NVIDIA DOCA Programmable Congestion Control Programming Guide

Programming Guide

MLNX-15-060603 _v2.2.1 | October 2023

Table of Contents

Chapter 1. Introduction

The DOCA PCC library provides a high-level programming interface that allows users to implement their own customized congestion control (CC) algorithm.

The DOCA PCC library provides an API to:

- Get the CC event/packet and access its fields
- Set a rate limit for a flow
- Maintain a context for each flow
- Initiate and configure CC algorithms

This library uses the NVIDIA® BlueField®-3 DPU hardware acceleration for CC management, while providing an API that simplifies hardware complexity, allowing users to focus on the functionality of the CC algorithm.

Chapter 2. Prerequisites

DOCA PCC-based applications can run either on the host machine or on the NVIDIA® BlueField®-3 (or later) target DPU.

To enable PCC:

1. Run the following on the host/VM:
   ```
   mlxconfig -d <mlx_device> -y s USER_PROGRAMMABLE_CC=1
   ```
2. Power cycle the host.

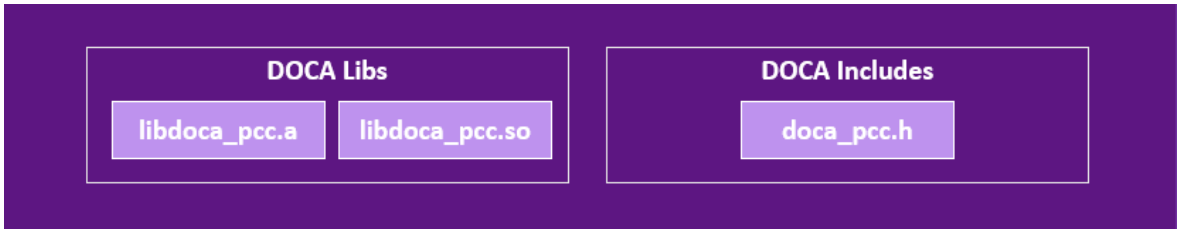
The DPACC tool is used to compile and link user algorithm and device code with the DOCA PCC device library to get applications that can be loaded from the host program.

DPACC is bundled as part of the DOCA SDK installation package. For more information on DPACC, refer to NVIDIA DOCA DPACC Compiler User Guide.

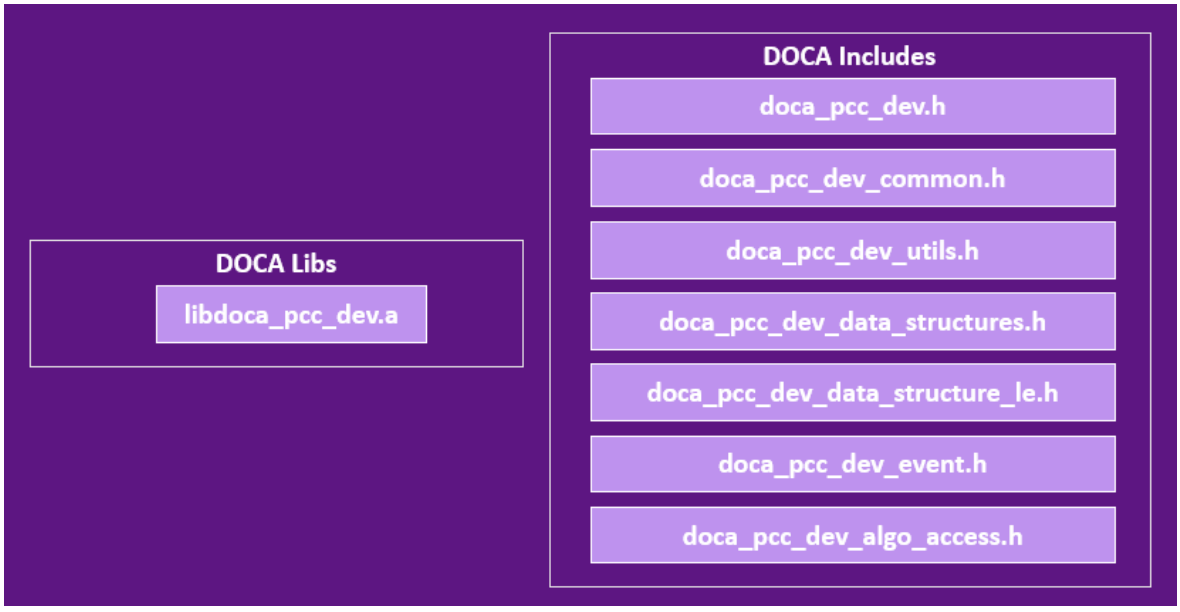
Chapter 3. Architecture

DOCA PCC is composed of two libraries which are part of the DOCA SDK installation package:

- Host/DPU library and header files



- Device library and header files



Currently, the device library and the user algorithm are implemented and managed over the BlueField's data-path accelerator (DPA) subsystem.

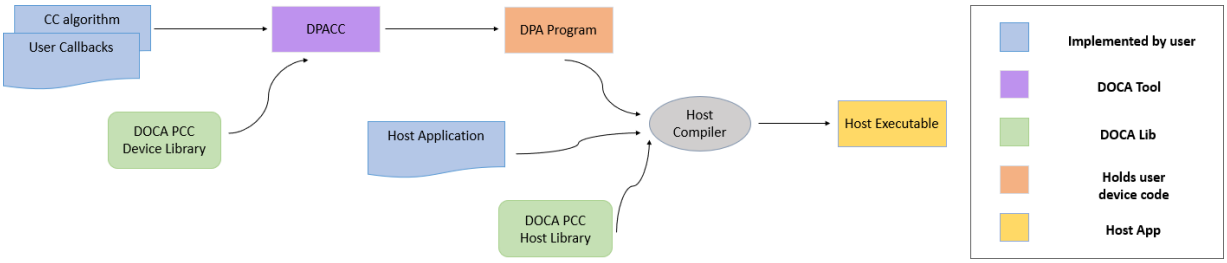
For more info on DPA, refer to the NVIDIA DOCA DPA Subsystem Programming Guide.

3.1. Development Flow

DOCA enables developers to program the congestion control algorithm into the system using the DOCA PCC library.

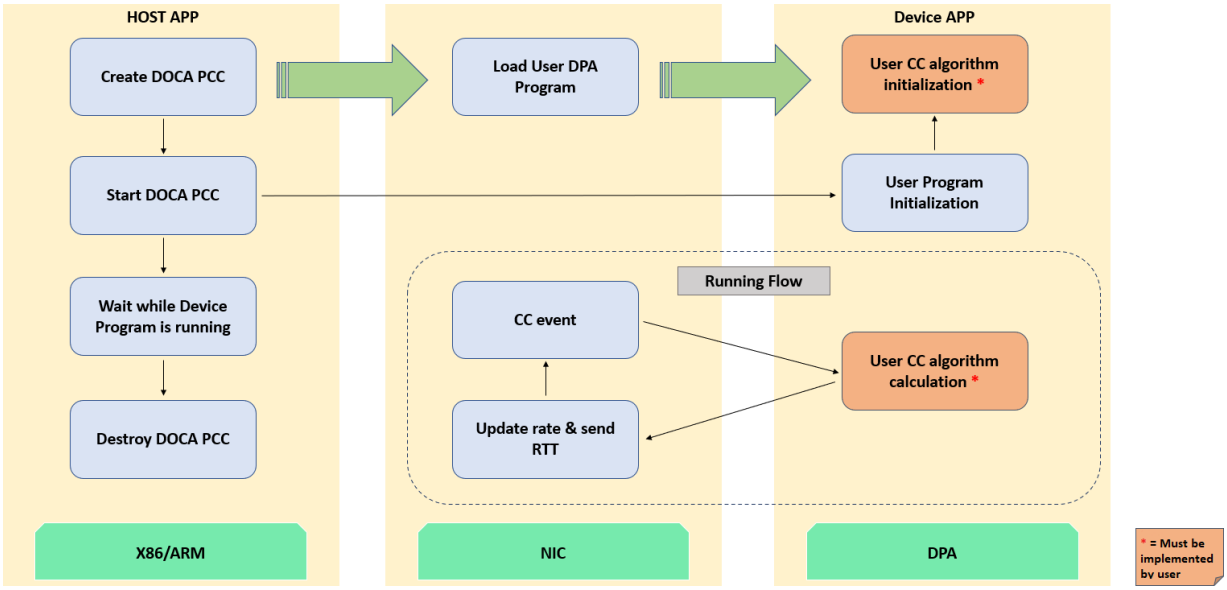
The following are the required steps to start programming:

1. (Optional) Implement CC algorithms using the API provided by the device header files to be run by the library.
2. Implement the following calls defined by the library: `doca_pcc_dev_user_init()`, `doca_pcc_dev_user_set_algo_params()`, `doca_pcc_dev_user_algo()`.
3. Use DPACC to build a DPA program (i.e., a host library which contains an embedded device executable). Input for DPACC are the files containing the implementation of the previous steps.
4. Build host executable using a host compiler. Inputs for the host compiler are the DPA program generated in the previous step and the user application source files.
5. In the host executable, create and start a DOCA PCC context which is set with the DPA program containing the device code.



For a more descriptive example, refer to the NVIDIA DOCA PCC Application Guide.

3.2. System Design


HOST APP
Create DOCA PCC
Start DOCA PCC
Wait while Device Program is running
Destroy DOCA PCC
X86/ARM
Load User DPA Program
NIC
Device APP
User CC algorithm initialization *
User Program Initialization
Running Flow
CC event
User CC algorithm calculation *
Update rate & send RTT
DPA
* = Must be implemented by user

Chapter 4. Dependencies

The library requires firmware version 32.38.1000 and higher.

Chapter 5. API

For the library API reference, refer to PCC API documentation in the NVIDIA DOCA Libraries API Reference Manual.

The following sections provide additional details about the library API.

5.1. Host API

The host library API consists of calls to set the PCC context attributes and observe availability of the process.

5.1.1. Selecting and Opening DOCA Device

To perform PCC operations, a device must be selected. To select a device, users may iterate over all DOCA devices using `doca_devinfo_list_create()` and check whether the device supports PCC using `doca_devinfo_get_is_pcc_supported()`.

5.1.2. Setting Up and Starting DOCA PCC Context

After selecting a DOCA device, a PCC context can be created using `doca_pcc_create()`.

Afterwards, the following attributes must be set for the PCC context:

- Context app – the name of the DPA program compiled using DPACC, consisting of the device algorithm and code. This is set using the call `doca_pcc_set_app()`.
- Context threads – the affinity of DPA threads to be used to handle CC events. This is set using the call `doca_pcc_set_thread_affinity()`. The number of threads to be used must be constrained between the minimum and maximum number of threads allowed to run the PCC process (see `doca_pcc_get_min_num_threads()` and `doca_pcc_get_max_num_threads()`). The availability and usage of the threads for PCC is dependent on the complexity of the CC algorithm, link rate, and other potential DPA users.

> Note: Users can manage DPA threads in the system using EU pre-configuration with the `dpaeumgmt` tool. For more information, refer to the NVIDIA DOCA DPA EU Management Tool.

After setting up the context attributes, the context can be started using `doca_pcc_start()`. Starting the context initiates the CC algorithm supplied by the user.

5.1.3. High Availability

The DOCA PCC library provides high availability, allowing fast recovery should the running PCC process malfunction. High availability can be achieved by running multiple PCC processes in parallel.

When calling `doca_pcc_start()`, the library registers the process with the BlueField firmware such that the first PCC process to be registered becomes the ACTIVE PCC process (i.e., actually runs on DPA and handles CC events).

The other processes operate in STANDBY mode. If the ACTIVE process stops processing events or hits an error, the firmware replaces it with one of the standby processes, making it ACTIVE.

The defunct process should call `doca_pcc_destroy()` to free its resources.

The state of the process may be observed periodically using `doca_pcc_get_process_state()`. A change in the state of the process returns the call `doca_pcc_wait()`.

The following values describe the state of the PCC process at any point:

```
typedef enum {
 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 was deactivated by NIC FW and should be destroyed */
    DOCA_PCC_PS_ERROR = 3,
    /**< The process is in error state and should be destroyed */
} doca_pcc_process_state_t;
```

5.2. Device API

The device library API consists of calls to setup the CC algorithm to handle CC events arriving on hardware.

5.2.1. Algorithm Access

The device library API provides a set of functions to initiate and and identify the different CC algorithms.

The DOCA PCC library is designed to support more than one PCC algorithm. The library comes with a default algorithm which can be used fully or partially by the user using `doca_pcc_dev_default_internal_algo()`, alongside other CC algorithms supplied by the user. This can be useful for fast comparative runs between the different algorithms. Each algorithm can run on a different device port using `doca_pcc_dev_init_algo_slot()`.

The algorithm can supply its own identifier, initiate its parameter (using `doca_pcc_dev_algo_init_param()`), counter (using `doca_pcc_dev_algo_init_counter()`), and metadata base (using `doca_pcc_dev_algo_init_metadata()`).

5.2.2. Events

The device library API provides a set of optimized CC event access functions. These functions serve as helpers to build the CC algorithm and to provide runtime data to analyze and inspect CC events arriving on hardware.

5.2.3. Utilities

The device library API provides a set of optimized utility macros that are set to support programming the CC algorithm. Such utilities are composed of fixed point operations, memory space fences, and more.

5.2.4. User Callbacks

The device library API consists of specific user callbacks used by the library to initiate and run the CC algorithm. These callbacks must be implemented by the user and, to be part of the DPA program, compiled by DPACC to provide to the DOCA PCC context.

The set of callbacks to be implemented is as follows:

- `doca_pcc_dev_user_init()` – this is called on PCC process load and should initialize the data of all user algorithms
- `doca_pcc_dev_user_algo()` – entry point to the user algorithm handling code
- `doca_pcc_dev_user_set_algo_params()` – called when the parameter change is set externally

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