DOCA Services

Table of contents

This is an overview of the set of services provided by DOCA and their purpose.

Introduction

DOCA services are DOCA-based products, wrapped in a container for fast and easy deployment on top of the NVIDIA® BlueField® DPU. DOCA services leverage DPU capabilities to offer telemetry, time synchronization, networking solutions, and more.

Services containers can be found under the official NGC catalog, labeled under the "DOCA" and "DPU" NGC labels, as well as the built-in NVIDIA platform option ("DOCA") on the container catalog.

For information on the deployment of the services, refer to the NVIDIA BlueField DPU Container Deployment Guide.

Development Lifecycle

DOCA-based containers consist of two main categories:

- DOCA Base Images – containerized DOCA environments for both runtime and development. Used either by developers for their development environment or in the process of containerizing a DOCA-based solution.
- DOCA Services – containerized DOCA-based products

The process of developing and containerizing a DOCA-based product is described in the following sections.

Development

Before containerizing a product, users must first design and develop it using the same process for a bare-metal deployment on the BlueField DPU.

This process consists of the steps:

1. Identifying the requirements for the DOCA-based solution.
2. Reviewing the feature set offered by the DOCA SDK libraries, as shown in detail in their respective programming guides.
3. Starting the development process by following our Developer Guide to make the best use of our provided tips and tools.

4. Testing the developed solution.

Once the developed product is mature enough, it is time to start containerizing it.

Containerization

In this process, it is recommended to make use of DOCA's provided base-images, as available on DOCA's NGC page.

Three image flavors are provided:

- `base-rt` – includes the DOCA runtime, using the most basic runtime environment required by DOCA's SDK
- `full-rt` – builds on the previous image and includes the full list of runtime packages, which are all user-mode components that can be found under the doca-runtime package
- `devel` – builds on the previous image and adds headers and development tools for developing and debugging DOCA applications. This image is particularly useful for multi-stage builds.

All images are preconfigured to use to the DOCA repository of the matching DOCA version. This means that installing an additional DOCA package as part of a Dockerfile / within the development container can be done using the following commands:

```
apt update
apt install <package name>
```

For DOCA and CUDA environments, there are similar flavors for these images combined with CUDA's images:

- `base-rt` (DOCA) + `base` (CUDA)
- `full-rt` (DOCA) + `runtime` (CUDA)
- `devel` (DOCA) + `devel` (CUDA)

Once the containerized solution is mature enough, users may start profiling it in preparation for a production-grade deployment.

Profiling

As mentioned in the NVIDIA BlueField DPU Container Deployment Guide, the current deployment model of containers on top of the DPU is based on kubelet-standalone. And more specifically, this Kubernetes-based deployment makes use of YAML files to describe the resources required by the pod such as:

- CPU
- RAM
- Huge pages

It is recommended to profile your product so as to estimate the resources it requires (under regular deployments, as well as under stress testing) so that the YAML would contain an accurate "resources" section. This allows an administrator to better understand what the requirements are for deploying your service, as well as allow the k8s infrastructure to ensure that the service is not misbehaving once deployed.

Once done, the containerized DOCA-based product is ready for the final testing rounds, after which it will be ready for deployment in production environments.

Services

Container Deployment

This page provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField® DPU.

DOCA BlueMan

DOCA BlueMan service runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface. This friendly, easy-to-use web dashboard acts as a one-stop shop for all the information needed to monitor the DPU.

DOCA Firefly

DOCA Firefly service provides precision time protocol (PTP) based time syncing services to the BlueField DPU . PTP is used to synchronize clocks in a network which, when used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, which is far better than what is normally obtainable with network time protocol (NTP).

DOCA Flow Inspector

DOCA Flow Inspector service allows monitoring real-time data and extraction of telemetry components which can be utilized by various services for security, big data and more.

Specific mirrored packets can be transferred to Flow Inspector for parsing and analyzing. These packets are forwarded to DTS, which gathers predefined statistics determined by various telemetry providers.

DOCA HBN

DOCA Host-based Networking service orchestrates network connectivity of dynamically created VMs/containers on cloud servers. HBN service is a BGP router that supports E-VPN extension to enable multi-tenant cloud.

At its core, HBN is the Linux networking acceleration driver of the DPU, Netlink to a DOCA daemon which seamlessly accelerates Linux networking using hardware programming APIs.

DOCA Telemetry

DOCA Telemetry service (DTS) collects data from built-in providers and from external telemetry applications. Collected data is stored in binary format locally on the DPU and can be propagated onwards using Prometheus endpoint pulling, pushing to Fluent Bit, or using other supported providers. Exporting NetFlow packets collected using the DOCA Telemetry NetFlow API is a great example of DTS usage.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

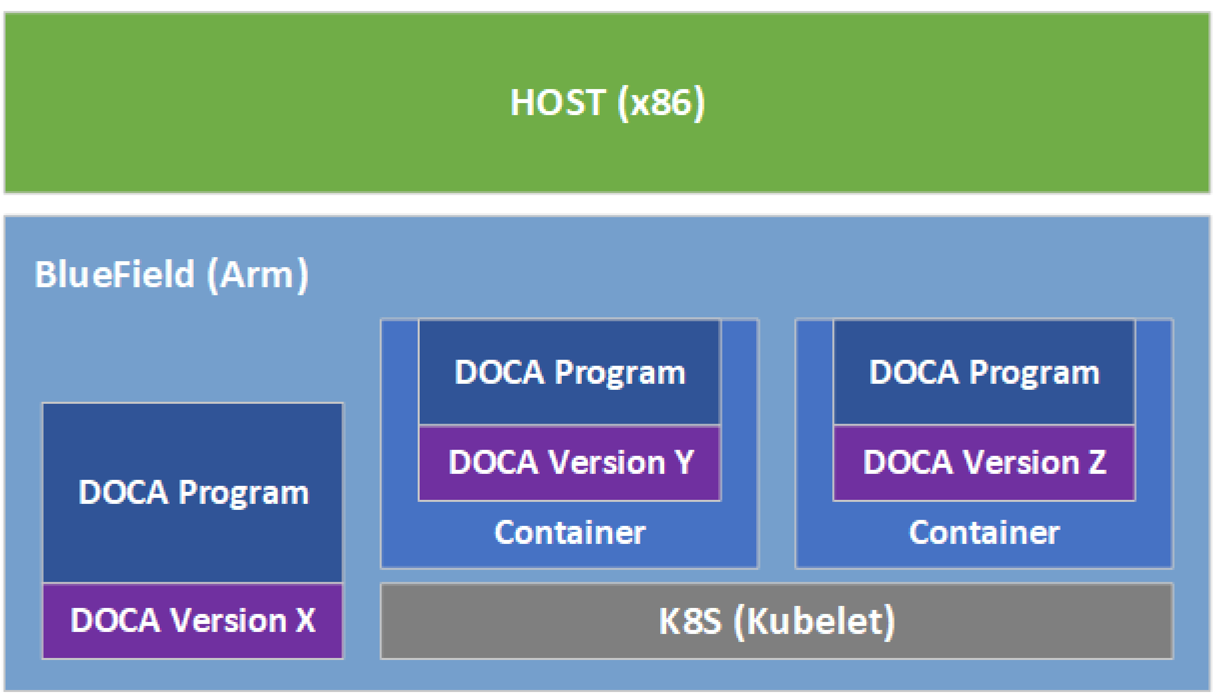
NVIDIA BlueField DPU Container Deployment Guide

This guide provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField® DPU.

Introduction

DOCA containers allow for easy deployment of ready-made DOCA environments to the DPU, whether it is a DOCA service bundled inside a container and ready to be deployed, or a development environment already containing the desired DOCA version.

Containerized environments enable the users to decouple DOCA programs from the underlying BlueField software. Each container is pre-built with all needed libraries and configurations to match the specific DOCA version of the program at hand. One only needs to pick the desired version of the service and pull the ready-made container of that version from NVIDIA's container catalog.



The different DOCA containers are listed on NGC, NVIDIA's container catalog, and can be found under both the "DOCA" and "DPU" labels.

Prerequisites

- Refer to the NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField related software
- BlueField image version required is 3.9.0 and higher

> **Note**
>
> Container deployment based on **standalone** Kubelet, as presented in this guide, is currently in **alpha version** and is subject to change in future releases.

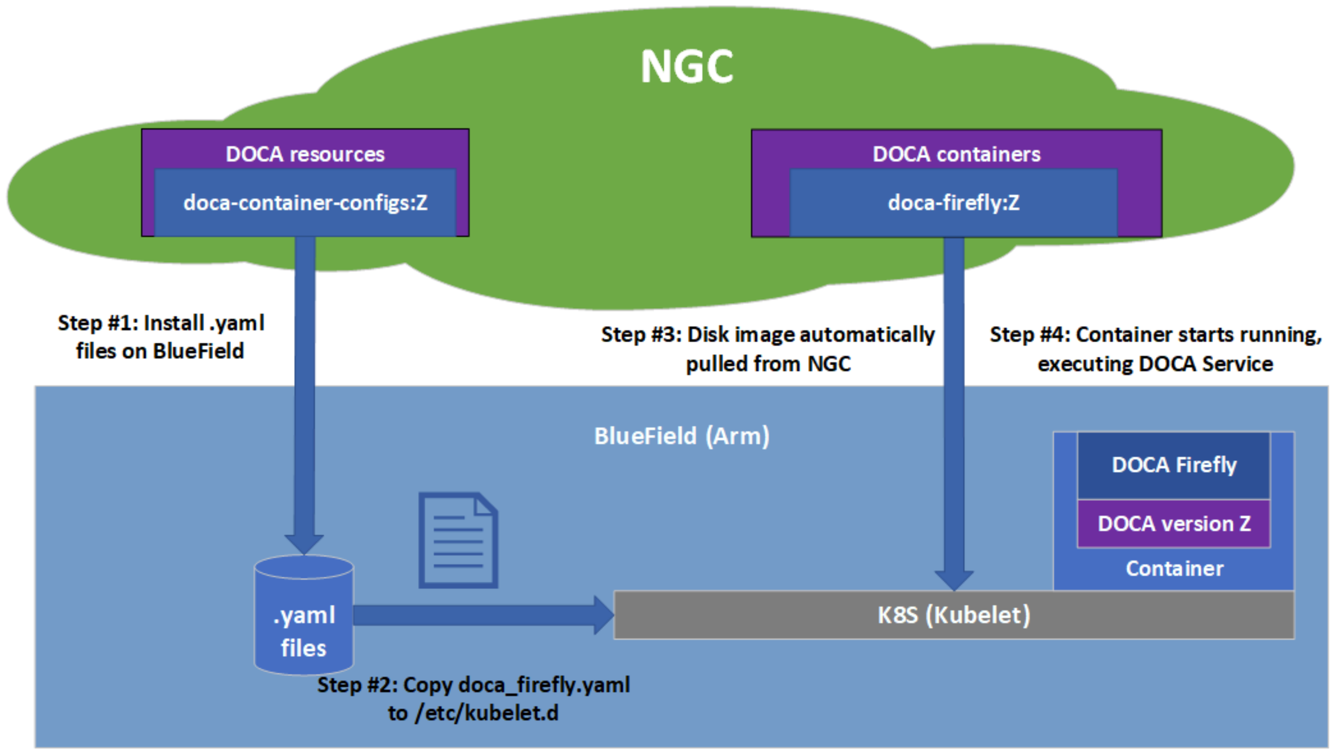
Container Deployment

Deploying containers on top of the BlueField DPU requires the following setup sequence:

1. Pull the container `.yaml` configuration files.
2. Modify the container's `.yaml` configuration file.
3. Deploy the container. The image is automatically pulled from NGC.

Some of the steps only need to be performed once, while others are required before the deployment of each container.

What follows is an example of the overall setup sequence using the DOCA application recognition (AR) container as an example.



Pull Container YAML Configurations

> **Note**
>
> This step pulls the `.yaml` configurations from NGC. If you have already performed this step for other DOCA containers you may skip to the next section.

Pulling the latest resource version can be done using the following command:

```
# Pull the entire resource as a *.zip file
wget --content-disposition
https://api.ngc.nvidia.com/v2/resources/nvidia/doca/doca_container_
-O doca_container_configs_2.5.0v1.zip
# Unzip the resource
```

```
unzip -o doca_container_configs_2.5.0v1.zip -d
doca_container_configs_2.5.0v1
```

More information about additional versions can be found in the NGC resource page.

Container-specific Instructions

Some containers require specific configuration steps for the resources used by the application running inside the container and modifications for the `.yaml` configuration file of the container itself.

Refer to the container-specific instructions listed under the container's relevant page on NGC.

Structure of NGC Resource

The DOCA NGC resource downloaded in section "Pull Container YAML Configurations" contains a `configs` directory under which a dedicated folder per DOCA version is located. For example, `2.0.2` will include all currently available `.yaml` configuration files for DOCA 2.0.2 containers.

```
doca_container_configs_2.0.2v1
    configs
        1.2.0
        ...
        2.0.2
            doca_application_recognition.yaml
            doca_blueman.yaml
            doca_devel.yaml
            doca_devel_cuda.yaml
            doca_firefly.yaml
            doca_flow_inspector.yaml
            doca_hbn.yaml
```

```
              doca_ips.yaml
              doca_snap.yaml
              doca_telemetry.yaml
              doca_url_filter.yaml
```

In addition, the resource also contains a `scripts` directory under which services may choose to provide additional helper-scripts and configuration files to use with their services.

The folder structure of the `scripts` directory is as follows:

```
+ doca_container_configs_2.0.2v1
+-+ configs
| +--  ...
+-+ scripts
  +-+ doca_firefly                <== Name of DOCA Service
  +-+ doca_hbn                    <== Name of DOCA Service
  | +-+ 1.3.0
  | | +-- ...                     <== Files for the DOCA HBN
version "1.3.0"
  | +-+ 1.4.0
  | | +-- ...                     <== Files for the DOCA HBN
version "1.4.0"
```

A user wishing to deploy an older version of the DOCA service would still have access to the suitable YAML file (per DOCA release under `configs`) and scripts (under the service-specific version folder which resides under `scripts`).

Spawn Container

Once the desired `.yaml` file is updated, simply copy the configuration file to Kubelet's input folder. Here is an example using the `doca_firefly.yaml`, corresponding to the DOCA Firefly service.

```
cp doca_firefly.yaml /etc/kubelet.d
```

Kubelet automatically pulls the container image from NGC and spawns a pod executing the container. In this example, the DOCA Firelfy service starts executing right away and its printouts would be seen via the container's logs.

Review Container Deployment

When deploying a new container, it is recommended to follow this procedure to ensure successful completion of each step in the deployment:

1. View currently active pods and their IDs:

```
sudo crictl pods
```

> **Info**
>
> It may take up to 20 seconds for the pod to start.

When deploying a new container, search for a matching line in the command's output:

```
POD ID              CREATED             STATE
NAME                                     NAMESPACE
ATTEMPT             RUNTIME
06bd84c07537e       4 seconds ago       Ready
doca-firefly-my-dpu                      default             0
(default)
```

2. If a matching line fails to appear, it is recommended to view Kubelet's logs to get more information about the error:

```
sudo journalctl -u kubelet --since -5m
```

Once the issue is resolved, proceed to the next steps.

> **Info**
>
> For more troubleshooting information and tips, refer to the matching section in our Troubleshooting Guide.

3. Verify that the container image is successfully downloaded from NGC into the DPU's container registry (download time may vary based on the size of the container image):

```
sudo crictl images
```

Example output:

```
IMAGE                                  TAG                 IMAGE
ID              SIZE
k8s.gcr.io/pause                       3.2
2a060e2e7101d        251kB
nvcr.io/nvidia/doca/doca_firefly       1.1.0-doca2.0.2
134cb22f34611        87.4MB
```

4. View currently active containers and their IDs:

```
sudo crictl ps
```

Once again, find a matching line for the deployed container (boot time may vary depending on the container's image size):

```
CONTAINER           IMAGE               CREATED
STATE               NAME                     ATTEMPT
POD ID              POD
b505a05b7dc23       134cb22f34611       4 minutes ago
Running             doca-firefly             0
06bd84c07537e       doca-firefly-my-dpu
```

5. In case of failure, to see a line matching the container, check the list of all recent container deployments:

```
sudo crictl ps -a
```

It is possible that the container encountered an error during boot and exited right away:

```
CONTAINER           IMAGE               CREATED
STATE               NAME                     ATTEMPT
POD ID              POD
de2361ec15b61       134cb22f34611       1 second ago
Exited              doca-firefly             1
4aea5f5adc91d       doca-firefly-my-dpu
```

6. During the container's lifetime, and for a short timespan after it exits, once can view the containers logs as were printed to the standard output:

```
sudo crictl logs <container-id>
```

In this case, the user can learn from the log that the wrong configuration was passed to the container:

```
$ sudo crictl logs de2361ec15b61
Starting DOCA Firefly - Version 1.1.0
...
Requested the following PTP interface: p10
Failed to find interface "p10". Aborting
```

> **Info**
>
> For additional information and guides on using `crictl`, refer to the Kubernetes documentation.

Stop Container

The recommended way to stop a pod and its containers is as follows:

1. Delete the `.yaml` configuration file for Kubelet to stop the pod:

```
rm /etc/kubelet.d/<file name>.yaml
```

2. Stop the pod directly (only if it still shows "Ready"):

```
sudo crictl stopp <pod-id>
```

3. Once the pod stops, it may also be necessary to stop the container itself:

```
sudo crictl stop <container-id>
```

Troubleshooting Common Errors

This section provides a list of common errors that may be encountered when spawning a container. These account for the vast majority of deployment errors and are easy to verify first before trying to parse the Kubelet journal log.

> **Info**
>
> If more troubleshooting is required, refer to the matching section in the Troubleshooting Guide.

Yaml Syntax

The syntax of the `.yaml` file is extremely sensitive and minor indentation changes may cause it to stop working. The file uses spaces (' ') for indentations (two per indent). Using any other number of spaces causes an undefined behavior.

Huge Pages

The container only spawns once all the required system resources are allocated on the DPU and can be reserved for the container. The most notable resource is huge pages.

1. Before deploying the container, make sure that:

    1. Huge pages are allocated as required per container.
    2. Both the amount and size of pages match the requirements precisely.
2. Once new huge pages are allocated, it is recommended to restart the container service to apply the change:

```
sudo systemctl restart kubelet.service
sudo systemctl restart containerd.service
```

3. Once the above operations are completed successfully, the container could be deployed (YAML can be copied to `/etc/kubelet.d`).

Advanced Troubleshooting

Manual Execution from Within Container - Debugging

> **Note**
>
> The deployment described in this section requires an in-depth knowledge of the container's structure. As this structure might change from version to version, it is only recommended to use this deployment for debugging, and only after other debugging steps have been attempted.

Although most containers define the `entrypoint.sh` script as the container's ENTRYPOINT, this option is only valid for interaction-less sessions. In some debugging scenarios, it is useful to have better control of the programs executed within the container via an interactive shell session. Hence, the `.yaml` file supports an additional execution option.

Uncommenting (i.e., removing `#` from) the following 2 lines in the `.yaml` file causes the container to boot without spawning the container's entrypoint script.

```
# command: ["sleep"]
# args: ["infinity"]
```

In this execution mode, users can attach a shell to the spawned container:

```
crictl exec -it <container-id> /bin/bash
```

Once attached, users get a full shell session enabling them to execute internal programs directly at the scope of the container.

Air-gapped Container Deployment

Container deployment on the BlueField DPU can be done in air-gapped networks and does not require an Internet connection. As explained previously, per DOCA service container, there are 2 required components for successful deployment:

- Container image – hosted on NVIDIA's NGC catalog
- YAML file for the container

From an infrastructure perspective, one additional module is required:

- `k8s.gcr.io/pause` container image

Pulling Container for Offline Deployment

When preparing an air-gapped environment, users must pull the required container images in advance so they could be imported locally to the target machine:

```
docker pull <container-image:tag>
docker save <container-image:tag> > <name>.tar
```

The following example pulls DOCA Firefly `1.1.0-doca2.0.2`:

```
docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2
docker save nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2 >
firefly_v1.1.0.tar
```

Note

Some of DOCA's container images support multiple architectures, causing the `docker pull` command to pull the image according to the architecture of the machine on which it is invoked. Users may force the operation to pull an Arm image by passing the `--platform` flag:

```
docker pull --platform=linux/arm64 <container-
image:tag>
```

Importing Container Image

After exporting the image from the container catalog, users must place the created `*.tar` files on the target machine on which to deploy them. The import command is as follows:

```
ctr --namespace k8s.io image import <name>.tar
```

For example, to import the firefly `.tar` file pulled in the previous section:

```
ctr --namespace k8s.io image import firefly_v1.1.0.tar
```

Examining the status of the operation can be done using the image inspection command:

```
crictl images
```

Built-in Infrastructure Support

The DOCA image comes pre-shipped with the `k8s.gcr.io/pause` image:

```
/opt/mellanox/doca/services/infrastructure/
    docker_pause_3_2.tar
    enable_offline_containers.sh
```

This image is imported by default during boot as part of the automatic activation of DOCA Telemetry Service (DTS).

> **Note**
>
> Importing the image independently of DTS can be done using the `enable_offline_container.sh` script located under the same directory as the image's `*.tar` file.

In versions prior to DOCA 4.2.0, this image can be pulled and imported as follows:

- Exporting the image:

```
docker pull k8s.gcr.io/pause:3.2
docker save k8s.gcr.io/pause:3.2 > docker_pause_3_2.tar
```

- Importing the image:

```
ctr --namespace k8s.io image import docker_pause_3_2.tar
crictl images
IMAGE                                   TAG                 IMAGE
ID              SIZE
k8s.gcr.io/pause                        3.2
2a060e2e7101d       487kB
```

DOCA Services for Host

A subset of the DOCA services are available for host-based deployment as well. This is indicated in those services' deployment and can also be identified by having container tags on NGC with the `*-host` suffix.

In contrast to the managed DPU environment, the deployment of DOCA services on the host is based on docker. This deployment can be extended further based on the user's own container runtime solution.

Docker Deployment

DOCA services for the host are deployed directly using Docker.

1. Make sure Docker is installed on your host. Run:

```
docker version
```

If it is not installed, visit the official Install Docker Engine webpage for installation instructions.

2. Make sure the Docker service is started. Run:

```
sudo systemctl daemon-reload
sudo systemctl start docker
```

3. Pull the container image directly from NGC (can also be done using the `docker run` command):

    1. Visit the NGC page of the desired container.

    2. Under the "Tags" menu, select the desired tag and click the paste icon so it is copied to the clipboard.

    3. The docker pull command will be as follows:

    ```
    sudo docker pull <NGC container tag here>
    ```

    For example:

    ```
    sudo docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2-host
    ```

    > **Note**
    >
    > For DOCA services with deployments on both DPU and host, make sure to select the tag ending with `-host`.

4. Deploy the DOCA service using Docker:

    1. The deployment is performed using the following command:

```
sudo docker run --privileged --net=host -v <host
directory>:<container directory> -e <env variables> -it
<container tag> /entrypoint.sh
```

> **Info**
>
> For more information, refer to Docker's official documentation.

    2. The specific deployment command for each DOCA service is listed in their respective deployment guide.

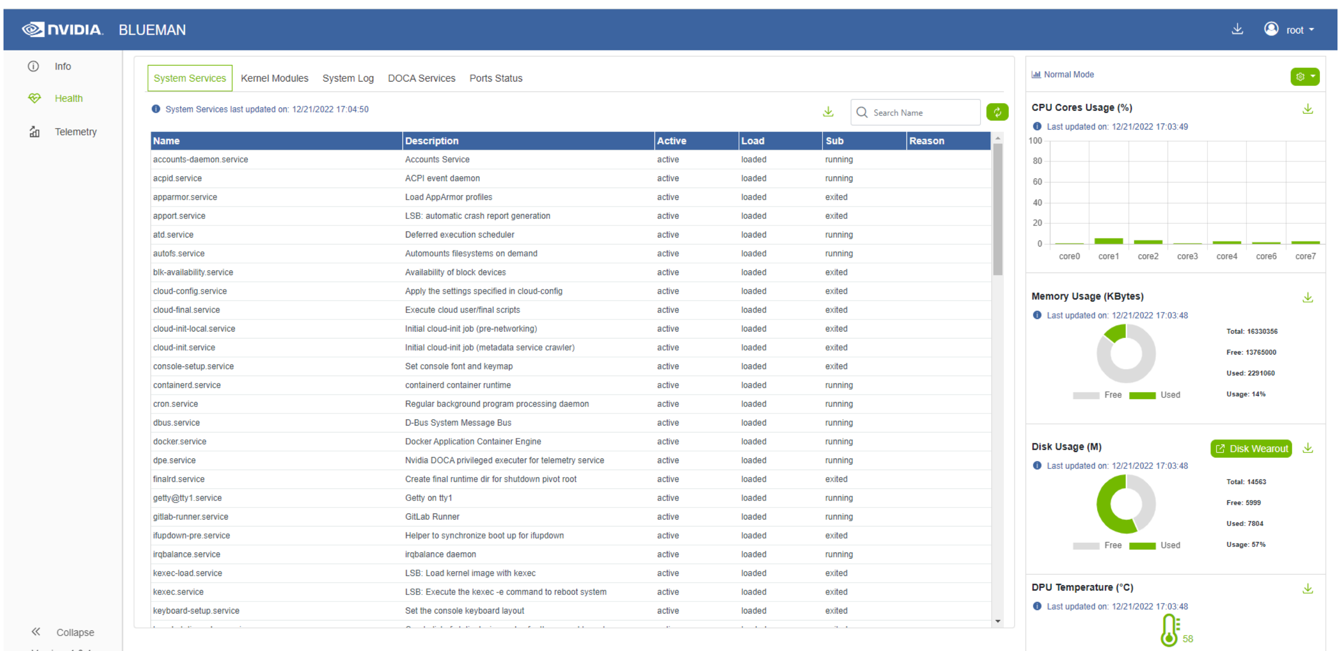
NVIDIA DOCA BlueMan Service Guide

This guide provides instructions on how to use the DOCA BlueMan service on top of NVIDIA® BlueField® DPU.

Introduction

DOCA BlueMan runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface.

All the information that BlueMan provides is gathered from the DOCA Telemetry Service (DTS), starting from DTS version 1.11.1-doca1.5.1.



Requirements

- BlueField image version 3.9.3.1 or higher
- DTS and the DOCA Privileged Executer (DPE) daemon must be up and running

Verifying DTS Status

All the information that BlueMan provides is gathered from DTS .

Verify that the state of the DTS pod is `ready`:

```
$ crictl pods --name doca-telemetry-service
```

Verify that the state of the DTS container is `running`:

```
$ crictl ps --name doca-telemetry-service
```

Verifying DPE Status

All the information that DTS gathers for BlueMan is from the the DPE daemon .

Verify that the DPE daemon is `active`:

```
$ systemctl is-active dpe.service
active
```

If the daemon is inactive, activate it by starting the `dpe.service`:

```
$ systemctl start dpe.service
```

Service Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to the NVIDIA DOCA Container Deployment Guide.

DOCA Service on NGC

BlueMan is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's container page.

Default Deployment – BlueField BSP

BlueMan service is located under `/opt/mellanox/doca/services/blueman` /.

The following is a list of the files under the BlueMan directory:

```
doca_blueman_fe_service_<version>-doca<version>_arm64.tar
doca_blueman_conv_service_<version>-doca<version>_arm64.tar
doca_blueman_standalone.yaml
bring_up_doca_blueman_service.sh
```

Enabling BlueMan Service

Using Script

Run `bring_up_doca_blueman_service.sh`:

```
$ chmod +x
/opt/mellanox/doca/services/blueman/bring_up_doca_blueman_service.s
$
/opt/mellanox/doca/services/blueman/bring_up_doca_blueman_service.s
```

Manual Procedure

1. Import images to crictl images:

```
$ cd /opt/mellanox/doca/services/blueman/
$ ctr --namespace k8s.io image import
doca_blueman_fe_service_<version>-doca<version>_arm64.tar
$ ctr --namespace k8s.io image import
doca_blueman_conv_service_<version>-doca<version>_arm64.tar
```

2. Verify that the DPE daemon is active:

```
$ systemctl is-active dpe.service
active
```

If the daemon is inactive, activate it by starting the `dpe.service`:

```
$ systemctl start dpe.service
```

3. Copy `blueman_standalone.yaml` to `/etc/kubelet.d/`:

```
$ cp doca_blueman_standalone.yaml /etc/kubelet.d/
```

Verifying Deployment Success

1. Verify that the DPE daemon is active:

```
$ systemctl is-active dpe.service
```

2. Verify that the state of the DTS container is `running`:

```
$ crictl ps --name doca-telemetry-service
```

3. Verify that the state of the BlueMan service container is `running`:

```
$ crictl ps --name doca-blueman-fe
$ crictl ps --name doca-blueman-conv
```

Configuration

The configuration of the BlueMan back end is located under `/opt/mellanox/doca/services/telemetry/config/blueman_config.ini`. Users can interact with the `blueman_config.ini` file which contains the default range values of the Pass, Warning, and Failed categories which are used in the health page. Changing these values gets reflected in the BlueMan webpage within 60 seconds.

Example of `blueman_config.ini`:

```
;Health Cpu usages Pass, warning, Failed
[Health:CPU_Usages:Pass]
range = 0,80
[Health:CPU_Usages:Warning]
range = 80,90
[Health:CPU_Usages:Failed]
range = 90,100
```

Collected Data

- Info
  - General info – OS name, kernel, part number, serial number, DOCA version, driver, board ID, etc.
  - Installed packages – list of all installed packages on the DPU including their version
  - CPU info – vendor, cores, model, etc.
  - FW info – all the mlxconfig parameters with default/current/next boot data
  - DPU operation mode
- Health
  - System service
  - Kernel modules
  - Dmesg
  - DOCA services
  - Port status of the PF and OOB
  - Core usage and processes running on each core
  - Memory usage
  - Disk usage
  - Temperature
- Telemetry – all telemetry counters that come from DTS according to the enabled providers displayed on tables
  - Users have the ability to build graphs of specific counters

Connecting to BlueMan Web Interface

To log into BlueMan, enter the IP address of the DPU's OOB interface (`http://<DPU_OOB_IP>`) to a web browser located in the same network as the DPU .

The login credentials to use are the same pair used for the SSH connection to the DPU.

Troubleshooting

For general troubleshooting, refer to the NVIDIA DOCA Troubleshooting Guide.

For container-related troubleshooting, refer to the "Troubleshooting" section in the NVIDIA DOCA Container Deployment Guide.

The following are additional troubleshooting tips for DOCA BlueMan:

- The following error message in the login page signifies a failure to connect to the DPE daemon: "The service is currently unavailable. Please check server up and running."

    1. Restart the DPE daemon:

    ```
    $ systemctl restart dpe.service
    ```

2. Verify that DTS is up and running by following the instructions in section "Verifying DTS Status".

- If the message "Invalid Credentials" appears in the login page, v erify that the username and password are the same ones used to SSH to the DPU.
- If all of the above is configured as expected and there is still some failure to log in, it is recommended to check if there are any firewall rules that block the connection.
- For other issues, check the `/var/log/syslog` and `/var/log/doca/telemetry/blueman_service.log` log file.

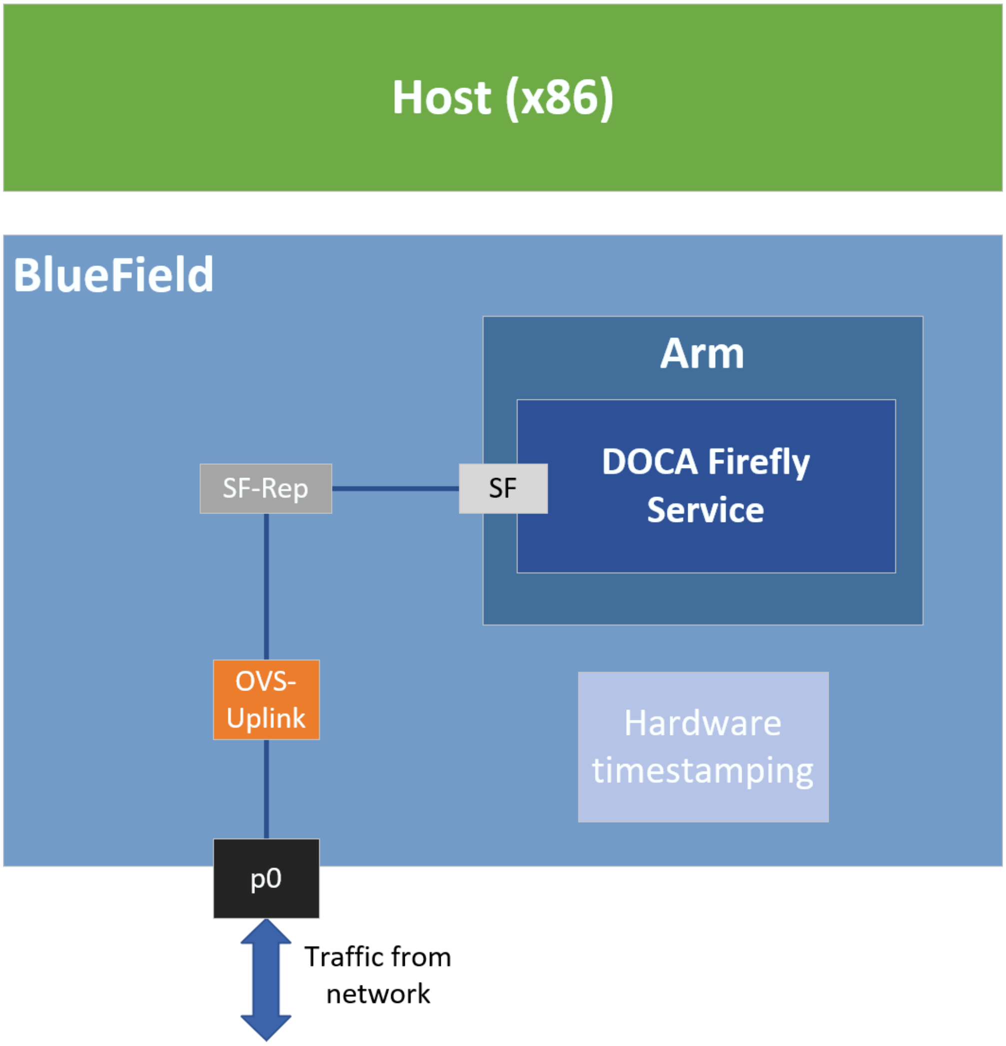
NVIDIA DOCA Firefly Service Guide

This guide provides instructions on how to use the DOCA Firefly service container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Firefly Service provides precision time protocol (PTP) based time syncing services to the BlueField DPU .

PTP is a protocol used to synchronize clocks in a network. When used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, which is far better than is what is normally obtainable with network time protocol (NTP). PTP support is divided between the kernel and user space. The ptp4l program implements the PTP boundary clock and ordinary clock. With hardware time stamping, it is used to synchronize the PTP hardware clock to the master clock.



Requirements

Some of the features provided by Firefly require specific BlueField DPU hardware capabilities:

- PTP – Supported by all BlueField DPUs
- PPS – Requires BlueField DPU with PPS capabilities
- SyncE - Requires converged card BlueField DPUs

Failure to run PPS due to missing hardware support will be noted in the service's output. However, the service will continue to run the timing services it can provide on the provided hardware.

Firmware Version

Firmware version must be 24.34.1002 or higher.

BlueField BSP Version

Supported BlueField image versions are 3.9.0 and higher.

Embedded Mode

Configuring Firmware Settings on DPU for Embedded Mode

1. Set the DPU to embedded mode (default mode):

```
sudo mlxconfig -y -d 03:00.0 s INTERNAL_CPU_MODEL=1
```

2. Enable the real time clock (RTC):

```
sudo mlxconfig -d 03:00.0 set REAL_TIME_CLOCK_ENABLE=1
```

3. Graceful shutdown and power cycle the DPU to apply the configuration.

4. You may check the DPU mode using the following command:

```
sudo mlxconfig -d 03:00.0 q | grep INTERNAL_CPU_MODEL
# Example output
         INTERNAL_CPU_MODEL                  EMBEDDED_CPU(1)
```

Ensuring OVS Hardware Offload

DOCA Firefly requires that hardware offload is activated in Open vSwitch (OVS). This is enabled by default as part of the BFB image installed on the DPU.

To verify the hardware offload configuration in OVS:

```
sudo ovs-vsctl get Open_vSwitch . other_config | grep hw-offload
# Example output
         {hw-offload="true"}
```

If inactive:

1. Activate hardware offloading by running:

   ```
   sudo ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
   ```

2. Restart the OVS service:

   ```
   sudo /etc/init.d/openvswitch-switch restart
   ```

3. Graceful shutdown and power cycle the DPU to apply the configuration.

Helper Scripts

Firefly's deployment contains a script to help with the configuration steps required for the network interface in embedded mode:

- `scripts/doca_firefly/<firefly-version>/prepare_for_embedded_mode.sh`
- `scripts/doca_firefly/<firefly-version>/set_new_sf.sh`

The latest DOCA Firefly version is `1.3.0`.

Both scripts are included as part of DOCA's container resource which can be downloaded according to the instructions in the NVIDIA DOCA Container Deployment Guide. For more information about the structure of the DOCA container resource, refer to section "Structure of NGC Resource" in the deployment guide.

> **Note**
>
> Due to technical limitations of the NGC resource, both scripts are provided without execute (+x) permissions. This could be resolved by running the following command:
>
> ```
> chmod +x scripts/doca_firefly/<firefly-version>/*.sh
> ```

prepare_for_embedded_mode.sh

This script automates all the steps mentioned in section "Setting Up Network Interfaces for Embedded Mode" and configures a freshly installed BFB image to the settings required by DOCA Firefly.

Notes:

- The script deletes all previous OVS settings and creates a single OVS bridge that matches the definitions in section "Setting Up Network Interfaces for Embedded Mode"
- The script should only be run once when connecting to the DPU for the first time or after a power cycle
- The only manual step required after using this script is configuring the IP address for the created network interface (step 5 in section "Setting Up Network Interfaces for Embedded Mode")

Script arguments:

- SF number (checks if already exists)

Examples:

- Prepare OVS settings using an SF indexed 4:

```
chmod +x ./*.sh
./prepare_for_embedded_mode.sh 4
```

The script makes use of `set_new_sf.sh` as a helper script.

set_new_sf.sh

Creates a new trusted SF and marks it as "trusted".

Script arguments:

- PCIe address
- SF number (checks if already exists)
- MAC address (if absent, a random address is generated)

Examples:

- Create SF with number "4" over port 0 of the DPU:

```
./set_new_sf.sh 0000:03:00.0 4
```

- Create SF with number "5" over port 0 of the DPU and a specific MAC address:

```
./set_new_sf.sh 0000:03:00.0 5 aa:bb:cc:dd:ee:ff
```

- Create SF with number "4" over port 1 of the DPU:

```
./set_new_sf.sh 0000:03:00.1 4
```

The first two examples should work out of the box for a BlueField-2 device and create SF4 and SF5 respectively.

Setting Up Network Interfaces for DPU Mode

1. Create a trusted SF to be used by the service according to the Scalable Function Setup Guide .

> **Note**
>
> The following instructions assume that the SF has been created using index 4.

2. Create the required OVS setting as is shown in the architecture diagram:

```
$ sudo ovs-vsctl add-br uplink
$ sudo ovs-vsctl add-port uplink p0
$ sudo ovs-vsctl add-port uplink en3f0pf0sf4
# This port is needed to ensure we have traffic host<->network as well
$ sudo ovs-vsctl add-port uplink pf0hpf
```

3. Verify the OVS settings:

```
sudo ovs-vsctl show
    Bridge uplink
        Port pf0hpf
```

```
            Interface pf0hpf
        Port en3f0pf0sf4
            Interface en3f0pf0sf4
        Port p0
            Interface p0
        Port uplink
            Interface uplink
                type: internal
```

4. Enable TX timestamping on the SF interface (not the representor):

```
# tx port timestamp offloading
sudo ethtool --set-priv-flags enp3s0f0s4 tx_port_ts on
```

5. Enable the interface and set an IP address for it:

```
# configure ip for the interface:
sudo ifconfig enp3s0f0s4 <ip-addr> up
```

6. Configure OVS to support TX timestamping over this SF and multicast traffic in general:

```
# Multicast-related definitions
$ sudo ovs-vsctl set Bridge uplink mcast_snooping_enable=true
$ sudo ovs-vsctl set Bridge uplink other_config:mcast-snooping-disable-flood-unregistered=true
$ sudo ovs-vsctl set Port p0 other_config:mcast-snooping-flood=true
$ sudo ovs-vsctl set Port p0 other_config:mcast-snooping-flood-reports=true
# PTP-related definitions
```

```
$ sudo ovs-ofctl add-flow uplink
in_port=en3f0pf0sf4,udp,tp_src=319,actions=output:p0
$ sudo ovs-ofctl add-flow uplink
in_port=p0,udp,tp_src=319,actions=output:en3f0pf0sf4
$ sudo ovs-ofctl add-flow uplink
in_port=en3f0pf0sf4,udp,tp_src=320,actions=output:p0
$ sudo ovs-ofctl add-flow uplink
in_port=p0,udp,tp_src=320,actions=output:en3f0pf0sf4
```

Note

If your OVS bridge uses a name other than `uplink`, make sure that the used name is reflected in the `ovs-vsctl` and `ovs-ofctl` commands. For instance:

```
$ sudo ovs-vsctl set Bridge <bridge-name>
mcast_snooping_enable=true
```

Separated Mode

Configuring Firmware Settings on DPU for Separated Mode

1. Set the BlueField mode of operation to "Separated":

```
sudo mlxconfig -y -d 03:00.0 s INTERNAL_CPU_MODEL=0
```

2. Enable RTC:

```
sudo mlxconfig -d 03:00.0 set REAL_TIME_CLOCK_ENABLE=1
```

3. Graceful shutdown and power cycle the DPU to apply the configuration.

4. You may check the BlueField's operation mode using the following command:

```
sudo mlxconfig -d 03:00.0 q | grep INTERNAL_CPU_MODEL
# Example output
         INTERNAL_CPU_MODEL
SEPARATED_HOST(0)
```

Setting Up Network Interfaces for Separated Mode

1. Make sure that that `p0` is not connected to an OVS bridge:

```
sudo ovs-vsctl show
```

2. Enable TX timestamping on the `p0` interface:

```
# TX port timestamp offloading (assuming PTP interface is p0)
sudo ethtool --set-priv-flags p0 tx_port_ts on
```

3. Enable the interface and set an IP address for it:

```
# Configure IP for the interface
sudo ifconfig p0 <ip-addr> up
```

Host-based Deployment

Host-based deployment requires the same configuration described under section "Separated Mode".

Service Deployment

DPU Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to NVIDIA DOCA Container Deployment Guide.

Service-specific configuration steps and deployment instructions can be found under the service's container page.

> **Note**
>
> DOCA Firefly can also be deployed on DPUs not connected to the Internet. For instructions, refer to the relevant section in the NVIDIA DOCA Container Deployment Guide.

Host Deployment

DOCA Firefly has a version adapted for host-based deployments. For more information about the deployment of DOCA containers on top of a host, refer to the NVIDIA BlueField DPU Container Deployment Guide.

The following is the docker command for deploying DOCA Firefly on the host:

```
sudo docker run --privileged --net=host -v
/var/log/doca/firefly:/var/log/firefly -v
/etc/firefly:/etc/firefly -e PTP_INTERFACE='eth2' -it
nvcr.io/nvidia/doca/doca_firefly:1.3.0-doca2.5.0-host
/entrypoint.sh
```

Where:

- Additional YAML configs may be passed as environment variables as additional `-e` key-value pairs as done with `PTP_INTERFACE` above
- The exact container tag should be the desired tag as chosen on DOCA Firefly's NGC page

Configuration

All modules within the service have configuration files that allow customizing various settings, both general and PTP-related.

Built-In Config File

Each profile has its own base PTP configuration file for `ptp4l`. For example, the Media profile PTP configuration file is `ptp4l-media.conf`.

The built-in PTP configuration files can be found in section "PTP Profile Default Config Files". For ease-of-use, those files are provided as part of DOCA's container resource as downloaded from NGC and are placed under Firefly's `configs` directory (`scripts/doca_firefly/<firefly version>/configs`).

Note

When using a built-in configuration file, Firefly uses the files as stored within the container itself in the `/etc/linuxptp` directory. The configuration files included in the NGC resource are only provided for ease of access. Modifying them does **not** impact the configuration used in practice by the container. Instead, updates to the configuration should be done as described in the following sections.

Custom Config File

Instead of using a profile's base config file, users can create a file of their own, for each of the modules.

To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

For example, to set a custom `linuxptp` config file, the user can set the parameter `PTP_CONFIG_FILE` in the YAML file:

```
- name: PTP_CONFIG_FILE
  value: my_custom_ptp.conf
```

In this example, `my_custom_ptp.conf` should be placed at `/etc/firefly/my_custom_ptp.conf`.

Note

A config file must not define values for the UDS-related ports (`/var/run/ptp4l` and `/var/run/ptp4lro`), as those will impact internal container behavior. Such settings will prompt a warning and will be ignored when preparing the finalized configuration (See more in the next sections).

Overriding Specific Config File Parameters

Instead of replacing the entire config file, users may opt to override specific parameters. This can be done using the following variable syntax in the YAML file: CONF_<TYPE>_<SECTION>_<PARAMETER_NAME>.

- `TYPE` – either `PTP` , `PHC2SYS`, SYNCE, `MONITOR`
- `SECTION` – the section in the config file that the parameter should be placed in

> **Note**
>
> If the specified section does not already exist in the config file, a new section is created unless it refers to a PTP network interface that has not been included in the `PTP_INTERFACE` YAML field.

- `PARAMETER_NAME` – the config parameter name as should be placed in the config file

> **Note**
>
> If the parameter name already exists in the config file, then the value is changed according to the value provided in the `.yaml` file. If the parameter name does not already exist in the config file, then it is added.

For example, the following variable in the YAML file definition changes the value of the parameter `priority1` under section `global` in the PTP config file to `64`.

```
- name: CONF_PTP_global_priority1
```

```
value: "64"
```

> **Note**
>
> Configuring `unicast_master_table` through the YAML file is not supported due to the structure of the table (i.e., multiple entries sharing the same key).

Ensuring and Debugging Correctness of Config Files

The previous sections describe 2 layers for the configuration file definitions:

- Basic configuration file – either a built-in config file or a custom config file
- Adding/overriding values to/from the YAML file

In practice, there are slightly more layers in place, and the precedence is as follows (presented in increasing order):

- Default configuration values of the PTP program (ptp4l for instance) – holds values of all available configuration options
- Your chosen configuration file – contains a subset of options
- Definitions from the YAML file – narrower subset
- Firefly mandatory values

When combining the supplied configuration file with the definitions from the YAML file, Firefly goes over those definitions and checks them against a predefined set of configuration options:

- Warning only – warns if a certain value leads to known issues in a supported deployment scenario

- Override – container-internal definitions that should not be set by the user and will be overridden by Firefly

Suitable log messages are provided in either case:

```
# Example for a warning
2023-01-31 11:55:13 - Firefly - Config - INFO    - Missing
explicit definition "fault_reset_interval", verifying default
value instead: "4"
2023-01-31 11:55:13 - Firefly - Config - WARNING - Value "4" for
definition "fault_reset_interval" will be invalid in Embedded
Mode, expected a value lesser or equal to "1"
2023-01-31 11:55:13 - Firefly - Config - WARNING - Continuing
with invalid value
# Example for an override
2023-01-31 11:21:00 - Firefly - Config - WARNING - Invalid value
"/var/run/ptp4l2" for definition "uds_address", expected
"/var/run/ptp4l"
2023-01-31 11:21:00 - Firefly - Config - INFO    - Setting
definition "uds_address" value to the following: "/var/run/ptp4l"
```

At the end of this process, an updated configuration file is generated by Firefly to be used later by the various time providers (as mentioned below). To avoid accidental modification of a user-supplied configuration file or permission issues, the finalized file is generated within the container under the `/tmp` directory.

For instance, if using a custom configuration file named `my_custom_ptp.conf` under the `/etc/firefly` directory on the DPU, the updated file will reside within the container at the following path: `/tmp/my_custom_ptp.conf`.

For troubleshooting possible issues with the configuration file, one can do one of the following:

- Connect to the container directly as is explained in the debugging finalized configuration file bullet under "Troubleshooting".

- Map the container's `/tmp` directory to the DPU using the built-in support in the YAML file:
  - Before the change:

```
    # Uncomment when debugging the finalized
configuration files used - Part #1
    #- name: debug-firefly-volume
    #  hostPath:
    #    path: /tmp/firefly
    #    type: DirectoryOrCreate
  containers:
    ...
      volumeMounts:
      - name: logs-firefly-volume
        mountPath: /var/log/firefly
      - name: conf-firefly-volume
        mountPath: /etc/firefly
      # Uncomment when debugging the finalized
configuration files used - Part #2
      #- name: debug-firefly-volume
      #  mountPath: /tmp
```

  - After the change:

```
    # Uncomment when debugging the finalized
configuration files used - Part #1
    - name: debug-firefly-volume
      hostPath:
        path: /tmp/firefly
        type: DirectoryOrCreate
  containers:
    ...
      volumeMounts:
```

```
      - name: logs-firefly-volume
        mountPath: /var/log/firefly
      - name: conf-firefly-volume
        mountPath: /etc/firefly
      # Uncomment when debugging the finalized
configuration files used - Part #2
      - name: debug-firefly-volume
        mountPath: /tmp
```

> **Note**
>
> The finalized configuration file keeps the sections and config options in the same order as they appear in the original file, yet the file is stripped from spare new lines or comment lines. This should be taken into considerations when directly accessing it during a debugging session.

Description

Providers

DOCA Firefly Service uses the following third-party providers to provide time syncing services:

- Linuxptp - Version v4.1
  - `PTP` – PTP service, provided by the PTP4L program
  - `PHC2SYS` – OS time calibration, provided by the PHC2SYS program
- Testptp
  - `PPS` - PPS settings service

In addition, DOCA Firefly Service also makes use of the following NVIDIA modules:

- SyncE
  - `SYNCE` – Synchronous Ethernet Deamon (`synced`)
- Firefly
  - `MONITOR` - Firefly PTP Monitor

Each of the providers can be enabled, disabled, or set to use the setting defined by the configuration profile:

- YAML setting – `<provider name>_STATE`
- Supported values – `enable`, `disable`, `defined_by_profile`

Note

For the default profile settings per provider, refer to the table under section "Profiles".

An example YAML setting for specifically disabling the `phc2sys` provider is the following:

```
- name: PHC2SYS_STATE
  value: "disable"
```

Note

The `defined_by_profile` setting is only available for well-defined profiles. As such, it cannot be used when the `custom` profile is selected. For more information about the profile settings, refer to the table under section "Profiles".

Profiles

DOCA Firefly Service includes profiles which represent common use cases for the Firefly service that provide a different default configuration per profile:

Profiles	Default	Media	Custom
Purpose	Any user that requires PTP	Media productions	Custom configuration for a dedicated user scenario
PTP	Enabled	Enabled	No default. Enable/disable should be set by the user.
PTP profile	PTP default profile	SMPTE 2059-2	Set by the user
PTP Client/Server(a)	Both	Client-only	Set by the user
PHC2SYS	Enabled	Enabled	No default. Enable/disable should be set by the user.
PPS (in/out)	Enabled	Enabled	No default. Enable/disable should be set by the user.
PTP Monitor	Disabled	Disabled	No default. Enable/disable should be set by the user.
SyncE	Disabled	Disabled	No default. Enable/disable should be set by the user.

Note

[a] Client-only is only relevant to a single PTP interface. If more than one PTP interface is provided in the YAML file, both modes are enabled.

Outputs

Container Output

While running, the full output of the DOCA Firefly Service container can be viewed using the following command:

```
sudo crictl logs <CONTAINER-ID>
```

Where `CONTANIER-ID` can be retrieved using the following command:

```
sudo crictl ps
```

For example, in the following output, the container ID is `8f368b98d025b`.

```
$ sudo crictl ps
CONTAINER           IMAGE               CREATED             STATE
              NAME                ATTEMPT             POD ID
         POD
8f368b98d025b       289809f312b4c       2 seconds ago
Running             doca-firefly        0
5af59511b4be4       doca-firefly-some-computer-name
```

The output of the container depends on the services supported by the hardware and enabled by configuration and the selected profile. However, note that any of the configurations runs PTP, so when DOCA FireFly is running successfully expect to see the line "`Running ptp4l`".

The following is an example of the expected container output when running the default profile on a DPU that supports PPS:

```
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Starting
DOCA Firefly - Version 1.3.0
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Selected
features:
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PTP
- Enabled - ptp4l will be used
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] MONITOR
- Enabled - PTP Monitor will be used
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PHC2SYS
- Enabled - phc2sys will be used
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [-] SyncE
- Disabled
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PPS
- Enabled - testptp will be used (if supported by hardware)
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Going to
analyze the configuration files
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Requested
the following PTP interface: p0
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Starting PPS
configuration
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PPS is
supported by hardware
2023-09-07 14:04:23 - Firefly - Init    - INFO     - set pin
function okay
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PPS in -
Activated
```

```
2023-09-07 14:04:23 - Firefly - Init    - INFO     - set pin
function okay
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PPS out
- Activated
2023-09-07 14:04:23 - Firefly - Init    - INFO     - name
mlx5_pps0 index 0 func 1 chan 0
2023-09-07 14:04:23 - Firefly - Init    - INFO     - name
mlx5_pps1 index 1 func 2 chan 0
2023-09-07 14:04:23 - Firefly - Init    - INFO     - periodic
output request okay
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Running
ptp4l
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Running
Firefly PTP Monitor
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Running
phc2sys
```

The following is an example of the expected container output when running the default profile on a DPU that does not support PPS:

```
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Starting
DOCA Firefly - Version 1.3.0
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Selected
features:
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PTP
- Enabled - ptp4l will be used
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] MONITOR
- Enabled - PTP Monitor will be used
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PHC2SYS
- Enabled - phc2sys will be used
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [-] SyncE
- Disabled
```

```
2023-09-07 14:04:23 - Firefly - Init    - INFO     - [+] PPS
- Enabled - testptp will be used (if supported by hardware)
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Going to
analyze the configuration files
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Requested
the following PTP interface: p0
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Starting PPS
configuration
2023-09-07 14:04:23 - Firefly - Init    - WARNING  - [-] PPS
capability is missing, seems that the card doesn't support PPS
2023-09-07 14:04:23 - Firefly - Init    - INFO     -
capabilities:
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   50000000
maximum frequency adjustment (ppb)
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0
programmable alarms
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 external
time stamp channels
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0
programmable periodic signals
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 pulse
per second
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0
programmable pins
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 cross
timestamping
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Running
ptp4l
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Running
Firefly PTP Monitor
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Running
phc2sys
```

Firefly Output

On top of the container's log, Firefly defines an additional, non-volatile log that can be found in `/var/log/doca/firefly/firefly.log`.

This file contains the same output described in section "Container Output", and is useful for debugging deployment errors should the container stop its execution.

> **Note**
>
> To avoid disk space issues, the `/var/log/doca/firefly/firefly.log` file only contains the log from Firefly's initialization, and not the logs of the rest of the modules (ptp4l, phc2sys, etc.) or that of the PTP monitor. The latter is still included in the container log and can be inspected using the command `sudo crictl logs <CONTAINER-ID>`.

ptp4l Output

The ptp4l output can be found in the file `/var/log/doca/firefly/ptp4l.log`.

Example output:

```
ptp4l[192710.691]: rms 1 max 1 freq -114506 +/- 0 delay -15 +/- 0
ptp4l[192712.692]: rms 6 max 9 freq -114501 +/- 3 delay -15 +/- 0
ptp4l[192714.692]: rms 7 max 9 freq -114511 +/- 3 delay -13 +/- 0
ptp4l[192716.692]: rms 5 max 7 freq -114502 +/- 1 delay -13 +/- 0
ptp4l[192718.693]: rms 4 max 6 freq -114509 +/- 2 delay -13 +/- 0
ptp4l[192720.693]: rms 3 max 3 freq -114506 +/- 2 delay -13 +/- 0
ptp4l[192722.694]: rms 4 max 6 freq -114510 +/- 3 delay -12 +/- 0
ptp4l[192724.694]: rms 5 max 7 freq -114510 +/- 5 delay -12 +/- 1
ptp4l[192726.695]: rms 4 max 5 freq -114508 +/- 3 delay -11 +/- 0
```

```
ptp4l[192728.695]: rms 6 max 9 freq -114504 +/- 4 delay -11 +/- 0
```

phc2sys Output

The phc2sys output can be found in the file `/var/log/doca/firefly/phc2sys.log`.

Example output:

```
phc2sys[1873325.928]: reconfiguring after port state change
phc2sys[1873325.928]: selecting CLOCK_REALTIME for
synchronization
phc2sys[1873325.928]: selecting enp3s0f0s4 as the master clock
phc2sys[1873325.928]: CLOCK_REALTIME phc offset      1378 s2 freq
-165051 delay    255
phc2sys[1873326.928]: CLOCK_REALTIME phc offset      1378 s2 freq
-163673 delay    240
phc2sys[1873327.928]: port 62b785.fffe.0c9369-1 changed state
phc2sys[1873327.929]: CLOCK_REALTIME phc offset        14 s2 freq
-164624 delay    255
phc2sys[1873328.936]: CLOCK_REALTIME phc offset        89 s2 freq
-164545 delay    240
```

SyncE Output

The SyncE output can be found in the file `/var/log/doca/firefly/synced.log`.

Example output:

```
INFO     [05/09/2023 05:11:01.493414]: SyncE Group #0: is in
TRACKING holdover acquired mode on p0, frequency_diff: 0 (ppb)
INFO     [05/09/2023 05:11:02.502963]: SyncE Group #0: is in
TRACKING holdover acquired mode on p0, frequency_diff: -113 (ppb)
```

```
INFO     [05/09/2023 05:11:03.512491]: SyncE Group #0: is in
TRACKING holdover acquired mode on p0, frequency_diff: 37 (ppb)
```

Note

The verbosity of the output from the `SYNCE` module is limited by default. To set the output to be more verbose, set the `verbose` option to `1` (True).

Before:

```
# Example #4 - Overwrite the value of verbose in
the [global] section of the SyncE configuration
file.
#- name: CONF_SYNCE_global_verbose
#  value: "1"
```

After:

```
# Example #4 - Overwrite the value of verbose in
the [global] section of the SyncE configuration
file.
- name: CONF_SYNCE_global_verbose
  value: "1"
```

Tx Timestamping Support on DPU Mode

When the BlueField is operating in DPU mode, additional OVS configuration is required as mentioned in step 6 of section "Setting Up Network Interfaces for DPU Mode". This configuration achieves the following:

- Proper support for incoming/outgoing multicast traffic
- Enabling Tx timestamping

Firefly only gets the packet timestamping for outgoing PTP messages (Tx timestamping) when they are offloaded to the hardware. As such, when working with OVS, users must ensure this traffic flow is properly recognized and offloaded. If offloading does not take place, Firefly gets stuck in a fault loop while waiting to receive the Tx timestamp events:

```
ptp4l[2912.797]: timed out while polling for tx timestamp
ptp4l[2912.797]: increasing tx_timestamp_timeout may correct this
issue, but it is likely caused by a driver bug
ptp4l[2912.797]: port 1 (enp3s0f0s4): send sync failed
ptp4l[2923.528]: timed out while polling for tx timestamp
ptp4l[2923.528]: increasing tx_timestamp_timeout may correct this
issue, but it is likely caused by a driver bug
ptp4l[2923.528]: port 1 (enp3s0f0s4): send sync failed
```

The solution to this issue:

- Activation of hardware offloading in OVS
- OpenFlow rules that ensure OVS properly recognizes the traffic and offloads it to the hardware
- Modification to the `fault_reset_interval` configuration value to ensure timely recovery from the fault induced by the first packet being always treated by software (until the rule is offloaded to hardware). As such, Firefly requires that the `fault_reset_interval` value is 1 or less. Proper warnings are raised if an improper value is detected. The value is updated accordingly in the built-in profiles.

When these configurations are in order, Firefly includes a report for a single fault during boot, but recovers from it and continues as usual:

```
ptp4l[3715.687]: timed out while polling for tx timestamp
ptp4l[3715.687]: increasing tx_timestamp_timeout may correct this
issue, but it is likely caused by a driver bug
ptp4l[3715.687]: port 1 (enp3s0f0s4): send delay request failed
```

Troubleshooting Tx Timestamp Issues

As explained earlier, there are several layers required to ensure Tx timestamping works as necessary by Firefly. The following is a list of commands to debug the state of each layer:

1. Inspect the OpenFlow rules:

```
$ sudo ovs-ofctl dump-flows uplink
cookie=0x0, duration=4075.576s, table=0, n_packets=2437,
n_bytes=209582, udp,in_port=en3f0pf0sf4,tp_src=319
actions=output:p0
cookie=0x0, duration=4075.549s, table=0, n_packets=1216,
n_bytes=109420, udp,in_port=p0,tp_src=319
actions=output:en3f0pf0sf4
cookie=0x0, duration=4075.521s, table=0, n_packets=13,
n_bytes=1242, udp,in_port=en3f0pf0sf4,tp_src=320
actions=output:p0
cookie=0x0, duration=4074.604s, table=0, n_packets=3034,
n_bytes=297376, udp,in_port=p0,tp_src=320
actions=output:en3f0pf0sf4
cookie=0x0, duration=4075.856s, table=0, n_packets=184,
n_bytes=12901, priority=0 actions=NORMAL
```

2. Inspect hardware TC rules while DOCA Firefly is deployed (the rules age out after 10 seconds without traffic):

```
$ sudo tc -s -d filter show dev en3f0pf0sf4 egress
filter ingress protocol ip pref 4 flower chain 0
```

```
filter ingress protocol ip pref 4 flower chain 0 handle 0x1
  eth_type ipv4
  ip_proto udp
  src_port 320
  ip_flags nofrag
  in_hw in_hw_count 1
        action order 1: mirred (Egress Redirect to device p0)
stolen
        index 3 ref 1 bind 1 installed 7 sec used 7 sec
        Action statistics:
        Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues
0)
        backlog 0b 0p requeues 0
        cookie bec8bd6ede4e86341e9045a6edb58ca2
        no_percpu

filter ingress protocol ip pref 4 flower chain 0 handle 0x2
  eth_type ipv4
  ip_proto udp
  src_port 319
  ip_flags nofrag
  in_hw in_hw_count 1
        action order 1: mirred (Egress Redirect to device p0)
stolen
        index 4 ref 1 bind 1 installed 6 sec used 6 sec
        Action statistics:
        Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues
0)
        backlog 0b 0p requeues 0
        cookie c568d97efd400de98608fbbf86ccdf3c
        no_percpu
```

ⓘ **Note**

If no TC rules are present when Firefly is running, this usually indicates that hardware offloading is disabled at the OVS level, in which case it should be activated as explained under "Ensuring OVS Hardware Offload".

PTP

Firefly uses the `ptp4l` utility to handle the Precision Time Protocol (IEEE 1588).

Through the YAML file, users can configure the network interfaces used for the protocol:

```
# Network interfaces to be used (For multiple interfaces use a
space ("") separated list)
- name: PTP_INTERFACE
  # Set according to used interfaces on the local setup
  value: "p0"
```

Before the deployment of the container, users should configure this field to point at the desired network interface(s) configured in the previous steps.

PHC2SYS

Firefly uses the `phc2sys` utility to synchronize the OS's clock to the accurate time stamps received by `ptp4l`.

Through the YAML file, users can configure the command-line arguments used by the `phc2sys` program:

```
- name: PHC2SYS_ARGS
```

```
value: "-a -r"
```

Firefly adds the following command-line arguments on top of the user-selected flags:

- Use of chosen configuration file (empty configuration file by default, or user-supplied file if specified in the YAML file)
- Redirection of output to a log file using the `-m` command line option

> **Note**
>
> `phc2sys` must use the same `domainNumber` setting used by `ptp4l`. If the same `domainNumber` is not set by the user, Firefly does that automatically.

> **Note**
>
> `phc2sys` is only able to accurately sync the clock of the hosting environment (usually the DPU, but may also be the host if deployed there) if other timing services, such as NTP, are disabled.
>
> So, for instance, on Ubuntu 22.04, users must ensure that the NTP timing service is disabled by running:
>
> ```
> systemctl stop systemd-timesyncd
> ```

SYNCE

> **Note**
>
> The SyncE module is supported at alpha level.

Firefly uses the proprietary `synced` utility to implement the Synchronous Ethernet protocol, aimed at ensuring synchronization of the clock's frequency with the reference clock. Once achieved, both clocks are declared as "syntonized".

Through the YAML file, users can configure the network interfaces used for the protocol:

```
# Network interfaces to be used (For multiple interfaces use a
space ("") separated list)
- name: SYNCE_INTERFACE
  # Set according to used interfaces on the local setup
  value: "p0"
```

Before the deployment of the container, one should configure this field to point at the desired network interface(s) configured in the previous steps.

> **Note**
>
> SyncE is currently only supported over network interfaces of the DPU's physical functions (i.e., `p0` or `p1`).

PTP Monitoring

> **Note**

Monitoring is still in beta phase. There will be updates to the API in the near future.

PTP monitoring periodically queries for various PTP-related information and prints it to the container's log.

The following is a sample output of this tool:

```
gmIdentity:                 48:B0:2D:FF:FE:5C:4D:24
(48b02d.fffe.5c4d24)
portIdentity:               48:B0:2D:FF:FE:5C:53:44
(48b02d.fffe.5c5344-1)
port_state:                 Active
domainNumber:               2
master_offset:              avg: 1      max:    -8      rms:    3
gmPresent:                  true
ptp_stable:                 Recovered
UtcOffset:                  37
timeTraceable:              0
frequencyTraceable:         0
grandmasterPriority1:       128
gmClockClass:               248
gmClockAccuracy:            0x6
grandmasterPriority2:       128
gmOffsetScaledLogVariance:  0xffff
ptp_time (TAI):             Thu Sep  7 11:22:50 2023
ptp_time (UTC adjusted):    Thu Sep  7 11:22:13 2023
system_time (UTC):          Thu Sep  7 11:22:13 2023
error_count:                1
last_err_time (UTC):        Thu Sep  7 09:55:48 2023
```

Among others, this monitoring provides the following information:

- Details about the Grandmaster the DPU is syncing with

- Current PTP timestamp
- Health information such as connection errors during execution and whether they have been recovered from

PTP monitoring is disabled by default and can be activated by replacing the `disable` value with the IP address for the monitor server to use:

```
- name: MONITOR_STATE
  Value: "<IP address for the monitoring server>"
```

Once activated, the information can viewed from the container using the following command:

```
sudo crictl logs --tail=20 <CONTAINER-ID>
```

It is recommended to use the following `watch` command to actively monitor the PTP state:

```
sudo watch -n 1 crictl logs --tail=20 <CONTAINER-ID>
```

When triaging deployment issues, additional logging information can be found in the monitor's developer logs: `/var/log/doca/firefly/firefly_monitor_dev.log`.

> **Note**
>
> The monitoring feature connects to ptp4l's local UDS server to query the necessary information. This is why the configuration manager prevents users from modifying the `uds_address` and `uds_ro_address` fields used by ptp4l within the container.

Configuration

The PTP monitor supports configuration options which are passed through a dedicated configuration file like the rest of DOCA Firefly's modules. The built-in monitor configuration file can be found in the section "PTP Monitor". For ease of use, the file is also provided as part of DOCA's container resource as downloaded from NGC.

"Firefly Modules Configuration Options" contains a complete explanation of each of the configuration options alongside their default values.

To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

```
- name: MONITOR_CONFIG_FILE
  value: my_custom_monitor.conf
```

In this example, `my_custom_monitor.conf` should be placed at `/etc/firefly/my_custom_monitor.conf`.

Time Representations (PTP Time vs System Time)

Under most deployment scenarios, the PTP time shown by the monitor is presented according to the International Atomic Time (TAI) standard, while the system time would most commonly use the Coordinated Universal Time (UTC). Due to the differences between these time representation models, the monitor provides 2 different time readings (each marked accordingly):

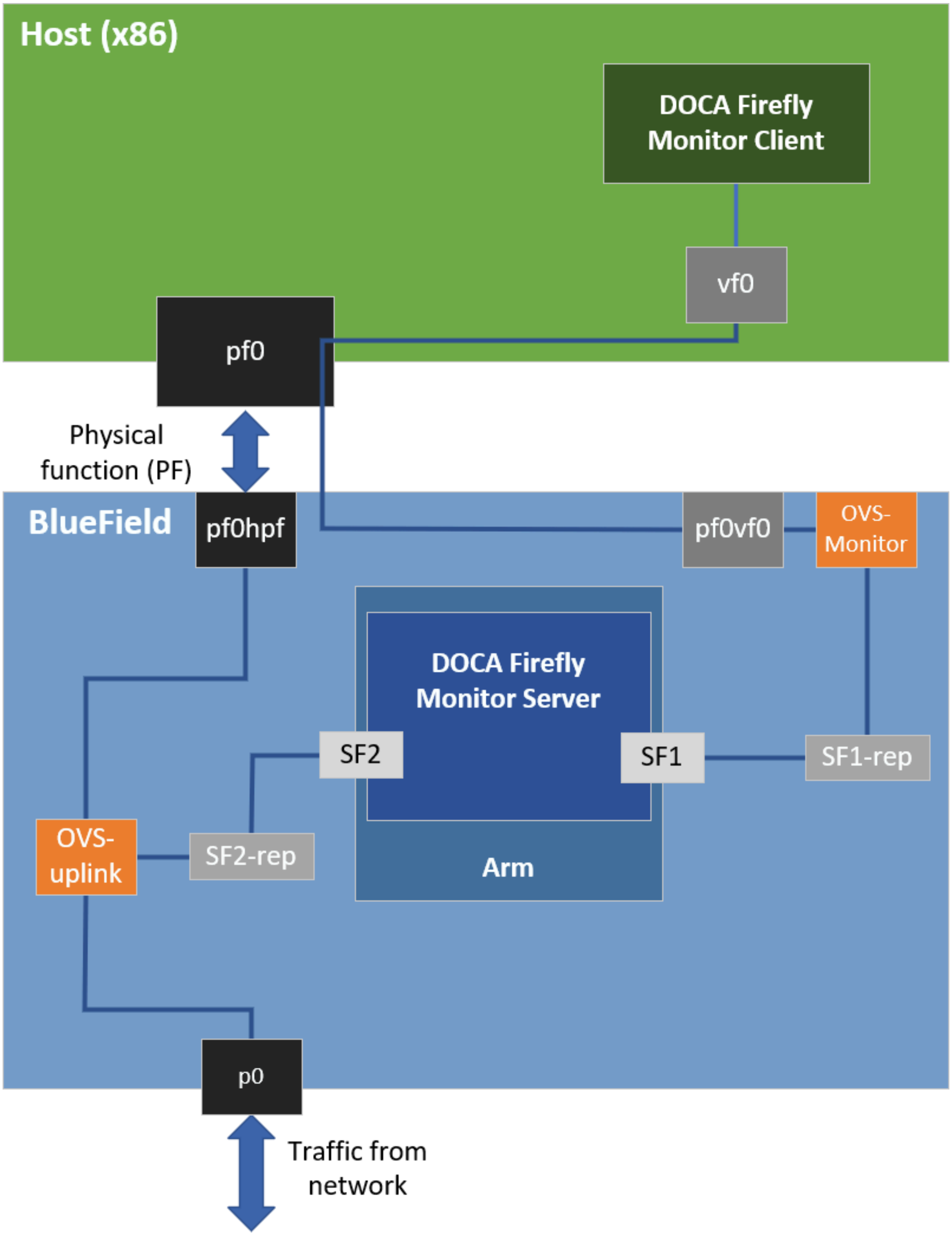
```
...
UtcOffset:                 37
...
ptp_time (TAI):            Thu Sep  7 11:22:50 2023
ptp_time (UTC adjusted):   Thu Sep  7 11:22:13 2023
system_time (UTC):         Thu Sep  7 11:22:13 2023
```

This difference (37 seconds in the above example) is intentional and stems from the amount of leap seconds since epoch. This is indicated by the `UtcOffset` field that is also included in the monitor's report.

Monitor Server

In addition to printing the monitoring data to the container's standard output available through the container logs, the monitoring data is also exposed through a gRPC server that clients can subscribe to. This allows a monitoring client on the host to subscribe to monitor events from the service running on top of the DPU, thus providing better visibility.

The following diagram presents the recommended deployment architecture for connecting the monitoring client (on the host) to the monitor server (on the DPU), based on the NVIDIA DOCA gRPC Infrastructure User Guide.



Based on the above, when activating the monitor feature, the user must provide the IP address to be used by the monitor server:

```
- name: MONITOR_STATE
  value: "<IP address for the monitoring server>"
```

Users can choose to only view the monitoring events through the container logs without connecting to the monitoring server. In this case, it is recommended to configure the local host IP address (127.0.0.1) in the YAML file to avoid exposing it to an unwanted network.

Monitor Client

All the required files for the monitor client are available under the service's dedicated installation directory:

- Linux installations – `/opt/mellanox/doca/services/firefly`
    - Example command line for executing the compiled monitor client from a Linux host:

    ```
    $
    /opt/mellanox/doca/services/firefly/bin/doca_firefly_monit
    -g <ip-address-for-the-monitoring-server>
    ```

    - Example command line for executing the python-based monitor client from a Linux host:

    ```
    $ export
    PYTHONPATH=${PYTHONPATH}:/opt/mellanox/grpc/python3/lib
    $
    /opt/mellanox/doca/services/firefly/bin/doca_firefly_monit
    <ip-address-for-the-monitoring-server>
    ```

    > **Note**

Reference source files and the `.proto` file used for Firefly's monitor are placed under `firefly/src/monitor`.

- Windows installation – `C:\Program Files\Mellanox\DOCA\SDK\firefly`
    - Example command line for executing the python-based monitor client from a Windows host:
        - Installing required pip packages:

```
$ pip3 install grpcio protobuf click
```

        - Running the client:

```
$ C:\Program
Files\Mellanox\DOCA\SDK\firefly\bin\doca_firefly_monit
<ip-address-for-the-monitoring-server>
```

VLAN Tagging

DOCA Firefly natively supports VLAN-tagging-enabled network interfaces.

Separated Mode

The name of the VLAN-enabled network interface should be the one passed through the YAML file in the `PTP_INTERFACE` field.

Embedded Mode

In addition to passing on the VLAN-enabled interface through the YAML as listed in the previous section, the user is also required to configure the network routing within the DPU to support the VLAN tagging:

1. The following example configures a VLAN tag of 10 to the `enp3s0f0s4` interface:

```
$ sudo ip link add link enp3s0f0s4 name enp3s0f0s4.10 type vlan
id 10
$ sudo ip link set up enp3s0f0s4.10
$ sudo ifconfig enp3s0f0s4.10 192.168.104.1 up
```

   In this example, `enp3s0f0s4.10` is the interface to be passed to DOCA Firefly.

2. Additional commands to route the traffic within the DPU:

```
$ sudo ovs-ofctl add-flow uplink
in_port=en3f0pf0sf4,dl_vlan=10,actions=output:p0
$ sudo ovs-ofctl add-flow uplink
in_port=p0,dl_vlan=10,actions=output:en3f0pf0sf4
```

Multiple Interfaces

DOCA Firefly can support multiple network interfaces through the following YAML file syntax:

```
- name: PTP_INTERFACE
  value: "<space (' ') separated list of interface names>"
```

For example:

```
- name: PTP_INTERFACE
  value: "p0 p1"
```

> **Note**
>
> The monitoring feature is supported for multiple interfaces only when the `clientOnly` configuration is enabled.

> **Note**
>
> Automatic mode (`-a`) for `phc2sys` is not supported when working with multiple interfaces. It is recommended to disable `phc2sys` in this mode.

Troubleshooting

When troubleshooting container deployment issues, it is highly recommended to follow the deployment steps and tips in the "Review Container Deployment" section of the NVIDIA DOCA Container Deployment Guide.

To debug the finalized configuration file used by Firefly, users can connect to the container as follows:

1. Open a shell session on the running container using the container ID:

```
sudo crictl exec -it <container-id> /bin/bash
```

2. Once connected the to container, the finalized configuration file can be found under the `/tmp` directory using the same filename as the original configuration file.

> **Info**
>
> More information regarding the configuration files can be found under section "Ensuring and Debugging Correctness of Config File".

Pod is Marked as "Ready" and No Container is Listed

Error

When deploying the container, the pod's STATE is marked as `Ready`, an image is listed, however no container can be seen running:

```
$ sudo crictl pods
POD ID              CREATED             STATE               NAME
NAMESPACE           ATTEMPT             RUNTIME
06bd84c07537e       4 seconds ago       Ready               doca-
firefly-my-dpu                      default             0
(default)

$ sudo crictl images
IMAGE                                 TAG                 IMAGE ID
SIZE
k8s.gcr.io/pause                      3.2
2a060e2e7101d       251kB
nvcr.io/nvidia/doca/doca_firefly      1.1.0-doca2.0.2
134cb22f34611       87.4MB

$ sudo crictl ps
CONTAINER           IMAGE               CREATED             STATE
NAME                     ATTEMPT             POD ID
```

```
POD
```

Solution

In most cases, the container did start, but immediately exited. This could be checked using the following command:

```
$ sudo crictl ps -a
CONTAINER           IMAGE               CREATED             STATE
NAME                     ATTEMPT             POD ID
POD
556bb78281e1d       134cb22f34611       7 seconds ago
Exited              doca-firefly             1
06bd84c07537e       doca-firefly-my-dpu
```

Should the container fail (i.e., state of `Exited`) it is recommended to examine Firefly's main log at `/var/log/doca/firefly/firefly.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the the container's ID:

```
$ sudo crictl logs 556bb78281e1d
Starting DOCA Firefly - Version 1.1.0
...
Requested the following PTP interface: p10
Failed to find interface "p10". Aborting
```

Custom Config File is Not Found

Error

When DOCA Firefly is deployed using a custom configuration file, a deployment error occurs and the following log message appears:

```
...
2023-09-07 14:04:23 - Firefly - Init    - ERROR    - Custom
config file not found: my_file.conf. Aborting
...
```

Solution

Check the custom file name written in the YAML file and make sure that you properly placed the file with that name under the `/etc/firefly/` directory of the DPU.

Profile is Not Supported

Error

When DOCA Firefly is deployed, a deployment error occurs and the following log message appears:

```
...
2023-09-07 14:04:23 - Firefly - Init    - ERROR    - profile
<name> is not supported. Aborting
...
```

Solution

Verify that the profile selected in the YAML file matches one of the supported profiles as listed in the profiles table.

> **Note**
>
> The profile name is case sensitive. The name must be specified in lower-case letters.

PPS Capability is Missing

Error

When DOCA Firefly is deployed and configured to use the `PPS` module, a deployment error occurs and the following log message appears:

```
...
2023-09-07 14:04:23 - Firefly - Init    - INFO     - Starting PPS configuration
2023-09-07 14:04:23 - Firefly - Init    - WARNING  - [-] PPS capability is missing, seems that the card doesn't support PPS
2023-09-07 14:04:23 - Firefly - Init    - INFO     - capabilities:
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   50000000 maximum frequency adjustment (ppb)
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 programmable alarms
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 external time stamp channels
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 programmable periodic signals
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 pulse per second
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 programmable pins
```

```
2023-09-07 14:04:23 - Firefly - Init    - INFO     -   0 cross
timestamping
...
```

Solution

This log indicates that the DPU hardware does not support PPS. However, PTP can still run on this hardware and you should see the line `Running ptp4l` in the container log, indicating that PTP is running successfully.

Timed Out While Polling for Tx Timestamp

Error

When the BlueField is operating in DPU mode, DOCA Firefly gets stuck in a fault loop while waiting to receive the Tx timestamp events:

```
ptp4l[2912.797]: timed out while polling for tx timestamp
ptp4l[2912.797]: increasing tx_timestamp_timeout may correct this
issue, but it is likely caused by a driver bug
ptp4l[2912.797]: port 1 (enp3s0f0s4): send sync failed
ptp4l[2923.528]: timed out while polling for tx timestamp
ptp4l[2923.528]: increasing tx_timestamp_timeout may correct this
issue, but it is likely caused by a driver bug
ptp4l[2923.528]: port 1 (enp3s0f0s4): send sync failed
```

Info

DOCA Firefly has a known gap leading to this error appearing once, after which ptp4l recovers from it. This section only covers the case in which there is a fault loop and no recovery occurs.

Solution

DOCA Firefly's configurations were already adjusted to accommodate for Tx port timestamping. For more information about the reason for this error and for the designed recovery mechanism from it, refer to section "Tx Timestamping Support on DPU Mode".

PTP Profile Default Config Files

Media Profile

```
#
# This config file contains configurations for media &
entertainment alongside
# DOCA Firefly specific adjustments.
#

[global]
domainNumber            127
priority1               128
priority2               127
use_syslog                1
logging_level             6
tx_timestamp_timeout     30
hybrid_e2e                1
dscp_event               46
dscp_general             46
```

```
logAnnounceInterval       -2
announceReceiptTimeout     3
logSyncInterval           -3
logMinDelayReqInterval    -3
delay_mechanism          E2E
network_transport      UDPv4
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval       1
# Required for multiple interfaces support
boundary_clock_jbod        1
```

Default Profile

```
#
# This config file extends linuxptp default.cfg config file with DOCA Firefly
# specific adjustments.
#

[global]
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval                 1
# Required for multiple interfaces support
boundary_clock_jbod                  1
```

Firefly Modules Configuration Options

PTP Monitor

monitor-default.conf

```
#
# Default values for all of Firefly's PTP monitor configuration
values.
#

[global]
# General
report_interval                    1000
# Debugging & Logging
doca_logging_level                    50
```

Configuration Options

- `report_interval` – once in how many milliseconds should the monitor publish a report to all defined output providers (standard output, gRPC clients, etc.)? Default is 1 second (1000 milliseconds).
- `doca_logging_level` – logging level for the module, based on DOCA's logging levels (default is 50):
    - 10=DISABLE
    - 20=CRITICAL
    - 30=ERROR
    - 40=WARNING
    - 50=INFO
    - 60=DEBUG

NVIDIA DOCA Flow Inspector Service Guide

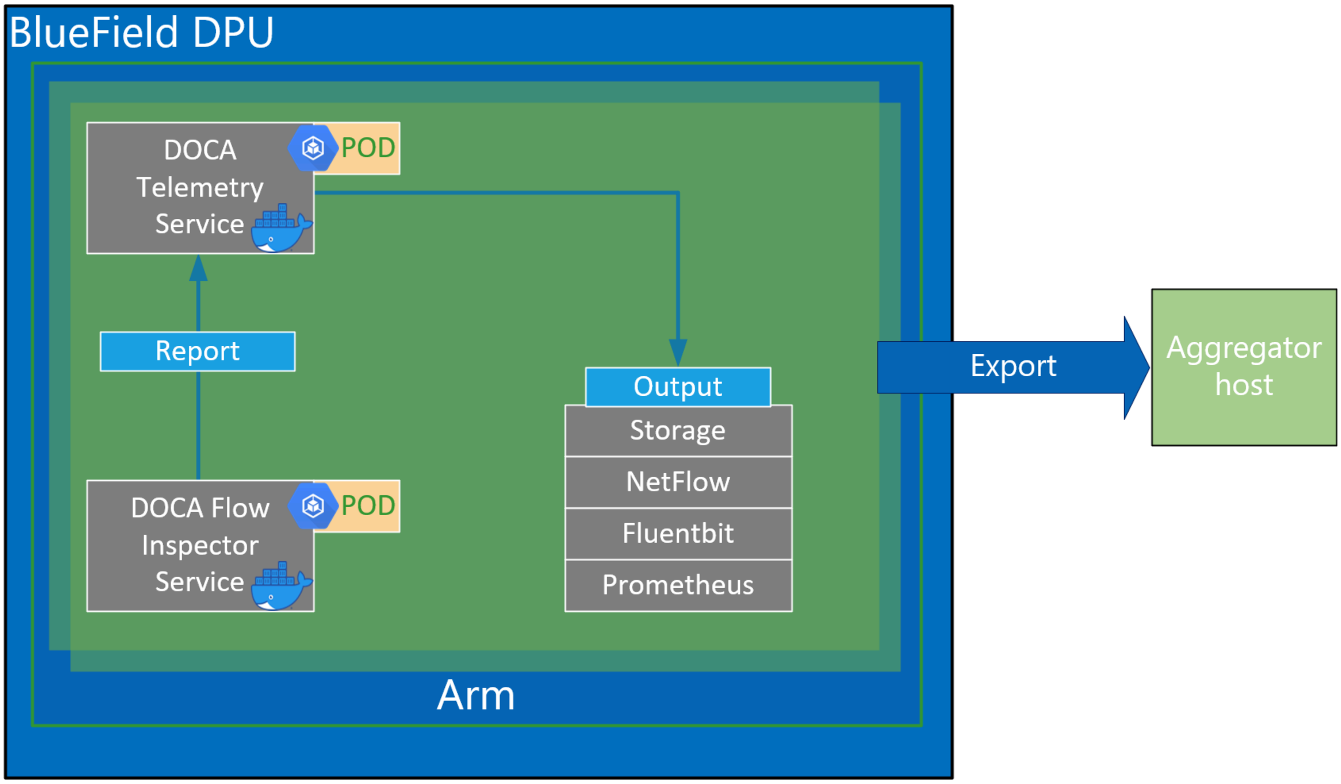
This guide provides instructions on how to use the DOCA Flow Inspector service container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Flow Inspector service enables real-time data monitoring and extraction of telemetry components. These components can be leveraged by various services, including those focused on security, big data, and other purposes.

DOCA Flow Inspector service is linked to DOCA Telemetry Service (DTS). It receives mirrored packets from the user parses the data, and forwards it to the DTS, which aggregates predefined statistics from various providers and sources. The service utilizes the DOCA Telemetry API to communicate with the DTS, while the DPDK infrastructure facilitates packet acquisition at a user-space layer.

DOCA Flow Inspector operates within its dedicated Kubernetes pod on BlueField, aimed at receiving mirrored packets for analysis. The received packets are parsed and transmitted, in a predefined structure, to a telemetry collector that manages the remaining telemetry aspects.



Service Flow

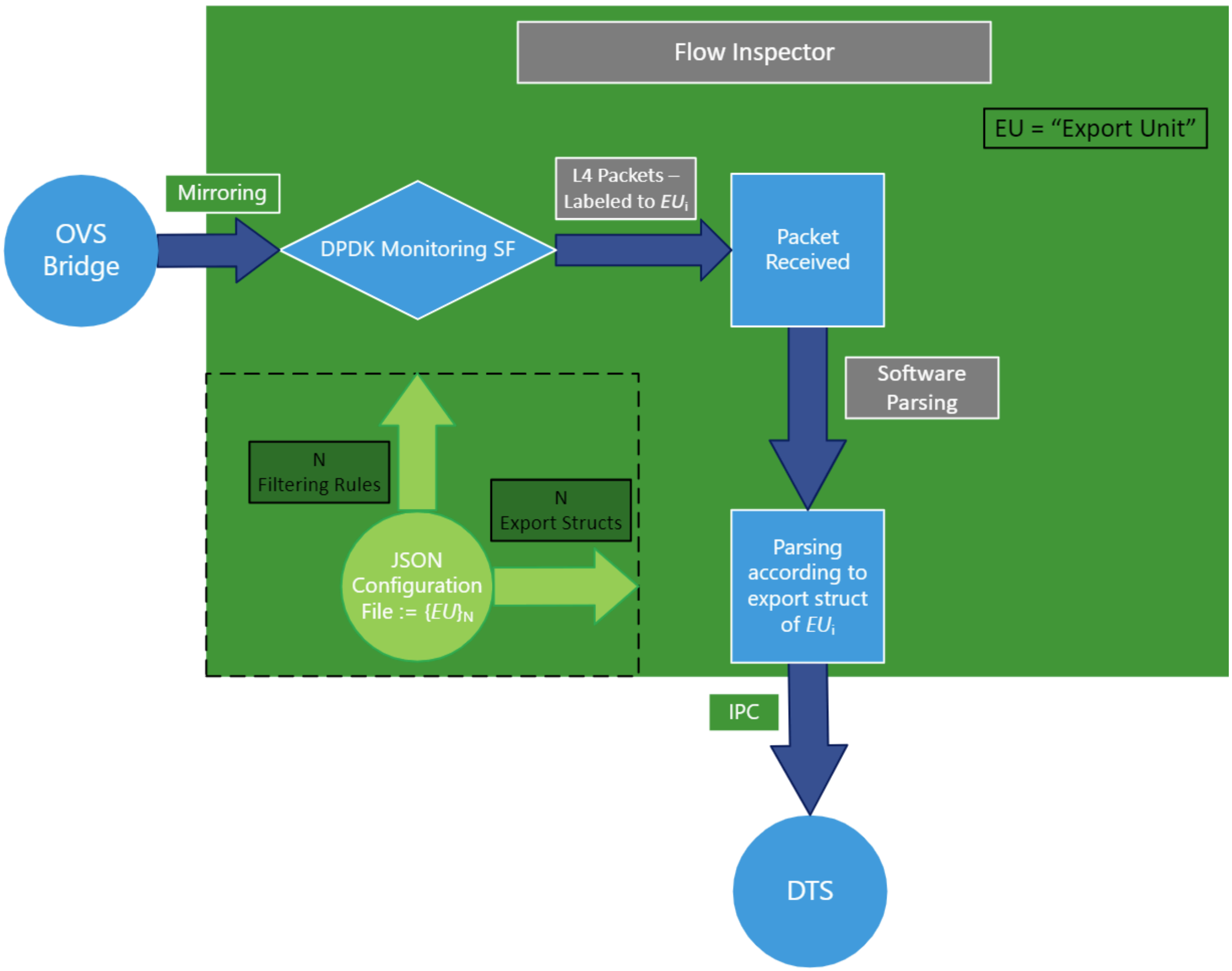
The DOCA Flow Inspector receives a configuration file in a JSON format which includes which of the mirrored packets should be filtered and which information should be sent to DTS for inspection.

The configuration file can include several export units under the "export-units" attribute. Each one is comprised of a "filter" and an "export". Each packet that matches one filter (based on the protocol and ports in the L4 header) is then parsed to the corresponding requested struct defined in the export. That information only is sent for inspection. A packet that does not match any filter is dropped.

In addition, the configuration file could contain FI optional configuration flags, see JSON format and example in the Configuration section.

The service watches for changes in the JSON configuration file in runtime and for any change that reconfigures the service.

The DOCA Flow Inspector runs on top of DPDK to acquire L4. The packets are then filtered and HW-marked with their export unit index. The packets are then parsed according to their export unit and export struct, and then forwarded to the telemetry collector using IPC.



Configuration phase:

1. A JSON file is used as input to configure the export units (i.e., filters and corresponding export structs).
2. The filters are translated to HW rules on the SF (scalable function port) using the DOCA Flow library.
3. The connection to the telemetry collector is initialized and all export structures are registered to DTS.

Inspection phase:

1. Traffic is mirrored to the relevant SF.
2. Ingress traffic is received through the configured SF.

3. Non-L4 traffic and packets that do not match any filter are dropped using hardware rules.
4. Packets matching a filter are marked with the export unit index they match and are passed to the software layer in the Arm cores.
5. Packets are parsed to the desired struct by the index of export unit.
6. The telemetry information is forwarded to the telemetry agent using IPC.
7. Mirrored packets are freed.
8. If the JSON file is changed, run the configuration phase with the updated file.

Requirements

Before deploying the flow inspector container, ensure that the following prerequisites are satisfied:

1. Create the needed files and directories. Folders should be created automatically. Make sure the `.json` file resides inside the folder:

   ```
   $ touch
   /opt/mellanox/doca/services/flow_inspector/bin/flow_inspector_
   ```

   Validate that DTS's configuration folders exist. They should be created automatically when DTS is deployed.

   ```
   $ sudo mkdir -p /opt/mellanox/doca/services/telemetry/config
   $ sudo mkdir -p
   /opt/mellanox/doca/services/telemetry/ipc_sockets
   $ sudo mkdir -p /opt/mellanox/doca/services/telemetry/data
   ```

2. Allocate huge pages as needed by DPDK. This requires root privileges.

```
$ sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-
2048kB/nr_hugepages
```

Or alternatively:

```
$ sudo echo '2048' | sudo tee -a
/sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
$ sudo mkdir /mnt/huge
$ sudo mount -t hugetlbfs nodev /mnt/huge
```

Deploy a scalable function according to NVIDIA BlueField DPU Scalable Function User Guide and mirror packets accordingly using the Open vSwitch command.

For example:

1. Mirror packets from `p0` to `sf4`:

```
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl add-port ovsbr1 p0
$ ovs-vsctl add-port ovsbr1 en3f0pf0sf4
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
            -- --id=@p2 get port p0 \
            -- --id=@m create mirror name=m0 select-dst-
port=@p2 select-src-port=@p2 output-port=@p1 \
            -- set bridge ovsbr1 mirrors=@m
```

2. Mirror packets from `pf0hpf` or `p0` that pass through `sf4`:

```
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl add-port ovsbr1 pf0hpf
$ ovs-vsctl add-port ovsbr1 p0
```

```
$ ovs-vsctl add-port ovsbr1 en3f0pf0sf4
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
            -- --id=@p2 get port pf0hpf \
            -- --id=@m create mirror name=m0 select-dst-
port=@p2 select-src-port=@p2 output-port=@p1 \
            -- set bridge ovsbr1 mirrors=@m
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
            -- --id=@p2 get port p0 \
            -- --id=@m create mirror name=m0 select-dst-
port=@p2 select-src-port=@p2 output-port=@p1 \
            -- set bridge ovsbr1 mirrors=@m
```

The output of last command (creating the mirror) should output a sequence of letters and numbers similar to the following:

```
0d248ca8-66af-427c-b600-af1e286056e1
```

Note

The designated SF must be created as a trusted function. Additional details can be found in the NVIDIA BlueField DPU Scalable Function User Guide.

Service Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to NVIDIA DOCA Container Deployment Guide.

DTS is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's container page.

> **Note**
>
> The order of running DTS and DOCA Flow Inspector is important. You must launch DTS, wait a few seconds, and then launch DOCA Flow Inspector.

Configuration

JSON Input

The DOCA Flow Inspector configuration file should be placed under `/opt/mellanox/doca/services/flow_inspector/bin/<json_file_name>.json` and be built in the following format:

```
{
        /* Optional param, time period to check for changes in JSON config file (in seconds) and flush telemetry buffer if enabled (default is 60 seconds) */
        "config-sample-rate" : <time>,

        /* Optional param, telemetry buffer size in bytes (default is 60KB) */
        "telemetry-buffer-size" : <size>,

        /* Optional param, enable periodic telemetry buffer flush and defining the period time (in seconds) */
        "telemetry-flush-rate" : <numeric value in seconds>,

        /* Mandatory param, Flow Inspector export units */
        "export-units" :
        [

                /* Export Unit 0 */
                {
                        "filter" :
```

```
                        {      "protocols": [<L4 protocols separated
by comma>], # What L4 protocols are allowed
                              "ports":
                                       [
                                                       [<source
port>, <destination port>],
                                [<source ports range>,
<destination ports range>],
                                <... more pairs of source, dest
ports>
                                ]
                        },
                        "export":
                        {
                        "fields": [<fields to be part of export
struct, separated by comma>] # the Telemetry event will contain
these fields.

                        }
                },
        <... More Export Units>
        ]
}
```

Export Unit Attributes

Allowed protocols:

- `"TCP"`
- `"UDP"`

Port range:

- It is possible to insert a range of ports for both source and destination
- Range should include borders `[start_port-end_port]`

Allowed ports:

- All ports in range `0`-`65535` as a string
- Or `*` to indicate any ports

Allowed fields in export struct:

- `timestamp` – timestamp indicating when it was received by the service
- `host_ip` – the IP of the host running the service
- `src_mac` – source MAC address
- `dst_mac` – destination MAC address
- `src_ip` – source IP
- `dst_ip` – destination IP
- `protocol` – L4 protocol
- `src_port` – source port
- `dst_port` – destination port
- `flags` – additional flags (relevant to TCP only)
- `data_len` – data payload length
- `data_short` – short version of data (payload sliced to first 64 bytes)
- `data_medium` – medium version of data (payload sliced to first 1500 bytes)
- `data_long` – long version of data (payload sliced to first 9*1024 bytes)

JSON example:

```
{
```

```
        /* Optional param, time period to check for changes in JSON
config file (in seconds) and flush telemetry buffer if enabled
(default is 60 seconds) */
        "config-sample-rate": 30,

        /* Optional param, telemetry maximum buffer size in bytes
*/
        "telemetry-buffer-size": 70000,

        /* Optional param, enable periodic telemetry buffer flush
and defining the period time (in seconds) */
        "telemetry-flush-rate": 1.5,

   /* Mandatory param, Flow Inspector export units */
        "export-units":
        [

                /* Export Unit 0 */
                {
                        "filter":
                        {
                                "protocols": ["tcp", "udp"],
                                "ports":
                                        [
                                                ["*","433-460"],
                                                ["20480","28341"],
                                                ["28341","20480"],
                                                ["68", "67"],
                                                ["67", "68"]
                                        ]
                        },
                        "export":
                        {
                                "fields": ["timestamp", "host_ip", "src_mac",
"dst_mac", "src_ip", "dst_ip", "protocol", "src_port",
```

```
                                            "dst_port",  "flags",  "data_len",
"data_long"]
                        }
                },

                /* Export Unit 1 */
                {
                        "filter":
                        {
                                "protocols":  ["tcp"],
                                "ports":
                                        [
                                                ["5-10", "422"],
                                                ["80", "80"]
                                        ]
                        },
                        "export":
                        {
                                "fields":  ["timestamp", "dst_ip",  "host_ip",
"data_len",  "flags",  "data_medium"]
                        }
                }
        ]
}
```

> **Note**
>
> If a packet header contains L4 ports or L4 protocol which are not specified in any filter, they are filtered out.

Yaml File

The `.yaml` file downloaded from NGC can be easily edited according to your needs.

```
env:
  # Set according to the local setup
  - name: SF_NUM_1
    value: "2"   # Additional EAL flags, if needed
  - name: EAL_FLAGS
    value: ""   # Service-Specific command line arguments
  - name: SERVICE_ARGS
    value: "--policy /flow_inspector/flow_inspector_cfg.json -l 60"
```

- The `SF_NUM_1` value can be changed according to the SF used in the OVS configuration and can be found using the command in NVIDIA BlueField DPU Scalable Function User Guide.
- The `EAL_FLAGS` value must be changed according to the DPDK flags required when running the container.
- The `SERVICE_ARGS` are the runtime arguments received by the service:
  - `-l`, `--log-level <value>` – sets the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  - `-p`, `--policy <json_path>` – sets the JSON path inside the container

Verifying Output

Enabling write to data in the DTS allows debugging the validity of the DOCA Flow Inspector.

To allow DTS to write locally, uncomment the following line in `/opt/mellanox/doca/services/telemetry/config/dts_config.ini`:

```
#output=/data
```

> **Note**
>
> Any changes in `dts_config.ini` necessitate restarting the pod for the new settings to apply.

The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example:

> **Note**
>
> Requires installing the `tree` runtime utility (`apt install tree`).

```
$ tree /opt/mellanox/doca/services/telemetry/data/
/opt/mellanox/doca/services/telemetry/data/
    {year}
        {mmdd}
             {hash}
                  {source_id}
                      {source_tag}{timestamp}.bin
                  {another_source_id}
                       {another_source_tag}{timestamp}.bin
    schema
        schema_{MD5_digest}.json
```

New binary files appear when:

- The service starts
- When the binary file's max age/size restriction is reached
- When JSON file is changed and new schemas of telemetry are created
- An hour passes

If no schema or no data folders are present, refer to the Troubleshooting section in NVIDIA DOCA Telemetry Service Guide.

Note

`source_id` is usually set to the machine hostname. `source_tag` is a line describing the collected counters, and it is often set as the provider's name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```
crictl exec -it <Container-ID>
/opt/mellanox/collectx/bin/clx_read -s /data/schema
/data/path/to/datafile.bin
```

The data written locally should be shown in the following format assuming a packet matching Export Unit 1 from the example has arrived:

```
{
    "timestamp": 1656427771076130,
    "host_ip": "10.237.69.238",
    "src_ip": "11.7.62.4",
    "dst_ip": "11.7.62.5",
```

```
        "data_len": 1152,
        "data_short": "Hello World"
}
```

Troubleshooting

When troubleshooting container deployment issues, it is highly recommended to follow the deployment steps and tips in the "Review Container Deployment" section of the NVIDIA DOCA Container Deployment Guide.

Pod is Marked as "Ready" and No Container is Listed

Error

When deploying the container, the pod's STATE is marked as `Ready`, an image is listed, however no container can be seen running:

```
$ sudo crictl pods
POD ID                  CREATED                 STATE                   NAME
NAMESPACE               ATTEMPT                 RUNTIME
3162b71e67677               4 seconds ago           Ready
doca-flow-inspector-my-dpu                                  default
0                       (default)

$ sudo crictl images
IMAGE                                                  TAG
IMAGE ID                SIZE
k8s.gcr.io/pause                                       3.2
    2a060e2e7101d                   487kB
nvcr.io/nvidia/doca/doca_flow_inspector   1.1.0-doca2.0.2
2af1e539eb7ab           86.8MB

$ sudo crictl ps
```

```
CONTAINER           IMAGE               CREATED             STATE
NAME                     ATTEMPT             POD ID
POD
```

Solution

In most cases, the container did start, but immediately exited. This could be checked using the following command:

```
$ sudo crictl ps -a
CONTAINER           IMAGE               CREATED             STATE
NAME                     ATTEMPT             POD ID
POD
556bb78281e1d       2af1e539eb7ab       6 seconds ago
Exited              doca-flow-inspector      1
3162b71e67677       doca-flow-inspector-my-dpu
```

Should the container fail (i.e., state of `Exited`), it is recommended to examine the Flow Inspector's main log at `/var/log/doca/flow_inspector/flow_inspector_fi_dev.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the container's ID:

```
$ sudo crictl logs 556bb78281e1d
...
2023-10-04 11:42:55 - flow_inspector - FI     - ERROR    - JSON
file was not found <config-file-path>.
```

Pod is Not Listed

Error

When placing the container's YAML file in the Kubelet's input folder, the service pod is not listed in the list of pods:

```
$ sudo crictl pods
POD ID              CREATED             STATE               NAME
NAMESPACE           ATTEMPT             RUNTIME
```

Solution

In most cases, the pod does not start due to the absence of the requested hugepages. This can be verified using the following command:

```
$ sudo journalctl -u kubelet -e. . .
Oct 04 12:12:19 <my-dpu> kubelet[2442376]: I1004 12:12:19.905064
2442376 predicate.go:103] "Failed to admit pod, unexpected error while attempting to
recover from admission failure" pod="default/doca-flow-inspector-<my-dpu>" err="preemption: error
finding a set of pods to preempt: no set of running pods found to reclaim resources: [(res: hugepages-2Mi,
q: 104563999874), ]"
```

NVIDIA DOCA HBN Service Guide

This guide provides instructions on how to use the DOCA HBN Service container on top of NVIDIA® BlueField® DPU.

Release Notes

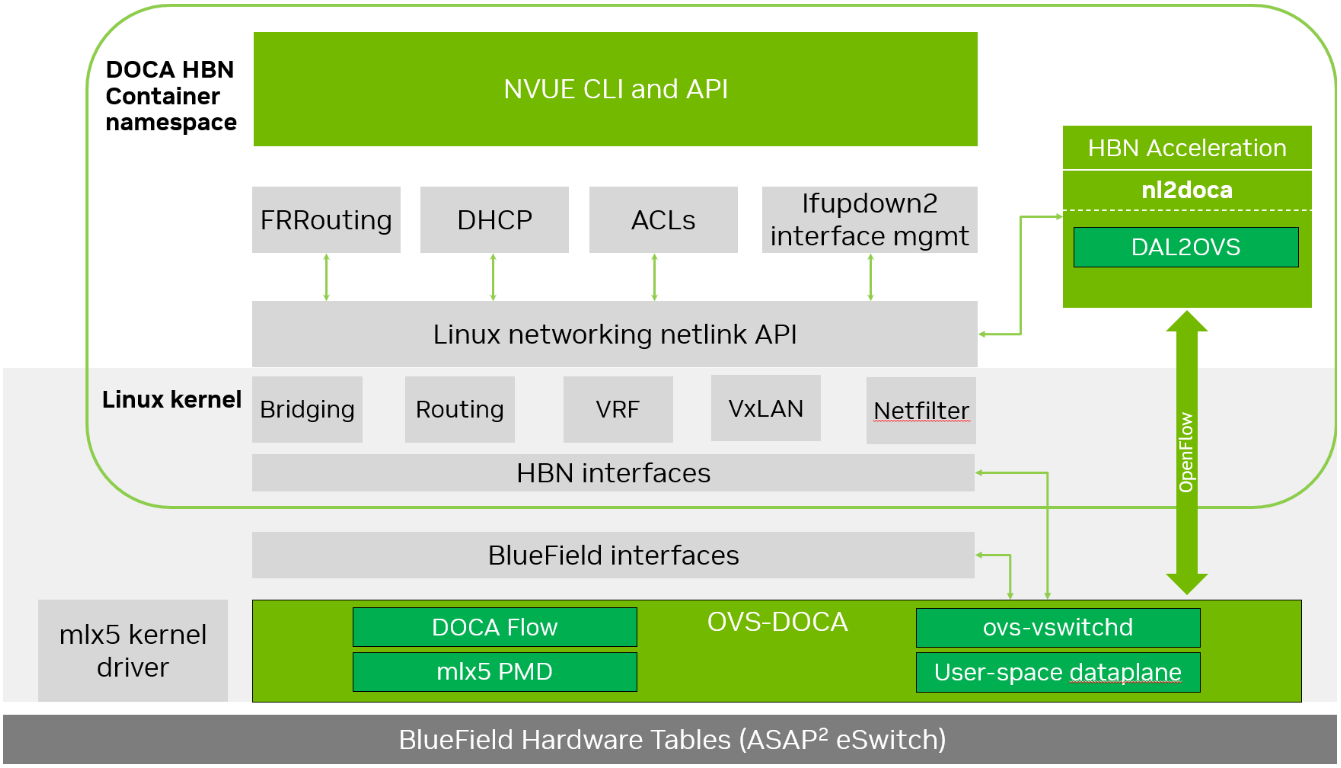
For the release notes of HBN 2.0.0, please refer to "HBN Service Release Notes".

HBN Overview

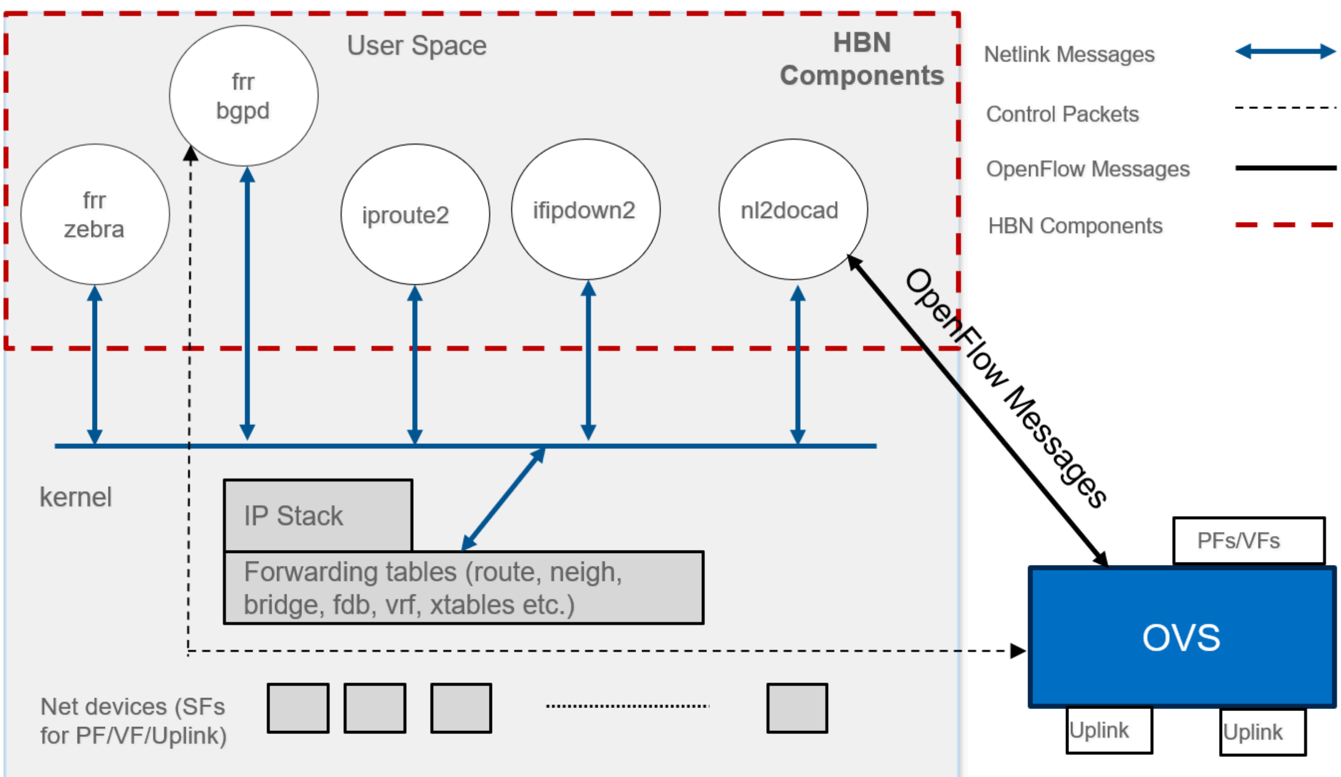
Host-based networking (HBN) is a DOCA service that enables the network architect to design a network purely on L3 protocols, enabling routing to run on the server-side of the network by using the DPU as a BGP router. The EVPN extension of BGP, supported by HBN, extends the L3 underlay network to multi-tenant environments with overlay L2 and L3 isolated networks.

The HBN solution packages a set of network functions inside a container which, itself, is packaged as a service pod to be run on the DPU. At the core of HBN is the Linux networking DPU acceleration driver. Netlink to DOCA daemon, or nl2docad, implements the DPU acceleration driver. nl2docad seamlessly accelerates Linux networking using DPU hardware programming APIs.

The driver mirrors the Linux kernel routing and bridging tables into the DPU hardware by discovering the configured Linux networking objects using the Linux Netlink API. Dynamic network flows, as learned by the Linux kernel networking stack, are also programmed by the driver into DPU hardware by listening to Linux kernel networking events.



The following diagram captures an overview of HBN and the interactions between various components of HBN.



- ifupdown2 is the interface manager which pushes all the interface related states to kernel
- The routing stack is implemented in FRR and pushes all the control states (EVPN MACs and routes) to kernel via netlink
- Kernel maintains the whole network state and relays the information using netlink. The kernel is also involved in the punt path and handling traffic that does not match any rules in the eSwitch.
- nl2docad listens for the network state via netlink and invokes the DOCA interface to accelerate the flows in the DPU hardware tables. nl2docad also offloads these flows to eSwitch.

Service Deployment

Preparing DPU for HBN Deployment

HBN requires service function chaining (SFC) to be activated on the DPU before running the HBN service container. SFC allows for additional services/containers to be chained to HBN and provides additional data manipulation capabilities.

The following subsections provide additional information about SFC and instructions on enabling it during DPU BFB installation.

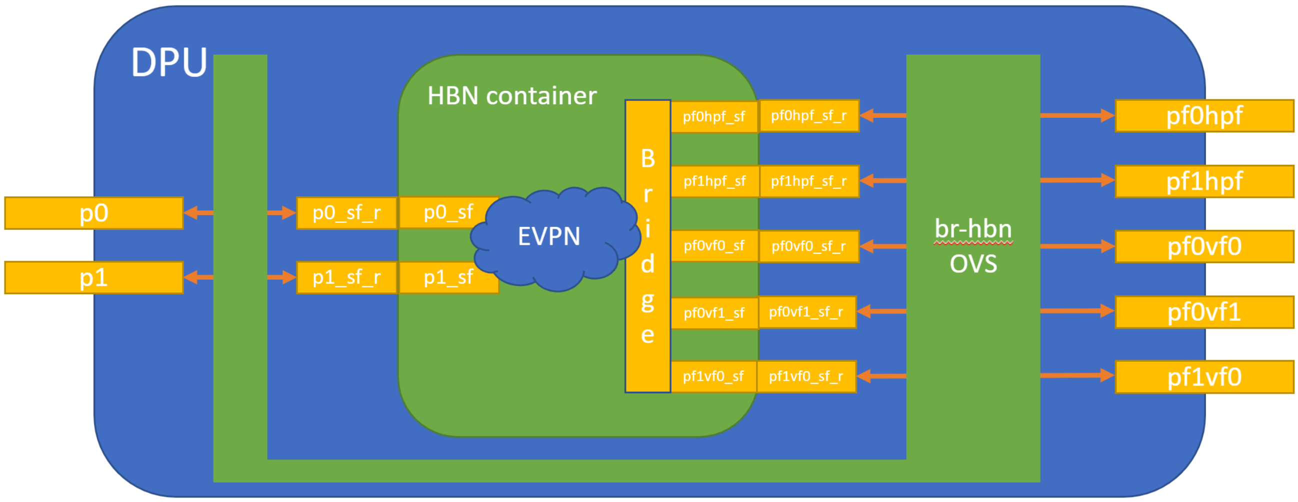
Service Function Chaining

The diagram below shows the fully detailed default configuration for HBN with Service Function Chaining (SFC).

In this setup, the HBN container is configured to use sub-function ports (SFs) instead of the actual uplinks, PFs and VFs. To illustrate, for example:

- Uplinks – use `p0_sf` instead of `p0`
- PF – use `pf0hpf_sf` instead of `pf0hpf`
- VF – use `pf0vf0_sf` instead of `pf0vf0`

The indirection layer between the SF and the actual ports is managed via a `br-hbn` OVS bridge automatically configured when the BFB image is installed on the DPU with HBN enabled. This indirection layer allows other services to be chained to existing SFs and provide additional functionality to transit traffic.



Enabling SFC for HBN Deployment

Deployment from BFB

DPU installation should follow the *NVIDIA DOCA Installation Guide for Linux*.

1. Make sure link type is set to ETH in step 5 of the "Installing Software on Host" section in the *NVIDIA DOCA Installation Guide for Linux*.
2. Add the following parameters to the `bf.cfg` configuration file:

```
ENABLE_SFC_HBN=yes
NUM_VFs_PHYS_PORT0=12  # <num VFs supported by HBN on Physical Port 0> (valid range: 0-127) Default 14
NUM_VFs_PHYS_PORT1=2   # <num VFs supported by HBN on Physical Port 1> (valid range: 0-127) Default 0
```

3. Then run:

```
# bfb-install -c bf.cfg -r rshim0 -b <BFB-image>
```

Deployment from PXE Boot

To enable HBN SFC using a PXE installation environment with BFB content, use the following configuration for PXE:

```
bfnet=<IFNAME>:<IPADDR>:<NETMASK> or <IFNAME>:dhcp
bfks=<URL of the kickstart script>
```

The kickstart script (bash) should include the following lines:

```
cat >> /etc/bf.cfg << EOF
ENABLE_SFC_HBN=yes
NUM_VFs_PHYS_PORT0=12 # <num VFs supported by HBN on Physical
Port 0> (valid range: 0-127) Default 14
NUM_VFs_PHYS_PORT1=2  # <num VFs supported by HBN on Physical
Port 1> (valid range: 0-127) Default 0
EOF
```

`/etc/bf.cfg` is sourced by the BFB `install.sh` script.

Note

It is recommended to verify the accuracy of the DPU's clock post-installation. This can be done using the following command:

```
$ date
```

Please refer to the known issues listed in the "NVIDIA DOCA Release Notes" for more information.

HBN Service Container Deployment

HBN service is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's container page. Make sure to follow the instructions in the NGC page to verify that the container is running properly.

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to NVIDIA DOCA Container Deployment Guide.

HBN Default Deployment Configuration

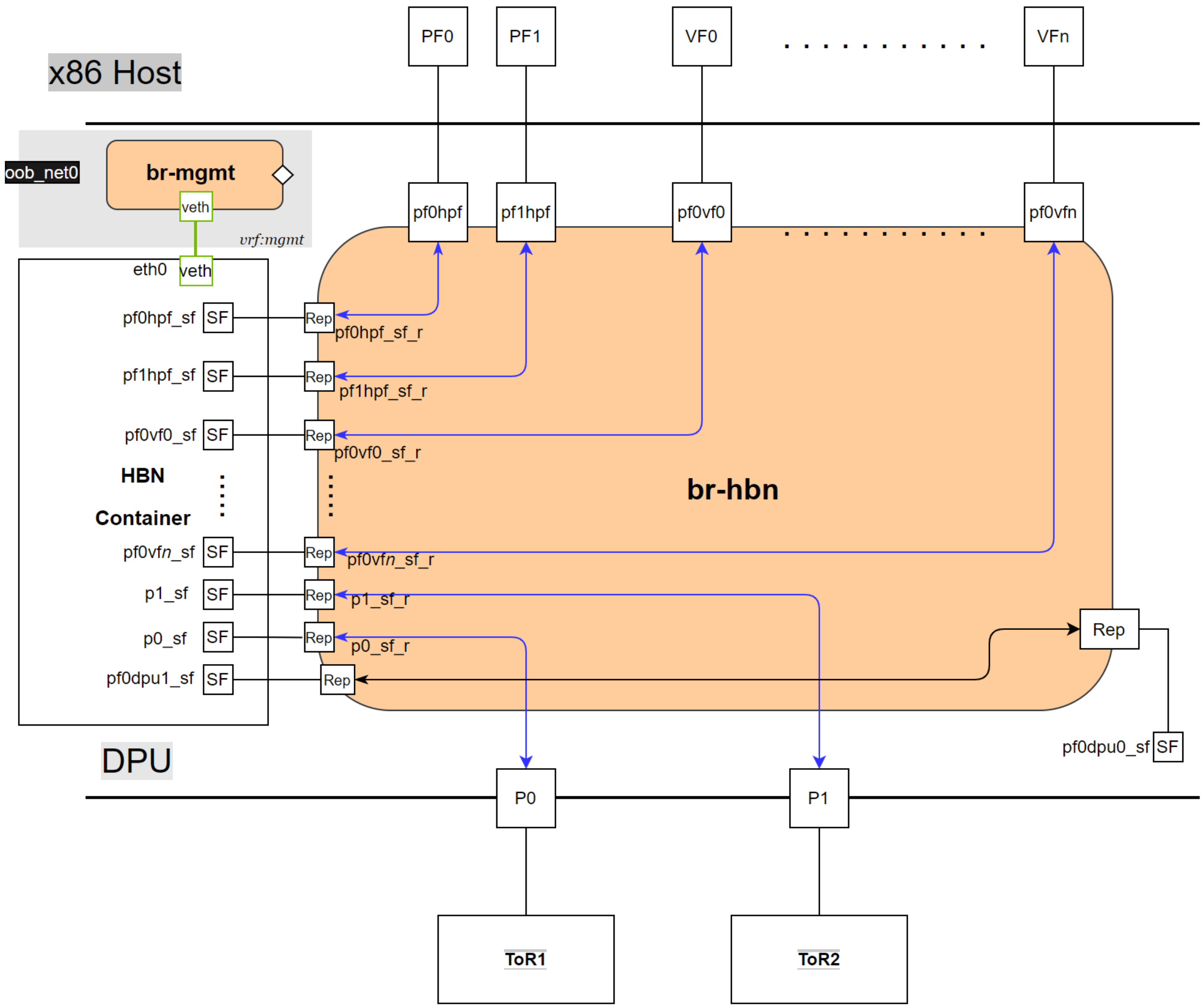
The HBN service comes with four types of configurable interfaces:

- Two uplinks (`p0_sf`, `p1_sf`)
- Two PF port representors (`pf0hpf_sf`, `pf1hpf_sf`)
- User-defined number of VFs (i.e., `pf0vf0_sf`, `pf0vf1_sf`, ..., `pf1vf0_sf`, `pf1vf1_sf`, ...)
- One interface to connect to services running on the DPU, outside of the HBN container (`pf0dpu1_sf`)

The `*_sf` suffix indicates that these are sub-functions and are different from the physical uplinks (i.e., PFs, VFs). They can be viewed as virtual interfaces from a virtualized DPU.

Each of these interfaces is connected outside the HBN container to the corresponding physical interface, see section "Service Function Chaining" (SFC) for more details.

The HBN container runs as an isolated namespace and does not see any interfaces outside the container (`oob_net0`, real uplinks and PFs, `*_sf_r` representors).



`pf0dpu1_sf` is a special interface for HBN to connect to services running on the DPU. Its counterpart `pf0dpu0_sf` is located outside the HBN container. See section "Connecting to Services on DPU" for deployment considerations when using the `dpu1_sf` interface in HBN.

`eth0` is equivalent to the `oob_net0` interface in the HBN container. It is part of the management VRF of the container. It is not configurable via NVUE and does not need any configuration from the user. See section "MGMT VRF in HBN Container" for more details on this interface and the management VRF.

HBN Deployment Considerations

SF Interface State Tracking

When HBN is deployed with SFC, the interface state of the following network devices is propagated to their corresponding SFs:

- Uplinks – `p0`, `p1`
- PFs – `pf0hpf`, `pf1hpf`
- VFs – `pf0vfX`, `pf1vfX` where `X` is the VF number

For example, if the p0 uplink cable gets disconnected:

- `p0` transitions to DOWN state with `NO-CARRIER` (default behavior on Linux); and
- `p0` state is propagated to `p0_sf` whose state also becomes DOWN with NO-CARRIER

After `p0` connection is reestablished:

- `p0` transitions to UP state; and
- `p0` state is propagated to `p0_sf` whose state becomes UP

Interface state propagation only happens in the uplink/PF/VF-to-SF direction.

A daemon called `sfc-state-propagation` runs on the DPU, outside of the HBN container, to sync the state. The daemon listens to netlink notifications for interfaces and transfers the state to SFs.

SF Interface MTU

In the HBN container, all the interfaces MTU are set to 9216 by default. MTU of specific interfaces can be overwritten using flat-files configuration or NVUE.

On the DPU side (i.e., outside of the HBN container), the MTU of the uplinks, PFs and VFs interfaces are also set to 9216. This can be changed by modifying `/etc/systemd/network/30-hbn-mtu.network` or by adding a new configuration file in the `/etc/systemd/network` for specific directories.

To reload this configuration, execute `systemctl restart systemd-networkd`.

Connecting to Services on DPU

`pf0dpu1_sf` can be used by HBN to connect to services running on the DPU. Its counterpart, `pf0dpu0_sf`, is located outside the HBN container.

Traffic between the DPU and the outside world is not hardware-accelerated in the HBN container when using a native L3 connection over `pf0dpu0_sf`/`pf0dpu1_sf`. To get hardware-acceleration, configure `pf0dpu1_sf` in the HBN container with `bridge-access` over an SVI.

Disabling DPU Uplinks

The uplink ports must be always kept administratively up for proper operation of HBN. Otherwise, the NVIDIA® ConnectX® firmware would bring down the corresponding representor port which would cause data forwarding to stop.

> **Note**
>
> Change in operational status of uplink (e.g., carrier down) would result in traffic being switched to the other uplink.

When using ECMP failover on the two uplink SFs, locally disabling one uplink does not result in traffic switching to the second uplink. Disabling local link in this case means to set one uplink admin DOWN directly on the DPU.

To test ECMP failover scenarios correctly, the uplink must be disabled from its remote counterpart (i.e., execute admin DOWN on the remote system's link which is connected to the uplink).

Configuration

General Network Configuration

Flat Files Configuration

Add network interfaces and FRR configuration files to the DPU to achieve the desired configuration:

- `/etc/network/interfaces`

  > **Note**
  >
  > Refer to NVIDIA® Cumulus® Linux documentation for more information.

- `/etc/frr/frr.conf`; `/etc/frr/daemons`

  > **Note**
  >
  > Refer to NVIDIA® Cumulus® Linux documentation for more information.

NVUE Configuration

This section assumes familiarity with NVIDIA user experience (NVUE) Cumulus Linux documentation. The following subsections, only expand on DPU-specific aspects of NVUE.

NVUE Service

HBN installs NVUE by default and enables NVUE service at boot.

NVUE REST API

HBN enables REST API by default.

Users may run the cURL commands from the command line. Use the default HBN username `nvidia` and password `nvidia`.

To change the default password of the `nvidia` user or add additional users for NVUE access, refer to section "NVUE User Credentials".

REST API example:

```
curl -u 'nvidia:nvidia' --insecure
https://10.188.108.58:8765/nvue_v1/interface/p0
{
  "ip": {
    "address": {
      "30.0.0.1/24": {}
    }
  },
  "link": {
    "auto-negotiate": "on",
    "duplex": "full",
    "fec": "auto",
    "mac": "b8:ce:f6:a8:83:9a",
    "mtu": 9216,
    "speed": "100G",
    "state": {
      "up": {}
    },
    "stats": {
      "carrier-transitions": 13,
      "in-bytes": 0,
```

```
        "in-drops": 0,
        "in-errors": 0,
        "in-pkts": 0,
        "out-bytes": 14111,
        "out-drops": 0,
        "out-errors": 0,
        "out-pkts": 161
      }
    },
    "pluggable": {
      "identifier": "QSFP28",
      "vendor-name": "Mellanox",
      "vendor-pn": "MCP1600-C00AE30N",
      "vendor-rev": "A4",
      "vendor-sn": "MT2105VB02844"
    },
    "type": "swp"
  }
```

> **Note**
>
> For information about using the NVUE REST API, refer to the NVUE API documentation .

NVUE CLI

For information about using the NVUE CLI, refer to the NVUE CLI documentation

NVUE Startup Configuration File

When the network configuration is saved using NVUE, HBN writes the configuration to the `/etc/nvue.d/startup.yaml` file.

Startup configuration is applied by following the supervisor daemon at boot time. `nvued-startup` will appear in `EXITED` state after applying the startup configuration.

```
# supervisorctl status nvued-startup
nvued-startup                    EXITED    Apr 17 10:04 AM
```

Note

`nv config apply startup` applies the yaml configuration saved at `/etc/nvue.d/`.

Note

`nv config save` saves the running configuration to `/etc/nvue.d/startup.yaml`.

NVUE User Credentials

The preconfigured default user credentials are as follows:

Username	nvidia
Password	nvidia

NVUE user credentials can be added post installation. This functionality is enabled by the HBN startup script by using the `--username` and `--password` script switches. For

example:

```
./hbn-dpu-setup.sh -u newuser -p newpassword
```

After executing this script, respawn the container or start the `decrypt-user-add` script:

```
supervisorctl start decrypt-user-add
decrypt-user-add: started
```

The script creates a user on the HBN container:

```
cat /etc/passwd | grep newuser
newuser:x:1001:1001::/home/newuser:/bin/bash
```

NVUE Interface Classification

Interface	Interface Type	NVUE Type
`p0_sf`	Uplink representor	swp
`p1_sf`	Uplink representor	swp
`lo`	Loopback	loopback
`pf0hpf_sf`	Host representor	swp
`pf1hpf_sf`	Host representor	swp
`pf0vfx_sf` (where `x` is 0-255)	VF representor	swp
`pf1vfx_sf` (where `x` is 0-255)	VF representor	swp

Configuration Persistence

The following directories are mounted from the host DPU to the HBN container and are persistent across HBN restarts and DPU reboots:

Host DPU Mount Point	HBN Container Mount Point
Configuration Files Mount Pints	
`/var/lib/hbn/etc/network/`	`/etc/network/`
`/var/lib/hbn/etc/frr/`	`/etc/frr/`
`/var/lib/hbn/etc/nvue.d/`	`/etc/nvue.d/`
`/var/lib/hbn/etc/supervisor/conf.d/`	`/etc/supervisor/conf.d/`
`/var/lib/hbn/var/lib/nvue/`	`/var/lib/nvue/`
Support and Log Files Mount Points	
`/var/lib/hbn/var/support/`	`/var/support/`
`/var/log/doca/hbn/`	`/var/log/hbn/`

SR-IOV Support

Creating VFs on Host Server

The first step to use SR-IOV is to create VFs on the host server. VFs can be created using the following command:

```
echo N > /sys/class/net/<host-rep>/device/sriov_numvfs
```

Where:

- `<host-rep>` is one of the two host representors (e.g., `ens1f0` or `ens1f1`)
- $0 \leqslant N \leqslant 16$ is the desired total number of VFs

- Set `N`=0 to delete all the VFs on 0⩽N⩽16
- `N`=16 is the maximum number of VFs supported on HBN across all representors

Automatic Creation of VF Representors on DPU

VFs created on the host must have corresponding SF representors on the DPU side. For example:

- `ens1f0vf0` is the first VF from the first host representor; this interface is created on the host server
- `pf0vf0` is the corresponding VF representor to `ens1f0vf0`; this interface is on the DPU and automatically created at the same time as `ens1f0vf0` is created
- `pf0vf0_sf` is the corresponding SF for `pf0vf0` which is used by HBN

The creation of the SF representor for VFs is done ahead of time when installing the BFB, see section "Enabling SFC for HBN Deployment" to see how to select how many SFs to create ahead of time.

The SF representors for VFs (i.e., `pfXvfY`) are pre-mapped to work with the corresponding VF representors when these are created with the command from section "Creating VFs on Host Server".

Management VRF

Two management VRFs are setup for HBN with SFC:

- The first management VRF is outside the HBN container on the DPU. This VRF provides separation between out-of-band (OOB) traffic (via `oob_net0` or `tmfifo_net0`) and data-plane traffic via uplinks and PFs.
- The second management VRF is inside the HBN container and provides similar separation. The OOB traffic (via `eth0`) is isolated from the traffic via the `*_sf` interfaces.

MGMT VRF on Host DPU

The management (mgmt) VRF is enabled by default when the DPU is deployed with SFC (see section "Enabling SFC for HBN Deployment"). The mgmt VRF provides separation between the OOB management network and the in-band data plane network.

The uplinks and PFs/VFs use the default routing table while the `oob_net0` (OOB Ethernet port) and the `tmifo_net0` netdevices use the mgmt VRF to route their packets.

When logging in either via SSH or the console, the shell is by default in mgmt VRF context. This is indicated by a mgmt added to the shell prompt:

```
root@bf2:mgmt:/home/ubuntu#
root@bf2:mgmt:/home/ubuntu# ip vrf identify
mgmt.
```

When logging into the HBN container with `crictl`, the HBN shell will be in the default VRF. Users must switch to MGMT VRF manually if OOB access is required. Use `ip vrf exec` to do so.

```
root@bf2:mgmt:/home/ubuntu# ip vrf exec mgmt bash
```

The user must run `ip vrf exec mgmt` to perform operations requiring OOB access (e.g., apt-get update).

Network devices belonging to the mgmt VRF can be listed with the `vrf` utility:

```
root@bf2:mgmt:/home/ubuntu# vrf link list

VRF: mgmt
--------------------
tmfifo_net0      UP              00:1a:ca:ff:ff:03
<BROADCAST,MULTICAST,UP,LOWER_UP>
```

```
oob_net0         UP              08:c0:eb:c0:5a:32
<BROADCAST,MULTICAST,UP,LOWER_UP>

root@bf2:mgmt:/home/ubuntu# vrf help
vrf <OPTS>

VRF domains:
    vrf list

Links associated with VRF domains:
    vrf link list [<vrf-name>]

Tasks and VRF domain asociation:
    vrf task exec <vrf-name> <command>
    vrf task list [<vrf-name>]
    vrf task identify <pid>

    NOTE: This command affects only AF_INET and AF_INET6 sockets
opened by the
          command that gets exec'ed. Specifically, it has *no*
impact on netlink
          sockets (e.g., ip command).
```

To show the routing table for the default VRF, run:

```
root@bf2:mgmt:/home/ubuntu# ip route show
```

To show the routing table for the mgmt VRF, run:

```
root@bf2:mgmt:/home/ubuntu# ip route show vrf mgmt
```

MGMT VRF in HBN Container

Inside the HBN container, a separate mgmt VRF is present. Similar commands as those listed under section "MGMT VRF on Host DPU" can be used to query management routes.

The `*_sf` interfaces use the default routing table while the `eth0` (OOB) uses the mgmt VRF to route out-of-band packets out of the container. The OOB traffic gets NATed through the DPU `oob_net0` interface, ultimately using the DPU OOB's IP address.

When logging into the HBN container via `crictl`, the shell enters the default VRF context by default. Switching to the mgmt VRF can be done using the command `ip vrf exec mgmt <cmd>`.

Existing Services in MGMT VRF on Host DPU

On the host DPU, outside the HBN container, a set of existing services run in the mgmt VRF context as they need OOB network access:

- containerd
- kubelet
- ssh
- docker

These services can be restarted and queried for their status using the command `systemctl` while adding `@mgmt` to the original service name. For example:

- To restart containerd:

```
root@bf2:mgmt:/home/ubuntu# systemctl restart containerd@mgmt
```

- To query containerd status:

```
root@bf2:mgmt:/home/ubuntu# systemctl status containerd@mgmt
```

> **Note**
>
> The original version of these services (without `@mgmt`) are not used and must not be started.

Running New Service in MGMT VRF

If a service needs OOB access to run, it can be added to the set of services running in mgmt VRF context. Adding such a service is only possible on the host DPU (i.e., outside the HBN container).

To add a service to the set of mgmt VRF services:

1. Add it to `/etc/vrf/systemd.conf` (if it is not present already). For example, NTP is already listed in this file.
2. Run the following:

   ```
   root@bf2:mgmt:/home/ubuntu# systemctl daemon-reload
   ```

3. Stop and disable to the non-VRF version of the service to be able to start the mgmt VRF one:

   ```
   root@bf2:mgmt:/home/ubuntu# systemctl stop ntp
   root@bf2:mgmt:/home/ubuntu# systemctl disable ntp
   root@bf2:mgmt:/home/ubuntu# systemctl enable ntp@mgmt
   ```

```
root@bf2:mgmt:/home/ubuntu# systemctl start ntp@mgmt
```

HBN Configuration Examples

HBN Default Configuration

After a fresh HBN installation, the default `/etc/network/interfaces` file would contain only the declaration of the two uplink SFs and a loopback interface.

```
source /etc/network/interfaces.d/*.intf

auto lo
iface lo inet loopback

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf
```

FRR configuration files would also be present under `/etc/frr/` but no configuration would be enabled.

Native Routing with BGP and ECMP

HBN supports unicast routing with BGP and ECMP for IPv4 and IPv6 traffic. ECMP is achieved by distributing traffic using hash calculation based on the source IP , destination IP, and protocol type of the IP header.

Info

For TCP and UDP packets, it also includes source port and destination port.

ECMP Configuration

ECMP is implemented any time routes have multiple paths over uplinks or host ports. For example, 20.20.20.0/24 has 2 paths using both uplinks, so a path is selected based on a hash of the IP headers.

```
20.20.20.0/24 proto bgp metric 20
        nexthop via 169.254.0.1 dev p0_sf weight 1 onlink <<<<<
via uplink p0_sf
        nexthop via 169.254.0.1 dev p1_sf weight 1 onlink <<<<<
via uplink p1_sf
```

Info

HBN supports up to 16 paths for ECMP.

Sample NVUE Configuration

```
nv set interface lo ip address 10.10.10.1/32
nv set interface lo ip address 2010:10:10::1/128
nv set interface vlan100 type svi
nv set interface vlan100 vlan 100
nv set interface vlan100 base-interface br_default
nv set interface vlan100 ip address 2030:30:30::1/64
nv set interface vlan100 ip address 30.30.30.1/24
nv set bridge domain br_default vlan 100
```

```
nv set interface pf0hpf_sf,pf1hpf_sf bridge domain br_default
access 100
nv set vrf default router bgp router-id 10.10.10.1
nv set vrf default router bgp autonomous-system 65501
nv set vrf default router bgp path-selection multipath aspath-
ignore on
nv set vrf default router bgp address-family ipv4-unicast enable
on
nv set vrf default router bgp address-family ipv4-unicast
redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable
on
nv set vrf default router bgp address-family ipv6-unicast
redistribute connected enable on
nv set vrf default router bgp neighbor p0_sf remote-as external
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p0_sf address-family ipv4-
unicast enable on
nv set vrf default router bgp neighbor p0_sf address-family ipv6-
unicast enable on
nv set vrf default router bgp neighbor p1_sf remote-as external
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf address-family ipv4-
unicast enable on
nv set vrf default router bgp neighbor p1_sf address-family ipv6-
unicast enable on
```

Sample Flat Files Configuration

Example `/etc/network/interfaces` configuration:

```
auto lo
iface lo inet loopback
```

```
    address 10.10.10.1/32
    address 2010:10:10::1/128

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf

auto pf0hpf_sf
iface pf0hpf_sf
        bridge-access 100

auto pf1hpf_sf
iface pf1hpf_sf
        bridge-access 100

auto vlan100
iface vlan100
    address 2030:30:30::1/64
    address 30.30.30.1/24
    vlan-raw-device br_default
    vlan-id 100

auto br_default
iface br_default
    bridge-ports pf0hpf_sf pf1hpf_sf
    bridge-vlan-aware yes
    bridge-vids 100
    bridge-pvid 1
```

Example `/etc/frr/daemons` configuration:

```
bgpd=yes
```

```
vtysh_enable=yes


FRR Config file @ /etc/frr/frr.conf -
!
frr version 7.5+cl5.3.0u0
frr defaults datacenter
hostname BLUEFIELD2
log syslog informational
no zebra nexthop kernel enable
!
router bgp 65501
 bgp router-id 10.10.10.1
 bgp bestpath as-path multipath-relax
 neighbor p0_sf interface remote-as external
 neighbor p0_sf advertisement-interval 0
 neighbor p0_sf timers 3 9
 neighbor p0_sf timers connect 10
 neighbor p1_sf interface remote-as external
 neighbor p1_sf advertisement-interval 0
 neighbor p1_sf timers 3 9
 neighbor p1_sf timers connect 10
 !
 address-family ipv4 unicast
  redistribute connected
  maximum-paths 64
  maximum-paths ibgp 64
 exit-address-family
 !
 address-family ipv6 unicast
  redistribute connected
  neighbor p0_sf activate
  neighbor p1_sf activate
  maximum-paths 64
  maximum-paths ibgp 64
 exit-address-family
```

```
!
line vty
!
end
```

BGP Peering with Host

HBN supports the ability to establish a BGP session between the host and DPU and allow the host to announce arbitrary route prefixes through the DPU into the underlay fabric. The host can use any standard BGP protocol stack implementation to establish BGP peering with HBN.

Traffic to and from endpoints on the host gets offloaded.

> **Note**
>
> Both IPv4 and IPv6 unicast AFI/SAFI are supported.

It is possible to apply route filtering for these prefixes to limit the potential security impact in this configuration.

Sample NVUE Configuration

The following code block shows configuration to peer to host at `45.3.0.4` and `2001:cafe:1ead::4`. The BGP session can be established using IPv4 or IPv6 address.

> **Note**
>
> Either of these sessions can support IPv4 unicast and IPv6 unicast AFI/SAFI.

NVUE configuration for peering with host:

```
nv set vrf default router bgp autonomous-system 63642
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 45.3.0.4 nexthop-
connected-check off
nv set vrf default router bgp neighbor 45.3.0.4 peer-group
dpu_host
nv set vrf default router bgp neighbor 45.3.0.4 type numbered
nv set vrf default router bgp neighbor 2001:cafe:1ead::4 nexthop-
connected-check off
nv set vrf default router bgp neighbor 2001:cafe:1ead::4 peer-
group dpu_host
nv set vrf default router bgp neighbor 2001:cafe:1ead::4 type
numbered
nv set vrf default router bgp peer-group dpu_host address-family
ipv4-unicast enable on
nv set vrf default router bgp peer-group dpu_host address-family
ipv6-unicast enable on
nv set vrf default router bgp peer-group dpu_host remote-as
external
```

Sample Flat Files Configuration

The following block shows configuration to peer to host at `45.3.0.4` and `2001:cafe:1ead::4`. The BGP session can be established using IPv4 or IPv6 address.

`frr.conf` file:

```
router bgp 63642
 bgp router-id 27.0.0.4
 bgp bestpath as-path multipath-relax
 neighbor dpu_host peer-group
```

```
neighbor dpu_host remote-as external
neighbor dpu_host bfd 3 1000 1000
neighbor dpu_host advertisement-interval 0
neighbor dpu_host timers 3 9
neighbor dpu_host timers connect 10
neighbor dpu_host disable-connected-check
neighbor fabric peer-group
neighbor fabric remote-as external
neighbor fabric advertisement-interval 0
neighbor fabric timers 3 9
neighbor fabric timers connect 10
neighbor 45.3.0.4 peer-group dpu_host
neighbor 2001:cafe:1ead::4 peer-group dpu_host
neighbor p0_sf interface peer-group fabric
neighbor p1_sf interface peer-group fabric
!
address-family ipv4 unicast
       neighbor dpu_host activate
!
address-family ipv6 unicast
       neighbor dpu_host activate
```

Sample Configuration on Host Running FRR

Any BGP implementation can be used on the host to peer to HBN and advertise endpoints. The following is an example using FRR BGP:

- Sample FRR configuration on the host:

```
bf2-s12# sh run
Building configuration...

Current configuration:
!
```

```
frr version 7.2.1
frr defaults traditional
hostname bf2-s12
no ip forwarding
no ipv6 forwarding
!
router bgp 1000008
!
router bgp 1000008 vrf v_200_2000
 neighbor 45.3.0.2 remote-as external
 neighbor 2001:cafe:1ead::2 remote-as external
 !
 address-family ipv4 unicast
  redistribute connected
 exit-address-family
 !
 address-family ipv6 unicast
  redistribute connected
  neighbor 45.3.0.2 activate
  neighbor 2001:cafe:1ead::2 activate
 exit-address-family
!
line vty
!
end
```

- Sample interface configuration on the host:

```
root@bf2-s12:/home/cumulus# ifquery -a
auto lo
iface lo inet loopback
        address 27.0.0.7/32
        address 2001:c15c:d06:f00d::7/128
```

```
auto v_200_2000
iface v_200_2000
        address 60.1.0.1
        address 60.1.0.2
        address 60.1.0.3
    address 2001:60:1::1
        address 2001:60:1::2
        address 2001:60:1::3
    vrf-table auto
auto ens1f0np0
iface ens1f0np0
        address 45.3.0.4/24
        address 2001:cafe:1ead::4/64
        gateway 45.3.0.1
        gateway 2001:cafe:1ead::1
        vrf v_200_2000
        hwaddress 00:03:00:08:00:12
        mtu 9162
```

L2 EVPN with BGP and ECMP

HBN supports VXLAN with EVPN control plane for intra-subnet bridging (L2) services for IPv4 and IPv6 traffic in the overlay.

For the underlay, only IPv4 or BGP unnumbered configuration is supported.

Single VXLAN Device

With a single VXLAN device, a set of VNIs represents a single device model. The single VXLAN device has a set of attributes that belong to the VXLAN construct. Individual VNIs include VLAN-to-VNI mapping which allows users to specify which VLANs are associated with which VNIs. A single VXLAN device simplifies the configuration and reduces the overhead by replacing multiple traditional VXLAN devices with a single VXLAN device.

Users may configure a single VXLAN device automatically with NVUE, or manually by editing the `/etc/network/interfaces` file. When users configure a single VXLAN device with NVUE, NVUE creates a unique name for the device in the following format using the bridge name as the hash key: `vxlan<id>`.

This example configuration performs the following steps:

1. Creates a single VXLAN device (vxlan21).
2. Maps VLAN 10 to VNI 10 and VLAN 20 to VNI 20.
3. Adds the VXLAN device to the default bridge.

```
cumulus@leaf01:~$ nv set bridge domain bridge vlan 10 vni 10
cumulus@leaf01:~$ nv set bridge domain bridge vlan 20 vni 20
cumulus@leaf01:~$ nv set nve vxlan source address 10.10.10.1
cumulus@leaf01:~$ nv config apply
```

Alternately, users may edit the file `/etc/network/interfaces` as follows, then run the `ifreload -a` command to apply the SVD configuration.

```
auto lo
iface lo inet loopback
    vxlan-local-tunnelip 10.10.10.1

auto vxlan21
iface vxlan21
    bridge-vlan-vni-map 10=10 20=20
    bridge-learning off

auto bridge
iface bridge
    bridge-vlan-aware yes
    bridge-ports vxlan21 pf0hpf_sf pf1hpf_sf
    bridge-vids 10 20
```

```
    bridge-pvid 1
```

> **Note**
>
> Users may not use a combination of single and traditional VXLAN devices.

Sample NVUE Configuration on DPU

The following is a sample NVUE configuration which has L2-VNIs (`2000`, `2001`) for EVPN bridging on DPU.

```
nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default vlan 200 vni 2000 flooding enable
auto
nv set bridge domain br_default vlan 200 vni 2000 mac-learning
off
nv set bridge domain br_default vlan 201 vni 2001 flooding enable
auto
nv set bridge domain br_default vlan 201 vni 2001 mac-learning
off

nv set evpn enable on
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.4
nv set router bgp enable on
nv set system global anycast-mac 44:38:39:42:42:07
nv set vrf default router bgp address-family ipv4-unicast enable
on
```

```
nv set vrf default router bgp address-family ipv4-unicast
redistribute connected enable on

nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 63642
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-
ignore on
nv set vrf default router bgp peer-group fabric address-family
ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family
ipv4-unicast policy outbound route-map MY_ORIGIN_ASPATH_ONLY
nv set vrf default router bgp peer-group fabric address-family
ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family
ipv6-unicast policy outbound route-map MY_ORIGIN_ASPATH_ONLY
nv set vrf default router bgp peer-group fabric address-family
l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family
l2vpn-evpn enable on
nv set vrf default router bgp peer-group fabric remote-as
external
nv set vrf default router bgp router-id 27.0.0.4

nv set interface lo ip address 2001:c15c:d06:f00d::4/128
nv set interface lo ip address 27.0.0.4/32
nv set interface lo type loopback
nv set interface p0_sf,p1_sf,pf0hpf_sf,pf1hpf_sf type swp
nv set interface pf0hpf_sf bridge domain br_default access 200
nv set interface pf1hpf_sf bridge domain br_default access 201

nv set interface vlan200-201 base-interface br_default
```

```
nv set interface vlan200-201 ip ipv4 forward on
nv set interface vlan200-201 ip ipv6 forward on
nv set interface vlan200-201 ip vrr enable on
nv set interface vlan200-201 ip vrr state up
nv set interface vlan200-201 link mtu 9050
nv set interface vlan200-201 type svi
nv set interface vlan200 ip address 2001:cafe:1ead::3/64
nv set interface vlan200 ip address 45.3.0.2/24
nv set interface vlan200 ip vrr address 2001:cafe:1ead::1/64
nv set interface vlan200 ip vrr address 45.3.0.1/24
nv set interface vlan200 vlan 200
nv set interface vlan201 ip address 2001:cafe:1ead:1::3/64
nv set interface vlan201 ip address 45.3.1.2/24
nv set interface vlan201 ip vrr address 2001:cafe:1ead:1::1/64
nv set interface vlan201 ip vrr address 45.3.1.1/24
nv set interface vlan201 vlan 201
```

Sample Flat Files Configuration on HBN

The following is a sample flat files configuration which has L2-VNIs (`vx-2000`, `vx-2001`) for EVPN bridging on DPU.

This file is located at `/etc/network/interfaces`:

```

auto lo
iface lo inet loopback
    address 2001:c15c:d06:f00d::4/128
    address 27.0.0.4/32
    vxlan-local-tunnelip 27.0.0.4

auto p0_sf
iface p0_sf

auto p1_sf
```

```
iface p1_sf

auto pf0hpf_sf
iface pf0hpf_sf
    bridge-access 200

auto pf1hpf_sf
iface pf1hpf_sf
    bridge-access 201

auto vlan200
iface vlan200
    address 2001:cafe:1ead::3/64
    address 45.3.0.2/24
    mtu 9050
    address-virtual 00:00:5e:00:01:01 2001:cafe:1ead::1/64
45.3.0.1/24
    vlan-raw-device br_default
    vlan-id 200

auto vlan201
iface vlan201
    address 2001:cafe:1ead:1::3/64
    address 45.3.1.2/24
    mtu 9050
    address-virtual 00:00:5e:00:01:01 2001:cafe:1ead:1::1/64
45.3.1.1/24
    vlan-raw-device br_default
    vlan-id 201

auto vxlan48
iface vxlan48
    bridge-vlan-vni-map 200=2000 201=2001
217=2017
    bridge-learning off
```

```
auto br_default
iface br_default
    bridge-ports pf0hpf_sf pf1hpf_sf vxlan48
    bridge-vlan-aware yes
    bridge-vids 200 201
    bridge-pvid 1
```

This file tells the `frr` package which daemon to start and is located at `/etc/frr/daemons`:

```

bgpd=yes
ospfd=no
ospf6d=no
isisd=no
pimd=no
ldpd=no
pbrd=no
vrrpd=no
fabricd=no
nhrpd=no
eigrpd=no
babeld=no
sharpd=no
fabricd=no
ripngd=no
ripd=no

vtysh_enable=yes
zebra_options="  -M cumulus_mlag -M snmp -A 127.0.0.1 -s
90000000"
bgpd_options="   -M snmp -A 127.0.0.1"
ospfd_options="  -M snmp -A 127.0.0.1"
ospf6d_options=" -M snmp -A ::1"
ripd_options="   -A 127.0.0.1"
```

```
ripngd_options=" -A ::1"
isisd_options="  -A 127.0.0.1"
pimd_options="   -A 127.0.0.1"
ldpd_options="   -A 127.0.0.1"
nhrpd_options="  -A 127.0.0.1"
eigrpd_options=" -A 127.0.0.1"
babeld_options=" -A 127.0.0.1"
sharpd_options=" -A 127.0.0.1"
pbrd_options="   -A 127.0.0.1"
staticd_options="-A 127.0.0.1"
fabricd_options="-A 127.0.0.1"
vrrpd_options="  -A 127.0.0.1"

frr_profile="datacenter"
```

This file is located at `/etc/frr/frr.conf`:

```
!---- Cumulus Defaults ----
frr defaults datacenter
log syslog informational
no zebra nexthop kernel enable
vrf default
outer bgp 63642 vrf default
bgp router-id 27.0.0.4
bgp bestpath as-path multipath-relax
timers bgp 3 9
bgp deterministic-med
! Neighbors
neighbor fabric peer-group
neighbor fabric remote-as external
neighbor fabric timers 3 9
neighbor fabric timers connect 10
neighbor fabric advertisement-interval 0
neighbor p0_sf interface peer-group fabric
```

```
neighbor p1_sf interface peer-group fabric
address-family ipv4 unicast
maximum-paths ibgp 64
maximum-paths 64
distance bgp 20 200 200
neighbor fabric activate
exit-address-family
address-family ipv6 unicast
maximum-paths ibgp 64
maximum-paths 64
distance bgp 20 200 200
neighbor fabric activate
exit-address-family
address-family l2vpn evpn
advertise-all-vni
neighbor fabric activate
exit-address-family
```

Sample Switch Configuration for EVPN

The following is a sample NVUE config for underlay switches (NVIDIA® Spectrum® with Cumulus Linux) for EVPN use case.

It assumes that the uplinks on DPUs are connected to ports `swp1-4` on the switch.

```
nv set evpn enable on
nv set router bgp enable on

nv set vrf default router bgp address-family ipv4-unicast enable
on
nv set vrf default router bgp address-family ipv4-unicast
redistribute connected enable on

nv set vrf default router bgp address-family l2vpn-evpn enable on
```

```
nv set vrf default router bgp autonomous-system 63640
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp1 peer-group fabric
nv set vrf default router bgp neighbor swp1 type unnumbered
nv set vrf default router bgp neighbor swp2 peer-group fabric
nv set vrf default router bgp neighbor swp2 type unnumbered
nv set vrf default router bgp neighbor swp3 peer-group fabric
nv set vrf default router bgp neighbor swp3 type unnumbered
nv set vrf default router bgp neighbor swp4 peer-group fabric
nv set vrf default router bgp neighbor swp4 type unnumbered
nv set vrf default router bgp path-selection multipath aspath-
ignore on
nv set vrf default router bgp peer-group fabric address-family
ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family
ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family
l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family
l2vpn-evpn enable on
nv set vrf default router bgp peer-group fabric remote-as
external
nv set vrf default router bgp router-id 27.0.0.10

nv set interface lo ip address 2001:c15c:d06:f00d::10/128
nv set interface lo ip address 27.0.0.10/32
nv set interface lo type loopback
nv set interface swp1,swp2,swp3,swp4 type swp
```

Access Control Lists

Access Control Lists (ACLs) are a set of rules that are used to filter network traffic. These rules are used to specify the traffic flows that must be permitted or blocked at networking device interfaces. There are two types of ACLs:

- Stateless ACLs – rules that are applied to individual packets. They inspect each packet individually and permit/block the packets based on the packet header information and the match criteria specified by the rule.
- Stateful ACLs – rules that are applied to traffic sessions/connections. They inspect each packet with respect to the state of the session/connection to which the packet belongs to determine whether to permit/block the packet.

ACL Ordering

ACL ordering ensures that the order in which ACLs are executed in DPU hardware is the same as the order in which the ACLs are configured. In general, IPv4 ACLs should be configured before IPv6 ACLs which in turn should be configured before L2 ACLs. ACLs should be configured in the following order:

1. IPv4 header match fields + UDP header match fields
2. IPv4 header match fields + TCP header match fields
3. IPv4 header match fields + ICMP header match fields
4. IPv4 header match fields
5. IPv6 header match fields + UDP header match fields
6. IPv6 header match fields + TCP header match fields
7. IPv6 header match fields + ICMP header match fields
8. IPv6 header match fields
9. Ethernet header match fields

Stateless ACLs

HBN supports configuration of stateless ACLs for IPv4 packets, IPv6 packets, and Ethernet frames. The following examples depict how stateless ACLs are configured for each case, with NVUE and with flat files (`cl-acltool`).

NVUE Examples for Stateless ACLs

NVUE IPv4 ACLs Example

The following is an example of an ingress IPv4 ACL that permits DHCP request packets ingressing on the `pf0hpf_sf` port towards the DHCP server:

```
root@hbn01-host01:~# nv set acl acl1_ingress type ipv4
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip
protocol udp
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip
dest-port 67
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip
source-port 68
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 action
permit
```

Bind the ingress IPv4 ACL to host representor port `pf0hpf_sf` of the DPU in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl1_ingress
inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress IPv4 ACL that permits DHCP reply packets egressing out of the `pf0hpf_sf` port towards the DHCP client:

```
root@hbn01-host01:~# nv set acl acl2_egress type ipv4
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip
protocol udp
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip
dest-port 68
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip
source-port 67
```

```
root@hbn01-host01:~# nv set acl acl2_egress rule 200 action
permit
```

Bind the egress IPv4 ACL to host representor port `pf0hpf_sf` of the DPU in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl2_egress
outbound
root@hbn01-host01:~# nv config apply
```

NVUE IPv6 ACLs Example

The following is an example of an ingress IPv6 ACL that permits traffic with matching `dest-ip` and `protocol tcp` ingress on port `pf0hpf_sf`:

```
root@hbn01-host01:~# nv set acl acl5_ingress type ipv6
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 match ip
protocol tcp
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 match ip
dest-ip 48:2034::80:9
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 action
permit
```

Bind the ingress IPv6 ACL to host representor port `pf0hpf_sf` of the DPU in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl5_ingress
inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress IPv6 ACL that permits traffic with matching `source-ip` and `protocol tcp` egressing out of port `pf0hpf_sf`:

```
root@hbn01-host01:~# nv set acl acl6_egress type ipv6
root@hbn01-host01:~# nv set acl acl6_egress rule 101 match ip
protocol tcp
root@hbn01-host01:~# nv set acl acl6_egress rule 101 match ip
source-ip 48:2034::80:9
root@hbn01-host01:~# nv set acl acl6_egress rule 101 action
permit
```

Bind the egress IPv6 ACL to host representor port `pf0hpf_sf` of the DPU in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl6_egress
outbound
root@hbn01-host01:~# nv config apply
```

NVUE L2 ACLs Example

The following is an example of an ingress MAC ACL that permits traffic with matching `source-mac` and `dest-mac` ingressing to port `pf0hpf_sf`:

```
root@hbn01-host01:~# nv set acl acl3_ingress type mac
root@hbn01-host01:~# nv set acl acl3_ingress rule 1 match mac
source-mac 00:00:00:00:00:0a
root@hbn01-host01:~# nv set acl acl3_ingress rule 1 match mac
dest-mac 00:00:00:00:00:0b
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl3_ingress
inbound
```

Bind the ingress MAC ACL to host representor port `pf0hpf_sf` of the DPU in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl3_ingress
inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress MAC ACL that permits traffic with matching `source-mac` and `dest-mac` egressing out of port `pf0hpf_sf`:

```
root@hbn01-host01:~# nv set acl acl4_egress type mac
root@hbn01-host01:~# nv set acl acl4_egress rule 2 match mac
source-mac 00:00:00:00:00:0b
root@hbn01-host01:~# nv set acl acl4_egress rule 2 match mac
dest-mac 00:00:00:00:00:0a
root@hbn01-host01:~# nv set acl acl4_egress rule 2 action permit
```

Bind the egress MAC ACL to host representor port `pf0hpf_sf` of the DPU in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl4_egress
outbound
root@hbn01-host01:~# nv config apply
```

Flat Files (cl-acltool) Examples for Stateless ACLs

For the same examples cited above, the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`. The rules in `/etc/cumulus/acl/policy.d/<rule_name.rules>` are configured using Linux iptables/ip6tables/ebtables.

Flat Files IPv4 ACLs Example

The following example configures an ingress IPv4 ACL rule matching with DHCP request under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of the DPU followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL acl1_ingress in dir inbound on interface pf0hpf_sf ##
-t filter -A FORWARD -i pf1vf1_sf -p udp --sport 68 --dport 67 -j
ACCEPT
```

The following example configures an egress IPv4 ACL rule matching with DHCP reply under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the egress interface as the host representor of the DPU followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL acl2_egress in dir outbound on interface pf0hpf_sf ##
-t filter -A FORWARD -o pf0hpf_sf -p udp --sport 67 --dport 68 -j
ACCEPT
```

Flat File IPv6 ACLs Example

The following example configures an ingress IPv6 ACL rule matching with `dest-ip` and `tcp` protocol under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of the DPU followed by invoking `cl-acltool -i`:

```
[ip6tables]
## ACL acl5_ingress in dir inbound on interface pf0hpf_sf ##
-t filter -A FORWARD -i pf0hpf_sf -d 48:2034::80:9 -p tcp -j
ACCEPT
```

The following example configures an egress IPv6 ACL rule matching with `source-ip` and `tcp` protocol under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the egress interface as the host representor of the DPU followed by invoking `cl-acltool -i`:

```
[ip6tables]
## ACL acl6_egress in dir outbound on interface pf0hpf_sf ##
-t filter -A FORWARD -o pf0hpf_sf -s 48:2034::80:9 -p tcp -j
ACCEPT
```

Flat Files L2 ACLs Example

The following example configures an ingress MAC ACL rule matching with `source-mac` and `dest-mac` under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of the DPU followed by invoking `cl-acltool -i`:

```
[ebtables]
## ACL acl3_ingress in dir inbound on interface pf0hpf_sf ##
-t filter -A FORWARD -i pf0hpf_sf -s
00:00:00:00:00:0a/ff:ff:ff:ff:ff:ff -d
00:00:00:00:00:0b/ff:ff:ff:ff:ff:ff -j ACCEPT
```

The following example configures an egress MAC ACL rule matching with `source-mac` and `dest-mac` under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with egress interface as host representor of DPU followed by invoking `cl-acltool -i`:

```
[ebtables]
## ACL acl4_egress in dir outbound on interface pf0hpf_sf ##
```

```
-t filter -A FORWARD -o pf0hpf_sf -s
00:00:00:00:00:0b/ff:ff:ff:ff:ff:ff -d
00:00:00:00:00:0a/ff:ff:ff:ff:ff:ff -j ACCEPT
```

Stateful ACLs

Stateful ACLs facilitate monitoring and tracking traffic flows to enforce per-flow traffic filtering (unlike stateless ACLs which filter traffic on a per-packet basis). HBN supports stateful ACLs using reflexive ACL mechanism. Reflexive ACL mechanism is used to permit initiation of connections from within the network to outside the network and allow only replies to the initiated connections from outside the network.

HBN supports stateful ACL configuration for IPv4 traffic.

Stateful ACLs can be applied for routed traffic (north-south traffic) or bridged traffic (east-west traffic). Stateful ACLs applied for routed traffic are called "L3 stateful ACLs" and for bridged traffic are called "L2 stateful ACLs". Currently, NVUE-based configuration is supported only for L3 stateful ACLs (L2 stateful ACLs must be configured using flat-file configuration).

Stateful ACLs in HBN are disabled by default. To enable stateful ACL functionality, use the following NVUE commands:

```
root@hbn03-host00:~# nv set system reflexive-acl enable
root@hbn03-host00:~# nv config apply
```

If using flat-file configuration (and not NVUE), edit the file `/etc/cumulus/nl2docad.d/acl.conf` and set the knob `rflx.reflexive_acl_enable` to `TRUE`. To apply this change, execute:

```
root@hbn03-host00:~# supervisorctl start nl2doca-reload
```

NVUE Examples for L3 Stateful ACLs

The following is an example of allowing HTTP (TCP) connection originated by the host where the DPU is hosted to an HTTP server (with the IP address 11.11.11.11) on an external network. Two sets of ACLs matching with CONNTRACK state must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with CONNTRACK state of NEW, ESTABLISHED and bind it to the host representor of the DPU and the associated VLAN's SVI in the inbound direction.
- Configure an ACL rule matching TCP/HTTP connection/flow details with CONNTRACK state of ESTABLISHED and bind it to the host representor of the DPU and the associated VLAN's SVI in the outbound direction.

In this example, the host representor on the DPU is `pf0hpf_sf` and it is part of VLAN 101 (SVI interface is `vlan101`).

1. Configure the ingress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule
11 action permit
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule
11 match conntrack new
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule
11 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule
11 match ip dest-ip 11.11.11.11/32
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule
11 match ip dest-port 80
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule
11 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type
ipv4
```

2. Bind this ACL to the host representor of the DPU and the associated VLAN's SVI interface in the inbound direction:

```
root@hbn03-host00:~# nv set interface pf0hpf_sf,vlan101 acl
allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv config apply
```

3. Configure the egress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server
rule 21 action permit
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server
rule 21 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server
rule 21 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server
type ipv4
root@hbn03-host00:~# nv config apply
```

4. Bind this ACL to the host representor of the DPU and the associated VLAN's SVI interface in the outbound direction:

```
root@hbn03-host00:~# nv set interface pf0hpf_sf,vlan101 acl
allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

> **Note**
>
> If virtual router redundancy (VRR) is set, L3 stateful ACLs must be bound to all the related SVI interfaces. For example, if VRR is configured on SVI `vlan101` as follows in the `/etc/network/interfaces` file:

```
auto vlan101
iface vlan101
    address 45.3.1.2/24
    address-virtual 00:00:5e:00:01:01 45.3.1.1/24
    vlan-raw-device br_default
    vlan-id 101
```

With this configuration, two SVI interfaces, `vlan101` and `vlan101-v0` would be created in the system:

```
root@hbn03-host00:~# ip -br addr show | grep
vlan101
vlan101@br_default UP             45.3.1.2/24
fe80::204:4bff:fe8a:f100/64
vlan101-v0@vlan101 UP             45.3.1.1/24
metric 1024 fe80::200:5eff:fe00:101/64
```

In this case, stateful ACLs must be bound to both SVI interfaces (`vlan101` and `vlan101-v0`). In the stateful ACL described in the current section, the binding would be:

```
root@hbn03-host00:~# nv set interface
pf0hpf_sf,vlan101,vlan101-v0 acl
allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv set interface
pf0hpf_sf,vlan101,vlan101-v0 acl
allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

Flat Files (cl-acltool) Examples for L3 Stateful ACLs

For the example described under section "NVUE Examples for L3 Stateful ACLs", the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i` to install the rules in DPU hardware.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the host representor of the DPU and the associated VLAN's SVI followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface
pf1vf7_sf ##
-t mangle -A PREROUTING -p tcp -d 11.11.11.11/32 --dport 80 -
m conntrack --ctstate EST,NEW -m connmark ! --mark 9998 -j
CONNMARK --set-mark 9999
-t filter -A FORWARD -i pf1vf7_sf -p tcp -d 11.11.11.11/32 --
dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT

## ACL allow_tcp_conn_from_host in dir inbound on interface
vlan118 ##
-t filter -A FORWARD -i vlan118 -p tcp –d 11.11.11.11/32--
dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT
```

> **Note**
>
> A mangle table rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for ingress ACL rules

are protocol dependent: 9999 for TCP, 9997 for UDP, and 9995 for ICMP.

2. Configure an egress ACL rule matching with TCP and CONNTRACK state of ESTABLISHED, RELATED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` file with the egress interface as the host representor of the DPU and the associated VLAN's SVI followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on interface pf1vf7_sf ##
-t mangle -A PREROUTING -p tcp -s 11.11.11.11/32 --sport 80 -m conntrack --ctstate EST -j CONNMARK --set-mark 9998
-t filter -A FORWARD -o pf1vf7_sf -p tcp -m conntrack --ctstate EST,REL -j ACCEPT

## ACL allow_tcp_resp_from_server in dir outbound on interface vlan118 ##
-t filter -A FORWARD -o vlan118 -p tcp -m conntrack --ctstate EST,REL -j ACCEPT
```

Note

A mangle table rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 9998 for TCP, 9996 for UDP, and 9994 for ICMP.

Flat Files (cl-acltool) Examples for L2 Stateful ACLs

For the same example cited above (HTTP server at IP address 192.168.5.5 accessible over bridged network), the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the host representor of the DPU and the associated VLAN's SVI followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface pf1vf7_sf
-t mangle -A PREROUTING -p tcp -d 192.168.5.5/32 --dport 80 -m conntrack --ctstate EST,NEW -m connmark ! --mark 9998 -j CONNMARK --set-mark 9999
-t filter -A FORWARD -m physdev --physdev-in pf1vf7_sf -p tcp -d 192.168.5.5/32 --dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT

## ACL allow_tcp_conn_from_host in dir inbound on interface vlan118 ##
-t filter -A FORWARD -i vlan118 -p tcp –d 192.168.5.5/32--dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT
```

> **Note**
>
> A mangle table rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for ingress ACL rules are protocol dependent: 9999 for TCP, 9997 for UDP, and 9995 for ICMP.

2. Configure an egress ACL rule matching with TCP and CONNTRACK state of ESTABLISHED, RELATED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` file with the egress interface as the host representor of the DPU and the associated VLAN's SVI followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on
interface pf1vf7_sf ##
-t mangle -A PREROUTING -p tcp -s 192.168.5.5/32 --sport 80 -
m conntrack --ctstate EST -j CONNMARK --set-mark 9998
-t filter -A FORWARD -m physdev --physdev-out pf1vf7_sf -p
tcp -m conntrack --ctstate EST,REL -j ACCEPT

## ACL allow_tcp_resp_from_server in dir outbound on
interface vlan118 ##
-t filter -A FORWARD -o vlan118 -p tcp -m conntrack --ctstate
EST,REL -j ACCEPT
```

Note

A mangle table rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 9998 for TCP, 9996 for UDP, and 9994 for ICMP.

DHCP Relay on HBN

DHCP is a client server protocol that automatically provides IP hosts with IP addresses and other related configuration information. A DHCP relay (agent) is a host that forwards DHCP packets between clients and servers. DHCP relays forward requests and replies between clients and servers that are not on the same physical subnet.

DHCP relay can be configured using either flat file (supervisord configuration) or through NVUE.

Configuration

HBN is a non-systemd based container. Therefore, the DHCP relay must be configured as explained in the following subsections.

Flat File Configuration (Supervisord)

The HBN initialization script installs default configuration files on the DPU in `/var/lib/hbn/etc/supervisor/conf.d/`. The DPU directory is mounted to `/etc/supervisor/conf.d` which achieves configuration persistence.

By default, DHCP relay is disabled. Default configuration applies to one instance of DHCPv4 relay and DHCPv6 relay in the default VRF.

NVUE Configuration

The user can use NVUE to configure and maintain DHCPv4 and DHCPv6 relays with CLI and REST API. NVUE generates all the required configurations and maintains the relay service.

DHCPv4 Relay Configuration

NVUE Example

The following configuration starts a relay service which listens for the DHCP messages on `p0_sf`, `p1_sf`, and `vlan482` and relays the requests to DHCP server 10.89.0.1 with `gateway-interface` as `lo`.

```
nv set service dhcp-relay default gateway-interface lo
```

```
nv set service dhcp-relay default interface p0_sf
nv set service dhcp-relay default interface p1_sf
nv set service dhcp-relay default interface vlan482 downstream
nv set service dhcp-relay default server 10.89.0.1
```

Flat Files Example

```
[program: isc-dhcp-relay-default]
command = /usr/sbin/dhcrelay --nl -d -i p0_sf -i p1_sf -id
vlan482 -U lo 10.89.0.1
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

Where:

Option	Description
`-i`	Network interface to listen on for requests and replies
`-iu`	Upstream network interface
`-id`	Downstream network interface
`-U [address]%%ifname`	Gateway IP address interface. Use `%%` for `IP%%ifname`. `%` is used as an escape character.
`--loglevel-debug`	Debug logging. Location: `/var/log/syslog`.
`-a`	Append an agent option field to each request before forwarding it to the server with default values for `circuit-id` and `remote-id`

Option	Description
`-r remote-id`	Set a custom remote ID string (max of 255 chars). To use this option, you must also enable the `-a` option.
`--use-pif-circuit-id`	Set the underlying physical interface which receives the packet as the `circuit-id`. To use this option you must also enable the `-a` option.

DHCPv4 Relay Option 82

NVUE Example

The following NVUE command is used to enable option 82 insertion in DHCP packets with default values:

```
nv set service dhcp-relay default agent enable on
```

To provide a custom `remote-id` (e.g., host10) using NVUE:

```
nv set service dhcp-relay default agent remote-id host10
```

To use the underlying physical interface on which the request is received as `circuit-id` using NVUE:

```
nv set service dhcp-relay default agent use-pif-circuit-id enable
on
```

Flat Files Example

```
[program: isc-dhcp-relay-default]
command = /usr/sbin/dhcrelay --nl -d -i p0_sf -i p1_sf -id
vlan482 -U lo -a --use-pif-circuit-id -r host10 10.89.0.1
```

```
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

DHCPv6 Relay Configuration

NVUE Example

The following NVUE command starts the DHCPv6 Relay service which listens for DHCPv6 requests on `vlan482` and sends relayed DHCPv6 requests towards `p0_sf` and `p1_sf`.

```
nv set service dhcp-relay6 default interface downstream vlan482
nv set service dhcp-relay6 default interface upstream p0_sf
nv set service dhcp-relay6 default interface upstream p1_sf
```

Flat Files Example

```
[program: isc-dhcp-relay6-default]
command = /usr/sbin/dhcrelay --nl -6 -d -l vlan482 -u p0_sf -u
p1_sf
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

Where:

Option	Description
`-l [address]%%ifname[#index]`	Downstream interface. Use `%%` for `IP%%ifname`. `%` is used as escape character.
`-u [address]%%ifname`	Upstream interface. Use `%` `%` for `IP%%ifname`. `%` is used as escape character.
`-6`	IPv6
`--loglevel-debug`	Debug logging located at `/var/log/syslog`

DHCP Relay and VRF Considerations

DHCP relay can be spawned inside a VRF context to handle the DHCP requests in that VRF. There can only be 1 instance each of DHCPv4 relay and DHCPv6 relay per VRF. To achieve that, the user can follow these guidelines:

- DHCPv4 on default VRF:

  ```
  /usr/sbin/dhcrelay --nl -i <interface> -U [address]%%
  <interface> <server_ip>
  ```

- DHCPv4 on VRF:

  ```
  /usr/sbin/ip vrf exec <vrf> /usr/sbin/dhcrelay --nl -i
  <interface> -U [address]%%<interface> <server_ip>
  ```

- DHCPv6 on default VRF:

```
/usr/sbin/dhcrelay --nl -6 -l <interface> -u <interface>
```

- DHCPv6 on VRF:

```
/usr/sbin/ip vrf exec <vrf> /usr/sbin/dhcrelay --nl -6 -l
<interface> -u <interface>
```

Troubleshooting

HBN Container Stuck in init-sfs

The HBN container starts as `init-sfs` and should transition to `doca-hbn` within 2 minutes as can be seen using `crictl ps`. But sometimes it may remain as `init-sfs`.

This can happen if interface `p0_sf` is missing. Run the command `ip -br link show dev p0_sf` in the DPU and inside the container to check if `p0_sf` is present or not. If its missing, make sure the firmware is upgraded to the latest version. Gracefully shutdown and power cycle the host for the new firmware to take effect.

BGP Session not Establishing

One of the main causes of a BGP session not getting established is a mismatch in MTU configuration. Make sure the MTU on all interfaces is the same. For example, if BGP is failing on `p0`, check and verify that there is a matching MTU value for `p0`, `p0_sf_r`, `p0_sf`, and the remote peer of `p0`.

Generating Support Dump

HBN support dump can be generated using the `cl-support` command, inside the HBN container:

```
root@bf2:/tmp# cl-support
Please send /var/support/cl_support_bf2-s02-1-
ipmi_20221025_180508.txz to Cumulus support
```

The generated dump would be available in `/var/support` in the HBN container and would contain any process core dump as well as log files.

The `/var/support` directory is also mounted on the host DPU at `/var/lib/hbn/var/support`.

SFC Troubleshooting

To troubleshoot flows going through SFC interfaces, the first step is to disable the `nl2doca` service in the HBN container:

```
root@bf2:/tmp# supervisorctl stop nl2doca
nl2doca: stopped
```

Stopping `nl2doca` effectively stops hardware offloading and switches to software forwarding. All packets would appear on `tcpdump` capture on the DPU interfaces.

`tcpdump` can be performed on SF interfaces as well as VLAN, VXLAN, and uplinks to determine where a packet gets dropped or which flow a packet is taking.

General nl2doca Troubleshooting

The following steps can be used to make sure the nl2doca daemon is up and running:

1. Make sure there are no errors in the nl2doca log file at `/var/log/hbn/nl2docad.log`.
2. To check the status of the nl2doca daemon under supervisor, run:

```
supervisorctl status nl2doca
```

3. Use `ps` to check that the actual nl2doca process is running:

```
ps -eaf | grep nl2doca
root           18       1  0 06:31 ?        00:00:00 /bin/bash
/usr/bin/nl2doca-docker-start
root         1437      18  0 06:31 ?        00:05:49
/usr/sbin/nl2docad
```

4. The core file should be in `/var/support/core/`.
5. Check if the `/cumulus/nl2docad/run/stats/punt` is accessible. Otherwise, nl2doca may be stuck and should be restarted:

```
supervisorctl restart nl2doca
```

nl2doca Offload Troubleshooting

If a certain traffic flow does not work as expected, disable nl2doca (i.e., disable hardware offloading):

```
supervisorctl stop nl2doca
```

With hardware offloading disabled, you can confirm it is an offloading issue if the traffic starts working. If it is not an offloading issue, use `tcpdump` on various interfaces to see where the packet gets dropped.

Offloaded entries can be checked in following files, which contain the programming status of every IP prefix and MAC address known to system.

- Bridge entries are available in the file `/cumulus/nl2docad/run/software-tables/17`. It includes all the MAC addresses in the system including local and remote MAC addresses.

  Example format:

  ```
  - flow-entry: 0xaaab0cef4190
        flow-pattern:
          fid: 112
          dst mac: 00:00:5e:00:01:01
        flow-actions:
          SET VRF: 2
          OUTPUT-PD-PORT: 20(TO_RTR_INTF)
          STATS:
            pkts: 1719
            bytes: 191286
  ```

- Router entries are available in the file `/cumulus/nl2docad/run/software-tables/18`. It includes all the IP prefixes known to the system.

  Example format for Entry with ECMP:

  ```
  Entry with ECMP:
  - flow-entry: 0xaaaada723700
  ```

```
  flow-pattern:
     IPV6: LPM
     VRF: 0
     destination-ip: ::/0
  flow-actions :
     ECMP: 2
     STATS:
        pkts: 0
        bytes: 0

Entry without ECMP: - flow-entry: 0xaaaada7e1400
    flow-pattern:
       IPV4: LPM
       VRF: 0
       destination-ip: 60.1.0.93/32
    flow-actions :
        SET FID: 200
        SMAC: 00:04:4b:a7:88:00
        DMAC: 00:03:00:08:00:12
        OUTPUT-PD-PORT: 19(TO_BR_INTF)
   STATS:
       pkts: 0
       bytes: 0
```

- ECMP entries are available in the file `/cumulus/nl2docad/run/software-tables/19`. It includes all the next hops in the system.

  Example format:

```
- ECMP: 2
  ref-count: 2
  num-next-hops: 2
  entries:
```

```
  - { index: 0, fid: 4100, src mac: 'b8:ce:f6:99:49:6a', dst
mac: '00:02:00:00:00:0a' }
  - { index: 1, fid: 4101, src mac: 'b8:ce:f6:99:49:6b', dst
mac: '00:02:00:00:00:0e' }
```

To check counters for packets going to the kernel, run:

```
cat /cumulus/nl2docad/run/stats/punt
PUNT miss pkts:3154 bytes:312326
PUNT miss drop pkts:0 bytes:0
PUNT control pkts:31493 bytes:2853186
PUNT control drop pkts:0 bytes:0
ACL PUNT pkts:68 bytes:7364
ACL drop pkts:0 bytes:0
```

For a specific type of packet flow, programming can be referenced in block specific files. The typical flow is as follows:

For example, to check L2 EVPN ENCAP flows for remote MAC `8a:88:d0:b1:92:b1` on port `pf0vf0_sf`, the basic offload flow should look as follows: RxPort (`pf0vf0_sf`) -> BR (Overlay) -> RTR (Underlay) -> BR (Underlay) -> TxPort (one of the uplink `p0_sf` or `p1_sf` based on ECMP hash).

Step-by-step procedure:

1. Navigate to the interface file `/cumulus/nl2docad/run/software-tables/20`.
2. Check for the RxPort (`pf0vf0_sf`):

```
Interface: pf0vf0_sf
    PD PORT: 6
    HW PORT: 16
    NETDEV PORT: 11
    Bridge-id: 61
```

```
    Untagged FID: 112
```

FID 112 is given to the receive port.

3. Check the bridge table file `/cumulus/nl2docad/run/software-tables/17` with destination MAC `8a:88:d0:b1:92:b1` and FID 112:

```
flow-pattern:
      fid: 112
        dst mac: 8a:88:d0:b1:92:b1
      flow-actions:
        VXLAN ENCAP:
          ENCAP dst ip: 6.0.0.26
          ENCAP vni id: 1000112
        SET VRF: 0
        OUTPUT-PD-PORT: 20(TO_RTR_INTF)
        STATS:
          pkts: 100
          bytes: 10200
```

4. Check the router table file `/cumulus/nl2docad/run/software-tables/18` with destination IP `6.0.0.26` and VRF 0:

```
flow-pattern:
        IPV4: LPM
        VRF: 0
        ip dst: 6.0.0.26/32
      flow-actions :
        ECMP: 1
        OUTPUT PD PORT: 2(TO_BR_INTF)
        STATS:
          pkts: 300
```

```
          bytes: 44400
```

5. Check the ECMP table file `/cumulus/nl2docad/run/software-tables/19` with ECMP 1:

```
- ECMP: 1
      ref-count: 7
      num-next-hops: 2
      entries:
        - { index: 0, fid: 4100, src mac: 
'b8:ce:f6:99:49:6a', dst mac: '00:02:00:00:00:2f' }
        - { index: 1, fid: 4115, src mac: 
'b8:ce:f6:99:49:6b', dst mac: '00:02:00:00:00:33' }
```

6. The ECMP hash calculation picks one of these paths for next-hop rewrite. Check bridge table file for them (`fid=4100, dst mac: 00:02:00:00:00:2f` or `fid=4115, dst mac: 00:02:00:00:00:33`):

```
flow-pattern:
        fid: 4100
        dst mac: 00:02:00:00:00:2f
flow-actions:
    OUTPUT-PD-PORT: 36(p0_sf)
    STATS:
       pkts: 1099
       bytes: 162652
```

This will show the packet going out on the uplink.

NVUE Troubleshooting

To check the status of the NVUE daemon, run:

```
supervisorctl status nvued
```

To restart the NVUE daemon, run:

```
supervisorctl restart nvued
```

HBN Service Release Notes

The following subsections provide information on HBN service new features, interoperability, known issues, and bug fixes.

Changes and New Features

HBN 2.0.0 offers the following new features and updates:

- New hardware-accelerated dataplane based on OVS-DOCA
- Added support for HBN interoperating with SNAP storage service for NVMe-oF and NVMe block device emulation scenario (for NVIDIA® BlueField®-2 only)

Supported Platforms and Interoperability

Supported BlueField Networking Platforms

HBN 2.0.0 has been validated on the following NVIDIA BlueField Networking Platforms:

- BlueField-2 DPU platforms:
    - BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
    - BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB

management; FHHL

    - BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
    - BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
- BlueField-3 DPU platforms:
    - BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
    - BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
    - BlueField-3 B3240 P-Series Dual-slot FHHL DPU; 400GbE / NDR IB (default mode); Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled

> **Note**
>
> Single-port BlueField platforms are currently not supported with HBN.

Supported BlueField OS

HBN 2.0.0 supports DOCA 2.5.0 (BSP 4.5.0) on Ubuntu 22.04 OS.

Verified Scalability Limits

HBN 2.0.0 has been tested to sustain the following maximum scalability limits:

Limit	BlueField-2	BlueField-3	Comments
VTEP peers (DPUs per control plane) in the fabric	2k	2k	Number of DPUs (VTEPs) within a single overlay fabric (reachable in the underlay)
VNIs/overlay networks in the fabric	18	18	Total number of L2 VNIs in the fabric (max VNIs = max VF + max PF)
DPUs per VNI/overlay network	3, 2000	3, 2000	Total number of DPUs configured with the same VNI (3 real DPUs, 2000 emulated VTEPs)
Tenants (L3 VNIs) per server	8	8	Maximum number of tenants on the same host server
VM/pods per server	16	16	Maximum number of IP addresses advertised by EVPN in DPU
Maximum number of L3 LPM routes (underlay)	256	256	IPv4 or IPv6 underlay LPM routes per DPU
Maximum number of EVPN type-2 entries	4K	4k	Remote overlay MAC/IP entries for compute peers stored on a single DPU (L2 EVPN use case)
Maximum number of EVPN type-5 entries	16K	16K	Remote overlay L3 LPM entries for compute peers stored on a single DPU (L3 EVPN use case) **Info** Supported at beta level.
Maximum number of PFs	2	2	Total number of PFs visible to the host
Maximum number of VFs	16	16	Total number of VFs created on the host

Known Issues

The following table lists the known issues and limitations for this release of HBN.

Reference	Description
3705894	Description: In an EVPN Symmetric Routing scenario, IPv6 traffic is not hardware offloaded. It is only IPv6 traffic that is routed using L3VNIs to remote hosts that is affected.
	Workaround: N/A
	Keyword: EVPN; IPv6
	Reported in HBN version: 2.0.0
3378928	Description: When an interface is brought down or deleted (e.g., an SVI deletion), the routes learned over that interface, though removed from kernel, are not notified to netlink. Hence, these routes are still present in nl2doca and consequently in the FDB. If upon raising an interface these older routes are not newly installed, then those stale routes in nl2doca remain until nl2doca is restarted or a suggested workaround is applied.
	Workaround: Resync netlink cache with kernel. `echo 1 > /cumulus/nl2docad/ctrl/netlink/resync`
	Keyword: Container
	Reported in HBN version: 2.0.0
3519324	Description: The DOCA HBN container takes about 1 minute longer to spawn, as compared to previous HBN release (1.4.0).
	Workaround: N/A
	Keyword: Container
	Reported in HBN version: 1.5.0
3605486	Description: When the DPU boots up after issuing a "reboot" command from the DPU itself, some host-side interfaces may remain down.
	Workaround: N/A
	Keyword: Reboot
	Reported in HBN version: 1.5.0
3547	Description: IPv6 stateless ACLs are not supported.

103	Workaround: N/A
	Keyword: IPv6 ACL
	Reported in HBN version: 1.5.0
3339304	Description: Statistics for hardware-offloaded traffic are not reflected on SFs inside an HBN container.
	Workaround: Look up the stats using `ip -s link show` on PFs outside of the HBN container. PFs would show Tx/Rx stats for traffic that is hardware-accelerated in the HBN container.
	Keyword: Statistics; container
	Reported in HBN version: 1.4.0
3352003	Description: NVUE show, config, and apply commands malfunction if the `nvued` and `nvued-startup` services are not in the `RUNNING` and `EXITED` states respectively.
	Workaround: N/A
	Keyword: NVUE commands
	Reported in HBN version: 1.3.0
3168683	Description: If many interfaces are participating in EVPN/routing, it is possible for the routing process to run out of memory.
	Workaround: Have a maximum of 8 VF interfaces participating in routing/VXLAN.
	Keyword: R outing; memory
	Reported in HBN version: 1.2.0
3219539	Description: TC rules are programmed by OVS to map uplink and host representor ports to HBN service. These rules are ageable and can result in packets needing to get software forwarded periodically to refresh the rules.
	Workaround: The timeout value can be adjusted by changing the OVS parameter `other_config : max-idle` as documented here. The shipped default value is 10000ms (10s).
	Keyword: SFC; aging
	Reported in HBN version: 1.2.0
3184745	Description: The command `nv show interface <intf> acl` does not show correct information if there are multiple ACLs bound to the interface.

	Workaround: Use the command `nv show interface <intf>` to view the ACLs bound to an interface. Keyword: ACLs Reported in HBN version: 1.2.0
3158934	Description: Deleting an NVUE user by removing their password file and restarting the `decrypt-user-add` service on the HBN container does not work. Workaround: Either respawn the container after deleting the file, or delete the password file corresponding to the user by running `userdel -r username`. Keyword: User deletion Reported in HBN version: 1.2.0
3185003	Description: When a packet is encapsulated with a VXLAN header, it adds extra bytes which may cause the packet to exceed the MTU of link. Typically, the packet would be fragmented but its silently dropped and no fragmentation happens. Workaround: Make sure that the MTU on the uplink port is always 50 bytes more than host ports so that even after adding VXLAN headers, ingress packets do not exceed the MTU. Keyword: MTU; VXLAN Reported in HBN version: 1.2.0
3184905	Description: On VXLAN encapsulation, the DF flag is not propagated to the outer header. Such a packet may be truncated when forwarded in the kernel, and it may be dropped when hardware offloaded. Workaround: Make sure that the MTU on the uplink port is always 50 bytes more than host ports so that even after adding VXLAN headers, ingress packets do not exceed the MTU. Keyword: VXLAN Reported in HBN version: 1.2.0
3188688	Description: When stopping the container using the command `crictl stop` an error may be reported because the command uses a timeout of 0 which is not enough to stop all the processes in the HBN container. Workaround: Pass a timeout value when stopping the HBN container by running: `crictl stop --timeout 60 <hbn-container>`

	Keyword: Timeout
	Reported in HBN version: 1.2.0
3129749	Description: The same ACL rule cannot be applied in both the inbound and outbound direction on a port.
	Workaround: N/A
	Keyword: ACLs
	Reported in HBN version: 1.2.0
3126560	Description: The system's time zone cannot be modified using NVUE in the HBN container.
	Workaround: The timezone can be manually changed by symlinking the `/etc/localtime` file to a binary time zone's identifier in the `/usr/share/zoneinfo` directory. For example: `sudo ln -sf /usr/share/zoneinfo/GMT /etc/localtime`
	Keyword: Time zone; NVUE
	Reported in HBN version: 1.2.0
3118204	Description: Auto-BGP functionality (where the ASN does not need to be configured but is dynamically inferred by the system based on the system's role as a leaf or spine device) is not supported on HBN.
	Workaround: If BGP is configured and used on HBN, the BGP ASN must be manually configured.
	Keyword: BGP
	Reported in HBN version: 1.2.0
3233088	Description: Since checksum calculation is offloaded to the hardware (not done by the kernel), it is expected to see an incorrect checksum in the tcpdump for locally generated, outgoing packets. BGP keepalives and updates are some of the packets that show such incorrect checksum in tcpdump.
	Workaround: N/A
	Keyword: BGP
	Reported in HBN version: 1.2.0
2821	Description: MAC addresses are not learned in the hardware but only in software. This may affect performance in pure L2 unicast traffic.

785	Workaround: N/A
	Keyword: MAC; L2
	Reported in HBN version: 1.3.0
3017202	Description: Due to disabled backend foundation units, some NVUE commands return `500 INTERNAL SERVER ERROR`/`404 NOT FOUND`. These commands are related to features or subsystems which are not supported on HBN.
	Workaround: N/A
	Keyword: Unsupported NVUE commands
	Reported in HBN version: 1.3.0
2828838	Description: NetworkManager and other services not directly related to HBN may display the following message in syslog: "netlink: read: too many netlink events. Need to resynchronize platform cache" The message has no functional impact and may be ignored.
	Workaround: N/A
	Keyword: Error
	Reported in HBN version: 1.3.0

Bug Fixes

The following table lists the known issues which have been fixed for this release of HBN.

Reference	Description
3610971	Description: The output of the command `nv show interface` does not display information about VRFs, VXLAN, and bridge.
	Fixed in HBN version: 2.0.0
3378928	Description: Service functions (`*_sf`) inside the HBN container are UP at container start irrespective of their presence/absence in the `/etc/network/interfaces` file. But once any of them are added to /e/n/i and later taken off from /e/n/i, they stay DOWN unless added back to /e/n/i.
	Fixed in HBN version: 2.0.0
3452914	Description: IPv6 OOB connectivity from the HBN container stops working if the br-mgmt interface on the DPU goes down. When going down, the br-mgmt interface loses its IPv6 address, which is used as the gateway address for the HBN container. If the br-mgmt interface comes back up, its IPv6 address is not added back and IPv6 OOB connectivity from the HBN container will not work
	Fixed in HBN version: 1.5.0
3191433	Description: ECMP selection for the underlay path uses the ingress port and identifies uplink ports via round robin. This may not result in uniform spread of the traffic.
	Fixed in HBN version: 1.4.0
3049879	Description: When reloading (`ifreload`) an empty `/etc/network/interfaces` file, the previously created interfaces are not deleted.
	Fixed in HBN version: 1.4.0
3284607	Description: When an ACL is configured for IPv4 and L4 parameters (protocol tcp/udp, source, and destination ports) match, the ACL also matches IPv6 traffic with the specified L4 parameters.
	Fixed in HBN version: 1.4.0
3282113	Description: Some DPUs experience an issue with the clock settings after installing a BlueField OS in an HBN setting in which the date reverts back to "Thu Sep 8, 2022".
	Fixed in HBN version: 1.4.0
3354	Description: If interfaces on which BGP unnumbered peering is configured are not defined in the `/etc/network/interfaces` configuration file, BGP peering does

02 9	not get established on them.
	Fixed in HBN version: 1.4.0

NVIDIA DOCA Telemetry Service Guide

This guide provides instructions on how to use the DOCA Telemetry Service (DTS) container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Telemetry Service (DTS) collects data from built-in providers and from external telemetry applications. The following providers are available:

- Data providers:
  - sysfs
  - ethtool
  - tc (traffic control)
- Aggregation providers:
  - fluent_aggr
  - prometheus_aggr

> **Note**
>
> Sysfs provider is enabled by default.

DTS stores collected data into binary files under the `/opt/mellanox/doca/services/telemetry/data` directory. Data write is disabled by default due to BlueField storage restrictions.

DTS can export the data via Prometheus Endpoint (pull) or Fluent Bit (push).

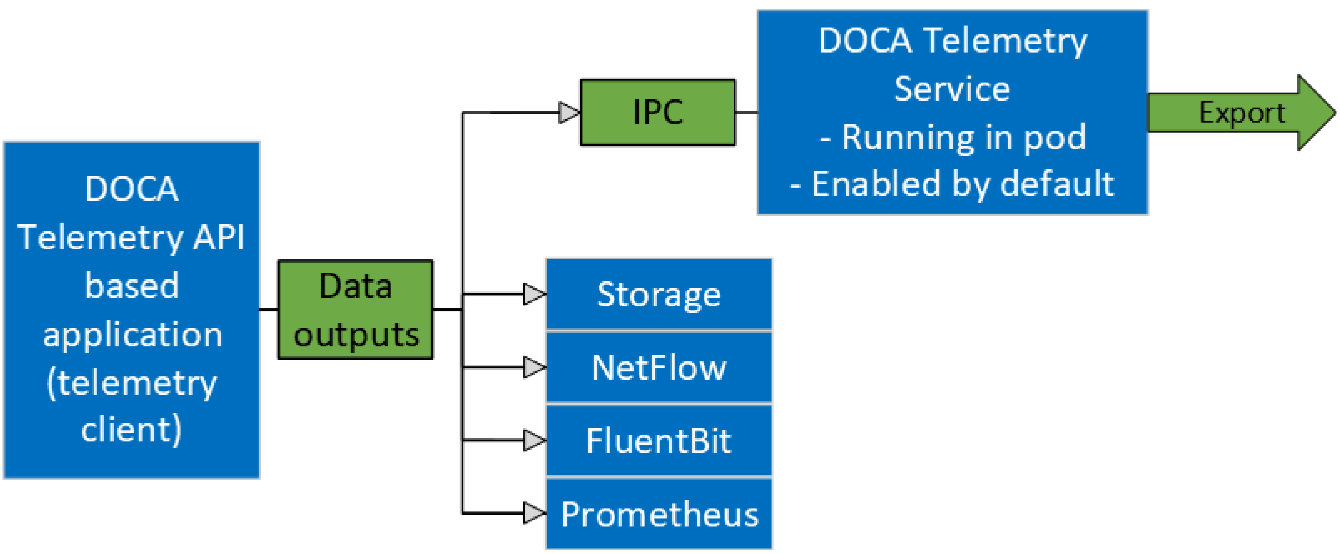
DTS allows exporting NetFlow packets when data is collected from the DOCA Telemetry NetFlow API client application. NetFlow exporter is enabled from `dts_config.ini` by setting NetFlow collector IP/address and port.

Service Deployment

Available Images

Built-in DOCA Service Image

DOCA Telemetry Service is enabled by default on the DPU and is shipped as part of the BlueField image. That is, every BlueField image contains a fixed service version so as to provide out-of-the-box support for programs based on the 2024-10-09_07-10-18_DOCA Telemetry library.



DOCA Service on NGC

In addition to the built-in image shipped with the BlueField boot image, DTS is also available on NGC, NVIDIA's container catalog. This is useful in case a new version of the service has been released and the user wants to upgrade from the built-in image. For service-specific configuration steps and deployment instructions, refer to the service's container page .

Info

For more information about the deployment of DOCA containers on top of the BlueField DPU, refer to NVIDIA DOCA Container Deployment Guide.

DPU Deployment

As mentioned above, DTS starts automatically on BlueField boot. This is done according to the `.yaml` file located at `/etc/kubelet.d/doca_telemetry_standalone.yaml`. Removing the `.yaml` file from this path stops the automatic DTS boot.

DTS files can be found under the directory `/opt/mellanox/doca/services/telemetry/`.

- Container folder mounts:
  - `config`
  - `data`
  - `ipc_sockets`
- Backup files:
  - `doca_telemetry_service_${version}_arm64.tar.gz` – DTS image
  - `doca_telemetry_standalone.yaml` – copy of the default boot `.yaml` file

Host Deployment

DTS supports x86_64 hosts. The providers and exporters all run from a single docker container.

1. Initialize and configure host DTS with:

```
export DTS_IMAGE=nvcr.io/nvidia/doca/doca_telemetry
docker run -v
"/opt/mellanox/doca/services/telemetry/config:/config" --rm -
-name doca-telemetry-init -it $DTS_IMAGE /bin/bash -c
"DTS_CONFIG_DIR=host /usr/bin/telemetry-init.sh"
```

2. Run with:

```
docker run -d --net=host --uts=host --ipc=host
\
              --privileged
\
              --ulimit stack=67108864 --ulimit memlock=-1
\
              --device=/dev/mst/
\
              --device=/dev/infiniband/
\
              --gpus all
\
              -v
"/opt/mellanox/doca/services/telemetry/config:/config"
\
              -v
"/opt/mellanox/doca/services/telemetry/ipc_sockets:/tmp/ipc_so
\
              -v
"/opt/mellanox/doca/services/telemetry/data:/data"
\
              -v "/usr/lib/mft:/usr/lib/mft"
\
              -v "/sys/kernel/debug:/sys/kernel/debug"
\
```

```
              --rm --name doca-telemetry -it $DTS_IMAGE
/usr/bin/telemetry-run.sh
```

> **Note**
>
> The following mounts are required by specific services only:
>
> - `hcaperf` provider:
>   - `--device=/dev/mst/`
>   - `-v "/usr/lib/mft:/usr/lib/mft"`
>   - `-v "/sys/kernel/debug:/sys/kernel/debug"`
> - UCX/RDMA export modes:
>   - `--device=/dev/infiniband/`
> - GPU providers (`nvidia-smi` and `dcgm`):
>   - `--gpu all`

Deployment with Grafana Monitoring

Refer to section "Deploying with Grafana Monitoring".

Configuration

The configuration of DTS is placed under `/opt/mellanox/doca/services/telemetry/config` by DTS during initialization. The user can interact with the `dts_config.ini` file and `fluent_bit_configs` folder.

`dts_config.ini` contains the main configuration for the service and must be used to enable/disable providers, exporters, data writing. More details are provided in the corresponding sections. For every update in this file, DST must be restarted. Interaction with `fluent_bit_configs` folder is described in section Fluent Bit.

Init Scripts

The `InitContainers` section of the `.yaml` file has 2 scripts for config initialization:

- `/usr/bin/telemetry-init.sh` – generates the default configuration files if, and only if, the `/opt/mellanox/doca/services/telemetry/config` folder is empty.
- `/usr/bin/enable-fluent-forward.sh` – configures the destination host and port for Fluent Bit forwarding. The script requires that both the host and port are present, and only in this case it would start. The script overwrites the `/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs` folder and configures the `.exp` file.

Enabling Fluent Bit Forwarding

To enable Fluent Bit forward, add the destination host and port to the command line found in the `initContainers` section of the `.yaml` file:

```
command: ["/bin/bash", "-c", "/usr/bin/telemetry-init.sh &&
/usr/bin/enable-fluent-forward.sh -i=127.0.0.1 -p=24224"]
```

Note

The host and port shown above are just an example. See section Fluent Bit to learn about manual configuration.

Generating Configuration

The configuration folder `/opt/mellanox/doca/services/telemetry/config` starts empty by default. Once the service starts, the initial scripts run as a part of the initial container and create configuration as described in section Enabling Fluent Bit Forwarding.

Resetting Configuration

Resetting the configuration can be done by deleting the content found in the configuration folder and restarting the service to generate the default configuration.

Enabling Providers

Providers are enabled from the `dts_config.ini` configuration file. Uncomment the `enable-provider=$provider-name` line to allow data collection for this provider. For example, uncommenting the following line enables the `ethtool` provider:

```
#enable-provider=ethtool
```

> **Note**
>
> More information about telemetry providers can be found under the Providers section.

Remote Collection

Certain providers or components are unable to execute properly within the container due to various container limitations. Therefore, they would have to perform remote collection or execution.

The following steps enable remote collection:

1. Activate DOCA privileged executer (DPE), as DPE is the means by which remote collection is achieved:

```
systemctl start dpe
```

2. Add `grpc` before `provider-name` (i.e., `enable-provider=grpc.$provider-name`). For example, the following line configures remote collection of the `hcaperf` provider:

```
enable-provider=grpc.hcaperf
```

3. If there are any configuration lines that are provider-specific, then add the `grpc` prefix as well. Building upon the previous example:

```
grpc.hcaperf.mlx5_0=sample
grpc.hcaperf.mlx5_1=sample
```

Enabling Data Write

Uncomment the following line in `dts_config.ini`:

```
#output=/data
```

> **Note**

Changes in `dts_config.ini` force the main DTS process to restart in 60 seconds to apply the new settings.

Enabling IPC with Non-container Program

For information on enabling IPC between DTS and an application that runs outside of a container, refer to section "Using IPC with Non-container Application" in the 2024-10-09_07-10-18_DOCA Telemetry.

Description

Providers

DTS supports on-board data collection from `sysf`, `ethtool`, and `tc` providers.

Fluent and Prometheus aggregator providers can collect the data from other applications.

Sysfs Counters List

The sysfs provider has several components: `ib_port`, `hw_port`, `mr_cache`, `eth`, `hwmon` and `bf_ptm` . By default, all the components (except `bf_ptm`) are enabled when the provider is enabled:

```
#disable-provider=sysfs
```

The components can be disabled separately. For instance, to disable `eth`:

```
enable-provider=sysfs
disable-provider=sysfs.eth
```

Note

`ib_port` and `ib_hvw` are state counters which are collected per port. These counters are only collected for ports whose state is active.

- `ib_port` counters:

```
{hca_name}:{port_num}:ib_port_state
{hca_name}:{port_num}:VL15_dropped
{hca_name}:{port_num}:excessive_buffer_overrun_errors
{hca_name}:{port_num}:link_downed
{hca_name}:{port_num}:link_error_recovery
{hca_name}:{port_num}:local_link_integrity_errors
{hca_name}:{port_num}:multicast_rcv_packets
{hca_name}:{port_num}:multicast_xmit_packets
{hca_name}:{port_num}:port_rcv_constraint_errors
{hca_name}:{port_num}:port_rcv_data
{hca_name}:{port_num}:port_rcv_errors
{hca_name}:{port_num}:port_rcv_packets
{hca_name}:{port_num}:port_rcv_remote_physical_errors
{hca_name}:{port_num}:port_rcv_switch_relay_errors
{hca_name}:{port_num}:port_xmit_constraint_errors
{hca_name}:{port_num}:port_xmit_data
{hca_name}:{port_num}:port_xmit_discards
{hca_name}:{port_num}:port_xmit_packets
{hca_name}:{port_num}:port_xmit_wait
{hca_name}:{port_num}:symbol_error
{hca_name}:{port_num}:unicast_rcv_packets
{hca_name}:{port_num}:unicast_xmit_packets
```

- `ib_hw` counters:

```
{hca_name}:{port_num}:hw_state
{hca_name}:{port_num}:hw_duplicate_request
{hca_name}:{port_num}:hw_implied_nak_seq_err
{hca_name}:{port_num}:hw_lifespan
{hca_name}:{port_num}:hw_local_ack_timeout_err
{hca_name}:{port_num}:hw_out_of_buffer
{hca_name}:{port_num}:hw_out_of_sequence
{hca_name}:{port_num}:hw_packet_seq_err
{hca_name}:{port_num}:hw_req_cqe_error
{hca_name}:{port_num}:hw_req_cqe_flush_error
{hca_name}:{port_num}:hw_req_remote_access_errors
{hca_name}:{port_num}:hw_req_remote_invalid_request
{hca_name}:{port_num}:hw_resp_cqe_error
{hca_name}:{port_num}:hw_resp_cqe_flush_error
{hca_name}:{port_num}:hw_resp_local_length_error
{hca_name}:{port_num}:hw_resp_remote_access_errors
{hca_name}:{port_num}:hw_rnr_nak_retry_err
{hca_name}:{port_num}:hw_rx_atomic_requests
{hca_name}:{port_num}:hw_rx_dct_connect
{hca_name}:{port_num}:hw_rx_icrc_encapsulated
{hca_name}:{port_num}:hw_rx_read_requests
{hca_name}:{port_num}:hw_rx_write_requests
```

- `ib_mr_cache` counters:

```
{hca_name}:mr_cache:size_{n}:cur
{hca_name}:mr_cache:size_{n}:limit
{hca_name}:mr_cache:size_{n}:miss
{hca_name}:mr_cache:size_{n}:size
```

Note

Where `n` ranges from 0 to 24.

- `eth` counters:

```
{hca_name}:{device_name}:eth_collisions
{hca_name}:{device_name}:eth_multicast
{hca_name}:{device_name}:eth_rx_bytes
{hca_name}:{device_name}:eth_rx_compressed
{hca_name}:{device_name}:eth_rx_crc_errors
{hca_name}:{device_name}:eth_rx_dropped
{hca_name}:{device_name}:eth_rx_errors
{hca_name}:{device_name}:eth_rx_fifo_errors
{hca_name}:{device_name}:eth_rx_frame_errors
{hca_name}:{device_name}:eth_rx_length_errors
{hca_name}:{device_name}:eth_rx_missed_errors
{hca_name}:{device_name}:eth_rx_nohandler
{hca_name}:{device_name}:eth_rx_over_errors
{hca_name}:{device_name}:eth_rx_packets
{hca_name}:{device_name}:eth_tx_aborted_errors
{hca_name}:{device_name}:eth_tx_bytes
{hca_name}:{device_name}:eth_tx_carrier_errors
{hca_name}:{device_name}:eth_tx_compressed
{hca_name}:{device_name}:eth_tx_dropped
{hca_name}:{device_name}:eth_tx_errors
{hca_name}:{device_name}:eth_tx_fifo_errors
{hca_name}:{device_name}:eth_tx_heartbeat_errors
{hca_name}:{device_name}:eth_tx_packets
{hca_name}:{device_name}:eth_tx_window_errors
```

- BlueField-2 `hwmon` counters:

```
{hwmon_name}:{l3cache}:CYCLES
{hwmon_name}:{l3cache}:HITS_BANK0
{hwmon_name}:{l3cache}:HITS_BANK1
{hwmon_name}:{l3cache}:MISSES_BANK0
{hwmon_name}:{l3cache}:MISSES_BANK1
{hwmon_name}:{pcie}:IN_C_BYTE_CNT
{hwmon_name}:{pcie}:IN_C_PKT_CNT
{hwmon_name}:{pcie}:IN_NP_BYTE_CNT
{hwmon_name}:{pcie}:IN_NP_PKT_CNT
{hwmon_name}:{pcie}:IN_P_BYTE_CNT
{hwmon_name}:{pcie}:IN_P_PKT_CNT
{hwmon_name}:{pcie}:OUT_C_BYTE_CNT
{hwmon_name}:{pcie}:OUT_C_PKT_CNT
{hwmon_name}:{pcie}:OUT_NP_BYTE_CNT
{hwmon_name}:{pcie}:OUT_NP_PKT_CNT
{hwmon_name}:{pcie}:OUT_P_PKT_CNT
{hwmon_name}:{tile}:MEMORY_READS
{hwmon_name}:{tile}:MEMORY_WRITES
{hwmon_name}:{tile}:MSS_NO_CREDIT
{hwmon_name}:{tile}:VICTIM_WRITE
{hwmon_name}:{tilenet}:CDN_DIAG_C_OUT_OF_CRED
{hwmon_name}:{tilenet}:CDN_REQ
{hwmon_name}:{tilenet}:DDN_REQ
{hwmon_name}:{tilenet}:NDN_REQ
{hwmon_name}:{trio}:TDMA_DATA_BEAT
{hwmon_name}:{trio}:TDMA_PBUF_MAC_AF
{hwmon_name}:{trio}:TDMA_RT_AF
{hwmon_name}:{trio}:TPIO_DATA_BEAT
{hwmon_name}:{triogen}:TX_DAT_AF
{hwmon_name}:{triogen}:TX_DAT_AF
```

- BlueField-3 `hwmon` counters:

```
{hwmon_name}:{llt}:GDC_BANK0_RD_REQ
{hwmon_name}:{llt}:GDC_BANK1_RD_REQ
{hwmon_name}:{llt}:GDC_BANK0_WR_REQ
{hwmon_name}:{llt}:GDC_BANK1_WR_REQ
{hwmon_name}:{llt_miss}:GDC_MISS_MACHINE_RD_REQ
{hwmon_name}:{llt_miss}:GDC_MISS_MACHINE_WR_REQ
{hwmon_name}:{mss}:SKYLIB_DDN_TX_FLITS
{hwmon_name}:{mss}:SKYLIB_DDN_RX_FLITS
```

- BlueField-3 `bf_ptm` counters:

```
bf:ptm:active_power_profile
bf:ptm:atx_power_available
bf:ptm:core_temp
bf:ptm:ddr_temp
bf:ptm:error_state
bf:ptm:power_envelope
bf:ptm:power_throttling_event_count
bf:ptm:power_throttling_state
bf:ptm:thermal_throttling_event_count
bf:ptm:thermal_throttling_state
bf:ptm:throttling_state
bf:ptm:total_power
bf:ptm:vr0_power
bf:ptm:vr1_power
```

Power Thermal Counters

The `bf_ptm` component collects BlueField-3 power thermal counters using remote collection. It is disabled by default and can be enabled as follows:

1. Load kernel module `mlxbf-ptm`:

```
modprobe -v mlxbf-ptm
```

2. Enable component using remote collection:

```
enable-provider=grpc.sysfs.bf_ptm
```

> **Note**
>
> DPE server should be active before changing the `dts_config.ini` file. See section "Remote Collection" for details.

Ethtool Counters

Ethtool counters is the generated list of counters which corresponds to Ethtool utility. Counters are generated on a per-device basis. See this community post for more information on mlx5 ethtool counters.

Traffic Control Info

The following TC objects are supported and reported regarding the ingress filters:

- Filters
  - flower
- Actions

    - mirred
    - tunnel_key

The info is provided as one of the following events:

- Basic filter event
- Flower/IPv4 filter event
- Flower/IPv6 filter event
- Basic action event
- Mirred action event
- Tunnel_key/IPv4 action event
- Tunnel_key/IPv6 action event

General notes:

- Actions always belong to a filter, so action events share the filter event's ID via the `event_id` data member
- Basic filter event only contains textual *kind* (so users can see which real life objects' support they are lacking)
- Basic action event only contains textual *kind* and some basic common statistics if available

Fluent Aggregator

`fluent_aggr` listens on a port for Fluent Bit Forward protocol input connections. Received data can be streamed via a Fluent Bit exporter.

The default port is 42442. This can be changed by updating the following option:

```
fluent-aggr-port=42442
```

Prometheus Aggregator

`prometheus_aggr` polls data from a list of Prometheus endpoints.

Each endpoint is listed in the following format:

```
prometheus_aggr_endpoint.{N}={host_name},{host_port_url},
{poll_inteval_msec}
```

Where N starts from 0.

Aggregated data can be exported via a Prometheus Aggr Exporter endpoint.

Network Interfaces

`ifconfig` collects network interface data. To enable, set:

```
enable-provider=ifconfig
```

If the Prometheus endpoint is enabled, add the following configuration to cache every collected network interface and arrange the index according to their names:

```
prometheus-fset-indexes=name
```

Metrices are collected for each network interface as follows:

```
name
rx_packets
tx_packets
```

```
rx_bytes
tx_bytes
rx_errors
tx_errors
rx_dropped
tx_dropped
multicast
collisions
rx_length_errors
rx_over_errors
rx_crc_errors
rx_frame_errors
rx_fifo_errors
rx_missed_errors
tx_aborted_errors
tx_carrier_errors
tx_fifo_errors
tx_heartbeat_errors
tx_window_errors
rx_compressed
tx_compressed
rx_nohandler
```

HCA Performance

`hcaperf` collects HCA performance data. Since it requires access to an RDMA device, it must use remote collection on the DPU. On the host, the user runs the container in privileged mode and RDMA device mount.

The counter list is device dependent.

hcaperf DPU Configuration

To enable `hcaperf` in remote collection mode, set:

```
enable-provider=grpc.hcaperf

# specify HCAs to sample
grpc.hcaperf.mlx5_0=sample
grpc.hcaperf.mlx5_1=sample
```

> **Note**
>
> DPE server should be active before changing the `dts_config.ini` file. See section "Remote Collection" for details.

hcaperf Host Configuration

To enable `hcaperf` in regular mode, set:

```
enable-provider=hcaperf

# specify HCAs to sample
hcaperf.mlx5_0=sample
hcaperf.mlx5_1=sample
```

NVIDIA System Management Interface

The `nvidia-smi` provider collects GPU and GPU process information provided by the NVIDIA system management interface.

This provider is supported only on x86_64 hosts with installed GPUs. All GPU cards supported by `nvidia-smi` are supported by this provider.

The counter list is GPU dependent. Additionally, per-process information is collected for the first 20 (by default) `nvidia_smi_max_processes` processes.

Counters can be either collected as string data "as is" in `nvidia-smi` or converted to numbers when `nvsmi_with_numeric_fields` is set.

To enable `nvidia-smi` provider and change parameters, set:

```
enable-provider=nvidia-smi

# Optional parameters:
#nvidia_smi_max_processes=20
#nvsmi_with_numeric_fields=1
```

NVIDIA Data Center GPU Manager

The `dcgm` provider collects GPU information provided by the NVIDIA data center GPU manager (DCGM) API.

This provider is supported only on x86_64 hosts with installed GPUs, and requires running the `nv-hostengine` service (refer to DCGM documentation for details).

DCGM counters are split into several groups by context:

- GPU – basic GPU information (always)
- COMMON – common fields that can be collected from all devices
- PROF – profiling fields
- ECC – ECC errors
- NVLINK / NVSWITCH / VGPU – fields depending on the device type

To enable DCGM provider and counter groups, set:

```
enable-provider=dcgm

dcgm_events_enable_common_fields=1
#dcgm_events_enable_prof_fields=0
#dcgm_events_enable_ecc_fields=0
#dcgm_events_enable_nvlink_fields=0
#dcgm_events_enable_nvswitch_fields=0
#dcgm_events_enable_vgpu_fields=0
```

Data Outputs

DTS can send the collected data to the following outputs:

- Data writer (saves binary data to disk)
- Fluent Bit (push-model streaming)
- Prometheus endpoint (keeps the most recent data to be pulled)

Data Writer

The data writer is disabled by default to save space on BlueField. Steps for activating data write during debug can be found under section Enabling Data Write.

The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example (`apt install tree`):

```
tree /opt/mellanox/doca/services/telemetry/data/
/opt/mellanox/doca/services/telemetry/data/
    {year}
        {mmdd}
             {hash}
                  {source_id}
```

```
                        {source_tag}{timestamp}.bin
                    {another_source_id}
                         {another_source_tag}{timestamp}.bin
    schema
        schema_{MD5_digest}.json
```

New binary files appears when the service starts or when binary file age/size restriction is reached. If no schema or no data folders are present, refer to the Troubleshooting section.

> **Note**
>
> `source_id` is usually set to the machine hostname. `source_tag` is a line describing the collected counters, and it is often set as the provider's name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```
crictl exec -it <Container ID>
/opt/mellanox/collectx/bin/clx_read -s /data/schema
/data/path/to/datafile.bin
```

> **Note**
>
> The path to the data file must be an absolute path.

Example output:

```
{
    "timestamp": 1634815738799728,
    "event_number": 0,
    "iter_num": 0,
    "string_number": 0,
    "example_string": "example_str_1"
}
{
    "timestamp": 1634815738799768,
    "event_number": 1,
    "iter_num": 0,
    "string_number": 1,
    "example_string": "example_str_2"
}
…
```

Prometheus

The Prometheus endpoint keeps the most recent data to be pulled by the Prometheus server and is enabled by default.

To check that data is available, run the following command on BlueField:

```
curl -s http://0.0.0.0:9100/metrics
```

The command dumps every counter in the following format:

```
counter_name {list of meta fields} counter_value timestamp
```

Additionally, endpoint supports JSON and CSV formats:

```
curl -s http://0.0.0.0:9100/json/metrics
curl -s http://0.0.0.0:9100/csv/metrics
```

Note

The default port for Prometheus can be changed in `dts_config.ini`.

Configuration Details

Prometheus is configured as a part of `dts_config.ini`.

By default, the Prometheus HTTP endpoint is set to port 9100. Comment this line out to disable Prometheus export.

```
prometheus=http://0.0.0.0:9100
```

Prometheus can use the data field as an index to keep several data records with different index values. Index fields are added to Prometheus labels.

```
# Comma-separated counter set description for Prometheus
indexing:
#prometheus-indexes=idx1,idx2

# Comma-separated fieldset description for prometheus indexing
#prometheus-fset-indexes=idx1,idx2
```

The default `fset` index is `device_name`. It allows Prometheus to keep ethtool data up for both the `p0` and `p1` devices.

```
prometheus-fset-indexes=device_name
```

If `fset` index is not set, the data from `p1` overwrites `p0`'s data.

For quick name filtering, the Prometheus exporter supports being provided with a comma-separated list of counter names to be ignored:

```
#prometheus-ignore-names=counter_name1,counter_name_2
```

For quick filtering of data by tag, the Prometheus exporter supports being provided with a comma-separated list of data source tags to be ignored.

Users should add tags for all streaming data since the Prometheus exporter cannot be used for streaming. By default, `FI_metrics` are disabled.

```
prometheus-ignore-tags=FI_metrics
```

Prometheus Aggregator Exporter

Prometheus aggregator exporter is an endpoint that keeps the latest aggregated data using `prometheus_aggr`.

This exporter labels data according to its source.

To enable this provider, users must set 2 parameters in `dts_config.ini`:

```
prometheus-aggr-exporter-host=0.0.0.0
```

```
prometheus-aggr-exporter-port=33333
```

Fluent Bit

Fluent Bit allows streaming to multiple destinations. Destinations are configured in `.exp` files that are documented in-place and can be found under:

```
/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs
```

Fluent Bit allows exporting data via "Forward" protocol which connects to the Fluent Bit/FluentD instance on customer side.

Export can be enabled manually:

1. Uncomment the line with `fluent_bit_configs=…` in `dts_config.ini`.
2. Set `enable=1` in required `.exp` files for the desired plugins.
3. Additional configurations can be set according to instructions in the `.exp` file if needed.
4. Restart the DTS.
5. Set up receiving instance of Fluent Bit/FluentD if needed.
6. See the data on the receiving side.

Export file destinations are set by configuring `.exp` files or creating new ones. It is recommended to start by going over documented example files. Documented examples exist for the following supported plugins:

- forward
- file
- stdout
- kafka

- es (elastic search)
- influx

> **Note**
>
> All `.exp` files are disabled by default if not configured by `initContainer` entry point through `.yaml` file.

> **Note**
>
> To forward the data to several destinations, create several `forward_{num}.exp` files. Each of these files must have their own destination host and port.

Export File Configuration Details

Each export destination has the following fields:

- `name` – configuration name
- `plugin_name` – Fluent Bit plugin name
- `enable` – 1 or 0 values to enable/disable this destination
- `host` – the host for Fluent Bit plugin
- `port` – port for Fluent Bit plugin
- `msgpack_data_layout` – the msgpacked data format. Default is `flb_std`. The other option is custom. See section Msgpack Data Layout for details.
- `plugin_key=val` – key-value pairs of Fluent Bit plugin parameter (optional)

- `counterset`/`fieldset` – file paths (optional). See details in section Cset/Fset Filtering.
- `source_tag=source_tag1,source_tag2` – comma-separated list of data page source tags for filtering. The rest tags are filtered out during export. Event tags are event provider names. All counters can be enabled/disabled only simultaneously with a `counters` keyword.

> **Note**
>
> Use `#` to comment a configuration line.

Msgpack Data Layout

Data layout can be configured using `.exp` files by setting `msgpack_data_layout=layout`. There are two available layouts: Standard and Custom.

The standard `flb_std` data layout is an array of 2 fields:

- timestamp double value
- a plain dictionary (key-value pairs)

The standard layout is appropriate for all Fluent Bit plugins. For example:

```
[timestamp_val, {"timestamp"->ts_val, type=>"counters/events",
"source"=>"source_val", "key_1"=>val_1, "key_2"=>val_2,...}]
```

The custom data layout is a dictionary of meta-fields and counter fields. Values are placed into a separate plain dictionary. Custom data format can be dumped with `stdout_raw` output plugin of Fluent-Bit installed or can be forwarded with `forward` output plugin.

Counters example:

```
{"timestamp"=>timestamp_val, "type"=>"counters",
"source"=>"source_val", "values"=> {"key_1"=>val_1,
"key_2"=>val_2,...}}
```

Events example:

```
{"timestamp"=>timestamp_val, "type"=>"events",
"type_name"=>"type_name_val", "source"=>" source_val", "values"=>
{"key_1"=>val_1, "key_2"=>val_2,...}}
```

Cset/Fset Filtering

Each export file can optionally use one `cset` and one `fset` file to filter UFM telemetry counters and events data.

- `cset` contains tokens per line to filter data with `"type"="counters"`.
- `fset` contains several blocks started with the header line `[event_type_name]` and tokens under that header. An Fset file is used to filter data with `"type"="events"`.

> **Note**
>
> Event type names could be prefixed to apply the same tokens to all fitting types. For example, to filter all ethtool events, use `[ethtool_event_*]`.

If several tokens must be matched simultaneously, use `<tok1>+<tok2>+<tok3>`. Exclusive tokens are available as well. For example, the line

`<tok1>+<tok2>-<tok3>-<tok4>` filters names that match both tok1 and tok2 and do not match tok3 or tok4.

The following are the details of writing `cset` files:

```
# Put tokens on separate lines
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port
# port+xmit # match names that contain both tokens "port" and
"xmit"
# port-support # match names that contain the token "port" and do
not match the "-" token "support"
#
# Tip: To disable counter export put a single token line that
fits nothing
```

The following are the details of writing `fset` files:

```
# Put your events here
# Usage:
#
# [type_name_1]
# tokens
# [type_name_2]
# tokens
# [type_name_3]
# tokens
# ...
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
```

```
# ^port$ # include name that is exact token "port
# port+xmit # match names that contain both tokens "port" and
"xmit"
# port-support # match names that contain the token "port" and do
not match the "-" token "support"

# The next example will export all the "tc" events and all events
with type prefix "ethtool_" "ethtool" are filtered with token
"port":
# [tc]
#
# [ethtool_*]
# packet

# To know which event type names are available check export and
find field "type_name"=>"ethtool_event_p0"
# ...
# Corner cases:
# 1. Empty fset file will export all events.
# 2. Tokens written above/without [event_type] will be ignored.
# 3. If cannot open fset file, warning will be printed, all event
types will be exported.
```

NetFlow Exporter

NetFlow exporter must be used when data is collected as NetFlow packets from the telemetry client applications. In this case, DOCA Telemetry NetFlow API sends NetFlow data packages to DTS via IPC. DTS uses NetFlow exporter to send data to the NetFlow collector (3rd party service).

To enable NetFlow exporter, set `netflow-collector-ip` and `netflow-collector-port` in `dts_config.ini`. `netflow-collector-ip` could be set either to IP or an address.

For additional information, refer to the `dts_config.ini` file.

DOCA Privileged Executer

DOCA Privileged Executer (DPE) is a daemon that allows specific DOCA services (DTS included) to access BlueField information that is otherwise inaccessible from a container due to technology limitations or permission granularity issues.

When enabled, DPE enriches the information collected by DTS. However, DTS can still be used if DPE is disabled (default).

DPE Usage

DPE is controlled by systemd, and can be used as follows:

- To check DPE status:

  ```
  sudo systemctl status dpe
  ```

- To start DPE:

  ```
  sudo systemctl start dpe
  ```

- To stop DPE:

  ```
  sudo systemctl stop dpe
  ```

DPE logs can be found in `/var/log/doca/telemetry/dpe.log`.

DPE Configuration File

DPE can be configured by the user. This section covers the syntax and implications of its configuration file.

> **Note**
>
> The DPU telemetry collected by DTS does not require for this configuration file to be used.

The DPE configuration file allows users to define the set of commands that DPE should support. This may be done by passing the `-f` option in the following line of `/etc/systemd/system/dpe.service`:

```
ExecStart=/opt/mellanox/doca/services/telemetry/dpe/bin/dpeserver
-vvv
```

To use the configuration file:

```
ExecStart=/opt/mellanox/doca/services/telemetry/dpe/bin/dpeserver
-vvv -f /path/to/dpe_config.ini
```

The configuration file supports the following sections:

- `[server]` - list of key=value lines for general server configuration. Allowed keys: `socket`.
- `[commands]` - list of bash command lines that are not using custom RegEx
- `[commands_regex]` - list of bash command lines that are using custom RegEx
- `[regex_macros]` - custom RegEx definitions used in the `commands_regex` section

Consider the following example configuration file:

```
[server]
socket=/tmp/dpe.sock

[commands]
hostname
cat /etc/os-release

[commands_regex]
crictl inspect $HEXA       # resolved as "crictl inspect [a-f0-9]+"
lspci $BDF                 # resolved as "lspci ([0-9a-f]{4}\:|)[0-9a-f]{2}\:[0-9a-f]{2}\.[0-9a-f]"

[regex_macros]
HEXA=[a-f0-9]+
BDF=([0-9a-f]{4}\:|)[0-9a-f]{2}\:[0-9a-f]{2}\.[0-9a-f]
```

> **Note**
>
> DPE is shipped with a preconfigured file that matches the commands used by the standalone DTS version included in the same DOCA installation. The file is located in `/opt/mellanox/doca/services/telemetry/dpe/etc/dpe_config.ini`.

> **Note**
>
> Using a DPE configuration file allows for a fine-grained control over the interface exposed by it to the rest of the DOCA services. However, even when using the pre-supplied configuration file mentioned above,

one should remember that it has been configured to match a fixed DTS version. That is, replacing the standalone DTS version with a new one downloaded from NGC means that the used configuration file might not cover additional features added in the new DTS version.

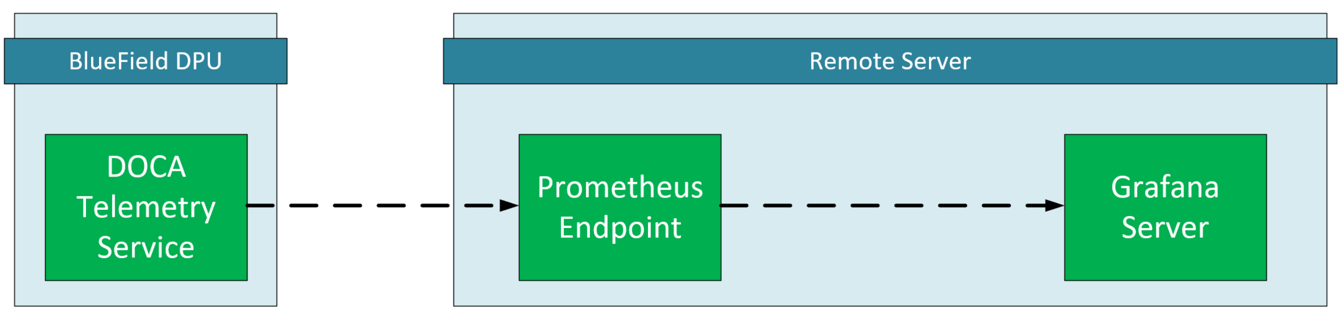
Deploying with Grafana Monitoring

This chapter provides an overview and deployment configuration of DOCA Telemetry Service with Grafana .

Grafana Deployment Prerequisites

- BlueField DPU running DOCA Telemetry Service.
- Optional remote server to host Grafana and Prometheus.
- Prometheus installed on the host machine. Please refer to the Prometheus website for more information.
- Grafana installed on the host machine. Please refer to Grafana Labs website for more information.

Grafana Deployment Configuration



DTS Configuration (DPU Side)

DTS will be configured to export the sysfs counter using the Prometheus plugin.

Note

Sysfs is used as an example, other counters are available. Please refer to the NVIDIA DOCA Telemetry Service Guide for more information.

1. Make sure the sysfs counter is enabled.

```
vim
/opt/mellanox/doca/services/telemetry/config/dts_config.ini

enable-provider=sysfs
```

2. Enable Prometheus exporter by setting the `prometheus` address and port.

```
vim
/opt/mellanox/doca/services/telemetry/config/dts_config.ini

prometheus=http://0.0.0.0:9100
```

Note

In this example, the Prometheus plugin exports data on localhost port 9100, this is an arbitrary value and can changed.

Note

DTS must be restarted to apply changes.

Prometheus Configuration (Remote Server)

Please download Prometheus for your platform.

Prometheus is configured via command-line flags and a configuration file, `prometheus.yml`.

1. Open the `prometheus.yml` file and configure the DPU as the endpoint target.

```
vim prometheus.yml
# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
- targets: ["<dpu-ip>:<prometheus-port>"]
```

   Where:

   - `<dpu-ip>` is the DPU IP address. Prometheus reaches to this IP to pull data.
   - `<prometheus-port>` the exporter port that set in DTS configuration.

2. Run Prometheus server:

```
./prometheus --config.file="prometheus.yml"
```

Tip

Prometheus services are available as Docker images. Please refer to Using Docker in Prometheus' Installation guide.

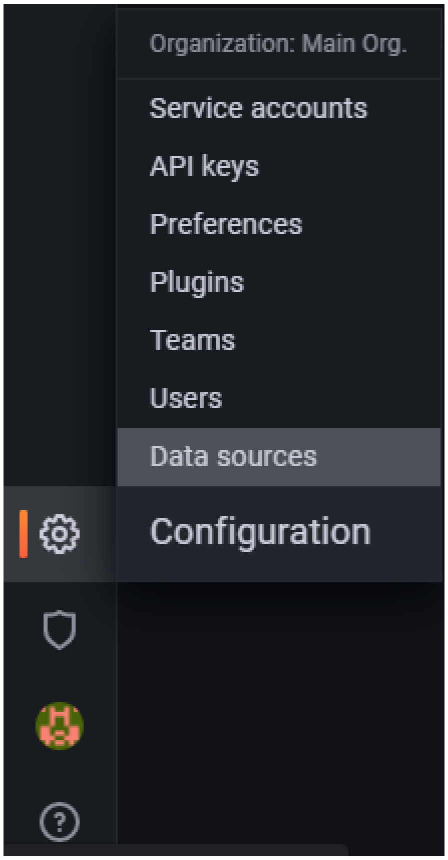
Grafana Configuration (Remote Server)

Please download and install Grafana for your platform.

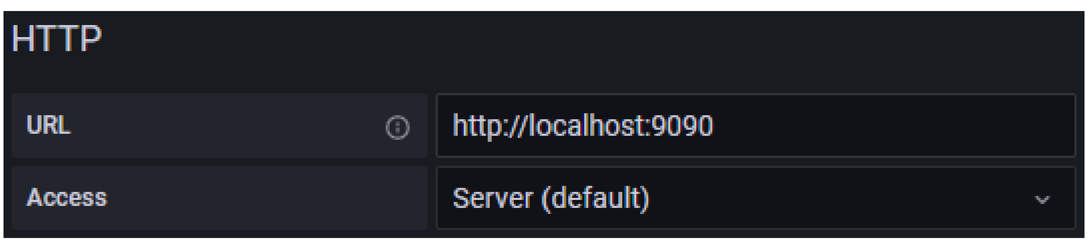
1. Setup Grafana. Please refer to Install Grafana guide in Grafana documentation.
2. Log into the Grafana dashboard at http://localhost:3000.

> **Note**
>
> Port 3000 is the default port number set by Grafana. This can be changed if needed. The default credentials are admin/admin.

3. Add Prometheus as data source by navigating to Settings → Data sources → Add data source → Prometheus.



4. Configure the Prometheus data source. Under the HTTP section, set the Prometheus server address.



> **Note**
>
> The Prometheus server's default listen port is 9090. Prometheus and Grafana are both running on the same server, thus the address is localhost.

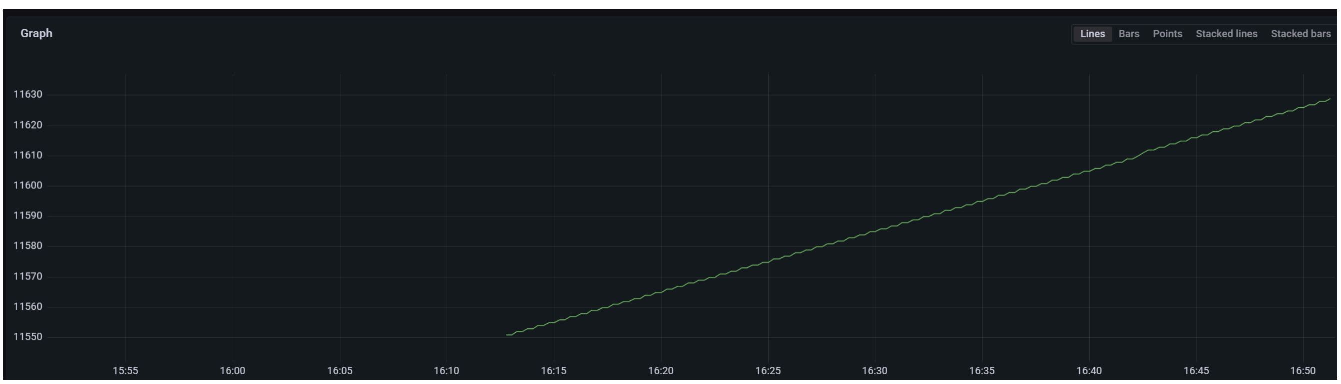
5. Save and test.

Exploring Telemetry Data

Go to the Explore page on the left-hand side, and choose a Prometheus provider.

Choose a metric to display and specify a label. The label can be used to filter out data based on the source and HCA devices.

Graph display after selecting a metric and specifying a label to filter by:



Troubleshooting

On top of the Troubleshooting section in the NVIDIA DOCA Container Deployment Guide, here are additional troubleshooting tips for DTS:

- For general troubleshooting, refer to the NVIDIA DOCA Troubleshooting Guide.
- If the pod's state fails to be marked as "Ready", refer to `/var/log/syslog`.
- Check if the service is configured to write data to the disk as this may cause the system to run out of disk space.

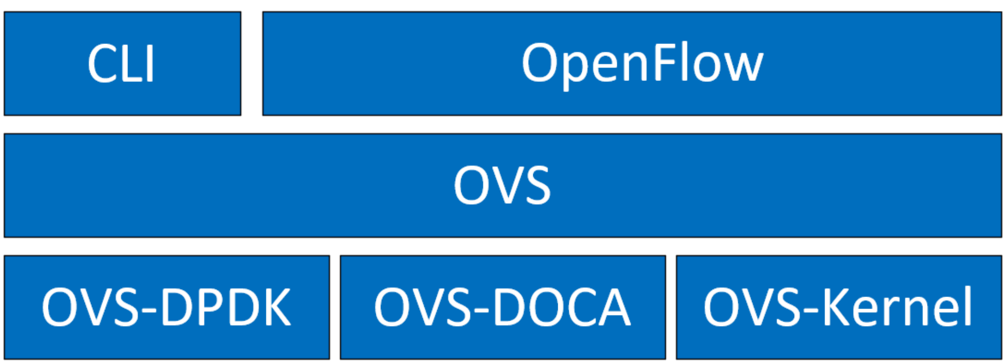
- If a PIC bus error occurs, configure the following files inside the container:

```
crictl exec -it <container-id> /bin/bash
# Add to /config/clx.env the following line:
"
export UCX_TLS=tcp
"
```

OpenvSwitch Offload

Open vSwitch (OVS) is a software-based network technology that enhances virtual machine (VM) communication within internal and external networks. Typically deployed in the hypervisor, OVS employs a software-based approach for packet switching, which can strain CPU resources, impacting system performance and network bandwidth utilization. Addressing this, NVIDIA's Accelerated Switching and Packet Processing (ASAP^2) technology offloads OVS data-plane tasks to specialized hardware, like the embedded switch (eSwitch) within the NIC subsystem, while maintaining an unmodified OVS control-plane. This results in notably improved OVS performance without burdening the CPU.

NVIDIA's OVS architecture extends the traditional OVS-DPDK and OVS-Kernel data-path offload interfaces, introducing OVS-DOCA as an additional implementation. OVS-DOCA, built upon NVIDIA's networking API, preserves the same interfaces as OVS-DPDK and OVS-Kernel while utilizing the DOCA Flow library. Unlike the other modes, OVS-DOCA exploits unique hardware offload mechanisms and application techniques, maximizing performance and features for NVIDA NICs and DPUs. This mode is especially efficient due to its architecture and DOCA library integration, enhancing e-switch configuration and accelerating hardware offloads beyond what the other modes can achieve.



NVIDIA OVS installation contains all three OVS flavors. The following subsections describe the three flavors (default is OVS-Kernel) and how to configure each of them.

OVS and Virtualized Devices

When OVS is combined with NICs and DPUs (such as NVIDIA® ConnectX®-6 Lx/Dx and NVIDIA® BlueField®-2 and later), it utilizes the hardware data plane of ASAP^2. This data plane can establish connections to VMs using either SR-IOV virtual functions (VFs) or virtual host data path acceleration (vDPA) with virtio.

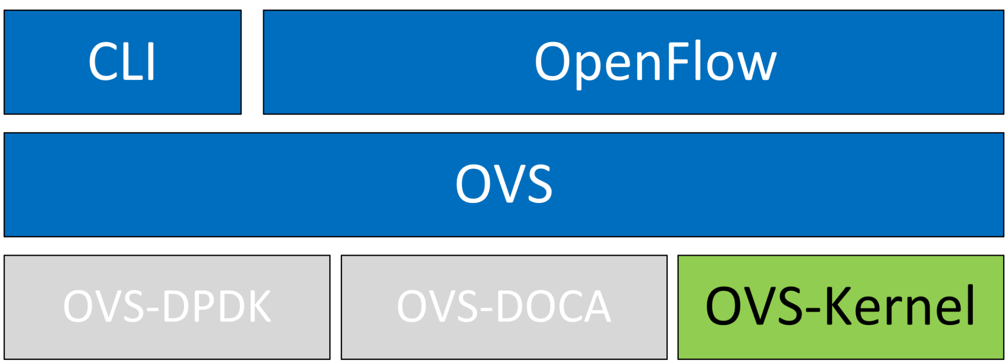
In both scenarios, an accelerator engine within the NIC accelerates forwarding and offloads the OVS rules. This integrated solution accelerates both the infrastructure (via VFs through SR-IOV or virtio) and the data plane. For DPUs (which include a NIC subsystem), an alternate virtualization technology implements full virtio emulation within the DPU, enabling the host server to communicate with the DPU as a software virtio device.

- When using ASAP^2 data plane over SR-IOV virtual functions (VFs), the VF is directly passed through to the VM, with the NVIDIA driver running within the VM.
- When using vDPA, the vDPA driver allows VMs to establish their connections through VirtIO. As a result, the data plane is established between the SR-IOV VF and the standard virtio driver within the VM, while the control plane is managed on the host by the vDPA application.

OVS Hardware Offloads Configuration

OVS-Kernel Hardware Offloads

OVS-Kernel is the default OVS flavor enabled on your NVIDIA device.



Switchdev Configuration

1. Unbind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
```

ⓘ **Note**

VMs with attached VFs must be powered off to be able to unbind the VFs.

2. Change the eSwitch mode from legacy to switchdev on the PF device:

```
# devlink dev eswitch set pci/0000:3b:00.0 mode switchdev
```

This also creates the VF representor netdevices in the host OS.

Note

Before changing the mode, make sure that all VFs are unbound.

Info

To return to SR-IOV legacy mode, run:

```
# devlink dev eswitch set pci/0000:3b:00.0 mode
legacy
```

This also removes the VF representor netdevices.

On OSes or kernels that do not support devlink, moving to switchdev mode can be done using sysfs:

```
# echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

3. At this stage, VF representors have been created. To map a representor to its VF, make sure to obtain the representor's `switchid` and `portname` by running:

```
# ip -d link show eth4
41: enp0s8f0_1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether ba:e6:21:37:bc:d4 brd ff:ff:ff:ff:ff:ff
promiscuity 0 addrgenmode eui64 numtxqueues 10 numrxqueues 10
gso_max_size 65536 gso_max_segs 65535 portname pf0vf1 switchid
f4ab580003a1420c
```

Where:

   - `switchid` – used to map representor to device, both device PFs have the same `switchid`
   - `portname` – used to map representor to PF and VF. Value returned is `pf<X>vf<Y>`, where `X` is the PF number and `Y` is the number of VF.

4. Bind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

Switchdev Performance Tuning

Switchdev tuning improves its performance.

Steering Mode

OVS-kernel supports two steering modes for rule insertion into hardware:

- SMFS (software-managed flow steering) – default mode; rules are inserted directly to the hardware by the software (driver). This mode is optimized for rule insertion.
- DMFS (device-managed flow steering) – rule insertion is done using firmware commands. This mode is optimized for throughput with a small amount of rules in the system.

The steering mode can be configured via sysfs or devlink API in kernels that support it:

- For sysfs:

```
echo <smfs|dmfs> > /sys/class/net/<pf-netdev>/compat/devlink/steering_mode
```

- For devlink:

```
devlink dev param set pci/0000:00:08.0 name flow_steering_mode value "<smfs|dmfs>" cmode runtime
```

Notes:

- The mode should be set prior to moving to switchdev, by echoing to the sysfs or invoking the devlink command.
- Only when moving to switchdev will the driver use the mode configured.
- Mode cannot be changed after moving to switchdev.
- The steering mode is applicable for switchdev mode only (i.e., it does not affect legacy SR-IOV or other configurations).

Troubleshooting SMFS

mlx5 debugfs supports presenting Software Steering resources. `dr_domain` including its tables, matchers and rules. The interface is read-only.

> **Note**
>
> New steering rules cannot be inserted/deleted w hile the dump is being created,

The steering information is dumped in the CSV form in the following format: `<object_type>,<object_ID>, <object_info>,...,<object_info>` .

This data can be read at the following path: `/sys/kernel/debug/mlx5/<BDF>/steering/fdb/<domain_handle>` .

Example:

```
# cat
/sys/kernel/debug/mlx5/0000:82:00.0/steering/fdb/dmn_000018644
3100,0x55caa4621c50,0xee802,4,65533
3101,0x55caa4621c50,0xe0100008
```

You can then use the steering dump parser to make the output more human-readable.

The parser can be found in this GitHub repository.

vPort Match Mode

OVS-kernel support two modes that define how the rules match on vport.

Mode	Description
Metadata	Rules match on metadata instead of vport number (default mode). This mode is needed to support SR-IOV live migration and dual-port RoCE.

Mode	Description
	Note Matching on Metadata can have a performance impact.
Legacy	Rules match on vport number. In this mode, performance can be higher in comparison to Metadata. It can be used only if SR-IOV live migration or dual port RoCE are enabled/used.

vPort match mode can be controlled via sysfs:

- Set legacy:

```
echo legacy > /sys/class/net/<PF
netdev>/compat/devlink/vport_match_mode
```

- Set metadata:

```
echo metadata > /sys/class/net/<PF
netdev>/compat/devlink/vport_match_mode
```

Note

This mode must be set prior to moving to switchdev.

Flow Table Large Group Number

Offloaded flows, including connection tracking (CT), are added to the virtual switch forwarding data base (FDB) flow tables. FDB tables have a set of flow groups, where each flow group saves the same traffic pattern flows. For example, for CT offloaded flow, TCP and UDP are different traffic patterns which end up in two different flow groups.

A flow group has a limited size to save flow entries. By default, the driver has 15 big FDB flow groups. Each of these big flow groups can save 4M/(15+1)=256k different 5-tuple flow entries at most. For scenarios with more than 15 traffic patterns, the driver provides a module parameter (`num_of_groups`) to allow customization and performance tuning.

The mode can be controlled via module param or devlink API for kernels that support it:

- Module param:

```
echo <num_of_groups> >
/sys/module/mlx5_core/parameters/num_of_groups
```

- Devlink:

```
devlink dev param set pci/0000:82:00.0 name fdb_large_groups
cmode driverinit value 20
```

Note

The change takes effect immediately if no flows are inside the FDB table (no traffic running and all offloaded flows are aged out). And it can be dynamically changed without reloading the driver. If there are still offloaded flows when changing this parameter, it takes effect after all flows have aged out.

Open vSwitch Configuration

OVS configuration is a simple OVS bridge configuration with switchdev.

1. Run the OVS service:

```
systemctl start openvswitch
```

2. Create an OVS bridge (named `ovs-sriov` here):

```
ovs-vsctl add-br ovs-sriov
```

3. Enable hardware offload (disabled by default):

```
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

4. Restart the OVS service:

```
systemctl restart openvswitch
```

   This step is required for hardware offload changes to take effect.

5. Add the PF and the VF representor netdevices as OVS ports:

```
ovs-vsctl add-port ovs-sriov enp4s0f0
ovs-vsctl add-port ovs-sriov enp4s0f0_0
ovs-vsctl add-port ovs-sriov enp4s0f0_1
```

   Make sure to bring up the PF and representor netdevices:

```
ip link set dev enp4s0f0 up
ip link set dev enp4s0f0_0 up
ip link set dev enp4s0f0_1 up
```

The PF represents the uplink (wire):

```
# ovs-dpctl show
system@ovs-system:
        lookups: hit:0 missed:192 lost:1
        flows: 2
        masks: hit:384 total:2 hit/pkt:2.00
        port 0: ovs-system (internal)
        port 1: ovs-sriov (internal)
        port 2: enp4s0f0
        port 3: enp4s0f0_0
        port 4: enp4s0f0_1
```

6. Run traffic from the VFs and observe the rules added to the OVS data-path:

```
# ovs-dpctl dump-flows

recirc_id(0),in_port(3),eth(src=e4:11:22:33:44:50,dst=e4:1d:2d:a5
eth_type(0x0800),ipv4(frag=no), packets:33, bytes:3234,
used:1.196s, actions:2

recirc_id(0),in_port(2),eth(src=e4:1d:2d:a5:f3:9d,dst=e4:11:22:3
eth_type(0x0800),ipv4(frag=no), packets:34, bytes:3332,
used:1.196s, actions:3
```

In this example, the ping is initiated from VF0 (OVS port 3) to the outer node (OVS port 2), where the VF MAC is `e4:11:22:33:44:50` and the outer node MAC is `e4:1d:2d:a5:f3:9d`. As previously shown, two OVS rules are added, one in each direction.

> **Note**
>
> Users can also verify offloaded packets by adding `type=offloaded` to the command. For example:
>
> ```
> ovs-appctl dpctl/dump-flows type=offloaded
> ```

OVS Performance Tuning

Flow Aging

The aging timeout of OVS is given in milliseconds and can be controlled by running:

```
ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

TC Policy

Specifies the policy used with hardware offloading:

- `none` – adds a TC rule to both the software and the hardware (default)
- `skip_sw` – adds a TC rule only to the hardware
- `skip_hw` – adds a TC rule only to the software

Example:

```
ovs-vsctl set Open_vSwitch . other_config:tc-policy=skip_sw
```

> **Note**
>
> TC policy should only be used for debugging purposes.

max-revalidator

Specifies the maximum time (in milliseconds) for the revalidator threads to wait for kernel statistics before executing flow revalidation.

```
ovs-vsctl set Open_vSwitch . other_config:max-revalidator=10000
```

n-handler-threads

Specifies the number of threads for software datapaths to use to handle new flows.

```
ovs-vsctl set Open_vSwitch . other_config:n-handler-threads=4
```

The default value is the number of online CPU cores minus the number of revalidators.

n-revalidator-threads

Specifies the number of threads for software datapaths to use to revalidate flows in the datapath.

```
ovs-vsctl set Open_vSwitch . other_config:n-revalidator-threads=4
```

vlan-limit

Limits the number of VLAN headers that can be matched to the specified number.

```
ovs-vsctl set Open_vSwitch . other_config:vlan-limit=2
```

Basic TC Rules Configuration

Offloading rules can also be added directly, and not only through OVS, using the `tc` utility.

To create an offloading rule using TC:

1. Create an ingress qdisc (queueing discipline) for each interface that you wish to add rules into:

   ```
   tc qdisc add dev enp4s0f0 ingress
   tc qdisc add dev enp4s0f0_0 ingress
   tc qdisc add dev enp4s0f0_1 ingress
   ```

2. Add TC rules using flower classifier in the following format:

   ```
   tc filter add dev NETDEVICE ingress protocol PROTOCOL prio
   PRIORITY [chain CHAIN] flower [MATCH_LIST] [action
   ACTION_SPEC]
   ```

> **Note**
>
> A list of supported matches (specifications) and actions can be found in section "Classification Fields (Matches)".

3. Dump the existing `tc` rules using flower classifier in the following format:

```
tc [-s] filter show dev NETDEVICE ingress
```

SR-IOV VF LAG

SR-IOV VF LAG allows the NIC's physical functions (PFs) to get the rules that the OVS tries to offload to the bond net-device, and to offload them to the hardware e-switch.

The supported bond modes are as follows:

- Active-backup
- XOR
- LACP

SR-IOV VF LAG enables complete offload of the LAG functionality to the hardware. The bonding creates a single bonded PF port. Packets from the up-link can arrive from any of the physical ports and are forwarded to the bond device.

When hardware offload is used, packets from both ports can be forwarded to any of the VFs. Traffic from the VF can be forwarded to both ports according to the bonding state. This means that when in active-backup mode, only one PF is up, and traffic from any VF goes through this PF. When in XOR or LACP mode, if both PFs are up, traffic from any VF is split between these two PFs.

SR-IOV VF LAG Configuration on ASAP2

To enable SR-IOV VF LAG, both physical functions of the NIC must first be configured to SR-IOV switchdev mode, and only afterwards bond the up-link representors.

The following example shows the creation of a bond interface over two PFs:

1. Load the bonding device and subordinate the up-link representor (currently PF) net-device devices:

```
modprobe bonding mode=802.3ad
Ifup bond0 (make sure ifcfg file is present with desired bond
configuration)
ip link set enp4s0f0 master bond0
ip link set enp4s0f1 master bond0
```

2. Add the VF representor net-devices as OVS ports. If tunneling is not used, add the bond device as well.

```
ovs-vsctl add-port ovs-sriov bond0
ovs-vsctl add-port ovs-sriov enp4s0f0_0
ovs-vsctl add-port ovs-sriov enp4s0f1_0
```

3. Bring up the PF and the representor netdevices:

```
ip link set dev bond0 up
ip link set dev enp4s0f0_0 up
ip link set dev enp4s0f1_0 up
```

Note

Once the SR-IOV VF LAG is configured, all VFs of the two PFs become part of the bond and behave as described above.

Using TC with VF LAG

Both rules can be added either with or without shared block:

- With shared block (supported from kernel 4.16 and RHEL/CentOS 7.7 and above):

```
tc qdisc add dev bond0 ingress_block 22 ingress
tc qdisc add dev ens4p0 ingress_block 22 ingress
tc qdisc add dev ens4p1 ingress_block 22 ingress
```

1. Add drop rule:

```
# tc filter add block 22 protocol arp parent ffff: prio 3
\
    flower \
        dst_mac e4:11:22:11:4a:51 \
        action drop
```

2. Add redirect rule from bond to representor:

```
# tc filter add block 22 protocol arp parent ffff: prio 3
\
        flower \
        dst_mac e4:11:22:11:4a:50 \
        action mirred egress redirect dev ens4f0_0
```

3. Add redirect rule from representor to bond:

```
# tc filter add dev ens4f0_0 protocol arp parent ffff:
prio 3 \
    flower \
        dst_mac ec:0d:9a:8a:28:42 \
        action mirred egress redirect dev bond0
```

- Without shared block (supported from kernel 4.15 and below):

  1. Add redirect rule from bond to representor:

```
# tc filter add dev bond0 protocol arp parent ffff: prio
1 \
        flower \
        dst_mac e4:11:22:11:4a:50 \
        action mirred egress redirect dev ens4f0_0
```

  2. Add redirect rule from representor to bond:

```
# tc filter add dev ens4f0_0 protocol arp parent ffff:
prio 3 \
        flower \
        dst_mac ec:0d:9a:8a:28:42 \
        action mirred egress redirect dev bond0
```

Classification Fields (Matches)

OVS-Kernel supports multiple classification fields which packets can fully or partially match.

Ethernet Layer 2

- Destination MAC
- Source MAC
- Ethertype

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0
packets:1981, bytes:206024, used:0.440s, dp:tc, actions:eth7
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
flower \
dst_mac e4:1d:2d:5d:25:35 \
src_mac e4:1d:2d:5d:25:34 \
action mirred egress redirect dev $NIC
```

IPv4/IPv6

- Source address
- Destination address
- Protocol
    - TCP/UDP/ICMP/ICMPv6
- TOS
- TTL (HLIMIT)

Supported on all kernels.

In OVS dump flows:

```
Ipv4:
ipv4(src=0.0.0.0/0.0.0.0,dst=0.0.0.0/0.0.0.0,proto=17,tos=0/0,ttl=0/0,frag=nc
Ipv6:
ipv6(src=::/::,dst=1:1:1::3:1040:1008,label=0/0,proto=58,tclass=0/0x3,h
```

Using TC rules:

```
IPv4:
tc filter add dev $rep parent ffff: protocol ip pref 1 \
flower \
dst_ip 1.1.1.1 \
src_ip 1.1.1.2 \
ip_proto TCP \
ip_tos 0x3 \
ip_ttl 63 \
action mirred egress redirect dev $NIC


IPv6:
tc filter add dev $rep parent ffff: protocol ipv6 pref 1 \
flower \
dst_ip 1:1:1::3:1040:1009 \
src_ip 1:1:1::3:1040:1008 \
ip_proto TCP \
ip_tos 0x3 \
ip_ttl 63\
action mirred egress redirect dev $NIC
```

TCP/UDP Source and Destination Ports and TCP Flags

- TCP/UDP source and destinations ports
- TCP flags

Supported on kernel >4.13 and RHEL >7.5.

In OVS dump flows:

```
TCP: tcp(src=0/0,dst=32768/0x8000),
UDP: udp(src=0/0,dst=32768/0x8000),
TCP flags: tcp_flags(0/0)
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol ip pref 1 \
flower \
ip_proto TCP \
dst_port 100 \
src_port 500 \
tcp_flags 0x4/0x7 \
action mirred egress redirect dev $NIC
```

VLAN

- ID
- Priority
- Inner vlan ID and Priority

Supported kernels: All (QinQ: kernel 4.19 and higher, and RHEL 7.7 and higher).

In OVS dump flows:

```
eth_type(0x8100),vlan(vid=2347,pcp=0),
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol 802.1Q pref 1 \
                    flower \
                    vlan_ethtype 0x800 \
                    vlan_id 100 \
                    vlan_prio 0 \
                    action mirred egress redirect dev $NIC
QinQ:
tc filter add dev $rep parent ffff: protocol 802.1Q pref 1 \
                    flower \
                    vlan_ethtype 0x8100    \
                    vlan_id 100 \
                    vlan_prio 0 \
                    cvlan_id 20 \
                    cvlan_prio 0 \
                    cvlan_ethtype 0x800 \
                    action mirred egress redirect dev $NIC
```

Tunnel

- ID (Key)
- Source IP address
- Destination IP address
- Destination port
- TOS (supported from kernel 4.19 and above & RHEL 7.7 and above)
- TTL (support from kernel 4.19 and above & RHEL 7.7 and above)

- Tunnel options (Geneve)

Supported kernels:

- VXLAN: All
- GRE: Kernel >5.0, RHEL 7.7 and above
- Geneve: Kernel >5.0, RHEL 7.7 and above

In OVS dump flows:

```
tunnel(tun_id=0x5,src=121.9.1.1,dst=131.10.1.1,ttl=0/0,tp_dst=4789,flags(+
```

Using TC rules:

```
# tc filter add dev $rep protocol 802.1Q parent ffff: pref 1
flower \
vlan_ethtype 0x800 \
vlan_id 100 \
vlan_prio 0 \
action mirred egress redirect dev $NIC
QinQ:
# tc filter add dev vxlan100 protocol ip parent ffff: \
                flower \
                         skip_sw \
                         dst_mac e4:11:22:11:4a:51 \
                         src_mac e4+:11:22:11:4a:50 \
                         enc_src_ip 20.1.11.1 \
                         enc_dst_ip 20.1.12.1 \
                         enc_key_id 100 \
                         enc_dst_port 4789 \
                         action tunnel_key unset \
```

```
                                    action mirred egress redirect dev
ens4f0_0
```

Supported Actions

Forward

Forward action allows for packet redirection:

- From VF to wire
- Wire to VF
- VF to VF

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0
packets:1981, bytes:206024, used:0.440s, dp:tc, actions:eth7
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                                                  flower \
                                                  dst_mac
e4:1d:2d:5d:25:35 \
                                                  src_mac
e4:1d:2d:5d:25:34 \
                                          action mirred egress
redirect dev $NIC
```

Drop

Drop action allows to drop incoming packets.

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0
packets:1981, bytes:206024, used:0.440s, dp:tc, actions:drop
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                            flower \
                            dst_mac e4:1d:2d:5d:25:35 \
                            src_mac e4:1d:2d:5d:25:34 \
                            action drop
```

Statistics

By default, each flow collects the following statistics:

- Packets – number of packets which hit the flow
- Bytes – total number of bytes which hit the flow
- Last used – the amount of time passed since last packet hit the flow

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0
```

```
packets:1981, bytes:206024, used:0.440s, dp:tc, actions:drop
```

Using TC rules:

```
#tc -s filter show dev $rep ingress

filter protocol ip pref 2 flower chain 0
filter protocol ip pref 2 flower chain 0 handle 0x2
eth_type ipv4
ip_proto tcp
src_ip 192.168.140.100
src_port 80
skip_sw
in_hw
    action order 1: mirred (Egress Redirect to device p0v11_r)
stolen
    index 34 ref 1 bind 1 installed 144 sec used 0 sec
    Action statistics:
    Sent 388344 bytes 2942 pkt (dropped 0, overlimits 0 requeues 0)
    backlog 0b 0p requeues 0
```

Tunnels: Encapsulation/Decapsulation

OVS-kernel supports offload of tunnels using encapsulation and decapsulation actions.

- Encapsulation – pushing of tunnel header is supported on Tx
- Decapsulation – popping of tunnel header is supported on Rx

Supported Tunnels:

- VXLAN (IPv4/IPv6) – supported on all Kernels
- GRE (IPv4/IPv6) – supported on kernel 5.0 and above & RHEL 7.6 and above

- Geneve (IPv4/IPv6) – supported on kernel 5.0 and above & RHEL 7.6 and above

OVS configuration:

In case of offloading tunnel, the PF/bond should not be added as a port in the OVS datapath. It should rather be assigned with the IP address to be used for encapsulation.

The following example shows two hosts (PFs) with IPs 1.1.1.177 and 1.1.1.75, where the PF device on both hosts is enp4s0f0, and the VXLAN tunnel is set with VNID 98:

- On the first host:

```
# ip addr add 1.1.1.177/24 dev enp4s0f1
# ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0
type=vxlan options:local_ip=1.1.1.177 options:remote_ip=1.1.1.75
options:key=98
```

- On the second host:

```
# ip addr add 1.1.1.75/24 dev enp4s0f1
# ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0
type=vxlan options:local_ip=1.1.1.75 options:remote_ip=1.1.1.177
options:key=98
```

Info

For a GRE IPv4 tunnel, use `type=gre`. For a GRE IPv6 tunnel, use `type=ip6gre`. For a Geneve tunnel, use `type=geneve`.

Note

When encapsulating guest traffic, the VF's device MTU must be reduced to allow the host/hardware to add the encap headers without fragmenting the resulted packet. As such, the VF's MTU must be lowered by 50 bytes from the uplink MTU for IPv4 and 70 bytes for IPv6.

Tunnel offload using TC rules:

```
Encapsulation:
# tc filter add dev ens4f0_0 protocol 0x806 parent ffff: \
                flower \
                        skip_sw \
                        dst_mac e4:11:22:11:4a:51 \
                        src_mac e4:11:22:11:4a:50 \
                action tunnel_key set \
                src_ip 20.1.12.1 \
                dst_ip 20.1.11.1 \
                id 100 \
                action mirred egress redirect dev vxlan100

Decapsulation:
# tc filter add dev vxlan100 protocol 0x806 parent ffff: \
                flower \
                         skip_sw \
                         dst_mac e4:11:22:11:4a:51 \
                         src_mac e4:11:22:11:4a:50 \
                         enc_src_ip 20.1.11.1 \
                         enc_dst_ip 20.1.12.1 \
                         enc_key_id 100 \
                         enc_dst_port 4789 \
                action tunnel_key unset \
                action mirred egress redirect dev ens4f0_0
```

VLAN Push/Pop

OVS-kernel supports offload of VLAN header push/pop actions:

- Push – pushing of VLAN header is supported on Tx
- Pop – popping of tunnel header is supported on Rx

OVS Configuration

Add a tag=$TAG section for the OVS command line that adds the representor ports. For example, VLAN ID 52 is being used here.

```
# ovs-vsctl add-port ovs-sriov enp4s0f0
# ovs-vsctl add-port ovs-sriov enp4s0f0_0 tag=52
# ovs-vsctl add-port ovs-sriov enp4s0f0_1 tag=52
```

The PF port should not have a VLAN attached. This will cause OVS to add VLAN push/pop actions when managing traffic for these VFs.

Dump Flow Example

```
recirc_id(0),in_port(3),eth(src=e4:11:22:33:44:50,dst=00:02:c9:e9:bb:b
\
packets:0, bytes:0, used:never, actions:push_vlan(vid=52,pcp=0),2

recirc_id(0),in_port(2),eth(src=00:02:c9:e9:bb:b2,dst=e4:11:22:33:44:5
\
vlan(vid=52,pcp=0),encap(eth_type(0x0800),ipv4(frag=no)),
packets:0, bytes:0, used:never, actions:pop_vlan,3
```

VLAN Offload Using TC Rules Example

```
# tc filter add dev ens4f0_0 protocol ip parent ffff: \
                flower \
                        skip_sw \
                        dst_mac e4:11:22:11:4a:51 \
                        src_mac e4:11:22:11:4a:50 \
                action vlan push id 100 \
                action mirred egress redirect dev ens4f0

# tc filter add dev ens4f0 protocol 802.1Q parent ffff: \
                flower \
                        skip_sw \
                        dst_mac e4:11:22:11:4a:51 \
                        src_mac e4:11:22:11:4a:50 \
                        vlan_ethtype 0x800 \
                        vlan_id 100 \
                        vlan_prio 0 \
                action vlan pop \
                action mirred egress redirect dev ens4f0_0
```

TC Configuration

Example of VLAN Offloading with popping header on Tx and pushing on Rx using TC rules:

```
# tc filter add dev ens4f0_0 ingress protocol 802.1Q parent ffff:
\
        flower \
                vlan_id 100 \
        action vlan pop \
        action tunnel_key set \
                src_ip 4.4.4.1 \
                dst_ip 4.4.4.2 \
                dst_port 4789 \
```

```
                id 42 \
        action mirred egress redirect dev vxlan0

# tc filter add dev vxlan0 ingress protocol all parent ffff: \
        flower \
                enc_dst_ip 4.4.4.1 \
                enc_src_ip 4.4.4.2 \
                enc_dst_port 4789 \
                enc_key_id 42 \
        action tunnel_key unset \
        action vlan push id 100 \
        action mirred egress redirect dev ens4f0_0
```

Header Rewrite

This action allows for modifying packet fields.

Ethernet Layer 2

- Destination MAC
- Source MAC

Supported kernels:

- Kernel 4.14 and above
- RHEL 7.5 and above

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0
packets:1981, bytes:206024, used:0.440s, dp:tc, actions:
```

```
set(eth(src=68:54:ed:00:f4:ab,dst=fa:16:3e:dd:69:c4)),eth7
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                                                        flower \
                                                        dst_mac
e4:1d:2d:5d:25:35 \
                                                        src_mac
e4:1d:2d:5d:25:34 \
                                        action pedit ex \
                                        munge eth dst set
20:22:33:44:55:66 \
                                        munge eth src set
aa:ba:cc:dd:ee:fe \
                                        action mirred egress
redirect dev $NIC
```

IPv4/IPv6

- Source address
- Destination address
- Protocol
- TOS
- TTL (HLIMIT)

Supported kernels:

- Kernel 4.14 and above
- RHEL 7.5 and above

In OVS dump flows:

```
Ipv4:
    set(eth(src=de:e8:ef:27:5e:45,dst=00:00:01:01:01:01)),
    set(ipv4(src=10.10.0.111,dst=10.20.0.122,ttl=63))
Ipv6:
    set(ipv6(dst=2001:1:6::92eb:fcbe:f1c8,hlimit=63)),
```

Using TC rules:

```
IPv4:
tc filter add dev $rep parent ffff: protocol ip pref 1 \
                                                        flower \
                                                        dst_ip
1.1.1.1 \
                                                        src_ip
1.1.1.2 \
                                                        ip_proto
TCP \
                                                        ip_tos
0x3 \
                                                        ip_ttl 63
\
                                        pedit ex \
                                        munge ip src set 2.2.2.1 \
                                        munge ip dst set 2.2.2.2 \
                                        munge ip tos set 0 \
                                        munge ip ttl dec \
                                        action mirred egress
redirect dev $NIC


IPv6:
```

```
tc filter add dev $rep parent ffff: protocol ipv6 pref 1 \
                                                        flower \
                                                        dst_ip
1:1:1::3:1040:1009 \
                                                        src_ip
1:1:1::3:1040:1008 \
                                                        ip_proto
tcp \
                                                        ip_tos
0x3 \
                                                        ip_ttl
63\
                                        pedit ex \
                                        munge ipv6 src set
2:2:2::3:1040:1009 \
                                        munge ipv6 dst set
2:2:2::3:1040:1008 \
                                        munge ipv6 hlimit dec \
                                        action mirred egress
redirect dev $NIC
```

Note

IPv4 and IPv6 header rewrite is only supported with match on UDP/TCP/ICMP protocols.

TCP/UDP Source and Destination Ports

- TCP/UDP source and destinations ports

Supported kernels:

- Kernel 4.16 and above

- RHEL 7.6 and above

In OVS dump flows:

```
TCP:

        set(tcp(src= 32768/0xffff,dst=32768/0xffff)),
UDP:

        set(udp(src= 32768/0xffff,dst=32768/0xffff)),
```

Using TC rules:

```
TCP:

        tc filter add dev $rep parent ffff: protocol ip pref 1 \
                          flower \
                          dst_ip 1.1.1.1 \
                          src_ip 1.1.1.2 \
                          ip_proto tcp \
                          ip_tos 0x3 \
                          ip_ttl 63 \
                    pedit ex \
                    pedit ex munge ip tcp sport set 200
                    pedit ex munge ip tcp dport set 200
                    action mirred egress redirect dev $NIC

UDP:
        tc filter add dev $rep parent ffff: protocol ip pref 1 \
                          flower \
                          dst_ip 1.1.1.1 \
                          src_ip 1.1.1.2 \
                          ip_proto udp \
                          ip_tos 0x3 \
```

```
                ip_ttl 63 \
            pedit ex \
            pedit ex munge ip udp sport set 200
            pedit ex munge ip udp dport set 200
            action mirred egress redirect dev $NIC
```

VLAN

- ID

Supported on all kernels.

In OVS dump flows:

```
Set(vlan(vid=2347,pcp=0/0)),
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol 802.1Q pref 1 \
                    flower \
                    vlan_ethtype 0x800 \
                    vlan_id 100 \
                    vlan_prio 0 \
            action vlan modify id 11 pipe
            action mirred egress redirect dev $NIC
```

Connection Tracking

The TC connection tracking (CT) action performs CT lookup by sending the packet to netfilter conntrack module. Newly added connections may be associated, via the `ct commit` action, with a 32 bit mark, 128 bit label, and source/destination NAT values.

The following example allows ingress TCP traffic from the uplink representor to `vf1_rep`, while assuring that egress traffic from `vf1_rep` is only allowed on established connections. In addition, mark and source IP NAT is applied.

In OVS dump flows:

```
ct(zone=2,nat)
ct_state(+est+trk)
actions:ct(commit,zone=2,mark=0x4/0xffffffff,nat(src=5.5.5.5))
```

Using TC rules:

```
# tc filter add dev $uplink_rep ingress chain 0 prio 1 proto ip \
                    flower \
                    ip_proto tcp    \
                    ct_state -trk \
               action ct zone 2 nat pipe
               action goto chain 2
# tc filter add dev $uplink_rep ingress chain 2 prio 1 proto ip \
                     flower \
                     ct_state +trk+new  \
               action ct zone 2 commit mark 0xbb nat src addr
5.5.5.7 pipe \
               action mirred egress redirect dev $vf1_rep
# tc filter add dev $uplink_rep ingress chain 2 prio 1 proto ip \
                    flower \
                    ct_zone 2 \
                    ct_mark 0xbb \
                    ct_state +trk+est \
                action mirred egress redirect dev $vf1_rep

// Setup filters on $vf1_rep, allowing only established connections of zone 2 through, and reverse nat (dst nat in this case)
```

```
# tc filter add dev $vf1_rep ingress chain 0 prio 1 proto ip \
                     flower \
                     ip_proto tcp \
                     ct_state -trk \
                action ct zone 2 nat pipe \
                action goto chain 1
# tc filter add dev $vf1_rep ingress chain 1 prio 1 proto ip \
                     flower \
                     ct_zone 2 \
                     ct_mark 0xbb \
                     ct_state +trk+est \
                action mirred egress redirect dev eth0
```

CT Performance Tuning

- Max offloaded connections – specifies the limit on the number of offloaded connections. Example:

```
devlink dev param set pci/${pci_dev} name
ct_max_offloaded_conns value $max cmode runtime
```

- Allow mixed NAT/non-NAT CT – allows offloading of the following scenario:

```
•       cookie=0x0, duration=21.843s, table=0,
n_packets=4838718, n_bytes=241958846, ct_state=-
trk,ip,in_port=enp8s0f0 actions=ct(table=1,zone=2)
•       cookie=0x0, duration=21.823s, table=1, n_packets=15363,
n_bytes=773526, ct_state=+new+trk,ip,in_port=enp8s0f0
actions=ct(commit,zone=2,nat(dst=11.11.11.11)),output:"enp8s0f0_1" •
cookie=0x0, duration=21.806s, table=1, n_packets=4767594,
n_bytes=238401190, ct_state=+est+trk,ip,in_port=enp8s0f0
actions=ct(zone=2,nat),output:"enp8s0f0_1"
```

Example:

```
echo enable >
/sys/class/net/<device>/compat/devlink/ct_action_on_nat_conns
```

Forward to Chain (TC Only)

TC interface supports adding flows on different chains. Only chain 0 is accessed by default. Access to the other chains requires using the `goto` action.

In this example, a flow is created on chain 1 without any match and redirect to wire.

The second flow is created on chain 0 and match on source MAC and action `goto` chain 1.

This example simulates simple MAC spoofing:

```
#tc filter add dev $rep parent ffff: protocol all chain 1 pref 1
\
                    flower \
                action mirred egress redirect dev $NIC

#tc filter add dev $rep parent ffff: protocol all chain 1 pref 1
\
                    flower \
                    src_mac aa:bb:cc:aa:bb:cc \
                action goto chain 1
```

Port Mirroring: Flow-based VF Traffic Mirroring for ASAP²

Unlike para-virtual configurations, when the VM traffic is offloaded to hardware via SR-IOV VF, the host-side admin cannot snoop the traffic (e.g., for monitoring).

ASAP² uses the existing mirroring support in OVS and TC along with the enhancement to the offloading logic in the driver to allow mirroring the VF traffic to another VF.

The mirrored VF can be used to run traffic analyzer (e.g., tcpdump, wireshark, etc.) and observe the traffic of the VF being mirrored.

The following example shows the creation of port mirror on the following configuration:

```
# ovs-vsctl show
  09d8a574-9c39-465c-9f16-47d81c12f88a
      Bridge br-vxlan
                  Port "enp4s0f0_1"
                     Interface "enp4s0f0_1"
                  Port "vxlan0"
                      Interface "vxlan0"
                                  type: vxlan
                                  options: {key="100",
remote_ip="192.168.1.14"}
                  Port "enp4s0f0_0"
                      Interface "enp4s0f0_0"
                  Port "enp4s0f0_2"
                      Interface "enp4s0f0_2"
                  Port br-vxlan
                      Interface br-vxlan
                                  type: internal
      ovs_version: "2.14.1"
```

- To set `enp4s0f0_0` as the mirror port and mirror all the traffic:

```
# ovs-vsctl -- --id=@p get port enp4s0f0_0 \
             -- --id=@m create mirror name=m0 select-all=true
output-port=@p \
```

```
            -- set bridge br-vxlan mirrors=@m
```

- To set `enp4s0f0_0` as the mirror port, only mirror the traffic, and set `enp4s0f0_1` as the destination port:

```
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
            -- --id=@p2 get port enp4s0f0_1 \
            -- --id=@m create mirror name=m0 select-dst-
port=@p2 output-port=@p1 \
            -- set bridge br-vxlan mirrors=@m
```

- To set `enp4s0f0_0` as the mirror port, only mirror the traffic, and set `enp4s0f0_1` as the source port:

```
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
            -- --id=@p2 get port enp4s0f0_1 \
            -- --id=@m create mirror name=m0 select-src-
port=@p2 output-port=@p1 \
            -- set bridge br-vxlan mirrors=@m
```

- To set `enp4s0f0_0` as the mirror port and mirror all the traffic on `enp4s0f0_1`:

```
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
            -- --id=@p2 get port enp4s0f0_1 \
            -- --id=@m create mirror name=m0 select-dst-
port=@p2 select-src-port=@p2 output-port=@p1 \
            -- set bridge br-vxlan mirrors=@m
```

To clear the mirror port:

```
ovs-vsctl clear bridge br-vxlan mirrors
```

Mirroring using TC:

- Mirror to VF:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                                                flower \
                                                dst_mac
e4:1d:2d:5d:25:35 \
                                                src_mac
e4:1d:2d:5d:25:34 \
                                                action mirred
egress mirror dev $mirror_rep pipe \
                                                action mirred
egress redirect dev $NIC
```

- Mirror to tunnel:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                                                flower \
                                                dst_mac
e4:1d:2d:5d:25:35 \
                                                src_mac
e4:1d:2d:5d:25:34 \
                                action tunnel_key set \
                                src_ip 1.1.1.1 \
                                dst_ip 1.1.1.2 \
                                dst_port 4789 \
                                id 768 \
                                pipe \
```

```
                                action mirred egress mirror
dev vxlan100 pipe \
                                action mirred egress redirect
dev $NIC
```

Forward to Multiple Destinations

Forwarding to up 32 destinations (representors and tunnels) is supported using TC:

- Example 1 – forwarding to 32 VFs:

```
tc filter add dev $NIC parent ffff: protocol arp pref 1 \
                        flower \
                        dst_mac e4:1d:2d:5d:25:35 \
                        src_mac e4:1d:2d:5d:25:34 \
                        action mirred egress mirror dev $rep0
pipe \
                        action mirred egress mirror dev $rep1
pipe \
...
                        action mirred egress mirror dev
$rep30 pipe \
                        action mirred egress redirect dev
$rep31
```

- Example 2 – forwarding to 16 tunnels:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                        flower \
                        dst_mac e4:1d:2d:5d:25:35 \
                        src_mac e4:1d:2d:5d:25:34 \
```

```
                        action tunnel_key set src_ip $ip_src
dst_ip $ip_dst \
                        dst_port 4789 id 0 nocsum \
                        pipe action mirred egress mirror dev
vxlan0 pipe \
                        action tunnel_key set src_ip $ip_src
dst_ip $ip_dst \
                        dst_port 4789 id 1 nocsum \
                        pipe action mirred egress mirror dev
vxlan0 pipe \
                        ...
                        action tunnel_key set src_ip $ip_src
dst_ip $ip_dst \
                        dst_port 4789 id 15 nocsum \
                        pipe action mirred egress redirect
dev vxlan0
```

Note

TC supports up to 32 actions.

Note

If header rewrite is used, then all destinations should have the same header rewrite.

Note

If VLAN push/pop is used, then all destinations should have the same VLAN ID and actions.

sFlow

sFlow allows for monitoring traffic sent between two VMs on the same host using an sFlow collector.

The following example assumes the environment is configured as described later.

```
# ovs-vsctl show
  09d8a574-9c39-465c-9f16-47d81c12f88a
      Bridge br-vxlan
                  Port "enp4s0f0_1"
                    Interface "enp4s0f0_1"
                  Port "vxlan0"
                    Interface "vxlan0"
                                type: vxlan
                                options: {key="100",
remote_ip="192.168.1.14"}
                  Port "enp4s0f0_0"
                    Interface "enp4s0f0_0"
                  Port "enp4s0f0_2"
                    Interface "enp4s0f0_2"
                  Port br-vxlan
                    Interface br-vxlan
                                type: internal
      ovs_version: "2.14.1"
```

To sample all traffic over the OVS bridge:

```
# ovs-vsctl -- --id=@sflow create sflow agent=\"$SFLOW_AGENT\" \

target=\"$SFLOW_TARGET:$SFLOW_PORT\" \
                                        header=$SFLOW_HEADER \
                                        sampling=$SFLOW_SAMPLING
polling=10 \
               -- set bridge br-vxlan sflow=@sflow
```

Parameter	Description
`SFLOW_AGENT`	Indicates that the sFlow agent should send traffic from `SFLOW_AGENT`'s IP address
`SFLOW_TARGET`	Remote IP address of the sFlow collector
`SFLOW_HEADER`	Size of packet header to sample (in bytes)
`SFLOW_SAMPLING`	Sample rate

To clear the sFlow configuration:

```
# ovs-vsctl clear bridge br-vxlan sflow
```

To list the sFlow configuration:

```
# ovs-vsctl list sflow
```

sFlow using TC:

```
Sample to VF
```

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
                                                flower \
                                                dst_mac
e4:1d:2d:5d:25:35 \
                                                src_mac
e4:1d:2d:5d:25:34 \
                                                action sample
rate 10 group 5 trunc 96 \
                                                action mirred
egress redirect dev $NIC
```

Note

A userspace application is needed to process the sampled packet from the kernel. An example is available on Github.

Rate Limit

OVS-kernel supports offload of VF rate limit using OVS configuration and TC.

The following example sets the rate limit to the VF related to representor `eth0` to 10Mb/s:

- OVS:

  ```
  ovs-vsctl set interface eth0 ingress_policing_rate=10000
  ```

- TC:

```
tc_filter add dev eth0 root prio 1 protocol ip matchall
skip_sw action police rate 10mbit burst 20k
```

Kernel Requirements

This kernel config should be enabled to support switchdev offload.

- `CONFIG_NET_ACT_CSUM` – needed for action csum
- `CONFIG_NET_ACT_PEDIT` – needed for header rewrite
- `CONFIG_NET_ACT_MIRRED` – needed for basic forward
- `CONFIG_NET_ACT_CT` – needed for CT (supported from kernel 5.6)
- `CONFIG_NET_ACT_VLAN` – needed for action vlan push/pop
- `CONFIG_NET_ACT_GACT`
- `CONFIG_NET_CLS_FLOWER`
- `CONFIG_NET_CLS_ACT`
- `CONFIG_NET_SWITCHDEV`
- `CONFIG_NET_TC_SKB_EXT` – needed for CT (supported from kernel 5.6)
- `CONFIG_NET_ACT_CT` – needed for CT (supported from kernel 5.6)
- `CONFIG_NFT_FLOW_OFFLOAD`
- `CONFIG_NET_ACT_TUNNEL_KEY`
- `CONFIG_NF_FLOW_TABLE` – needed for CT (supported from kernel 5.6)
- `CONFIG_SKB_EXTENSIONS` – needed for CT (supported from kernel 5.6)

- `CONFIG_NET_CLS_MATCHALL`
- `CONFIG_NET_ACT_POLICE`
- `CONFIG_MLX5_ESWITCH`

VF Metering

OVS-kernel supports offloading of VF metering (TX and RX) using sysfs. Metering of number of packets per second (PPS) and bytes per second (BPS) is supported.

The following example sets Rx meter on VF 0 with value 10Mb/s BPS:

```
echo 10000000 >
/sys/class/net/enp4s0f0/device/sriov/0/meters/rx/bps/rate
echo 65536 >
/sys/class/net/enp4s0f0/device/sriov/0/meters/rx/bps/burst
```

The following example sets Tx meter on VF 0 with value 1000 PPS:

```
echo 1000 >
/sys/class/net/enp4s0f0/device/sriov/0/meters/tx/pps/rate
echo 100 >
/sys/class/net/enp4s0f0/device/sriov/0/meters/tx/pps/burst
```

Note

Both `rate` and `burst` must not be zero and `burst` may need to be adjusted according to the requirements.

The following counters can be used to query the number dropped packet/bytes:

```
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/rx/pps/packets_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/rx/pps/bytes_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/rx/bps/packets_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/rx/bps/bytes_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/tx/pps/packets_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/tx/pps/bytes_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/tx/bps/packets_dropped
cat
/sys/class/net/enp8s0f0/device/sriov/0/meters/tx/bps/bytes_dropped
```

Representor Metering

Info

Metering for uplink and VF representors traffic is supported.

Traffic going to a representor device can be a result of a miss in the embedded switch (eSwitch) FDB tables. This means that a packet which arrives from that representor into the eSwitch has not matched against the existing rules in the hardware FDB tables and must be forwarded to software to be handled there and is, therefore, forwarded to the originating representor device driver.

The meter allows to configure the max rate [packets per second] and max burst [packets] for traffic going to the representor driver. Any traffic exceeding values provided by the user are dropped in hardware. There are statistics that show the number of dropped packets.

The configuration of representor metering is done via `miss_rl_cfg`.

- Full path of the `miss_rl_cfg` parameter: `/sys/class/net//rep_config/miss_rl_cfg`
- Usage: `echo "<rate> <burst>" > /sys/class/net//rep_config/miss_rl_cfg`.
    - `rate` is the max rate of packets allowed for this representor (in packets/sec units)
    - `burst` is the max burst size allowed for this representor (in packets units)
    - Both values must be specified. Both of their default values is 0, signifying unlimited rate and burst.

To view the amount of packets and bytes dropped due to traffic exceeding the user-provided rate and burst, two read-only sysfs for statistics are available:

- `/sys/class/net//rep_config/miss_rl_dropped_bytes` – counts how many FDB-miss bytes are dropped due to reaching the miss limits
- `/sys/class/net//rep_config/miss_rl_dropped_packets` – counts how many FDB-miss packets are dropped due to reaching the miss limits

OVS Metering

There are two types of meters, kpps (kilobits per second) and pktps (packets per second). OVS-Kernel supports offloading both of them.

The following example is to offload a kpps meter.

1. Create OVS meter with a target rate:

```
ovs-ofctl -O OpenFlow13 add-meter ovs-sriov
meter=1,kbps,band=type=drop,rate=204800
```

2. Delete the default rule:

```
ovs-ofctl del-flows ovs-sriov
```

3. Configure OpenFlow rules:

```
ovs-ofctl -O OpenFlow13 add-flow ovs-sriov
'ip,dl_dst=e4:11:22:33:44:50,actions= meter:1,output:enp4s0f0_0'
ovs-ofctl -O OpenFlow13 add-flow ovs-sriov
'ip,dl_src=e4:11:22:33:44:50,actions= output:enp4s0f0'
ovs-ofctl -O OpenFlow13 add-flow ovs-sriov 'arp,actions=normal'
```

Here, the VF bandwidth on the receiving side is limited by the rate configured in step 1.

4. Run iperf server and be ready to receive UDP traffic. On the outer node, run iperf client to send UDP traffic to this VF. After traffic starts, check the offloaded meter rule:

```
ovs-appctl dpctl/dump-flows --names type=offloaded

recirc_id(0),in_port(enp4s0f0),eth(dst=e4:11:22:33:44:50),eth_typ
packets:11626587, bytes:17625889188, used:0.470s,
actions:meter(0),enp4s0f0_0
```

To verify metering, iperf client should set the target bandwidth with a number which is larger than the meter rate configured. Then it should apparent that packets are received with the limited rate on the server side and the extra packets are dropped by hardware.

Multiport eSwitch Mode

The multiport eswitch mode allows adding rules on a VF representor with an action forwarding the packet to the physical port of the physical function. This can be used to implement failover or forward packets based on external information such as the cost of the route.

1. To configure multiport eswitch mode , the nvconig parameter `LAG_RESOURCE_ALLOCATION` must be set.

2. After the driver loads, configure multiport eSwitch for each PF where `enp8s0f0` and `enp8s0f1` represent the netdevices for the PFs:

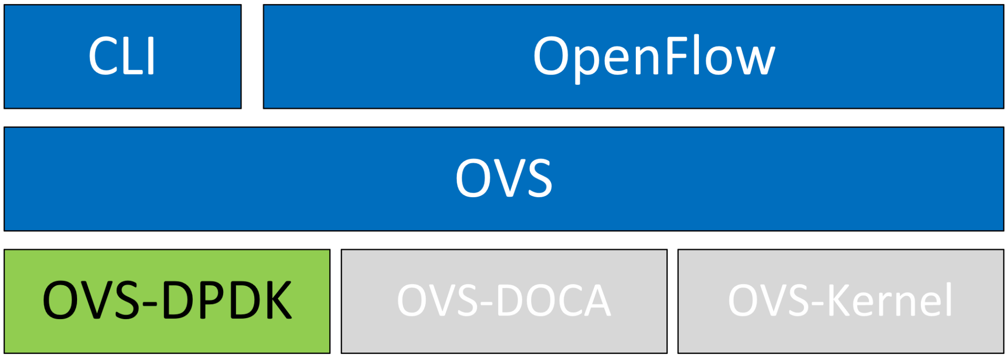
   ```
   echo multiport_esw >
   /sys/class/net/enp8s0f0/compat/devlink/lag_port_select_mode
   echo multiport_esw >
   /sys/class/net/enp8s0f1/compat/devlink/lag_port_select_mode
   ```

   The mode becomes operational after entering switchdev mode on both PFs.

Rule example:

```
tc filter add dev enp8s0f0_0 prot ip root flower dst_ip 7.7.7.7
action mirred egress redirect dev enp8s0f1
```

OVS-DPDK Hardware Offloads



OVS-DPDK Hardware Offloads Configuration

To configure OVS-DPDK HW offloads:

1. Unbind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
```

> **Note**
>
> VMs with attached VFs must be powered off to be able to unbind the VFs.

2. Change the e-switch mode from legacy to switchdev on the PF device (make sure all VFs are unbound). This also creates the VF representor netdevices in the host OS.

```
echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

To revert to SR-IOV legacy mode:

```
echo legacy > /sys/class/net/enp4s0f0/compat/devlink/mode
```

> **Note**
>
> This command removes the VF representor netdevices.

3. Bind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

4. Run the OVS service:

```
systemctl start openvswitch
```

5. Enable hardware offload (disabled by default):

```
ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-
init=true
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

6. Configure the DPDK whitelist:

```
ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-
extra="-a 0000:01:00.0,representor=[0],dv_flow_en=1,dv_esw_en=1,dv_xmeta_en=1"
```

Where `representor=[0-N]`.

7. Restart the OVS service:

```
systemctl restart openvswitch
```

Info

This step is required for the hardware offload changes to take effect.

8. Create OVS-DPDK bridge:

```
ovs-vsctl --no-wait add-br br0-ovs -- set bridge br0-ovs
datapath_type=netdev
```

9. Add PF to OVS:

```
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk
options:dpdk-devargs=0000:88:00.0
```

10. Add representor to OVS:

```
ovs-vsctl add-port br0-ovs representor -- set Interface
representor type=dpdk options:dpdk-
devargs=0000:88:00.0,representor=[0]
```

Where `representor=[0-N]`.

Offloading VXLAN Encapsulation/Decapsulation Actions

vSwitch in userspace requires an additional bridge. The purpose of this bridge is to allow use of the kernel network stack for routing and ARP resolution.

The datapath must look up the routing table and ARP table to prepare the tunnel header and transmit data to the output port.

Configuring VXLAN Encap/Decap Offloads

> **Note**
>
> The configuration is done with:
>
> - PF on 0000:03:00.0 PCIe and MAC 98:03:9b:cc:21:e8
> - Local IP 56.56.67.1 – br-phy interface is configured to this IP
> - Remote IP 56.56.68.1

To configure OVS-DPDK VXLAN:

1. Create a br-phy bridge:

```
ovs-vsctl add-br br-phy -- set Bridge br-phy
datapath_type=netdev -- br-set-external-id br-phy bridge-id
br-phy -- set bridge br-phy fail-mode=standalone
other_config:hwaddr=98:03:9b:cc:21:e8
```

2. Attach PF interface to br-phy bridge:

```
ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk
options:dpdk-devargs=0000:03:00.0
```

3. Configure IP to the bridge:

```
ip addr add 56.56.67.1/24 dev br-phy
```

4. Create a br-ovs bridge:

```
ovs-vsctl add-br br-ovs -- set Bridge br-ovs
datapath_type=netdev -- br-set-external-id br-ovs bridge-id
br-ovs -- set bridge br-ovs fail-mode=standalone
```

5. Attach representor to br-ovs:

```
ovs-vsctl add-port br-ovs pf0vf0 -- set Interface pf0vf0
type=dpdk options:dpdk-devargs=0000:03:00.0,representor=[0]
```

6. Add a port for the VXLAN tunnel:

```
ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0
type=vxlan options:local_ip=56.56.67.1
options:remote_ip=56.56.68.1 options:key=45 options:dst_port=4789
```

CT Offload

CT enables stateful packet processing by keeping a record of currently open connections. OVS flows using CT can be accelerated using advanced NICs by offloading established connections.

To view offloaded connections, run:

```
ovs-appctl dpctl/offload-stats-show
```

SR-IOV VF LAG

To configure OVS-DPDK SR-IOV VF LAG:

1. Enable SR-IOV on the NICs:

```
mlxconfig -d <PCI> set SRIOV_EN=1
```

2. Allocate the desired number of VFs per port:

```
echo $n > /sys/class/net/<net name>/device/sriov_numvfs
```

3. Unbind all VFs:

```
echo <VF PCI> >/sys/bus/pci/drivers/mlx5_core/unbind
```

4. Change both devices' mode to switchdev:

```
devlink dev eswitch set pci/<PCI> mode switchdev
```

5. Create Linux bonding using kernel modules:

```
modprobe bonding mode=<desired mode>
```

Info

Other bonding parameters can be added here. The supported bond modes are: Active-backup, XOR and LACP.

6. Bring all PFs and VFs down:

```
ip link set <PF/VF> down
```

7. Attach both PFs to the bond:

```
ip link set <PF> master bond0
```

8. To use VF-LAG with OVS-DPDK, add the bond master (PF) to the bridge:

```
ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk
options:dpdk-devargs=0000:03:00.0 options:dpdk-lsc-
interrupt=true
```

9. Add representor `$N` of PF0 or PF1 to a bridge:

```
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N
type=dpdk options:dpdk-devargs=<PF0 PCI>,representor=pf0vf$N
```

Or:

```
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N
type=dpdk options:dpdk-devargs=<PF0 PCI>,representor=pf1vf$N
```

VirtIO Acceleration Through VF Relay: Software and Hardware vDPA

> **Note**
>
> Hardware vDPA is enabled by default. In case your hardware does not support vDPA, the driver will fall back to Software vDPA.
>
> To check which vDPA mode is activated on your driver, run: `ovs-ofctl -O OpenFlow14 dump-ports br0-ovs` and look for `hw-mode` flag.

> **Note**
>
> This feature has not been accepted to the OVS-DPDK upstream yet, making its API subject to change.

In user space, there are two main approaches for communicating with a guest (VM), either through SR-IOV or virtio.

PHY ports (SR-IOV) allow working with port representor, which is attached to the OVS and a matching VF is given with pass-through to the guest. HW rules can process packets from up-link and direct them to the VF without going through SW (OVS). Therefore, using SR-IOV achieves the best performance.

However, SR-IOV architecture requires the guest to use a driver specific to the underlying HW. Specific HW driver has two main drawbacks:

- Breaks virtualization in some sense (guest is aware of the HW). It can also limit the type of images supported.
- Gives less natural support for live migration.

Using a virtio port solves both problems, however, it reduces performance and causes loss of some functionalities, such as, for some HW offloads, working directly with virtio. The netdev type dpdkvdpa solves this conflict as it is similar to the regular DPDK netdev yet introduces several additional functionalities.

dpdkvdpa translates between the PHY port to the virtio port. It takes packets from the Rx queue and sends them to the suitable Tx queue, and allows transfer of packets from the virtio guest (VM) to a VF and vice-versa, benefitting from both SR-IOV and virtio.

To add a vDPA port:

```
ovs-vsctl add-port br0 vdpa0 -- set Interface vdpa0 type=dpdkvdpa
\
options:vdpa-socket-path=<sock path> \
options:vdpa-accelerator-devargs=<vf pci id> \
options:dpdk-devargs=<pf pci id>,representor=[id] \
options: vdpa-max-queues =<num queues> \
options: vdpa-sw=<true/false>
```

> **Note**
>
> `vdpa-max-queues` is an optional field. When the user wants to configure 32 vDPA ports, the maximum queues number is limited to 8.

vDPA Configuration in OVS-DPDK Mode

Prior to configuring vDPA in OVS-DPDK mode, perform the following:

1. Generate the VF:

```
echo 0 > /sys/class/net/enp175s0f0/device/sriov_numvfs
echo 4 > /sys/class/net/enp175s0f0/device/sriov_numvfs
```

2. Unbind each VF:

```
echo <pci> > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Switch to switchdev mode:

```
echo switchdev >> /sys/class/net/enp175s0f0/compat/devlink/mode
```

4. Bind each VF:

```
echo <pci> > /sys/bus/pci/drivers/mlx5_core/bind
```

5. Initialize OVS:

```
ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
ovs-vsctl --no-wait set Open_vSwitch . other_config:hw-offload=true
```

To configure vDPA in OVS-DPDK mode:

1. OVS configuration:

```
ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-
extra="-a 0000:01:00.0,representor=[0],dv_flow_en=1,dv_esw_en=1,dv_xmeta_en=1"
/usr/share/openvswitch/scripts/ovs-ctl restart
```

2. Create OVS-DPDK bridge:

```
ovs-vsctl add-br br0-ovs -- set bridge br0-ovs
datapath_type=netdev
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk
options:dpdk-devargs=0000:01:00.0
```

3. Create vDPA port as part of the OVS-DPDK bridge:

```
ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0
type=dpdkvdpa options:vdpa-socket-path=/var/run/virtio-
forwarder/sock0 options:vdpa-accelerator-devargs=0000:01:00.2
options:dpdk-devargs=0000:01:00.0,representor=[0] options:
vdpa-max-queues=8
```

To configure vDPA in OVS-DPDK mode on BlueField DPUs, set the bridge with the software or hardware vDPA port:

- To create the OVS-DPDK bridge on the Arm side:

```
ovs-vsctl add-br br0-ovs -- set bridge br0-ovs
datapath_type=netdev
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk
options:dpdk-devargs=0000:af:00.0
```

```
ovs-vsctl add-port br0-ovs rep-- set Interface rep type=dpdk
options:dpdk-devargs=0000:af:00.0,representor=[0]
```

- To create the OVS-DPDK bridge on the host side:

```
ovs-vsctl add-br br1-ovs -- set bridge br1-ovs
datapath_type=netdev protocols=OpenFlow14
ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0
type=dpdkvdpa options:vdpa-socket-path=/var/run/virtio-
forwarder/sock0 options:vdpa-accelerator-devargs=0000:af:00.2
```

> **Note**
>
> To configure SW vDPA, add `options:vdpa-sw=true` to the command.

Software vDPA Configuration in OVS-Kernel Mode

Software vDPA can also be used in configurations where hardware offload is done through TC and not DPDK.

1. OVS configuration:

```
ovs-vsctl set Open_vSwitch . other_config:dpdk-extra="-a
0000:01:00.0,representor=[0],dv_flow_en=1,dv_esw_en=0,idv_xmeta_en=0,isolated_mode=1"
/usr/share/openvswitch/scripts/ovs-ctl restart
```

2. Create OVS-DPDK bridge:

```
ovs-vsctl add-br br0-ovs -- set bridge br0-ovs
datapath_type=netdev
```

3. Create vDPA port as part of the OVS-DPDK bridge:

```
ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0
type=dpdkvdpa options:vdpa-socket-path=/var/run/virtio-
forwarder/sock0 options:vdpa-accelerator-devargs=0000:01:00.2
options:dpdk-devargs=0000:01:00.0,representor=[0] options:
vdpa-max-queues=8
```

4. Create Kernel bridge:

```
ovs-vsctl add-br br-kernel
```

5. Add representors to Kernel bridge:

```
ovs-vsctl add-port br-kernel enp1s0f0_0
ovs-vsctl add-port br-kernel enp1s0f0
```

Large MTU/Jumbo Frame Configuration

To configure MTU/jumbo frames:

1. Verify that the Kernel version on the VM is 4.14 or above:

```
cat /etc/redhat-release
```

2. Set the MTU on both physical interfaces in the host:

```
ifconfig ens4f0 mtu 9216
```

3. Send a large size packet and verify that it is sent and received correctly:

```
tcpdump -i ens4f0 -nev icmp &
ping 11.100.126.1 -s 9188 -M do -c 1
```

4. Enable `host_mtu` in XML and add the following values:

```
host_mtu=9216,csum=on,guest_csum=on,host_tso4=on,host_tso6=on
```

Example:

```
<qemu:commandline>
<qemu:arg value='-chardev'/>
<qemu:arg value='socket,id=charnet1,path=/tmp/sock0,server'/>
<qemu:arg value='-netdev'/>
<qemu:arg value='vhost-user,chardev=charnet1,queues=16,id=hostnet1'/>
<qemu:arg value='-device'/>
<qemu:arg value='virtio-net-
pci,mq=on,vectors=34,netdev=hostnet1,id=net1,mac=00:21:21:24:02:01,bus=pci.0,addr=0xC,page-
per-
vq=on,rx_queue_size=1024,tx_queue_size=1024,host_mtu=9216,csum=on,guest_csum=on,host_tso4=o
```

```
</qemu:commandline>
```

5. Add the `mtu_request=9216` option to the OVS ports inside the container and restart the OVS:

```
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk
options:dpdk-devargs=0000:c4:00.0 mtu_request=9216
```

Or:

```
ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0
type=dpdkvdpa options:vdpa-socket-path=/tmp/sock0
options:vdpa-accelerator-devargs=0000:c4:00.2 options:dpdk-
devargs=0000:c4:00.0,representor=[0] mtu_request=9216
/usr/share/openvswitch/scripts/ovs-ctl restart
```

6. Start the VM and configure the MTU on the VM:

```
ifconfig eth0 11.100.124.2/16 up
ifconfig eth0 mtu 9216
ping 11.100.126.1 -s 9188 -M do -c1
```

E2E Cache

Note

This feature is supported at beta level.

OVS offload rules are based on a multi-table architecture. E2E cache enables merging the multi-table flow matches and actions into one joint flow.

This improves CT performance by using a single-table when an exact match is detected.

To set the E2E cache size (default is 4k):

```
ovs-vsctl set open_vswitch . other_config:e2e-size=<size>
systemctl restart openvswitch
```

To enable E2E cache (disabled by default):

```
ovs-vsctl set open_vswitch . other_config:e2e-enable=true
systemctl restart openvswitch
```

To run E2E cache statistics:

```
ovs-appctl dpctl/dump-e2e-stats
```

To run E2E cache flows:

```
ovs-appctl dpctl/dump-e2e-flows
```

Geneve Encapsulation/Decapsulation

Geneve tunneling offload support includes matching on extension header.

To configure OVS-DPDK Geneve encap/decap:

1. Create a br-phy bridge:

```
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy
datapath_type=netdev -- br-set-external-id br-phy bridge-id
br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach PF interface to br-phy bridge:

```
ovs-vsctl add-port br-phy pf -- set Interface pf type=dpdk
options:dpdk-devargs=<PF PCI>
```

3. Configure IP to the bridge:

```
ifconfig br-phy <$local_ip_1> up
```

4. Create a br-int bridge:

```
ovs-vsctl --may-exist add-br br-int -- set Bridge br-int
datapath_type=netdev -- br-set-external-id br-int bridge-id
br-int -- set bridge br-int fail-mode=standalone
```

5. Attach representor to br-int:

```
ovs-vsctl add-port br-int rep$x -- set Interface rep$x
type=dpdk options:dpdk-devargs=<PF PCI>,representor=[$x]
```

6. Add a port for the Geneve tunnel:

```
ovs-vsctl add-port br-int geneve0 -- set interface geneve0
type=geneve options:key=<VNI> options:remote_ip=
<$remote_ip_1> options:local_ip=<$local_ip_1>
```

Parallel Offloads

OVS-DPDK supports parallel insertion and deletion of offloads (flow and CT). While multiple threads are supported (only one is used by default).

To configure multiple threads:

```
ovs-vsctl set Open_vSwitch . other_config:n-offload-threads=3
systemctl restart openvswitch
```

Note

Refer to the OVS user manual for more information.

sFlow

sFlow allows monitoring traffic sent between two VMs on the same host using an sFlow collector.

To sample all traffic over the OVS bridge, run the following:

```
# ovs-vsctl -- --id=@sflow create sflow agent=\"$SFLOW_AGENT\" \

target=\"$SFLOW_TARGET:$SFLOW_HEADER\" \
```

```
                                             header=$SFLOW_HEADER \
                                             sampling=$SFLOW_SAMPLING
polling=10 \
                -- set bridge sflow=@sflow
```

Parameter	Description
`SFLOW_AGENT`	Indicates that the sFlow agent should send traffic from `SFLOW_AGENT`'s IP address
`SFLOW_TARGET`	Remote IP address of the sFlow collector
`SFLOW_PORT`	Remote IP destination port of the sFlow collector
`SFLOW_HEADER`	Size of packet header to sample (in bytes)
`SFLOW_SAMPLING`	Sample rate

To clear the sFlow configuration, run:

```
# ovs-vsctl clear bridge br-vxlan mirrors
```

> **Note**
>
> Currently sFlow for OVS-DPDK is supported without CT.

CT CT NAT

To enable ct-ct-nat offloads in OVS-DPDK (disabled by default), run:

```
ovs-vsctl set open_vswitch . other_config:ct-action-on-nat-conns=true
```

If disabled, ct-ct-nat configurations are not fully offloaded, improving connection offloading rate for other cases (ct and ct-nat).

If enabled, ct-ct-nat configurations are fully offloaded but ct and ct-nat offloading would be slower to create.

OpenFlow Meters (OpenFlow13+)

OpenFlow meters in OVS are implemented according to RFC 2697 (Single Rate Three Color Marker—srTCM).

- The srTCM meters an IP packet stream and marks its packets either green, yellow, or red. The color is decided on a Committed Information Rate (CIR) and two associated burst sizes, Committed Burst Size (CBS), and Excess Burst Size (EBS).
- A packet is marked green if it does not exceed the CBS, yellow if it exceeds the CBS but not the EBS, and red otherwise.
- The volume of green packets should never be smaller than the CIR.

To configure a meter in OVS:

1. Create a meter over a certain bridge, run:

   ```
   ovs-ofctl -O openflow13 add-meter $bridge
   meter=$id,$pktps/$kbps,band=type=drop,rate=$rate,
   [burst,burst_size=$burst_size]
   ```

   Parameters:

Parameter	Description
`bridge`	Name of the bridge on which the meter should be applied.
`id`	Unique meter ID (32 bits) to be used as an identifier for the meter.
`pktps` / `kbps`	Indication if the meter should work according to packets or kilobits per second.
`rate`	Rate of `pktps`/`kbps` of allowed data transmission.
`burst`	If set, enables burst support for meter bands through the `burst_size` parameter.
`burst_size`	If burst is specified for the meter entry, configures the maximum burst allowed for the band in kilobits/packets, depending on whether `kbps` or `pktps` has been specified. If unspecified, the switch is free to select some reasonable value depending on its configuration. Currently, if burst is not specified, the `burst_size` parameter is set the same as `rate`.

2. Add the meter to a certain OpenFlow rule. For example:

```
ovs-ofctl -O openflow13 add-flow $bridge
"table=0,actions=meter:$id,normal"
```

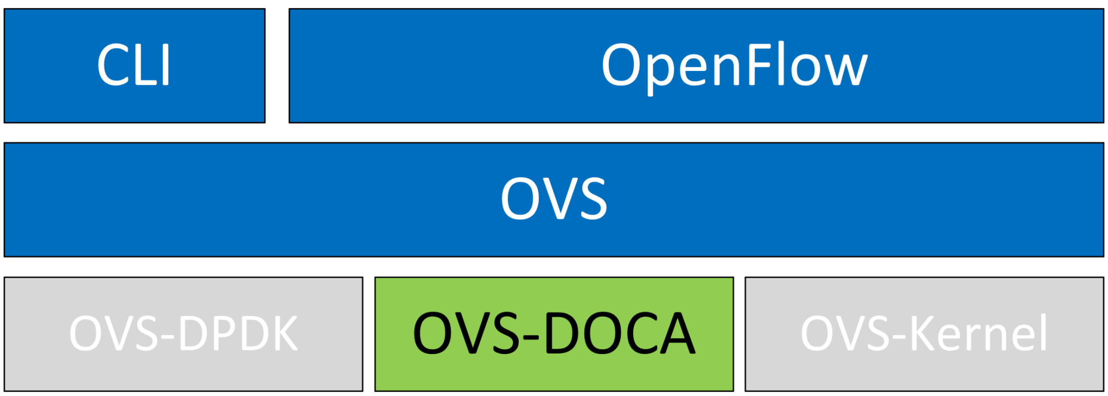
3. View the meter statistics:

```
ovs-ofctl -O openflow13 meter-stats $bridge meter=$id
```

4. For more information, refer to official OVS documentation.

OVS-DOCA Hardware Offloads

OVS-DOCA is designed on top of NVIDIA's networking API to preserve the same OpenFlow, CLI, and data interfaces (e.g., vdpa, VF passthrough), and northbound API as OVS-DPDK and OVS-Kernel. While all OVS flavors make use of flow offloads for hardware acceleration, due to its architecture and use of DOCA libraries, the OVS-DOCA mode provides the most efficient performance and feature set among them, making the most out of NVIDA NICs and DPUs.



The following subsections provide the necessary steps to launch/deploy OVS DOCA.

Configuring OVS-DOCA

To configure OVS DOCA HW offloads:

1. Unbind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
```

> **Note**

VMs with attached VFs must be powered off to be able to unbind the VFs.

2. Change the e-switch mode from `legacy` to `switchdev` on the PF device (make sure all VFs are unbound):

```
echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

Note

This command also creates the VF representor netdevices in the host OS.

To revert to SR-IOV `legacy` mode:

```
echo legacy > /sys/class/net/enp4s0f0/compat/devlink/mode
```

3. Bind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

4. Configure huge pages:

```
mkdir -p /hugepages
mount -t hugetlbfs hugetlbfs /hugepages
```

```
echo 4096 >
/sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
```

5. Run the Open vSwitch service:

```
systemctl start openvswitch
```

6. Enable DOCA mode and hardware offload (disabled by default):

```
ovs-vsctl --no-wait set Open_vSwitch . other_config:doca-
init=true
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

7. Restart the Open vSwitch service.

```
systemctl restart openvswitch
```

> **Info**
>
> This step is required for HW offload changes to take effect.

8. Create OVS-DOCA bridge:

```
ovs-vsctl --no-wait add-br br0-ovs -- set bridge br0-ovs
datapath_type=netdev
```

9. Add PF to OVS:

```
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk
options:dpdk-devargs=0000:88:00.0,dv_flow_en=2,dv_xmeta_en=4
```

> **Info**
>
> OVS-DOCA uses DPDK ports and configuration. Note the different dpdk-devargs parameters.

10. Add representor to OVS:

```
ovs-vsctl add-port br0-ovs representor -- set Interface
representor type=dpdk options:dpdk-
devargs=0000:88:00.0,representor=[<vf-
number>],dv_flow_en=2,dv_xmeta_en=4
```

> **Note**
>
> Note that `<vf-number>` must be replaced by the number of the VF.

11. Optional configuration:

    1. To set port MTU, run:

```
ovs-vsctl set interface pf mtu_request=9000
```

> **Note**
>
> OVS restart is required for changes to take effect.

2. To set VF/SF MAC, run:

```
ovs-vsctl add-port br0-ovs representor -- set Interface
representor type=dpdk options:dpdk-
devargs=0000:88:00.0,representor=[<vf-
number>],dv_flow_en=2,dv_xmeta_en=4 options:dpdk-vf-
mac=00:11:22:33:44:55
```

> **Note**
>
> Unbinding and rebinding the VFs/SFs is required for the change to take effect.

Notable Differences Between OVS-DPDK and OVS-DOCA

OVS-DOCA shares most of its structure with OVS-DPDK. To benefit from the DOCA offload design, some of the behavior of userland datapath and ports are however modified.

Eswitch Dependency

Configured in `switchdev` mode, the physical port and all supported functions share a single general domain to execute the offloaded flows, the `eswitch`.

All ports on the same eswitch are dependent on its physical function. If this main physical function is deactivated (e.g., removed from OVS or its link set down), dependent ports are disabled as well.

Pre-allocated Offload Tables

To offer the highest insertion speed, DOCA offloads pre-allocate offload structures (entries and containers).

When starting the vSwitch daemon, offloads are thus configured with sensible defaults. If different numbers of offloads are required, configuration entries specific to OVS-DOCA are available and are described in the next section.

Unsupported CT-CT-NAT

The special ct-ct-nat mode that can be configured in OVS-kernel and OVS-DPDK is not supported by OVS-DOCA.

OVS-DOCA Specific vSwitch Configuration

The following configuration is particularly useful or specific to OVS-DOCA mode.

> **Info**
>
> The full list of OVS vSwitch configuration is documented in `man ovs-vswitchd.conf.db`.

other_config

The following table provides `other_config` configurations which are global to the vSwitch (non-exhaustive list, check manpage for more):

Configuration	Description
`other_config:doca-init`	• Optional string, either true or false • Set this value to true to enable DOCA Flow HW offload • The default value is false. Changing this value requires restarting the daemon. • This is only relevant for userspace datapath
`other_config:hw-offload-ct-size`	• Optional string, containing an integer, at least 0 • Only for the DOCA offload provider on netdev datapath • Configure the usable amount of connection tracking (CT) offload entries • The default value is 250000. Changing this value requires restarting the daemon. • Setting a value of 0 disables CT offload • Changing this configuration affects the OVS memory usage as CT tables are allocated on OVS start
`other_config:hw-offload-ct-ipv6-enabled`	• Optional string, either true or false • Only for the DOCA offload provider on netdev datapath • Set this value to true to enable IPv6 CT offload • The default value is false. Changing this value requires restarting the daemon. • Changing this configuration affects the OVS memory usage as CT tables are allocated on OVS start
`other_config:doca-congestion-threshold`	• Optional string, containing an integer, in range 30 to 90 • The occupancy rate of DOCA offload structures that triggers a resize, as a percentage • Default to 80, but only relevant if `other_config:doca-init` is true. Changing this value requires restarting the daemon.
`other_config:ctl-pipe-size`	• Optional string, containing an integer • The initial size of DOCA control pipes • Default to 0, which is DOCA's internal default value
`other_config:ctl-pipe-infra-size`	• Optional string, containing an integer • The initial size of infrastructure DOCA control pipes: root, post-hash, post-ct, post-meter, split, miss.

Configuration	Description
	• Default to 0, which fallbacks to `other_config:ctl-pipe-size`
`other_config:pmd-quiet-idle`	• Optional string, either true or false • Allow the PMD threads to go into quiescent mode when idling. If no packets are received or waiting to be processed and sent, enter a continuous quiescent period. End this period as soon as a packet is received. • This option is disabled by default
`other_config:pmd-maxsleep`	• Optional string, containing an integer, in range 0 to 10,000 • Specifies the maximum sleep time in microseconds per iteration for a PMD thread which has received zero or a small amount of packets from the Rx queues it is polling. • The actual sleep time requested is based on the load of the Rx queues that the PMD polls and may be less than the maximum value • The default value is 0 microseconds, which means that the PMD does not sleep regardless of the load from the Rx queues that it polls • To avoid requesting very small sleeps (e.g., less than 10 μs) the value is rounded up to the nearest 10 μs • The maximum value is 10000 microseconds.
`other_config:dpdk-max-memzones`	• Optional string, containing an integer • Specifies the maximum number of memzones that can be created in DPDK • The default is empty, keeping DPDK's default. Changing this value requires restarting the daemon.

netdev-dpdk

The following table provides `netdev-dpdk` configurations which only userland (DOCA or DPDK) netdevs support (non-exhaustive list, check manpage for more):

Configuration	Description
`options:iface-name`	• Specifies the interface name of the port • Providing this option accelerates processing the port reconfiguration by querying the sysfs to check if the interface exists before DPDK attempts to probe the port

Offloading VXLAN Encapsulation/Decapsulation Actions

vSwitch in userspace rather than kernel-based Open vSwitch requires an additional bridge. The purpose of this bridge is to allow use of the kernel network stack for routing and ARP resolution.

The datapath must look up the routing table and ARP table to prepare the tunnel header and transmit data to the output port.

VXLAN encapsulation/decapsulation offload configuration is done with:

- PF on `0000:03:00.0` PCIe and MAC `98:03:9b:cc:21:e8`
- Local IP `56.56.67.1` – the `br-phy` interface is configured to this IP
- Remote IP `56.56.68.1`

To configure OVS DOCA VXLAN:

1. Create a `br-phy` bridge:

```
ovs-vsctl add-br br-phy -- set Bridge br-phy
datapath_type=netdev -- br-set-external-id br-phy bridge-id
br-phy -- set bridge br-phy fail-mode=standalone
other_config:hwaddr=98:03:9b:cc:21:e8
```

2. Attach PF interface to `br-phy` bridge:

```
ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk
options:dpdk-devargs=0000:03:00.0,dv_flow_en=2,dv_xmeta_en=4
```

3. Configure IP to the bridge:

```
ip addr add 56.56.67.1/24 dev br-phy
```

4. Create a `br-ovs` bridge:

```
ovs-vsctl add-br br-ovs -- set Bridge br-ovs
datapath_type=netdev -- br-set-external-id br-ovs bridge-id
br-ovs -- set bridge br-ovs fail-mode=standalone
```

5. Attach representor to `br-ovs`:

```
ovs-vsctl add-port br-ovs pf0vf0 -- set Interface pf0vf0
type=dpdk options:dpdk-devargs=0000:03:00.0,representor=
[0],dv_flow_en=2,dv_xmeta_en=4
```

6. Add a port for the VXLAN tunnel:

```
ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0
type=vxlan options:local_ip=56.56.67.1
options:remote_ip=56.56.68.1 options:key=45
options:dst_port=4789
```

Offloading Connection Tracking

Connection tracking enables stateful packet processing by keeping a record of currently open connections.

OVS flows utilizing connection tracking can be accelerated using advanced NICs by offloading established connections.

To view offload statistics, run:

```
ovs-appctl dpctl/offload-stats-show
```

SR-IOV VF LAG

To configure OVS-DOCA SR-IOV VF LAG:

1. Enable SR-IOV on the NICs:

   ```
   mlxconfig -d <PCI> set SRIOV_EN=1
   ```

2. Allocate the desired number of VFs per port:

   ```
   echo $n > /sys/class/net/<net name>/device/sriov_numvfs
   ```

3. Unbind all VFs:

   ```
   echo <VF PCI> >/sys/bus/pci/drivers/mlx5_core/unbind
   ```

4. Change both NICs' mode to SwitchDev:

```
devlink dev eswitch set pci/<PCI> mode switchdev
```

5. Create Linux bonding using kernel modules:

```
modprobe bonding mode=<desired mode>
```

> **Note**
>
> Other bonding parameters can be added here. The supported bond modes are Active-Backup, XOR, and LACP.

6. Bring all PFs and VFs down:

```
ip link set <PF/VF> down
```

7. Attach both PFs to the bond:

```
ip link set <PF> master bond0
```

8. Bring PFs and bond link up:

```
ip link set <PF0> up
ip link set <PF1> up
ip link set bond0 up
```

9. To work with VF-LAG with OVS-DPDK, add the bond master (PF) to the bridge:

```
ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk
options:dpdk-devargs=0000:03:00.0,dv_flow_en=2,dv_xmeta_en=4
options:dpdk-lsc-interrupt=true
```

Add representor `$N` of PF0 or PF1 to a bridge:

```
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N type=dpdk
options:dpdk-devargs=<PF0-
PCI>,representor=pf0vf$N,dv_flow_en=2,dv_xmeta_en=4
```

Or:

```
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N type=dpdk
options:dpdk-devargs=<PF0-
PCI>,representor=pf1vf$N,dv_flow_en=2,dv_xmeta_en=4
```

Multiport eSwitch Mode

Multiport eswitch mode allows adding rules on a VF representor with an action, forwarding the packet to the physical port of the physical function. This can be used to implement failover or to forward packets based on external information such as the cost of the route.

1. To configure multiport eswitch mode , the nvconig parameter `LAG_RESOURCE_ALLOCATION=1` must be set.

2. After the driver loads, and before moving to switchdev mode, configure multiport eswitch for each PF where p0 and p1 represent the netdevices for the PFs:

```
devlink dev param set pci/0000:03:00.0 name esw_multiport value 1 cmode runtime
devlink dev param set pci/0000:03:00.1 name esw_multiport value 1 cmode runtime
```

> **Info**
>
> The mode becomes operational after entering switchdev mode on both PFs.

3. This mode can be activated by default in BlueField by adding the following line into `/etc/mellanox/mlnx-bf.conf`:

```
ENABLE_ESWITCH_MULTIPORT="yes"
```

While in this mode, the second port is not an eswitch manager, and should be add to OVS using this command:

```
ovs-vsctl add-port br-phy p1 -- set interface p1 type=dpdk options:dpdk-devargs="0000:08:00.0,dv_xmeta_en=4,dv_flow_en=2,representor=pf1
```

VFs for the second port can be added using this command:

```
ovs-vsctl add-port br-phy p1vf0 -- set interface p1 type=dpdk options:dpdk-devargs="0000:08:00.0,dv_xmeta_en=4,dv_flow_en=2,representor=pf1vf0
```

Offloading Geneve Encapsulation/Decapsulation

Geneve tunneling offload support includes matching on extension header.

> **Note**
>
> OVS-DOCA Geneve option limitations:
>
> - Only 1 Geneve option is supported
> - Max option len is 7
> - To change the Geneve option currently being matched and encapsulated, users must remove all ports or restart OVS and configure the new option
> - Users must change firmware configuration to enable the flex parser by running the following commands:
>
> ```
> mst start
> mlxconfig -d <mst device> s
> FLEX_PARSER_PROFILE_ENABLE=8
> mlxfwreset -d <mst device> r -y
> ```

To configure OVS-DOCA Geneve encapsulation/decapsulation:

1. Create a `br-phy` bridge:

```
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy
datapath_type=netdev -- br-set-external-id br-phy bridge-id
br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to `br-phy` bridge:

```
ovs-vsctl add-port br-phy pf -- set Interface pf type=dpdk
options:dpdk-devargs=<PF PCI>,dv_flow_en=2,dv_xmeta_en=4
```

3. Configure an IP to the bridge:

```
ifconfig br-phy <$local_ip_1> up
```

4. Create a `br-int` bridge:

```
ovs-vsctl --may-exist add-br br-int -- set Bridge br-int
datapath_type=netdev -- br-set-external-id br-int bridge-id
br-int -- set bridge br-int fail-mode=standalone
```

5. Attach a representor to `br-int`:

```
ovs-vsctl add-port br-int rep$x -- set Interface rep$x
type=dpdk options:dpdk-devargs=<PF PCI>,representor=
[$x],dv_flow_en=2,dv_xmeta_en=4
```

6. Add a port for the Geneve tunnel:

```
ovs-vsctl add-port br-int geneve0 -- set interface geneve0
type=geneve options:key=<VNI> options:remote_ip=
<$remote_ip_1> options:local_ip=<$local_ip_1>
```

GRE Tunnel Offloads

To configure OVS-DOCA GRE encapsulation/decapsulation:

1. Create a `br-phy` bridge:

```
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy
datapath_type=netdev -- br-set-external-id br-phy bridge-id
br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to `br-phy` bridge:

```
ovs-vsctl add-port br-phy pf -- set Interface pf type=dpdk
options:dpdk-devargs=<PF PCI>,dv_flow_en=2,dv_xmeta_en=4
```

3. Configure an IP to the bridge:

```
ifconfig br-phy <$local_ip_1> up
```

4. Create a `br-int` bridge:

```
ovs-vsctl --may-exist add-br br-int -- set Bridge br-int
datapath_type=netdev -- br-set-external-id br-int bridge-id
br-int -- set bridge br-int fail-mode=standalone
```

5. Attach a representor to `br-int`:

```
ovs-vsctl add-port br-int rep$x -- set Interface rep$x
type=dpdk options:dpdk-devargs=<PF PCI>,representor=
```

```
[$x],dv_flow_en=2,dv_xmeta_en=4
```

6. Add a port for the Geneve tunnel:

```
ovs-vsctl add-port br-int gre0 -- set interface gre0 type=gre options:key=<VNI> options:remote_ip=<$remote_ip_1> options:local_ip=<$local_ip_1>
```

DP-HASH Offloads

OVS supports group configuration. The "select" type executes one bucket in the group, balancing across the buckets according to their weights. To select a bucket, for each live bucket, OVS hashes flow data with the bucket ID and multiplies that by the bucket weight to obtain a "score". The bucket with the highest score is selected.

> **Info**
>
> For more details, refer to the ovs-ofctl man.

For example:

- `ovs-ofctl add-group br-int 'group_id=1,type=select,bucket=<port1>'`
- `ovs-ofctl add-flow br-int in_port=<port0>,actions=group=1`

Limitations:

- Offloads are supported on IP traffic only (IPv4 or IPv6)
- The hash calculation may be different for packets going into software vs. ones that are offloaded

- Does not work concurrently with CT (i.e., configure `hw-offload-ct-size="0"` beforehand)

OVS-DOCA Known Limitations

- Only one insertion thread is supported (n-offload-threads=1)
- Only 250K connection are offloadable by default (can be configured)
- Only 8 CT zones are supported by CT offload
- Offload of IPv6 tunnels are not supported

OVS-DOCA Debugging

Additional debugging information can be enabled in the vSwitch log file using the `dbg` log level:

```
(
    topics='netdev|ofproto|ofp|odp|doca'
    IFS=$'\n'; for topic in $(ovs-appctl vlog/list | grep -E "$topics" | cut -d' ' -f1)
    do
        printf "$topic:file:dbg "
    done
) | xargs ovs-appctl vlog/set
```

The listed topics are relevant to DOCA offload operations.

Coverage counters specific to the DOCA offload provider have been added. The following command should be used to check them:

```
ovs-appctl coverage/show # Print the current non-zero coverage
```

```
counters
```

The following table provides the meaning behind these DOCA-specific counters:

Counter	Description
`doca_async_queue_full`	The asynchronous offload insertion queue was full while the daemon attempted to insert a new offload. The queue will have been flushed and insertion attempted again. This is not a fatal error but is the sign of a slowed down hardware.
`doca_async_queue_blocked`	The asynchronous offload insertion queue has remained full even after several attempts to flush its currently enqueued requests. While not a fatal error, it should never happen during normal offload operations and should be considered a bug.
`doca_async_add_failed`	An asynchronous insertion failed specifically due to its asynchronous nature. This is not expected to happen and should be considered a bug.
`doca_pipe_resize`	The number of time a DOCA pipe has been resized. This is normal and expected as DOCA pipes receives more entries.
`doca_pipe_resize_over_10_ms`	A DOCA pipe resize took longer than 10ms to complete. It can happen infrequently. If a sudden drop in insertion rate is measured, this counter could help identify the root cause.

OVS Metrics

OVS exposes Prometheus metrics through its control socket (experimental feature). These metrics can be accessed using the command:

```
ovs-appctl metrics/show
```

A terminal dashboard is also installed with OVS, `ovs-metrics`. This script is dependent on the OVS Python API (package `python3-openvswitch`). Its default mode currently watches over a set of offload-related metrics.

OVS Inside the DPU

Verifying Host Connection on Linux

When the DPU is connected to another DPU on another machine, manually assign IP addresses with the same subnet to both ends of the connection.

1. Assuming the link is connected to p3p1 on the other host, run:

```
$ ifconfig p3p1 192.168.200.1/24 up
```

2. On the host which the DPU is connected to, run:

```
$ ifconfig p4p2 192.168.200.2/24 up
```

3. Have one ping the other. This is an example of the DPU pinging the host:

```
$ ping 192.168.200.1
```

Verifying Connection from Host to BlueField

There are two SFs configured on the BlueFIeld-2 device, `enp3s0f0s0` and `enp3s0f1s0`, and their representors are part of the built-in bridge. These interfaces will get IP addresses from the DHCP server if it is present. Otherwise it is possible to configure IP address from the host. It is possible to access BlueField via the SF netdev interfaces.

For example:

1. Verify the default OVS configuration. Run:

```
# ovs-vsctl show
5668f9a6-6b93-49cf-a72a-14fd64b4c82b
    Bridge ovsbr1
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port p0
            Interface p0
        Port en3f0pf0sf0
            Interface en3f0pf0sf0
    Bridge ovsbr2
        Port en3f1pf1sf0
            Interface en3f1pf1sf0
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port pf1hpf
            Interface pf1hpf
        Port p1
            Interface p1
    ovs_version: "2.14.1"
```

2. Verify whether the SF netdev received an IP address from the DHCP server. If not, assign a static IP. Run:

```
# ifconfig enp3s0f0s0
enp3s0f0s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.200.125  netmask 255.255.255.0  broadcast 192.168.200.255
```

```
        inet6 fe80::8e:bcff:fe36:19bc  prefixlen 64  scopeid 
0x20<link>
        ether 02:8e:bc:36:19:bc  txqueuelen 1000  (Ethernet)
        RX packets 3730  bytes 1217558 (1.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22  bytes 2220 (2.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0 
collisions 0
```

3. Verify the connection of the configured IP address. Run:

```
# ping 192.168.200.25 -c 5
PING 192.168.200.25 (192.168.200.25) 56(84) bytes of data.
64 bytes from 192.168.200.25: icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 192.168.200.25: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 192.168.200.25: icmp_seq=3 ttl=64 time=0.232 ms
64 bytes from 192.168.200.25: icmp_seq=4 ttl=64 time=0.174 ms
64 bytes from 192.168.200.25: icmp_seq=5 ttl=64 time=0.168 ms

--- 192.168.200.25 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 91ms
rtt min/avg/max/mdev = 0.168/0.195/0.232/0.031 ms
```

Verifying Host Connection on Windows

Set IP address on the Windows side for the RShim or Physical network adapter, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> New-NetIPAddress -InterfaceAlias 
"Ethernet 16" -IPAddress "192.168.100.1" -PrefixLength 22
```

To get the interface name, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> Get-NetAdapter
```

Output should give us the interface name that matches the description (e.g. NVIDIA BlueField Management Network Adapter).

```
Ethernet 2                     NVIDIA ConnectX-4 Lx Ethernet
Adapter          6 Not Present  24-8A-07-0D-E8-1D
Ethernet 6                     NVIDIA ConnectX-4 Lx Ethernet
Ad...#2         23 Not Present  24-8A-07-0D-E8-1C
Ethernet 16                    NVIDIA BlueField Management
Netw...#2               15 Up                  CA-FE-01-CA-FE-02
```

Once IP address is set, Have one ping the other.

```
C:\Windows\system32>ping 192.168.100.2

Pinging 192.168.100.2 with 32 bytes of data:
Reply from 192.168.100.2: bytes=32 time=148ms TTL=64
Reply from 192.168.100.2: bytes=32 time=152ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
```

order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
