



DOCA Rivermax

Table of contents

Introduction

Prerequisites

Environment

Architecture

Objects

Configuration Phase

Configurations

Mandatory Configurations

Optional Configurations

Device Support

Buffer Support

Execution Phase

Events

Rx Data

Runtime Configurations

State Machine

Idle

Starting

Running

Stopping

DOCA Rivermax Samples

Running the Samples

Samples

List Devices

Set CPU Affinity

Set Clock

Create Stream

Create Stream – Header-data Split Mode

This guide provides instructions on building and developing applications that require media/data streaming.

Introduction

DOCA Rivermax (RMAX) is a DOCA API for NVIDIA® Rivermax®, an optimized networking SDK for media and data streaming applications. Rivermax leverages NVIDIA® BlueField® DPU hardware streaming acceleration technology which enables direct data transfers to and from the GPU, delivering best-in-class throughput and latency with minimal CPU utilization for streaming workloads.

This document is intended for software developers wishing to accelerate their networking operations.

Prerequisites

This library follows the architecture of DOCA Core Context. it is recommended read the following content before proceeding:

- [DOCA Core Execution Model](#)
- [DOCA Core Device](#)
- [DOCA Core Memory Subsystem](#)

Environment

Info

DOCA Rivermax-based applications can run on the target DPU only.

Info

DOCA Rivermax-based application must be run with root privileges.

- The Rivermax library must compile and run and Rivermax license to run applications. Refer to [NVIDIA Rivermax SDK page](#) to obtain that license.
- An IP address to the device being used must be set up .
- It is recommended to have at least 800 huge pages enabled to achieve maximum performance:

```
dpu> echo 1000000000 > /proc/sys/kernel/shmmax
dpu> echo 800 > /proc/sys/vm/nr_hugepages
```

Architecture

- DOCA Rivermax Input Stream is a [DOCA Context](#) as defined by DOCA Core
- DOCA Rivermax leverages DOCA Core architecture to expose asynchronous events that are offloaded to hardware
- DOCA Rivermax can be used to define input streams that allow packet acquisition on an IP port. Furthermore, the input stream can be split to TCP/UDP 5-tuples to allow separate handling of flows.

Objects

- `doca_rmax_flow` – is a flow object that represents an IP/port tuple
- `doca_rmax_in_stream` – is a `doca_ctx` that represents the input stream and can be thought of as a receive queue which scatters the received data into memory. Each stream can receive one or more flows.

Configuration Phase

To start using the library users must first go through a configuration phase as described in [DOCA Core Context Configuration Phase](#).

This section describes how to configure and start the context to allow execution of tasks and retrieval of events.

Configurations

The context can be configured to match the application use case.

To find if a configuration is supported or its min/max value, refer to section "[Device Support](#)".

Mandatory Configurations

These configurations must be set by the application before attempting to start the context:

- An event type must be configured. See configuration of [Events](#).
- CPU affinity and then Rivermax library global initialization in this order. The following APIs can be used to achieve this `doca_rmax_set_cpu_affinity_mask()` and `doca_rmax_init()`
- The memory block that holds packet memory
- The number of stream elements
- Minimal packet segment size(s)
- Maximal packet segment size(s)

Optional Configurations

If the following configurations are not set, then a default value is used:

- The input stream type – defaults to generic
- The input stream packet's data scatter type – defaults to raw
- The input stream timestamp format – defaults to raw counter

Device Support

DOCA Rivermax Input Stream requires a device to operate. For picking a device see [DOCA Core Device Discovery](#).

The device must be from within the DPU: Either a PF or SF.

It is recommended to choose your device using the following method:

- `doca_devinfo_get_ipv4_addr()`

Some devices can allow different capabilities as follows:

- PTP clock support.

Buffer Support

Memory block support buffers with the following features:

Buffer Type	Memory Block
Local mmap buffer	Yes
Mmap from PCIe export buffer	Yes
Mmap from RDMA export buffer	No
Linked list buffer	Yes (header split mode)

Execution Phase

This section describes execution on CPU using [DOCA Core Progress Engine](#).

Events

DOCA Rivermax exposes asynchronous events to notify about changes that happen unexpectedly according to the DOCA Core architecture.

Common events are described in [DOCA Core Event](#).

Rx Data

The Rx Data event is used by the stream to notify application that data has been received from the network.

Event Configuration

Description	API to Set the Configuration	API to Query Support
Register to the event	<code>doca_rmax_in_stream_event_rx_data_register</code>	–

Event Trigger Condition

The event is triggered anytime packet(s) arrive.

Event Output

Common output as described in [DOCA Core Event](#).

In case of success, the following is provided:

- Number of packets received
- Time of arrival of the first packet
- Time of arrival of the last packet
- Sequence number of the first packet
- Array of memory blocks as configured by input stream

In case of error, the following is provided:

- An error code
- A human readable message

Note

The parameters are valid only inside the event callback.

Event Handling

Once an event is triggered, the application may decide to process the received data.

Runtime Configurations

These configurations can be made after the context has been started:

- The minimal number of packets that the input stream must return in Rx event.
- The maximal number of packets that the input stream must return in Rx event.
- The receive timeout. The number of μ secs that library would do busy wait (polling) for reception of at least `min_packets` number of packets.

State Machine

The DOCA RMAX library follows the Context state machine as described in [DOCA Core Context State Machine](#)

The following section describes how to move to the state and what is allowed in each state.

Idle

In this state, it is expected that application either:

- Destroys the context
- Starts the context

Allowed operations:

- Configuring the context according to [Configurations](#)
- Starting the context

It is possible to reach this state as follows:

Previous State	Transition Action
None	Create the context
Running	Call stop

Starting

This state is not expected to be reached.

Running

In this state, it is expected that application:

- Calls progress to receive events

Allowed operations:

- Calling stop
- Changing runtime configurations as described in [Runtime Configurations](#)

It is possible to reach this state as follows:

Previous State	Transition Action
Idle	Call start after configuration

Stopping

This state is not expected to be reached.

DOCA Rivermax Samples

The samples illustrate how to use the DOCA Rivermax API to:

- List available devices, including their IP and supported capabilities
- Set CPU affinity for the internal Rivermax thread to achieve better performance
- Set the PTP clock device to be used internally in DOCA Rivermax
- Create a stream, create a flow and attach it to the created stream, and finally to start receiving data buffers (based on the attached flow)
- Create a stream in header-data split mode when packet headers and payload are split to different RX buffers

Info

All the DOCA samples described in this section are governed under the BSD-3 software license agreement.

Running the Samples

1. Refer to the following documents:

- [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software
- [DOCA Troubleshooting](#) 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_rmax/<sample_name>
```

```
meson /tmp/build
ninja -C /tmp/build
```

(i) Info

The binary `doca_<sample_name>` is created under `/tmp/build/`.

3. Sample (e.g., `doca_rmax_create_stream`) usage:

```
Usage: doca_rmax_create_stream [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>
```

```
-j, --json <path>      Parse all command
flags from an input json file
```

Program Flags:

```
-p, --pci_addr <PCI-ADDRESS>  PCI device address
```

(i) Note

When running DOCA Rivermax samples, the IPv4 address 192.168.105.2 must be configured to an available uplink prior to

running it for the samples to run as expected :

```
$ifconfig p0 192.168.105.2
```

4. For additional information per sample, use the `-h` option:

```
/tmp/build/<sample_name> -h
```

Samples

List Devices

This sample illustrates how to list all available devices, dump their IPv4 addresses, and tell whether or not the PTP clock is supported.

The sample logic includes:

1. Initializing DOCA Rivermax library.
2. Iterating over the available devices.
3. Dumping their IPv4 addresses
4. Dumping whether a PTP clock is supported for each device.
5. Releasing DOCA Rivermax library.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_list_devices/rmax_list_`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_list_devices/rmax_list_`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_list_devices/meson.buil`

- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`;
`/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

Set CPU Affinity

This sample illustrates how to set the CPU affinity mask for Rivermax internal thread to achieve better performance. This parameter must be set before library initialization otherwise it will not be applied.

The sample logic includes:

1. Setting CPU affinity using the DOCA Rivermax API.
2. Initializing DOCA Rivermax library.
3. Releasing DOCA Rivermax library.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_affinity/rmax_set_a`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_affinity/rmax_set_a`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_affinity/meson.build`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`;
`/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

Set Clock

This sample illustrates how to set the PTP clock device to be used internally in DOCA Rivermax.

The sample logic includes:

1. Opening a DOCA device with a given PCIe address.
2. Initializing the DOCA Rivermax library.

3. Setting the device to use for obtaining PTP time.
4. Releasing the DOCA Rivermax library.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_clock/rmax_set_cloc`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_clock/rmax_set_cloc`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_clock/meson.build`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`;
`/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

Create Stream

This sample illustrates how to create a stream, create a flow and attach it to the created stream, and finally to start receiving data buffers (based on the attached flow).

The sample logic includes:

1. Opening a DOCA device with a given PCIe address.
2. Initializing the DOCA Rivermax library.
3. Creating an input stream.
4. Creating the context from the created stream.
5. Initializing DOCA Core related objects.
6. Setting the attributes of the created stream.
7. Creating a flow and attaching it to the created stream.
8. Starting to receive data buffers.
9. Clean up—detaches flow and destroys it, destroys created stream and DOCA Core related objects.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream/rmax_crea`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream/rmax_crea`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream/meson.bui`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`;
`/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

Create Stream – Header-data Split Mode

This sample illustrates how to create a stream in header-data split mode when packet headers and payload are split to different RX buffers.

The sample logic includes:

1. Opening a DOCA device with a given PCIe address.
2. Initialize the DOCA Rivermax library.
3. Creating an input stream.
4. Creating a context from the created stream.
5. Initializing DOCA Core related objects.
6. Setting attributes of the created stream. Chaining buffers and setting header size to non-zero is essential to create a stream with header-data split mode.
7. Creating a flow and attaching it to the created stream.
8. Starting to receive data to split buffers.
9. Clean up—detaches flow and destroys it, destroys created stream and DOCA Core related objects.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream_hds/rmax_`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream_hds/rmax_`

- /opt/mellanox/doca/samples/doca_rmax/rmax_create_stream_hds/meson
- /opt/mellanox/doca/samples/doca_rmax/rmax_common.h;
/opt/mellanox/doca/samples/doca_rmax/rmax_common.c

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 (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes 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 and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

