NVIDIA DOCA PCC

Application Guide

MLNX-15-060604 _v2.2.0 | October 2023

Table of Contents

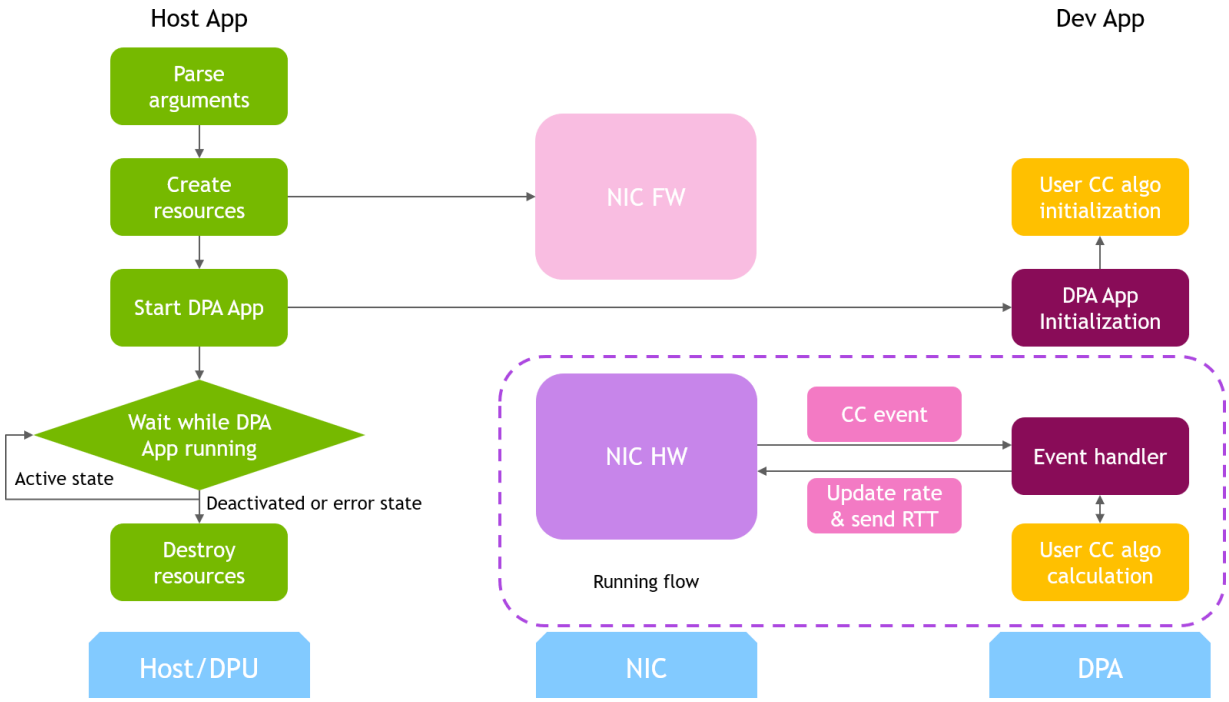
Chapter 1. Introdution

Programmable congestion control (PCC) allows users to design and implement their own congestion control (CC) algorithm, giving them good flexibility to work out an optimal solution to handle congestion in their clusters. On BlueField-3, PCC is provided as a component of DOCA.

The DOCA PCC application provides users the flexibility to manage allocation of DPA resources according to their needs. The application leverages the DOCA PCC library to generate an executable binary file.

Typical DOCA application includes App running on host/Arm and App running on DPA. Developers are advised to use the host/Arm application with minimal changes and focus on developing their algorithm and integrating it into the DPA application.

Chapter 2. System Design



DOCA PCC application consists of two parts:

- Host/Arm app is the control plane. It is responsible for allocating all resources and handover to the DPA app initially, then destroying everything when the DPA app finishes its operation. The host app must always be alive to stay in control while the device app is working.
- Device/DPA app is the data plane. It is mainly for CC event handler. When the first thread is activated, DPA App initialization is done in the DOCA PCC library by calling the algorithm initialization function implemented by the user in the app. Moreover, the user algorithm execution function is called when a CC event arrives. The user algorithm takes event data as input and performs a calculation using per-flow context and replies with updated rate value and a flag to sent RTT request.

The host/Arm application sends command to NIC firmware when allocating or destroying resources. CC events are generated by NIC hardware automatically when sending data or receiving ACK/NACK/CNP/RTT packets, then the device application handles these events

by calling the user algorithm. After the DPA application replies to hardware, handling of current event is done and the next event can arrive.

Chapter 3. Application Architecture

```
/opt/mellanox/doca/applications/pcc/src
├── host
│   ├── pcc.c
│   ├── pcc_core.c
│   └── pcc_core.h
└── device
    ├── algo
    │   ├── rtt_template.h
    │   ├── rtt_template_algo_params.h
    │   ├── rtt_template_ctxt.h
    │   └── rtt_template.c
    └── pcc_dev_main.c
```

The main content of the reference DOCA PCC application files are the following:

- `host/pcc.c` – entry point to entire application
- `host/pcc_core.c` – host functions to initialize and destroy the PCC application resources, parsers for PCC command line parameters
- `device/pcc_dev_main.c` – callbacks for user CC algorithm initialization, user CC algorithm calculation, algorithm parameter change notification
- `device/algo/*` – user CC algorithm reference template. Put user algorithm code here.

Chapter 4. DOCA Libraries

This application leverages the following DOCA libraries:

- DOCA PCC library

Chapter 5. Configuration Flow

1. Parse application argument.
   a). Initialize arg parser resources and register DOCA general parameters.
   ```
   doca_argp_init();
   ```
   b). Register PCC application parameters.
   ```
   register_pcc_params();
   ```
   c). Parse all registered parameters.
   ```
   doca_argp_start();
   ```
      i. Parse DOCA flags.
      ii. Parse DOCA PCC parameters.
2. PCC initialization.
   ```
   pcc_init();
   ```
   a). Open DOCA device that supports PCC.
   b). Create DOCA PCC context.
   c). Configure affinity of threads handling CC events.
3. Start DOCA PCC.
   ```
   doca_pcc_start();
   ```
   a). Create PCC process and other resources.
   b). Trigger initialization of PCC on device.
   c). Register the PCC in the NIC hardware so CC events can be generated and an event handler can be triggered.
4. Process state monitor loop.
   ```
   doca_pcc_get_process_state();
   doca_pcc_wait();
   ```
   a). Get the state of the process:

State	Description
`DOCA_PCC_PS_ACTIVE = 0`	The process handles CC events (only one process is active at a given time)
`DOCA_PCC_PS_STANDBY = 1`	The process is in standby mode (another process is already `ACTIVE`)
`DOCA_PCC_PS_DEACTIVATED = 2`	The process has been deactivated by NIC firmware and should be destroyed
`DOCA_PCC_PS_ERROR = 3`	The process is in error state and should be destroyed

b). Wait on process events from the device.

5. PCC destroy.

```
doca_pcc_destroy();
```

a). Destroy PCC resources. The process stops handling PCC events.

b). Close DOCA device.

6. Arg parser destroy.

```
doca_argp_destroy();
```

Chapter 6. Running the Application

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 applications.
   - NVIDIA DOCA Applications Overview for additional compilation instructions and development tips of DOCA applications.
2. The pre-built PCC binary is located under `/opt/mellanox/doca/applications/pcc/bin/doca_pcc`. To build all the applications together, run:

   ```
   cd /opt/mellanox/doca/applications/
   meson /tmp/build
   ninja -C /tmp/build
   ```
3. To build only the allreduce application:

   a). Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:
   - Set `enable_all_applications` to `false`
   - Set `enable_pcc` to `true`

   b). Run the commands in step 2.

   Note: `doca_pcc` is created under `/tmp/build/pcc/src/`.
4. Pre-run setup:

   a). Enable `USER_PROGRAMMABLE_CC` in `mlxconfig`:

   ```
   mlxconfig -y -d /dev/mst/mt41692_pciconf0 set USER_PROGRAMMABLE_CC=1
   ```

   b). Reset firmware or power cycle the host.
5. Running the application on the host or BlueField, CLI example:

   ```
   /opt/mellanox/doca/applications/pcc/bin/doca_pcc -d mlx5_0
   ```

   Application usage:

   ```
   Usage: doca_pcc [DOCA Flags] [Program 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>
   ```

```
Program Flags:
  -d, --device <IB device names>       IB device name that supports PCC
 (mandatory).
  -w, --wait-time <PCC wait time>      The duration of the DOCA PCC wait
 (optional), can provide negative values which means infinity. If not provided
 then -1 will be chosen.
  -p, --pcc-threads <pcc-threads-list> A list of the PCC threads numbers to be
 chosen for the DOCA PCC context to run on (optional). Must be provided as a
 string, such that the number are separated by a space.
```

For additional information on available flags, use `-h`:

```
/opt/mellanox/doca/applications/pcc/bin/doca_pcc -h
```

6. To run `doca_pcc` using a JSON file:

   ```
   doca_pcc --json [json_file]
   ```

   For example:

   ```
   cd /opt/mellanox/doca/applications/pcc/bin
   ./doca_pcc --json ./pcc_params.json
   ```

Chapter 7. Arg Parser DOCA Flags

Refer to NVIDIA DOCA Arg Parser Programming Guide for more information.

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
General flags	`l`	`log-level`	Set the log level for the application: ▶ CRITICAL=20 ▶ ERROR=30 ▶ WARNING=40 ▶ INFO=50 ▶ DEBUG=60	`"log-level": 60`
	`v`	`version`	Print program version information	N/A
	`h`	`help`	Print a help synopsis	N/A
Program flags	`d`	`device`	IB device name that supports PCC	`"device": ""`
	`w`	`wait-time`	(Optional) In seconds, the duration of the DOCA PCC wait. Negative values mean infinity.	`"wait-time": -1`
	`p`	`pcc-threads`	(Optional) A list of the PCC EU indexes to be chosen for the DOCA PCC event handler threads to run on. Must be provided as a string, such that	`"pcc-threads": "176 177 178 179 180 181 182 183 184 185 186 187 192 193 194 195 196 197 198 199 200 201 202 203 208 209 210`

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
			the numbers are separated by a space. The placement of the PCC threads per core can be controlled using the EU indexes. Utilizing a large number of EUs, while limiting the number of threads per core, gives the best event handling rate and lowest event latency. The last EU is used for communication with the NIC while all others are for data path CC event handling.	211 212 213 214 215 216 217 218 219 224 225 226 227 228 229 230 231 232 233 234 235 240"

Chapter 8. Port Programmable Congestion Control Register

The Port Programmable Congestion Control (PPCC) register allows the user to configure and read PCC algorithm parameters.

It supports the following functionalities:

- Enabling different algorithms on different ports
- Querying information of both algorithms and tunable parameters/counters
- Changing algorithm parameters without compiling and reburning user image
- Querying or clearing programmable counters

8.1. Usage

The PPCC register can be accessed using a string similar to the following:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=0" --reg_name PPCC
 --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --set "cmd_type=1" --reg_name PPCC --
indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

Where you must:

- Set the `cmd_type` and the indexes
- Give values for `algo_slot`, `algo_param_index`
- Keep `local_port=1`, `pnat=0`, `lp_msb=0`
- Keep `doca_pcc` application running

cmd_type	Description	Method	Index	Input (in --set)	Output
0x0	Get algorithm info	Get	`algo_slot`	N/A	▶ Value – 32-bit `algo_num` or 0 if no

cmd_type	Description	Method	Index	Input (in --set)	Output
					algo is available at this index ▶ Text – algorithm description ▶ `sl_bitmask_support` – Indicates whether the device supports `sl_bitmask` logic

0x1	Enable algorithm	Set		`sl_bitmask`
				`trace_en`
				`counter_en`
					N/A

0x2	Disable algorithm	Get		N/A	N/A
0x3	Get algorithm enabling status	Get		N/A	▶ Value – ▶ 0: Disabled ▶ 1: Enabled ▶ `sl_bitmask` – This field allows to apply to specific SLs based on the bitmask. ▶ `sl_bitmask_support` – Indicates whether the device supports `sl_bitmask` logic
0x4	Get number of parameters	Get		N/A	▶ Value – num of params of algo

cmd_type	Description	Method	Index	Input (in --set)	Output
0x5	Get parameter information	Get	`algo_slot` `algo_param_index`	N/A	▶ `param_value1` – default value of param ▶ `param_value2` – min value of param ▶ `param_value3` – max value of param ▶ prm – ▶ 0: read-only ▶ 1: read-write ▶ 2: read-only but may be cleared using the "get and clear" command
0x6	Get parameter value	Get		N/A	▶ Value – param value
0x7	Get and clear parameter	Get		N/A	▶ Value – param value
0x8	Set parameter value	Set		Parameter value	N/A
0xA	Bulk get parameters	Get	`algo_slot`	N/A	▶ `text_length` – param num x 4 bytes

cmd_type	Description	Method	Index	Input (in --set)	Output
					▶ `text[0]…text[n]` – param values
0xB	Bulk set parameters	Set		`text_length` - param num x 4 `text[0]…text[n]` - param values	N/A
0xC	Bulk get counters	Get		N/A	▶ `text_length` – counter num x 4 bytes ▶ `text[0]…text[n]` – counter values
0xD	Bulk get and clear counters	Get		N/A	▶ `text_length` – counter num x 4 bytes ▶ `text[0]…text[n]` – counter values
0xE	Get number of counters	Get		N/A	▶ Value – num of counters of algo
0xF	Get counter information	Get	`algo_slot` `algo_param_index`	N/A	▶ `param_value3` – max value of parameter ▶ prm ▶ 0- read-only ▶ 1- read-write ▶ 2- read-

cmd_type	Description	Method	Index	Input (in --set)	Output
					only but may be cleared via "get & clear" command
0x10	Get algorithm info array	Get	N/A	N/A	▶ `text_length` – algo slot initialized x 4 bytes ▶ `text[0]…text[n]` – 32-bit `algo_num` or 0 if no algorithm is available at this slot index

8.2. Internal Default Algorithm

The internal default algorithm is used when enhanced connection establishment (ECE) negotiation fails. It is mainly used for backward compatibility and can be disabled using "force mode". Otherwise, users may change `doca_pcc_dev_user_algo()` in the device app to run a specific algorithm without considering the algorithm negotiation.

The force mode command is per port:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=2" --reg_name PPCC
 --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=15,algo_param_index=0"
sudo mlxreg -d /dev/mst/mt41692_pciconf0.1 -y --get --op "cmd_type=2" --reg_name
 PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=15,algo_param_index=0"
```

8.3. Counters

Counters are shared on the port and are only enabled on one `algo_slot` per port. The following command enables the counters while enabling the algorithm according to the `algo_slot`:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --set "cmd_type=1,counter_en=1" --
reg_name PPCC --
indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

After counters are enabled on the `algo_slot`, they can be queried using `cmd_type` 0xC or 0xD.

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=12" --reg_name PPCC
 --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=13" --reg_name PPCC
 --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

Chapter 9. References

- `/opt/mellanox/doca/applications/pcc/src`

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