Unfortunately, these queues can’t be disabled on the virtual machine. Therefore, this setup is faulty.



## dpdk\_nic\_bind.py

Using dpdk\_nic\_bind.py python script provided into Contrail DPDK vrouter container, we can get the current Ethernet devices that are used by Contrail vrouter. We can also see all devives that are compatible with DPDK and that could be used for the vrouter physical port.

Here we have 2 PCI Ethernet ports, 0000:02:01.0 and 0000:02:02.0 that have been selected by the vrouter.

$ sudo docker exec contrail-vrouter-agent-dpdk /opt/contrail/bin/dpdk\_nic\_bind.py -s

Network devices using DPDK-compatible driver

============================================

0000:02:01.0 '82540EM Gigabit Ethernet Controller' drv=uio\_pci\_generic unused=e1000

0000:02:02.0 '82540EM Gigabit Ethernet Controller' drv=uio\_pci\_generic unused=e1000

Network devices using kernel driver

===================================

0000:03:00.0 'Virtio network device' if= drv=virtio-pci unused=virtio\_pci,uio\_pci\_generic

Other network devices

=====================

<none>

PS: this tool has to be run into contrail DPDK vrouter container.

## vifdump

When DPDK vrouter is used, Physical and Virtual NIC connected to the vrouter are no more seen in the Linux Operating system of the compute node, as they are all been detached from the Linux Kernel to be used in user space by the vrouter DPDK application.

All Linux regular interface commands or tools, like ifconfig, tcpdump can no more be used with all the ports connected onto the vrouter. Therefore, a traffic capture tool, named vifdump, has been developed by Juniper in order to be able to capture the traffic on the vrouter vif interface instead of the user interface side (physical NIC or tap interface associated to the virtual machine NIC).

As vifdump is supported by both DPDK and Kernel mode vrouter, it also offer a unified way to investigate about the traffic which is entering and is delivered by the vrouter.

Vifdump syntqx is:

vifdump -i <vif number> [<vifdump options>]

with:

<vif number>: is the second digit of vif 0/N interface. For instance, 4 for vif 0/4.

<vifdump option>: are capture option using tcpdump syntax.

For instance, in order to capture in the traffic crossing vif 0/4, we have to use:

# vifdump -i 4

vif0/4      PMD: tap7f8e1617-ae

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on mon4, link-type EN10MB (Ethernet), capture size 262144 bytes

15:01:17.004641 ARP, Request who-has 192.168.0.17 tell 192.168.0.2, length 28

15:01:19.004610 ARP, Request who-has 192.168.0.17 tell 192.168.0.2, length 28

vifdump is taking the same options as tcpdump regular Linux Tools. In order to capture into a file the traffic processed by vrouter vif 0/4 interface we can run:

# vifdump -i 4 w test.pcap

## dropstats

With dropstats command we can get details about drops performed by the vrouter.

We can get all parameters supported by this command using :

$ sudo docker exec contrail-vrouter-agent-dpdk dropstats --help

When some packet loss issue is encountered, this is a good starting point to get an overview of all drops that have been performed on the vrouter.

dropstats command run without any arguments is returning drop statistics for all vrouter cores :

$ sudo docker exec contrail-vrouter-agent-dpdk dropstats

…

In order to get statistics for a given vrouter core, we can use the --core option. Keep in mind that 10 is the first processing vrouter lcore:

$ sudo docker exec contrail-vrouter-agent-dpdk dropstats --core 10

...

In order to clear stats counters on all cores we can use this command:

$ sudo docker exec contrail-vrouter-agent-dpdk dropstats --clear

Dropstats is also supporting a log option (said as droplog), that is allowing to get more details about the packets that have been dropped by the vrouter and the reason why:

In order to display all packets dropped by the vrouter:

$

In order to display all packets dropped by a given core of the vrouter:

$ sudo docker exec contrail-vrouter-agent-dpdk dropstats -log <core number>

PS: this tool has to be run into contrail DPDK vrouter container.

## dpdkinfo

Since Contrail 20.08 a new troubleshooting tool, named *dpdkinfo* is proposed with Contrail. Using this tool Contrail operators can collect information about DPDK connectivity (physical NIC bond), DPDK library release, and some other statistics.

This dpdk command must be started from “contrail-tools” environment.

$ sudo contrail-tools

(contrail-tools)[root@jnprctdpdk01 /]$ dpdkinfo

Usage: dpdkinfo [--help]

   --version|-v                       Show DPDK Version

   --bond|-b                          Show Master/Slave bond information

   --lacp|-l  <all/conf>              Show LACP information from DPDK

   --mempool|-m  <all/<mempool-name>> Show Mempool information

   --stats|-n  <eth>                  Show Stats information

   --xstats|-x   < =all/ =0(Master)/ =1(Slave(0))/ =2(Slave(1))>

                                      Show Extended Stats information

   --lcore|-c                         Show Lcore information

   --app|-a                           Show App information

Optional: --buffsz <value>            Send output buffer size (less than 1000Mb)

This is also possible to run dpdkinfo command directly from Linux OS, using following syntax:

$ sudo contrail-tools dpdkinfo

Lots of information, but not only, that are present in DPDK log files are displayed in an easiest way.

## Hugepages Memory

Huge pages memory is an important piece of DPDK vrouter. We have to take care that enough hugepages are configured on the system and that free hugepages are still available for a new need.

Several hugepages sizes can be configured onto the Compute node. In order to check the number of allocated and free hugepages for each numa node can be retrieved from /sys filesystem. With following commands we can retrieve the number of 2MB and 1GB configured and free hugepages on each numa node (numa0 and numa1):

# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr\_hugepages

# cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/free\_hugepages

# cat /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr\_hugepages

# cat /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/free\_hugepages

# cat /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr\_hugepages

# cat /sys/devices/system/node/node1/hugepages/hugepages-2048kB/free\_hugepages

# cat /sys/devices/system/node/node1/hugepages/hugepages-1048576kB/nr\_hugepages

# cat /sys/devices/system/node/node1/hugepages/hugepages-1048576kB/free\_hugepages

It’s also interesting to have a look onto the hugetblfs mountpoint which is used by Contrail vRouter to get access to hugepages:

$ grep hugetlbfs /etc/fstab

hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,seclabel,pagesize=1G)

$ cat /proc/mounts | grep hugepage

hugetlbfs /dev/hugepages hugetlbfs rw,seclabel,relatime,pagesize=1G 0 0

Contrail vRouter is only using hugepage having all the same size. The hugepage size to be used by the vrouter is:

* 2MB: if no pagesize information is given into hugetblfs
* Specific: if a pagesize information is given into hugetblfs

If not enough hugepages are available with size specified by the hugetblfs mountpoint (2MB or specific), the vrouter startup will fail to start.