NVIDIA DOCA Comm Channel

Programming Guide

MLNX-15-060545 _v2.2.0 | October 2023

Table of Contents

Chapter 1. Introduction

The DOCA Comm Channel provides a secure, network-independent communication channel between the host and the DPU.

The communication channel allows the host to control services on the DPU or to activate certain offloads.

The DOCA Comm Channel is reliable, message-based, and supports connecting multiple clients to a single service. The API allows communication between a client using any PF/VF/SF on the host to a service on the DPU.

Chapter 2. Prerequisites

The CC service can only run on the DPU while the client can only run on a host connected to the DPU.

Refer to NVIDIA DOCA Release Notes for the supported versions of firmware, OS, and MLNX_OFED.

Chapter 3. API

3.1. Objects

3.1.1. struct doca_comm_channel_ep_t

Represents a Comm Channel endpoint either on the client or service side. The endpoint is needed for every other Comm Channel API function.

```
struct doca_comm_channel_ep_t;
```

3.1.2. struct doca_comm_channel_addr_t

Also referred to as `peer_address`, represents a connection and can be used to identify the source of a received message. It is required to send a message using `doca_comm_channel_ep_sendto()`.

3.2. Querying Device Capabilities

Querying the device capabilities allows users to know the derived Comm Channel limitation (see section Limitations for more information) and to set the properties of an endpoint accordingly.

The capabilities under this section, apart from maximal service name length, may vary between different devices. To select the device you wish to establish a connection upon, you may query each of the devices for its capabilities.

3.2.1. doca_comm_channel_get_max_service_name_len()

As each connection requires a name, users must know the maximal length of the name and may use this function to query it. This length includes the null-terminating character, and any name longer than this length is not accepted when trying to establish a connection with Comm Channel.

```
doca_error_t doca_comm_channel_get_max_service_name_len(uint32_t
 *max_service_name_len);
```

max_service_name_len [out]
Pointer to a parameter that holds the max service name length on success.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.2.2. doca_comm_channel_get_max_message_size()

Each connection has an upper limit for messages size. This function returns the maximal value that can be set for this property, for a given device. This limitation is important when trying to set the max message size for an endpoint with doca_comm_channel_ep_set_max_msg_size().

```
doca_error_t doca_comm_channel_get_max_message_size(struct doca_devinfo *devinfo,
 uint32_t *max_message_size);
```

devinfo [in]
Pointer to a `doca_devinfo` which should be queried for this capability.

max_message_size [out]
Pointer to a parameter that on success holds the maximal value that can be set for max message size when communicating on the provided `devinfo`.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.2.3. doca_comm_channel_get_max_send_queue_size()

Returns the maximum send queue size that can be set for a given device. This value describes the maximum possible amount of outgoing in-flight messages for a connection. This limitation is important when trying to set the max message size for an endpoint with doca_comm_channel_ep_set_send_queue_size().

```
doca_error_t doca_comm_channel_get_max_send_queue_size(struct doca_devinfo *devinfo,
 uint32_t *max_send_queue_size);
```

devinfo [in]
Pointer to a `doca_devinfo` which should be queried for this capability.

max_message_size [out]
Pointer to a parameter that on success holds the maximal value that can be set for max message size when communicating on the provided `devinfo`.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.2.4. doca_comm_channel_get_max_recv_queue_size()

Returns the maximum receive queue size that can be set for a given device. This value describes the maximum possible amount of incoming in-flight messages for a connection. This limitation is important when trying to set the max message size for an endpoint with doca_comm_channel_ep_set_send_queue_size().

```
doca_error_t doca_comm_channel_get_max_recv_queue_size(struct doca_devinfo *devinfo,
 uint32_t *max_recv_queue_size);
```

devinfo [in]
: Pointer to a `doca_devinfo` which should be queried for this capability.

max_message_size [out]
: Pointer to a parameter that on success holds the maximal value that can be set for max message size when communicating on the provided `devinfo`.

Returns
: `doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.2.5. doca_comm_channel_get_service_max_num_conne

Returns the maximum amount of connections a service on the DPU can maintain for a given device. If the maximum amount returned is zero, the number of connections is unlimited.

```
doca_error_t doca_comm_channel_get_service_max_num_connections(struct doca_devinfo
 *devinfo, uint32_t *max_num_connections);
```

devinfo [in]
: Pointer to a `doca_devinfo` which should be queried for this capability.

max_num_connections [out]
: Pointer to a parameter that on success holds the maximal value that can be set for max message size when communicating on the provided `devinfo`.

Returns
: `doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.3. Creating and Configuring Endpoint

3.3.1. doca_comm_channel_ep_create()

This function is used to create and initialize the endpoint used for all Comm Channel functions.

```
doca_error_t doca_comm_channel_ep_create(struct doca_comm_channel_ep_t **ep);
```

ep [out]

: Pointer to the created endpoint object.

Returns
: `doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.3.2. doca_comm_channel_ep_set_*() and doca_comm_channel_ep_get_*()

Use `doca_comm_channel_ep_set_*()` functions to set the properties of the endpoint and corresponding `doca_comm_channel_ep_get_*()` functions to retrieve the current properties of the endpoint.

3.3.2.1. Mandatory Properties

To use the endpoint, the following properties must be set before calling `doca_comm_channel_ep_listen()` and `doca_comm_channel_ep_connect()`.

3.3.2.1.1. doca_comm_channel_ep_set_device()

This function sets the local device through which the communication should be established.

```
doca_error_t doca_comm_channel_ep_set_device(struct doca_comm_channel_ep_t
 *local_ep, struct doca_dev *device);
```

`local_ep [in]`
Pointer to the endpoint for which the property should be set.

`device [in]`
The `doca_dev` object which should be used for communication.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.3.2.1.2. doca_comm_channel_ep_set_device_rep()

This function sets the device representor through which the communication should be established on the service side.

```
doca_error_t doca_comm_channel_ep_set_device_rep(struct doca_comm_channel_ep_t
 *local_ep, struct doca_dev_rep *device_rep);
```

`local_ep [in]`
Pointer to the endpoint for which the property should be set.

`device_rep [in]`
The `doca_dev_rep` object which should be used for communication

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful, or an error value upon failure. Possible error values are documented in the header file.

3.3.2.2. Optional Properties

The following properties have a default value and may be set as long as the EP is not yet active.

3.3.2.2.1. doca_comm_channel_ep_set_max_msg_size()

This function sets an upper limit to the size of the messages the application wishes to handle in this EP while communicating with a given endpoint. The actual `max_msg_size` may be increased by this function. If this property was not set by the user, a default value is used and may be queried using `doca_comm_channel_ep_get_max_msg_size()` function.

```
doca_error_t doca_comm_channel_ep_set_max_msg_size(struct doca_comm_channel_ep_t
 *local_ep, uint16_t max_msg_size);
```

`local_ep [in]`
Pointer to a parameter that holds the max service name length on success.

`max_msg_size [in]`
The preferred maximal message size.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful
- `DOCA_ERROR_INVALID_VALUE` if a null pointer to the endpoint is given or if `max_msg_size` is equal to 0 or above the maximal value possible for this property

3.3.2.2.2. doca_comm_channel_ep_set_send_queue_size()

This function sets the send queue size used when communicating with a given endpoint. The actual `send_queue_size` may be increased by this function. If this property has not been set by the user, a default value is used and may be queried using the `doca_comm_channel_ep_get_send_queue_size()` function.

```
doca_error_t doca_comm_channel_ep_set_send_queue_size(struct doca_comm_channel_ep_t
 *local_ep, uint16_t send_queue_size);
```

`local_ep [in]`

Pointer to a parameter that holds the max service name length on success.

`max_msg_size [in]`

The preferred maximal message size.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful
- `DOCA_ERROR_INVALID_VALUE` if a null pointer to the endpoint is given or if `max_msg_size` is equal to 0 or above the maximal value possible for this property
- The rest of the error values that may be returned are documented in the header file

3.3.2.2.3. doca_comm_channel_ep_set_recv_queue_size()

This function sets the receive queue size used when communicating with a given endpoint. The actual `recv_queue_size` may be increased by this function. If this property has not been set by the user, a default value is used which may be queried using `doca_comm_channel_ep_get_recv_queue_size()` function.

```
doca_error_t doca_comm_channel_ep_set_recv_queue_size(struct doca_comm_channel_ep_t
 *local_ep, uint16_t rcv_queue_size);
```

`local_ep [in]`

Pointer to a parameter that holds the max service name length on success.

`max_msg_size [in]`

The preferred maximal message size.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful
- `DOCA_ERROR_INVALID_VALUE` if a null pointer to the endpoint is given or if `rcv_queue_size` is equal to 0 or above the maximal value possible for this property
- The rest of the error values that may be returned are documented in the header file

3.4. Establishing Connection over Endpoint

The Comm Channel connection is established between endpoints, one on the host and the other on the DPU.

For a client, each connection requires its own EP. On the DPU side, all of the clients with the same service name on a specific representor are connected to a single EP, through which the connections are managed.

The following functions are relevant for the endpoint.

3.4.1. doca_comm_channel_ep_listen()

Used to listen on service endpoint, this function can only be called on the DPU. The service listens on the DOCA device representor provided using `doca_comm_channel_ep_set_device_rep()`.

Calling listen allows clients to connect to the service.

```
doca_error_t doca_comm_channel_ep_listen(struct doca_comm_channel_ep_t
 *local_ep, const char *name);
```

`local_ep`

Pointer to an endpoint to listen on.

`name [in]`

The name for the service to listen on. Clients must provide the same name to connect to the service.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_BAD_STATE` if mandatory properties (`doca_dev` and `doca_dev_rep`) were not set.
- `DOCA_ERROR_NOT_PERMITTED` if called on the host and not on the DPU.
- The rest of the error values that may be returned are documented in the header file.

3.4.2. doca_comm_channel_ep_connect()

Used to create a connection between a client and a service. This function can only be called on the host.

```
doca_error_t doca_comm_channel_ep_connect(struct doca_comm_channel_ep_t *local_ep,
      const char *name, struct doca_comm_channel_addr_t **peer_addr);
```

`local_ep [in]`

A pointer to an endpoint to connect from.

`name [in]`

The name of the service that the client connects to. Must be the same name the service listens on.

peer_addr [out]

Contains the pointer to the new connection.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_BAD_STATE` if mandatory property (`doca_dev`) was not set.
- `DOCA_ERROR_NOT_PERMITTED` if called on the host and not on the DPU.
- The rest of the error values that may be returned are documented in the header file.

3.5. Event Channel

Getting notifications for messages sent and received through an EP is managed by the event channel, using the functions listed here.

3.5.1. doca_comm_channel_ep_get_event_channel()

After a connection is established through the EP, this function extracts send/receive handles which can be used to get an interrupt when a new event happens using `epoll()` or a similar function.

- A send event happens when at least one in-flight message processing ends
- A receive event happens when a new incoming message is received

Users may decide to extract only one of the handles and send a NULL parameter for the other.

The event channels are owned by the endpoint and they are released when `doca_comm_channel_ep_destroy()` is called.

```
doca_error_t doca_comm_channel_ep_get_event_channel(struct doca_comm_channel_ep_t
 *local_ep,
                                                    doca_event_channel_t
 *send_event_channel, doca_event_channel_t *recv_event_channel);
```

local_ep [in]

Pointer to the endpoint for which a handle should be returned.

send_event_channel [out]

Pointer that holds a handle for sent messages if successful.

recv_event_channel [out]

Pointer that holds a handle for received messages if successful.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_BAD_STATE` if no connection has been established (i.e., `doca_comm_channel_ep_listen()` or `doca_comm_channel_ep_connect()` has not been called beforehand).

- The rest of the error values that may be returned are documented in the header file.

3.5.2. doca_comm_channel_ep_event_handle_arm_send()

After an interrupt caused by an event on the handle for sent messages, the handle should be re-armed to enable interrupts on it:

```
doca_error_t doca_comm_channel_ep_event_handle_arm_send(struct
 doca_comm_channel_ep_t *local_ep);
```

`local_ep [in]`
  Pointer to the endpoint from which the handle has been extracted.

Returns
  `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file.

3.5.3. doca_comm_channel_ep_event_handle_arm_recv()

After an interrupt caused by an event on the handle for received messages, the handle should be re-armed to enable interrupts on it:

```
doca_error_t doca_comm_channel_ep_event_handle_arm_recv(struct
 doca_comm_channel_ep_t *local_ep);
```

`local_ep [in]`
  Pointer to the endpoint from which the handle has been extracted.

Returns
  `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file.

3.6. doca_comm_channel_ep_sendto()

Used to send a message from one side to the other. This function runs in blocking or non-blocking mode. Refer to chapter Usage for more details.

```
doca_error_t doca_comm_channel_ep_sendto(struct doca_comm_channel_ep_t
 *local_ep, const void *msg
                 size_t len, int flags, struct doca_comm_channel_addr_t
 *peer_addr);
```

`local_ep [in]`
  Pointer to an endpoint to send the message from.

`msg [in]`
  Pointer to the buffer that contains the data to be sent.

`len [in]`
  Length of data to be sent.

`flags [in]`
  Currently, only DOCA_CC_MSG_FLAG_NONE is valid.

`peer_addr [in]`
Peer address to send the message to (see also struct struct doca_comm_channel_addr_t) that has been returned by `doca_comm_channel_ep_connect()` or `doca_comm_channel_rp_recvfrom()`.

Returns
`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_AGAIN` if the send queue is full and this function should be called again.
- `DOCA_ERROR_CONNECTION_RESET` if the provided `peer_addr` experienced an error and must be disconnected.
- The rest of the error values that may be returned are documented in the header file.

3.7. doca_comm_channel_ep_recvfrom

Used to receive a packet of data on either the service or the host. This function runs in non-blocking mode. Refer to chapter Usage for more details.

```
doca_error_t doca_comm_channel_ep_recvfrom(struct doca_comm_channel_ep_t
 *local_ep, void *msg,
                                                 size_t *len, int flags, struct
 doca_comm_channel_addr_t **peer_addr);
```

`local_ep [in]`
Pointer to an endpoint to receive the message on.

`msg [out]`
Pointer to a buffer that message should be written to.

`len [in/out]`
The input is the length of the given message buffer (`msg`). The output is the actual length of the received message.

`flags [in]`
DOCA_CC_MSG_FLAG_NONE.

`peer_addr [out]`
Handle to `peer_addr` that represents the connection the message arrived from.

Returns
`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_AGAIN` if no message is received.
- `DOCA_ERROR_CONNECTION_RESET` if the message received is from a `peer_addr` that has an error.
- The rest of the error values that may be returned are documented in the header file.

3.8. Information Regarding Each Connection

Each connection established over the EP is represented by a `doca_comm_channel_addr_t` structure, which can also be referred to as a `peer_addr`. This structure is returned by either `doca_comm_channel_ep_connect()` when a connection is established or by `doca_comm_channel_ep_recvfrom()` to identify the connection from which the message has been received.

3.8.1. doca_comm_channel_peer_addr_set_user_data() and doca_comm_channel_peer_addr_get_user_data()

Using `doca_comm_channel_peer_addr_set_user_data()`, users may give each connection a context, similar to an ID, to identify it later, using `doca_comm_channel_peer_addr_get_user_data()`. If a context is not set for a `peer_addr`, it is given the default value "0".

```
doca_error_t doca_comm_channel_ep_recvfrom(struct doca_comm_channel_ep_t
 *local_ep, void *msg,
                                          size_t *len, int flags, struct
 doca_comm_channel_addr_t **peer_addr);
```

`peer_addr [in]`

Pointer to `doca_comm_channel_addr_t` structure representing the connection.

`user_context [in]`

Context that should be set for the connection.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_INVALID_VALUE` if `peer_address` is `NULL`.

```
doca_error_t doca_comm_channel_peer_addr_get_user_data(struct
 doca_comm_channel_addr_t *peer_addr, uint64_t *user_context);
```

`peer_addr [in]`

Pointer to `doca_comm_channel_addr_t` structure representing the connection.

`user_context [out]`

Pointer to a parameter that holds the context if successful.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_INVALID_VALUE` if the parameters is `NULL`.

3.8.2. Querying Statistics for Connection

Using the `peer_addr`, users may gather and query the following statistics:

- The number of messages sent
- The number of bytes sent
- The number of messages received
- The number of bytes received
- The number of outgoing messages yet to be sent

3.8.2.1. doca_comm_channel_peer_addr_update_info()

Takes a snapshot with the current statistics of the connection. This function should be called prior to any statistics querying function. It is also used to check the connection status. See Connection Flow for more.

```
doca_error_t doca_comm_channel_peer_addr_update_info(struct doca_comm_channel_addr_t
 *peer_addr);
```

peer_addr [in]
  Pointer to `doca_comm_channel_addr_t` structure representing the connection.

Returns
  `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_CONNECTION_INPROGRESS` if the connection has yet to be established
- `DOCA_ERROR_CONNECTION_ABORTED` if the connection is in an error state
- The rest of the error values that may be returned are documented in the header file

3.8.2.2. doca_comm_channel_peer_addr_get_send_messages()

This function returns the total number of messages sent to a given `peer_addr` as measured when `doca_comm_channel_peer_addr_update_info()` has been last called.

```
doca_error_t doca_comm_channel_peer_addr_get_send_messages(const struct
 doca_comm_channel_addr_t *peer_addr, uint64_t *send_messages);
```

peer_addr [in]
  Pointer to `doca_comm_channel_addr_t` structure representing the connection.

send_messages [out]
  Pointer to a parameter that holds the number of messages sent through the `peer_addr` on success.

Returns
  `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.8.2.3. doca_comm_channel_peer_addr_get_send_bytes()

This function returns the total number of bytes sent to a given `peer_addr` as measured when `doca_comm_channel_peer_addr_update_info()` has been last called.

```
doca_error_t doca_comm_channel_peer_addr_get_send_bytes(const struct
 doca_comm_channel_addr_t *peer_addr, uint64_t *send_bytes);
```

`peer_addr [in]`
: Pointer to `doca_comm_channel_addr_t` structure representing the connection.

`send_bytes [out]`
: Pointer to a parameter that holds the number of bytes sent through the `peer_addr` on success.

Returns
: `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.8.2.4. doca_comm_channel_peer_addr_get_recv_messages()

This function return the total number of messages received from a given `peer_addr` as measured when `doca_comm_channel_peer_addr_update_info()` has been last called.

```
doca_error_t doca_comm_channel_peer_addr_get_recv_messages(const struct
 doca_comm_channel_addr_t *peer_addr, uint64_t *recv_messages);
```

`peer_addr [in]`
: Pointer to `doca_comm_channel_addr_t` structure representing the connection.

`recv_messages [out]`
: pointer to a parameter that holds the number of messages received from the `peer_addr` on success.

Returns
: `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.8.2.5. doca_comm_channel_peer_addr_get_recv_bytes()

This function will return the total number of bytes received from a given `peer_addr` as measured when `doca_comm_channel_peer_addr_update_info()` has been last called.

```
doca_error_t doca_comm_channel_peer_addr_get_recv_bytes(const struct
 doca_comm_channel_addr_t *peer_addr, uint64_t *recv_bytes);
```

`peer_addr [in]`
: Pointer to `doca_comm_channel_addr_t` structure representing the connection.

`recv_messages [out]`
: Pointer to a parameter that holds the number of bytes sent through the `peer_addr` on success.

Returns
: `doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.8.2.6. doca_comm_channel_peer_addr_get_send_in_flight_mes

This function will return the total number of bytes received from a given `peer_addr` as measured when `doca_comm_channel_peer_addr_update_info()` has been last called.

```
doca_error_t doca_comm_channel_peer_addr_get_send_in_flight_messages(const struct
 doca_comm_channel_addr_t *peer_addr,
              uint64_t *send_in_flight_messages);
```

`peer_addr [in]`
Pointer to `doca_comm_channel_addr_t` structure representing the connection.
`send_in_flight_messages [out]`
Pointer to a parameter that holds the number of in-flight messages to the `peer_addr` on success.
Returns
`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.9. Service State and Events

The service state and events API provides information about the state of the service including current connected clients, pending connections, and service state. All the functions in this section are relevant and can be run on the service side only.

3.9.1. doca_comm_channel_ep_get_service_event_chann

After a service is created and starts listening, this function extracts a handle which can be used to get an interrupt when a new service event happens using `epoll()` or a similar function.

The currently supported events are service failure, new client connection, and client disconnection. After an event is triggered, the application can call doca_comm_channel_ep_update_service_state_info() and the following getter functions to query the service state and connections.

The service event channel is armed automatically when calling doca_comm_channel_ep_update_service_state_info().

```
doca_error_t doca_comm_channel_ep_get_service_event_channel(struct
 doca_comm_channel_ep_t *local_ep,
                                 doca_event_channel_t *service_event_channel);
```

`local_ep [in]`

Pointer to the service endpoint that should be queried.

service_event_channel [out]

Event handle for service events.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.9.2. doca_comm_channel_ep_update_service_state_info

Tip: This function should be called prior to calling service status get functions.

Takes a snapshot of the current state of the service. The return value may indicate the service state. If the service is in error state, then it is non-recoverable and the endpoint must be destroyed.

Note: Calling this function invalidates any array received using doca_comm_channel_ep_get_peer_addr_list()

```
doca_error_t doca_comm_channel_ep_update_service_state_info(struct
 doca_comm_channel_ep_t *local_ep);
```

local_ep [in]

Pointer to the service endpoint that should be queried.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.9.3. doca_comm_channel_ep_get_peer_addr_list()

This function returns the list of connected `peer_addr`s as present when `doca_comm_channel_ep_update_service_state_info()` was last called.

This list includes only active `peer_addr`s which have not been disconnected from the client side or the service side.

The output array is only valid until `doca_comm_channel_ep_update_service_state_info()` is called again.

```
doca_error_t doca_comm_channel_ep_get_peer_addr_list(const struct
 doca_comm_channel_ep_t *local_ep,
                                                     struct doca_comm_channel_addr_t
 ***peer_addr_array,
                                                     uint32_t *peer_addr_array_len);
```

`local_ep [in]`

Pointer to the service endpoint that should be queried.

`peer_addr_array [out]`

Pointer to array of peer addresses.

`peer_addr_array_len [out]`

The number of entries in `peer_addr_array`.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.9.4. doca_comm_channel_ep_get_pending_connections

This function returns the list of pending connections as present when `doca_comm_channel_ep_update_service_state_info()` was last called. Pending connections are connections that were initiated by the client side but not complete from the service side.

If a pending connection exists, the application is expected to call `doca_comm_channel_ep_recvfrom()` to complete the connection. See Connection Flow for more.

The output array is only valid until `doca_comm_channel_ep_update_service_state_info()` is called again.

```
doca_error_t doca_comm_channel_ep_get_pending_connections(const struct
 doca_comm_channel_ep_t *local_ep,
                                                          uint32_t
 *pending_connections);
```

`local_ep [in]`

Pointer to the service endpoint that should be queried.

`pending_connections [out]`

The number of pending connections.

Returns

`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

3.10. doca_comm_channel_ep_disconnect()

Disconnects an endpoint from a specific `peer_address`. The disconnection is one-sided and the other side is unaware of it. New connections can be created afterwards. Refer to Usage for more details.

```
doca_error_t doca_comm_channel_ep_disconnect(struct doca_comm_channel_ep_t
 *local_ep, struct doca_comm_channel_addr_t *peer_addr);
```

`local_ep [in]`
Pointer to the endpoint that should be disconnected.

`peer_addr [in]`
The connection from which the endpoint should be disconnected.

Returns
`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- `DOCA_ERROR_NOT_CONNECTED` if there is no connection between the endpoint and the peer address

3.11. doca_comm_channel_ep_destroy

Disconnects all connections of the endpoint, destroys the endpoint object, and frees all related resources.

```
doca_error_t doca_comm_channel_ep_destroy(struct doca_comm_channel_ep_t *ep);
```

`local_ep [in]`
Endpoint to destroy.

Returns
`doca_error_t` value:

- `DOCA_SUCCESS` if successful.
- The rest of the error values that may be returned are documented in the header file

Chapter 4. Limitations

4.1. Endpoint Properties

The maximal values of all endpoint properties can be queried using the proper get functions (see Querying Device Capabilities). The `max_message_size`, `send_queue_size`, and `recv_queue_size` attributes may be increased internally. The updated property value can be queried with the proper get functions.

See the following table and doca_comm_channel_ep_set_*() and doca_comm_channel_ep_get_*() for more details.

Property	Get Function
Message size	`doca_comm_channel_get_max_message_size()`
Send queue size	`doca_comm_channel_get_max_send_queue_size()`
Receive queue size	`doca_comm_channel_get_max_recv_queue_size()`
Service name length	`doca_comm_channel_get_max_service_name_len()`

4.2. Multi-client

A single service on the DPU can serve multiple clients but a client can only connect to a single service.

The maximal number of clients connected to a single service can be queried using `doca_comm_channel_get_service_max_num_connections()`.

4.3. Multiple Services

Multiple endpoints can be created on the same DPU but different services listening on the same representor must have different names. Services listening on different representors can have the same name.

4.4. Threads

The DOCA Comm Channel is not thread-safe. Using a single endpoint over multiple threads is possible only with the use of locks to prevent parallel usage of the same resources. Different endpoints can be used over different threads with no restriction as each endpoint has its own resources.

Chapter 5. Usage

5.1. Objects

While working with DOCA Comm Channel, one must maintain two objects:

- `struct doca_comm_channel_ep_t`, referred to as "endpoint"
- `struct doca_comm_channel_addr_t`, referred to as "peer_address"

5.1.1. Endpoint

The endpoint object represents the endpoint of the Comm Channel, either on the client or service side. The endpoint is created by calling the `doca_comm_channel_ep_create()` function. It is required for every other Comm Channel function.

5.1.2. Peer_address

The `peer_address` structure represents a connection. It is created when a new connection is made (i.e., client calls `doca_comm_channel_ep_connect()` or a service receives a connection through `doca_comm_channel_ep_recvfrom()`). Refer to section Connection Flow for more details on connections.

The `peer_address` structure can be used to identify the source of a received message and is necessary to send a message using `doca_comm_channel_ep_sendto()`. `peer_address` has an identifier, `user_data`, which can be set by the user using `doca_comm_channel_peer_addr_user_data_set()` and retrieved using `doca_comm_channel_peer_addr_user_data_get()`. The default value for `user_data` is 0. The `user_data` field can be used to identify the `peer_address` object.

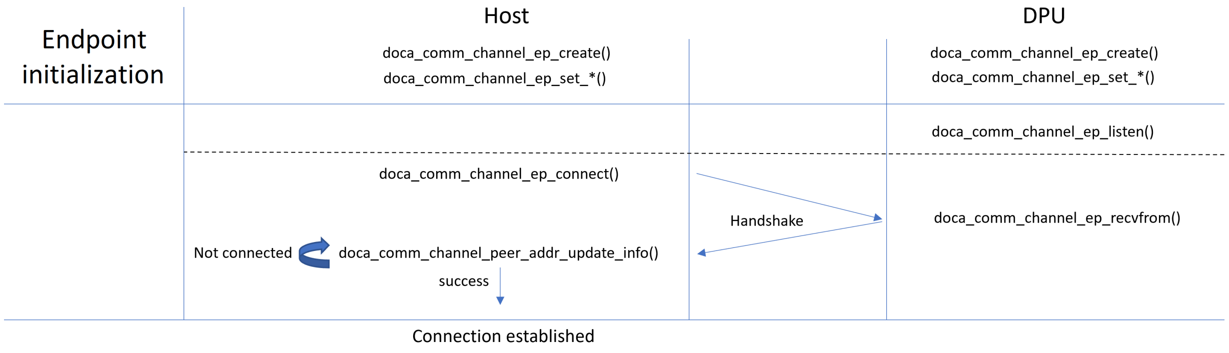
5.2. Endpoint Initialization

To start using the DOCA Comm Channel, the user must create an endpoint object using the `doca_comm_channel_ep_create()` function. After creating the endpoint object, the user must set the mandatory endpoint properties: `doca_dev` for client and service, `doca_dev_rep` for service only. The user may also set the optional endpoint properties.

For further information about endpoint initialization, refer to Establishing Connection over Endpoint.

5.3. Connection Flow

The following diagram illustrates the process of establishing a connection between the host and a service.


Host
DPU
Endpoint initialization
doca_comm_channel_ep_create()
doca_comm_channel_ep_set_*()
doca_comm_channel_ep_create()
doca_comm_channel_ep_set_*()
doca_comm_channel_ep_listen()
doca_comm_channel_ep_connect()
Handshake
doca_comm_channel_ep_recvfrom()
Not connected
doca_comm_channel_peer_addr_update_info()
success
Connection established

1. After initializing the endpoint on the service side, one should call `doca_comm_channel_ep_listen()` with a legal service name (see Limitations) to start listening.
2. After the service starts listening and the client endpoint is created, the client calls `doca_comm_channel_ep_connect()` with the same service name used for listening.

As part of the connect function, the client starts a handshake protocol with the server, which then waits until the service completes the handshake. If connect is called before the service is listening or the handshake process fails, then the connect function fails.

From the connect function, the client receives a `peer_addr` object representing the new connection to the service:

1. To check whether the connection is complete or not, the client must call `doca_comm_channel_peer_addr_update_info()` with the new `peer_addr`. Depending on the function return code, the client would know whether the connection is complete (`DOCA_SUCCESS`), rejected (`DOCA_ERROR_CONNECTION_ABORTED`) or still in progress (`DOCA_ERROR_CONNECTION_INPROGRESS`).
2. The service receiving new connections is done using `doca_comm_channel_ep_recvfrom()`. No indication is given that a new connection is made. The server keeps waiting to receive packets. If the handshake fails or is done for an existing client, then the receive function fails.

For more information, see section doca_comm_channel_ep_listen().

5.4. Data Transfer Flow

After a connection is established between client and service, both sides can send and receive data using the `doca_comm_channel_ep_sendto()` and `doca_comm_channel_ep_recvfrom()` functions, respectively.

If multiple clients are connected to the same service, then the `doca_comm_channel_ep_recvfrom()` function reads the messages in the order of their arrival, regardless of their source.

To send a message, the endpoint must obtain the target's `peer_address` object. This restriction necessitates the client to start the communication (not including the handshake), by sending the first message, for the server to obtain the client's `peer_address` object and send data back.

The `doca_comm_channel_ep_sendto()` function adds the message to an internal send queue where it is processed asynchronously. This means that even if the `doca_comm_channel_ep_sendto()` function returns with `DOCA_SUCCESS`, the message itself may fail to send (e.g., if the other side has been disconnected). If a message fails to send, the relevant `peer_address` moves to `error_state`. See Connection Errors for more.

For more information, see doca_comm_channel_ep_sendto().

5.5. Event Channel and Event Handling

When trying to send or receive messages, the application may face a situation where the resources are not ready—send queue full or no new messages received. In this case, the Comm Channel returns `DOCA_ERROR_AGAIN` for the call. This return value indicates that the function must be called again later in order to complete. To know when to call the send/receive function again, the application can use two approaches:

- Active polling – that is, to use a loop to call the send/receive functions immediately or after a certain time until the `DOCA_SUCCESS` return code is received
- Using CC event channel to know when to call the send/receive function again

  The CC event channel is a mechanism that enables getting an event when a new CC event happens. It is divided to send and receive event channels which can be retrieved using `doca_comm_channel_ep_get_event_channel()`. After retrieving the event channels, the application can use `poll` in Linux or `GetQueuedCompletionStatus` in Windows to sleep and wait for events.

  When first using the event channels and after each event is received using the event channel, it must be armed using `doca_comm_channel_ep_event_handle_arm_send()` or `doca_comm_channel_ep_event_handle_arm_recv()` to receive more events.

  For more information, see Event Channel.

5.6. Connection Errors

In certain cases, for example if a remote peer disconnects and the local endpoint tries sending a message, a `peer_addr` can move to error state. In such cases, no new messages can be sent to or received from the certain `peer_addr`.

The Comm Channel indicates a `peer_addr` is in an error state by returning `DOCA_ERROR_CONNECTION_RESET` on `doca_comm_channel_ep_sendto()` if trying to send a message to an errored `peer_addr` or on `doca_comm_channel_ep_recvfrom()` when receiving a message from a `peer_addr` marked as errored, or when calling `doca_comm_channel_peer_addr_update_info()`.

When a `peer_addr` is in an error state, it is the application's responsibility to disconnect the said `peer_addr` using `doca_comm_channel_ep_disconnect()`.

5.7. Connection Statistics

The `peer_addr` object provides a statistics mechanism. To get the updated statistics, the application should call `doca_comm_channel_peer_addr_update_info()` which saves a snapshot of the current statistics.

After calling the update function, the application can query the following statistics which return the data from that snapshot:

Statistic Function	**Returns**
`doca_comm_channel_peer_addr_get_send_messages()`	Number of messages sent to the specific `peer_addr`
`doca_comm_channel_peer_addr_get_send_bytes()`	Number of bytes sent to the specific `peer_addr`
`doca_comm_channel_peer_addr_get_recv_messages()`	Number of messages received from the specific `peer_addr`
`doca_comm_channel_peer_addr_get_recv_bytes()`	Number of bytes received from the specific `peer_addr`
`doca_comm_channel_peer_addr_get_send_in_flight_messages()`	Number of messages sent to the specific `peer_addr` and without returning a confirmation yet

The in-flight messages can be used to make sure all messages have been successfully sent before disconnecting or destroying the endpoint.

For more information, see Querying Statistics for Connection.

5.8. Service State and Connections

DOCA Comm Channel provides an API, `doca_comm_channel_ep_update_service_state_info()`, to query for the service state and connections which an application can call.

The service state is returned as the return value from the update function:

- If the return value is `DOCA_SUCCESS` the service state is operational
- If the return value is `DOCA_ERROR_CONNECTION_RESET` the service is down and cannot be recovered, and the endpoint should be destroyed

After calling the update function, the application can query the following functions which return the connection data from that snapshot:

Information Function	Returns
`doca_comm_channel_ep_get_peer_addr_list()`	Returns the list of connected `peer_addrs`
`doca_comm_channel_ep_get_pending_connections()`	Number of pending connections waiting for the service. If there are pending connections, `doca_comm_channel_ep_recvfrom()` should be called to handle them.

5.9. Disconnection Flow

Disconnection can occur specifically by using `doca_comm_channel_ep_disconnect()` or when destroying the whole endpoint.

Disconnection is one-sided, which means that the other side is unaware of the channel being closed and experiences errors when sending data. It is up to the application to synchronize the connection teardown.

Disconnection of a `peer_addr` destroys all of the resources related to it.

It is possible to perform another handshake and establish a new channel connection after disconnection.

For more information, see doca_comm_channel_ep_disconnect().

5.10. Endpoint Destruction

When calling `doca_comm_channel_ep_destroy()`, all resources related to the endpoint are freed immediately which means that if there are any messages in the send queue that have not been sent yet, they are aborted.

To make sure all messages have been successfully sent before disconnection, the application can use the `doca_comm_channel_peer_addr_get_send_in_flight_messages()` statistics function. See Connection Statistics for more information.

Chapter 6. DOCA Comm Channel Samples

This section provides Comm Channel sample implementation on top of the BlueField DPU.

6.1. Running the Sample

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.
2. To build a given sample:

```
cd /opt/mellanox/doca/samples/doca_comm_channel/<sample_name>
meson build
ninja -C build
```

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

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

```
Usage: doca_cc_server [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:
  -p, --pci-addr                    DOCA Comm Channel device PCI address
  -r, --rep-pci                     DOCA Comm Channel device representor PCI address
 (needed only on DPU)
  -t, --text                        Text to be sent to the other side of channel
```

   Note: The flag `--rep-pci` is relevant only on the DPU.

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

```
./build/doca_<sample_name> -h
```

6.2. Samples

6.2.1. CC Server

Note: This sample should be run after CC Client.

This sample illustrate how to create a simple server on the DPU to communicate with a client on the host.

The sample logic includes:

1. Creating Comm Channel endpoint.
2. Parsing PCIe address.
3. Opening Comm Channel DOCA device based on the PCIe address.
4. Opening Comm Channel DOCA device representor based on the PCIe address.
5. Setting Comm Channel endpoint properties.
6. Listening for new connections.
7. Waiting until new message arrives.
8. Sending the entered text message as a response.
9. Closing connection and freeing resources.

Reference:

- `/opt/mellanox/doca/samples/doca_comm_channel/cc_server/cc_server_sample.c`
- `/opt/mellanox/doca/samples/doca_comm_channel/cc_server/cc_server_main.c`
- `/opt/mellanox/doca/samples/doca_comm_channel/cc_server/meson.build`

6.2.2. CC Client

Note: This sample should be run after CC Server.

This sample illustrates how to create a simple client on the host to communicate with a server on the DPU.

The sample logic includes:

1. Creating Comm Channel endpoint.
2. Parsing PCIe address.
3. Opening Comm Channel DOCA device based on the PCIe address.
4. Setting Comm Channel endpoint properties.
5. Connecting current endpoint to server side.
6. Sending the entered text message.

7. Receiving server response.
8. Closing connection and freeing resources.

Reference:

- `/opt/mellanox/doca/samples/doca_comm_channel/cc_client/cc_client_sample.c`
- `/opt/mellanox/doca/samples/doca_comm_channel/cc_client/cc_client_main.c`
- `/opt/mellanox/doca/samples/doca_comm_channel/cc_client/meson.build`