# Troubleshooting

**Version 24.8.1, 11 Sept 2024**

# Table of Contents

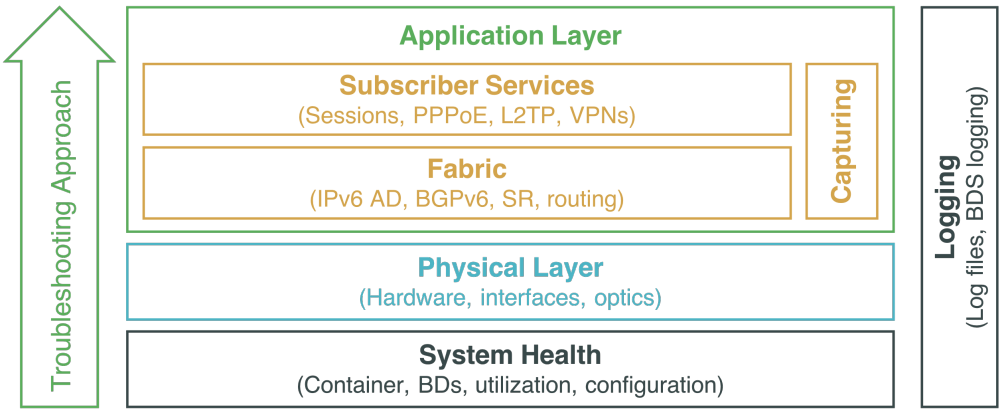
# 1. Troubleshooting Guide Overview

## Scope

This troubleshooting guide outlines how to troubleshoot devices running RBFS. It is intended for Network Operations teams typically performing monitoring, 1st or 2nd level support in a production environment. This guide helps the operator to identify if it is an RBFS issue, and to narrow down the device and the process or application that causes the issue. Troubleshooting for network engineering, or in-depth analysis of BDS tables is out-of-scope of this guide. It is assumed that the user of this guide is familiar with the basics of RBFS, as well as the basics of using a Linux-based network operating system. This guide will be continuously extended to cover failure scenarios which have occurred in production environments.

## Troubleshooting Approach

There is no strict order in which various troubleshooting steps must be performed. The best approach will always depend on the nature of the problem observed. Generally speaking it is recommended to first perform some basic system health checks as described in section 2. Next, it's a general best network troubleshooting practice to verify the OSI layers from bottom to top. Applying this best practice to an RBFS fabric, it is recommended to first verify the physical layer as described in section 3, then verify the operation of the fabric as described in section 4.1, and finally verify services that run on top of the fabric as described in section 4.2. Fabric and services protocol operation can be verified using the built-in RBFS capture tool explained in section 4.3. Section 5 describes RBFS logging capabilities that apply to all layers. Finally, section 6 provides guidance when and how to report an issue to RtBrick. If you already know or have an indication what seems to be the trouble, you can of course directly proceed to that area and skip all prior steps proposed in this guide.




# Supported Platforms

Not all features are necessarily supported on each hardware platform. Refer to the *Platform Guide* for the features and the sub-features that are or are not supported by each platform.

# 2. System Health

## Container

The steps described in this section are performed on the host OS. For logging into the host OS instead of into the Linux container, use SSH port 1022 instead of the default port 22. Example:

```
ssh supervisor@10.10.0.100 -p 1022
```

## Checking the Container Status

The RBFS services run in a Linux container (LXC). If you are able to log in into the container, obviously the container is running. If you are not able to log in, you can verify the status of the container on the host OS, i.e. ONL on hardware switches, as follows:

```
supervisor@spine1:~$ sudo lxc-ls -f
NAME    STATE   AUTOSTART GROUPS IPV4      IPV6
rtbrick RUNNING 1         -      10.0.3.10 -
```

On a hardware switch, there will be a single container called "rtbrick" and the state shall be "Running". If the state is "Stopped" or "Failed", the container has failed and the system is in a non-operational state.

## Recovering from a Container Failure

If the container exists but is not running, you can start it using the rtb-image tool:

```
supervisor@spine1:~$ sudo rtb-image container start rtbrick
```

Alternatively you can use the following lxc command:

```
supervisor@spine1:~$ sudo lxc-start -n rtbrick
```

If the container does not exist, or if starting it fail, you can try to recover by restarting the device at the ONL layer:

```
supervisor@spine1:~$ sudo reboot
```

# Brick Daemons

## Checking the BD’s Status

RBFS runs multiple Brick Daemons (BD). You can verify the status of the daemons using the Ubuntu system control (systemctl) or using the RBFS show command: 'show bd running-status'. The following commands will show all rtbrick services. The status should be "running":

Example 1: Show output using the Ubuntu system control command

```
supervisor@rtbrick:~$ sudo systemctl list-units | grep rtbrick
var-rtbrick-auth.mount                   loaded active     mounted   /var/rtbrick/auth
rtbrick-alertmanager.service             loaded active     running   rtbrick-alertmanager service
rtbrick-bgp.appd.1.service               loaded active     running   rtbrick-bgp.appd.1 service
rtbrick-bgp.iod.1.service                loaded active     running   rtbrick-bgp.iod.1 service
<...>
```

Example 2: Show output using the 'show bd running-status' command

```
supervisor@rtbrick: op> show bd running-status
Daemon              Status
alertmanager        running
bgp.appd.1          running
bgp.iod.1           running
confd               running
etcd                running
fibd                running
<...>
```

Please note the supported BDs differ by role and may change in a future release. You can display further details as shown in the following example:

```
supervisor@rtbrick:~$ sudo systemctl status rtbrick-fibd.service

 rtbrick-fibd.service - rtbrick-fibd service
   Loaded: loaded (/lib/systemd/system/rtbrick-fibd.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-10-19 05:01:16 UTC; 4h 41min ago
  Process: 248 ExecStartPost=/bin/bash -c [ -f "/usr/local/bin/rtbrick-ems-service-event" ] && {
/usr/bin/python3 /usr/local/bin/rtbr
  Process: 240 ExecStartPre=/bin/mkdir -p /var/run/rtbrick/fibd_pipe (code=exited, status=0/SUCCESS)
  Process: 225 ExecStartPre=/usr/local/bin/rtbrick-bcm-sdk-symlink.bash (code=exited, status=0/SUCCESS)
  Process: 150 ExecStartPre=/usr/local/bin/rtbrick-vpp-startup-conf.bash (code=exited, status=0/SUCCESS)
 Main PID: 246 (vpp_main)
   CGroup: /system.slice/rtbrick-fibd.service
           246 /usr/local/bin/bd -i /etc/rtbrick/bd/config/fibd.json
```

If the status is "failed", the respective service or daemon has failed. A daemon failure is a major issue, and depending on which BD has failed, the system is likely to be in a non-operational state. If a BD has failed, inspect the system log (syslog) file as well as the respective BD log file as described in section Logging, then proceed to section Verifying the Configuration, and finally report the failure to RtBrick as described in section Reporting Issues to RtBrick.

## Core Dump

If a BD fails, it shall create a core dump file. A core dump is a file containing a process's state like its address space (memory) when the process terminates unexpectedly. These files are located in /var/crash/rtbrick. If you have identified or suspect a BD failure, navigate to this directory and check for core dump files:

```
supervisor@rtbrick:~$ cd /var/crash/rtbrick/
supervisor@rtbrick:/var/crash/rtbrick$ ls -l
-rw-r--r-- 1 root root 3236888576 Apr  9 10:17 core.fibd_136_2020-04-09_10-16-52
```

If there is a core dump file, you can decode it using the GNU debugger tool like in the following example:

```
supervisor@rtbrick:/var/crash/rtbrick$ sudo gdb bd core.fibd_136_2020-04-09_10-16-52
<...>
```

At the resulting 'gdb' prompt, type and enter 'bt' for backtrace:

```
(gdb) bt
<...>
```

Report the resulting output to RtBrick as described in section 6. Analysing the core dump file will typically require support from RtBrick and is beyond the scope of this guide.

## Recovering from a BD Failure

If a brick daemon fails, RBFS will automatically restart it. If the automatic restart does not succeed, you can use the Ubuntu system control to start a daemon like in the following example:

```
supervisor@rtbrick:~$ sudo systemctl start rtbrick-fibd.service
```

Alternatively you can recover from a BD failure by rebooting the container from the Linux container shell:

```
supervisor@rtbrick:~$ sudo reboot
```

# Running Configuration

## Verifying the Configuration

A missing running configuration is another possible problem scenario. There are several reasons why the system might be missing its configuration. Posting the running configuration might have failed for example due an invalid configuration syntax, or there was a connectivity issue between the device and the provisioning system.

You can easily verify via CLI if there is a running configuration:

```
supervisor@rtbrick: op> show config
```

If you suspect a configuration load issue, inspect the confd logs as well as the CtrlD log as describe in section 5.

## Restoring a Configuration

It depends on the customer deployment scenario how a running configuration shall be applied or restored in case of an issue.

If the device already had a configuration previously, and you have saved it in a file, you can simply load it via the CLI:

```
supervisor@rtbrick: cfg> load config spine1-2020-10-19.json
```

If the device already had a configuration previously, and has been configured to load the last configuration with the 'load-last-config: true' attribute, you can restore it by rebooting the container at the Linux container shell:

```
supervisor@rtbrick:~$ sudo reboot
```

Otherwise you can also copy an automatically stored running configuration file into your user directory and load it manually like in the following example:

```
supervisor@leaf1:~$ sudo cp
/var/rtbrick/commit_rollback/766e102957bf99ec79100c2acfa9dbb9/config/running_config.json running_config.json

supervisor@leaf1:~$ ls -l
total 12
-rw-r--r-- 1 root root 8398 Oct 21 09:45 running_config.json

supervisor@leaf1:~$ cli
supervisor@leaf1: op> switch-mode config
Activating syntax mode : cfg [config]
supervisor@leaf1: cfg> load config running_config.json
supervisor@leaf1: cfg> commit
```

If it's a newly deployed or upgraded device, and there is out-of-band connectivity from your network management system (for example RBMS), you can trigger the configuration from your NMS.

If it's a newly deployed or upgraded device, and the configuration shall be applied via a ZTP process from a local ZTP server, you need to reboot the device at the ONL layer in order to trigger the ZTP process:

```
supervisor@spine1:~$ sudo reboot
```

There is also an option to manually copy a configuration file to the device and into the container. If you have copied a configuration file via an out-of-band path to the ONL layer of the device, you can copy it into the container as follows. Please note the name in the directory path needs to match the name of the container, like "spine" in this example:

```
supervisor@spine1:~$ cp spine1-2020-10-19.json /var/lib/lxc/spine1/rootfs/home/supervisor/
```

Next you can load this configuration via the CLI as already described above:

```
supervisor@rtbrick: cfg> load config spine1-2020-10-19.json
```

# License

RBFS software requires a license to ensure its ligitimate use. The license will be automatically validated and enforced. After an initial grace period of 7 days, if a license is missing or expired, RBFS will be restricted. The CLI as well as the BDS APIs will not work anymore.

If all CLI commands do not work, the license might be missing or expired.

## Verifying a License

You can verify the license via the CLI:

```
supervisor@rtbrick: op> show system license
License Validity:
  License index 1:
    Start date : Fri Mar 12 06:43:25 GMT +0000 2021
    End date   : Sat Mar 12 06:43:25 GMT +0000 2022
```

The output will indicate if there is a valid license, no license, or if the license is expired.

## Restoring or Updating a License

The license is installed by configuration. If the license is missing but the device already had a license configuration previously, please restore the configuration as described in the Restoring a Configuration section above.

If the license is expired, please configure a new valid license key. If you do not have a license key yet, please contact your RtBrick support or sales representative to obtain a license.

# Control Daemon

In addition to the Brick Daemons running inside the LXC container, there are some RBFS services running on the host OS. The most important one is CtrlD (Control Daemon). CtrlD acts as the single entry point to the system. Verify the status of CtrlD:

```
supervisor@spine1:~$ sudo service rtbrick-ctrld status
[....] Checking the rtbrick ctrld service:3751
. ok
```

If the status is not "ok", restart, start, or stop and start CtrlD:

```
supervisor@spine1:~$ sudo service rtbrick-ctrld restart

supervisor@spine1:~$ sudo service rtbrick-ctrld stop
supervisor@spine1:~$ sudo service rtbrick-ctrld start
```

If the status is "ok", but you suspect an issue related to CtrlD, inspect the ctrld logs as also described in section 5:

```
supervisor@spine1:/var/log$ more rtbrick-ctrld.log
```

# System Utilization

There are cases when system-related information has to be checked: symptoms like sluggish response, or daemons/processes crashing repeatedly can mean that system resources are overutilized. In such cases first steps are to verify CPU, memory, and disk. Before, it is good to remember the general architecture of an RtBrick switch: we have the physical box, on which ONL (Open Network Linux) sits. In ONL we run the LXC container (which has Ubuntu 22.04 installed), which in turn has RBFS running inside it.

In the following sections we'll mainly concentrate on the LXC container verifications, and will specify where the commands are executed in ONL. The order in which the commands are shown in this section can also be used to do the basic system troubleshooting.

## Memory and CPU Verification

When suspecting the memory is overutilized, a quick way to verify that use free: this command provides information about unused and used memory and swap space. By providing the -h (human readable) flag, we can quickly see the memory availability of the system:

```
supervisor@rtbrick:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            31G        4.3G         24G        469M        2.3G         26G
Swap:            0B          0B          0B
```

The output from free is based on what the system reads from /proc/meminfo; of importance are the available and used columns. The description of the fields can be seen below (since man is not available on the switches):

free *command fields description*

| Name | Description |
|---|---|
| total | Total amount of memory that can be used by the applications. |
| used | Used memory, which is calculated as total - free - buffers - cache |
| free | Unused memory. |
| shared | Backwards compatibility, not used. |
| buff/cache | The combined memory used by the kernel buffers and page cache. This memory can be reclaimed at any time if needed by the applications. |
| available | Estimate of the amount of memory that is available for starting new applications. Does not account swap memory. |

free has a few useful options that can be used:

- -h - human readable: makes the output easier to read, by using the common shortcuts for units (e.g M for mebibytes, G fo gibibytes etc)
- -t - total: will display a total at the bottom of each column (basically adding physical+swap memory)
- -s - continuous print output (can be interrupted with Ctrl+C): by giving a seconds value at which the output is refreshed, you will get a continuous display of values (similar to the watch command; e.g free -s 5)

As it can be seen, free is a basic utility that displays relevant information in a compressed format. It does not offer detailed or real-time information about the running processes. As with the CPU, we can use top to obtain realtime information about memory usage

Another way to check memory consumption, as well as CPU utlization, is to use top; it is one of the most common ways to start troubleshooting a Linux-based system, because it provides a wealth of information, and, in general, is a good starting point for system troubleshooting.

Basically, this command allows users to monitor processes and CPU/memory usage, but, unlike many other commands, it does so in an interactive way. top output can be customized in many ways, depending on the information we want to

focus on, but in this guide we will not go through all the possible options top has.

A typical top output looks like the one below:

```
supervisor@rtbrick:~$ top

top - 21:12:41 up 1 day,  8:13,  1 users,  load average: 2.66, 2.72, 2.73
Tasks:  46 total,   1 running,  45 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12.4 us,  8.5 sy,  0.0 ni, 79.0 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 32785636 total, 26181044 free,  4135516 used,  2469076 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 27834804 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  260 root      20   0 10.563g 1.473g 177440 S 109.3  4.7   2280:59 vpp_main
  108 root      20   0  406552 121648  46196 S  31.9  0.4  36:49.48 pppoed.1
  168 root      20   0 3203196  40704   9824 S   7.0  0.1 117:40.63 rtbrick-resmond
  156 root      20   0  461888 134852  59328 S   2.3  0.4  38:11.79 subscriberd.1
  112 root      20   0  438592 140644  46892 S   2.0  0.4  36:35.40 igmp.iod.1
  166 root      20   0  408076 117936  43416 S   2.0  0.4  36:35.14 pim.iod.1
  176 root      20   0  392036 114596  40312 S   2.0  0.3  37:15.43 l2tpd.1
  183 root      20   0  586944 144644  51256 S   1.7  0.4  38:27.94 bgp.iod.1
  136 root      20   0  425212 147080  35636 S   1.3  0.4  22:50.32 resmond
  193 root      20   0 1453416 929168  93672 S   0.7  2.8  17:09.07 confd
  266 root      20   0  836892 107804  35424 S   0.7  0.3  17:59.60 prometheus
  <...output omitted...>
```

top output is divided in two different sections: the upper half (summary area) of the output contains statistics on processes and resource usage, while the lower half contains a list of the currently running processes. You can use the arrow keys and Page Up/Down keys to browse through the list. If you want to quit, press “q” or Ctrl+C.

On the first line, you will notice the system time and the uptime, followed by the number of users logged into the system. The first row concludes with load average over one, five and 15 minutes. “Load” means the amount of computational work a system performs. In our case, the load is the number of processes in the R (runnable) and D (uninterruptible sleep) states at any given moment.

*A word on process states*

We’ve mentioned above about a "process state". In Linux, a process may be in of these states:

- Runnable ( R ): The process is either executing on the CPU, or it is present in the run queue, ready to be executed.
- Interruptible sleep (S): The process is waiting for an event to complete.

- Uninterruptible sleep (D): The process is waiting for an I/O operation to complete.
- Zombie (Z): A process may create a number of child processes, which can exit while the parent is still active. However, the data structures that the kernel creates in memory to keep track of these child processes have to be maintained until the parent find out about the status of its child. These terminated processes who still have associated data structures in memory are called zombies.

While looking at the summary area, it is also good practice to check if any zombie processes exist (on the Tasks row); a high number of zombies is indicative of a system problem.

The CPU-related statistics are on the %CPU(s) row:

- us: Time the CPU spends executing processes for users in “user space.”
- sy: Time spent running system “kernel space” processes.
- ni: Time spent executing processes with a manually set "nice" value ( nice values determine the priority of a process relative to others - higer nice values of a process means that process will get a lower priority to run).
- id: CPU idle time.
- wa: Time the CPU spends waiting for I/O to complete.
- hi: Time spent servicing hardware interrupts.
- si: Time spent servicing software interrupts.
- st: Time lost due to running virtual machines (“steal time”).

For systems with multiple CPU cores, we can see the per-core load by pressing "1" in the interface; another useful way of visualisation is to have a graphical display of CPU load: this can be done by pressing "t" in the top interface. Below is an example of top with both "1" and "t" pressed:

```
top - 12:11:56 up 1 day, 23:12,  4 users,  load average: 2.76, 2.86, 2.97
Tasks:  58 total,   1 running,  57 sleeping,   0 stopped,   0 zombie
%Cpu0  :   5.6/1.6     7[||||||||
]
%Cpu1  :  59.8/40.2
100[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
%Cpu2  :  19.0/2.0    21[||||||||||||||||||||||
]
%Cpu3  :  20.5/3.1    24[||||||||||||||||||||||||
```

```
]
%Cpu4  :   2.1/3.4     5[|||||
]
%Cpu5  :   0.0/0.3     0[
]
%Cpu6  :   1.7/0.7     2[|||
]
%Cpu7  :   2.0/2.0     4[||||
]
GiB Mem :   31.267 total,   24.086 free,    4.059 used,    3.122 buff/cache
GiB Swap:    0.000 total,    0.000 free,    0.000 used.   26.328 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  260 root      20   0 10.783g 1.486g 191704 S 142.7  4.8 521:41.08 vpp_main
  148 root      20   0  399540 120656  44336 S   2.3  0.4   3:15.48 lldpd
  124 root      20   0  461888 137532  59312 S   2.0  0.4   8:22.26 subscriberd.1
  136 root      20   0  485168 183640  58500 S   2.0  0.6  98:59.56 igmp.iod.1
  180 root      20   0  588120 152332  52132 S   2.0  0.5   8:28.70 bgp.iod.1
  117 root      20   0  441872 137428  52904 S   1.7  0.4   9:59.74 pim.iod.1
  169 root      20   0  410648 131128  52828 S   1.7  0.4   8:50.00 pppoed.1
  171 root      20   0  425068 148420  36016 S   1.7  0.5   4:50.66 resmond
  176 root      20   0  388964 116732  40536 S   1.7  0.4   8:08.61 l2tpd.1
  <...output omitted...>
```

The next two lines are dedicated to memory information, and as expected, the “total”, “free” and “used” values have their usual meanings. The “avail mem” value is the amount of memory that can be allocated to processes without causing more swapping.

As with "t", the same thing can be done for displaying memory usage, but this time we will press "m". It is also worth noting that we can change the units in which memory values are displayed by pressing "E" (pressing repeatedly will cycle through kibibytes, mebibytes, gibibytes, tebibytes, and pebibytes). The following example shows the unit changed from kibi to gibibytes:

```
top - 12:25:19 up 1 day, 23:26,  4 users,  load average: 3.29, 3.12, 3.05
Tasks:  58 total,   1 running,  57 sleeping,   0 stopped,   0 zombie
%Cpu(s):  17.2/6.9    24[||||||||||||||||||||||||
]
GiB Mem :   31.267 total,   24.081 free,    4.063 used,    3.123 buff/cache
GiB Swap:    0.000 total,    0.000 free,    0.000 used.   26.323 avail Mem
<...output omitted...>
```

Moving to the lower half of the output (the task area), here we can see the list of processes that are running on the system. Below you can find a short explanation for each of the columns in the task area:

top *task area columns description*

| Name | Description |
|---|---|
| PID | process ID, a unique positive integer that identifies a process. |

| Name | Description |
|---|---|
| USER | the "effective" username of the user who started the process; Linux assigns a real user ID and an effective user ID to processes; the second one allows a process to act on behalf of another user (e.g.: a non-root user can elevate to root in order to install a package). |
| PR and NI | NI show the "nicety" value of a process, while the "PR" shows the scheduling priority fomr the perspective of the kernel. Higher nice values give a processa lower priority. |
| VIRT, RES, SHR and %MEM | these fields are related to the memory consumed by each process. “VIRT” is the total amount of memory consumed by a process. “RES” is the memory consumed by the process in RAM, and “%MEM” shows this value as a percentage of the total RAM available. “SHR” is the amount of memory shared with other processes. |
| S | state of the process, in single letter form. |
| TIME+ | total CPU time used by the process since it started, in seconds/100. |
| COMMAND | the name of the process. |

From a troubleshooting standpoint, check for processes that consume large amounts of CPU and/or memory. In the task area, of interest are the RES, for memory, and %CPU columns.

For a cleaner (and possibly more relevant) output of top, we can sort only the active processes to be displayed, by running top -i, and we can sort even further by CPU usage, by pressing Shift+P while running top (or by initially running top -o %CPU):

```
supervisor@rtbrick:~$ top -i

top - 23:55:20 up 1 day, 10:56,  0 users,  load average: 2.98, 2.87, 2.79
Tasks:  46 total,   1 running,  45 sleeping,   0 stopped,   0 zombie
%Cpu(s):  9.6 us,  6.5 sy,  0.0 ni, 83.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem : 32785636 total, 26168764 free,  4137340 used,  2479532 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 27832552 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  260 root      20   0 10.564g 1.474g 177952 S 110.0  4.7   2475:13 vpp_main
```

```
  112 root      20   0  438592 140908  47112 S   2.0  0.4  39:40.10 igmp.iod.1
  129 root      20   0  399544 118640  44244 S   2.0  0.4  16:40.52 lldpd
  156 root      20   0  461888 134852  59328 S   2.0  0.4  41:22.55 subscriberd.1
  166 root      20   0  408076 117936  43416 S   2.0  0.4  39:39.87 pim.iod.1
  183 root      20   0  586944 144644  51256 S   2.0  0.4  41:42.46 bgp.iod.1
  108 root      20   0  406552 122028  46576 S   1.7  0.4  39:57.25 pppoed.1
  136 root      20   0  425212 147080  35636 S   1.7  0.4  24:43.47 resmond
  176 root      20   0  392036 114596  40312 S   1.7  0.3  40:23.35 l2tpd.1
  193 root      20   0 1453420 929224  93728 S   1.0  2.8  18:35.60 confd
  168 root      20   0 3424392  41132   9824 S   0.7  0.1 127:35.58 rtbrick-resmond
  125 root      20   0  434464 132772  50396 S   0.3  0.4   2:54.41 mribd
  215 root      20   0 1527800  12736   5952 S   0.3  0.0   0:10.43 rtbrick-restcon
  266 root      20   0  837180 107808  35424 S   0.3  0.3  19:30.23 prometheus
```

As with the example above, we can also filter by any column present in the task area.

If, for example, a process is hogged and starts consuming too much CPU or memory, thus preventing the good functioning of the system, top offers the option to kill the respective process: you can press "k" and enter the process ID of the process to be killed; in the example below, the operator will terminate the cron process (make sure to run top as root when terminating processes spawned with the root user):

```
top - 07:39:16 up 1 day, 18:40,  3 users,  load average: 2.89, 2.90, 2.91
Tasks:  56 total,   2 running,  54 sleeping,   0 stopped,   0 zombie
%Cpu(s): 10.7 us,  7.5 sy,  0.0 ni, 81.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem : 32785636 total, 26042276 free,  4145480 used,  2597880 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 27766808 avail Mem
PID to signal/kill [default pid = 260] 126
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  260 root      20   0 10.559g 1.471g 181376 R 106.2  4.7 174:41.75 vpp_main
12192 supervi+  20   0   39572   3564   3048 R   6.2  0.0   0:00.01 top
    1 root      20   0   77448   8740   6840 S   0.0  0.0   0:00.27 systemd
   21 root      19  -1   70268  12152  11476 S   0.0  0.0   0:00.37 systemd-journal
   33 root      20   0   42584   3992   2980 S   0.0  0.0   0:00.20 systemd-udevd
   75 systemd+  20   0   71860   5388   4792 S   0.0  0.0   0:00.03 systemd-network
   81 systemd+  20   0   70640   5088   4532 S   0.0  0.0   0:00.05 systemd-resolve
  107 root      20   0 1604588  15864   8376 S   0.0  0.0   0:00.77 rtbrick-hostcon
  109 root      20   0  612252 189544  90092 S   0.0  0.6   0:10.33 etcd
  114 syslog    20   0  263036   4164   3652 S   0.0  0.0   0:00.10 rsyslogd
  117 root      20   0  408076 119448  43908 S   0.0  0.4   2:50.89 pim.iod.1
  120 root      20   0  503648 151880  66984 S   0.0  0.5   0:11.60 ifmd
  <...output omitted...>
```

Alternatively, ps can be used; ps is an utility for viewing information related with the processes on a system; it's abbreviated from "Process Status", and gets its information from /proc. It can be used in conjunction with tools like top, or standalone. Usually you would run ps after seeing a summary with top, for example. ps is useful to get more information about some specific process (for

example the command - or arguments - a process is executed with). Normally ps is executed with one or more options, in order to obtain a meaningful output.

*Common ps options*

| Name | Description |
|---|---|
| e | Show all processes |
| u | Select processes by effective user ID (EUID) or name |
| f | Full-format listing (there is also F - Extra full format) |
| L | Show threads |

Some common example are:

- listing all running processes, detailed

```
supervisor@rtbrick:~$ ps -ef
UID         PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 17:07 ?        00:00:00 /sbin/init
root         23     1  0 17:07 ?        00:00:00 /lib/systemd/systemd-journald
root         31     1  0 17:07 ?        00:00:00 /lib/systemd/systemd-udevd
systemd+     53     1  0 17:07 ?        00:00:00 /lib/systemd/systemd-networkd
systemd+     94     1  0 17:07 ?        00:00:00 /lib/systemd/systemd-resolved
root        136     1  1 17:07 ?        00:03:46 /usr/local/bin/bd -i /etc/rtbrick/bd/config/lldpd.json
syslog      138     1  0 17:07 ?        00:00:00 /usr/sbin/rsyslogd -n
root        139     1  1 17:07 ?        00:06:53 /usr/local/bin/bd -i /etc/rtbrick/bd/config/pim_iod.json
root        142     1  1 17:07 ?        00:07:06 /usr/local/bin/bd -i /etc/rtbrick/bd/config/bgp_iod.json
root        145     1  0 17:07 ?        00:00:18 /usr/local/bin/bd -i /etc/rtbrick/bd/config/isis_appd.json
<...output omitted...>
```

- listing all running processes and threads

```
supervisor@rtbrick:~$ ps -eLf
UID         PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
<...output omitted...>
root        136     1   136  1    1 17:07 ?        00:03:48 /usr/local/bin/bd -i
/etc/rtbrick/bd/config/lldpd.json
root        139     1   139  1    1 17:07 ?        00:06:56 /usr/local/bin/bd -i
/etc/rtbrick/bd/config/pim_iod.json
root        142     1   142  1    1 17:07 ?        00:07:09 /usr/local/bin/bd -i
/etc/rtbrick/bd/config/bgp_iod.json
root        145     1   145  0    1 17:07 ?        00:00:18 /usr/local/bin/bd -i
/etc/rtbrick/bd/config/isis_appd.json
root        147     1   147  1    1 17:07 ?        00:07:04 /usr/local/bin/bd -i
/etc/rtbrick/bd/config/isis_iod.json
<...output omitted...>
root        157     1   157  0    1 17:07 ?        00:00:18 /usr/local/bin/bd -i
/etc/rtbrick/bd/config/policy_server.json
root        160     1   160  0   19 17:07 ?        00:00:00 /usr/local/bin/rtbrick-hostconfd -proxy-onl-config
http://10.0.3.1:22022
root        160     1   202  0   19 17:07 ?        00:00:00 /usr/local/bin/rtbrick-hostconfd -proxy-onl-config
http://10.0.3.1:22022
root        160     1   203  0   19 17:07 ?        00:00:00 /usr/local/bin/rtbrick-hostconfd -proxy-onl-config
http://10.0.3.1:22022
<...output omitted...>
root        165     1   165  0    3 17:07 ?        00:00:18 /usr/bin/python3 /usr/local/bin/rtbrick-resmond-
agent
root        314     1   349  0   22 17:07 ?        00:00:01 /usr/local/bin/alertmanager
```

```
--config.file=/etc/prometheus/alertmanager.yml --storage.path=/var/db/alertmanager
root       314     1   366  0   22 17:07 ?        00:00:01 /usr/local/bin/alertmanager
--config.file=/etc/prometheus/alertmanager.yml --storage.path=/var/db/alertmanager
root       314     1   367  0   22 17:07 ?        00:00:01 /usr/local/bin/alertmanager
--config.file=/etc/prometheus/alertmanager.yml --storage.path=/var/db/alertmanager
<...output omitted...>
```

- listing all processes run by a user

```
supervisor@rtbrick:~$ ps -u syslog -f
UID         PID  PPID  C STIME TTY          TIME CMD
syslog      138     1  0 17:07 ?        00:00:00 /usr/sbin/rsyslogd -n
supervisor@rtbrick:~$
```

Along with ps you can use pgrep and pkill to search, and then terminate a process:

```
supervisor@rtbrick:~$ pgrep -u root -a
1 /sbin/init
23 /lib/systemd/systemd-journald
31 /lib/systemd/systemd-udevd
136 /usr/local/bin/bd -i /etc/rtbrick/bd/config/lldpd.json
139 /usr/local/bin/bd -i /etc/rtbrick/bd/config/pim_iod.json
142 /usr/local/bin/bd -i /etc/rtbrick/bd/config/bgp_iod.json
145 /usr/local/bin/bd -i /etc/rtbrick/bd/config/isis_appd.json
147 /usr/local/bin/bd -i /etc/rtbrick/bd/config/isis_iod.json
149 /usr/local/bin/bd -i /etc/rtbrick/bd/config/etcd.json
152 /usr/local/bin/bd -i /etc/rtbrick/bd/config/resmond.json
154 /usr/local/bin/bd -i /etc/rtbrick/bd/config/staticd.json
<...output omitted...>
314 /usr/local/bin/alertmanager --config.file=/etc/prometheus/alertmanager.yml
--storage.path=/var/db/alertmanager
316 /usr/local/bin/prometheus --config.file=/etc/prometheus/prometheus.yml --storage.tsdb.retention.time=5d
--storage.tsdb.path=/var/db/prometheus
<...output omitted...>
supervisor@rtbrick:~$ pkill prometheus
```

## Disk Space

Another issue that can affect the functioning of the system is the lack of disk space; in severe situations, the system will become unusable. From this standpoint, checking disk space is one of the first things you do when doing first troubleshooting steps.

On Linux-based systems there are two main tools to check disk space: du (disk usage) and df (disk free). As in the case with ps and top, it is important to understand the uses cases for the two, and how they can complement each other.

Normally, you would first use df to have a quick look of the overall system disk space, then you would use du to look deeper into the problem. This approach is due to how these two tools work: df reads the superblocks only and trusts it completely, while du traverses a directory and reads each object, then sums the values

> up. This means that, most of the times, there will be differences between the exact values reported by these two; you can say that df sacrifices accuracy for speed.

First, we can look at the total space on the switch (we run the command in ONL):

```
supervisor@5916-nbg1:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        1.0M     0  1.0M   0% /dev
/dev/sdb7       113G  5.2G  102G   5% /
/dev/sdb6       2.0G  1.2G  677M  64% /mnt/onl/images
/dev/sdb1       256M  252K  256M   1% /boot/efi
/dev/sdb4       120M   43M   69M  39% /mnt/onl/boot
/dev/sdb5       120M  1.6M  110M   2% /mnt/onl/config
tmpfs           3.2G  720K  3.2G   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           6.3G     0  6.3G   0% /run/shm
cgroup           12K     0   12K   0% /sys/fs/cgroup
tmpfs           6.0G  546M  5.5G   9% /shm
supervisor@5916-nbg1:~$
```

We then verify container disk space, by looking at the general snapshot of the system:

```
supervisor@rtbrick:~$ df -h
Filesystem
Size  Used Avail Use% Mounted on
/var/cache/rtbrick/imagestores/847c6ecd-df58-462e-a447-38c620a12fe1/rbfs-cont/rbfs-accessleaf-qmx-20.10.0-
g4internal.20201103065150+Bmvpn.C1067d22e/rootfs  113G  5.1G  102G   5% /
none
492K     0  492K   0% /dev
/dev/sdb7
113G  5.1G  102G   5% /var/log
tmpfs
6.0G  546M  5.5G   9% /shm
devtmpfs
1.0M     0  1.0M   0% /dev/mem
tmpfs
16G  4.3M   16G   1% /dev/shm
tmpfs
16G  9.0M   16G   1% /run
tmpfs
5.0M     0  5.0M   0% /run/lock
tmpfs
16G     0   16G   0% /sys/fs/cgroup
tmpfs
3.2G     0  3.2G   0% /run/user/1000
supervisor@rtbrick:~$
```

At a quick glance we can see here that the root partition has a 5% usage, from a total of 113GB. You will also notice that /dev/sdb7 in the container has the same values as the output reported in ONL. It also has the same total size and same used space as the root filesystem. Notice the usage of the -h flag, which makes the output easier to read ("human readable").

Then you can verify the details of a specific directory, let's say you want too see how much disk space is used by user files in /usr:

```
supervisor@rtbrick:~$ ls -l /usr/
total 44
drwxr-xr-x  1 root root 4096 Nov  3 11:54 bin
drwxr-xr-x  2 root root 4096 Apr 24  2018 games
drwxr-xr-x 37 root root 4096 Nov  3 06:59 include
drwxr-xr-x  1 root root 4096 Nov  3 11:54 lib
drwxr-xr-x  1 root root 4096 Nov  3 06:57 local
drwxr-xr-x  2 root root 4096 Nov  3 06:59 sbin
drwxr-xr-x  1 root root 4096 Nov  3 11:54 share
drwxr-xr-x  2 root root 4096 Apr 24  2018 src
supervisor@rtbrick:~$ du -sh /usr/
2.6G    /usr/
supervisor@rtbrick:~$ ls -l /usr
total 44
drwxr-xr-x  1 root root 4096 Nov  3 11:54 bin
drwxr-xr-x  2 root root 4096 Apr 24  2018 games
drwxr-xr-x 37 root root 4096 Nov  3 06:59 include
drwxr-xr-x  1 root root 4096 Nov  3 11:54 lib
drwxr-xr-x  1 root root 4096 Nov  3 06:57 local
drwxr-xr-x  2 root root 4096 Nov  3 06:59 sbin
drwxr-xr-x  1 root root 4096 Nov  3 11:54 share
drwxr-xr-x  2 root root 4096 Apr 24  2018 src
```

We then go even deeper, to check what takes most space in the /usr directory

```
supervisor@rtbrick:~$ du -h /usr/ | sort -rh | head -5
2.6G    /usr/
1.8G    /usr/local
1.7G    /usr/local/lib
506M    /usr/lib
169M    /usr/share
```

We used du in conjunction with sort (options r - reverse the result -, and h - compare human readable numbers -), as well as with head, to get only the biggest 5 directories from the output.

# 3. Physical Layer

## Link Failures

Link failures are one of the most common issues. First, verify the status of the interface using one of the following show commands:

```
supervisor@rtbrick: op> show interface summary
supervisor@rtbrick: op> show interface <ifp-name>
```

The administrative, link, and operational status should be "Up". Depending on your configuration, logical interfaces should have been created and IP addresses should be assigned like in this example:

```
supervisor@rtbrick: op> show interface ifp-0/0/52
Interface               Admin    Link     Oper          IPv4 Address            IPv6 Address
ifp-0/0/52              Up       Up       Up
  ifl-0/0/52/13         Up       Up       Up            -
fe80::ba6a:97ff:fea5:923d/128
```

Verify further details of the interface using the 'detail' version of the show command:

```
supervisor@rtbrick: op> show interface ifp-0/0/52 detail
Interface:ifp-0/0/52
  Admin/Link/Operational status: Up/Up/Up
  Speed configured: 100G
  Speed maximum: 100G
  Duplex: Full
  Autonegotiation: Disabled
  Encapsulation mode: ieee
  MTU: 16360
  Maximum frame size: 16360
  Interface type: ethernet
  Interface index: 124929
  MAC: b8:6a:97:a5:92:3d
  Uptime:  Mon Nov 23 14:18:46 GMT +0000 2020
  Description: Physical interface #52 from node 0, chip 0
  Packet statistics:
    Rx packets: 263892      Tx packets: 280356
    Rx bytes: 23377027      Tx bytes: 154437883
  Interface: ifl-0/0/52/13, Instance: default
    Admin/Link/Operational status: Up/Up/Up
    IPv4/IPv6/MPLS Status: Up/Up/Up
    IPv4/IPv6/MPLS MTU: 1500/1500/1500
    Interface type: Logical Sub interface
    Interface index: 106497
    MAC: b8:6a:97:a5:92:3d
    IPv4 Address             IPv6 Address
    -                        fe80::ba6a:97ff:fea5:923d/128
```

```
    Packet statistics:
      Ingress forwarded packets: 262991            Ingress forwarded bytes: 23313212
      Ingress drop Packets: 0                      Ingress drop bytes: 0
      Egress forwarded packets: 15490              Egress forwarded bytes: 3063609
      Egress drop packets: 0                       Egress drop bytes: 0
```

Next, verify the interface statistics. These will show common link errors:

```
supervisor@rtbrick: op> show interface <ifp-name> statistics
```

You can also inspect the following BDS tables for even more detailed information:

```
supervisor@rtbrick: op> show datastore ifmd table global.interface.physical
supervisor@rtbrick: op> show datastore ifmd table global.interface.logical
supervisor@rtbrick: op> show datastore ifmd table global.interface.address
supervisor@rtbrick: op> show datastore fibd table local.bcm.qmx.port
```

Identify the type of optics and check the optics data:

```
supervisor@rtbrick: op> show optics inventory
supervisor@rtbrick: op> show optics interface <ifp-name>
```

The output of the 'show optics' command will display the Tx and Rx levels like in the following example. In addition, particularly check for warnings or alarms. These will typically indicate the cause of the issue. Please note the 'show optics' command does not work for passive Direct Attach Cable (DAC).

```
supervisor@rtbrick: op> show optics interface ifp-0/0/52
Physical Interface: ifp-0/0/52
  Lane Id                                              : 1
  Laser bias current                                   : 85.100 mA
  Laser tx power                                       : 1.1107 mW / 0.46 dBm
  Laser rx power                                       : 0.5521 mW / -2.58 dBm
  Module temperature                                   : 30.41 degree celsius
  Module voltage                                       : 3.199 V
  TX disable                                           : false
  High power class enable                              : true
  Laser TX loss of signal                              : false
  Laser TX loss of lock                                : false
  Laser RX loss of signal                              : false
  Laser RX loss of lock                                : false
  <...>
```

You can also show detailed optics information using the BDS tables:

```
supervisor@rtbrick: op> show datastore resmond table global.chassis_0.resource.optics.inventory
supervisor@rtbrick: op> show datastore resmond table global.chassis_0.resource.optics.module
```

If there are continuous or reoccurring interface issues like interface flapping, enable logging for the ifm module and inspect the log table as described in section 5.

## Hardware Failures

If you suspect a hardware failure or issue, verify sensor information available at the RFBS container layer. Please note this section applies to hardware switches only. Sensor information is not available on virtual deployments.

```
supervisor@rtbrick: op> show sensor system-led
supervisor@rtbrick: op> show sensor power-supply
supervisor@rtbrick: op> show sensor fan
supervisor@rtbrick: op> show sensor temperature
```

The sensor information might show hardware failures like in this example:

```
supervisor@rtbrick: op> show sensor power-supply
Name              Current In  Current Out Voltage In  Voltage Out Power In    Power Out   Status
PSU-1             0 mA        11593 mA    0 mV        11984 mV    0 mW        139000 mW   PRESENT
PSU-2             0 mA        0 mA        0 mV        0 mV        0 mW        0 mW        PRESENT, FAILED
```

Next, verify the status of the hardware at the ONL layer:

```
supervisor@spine1:~# sudo onlpdump
```

Due to a known issue, the 'sudo onlpdump' command does not work in the current release for the user supervisor. As a workaround switch to user root using 'sudo -i' and then enter the 'onlpdump' command.

The onlpdump tool provides detailed information about the system and its components, and might show hardware failures like in the following example:

```
supervisor@spine1:~# sudo onlpdump
   <...>
   psu @ 1 = {
       Description: PSU-1
       Model:  NULL
       SN:     NULL
       Status: 0x00000003 [ PRESENT,FAILED ]
       Caps:   0x00000000
       Vin:    0
       Vout:   0
       Iin:    0
```

```
        Iout:   0
        Pin:    0
        Pout:   0
    }
```

If you have identified or suspect an optics issue, verify the status of the optics at the ONL layer. The onlpdump -S tool will show the type of optic installed:

```
supervisor@spine1:~# sudo onlpdump -S
Port  Type            Media   Status  Len    Vendor            Model             S/N
----  --------------  ------  ------  -----  ----------------  ----------------  ----------------
   0  NONE
   <...>
   52  100GBASE-CR4    Copper          3m     Fiberstore        QSFP28-100G-DAC   I2706060007
   <...>
```

The following example shows an optic failure:

```
supervisor @localhost:~# sudo onlpdump -S
Port  Type            Media   Status  Len    Vendor            Model             S/N
----  --------------  ------  ------  -----  ----------------  ----------------  ----------------
<...>
06-27 08:27:49.823107 [x86_64_accton_as5916_54xk] Unable to read eeprom from port(51), size is different!
  51  Error E_INTERNAL
```

# 4. Application Layer

## Fabric Operation

### Fabric Protocols

Verify L2 connectivity using LLDP. LLDP allows to quickly verify the topology and connectivity between your devices:

```
supervisor@rtbrick>LEAF01: op> show lldp neighbor
Neighbor name      Status  Remote port ID     Local port ID      Neighbor MAC address  Last received     Last
sent
spine2             Up      memif-0/1/1        memif-0/1/1        7a:52:68:60:01:01     0:00:11 ago
0:00:12 ago
spine2             Up      memif-0/1/2        memif-0/1/2        7a:52:68:60:01:02     0:00:06 ago
0:00:09 ago
leaf1              Up      memif-0/1/1        memif-0/2/1        7a:47:fc:60:01:01     0:00:07 ago
0:00:10 ago
leaf2              Up      memif-0/1/1        memif-0/2/2        7a:28:3b:60:01:01     0:00:13 ago
0:00:14 ago
```

Verify IPv6 neighbor discovery:

```
supervisor@rtbrick>LEAF01: op> show neighbor ipv6
Instance                MAC Address          Interface             IP Address                Dynamic  Entry
Time
default                 7a:52:68:60:01:01    memif-0/1/1/1         fd3d:3d:100:a::2          true     Wed Nov
18 18:33:28
default                 7a:52:68:60:01:01    memif-0/1/1/1         fe80::7852:68ff:fe60:101  true     Wed Nov
18 18:32:30
default                 7a:52:68:60:01:02    memif-0/1/2/1         fe80::7852:68ff:fe60:102  true     Wed Nov
18 18:32:30
default                 7a:47:fc:60:01:01    memif-0/2/1/1         fe80::7847:fcff:fe60:101  true     Wed Nov
18 18:32:32
default                 7a:28:3b:60:01:01    memif-0/2/2/1         fe80::7828:3bff:fe60:101  true     Fri Nov
20 14:23:27
```

If there is no LLDP peer or no IPv6 neighbor discovered on an interface, it typically indicates a connectivity issue. BGPv6 peers cannot be established. At this point, proceed with the following steps:

- Verify the interface as described in section 3.1
- Verify connectivity to the neighbor using Ping as described in section 4.1.3
- Check the running configuration for the fabric interfaces.

If IPv6 neighbors have been discovered, verify the BGP sessions. BGP peers should have been auto-discovered on all fabric interfaces. If a BGP session is operational, it will be in "Established" state. The "PfxRcvd" and "PfxSent" show that BGP routes

are exchanged:

```
supervisor@rtbrick>LEAF01: op> show bgp peer
Instance name: default
Peer                                Remote AS    State         Up/Down Time              PfxRcvd
PfxSent
leaf1                               4200000201   Established   4d:17h:00m:27s            4
14
spine2                              4200000100   Established   0d:00h:05m:11s            8
14
```

If IPv6 neighbors have been discovered, but BGP sessions are not established, perform the following steps:

- Inspect the output of the 'show bgp peer detail' command
- Check the BGP running configuration
- Enable and verify BDS logging for bgp.iod and the BGP module as described in section 5

If BGP sessions are established and routes are being exchanged, BGP will typically be fully operational. Next, verify BGP routes for IPv6 unicast and IPv6 labeled unicast:

```
supervisor@rtbrick>LEAF01: op> show bgp rib-local ipv6 unicast
supervisor@rtbrick>LEAF01: op> @spine1: op> show bgp rib-local ipv6 labeled-
unicast
```

If you have multiple instances with a high number of routes, you can optionally filter the output using the 'instance default' command option. For both IPv6 and IPv6 LU, there should be one or multiple route(s) for each of the spine and leaf IPv6 loopback addresses like in this example:

```
supervisor@rtbrick>LEAF01: op> show bgp rib-local ipv6 unicast
Instance: default, AFI: ipv6, SAFI: unicast
Prefix                              Snd-Path-ID     Rcv-Path-ID     Peer                                Next-
Hop                           Up Time
fd3d:3d:0:99::1/128                 513421047       2               ::
0d:00h:00m:48s
fd3d:3d:0:99::2/128                 748525752       0               fe80::7852:68ff:fe60:101
fe80::7852:68ff:fe60:101            0d:00h:00m:36s
fd3d:3d:0:99::3/128                 30278035        0               fe80::7847:fcff:fe60:101
fe80::7847:fcff:fe60:101            0d:00h:00m:36s
fd3d:3d:0:99::4/128                 748525752       0               fe80::7852:68ff:fe60:101
fe80::7852:68ff:fe60:101            0d:00h:00m:36s
```

## Transport-layer Routing

The BGP routes described in section 4.1.1 above are subscribed by ribd. ribd will

the select the best routes from multiple sources and add them to the actual routing table. In this guide we refer to the connectivity between the fabric devices as transport-layer, as opposed to the service-laver connectivity in the VPNs which are deployed on top.

Verify the IPv6 unicast and IPv6 labeled unicast routing tables. Same like for the BGP commands, you can optionally filter the output using the 'instance default' command option:

```
supervisor@rtbrick>LEAF01: op> show route ipv6 unicast
supervisor@rtbrick>LEAF01: op> show route ipv6 labeled-unicast
```

For both IPv6 and IPv6 LU, there should be one or multiple route(s) for each of the spine and leaf IPv6 loopback addresses, each with a valid IPv6 nexthop address and exit interface. Assuming you are using BGPv6 as a fabric protocol, i.e. no additional protocols like IS-IS in the default instance, these will be BGP routes only. If all expected routes exist, it typically indicates that the fabric is working fine from a control-plane perspective.

## Fabric Connectivity

In order to troubleshoot date-plane issues, you can verify connectivity using the RBFS ping tool. First, verify connectivity to the auto-discovered link-local neighbors. Specify the interface on which the neighbor has been discovered as the source interface. Example:

```
supervisor@rtbrick>LEAF01: op> show neighbor ipv6
Instance                MAC Address          Interface             IP Address                Dynamic  Entry
Time
default                 7a:52:68:60:01:01    memif-0/1/1/1         fe80::7852:68ff:fe60:101  true     Wed Nov
18 18:32:30
<...>

supervisor@rtbrick>LEAF01: op> ping fe80::7852:68ff:fe60:101 source-interface memif-0/1/1/1
68 bytes from fe80::7852:68ff:fe60:101: icmp_seq=1 ttl=63 time=8.6318 ms
<...>
Statistics: 5 sent, 5 received, 0% packet loss
```

Second, verify connectivity to the spine and leaf loopback addresses learned via BGP. As a source address that is advertised via BGP in the default instance, so that it is reachable from the remote device. This depends on your deployment, but typically it is the loopback interface in the default instance. Example:

```
supervisor@rtbrick>LEAF01: op> show route ipv6 unicast
Instance: default, AFI: ipv6, SAFI: unicast
```

```
Prefix/Label                              Source            Pref    Next Hop
Interface
fd3d:3d:0:99::1/128                       direct            0       fd3d:3d:0:99::1
lo-0/0/0/1
fd3d:3d:0:99::3/128                       bgp               20
fe80::7847:fcff:fe60:101                  memif-0/2/1/1
fd3d:3d:0:99::4/128                       bgp               200
 <...>

supervisor@rtbrick>LEAF01: op> ping fd3d:3d:0:99::3 source-interface lo-0/0/0/1
68 bytes from fd3d:3d:0:99::3: icmp_seq=1 ttl=63 time=10.0001 ms
<...>
```

Next, verify MPLS connectivity by specifying IPv6 LU with the ping tool. Example:

```
supervisor@rtbrick>LEAF01: op> ping fd3d:3d:0:99::3 instance default afi ipv6 safi
labeled-unicast source-interface lo-0/0/0/1
68 bytes from fd3d:3d:0:99::3: icmp_seq=1 ttl=63 time=2.8520 ms
<...>
```

If the fabric connectivity is broken, use the RBFS traceroute tool to narrow down the location of the issue. Same as for ping, you need to use a source address that is advertised via BGP in the default instance and reachable from the remote device. Example:

```
supervisor@rtbrick>LEAF01: op> traceroute fd3d:3d:0:99::4 source-interface lo-0/0/0/1
traceroute to fd3d:3d:0:99::4 30 hops max, 60 byte packets
1    fd3d:3d:100:a::2    13.270 ms      4.973 ms      6.294 ms
2    fd3d:3d:0:99::4    18.825 ms      17.058 ms      17.764 ms
```

# Subscriber Services

The term subscriber describes an access user or session from a higher level decoupled from underlying protocols like PPPoE or IPoE. Subscribers in RBFS can be managed locally or remote via RADIUS. Each subscriber is uniquely identified by a 64bit number called subscriber-id.

## Subscriber Sessions

A good starting point for troubleshooting subscriber services is to verify the status of the subscriber sessions. If a session is fully operational, its state will be ESTABLISHED like in the following example:

```
supervisor@rtbrick>LEAF01: op> show subscriber
Subscriber-Id          Interface        VLAN      Type   State
```

```
72339069014638600      ifp-0/0/1        1:1       PPPoE  ESTABLISHED
72339069014638601      ifp-0/0/1        1:2       PPPoE  ESTABLISHED
72339069014638602      ifp-0/0/1        1:3       PPPoE  ESTABLISHED
72339069014638603      ifp-0/0/3        2000:7    L2TP   ESTABLISHED
```

Alternative use show subscriber detail which shows further details like username, Agent-Remote-Id (aka Line-Id) or Agent-Circuit-Id if screen width is large enough to print all those information. The following table describes all possible subscriber session states:

| State | Description |
|---|---|
| INIT | Initial subscriber state. |
| AUTHENTICATING | The subscriber is waiting for authentication response. |
| AUTH ACCEPTED | Authentication is accepted. |
| AUTH REJECTED | Authentication failed. |
| TUNNEL SETUP | Subscriber is tunnelled via L2TPv2 waiting for L2TP session setup completed. |
| ADDRESS ALLOCATED | IP addresses allocated. |
| ADDRESS REJECTED | IP addresses rejected (pool exhaust, duplicate or wrong addresses). |
| FULL | Subscriber forwarding state established. |
| ACCOUNTING | Subscriber accounting started sending RADIUS Accounting-Request-Start. |
| ESTABLISHED | The subscriber becomes ESTABLISHED after response to RADIUS Accounting-Request-Start if RADIUS accounting is enabled otherwise immediately after FULL. |
| TERMINATING | The subscriber is terminating and remains in this state until response to RADIUS Accounting-Request-Start if RADIUS accounting is enabled |

Further details per subscriber can be shown with the following commands.

```
supervisor@rtbrick>LEAF01: op> show subscriber 72339069014638600
  <cr>
  access-line           Subscriber access line information
  accounting            Subscriber accounting information
  acl                   Subscriber ACL information (filter)
  detail                Detailed subscriber information
```

```
  qos                   Subscriber QoS information
```

If a subscriber has been torn down or is not able to setup, inspect the the terminate history which indicates the teardown reason.

If a previously working subscriber session has been torn down, inspect the termination history which tells the actual reason.

```
supervisor@rtbrick>LEAF01: op> show subscriber history
Subscriber-Id          Timestamp                           Terminate Code
72339069014638594      Tue Nov 17 08:13:17 GMT +0000 2020   PPPoE LCP Terminate Request Received
72339069014638595      Tue Nov 17 08:13:17 GMT +0000 2020   PPPoE LCP Terminate Request Received
72339069014638596      Tue Nov 17 08:13:17 GMT +0000 2020   PPPoE LCP Terminate Request Received
72339069014638597      Tue Nov 17 08:13:17 GMT +0000 2020   PPPoE LCP Terminate Request Received
72339069014638598      Tue Nov 17 08:13:17 GMT +0000 2020   PPPoE LCP Terminate Request Received
72339069014638599      Tue Nov 17 08:13:46 GMT +0000 2020   L2TP CDN Request
72339069014638600      Tue Nov 17 08:39:01 GMT +0000 2020   PPPoE Clear Session
```

This command shows also further information like interface, VLAN and MAC address if screen is width enough.

Optionally you can view even more detailed information by inspecting the following key BDS tables used for subscriber management:

- Subscriber table – main table including all subscribers with all states and parameters:

```
supervisor@rtbrick>LEAF01: op> show datastore subscriberd.1 table
local.access.subscriber
```

- Subscriber interface table:

```
supervisor@rtbrick>LEAF01: op> show datastore subscriberd.1 table
global.access.1.subscriber.ifl
```

- Subscriber termination history table:

```
supervisor@rtbrick>LEAF01: op> show datastore subscriberd.1 table
local.access.subscriber.terminate.history
```

## RADIUS

Remote Authentication Dial-In User Service (RADIUS) is a networking protocol that provides centralized Authentication, Authorization and Accounting (AAA)

management for all types of subscribers (PPPoE or IPoE). RADIUS servers can perform as authentication and accounting servers or change of authorization (CoA) clients. Authentication servers maintain authentication records for subscribers.

The subscriber daemon requests authentication in RADIUS access-request messages before permitting subscribers access. Accounting servers handle accounting records for subscribers. The subscriber daemon transmits RADIUS accounting-start, interim and stop messages to the servers. Accounting is the process of tracking subscriber activity and network resource usage in a subscriber session. This includes the session time called time accounting and the number of packets and bytes transmitted during the session called volume accounting. A RADIUS server can behave as a change of authorization (CoA) client allowing dynamic changes for subscriber sessions. The subscriber daemon supports both RADIUS CoA messages and disconnect messages. CoA messages can modify the characteristics of existing subscriber sessions without loss of service, disconnect messages can terminate subscriber sessions.

RBFS supports multiple RADIUS servers for high availability and scaling which are bundled using RADIUS profiles. The status of those profiles can be shown with the following command.

```
supervisor@rtbrick>LEAF01: op> show radius profile
RADIUS Profile: radius-default
    NAS-Identifier: BNG
    NAS-Port-Type: Ethernet
    Authentication:
        Algorithm: ROUND-ROBIN
        Server:
            radius-server-1
            radius-server-2
    Accounting:
        State: UP
        Stop on Reject: True
        Stop on Failure: True
        Backup: True
        Algorithm: ROUND-ROBIN
        Server:
            radius-server-1
            radius-server-2
```

The profile accounting state becomes immediately ACTIVE if at least one of the referenced RADIUS accounting servers is enabled for accounting. Otherwise the profile keeps DISABLED which may indicates a wrong configuration.

If RADIUS Accounting-On is enabled, the profile state becomes STARTING before UP. It is not permitted to send any accounting request start, interim or stop related

to a profile in this state. It is also not permitted to send authentication requests if accounting-on-wait is configured in addition. The state becomes UP if at least one server in the accounting server list is in a state UP or higher.

A new profile added which references existing used RADIUS servers must not trigger a RADIUS Accounting-On request if at least one of the referenced servers is in a state of UP or higher.

The state of the RADIUS servers is shown with the following commands.

```
supervisor@rtbrick>LEAF01: op> show radius server
RADIUS Server            Address          Authentication State Accounting State
radius-server-1          100.0.0.1        UP                   UP
radius-server-2          100.0.0.3        ACTIVE               ACTIVE
radius-server-3          100.0.0.4        ACTIVE               ACTIVE
```

The following table explains the meaning of the different state where some of those state are applicable for accounting only.

| State | Description |
|---|---|
| DISABLED | RADIUS authentication (authentication_state) or accounting (accounting-state) is disabled or server not referenced by profile. |
| ACTIVE | Server referenced by RADIUS profile but no valid response received. |
| STARTING | This state is valid for accounting (accounting-state) only during accounting-on is sending (wait for accounting-on response). |
| STOPPING | This state is valid for accounting (accounting-state) only during accounting-off is sending (wait for accounting-off response). |
| FAILED | This state is valid for accounting (accounting-state) only if accounting-on/off timeout occurs. |
| UP | Valid RADIUS response received |
| UNREACHABLE | No response received/timeout but server is still usable. |
| DOWN | Server is down but can be selected. |
| TESTING | Send a request to test if server is back again. The server will not be selected for another request in this state (use a single request to check if server is back again). |
| DEAD | Server is down and should not be selected. |

Alternative use show radius server <radius-server> for further details and statistics per RADIUS server. Those statics can be cleared with clear radius server-statistics without any service impact.

## PPPoE Sessions

For PPPoE sessions the state should be ESTABLISHED if local terminated or TUNNELLED for L2TPv2 tunnelled sessions.

```
supervisor@rtbrick>LEAF01: op> show pppoe session
Subscriber-Id          Interface        VLAN      MAC               State
72339069014638604      ifp-0/0/1        1:1       00:04:0e:00:00:01 ESTABLISHED
72339069014638601      ifp-0/0/1        1:2       00:04:0e:00:00:02 ESTABLISHED
72339069014638602      ifp-0/0/1        1:3       00:04:0e:00:00:03 ESTABLISHED
72339069014638603      ifp-0/0/3        2000:7    52:54:00:57:c8:29 TUNNELLED
```

Alternative use show pppoe session detail which shows further details like username, Agent-Remote-Id (aka Line-Id) or Agent-Circuit-Id if screen width is large enough to print all those information.

| State | Description |
|---|---|
| LINKING | PPP LCP setup. |
| AUTHENTICATING | PPP authentication (PAP or CHAP). |
| NETWORKING | PPP IPCP (IPv4) and IP6CP (IPv6) setup. |
| ESTABLISHED | The PPPoE session becomes established if at least one NCP (IPCP or IP6CP) is established (state OPEN). |
| TUNNELLED | This state indicates that a PPPoE session is tunnelled via L2TPv2. |
| TERMINATING | PPP session teardown. |
| TERMINATED | PPPoE session terminated. |

If PPPoE session remain in state TERMINATED, the subscriber state should be checked. Typically this happens if RADIUS Accounting-Request-Stop is still pending.

Further details per PPPoE session can be shown with the following commands.

```
supervisor@rtbrick>LEAF01: op> show pppoe session 72339069014638601
  <cr>
  detail                Detailed session information
```

```
  statistics           Protocol statistics
```

The detail command shows the states of the session and all sub-protocols with extensive information and negotiated parameters.

Session statistics are available global and per session.

```
supervisor@rtbrick>LEAF01: op> show pppoe session statistics
supervisor@rtbrick>LEAF01: op> show pppoe session 72339069014638601 statistics
```

The PPPoE discovery statistics are helpful if session setup fails in initial PPPoE tunnel setup before actual PPP negotiation is starting.

```
supervisor@rtbrick>LEAF01: op> show pppoe discovery packets
Packet           Received         Sent
PADI             17               0
PADO             0                17
PADR             17               0
PADS             0                17
PADT             1                13

supervisor@rtbrick>LEAF01: op> show pppoe discovery errors
PADI Drop No Config             : 0
PADI Drop Session Protection    : 0
PADI Drop Session Limit         : 0
PADI Drop Dup Session           : 0
PADI Drop Interface Down        : 0
PADR Drop No Config             : 0
PADR Drop Wrong MAC             : 0
PADR Drop Interface Down        : 0
PADR Drop Session Limit         : 0
PADR Drop Session Protection    : 0
PADR Drop Bad Cookie            : 0
PADR Drop Bad Session           : 0
PADR Drop Dup Session           : 0
PADR Drop No mapping Id         : 0
PADT Drop No Session            : 0
PADT Drop Wrong MAC             : 0
PADX Interface Get Failure      : 0
```

If PPPoE session protection is enabled in access configuration profile, short lived or failed sessions will be logged in the PPPoE session protection table (local.pppoe.session.protection).

Every session not established for at least 60 seconds per default is considered as failed or short lived session. This will block new sessions on this IFP and VLANs for one second per default which increase exponential with any further failed session until the max time of per default 300 seconds is reached. The interval is reset after 900 seconds without failed sessions.

The PPPoE session protection table include also last subscriber-id and terminate code which indicates the reason for session failures.

```
supervisor@rtbrick>LEAF01: op> show pppoe discovery protection
Interface        VLAN      Status  Attempts   Last Terminate Code
ifp-0/0/1        1:1       OK      1          PPPoE LCP Terminate Request Received
ifp-0/0/1        1:2       OK      1          PPPoE LCP Terminate Request Received
ifp-0/0/1        1:3       OK      1          PPPoE LCP Terminate Request Received
```

If status OK indicates that new session are accepted where BLOCKED means that sessions will be rejected.

## L2TP Sessions

For L2TPv2 tunnelled PPPoE sessions the global unique subscriber-id can be used to get information about the L2TP session.

```
supervisor@rtbrick>LEAF01: op> show l2tp subscriber 72339069014638621
Subscriber-Id: 72339069014638621
    State: ESTABLISHED
    Local TID: 45880
    Local SID: 39503
    Peer TID: 1
    Peer SID: 1
    Call Serial Number: 10
    TX Speed: 10007000 bps
    RX Speed: 1007000 bps
    CSUN: disabled
```

The following command gives a good overview over the corresponding tunnels.

```
supervisor@rtbrick>LEAF01: op> show l2tp tunnel sessions
Role Local TID Peer TID State        Preference Sessions Established Peer Name
LAC       2022        1 ESTABLISHED       10000        1           1 LNS3
LAC       3274        1 ESTABLISHED       10000        1           1 LNS8
LAC      14690        1 ESTABLISHED       10000        1           1 LNS6
LAC      29489        1 ESTABLISHED       10000        1           1 LNS9
LAC      33323        1 ESTABLISHED       10000        1           1 LNS4
LAC      35657        1 ESTABLISHED       10000        1           1 LNS10
LAC      37975        1 ESTABLISHED       10000        1           1 LNS1
LAC      45880        1 ESTABLISHED       10000        1           1 LNS7
LAC      46559        1 ESTABLISHED       10000        1           1 LNS2
LAC      58154        1 ESTABLISHED       10000        1           1 LNS5
```

Detailed information per tunnel are available via show l2tp tunnel <TID> detail.

L2TP tunnel statistics are available global and per tunnel.

```
supervisor@rtbrick>LEAF01: op> show l2tp tunnel statistics
supervisor@rtbrick>LEAF01: op> show l2tp tunnel 37975 statistics
```

## Service-Layer Connectivity

A different type of issues can occur if a subscriber has successfully connected to a leaf switch, but does not have connectivity to his services, for example connecting to the Internet. The actual user traffic is carried in a VPN across the RBFS fabric. First, verify the VPN routing table on both the spine and the leaf switches. Depending on your design, there will be specific routes and/or a default route only:

```
supervisor@rtbrick>LEAF01: op> show route ipv4 unicast instance services
Instance: services, AFI: ipv4, SAFI: unicast
Prefix/Label                    Source      Pref    Next Hop            Interface
192.168.0.3/32                  direct      0       192.168.0.3         lo-0/0/0/2
192.168.0.4/32                  bgp         20      fd3d:3d:0:99::4     memif-0/1/1/1
 <...>
```

If routes are missing already on the spine switch, there might be a routing issue between the spine and the upstream core routers or route reflectors. Further troubleshooting steps will depend on how the fabric is connected to the upstream network in your deployment. If all expected routes exists on the spine switch, but are missing on the leaf switch, verify the VPN route exchange between them. Example for verifying VPN routes advertised by the spine switch:

```
supervisor@rtbrick>LEAF01: op> show bgp rib-out ipv4 vpn-unicast peer leaf1
Instance: default, AFI: ipv4, SAFI: vpn-unicast
  Peer: leaf1, Sent routes: 2
    Prefix                      MED     LocalPref   Origin          Next Hop                AS Path
    192.168.0.3/32              0       -           Incomplete      fd3d:3d:0:99::3         4200000100,
4200000201
    192.168.0.4/32              1       -           Incomplete      fd3d:3d:0:99::4         4200000100,
4200000202
<...>
```

Example for verifying VPN routes received by the leaf switch:

```
supervisor@rtbrick>LEAF01: op> show bgp rib-in ipv4 vpn-unicast peer spine1
Instance: default, AFI: ipv4, SAFI: vpn-unicast
  Peer: spine1, Received routes: 1
    Prefix                      Path ID    Next Hop                MED        LocalPref  AS Path
    192.168.0.4/32              0          fd3d:3d:0:99::4         1          -          4200000100,
4200000202
```

If you have a publically routed loopback address in the services VPN, you can verify the connectivity to any well-known destination address using the RBFS Ping tool within the VPN instance:

```
supervisor@rtbrick>LEAF01: op> ping 8.8.8.8 instance services source-interface lo-0/0/1/0
68 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=21.6622 ms
<...>
Statistics: 5 sent, 5 received, 0% packet loss
```

If there is no connectivity to the IP address of your service, verify connectivity across the fabric within the instance by sending a ping between two leaf switches. This will indicate if the connectivity problem lies in the spine/leaf fabric or in the upstream network:

```
supervisor@rtbrick>LEAF01: op> ping 192.168.21.5 instance services source-interface lo-0/0/1/0
68 bytes from 192.168.21.5: icmp_seq=1 ttl=63 time=1.5511 ms
<...>
Statistics: 5 sent, 5 received, 0% packet loss
```

# Host Path Capturing

You can use the RBFS built-in capture tool to verify and troubleshoot fabric as well as services protocol operation. It captures and displays all host-path traffic, that is control-plane packets sent to the CPU. It does not apply to transit traffic. This section explains the options available of the capturing tool to troubleshoot host path issues.

## Physical Interface

You can capture all host path packets on a physical interface, including all sub-interfaces, by specifying the physical interface (IFP) name with the capture command.

**capture interface** <physical-interface-name> **direction** <dir>

### Example

```
capture interface ifp-0/0/52 direction both
```

## Logical Interface

If you specify a logical interface (IFL) name with the capture command, the traffic on that sub-interface will be captured only. This allows to filter for example on a specific VLAN.

**capture interface** <logical-interface-name> **direction** <dir>

### Example

```
capture interface ifl-0/0/52/1 direction both
```

## Shared Memory Interface

There is no BDS packet table in fibd. Instead there is a pseudo network interface of the form shm-0/0/<trap-id>, where the trap ID identifies the protocol (BGP, ISIS, PPPoE, L2TP, RADIUS). You can use the VPP internal command **show rtb-shm** to find the mapping of protocol to trap ID. This command captures the packet exchanges between fibd and other protocol daemons.

**capture interface** <shm-interface-name> **direction** <dir>

### Example

```
capture interface shm-0/0/1 direction both
```

## ACL-based Packet Capturing

You can use an ACL to more granularly define the traffic to be captured.

**capture acl** <acl-name> **direction** <direction> **interface** <interface> <options>

| Option | Description |
|---|---|
| <acl-name> | ACL name. |
| <direction> | Direction of the packet. The supported values are: in, out, or both. |
| <interface> | Specifies the interface that is used to capture packets onto console. For ACL-based packet capturing, the interface is mandatory. |
| file <filename> start | You can use this option to save the packets in the PCAP file and later use a tool like Wireshark to analyse the captured traffic |
| raw | Raw packet capture |

### Example

```
{
  "rtbrick-config:acl": {
    "l3v6": {
      "rule": [
        {
          "rule-name": "from_to_spine1",
          "ordinal": [
            {
              "ordinal-value": 10,
              "match": {
                "direction": "ingress",
                "source-ipv6-prefix": "fd3d:3d:100:a::1/128"
              },
              "action": {
                "capture": "true"
              }
            },
            {
              "ordinal-value": 20,
              "match": {
                "destination-ipv6-prefix": "fd3d:3d:100:a::1/128",
                "direction": "ingress"
              },
              "action": {
                "capture": "true"
              }
            }
          ]
        }
      ]
    }
  }
}

supervisor@rtbrick>LEAF01: op> capture acl from_to_spine1 direction both interface
hostif-0/0/3
Success : ifp capture started

2022-06-09T07:45:48.358898+0000 7a:2f:78:c0:00:03 > 7a:3f:3e:c0:00:03, ethertype
IPv6 (0x86dd), length 122: (hlim 255, next-header ICMPv6 (58) payload length: 68)
fd3d:3d:100:a::1 > fd3d:3d:100:a::2: [icmp6 sum ok] ICMP6, echo request, seq 1

2022-06-09T07:45:48.359027+0000 7a:3f:3e:c0:00:03 > 7a:2f:78:c0:00:03, ethertype
IPv6 (0x86dd), length 122: (hlim 64, next-header ICMPv6 (58) payload length: 68)
fd3d:3d:100:a::2 > fd3d:3d:100:a::1: [icmp6 sum ok] ICMP6, echo reply, seq 1

<...>
```

## Filtering by Protocol

In most cases, while using the logical interface and physical interface, you may want to select a packet belonging to a specific protocol. In that case you can use the protocol filter option.

**capture interface** <interface-name> **direction** <direction> **protocol** <protocol-name>

### Example

```
supervisor@rtbrick>LEAF01: op> capture interface ifp-0/0/52 direction both
protocol bgp

supervisor@rtbrick>LEAF01: op> capture interface ifl-0/0/52/1 direction both
protocol bgp
```

## Raw Format

The raw option of the capture tool allows to decode as well as dump the packet in raw format. The **raw** option is useful if you want to examine packets in hex to check for malformed packets, etc.

**capture interface** <interface-name> **direction** <direction> **raw**

### Example

```
supervisor@rtbrick>LEAF01: op> capture interface ifl-0/0/52/1 direction both raw

supervisor@rtbrick>LEAF01: op> capture interface ifp-0/0/52 direction both raw
```

## PCAP File

While debugging a setup with real traffic, analysing all packets on a terminal might be cumbersome. You can use the **pcap** option to save the packets in the PCAP file and later use a tool like Wireshark to analyse the captured traffic.

To start capturing the traffic in a file, enter the following command:

**capture interface** <interface-name> **direction** <direction> **file** <file_name.pcap> **start**

To stop capturing the traffic in a file, enter the following command:

**capture interface** <interface-name> **direction** <direction> **file** <file_name.pcap> **stop**

## Example

```
supervisor@rtbrick>LEAF01: op> capture interface ifp-0/0/52 direction both file
test.pcap start

supervisor@rtbrick>LEAF01: op> capture interface ifp-0/0/52 direction both file
test.pcap stop
```

# 5. Logging

## Log Files

### Log Files on the Host OS

On the host OS, i.e. ONL in case of hardware switches, navigate to the /var/log/ folder and list the available log files:

```
supervisor@spine1:/var/log$ ls -l
total 3064
<...>
-rw-r--r-- 1 root root    2242 Nov  4 11:13 rtbrick-apigwd.log
-rw-r--r-- 1 root root  458617 Nov  4 14:27 rtbrick-ctrld.log
-rw-r--r-- 1 root root     831 Nov  4 11:13 rtbrick-hostconfd.log
-rw-r--r-- 1 root root   73509 Nov  4 14:27 rtbrick-hostnetconfd.log
-rw-r--r-- 1 root root     668 Nov  4 11:13 rtbrick-lxcd.log
-rw-r----- 1 root adm   286356 Nov  4 14:27 syslog
<...>
```

Inspect the log file of interest using one of the standard Linux tools like cat, more, or less. For example:

```
supervisor@spine1:/var/log$ more rtbrick-ctrld.log
```

### Log Files in the LXC

In the RBFS container, navigate to the /var/log/ folder and list the available log files. There will be one active log file per daemon or service, and up to ten compressed (.gz) log files created during log file rotation:

```
supervisor@spine1:/var/log$ ls -l
total 2376
<...>
-rw-r--r-- 1 root   root  48268 Nov  4 15:51 rtbrick-alertmanager-service-out.log
-rw-r--r-- 1 root   root   3451 Oct 30 17:51 rtbrick-alertmanager-service-out.log.1.gz
-rw-r--r-- 1 root   root   2509 Oct 25 00:02 rtbrick-alertmanager-service-out.log.2.gz
-rw-r--r-- 1 root   root      0 Nov  1 01:17 rtbrick-bgp.appd.1-service-out.log
-rw-r--r-- 1 root   root   3333 Oct 30 17:51 rtbrick-bgp.appd.1-service-out.log.1.gz
-rw-r--r-- 1 root   root   3192 Oct 19 13:17 rtbrick-bgp.appd.1-service-out.log.2.gz
<...>
```

Please note log files for brick daemons include only basic logs referring to the operation of the daemon itself. The actual applications like routing or service

protocols use log tables described in section 5.2. In contrast, Prometheus and Alertmanager are no BDs and use log files as the primary means of logging. Inspect the log file of interest using one of the standard Linux tools like cat, more, or less. For example:

```
supervisor@spine1:/var/log$ more rtbrick-prometheus-service-out.log
```

# BDS Logging

## Enabling BDS Logging

BDS logging is enabled per BD, per module, and per group by configuration. By default, BDS logging is enabled for all daemons and all modules with level Error. Verify the log status of the module and optionally per daemon. The following output reflects a default configuration:

```
supervisor@spine2: op> show log status module bgp bd bgp.iod.1
Module log status:
  bgp:
    bgp.iod.1:
      Level: error, Plugin: None
      Log group status:
        Group                   Level          Plugin                   Rate limit
        config                  error          None                     10
        general                 error          None                     10
        generic                 error          None                     10
        <...>
```

For troubleshooting an application, enable a more detailed log level. First example, enable and verify logging with level Info for the bgp.iod and all groups of the BGP module:

```
supervisor@spine2: cfg> set log bd bgp.iod.1 module bgp level info
supervisor@spine2: cfg> commit

supervisor@spine2: op> show log bgp.iod.1 status
<...>
supervisor@spine2: op> show log status module bgp bd bgp.iod.1
Module log status:
  bgp:
    bgp.iod.1:
      Level: info, Plugin: None
      Log group status:
        Group                   Level          Plugin                   Rate limit
        config                  info           None                     10
        general                 info           None                     10
        generic                 info           None                     10
```

```
        <...>
```

Second example, enable and verify logging with level Debug specifically for the PPP and PPPoE group of logs:

```
supervisor@leaf2: cfg> set log bd pppoed.1 module pppoed group ppp level debug
supervisor@leaf2: cfg> set log bd pppoed.1 module pppoed group pppoe level debug
supervisor@leaf2: cfg> commit
supervisor@leaf2: cfg>

supervisor@leaf2: op> show log status module pppoe
Module log status:
  pppoe:
    pppoed.1:
      Level: error, Plugin: None
      Log group status:
        Group                   Level          Plugin                  Rate limit
        config                  error          None                    10
        dhcpv6                  error          None                    10
        general                 error          None                    10
        generic                 error          None                    10
        ppp                     debug          None                    10
        pppoe                   debug          None                    10
        subscriber              error          None                    10
```

For a complete and detailed description of the logging configuration, please refer to the RBFS Logging Guide.

## Inspecting Log Tables

BDS log tables are created on demand by the daemons if logging is enabled and there is an event to log. Use the 'show log table' command to look into the log tables:

```
supervisor@leaf2: op> show log table bgp.logs
[   Info   ] <2021-07-16T06:57:14.434358+0000> BGP peer fe80::7825:1dff:fe60:202, source
fe80::7828:3bff:fe60:101, hostname spine1, instance default reset, reason Notification received
[   Info   ] <2021-07-16T06:57:14.435583+0000> BGP FSM change, peer fe80::7825:1dff:fe60:202, source
fe80::7828:3bff:fe60:101, hostname -, instance default changed state from Established to Idle, reason Cease,
Sub-Code: Admin reset
[   Info   ] <2021-07-16T06:58:01.033211+0000> Created table(default.bgp.rib-
in.ipv6.unicast.fe80::7825:1dff:fe60:202.fe80::7828:3bff:fe60:101)
[   Info   ] <2021-07-16T06:58:01.033290+0000> Created table(default.bgp.rib-in.ipv6.labeled-
unicast.fe80::7825:1dff:fe60:202.fe80::7828:3bff:fe60:101)
[   Info   ] <2021-07-16T06:58:01.033347+0000> Created table(default.bgp.rib-in.ipv4.vpn-
unicast.fe80::7825:1dff:fe60:202.fe80::7828:3bff:fe60:101)
[   Info   ] <2021-07-16T06:58:01.033401+0000> Created table(default.bgp.rib-in.ipv6.vpn-
unicast.fe80::7825:1dff:fe60:202.fe80::7828:3bff:fe60:101)
[   Info   ] <2021-07-16T06:58:01.344927+0000> BGP FSM change, peer fe80::7825:1dff:fe60:202, source
fe80::7828:3bff:fe60:101, hostname spine1, instance default changed state from Idle to Established, reason
None
[   Info   ] <2021-07-16T06:58:03.471937+0000> BGP message received from peer fe80::7825:1dff:fe60:202,
source fe80::7828:3bff:fe60:101, hostname spine1, instance default decode failed. Message type 2
<...>
```

The 'show log table' command supports various filter options. You can filter on the level, module, or a regular expression. Example:

```
supervisor@spine2: op> show log table bgp.logs filter level error
supervisor@spine2: op>
```

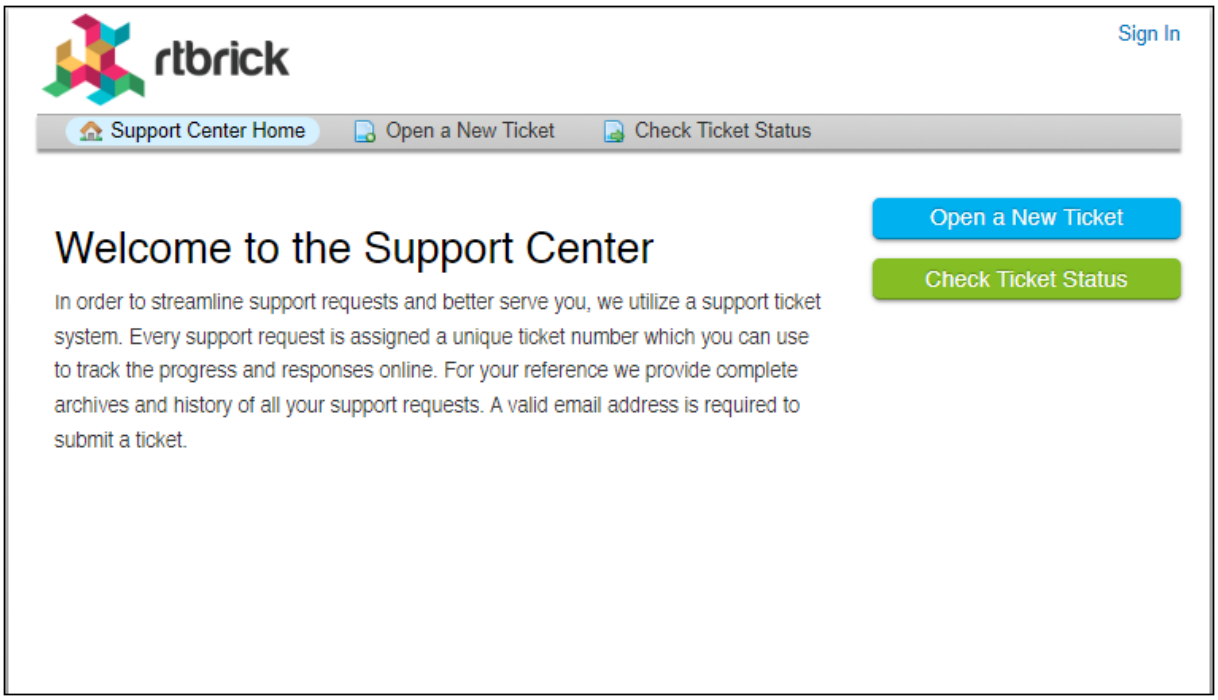
For a complete and detailed description of the filter options, please refer to the RBFS Logging Guide.

# 6. Reporting Issues to RtBrick

If all the above steps do not provide sufficient information to help you solve the issue, or if you suspect, based on the data gathered, that the system is not working as it should ( e.g. a software bug), you can contact RtBrick TAC. NOTE: Please note that contacting RtBrick is done via a designated contact, either in your organization or in your team.

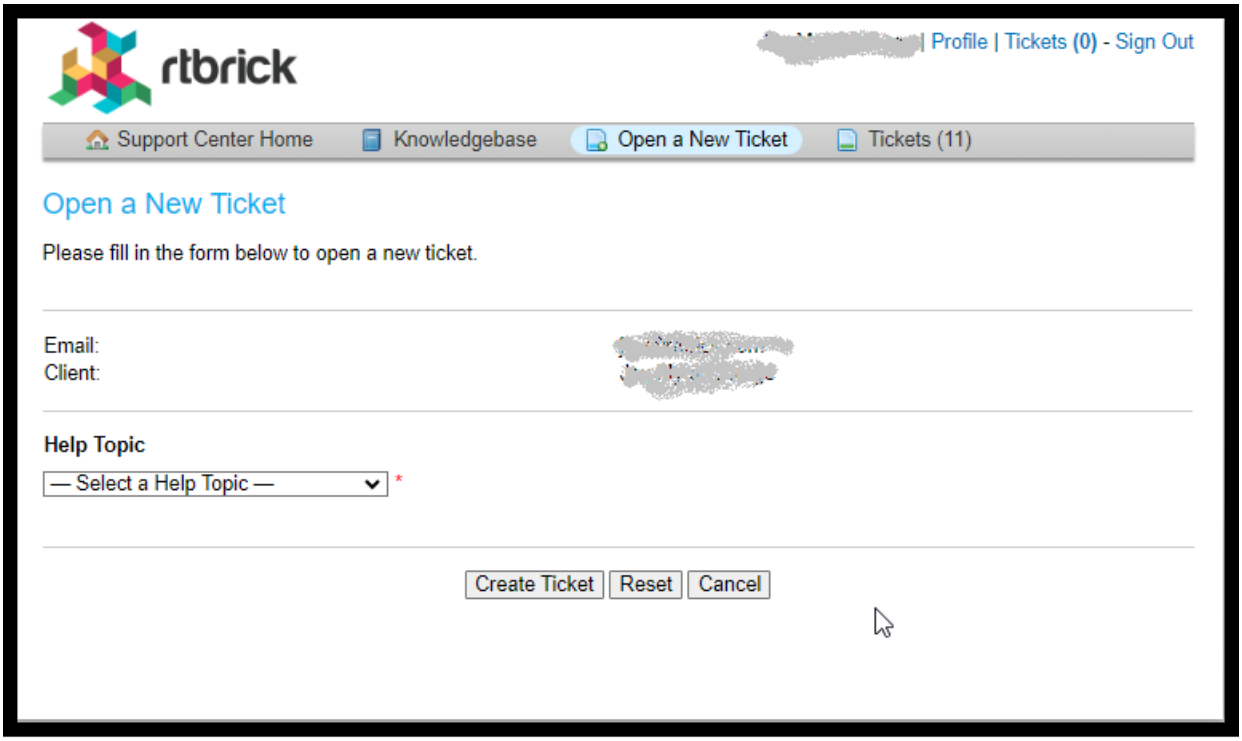
Accessing RtBrick support is done via the RtBrick webpage, in the support section: https://www.rtbrick.com/support/contact-our-experts-now. To be able to open a support case, you need an active business contract, and a support account at RtBrick.

## Support Portal login page




On the https://www.rtbrick.com/support/contact-our-experts-now page, click the *customer support portal* link; you will be redirected to the Customer Portal. In the support page that appears, if you don't have any cases already opened, you will be presented with a form to open a case; on the righthand side you will have quick links to documentation and products. Note that fields marked with an asterisk are mandatory. After filling in all the information, press the submit button on the bottom of the page.

# Opening a case




To help solving the case in a timely manner, basic information must be given in order to pinpoint the issue:

- If the issue was encountered on a virtual installation or on a physical box
- The RBFS image role (spine/access leaf), the version, and the patch version
- Any logs and/or comments: for logs you can use the "Add attachment" link in the support page, and comments can be posted after the case is submitted.

| Registered Address | Support | Sales |
|---|---|---|
| 40268, Dolerita Avenue<br>Fremont CA 94539 | | |
| +1-650-351-2251 | | +91 80 4850 5445 |
| http://www.rtbrick.com | support@rtbrick.com | sales@rtbrick.com |

# Debugger Information Utility Program

In RBFS, every daemon maintains a set of tables containing data specific to its functions. These daemon-specific tables help in a deeper analysis of issues, allowing for targeted debugging based on the specific functions and responsibilities of each daemon.

RBFS implements a debugger information utility program. This utility allows you to retrieve table information from all or a few specific modules. This information also includes log files, diagnostic messages, CPU and memory states, and current and previous configurations. This utility program can be run whenever an issue is encountered with RBFS and needs to be reported to customer support.

The utility program collects data and converts this into a JSON file format. Each collected file is named after the respective table name or data it contains and is stored in a directory named with the corresponding timestamp. This directory is compressed into a zip file and saved in the "/var/crash/debug_info" directory.

You can then attach and send this zipped file to the RtBrick support team. This information is valuable for troubleshooting many types of network issues, as it allows the support team to analyze the router's state and configurations without directly accessing the router.

When users, who do not have the necessary permissions, attempt to access data from a table by executing the command, they will not be able to view the retrieved data.

## How to Retrieve Data Using the Utility Program

To retrieve data, you must run the collect debug-information command in the debug mode to retrieve the data.

The following is the command syntax:

**collect debug-information** <option>

| Attribute | Description |
|---|---|
| all | This option retrieves data from all daemons. |

| Attribute | Description |
|---|---|
| <module-name> | This option allows you to specify the daemon/module name so that the utility retrieves data only for that specific module/daemon. |

To collect debug information for all modules, use the following command:

collect debug-information all

After executing the command, the data collection process takes a few seconds or minutes to complete.

The following example command execution shows the date and time in sequential order as each daemon's data is fetched. It also shows the total time taken to complete the collection process at the end

```
supervisor@rtbrick.net:/var/crash/debug_info $ cat debug_info.log
2024-09-09 09:46:09,099 - Executed Command collect debug-information all internal
2024-09-09 09:46:36,492 - Fetching contents for Module Name: bds-infra BDS tables
2024-09-09 09:46:38,541 - Fetching contents for Module Name: device-overview
2024-09-09 09:46:38,541 - Fetching contents for Module Name: ifmd
2024-09-09 09:46:38,541 - Fetching contents for Module Name: staticd
2024-09-09 09:46:51,247 - Fetching contents for Module Name: fibd
2024-09-09 09:46:51,247 - Fetching contents for Module Name: platform-dependent
2024-09-09 09:46:51,268 - Fetching contents for Module Name: mtu
2024-09-09 09:46:51,268 - Fetching contents for Module Name: ribd
2024-09-09 09:46:51,268 - Fetching contents for Module Name: pimd
2024-09-09 09:46:51,268 - Fetching contents for Module Name: mribd
2024-09-09 09:47:20,041 - Output directory: /var/crash/debug_info/manual/
debug_info_Sep-9-09-46-09-AM.tar.gz
2024-09-09 09:47:20,041 - Total Time Taken for executing collect debug-information all
internal command is : 70.944 seconds
```

## Periodic Data Collection

This utility program can also help in collecting system states and other information at scheduled intervals. This periodically collected information helps in understanding the system state before an issue occurs, which is useful in many scenarios.

The utility can be configured to automatically run at predefined intervals to collect data. After each run, it saves reports that can be used for further analysis and troubleshooting. You can define the intervals for periodic data collection to suit your needs.

The system employs Cron Job functionality to collect this periodic debugging information. A Cron Job is a scheduled task that runs automatically at specified intervals.

You can configure the Cron Job functionality to enable periodic data collection.

## Setting Up Cron Job

You can specify the data collection interval, limit the number of files, and initiate the Cron job function.

Specify the Interval:

Define how frequently the system should collect debug information. The interval is specified in hours. The default interval for running the utility is 12 hours. Use the command:

set debug-information collector interval <interval>

Specify the File Limit:

Specify the maximum number of files to be stored for such runs. Once the limit is reached, the system will start overwriting the oldest files and this process continues. The default maximum number of files is 10. Use the command:

set debug-information collector max-files <max_files>

### Stopping the Cron Job Function

You can use the following delete form of the command to halt the ongoing Cron Job process.

delete debug-information collector