NVIDIA DOCA Flow

Sample Guide

MLNX-15-060525 _v1.4 | November 2022

Table of Contents

Chapter 1. Introduction

DOCA Flow is the most fundamental API for building generic execution pipes in HW.

The library provides an API for building a set of pipes, where each pipe consists of match criteria, monitoring, and a set of actions. Pipes can be chained so that after a pipe-defined action is executed, the packet may proceed to another pipe.

For more information about DOCA Flow library, refer to NVIDIA DOCA Flow Programming Guide.

Chapter 2. Dependencies

N/A

Chapter 3. Prerequisites

The DOCA Flow samples are based on DPDK libraries. Therefore, the user is required to provide DPDK flags, and allocate huge pages.

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

Chapter 4. Running the Sample

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA samples.
2. To build a given sample:
   ```
   cd /opt/mellanox/doca/samples/doca_flow/<sample_name>
   meson build
   ninja -C build
   ```

   Note: The binary `doca_<sample_name>` will be created under `./build/`.
3. Sample (e.g., `flow_aging`) usage:
   ```
   Usage: doca_flow_aging [DPDK Flags] -- [DOCA Flags]
   DOCA Flags:
     -h, --help                    Print a help synopsis
     -v, --version                 Print program version information
     -l, --log-level               Set the log level for the program <CRITICAL=20,
    ERROR=30, WARNING=40, INFO=50, DEBUG=60>
   ```
4. For additional information per sample, use the `-h` option after the `--` separator:
   ```
   ./build/doca_<sample_name> -- -h
   ```
5. DOCA Flow samples are based on DPDK libraries. Therefore, the user is required to provide DPDK flags. The following is an example from an execution on the DPU:
   - CLI example for running the samples with "vnf" mode:
     ```
     ./build/doca_<sample_name> -a auxiliary:mlx5_core.sf.2 -a
      auxiliary:mlx5_core.sf.3 -- -l 60
     ```
   - CLI example for running the samples with "vnf,hws" mode:
     ```
     ./build/doca_<sample_name> -a auxiliary:mlx5_core.sf.2,dv_flow_en=2 -a
      auxiliary:mlx5_core.sf.3,dv_flow_en=2 -- -l 60
     ```

     Note: When running on the DPU using the command above, sub-functions must be enabled according to the Scalable Function Setup Guide.

     Note: When running on the host, virtual functions must be used according to the instructions in the NVIDIA DOCA Virtual Functions User Guide.

Chapter 5. Samples

5.1. Flow Aging

This sample illustrates the use of DOCA Flow's aging functionality. It demonstrates how to build a pipe and add different entries with different aging times and user data.

The sample logic includes:

1. Initializing DOCA Flow with `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow port.
3. On each port:
   a). Building a pipe with changeable 5-tuple match and forward port action.
   b). Adding 10 entries with different 5-tuple match, a monitor with different aging time (5-60 seconds), and setting user data in the monitor. The user data will contain the port ID, entry number, and entry pointer.
4. Handling aging every 5 seconds and removing each entry after age-out.
5. Running these commands until all entries age out.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_aging/flow_aging_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_aging/flow_aging_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_aging/meson.build`

5.2. Flow Control Pipe

This sample shows how to use the DOCA Flow control pipe and decap action.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:

a). Building VXLAN pipe with match on VNI field, decap action, and forwarding the matched packets to the second port.

b). Building GRE pipe with match on GRE key field, decap and build `eth` header actions, and forwarding the matched packets to the second port.

c). Building a control pipe with the following entries:

- If L4 type is UDP and destination port is 4789, forward to VXLAN pipe
- If tunnel type and L4 type is GRE, forward to GRE pipe

Note: When GRE tunnel is decapped, the complete outer layer is removed (including L2) and the inner layer (IP layer) is exposed. To keep the packet valid, you must also modify the layer 2 source/destination MAC addresses and VLAN may optionally be modified. For example:

```
actions.decap = true;
/* append eth header after decap GRE tunnel */
SET_MAC_ADDR(actions.mod_src_mac, src_mac[0], src_mac[1], src_mac[2],
 src_mac[3], src_mac[4], src_mac[5]);
SET_MAC_ADDR(actions.mod_dst_mac, dst_mac[0], dst_mac[1], dst_mac[2],
 dst_mac[3], dst_mac[4], dst_mac[5]);
actions.mod_dst_ip.type = DOCA_FLOW_IP4_ADDR;
```

The same requirement applies to GTP tunnels or any other tunnel protocol which lacks an L2 layer.

However, for VXLAN it is not necessary to add an L2 since VXLAN itself already includes the L2, so the packet remains valid even after it is decapped.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_control_pipe/flow_control_pipe_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_control_pipe/flow_control_pipe_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_control_pipe/meson.build`

5.3. Flow Copy to Meta

This sample shows how to use the DOCA Flow copy-to-metadata action to copy the source MAC address and then match on it.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:

   a). Building a pipe with changeable match on `meta_data` and forwarding the matched packets to the second port.

   b). Adding an entry that matches an example source MAC that has been copied to metadata.

c). Building a pipe with changeable 5-tuple match, copying source MAC action, and fwd to the first pipe.

d). Adding example 5-tuple entry to the pipe.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_copy_to_meta/flow_copy_to_meta_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_copy_to_meta/flow_copy_to_meta_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_copy_to_meta/meson.build`

5.4. Flow Drop

This sample illustrates how to build a pipe with 5-tuple match, forward action drop, and forward miss action to hairpin pipe. The sample also demonstrates how to dump pipe information to a file and query entry.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:
   a). Building a hairpin pipe with an entry that matches all traffic and forwarding traffic to the second port.
   b). Building a pipe with a changeable 5-tuple match, forwarding action drop and miss forward to the hairpin pipe. This pipe serves as a root pipe.
   c). Adding example 5-tuple entry to the drop pipe with counter as monitor for query the entry later.
4. Waiting 5 seconds and querying the drop entry (total bytes and total packets).
5. Dumping the pipe information to a file.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_drop/flow_drop_sample_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_drop/flow_drop_sample_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_drop/meson.build`

5.5. Flow Hairpin

This sample illustrates how to build a pipe with 5-tuple match and to forward packets to the other port.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.

2. Starting two DOCA Flow ports.
3. On each port:

   a). Building a pipe with changeable 5-tuple match and forwarding port action.

   b). Adding example 5-tuple entry to the pipe.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_hairpin/flow_hairpin_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_hairpin/flow_hairpin_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_hairpin/meson.build`

5.6. Flow LPM

This sample illustrates how to use LPM (Longest Prefix Match) pipe.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:

   a). Building an LPM pipe that matches changeable source IPv4 address.

   b). Adding two example 5-tuple entries:

   - The first entry with full mask and forward port action
   - The second entry with 16-bit mask and drop action

   c). Building a control pipe with one entry that forwards IPv4 traffic to the LPM pipe.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_lpm/flow_lpm_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_lpm/flow_lpm_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_lpm/meson.build`

5.7. Flow Modify Header

This sample illustrates how to use DOCA Flow actions to decrease TTL by 1 and modify the destination MAC address.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:

a). Building a pipe with action `dec_ttl=true` and changeable `mod_dst_mac`. The pipe matches IPv4 traffic with a changeable destination IP and forwards the matched packets to the second port.

b). Adding an entry with an example destination IP and `mod_dst_mac` value.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_modify_header/flow_modify_header_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_modify_header/flow_modify_header_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_modify_header/meson.build`

5.8. Flow Monitor Meter

This sample illustrates how to use DOCA Flow monitor meter.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:

   a). Building a pipe with monitor meter flag and changeable 5-tuple match. The pipe forwards the matched packets to the second port.

   b). Adding an entry with an example CIR and CBS values.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_monitor_meter/flow_monitor_meter_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_monitor_meter/flow_monitor_meter_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_monitor_meter/meson.build`

5.9. Flow Multi-actions

This sample shows how to use a DOCA Flow array of actions in a pipe.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:

   a). Building a pipe with changeable source IP match which forwards the matched packets to the second port and sets different actions in the actions array:

i. Changeable modify source MAC address
ii. Changeable modify source IP address

b). Adding two entries to the pipe with different source IP match:

i. The first entry with an example modify source MAC address.
ii. The second with a modify source IP address.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_multi_actions/flow_multi_actions_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_multi_actions/flow_multi_actions_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_multi_actions/meson.build`

5.10. Flow RSS Meta

This sample shows how to use DOCA Flow forward RSS, set meta action, and then retrieve the matched packets in the sample.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:
   a). Building a pipe with a changeable 5-tuple match, forwarding to RSS queue with index 0, and setting changeable packet meta data.
   b). Adding an entry with an example 5-tuple and metadata value to the pipe.
   c).
4. Retrieving the packets on both ports from a receive queue, and printing the packet metadata value.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_rss_meta/flow_rss_meta_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_rss_meta/flow_rss_meta_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_rss_meta/meson.build`

5.11. Flow Set Meta

This sample shows how to use the DOCA Flow set metadata action and then match on it.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.

3. On each port:
   a). Building a pipe with a changeable match on metadata and forwarding the matched packets to the second port.
   b). Adding an entry that matches an example metadata value.
   c). Building a pipe with changeable 5-tuple match, changeable metadata action, and fwd to the first pipe.
   d). Adding entry with an example 5-tuple and metadata value to the pipe.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_set_meta/flow_set_meta_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_set_meta/flow_set_meta_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_set_meta/meson.build`

5.12. Flow Shared Counter

This sample shows how to use the DOCA Flow shared counter and query it to get the counter statistics.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:
   a). Binding the shared counter to the port.
   b). Building a pipe with changeable 5-tuple match with UDP protocol, changeable shared counter ID and forwarding the matched packets to the second port.
   c). Adding an entry with an example 5-tuple match and shared counter with ID=`port_id`.
   d). Building a pipe with changeable 5-tuple match with TCP protocol, changeable shared counter ID and forwarding the matched packets to the second port.
   e). Adding an entry with an example 5-tuple match and shared counter with ID=`port_id`.
   f). Building a control pipe with the following entries:
      - If L4 type is UDP, forwards the packets to the UDP pipe
      - If L4 type is TCP, forwards the packets to the TCP pipe
4. Waiting 5 seconds and querying the shared counters (total bytes and total packets).

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_shared_counter/flow_shared_counter_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_shared_counter/flow_shared_counter_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_shared_counter/meson.build`

5.13. Flow Shared Meter

This sample shows how to use the DOCA Flow shared meter.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:
   a). Config a shared meter with specific cir and cbs values.
   b). Binding the shared meter to the port.
   c). Building a pipe with a changeable 5-tuple match with UDP protocol, changeable shared meter ID and forwarding the matched packets to the second port.
   d). Adding an entry with an example 5-tuple match and shared meter with ID=`port_id`.
   e). Building a pipe with a changeable 5-tuple match with TCP protocol, changeable shared meter ID and forwarding the matched packets to the second port.
   f). Adding an entry with an example 5-tuple match and shared meter with ID=`port_id`.
   g). Building a control pipe with the following entries:
      - If L4 type is UDP, forwards the packets to the UDP pipe
      - If L4 type is TCP, forwards the packets to the TCP pipe

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_shared_meter/flow_shared_meter_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_shared_meter/flow_shared_meter_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_shared_meter/meson.build`

5.14. Flow VXLAN Encap

This sample shows how to use DOCA Flow actions to create a VXLAN tunnel.

The sample logic includes:

1. Initializing DOCA Flow by indicating `mode_args="vnf,hws"` in the `doca_flow_cfg` struct.
2. Starting two DOCA Flow ports.
3. On each port:
   a). Building a pipe with changeable 5-tuple match, encap action, and forward port action.
   b). Adding example 5-tuple and encapsulation values entry to the pipe.

Reference:

- `/opt/mellanox/doca/samples/doca_flow/flow_vxlan_encap/flow_vxlan_encap_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_vxlan_encap/flow_vxlan_encap_main.c`
- `/opt/mellanox/doca/samples/doca_flow/flow_vxlan_encap/meson.build`

5.15. Flow gRPC Counter

This sample shows how to use DOCA Flow gRPC library to create a pipe and entry with a counter and to query the entry stats.

The sample logic includes:

1. Creating gRPC environment.
2. Initializing DOCA Flow.
3. Starting two DOCA Flow ports.
4. On each port:
   a). Building a pipe with changeable 5-tuple match.
   b). Adding example 5-tuple and monitoring with counter flag.
   c). Waiting 5 seconds and querying the entries (total bytes and total packets).

References:

- `/opt/mellanox/doca/samples/doca_flow/grpc_flow_counter/grpc_flow_counter_sample.c`
- `/opt/mellanox/doca/samples/doca_flow/grpc_flow_counter/grpc_flow_counter_main.c`
- `/opt/mellanox/doca/samples/doca_flow/grpc_flow_counter/meson.build`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of 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, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com