# Table of contents

- **Introduction**
- **System Design**
- **Application Architecture**
- **DOCA Libraries**
- **Compiling the Application**
  - Compiling All Applications
  - Compiling Simple Forward Application Only
  - Troubleshooting
- **Running the Application**
  - Prerequisites
  - Application Execution
  - Command Line Flags
  - Troubleshooting
- **Application Code Flow**
- **References**
This guide provides a Simple Forward implementation on top of NVIDIA® BlueField® DPU.

**Introduction**

Simple forward is a forwarding application that leverages the DOCA Flow API to take either VXLAN, GRE, or GTP traffic from a single RX port and transmits it on a single TX port.

For every packet received on an RX queue on a given port, DOCA Simple Forward checks the packet’s key, which consists of a 5-tuple. If it finds that the packet matches an existing flow, then it does not create a new one. Otherwise, a new flow is created with a FORWARDING component. Finally, the packet is forwarded to the TX queue of the egress port if the "rx-only" mode is not set.

The FORWARDING component type depends on the flags delivered when running the application. For example, if the hairpin flag is provided, then the FORWARDING component would be hairpin. Otherwise, it would be RSS'd to software, and hence every VXLAN, GTP, or GRE packet would be received on RX queues.

Simple forward should be run with dual ports. By using a traffic generator, the RX port receives the VXLAN, GRE, or GTP packets and forwarding forwards them back to the traffic generator.

**System Design**

The following diagram illustrates simple forward's packet flows. It receives traffic coming from the wire and passes it to the other port.
Application Architecture

Simple forward first initializes DPDK, after which the application handles the incoming packets.

The following diagram illustrates the initialization process.
1. **Init DPDK** – EAL init, parse argument from command line and register signal.

2. **Start port** – `mbuf_create`, `dev_configure`, `rx/tx/hairpin queue setup` and start the port.

3. **Simple_fwd INIT** – create flow tables, build default forward pipes.

The following diagram illustrates how to process the packet.

1. Based on the packet's info, find the key values (e.g. src/dst IP, src/dst port, etc).

2. Traverse the inner flow tables, check if the keys exist or not.
   - If yes, update inner counter
   - If no, a new flow table is added to the DPU

3. Forward the packet to the other port.

**DOCA Libraries**

This application leverages the following DOCA library:

- **DOCA Flow**

Refer to its respective programming guide for more information.
Compiling the Application

⚠️ Info

Please refer to the NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

⚠️ Tip

For more information about the applications as well as development and compilation tips, refer to the DOCA Applications page.

The sources of the application can be found under the application's directory:

```
/opt/mellanox/doca/applications/simple_fwd_vnf/
```

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```

cd /opt/mellanox/doca/applications/
meson /tmp/build
ninja -C /tmp/build
```
Compiling Simple Forward Application Only

To directly build only the simple forward application:

```
cd /opt/mellanox/doca/applications/
meson /tmp/build -Denable_all_applications=false -Denable_simple_fwd_vnf=true
ninja -C /tmp/build
```

Info
doca_simple_fwd_vnf is created under /tmp/build/simple_fwd_vnf/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
   - Set enable_all_applications to false
   - Set enable_simple_fwd_vnf to true

2. Run the following compilation commands:

```
cd /opt/mellanox/doca/applications/
meson /tmp/build
```
Troubleshooting

Refer to the NVIDIA DOCA Troubleshooting Guide for any issue encountered with the compilation of the application.

Running the Application

Prerequisites

1. A FLEX profile number should be manually set to 3 on the system for the application to build the GRE, standard VXLAN and GRE pipes.

   1. Set FLEX profile number to 3 from the DPU.

   ```
   sudo mlxconfig -d <pcie_address> s FLEX_PARSER_PROFILE_ENABLE=3
   ```

2. Perform a BlueField system reboot for the mlxconfig settings to take effect.

   Info

   Resetting the firmware can be done from the BlueField as well. For more information, refer to step 3.b of the
2. The Simple Forward application is based on DOCA Flow. Therefore, the user is required to allocate huge pages.

```bash
echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

On some operating systems (RockyLinux, OpenEuler, CentOS 8.2) the default huge page size on the DPU (and Arm hosts) is larger than 2MB, and is often 512MB instead. Once can find out the size of the huge pages using the following command:

```bash
$ grep -i huge /proc/meminfo
```

```
AnonHugePages:    0 kB
ShmemHugePages:   0 kB
FileHugePages:    0 kB
HugePages_Total:  4
HugePages_Free:   4
HugePages_Rsvd:   0
HugePages_Surp:   0
Hugepagesize:     524288 kB
Hugetlb:       6291456 kB
```

Given that the guiding principal is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, one should allocate the matching amount (8 pages):

```bash
echo '8' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-524288kB/nr_hugepages
```

---

**Application Execution**
The simple forward application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_simple_forward_vnf [DPDK Flags] -- [DOCA Flags] [Program Flags]

DOCA Flags:
- -h, --help                        Print a help synopsis
- -v, --version                     Print program version information
- -l, --log-level                   Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL,
                                           30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
                                          --sdk-log-level                   Set the SDK (numeric) log level for the program <10=DISABLE,
                                           20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
- -j, --json <path>                 Parse all command flags from an input json file

Program Flags:
- -t, --stats-timer <time>          Set interval to dump stats information
- -q, --nr-queues <num>             Set queues number
- -r, --rx-only                     Set rx only
- -o, --hw-offload                  Set PCI address of the RXP engine to use
- -hq, --hairpinq                   Set forwarding to hairpin queue
- -a, --age-thread                  Start thread do aging
```

**Info**

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_simple_fwd_vnf -- -h
```
For additional information, refer to section "Command Line Flags".

2. CLI example for running the application on the BlueField:

```shell
./doca_simple_fwd_vnf -a auxiliary:mlx5_core.sf.4 -a auxiliary:mlx5_core.sf.5 -- -l 60
```

**Note**

SFs must be enabled according to the NVIDIA BlueField DPU Scalable Function User Guide.

Before creating SFs on a specific physical port, it is important to verify the encap mode on the respective PF FDB. The default mode is basic. To check the encap mode, run:

```
cat /sys/class/net/p0/compat/devlink/encap
```

In this case, disable encap on the PF FDB before creating the SFs by running:

```
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch setpci/0000:03:00.1 mode legacy
echo none > /sys/class/net/p0/compat/devlink/encap
echo none > /sys/class/net/p1/compat/devlink/encap
/opt/mellanox/iproute2/sbin/devlink dev eswitch setpci/0000:03:00.0 mode switchdev
/opt/mellanox/iproute2/sbin/devlink dev eswitch setpci/0000:03:00.1 mode switchdev
```
If the encap mode is set to basic then the application fails upon initialization.

Note

The flag -a auxiliary:mlx5_core.sf.4 -a auxiliary:mlx5_core.sf.5 is mandatory for proper usage of the application.

1. Modifying this flag results unexpected behavior as only 2 ports are supported.

2. The SF number is arbitrary and configurable.

Note

The SF numbers must match the desired SF devices.

3. CLI example for running the application on the host:

```
./doca_simple_fwd_vnf -a 04:00.3 -a 04:00.4 -- -l 60
```

Note

The device identifiers must match the desired network devices.
4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_simple_fwd_vnf --json [json_file]
```

For example:

```
./doca_simple_fwd_vnf --json ./simple_fwd_params.json
```

**Note**

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

---

**Command Line Flags**

<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
</table>
| DPDK Flags | a          | devices            | Add a PCIe device into the list of devices to probe. | "devices": [ {"device": "sf", "id": "4","sft": "...

For more information, refer to section "Running DOCA Application on Host" in NVIDIA DOCA Virtual Functions User Guide.
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
<td>Short Flag</td>
<td>Long Flag/JSON Key</td>
<td>Description</td>
<td>JSON Content</td>
</tr>
<tr>
<td></td>
<td>h</td>
<td>help</td>
<td>Prints a help synopsis</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>v</td>
<td>version</td>
<td>Prints program version information</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>log-level</td>
<td>Set the log level for the application:</td>
<td>&quot;log-level&quot;: 60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DISABLE=10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CRITICAL=20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ERROR=30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- WARNING=40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- INFO=50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DEBUG=60</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- TRACE=70 ( requires compilation with TRACE log level support )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>json</td>
<td>Parse all command flags from an input JSON file</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>stats-timer</td>
<td>Set interval to dump stats information.</td>
<td>&quot;stats-timer&quot;: 2</td>
</tr>
</tbody>
</table>

**General flags**

**Program flags**
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>nr-queues</td>
<td>Set queues number.</td>
<td></td>
<td>&quot;nr-queues&quot;: 4</td>
</tr>
<tr>
<td>r</td>
<td>rx-only</td>
<td>Set RX only. When set, the packets will not be sent to the TX queues.</td>
<td></td>
<td>&quot;rx-only&quot;: false</td>
</tr>
<tr>
<td>o</td>
<td>hw-offload</td>
<td>Set HW offload of the RXP engine to use.</td>
<td></td>
<td>&quot;hw-offload&quot;: false</td>
</tr>
<tr>
<td>hq</td>
<td>hairpinq</td>
<td>Set forwarding to hairpin queue.</td>
<td></td>
<td>&quot;hairpinq&quot;: false</td>
</tr>
<tr>
<td>a</td>
<td>age-thread</td>
<td>Start a dedicated thread that handles the aged flows.</td>
<td></td>
<td>&quot;age-thread&quot;: false</td>
</tr>
</tbody>
</table>

**Info**

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

**Troubleshooting**

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.
Application Code Flow

1. Parse application argument.
   1. Initialize arg parser resources and register DOCA general parameters.
      ```
      doca_argp_init();
      ```

2. Register application parameters.
   ```
   register_simple_fwd_params();
   ```

3. Parse the arguments.
   ```
   doca_argp_start();
   ```
   1. Parse DPDK flags and invoke handler for calling the `rte_eal_init()` function.
   2. Parse app parameters.

2. DPDK initialization.
   ```
   dpdk_init();
   ```
   Calls `rte_eal_init()` to initialize EAL resources with the provided EAL flags.

3. DPDK port initialization and start.
   ```
   dpdk_queues_and_ports_init();
   ```
   1. Initialize DPDK ports.
2. Create mbuf pool using `rte_pktmbuf_pool_create`.

3. Driver initialization – use `rte_eth_dev_configure` to configure the number of queues.

4. Rx/Tx queue initialization – use `rte_eth_rx_queue_setup` and `rte_eth_tx_queue_setup` to initialize the queues.

5. Rx hairpin queue initialization – use `rte_eth_rx_hairpin_queue_setup` to initialize the queues.

6. Start the port using `rte_eth_dev_start`.

4. Simple forward initialization.

```c
simple_fwd_init();
```

1. `simple_fwd_create_ins` – create flow tables using `simple_fwd_ft_create`.

2. `simple_fwd_init_ports_and_pipes` – initialize DOCA port using `simple_fwd_init_doca_port` and build default pipes for each port.

5. Main loop.

```c
simple_fwd_process_pkts();
```

1. Receive packets using `rte_eth_rx_burst` in a loop.


3. Transmit the packets on the other port by calling `rte_eth_tx_burst`. Or free the packet mbuf if `rx_only` is set to true.


```c
simple_fwd_process_offload();
```
1. Parse the packet's `rte_mbuf` using `simple_fwd_pkt_info`.

2. Handle the packet using `simple_fwd_handle_packet`. If the packet's key does not match the existed the flow entry, create a new flow entry and PIPE using `simple_fwd_handle_new_flow`. Otherwise, increase the total packet's counter.

7. Simple forward destroy.

```c
simple_fwd_destroy();
```

Simple forward close port and clean the flow resources.

8. DPDK ports and queues destruction.

```c
dpdk_queues_and_ports_fini();
```

9. DPDK finish.

```c
dpdk_fini();
```

Calls `rte_eal_destroy()` to destroy initialized EAL resources.

10. Arg parser destroy.

```c
doca_argp_destroy();
```

- Free DPDK resources by call `rte_eal_cleanup()` function.

**References**

- `/opt/mellanox/doca/applications/simple_fwd_vnf/
- `/opt/mellanox/doca/applications/simple_fwd_vnf/simple_fwd_params.json`