Table of contents

Introduction

Prerequisites

Changes From Previous Releases

Changes in 2.8.0

Added

Changed

Dependencies

Architecture

Host Library

Device Libraries

Development Flow

System Design

API

Host API

Selecting and Opening DOCA Device

Setting Up and Starting DOCA PCC Context

Configuring Probe Packets

Configuring Dedicated Fields for Different Probe Types

Configuring Remote NP Handler

Debbuggability

Host - Device Mailbox
High Availability

Device API

Counter Sampling

Algorithm Access

Events

Utilities

User Callbacks
This guide provides an overview and configuration instructions for DOCA Programmable Congestion Control (PCC) API.

**Introduction**

The DOCA PCC library provides a high-level programming interface that allows users to implement their own customized congestion control (CC) algorithm, facilitating efficient management of network congestion in their applications.

The DOCA PCC library provides an API to:

- Configure probe packets to send and receive
- 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
- Obtain request packets arriving from the network and setup response packets in return

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

**Prerequisites**

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

ℹ️ **Info**

Currently, DOCA PCC is only supported for ETHERNET link type.
To enable DOCA PCC RP:

1. Run the following on the host/VM:

   mlxconfig -d <mlx_device> -y s USER_PROGRAMMABLE_CC=1

2. Perform graceful shutdown then power cycle the host.

To enable DOCA PCC NP:

1. Run the following on the host/VM

   mlxconfig -d <mlx_device> -y s PCC_INT_EN=0

2. Perform graceful shutdown then power cycle the host.

**Note**

Configuring `PCC_INT_EN` to 1 blocks the creation of DOCA PCC NP context and enables legacy NP solution. In addition, it only supports DOCA PCC RP context to set Congestion Control Message After Drop (CCMAD) probe packet format.

If IFA2.0 support is needed, user needs to enable DOCA PCC RP and DOCA PCC NP on all nodes of the cluster.

**Note**

If running from the host in NIC mode, users must have PRIVILIGED permission to configure the above parameters. To check privileging level, run:
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](#).

**Changes From Previous Releases**

**Changes in 2.8.0**

**Added**

- `doca_pcc_dump_debug(const struct doca_pcc *pcc)`

- `doca_pcc_enable_debug(const struct doca_pcc *pcc, bool enable)`

**Changed**

- `DOCA_PCC_DEV_MAX_NUM_PARAMS_PER_ALGO` (0x1E) to (0x26)

- `DOCA_PCC_DEV_MAX_NUM_COUNTERS_PER_ALGO` (0xF) to (0x3F)

- `struct mlx privhost -d <mlx_device> q`

  ```c
  struct mlx_cc_event_general_attr_t {
      /* Little Endian */
      uint32_t ev_type:8;  /* event type */
      uint32_t ev_subtype:8;  /* event subtype */
      - uint32_t port_num:8;  /* port id */
      + uint32_t port_num:4;  /* port id */
      + uint32_t reserved:4;
      uint32_t flags:8;  /* event flags */
  }
  ```
• struct mlnx_cc_event_t – added 12B reserved

```c
struct mlnx_cc_event_t { /* Little Endian */
+ uint32_t reserved[3]; /* reserved */
struct mlnx_cc_event_general_attr_t ev_attr; /* event general attributes */
uint32_t flow_tag; /* unique flow id */
uint32_t sn; /* serial number */
uint32_t timestamp; /* event timestamp */
union mlnx_cc_event_spec_attr_t ev_spec_attr; /* attributes which are different for different events */
};
```

**Dependencies**

The library requires firmware version 32.38.1000 and higher.

**Architecture**

DOCA PCC comprises three main components which are part of the DOCA SDK installation package:

**Host Library**

The host library offers a unified interface for managing the DOCA PCC context configuration.

As part of the control path, the host library integrates passively within the application, orchestrating congestion control activities without directly handling data transmission.

Host/device library and header files:
Device Libraries

The DOCA PCC context assumes one of two roles:

- **Reaction point (RP):** Monitors network conditions actively, dynamically adjusting data transmission rates to alleviate congestion promptly. RP context is global per NIC.

  Device library and header files:

  - **Notification point (NP):** Passively receives congestion notifications from external sources, processing them intelligently to facilitate informed decisions within the application. NP context is global per e-switch owner.

  Device library and header files:
Both RP and NP device libraries share common headers:

- doca_pcc_dev_common.h
- doca_pcc_dev_services.h
- doca_pcc_dev_utils.h

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 [DPA Subsystem](#).

**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. Implement CC algorithms and probe packet handling using the API provided by the device header files.

2. Implement the user callbacks defined by the library for DataPath:
   - For RP: doca_pcc_dev_user_init(), doca_pcc_dev_user_set_algo_params(), doca_pcc_dev_user_algo().
   - For NP: doca_pcc_dev_np_user_packet_handler()

3. Use DPACC to build a DPA application (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 application generated in the previous step and the user application host source files.

5. In the host executable, create and start a DOCA PCC context which is set with the DPA application containing the device code.

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

**System Design**

DOCA PCC flow for implementing an RP program:

DOCA PCC flow for implementing an NP program:
API

For the library API reference, refer to PCC API documentation in the *NVIDIA DOCA Library APIs*.

The following sections provide additional details about the library API.

**Host API**

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

**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 the desired PCC role either via `doca_devinfo_get_is_pcc_supported()` for RP, or `doca_pcc_np_cap_is_supported()` for NP.

**Setting Up and Starting DOCA PCC Context**

After selecting a DOCA device, a PCC context can be created.
As described in the Architecture section, The DOCA PCC library provides APIs to leverage Reaction Points (RP) and Notification Points (NP) to implement programmable congestion control strategies.

Call `doca_pcc_create()` to create a DOCA PCC RP context, and `doca_pcc_np_create()` to create a DOCA PCC NP context.

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

- **Context app** – the name of the DPA application 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](image)

Users can manage DPA threads in the system using EU pre-configuration with the `dpaeumgmt` tool. For more information, refer to [NVIDIA DOCA DPA Execution Unit 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.

**Configuring Probe Packets**

The DOCA PCC library provides APIs to configure the probe packet settings to tailor congestion control behaviors according to specific network conditions.

The probe packet serves to probe the network for congestion and gather essential feedback for congestion control algorithms.
The DOCA PCC Library supports the following probe packet types:

- **CCMAD** – Provides information about the network's round-trip time so the algorithm can detect and adapt to congestion proactively
- **IFA1** – In-band Flow Analyzer 1 packets provide in-band congestion feedback for proactive congestion control
- **IFA2** – In-band Flow Analyzer 2 packets offer an alternative method for in-band congestion feedback, optimized for specific network environments

### Configuring Dedicated Fields for Different Probe Types

The DOCA PCC library provides APIs to configure specific fields in different supported probe packet types.

- **IFA1** – support to configure probe marker
- **IFA2** – support to configure gns and hop limit

### Configuring Remote NP Handler

To enable Reaction Point contexts to interact with remote Notification Point contexts, the DOCA PCC library provides an API to set the expected remote handler type.

When the DOCA PCC RP process expects CCMAD probe packet responses from a DOCA PCC NP process, it should set it as so using the API `doca_pcc_rp_set_ccmad_remote_sw_handler()`. If not set, the DOCA PCC RP process expects that no remote DOCA PCC NP process is activated, and that responses are handled by the remote node's hardware. Note that if using other probe types than CCMAD, probe packet responses are always expected to be generated from a remote DOCA Notification Point process.

### Debuggability

The DOCA PCC library provides a set of debugging APIs to allow the user to diagnose and troubleshoot any issues on the device, as well as accessing real-time information from
the running application:

- **doca_pcc_set_dev_coredump_file()** – API to set a filename to write crash data and core dump into should a fatal error occur on the device side of the application. The data written into the file would include a memory snapshot at the time of the crash, which would contain information instrumental in pinpointing the cause of a crash (e.g., the program's state, variable values, and the call stack).

- **doca_pcc_set_trace_message()** – API to enable tracing on the device side of the application by setting trace message formats that can be printed from the device. The tracer provided by the library is of high-frequency and is designed to not have significant impact on the application's performance. This API can help the user to monitor and gain insight into the behavior of the running device algorithm, identify performance bottlenecks, and diagnose issues, without incurring any notable performance degradation.

- **doca_pcc_set_print_buffer_size()** – API to set the buffer size to be printed by the print API provided by the device library.

### Host - Device Mailbox

The DOCA PCC library provides a set of APIs for sending and receiving messages through a mailbox. This service allows communication between the host and device:

- **doca_pcc_set_mailbox()** – API to set the mailbox attributes for the process.

- **doca_pcc_mailbox_get_request_buffer() and doca_pcc_mailbox_get_response_buffer()** – API to get the buffers with which the communication will be handled. User can set the request he wants to send to the device, and get a response back.

- **doca_pcc_mailbox_send()** – API to send the mailbox request to the device. This is a blocking call which invokes a callback on the device **doca_pcc_dev_user_mailbox_handle()** which user can handle.

### 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:

```c
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;
```

**Device API**

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

**Counter Sampling**

The device libraries APIs provide an API to sample the NIC bytes counters. These counters help monitor the amount of data transmitted and received through the NIC.
The user can prepare the list of counters to read using `doca_pcc_dev_nic_counters_config()` and sample the new counters values with the call `doca_pcc_dev_nic_counters_sample()`.

**Algorithm Access**

The Reaction Point (RP) device library API provides a set of functions to initiate 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()`).

**Events**

The RP 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.

**Utilities**

The device library APIs provide 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.

**User Callbacks**

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

The set of callbacks to be implemented for RP:

- `doca_pcc_dev_user_init()` – 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

The set of callbacks to be implemented for NP:

- `doca_pcc_dev_np_user_packet_handler()` – called on probe packets arrival