DOCA LIBRARIES API

Reference Manual

Table of Contents

Chapter 1. Change Log

This chapter list changes in API that were introduced to the library.

1.3.0

- Field Groups, GPU Groups, and field watches created with a handle returned from dcgmConnect() are now cleaned up upon disconnect. dcgmConnect_v2() can be used to get the old behavior of objects persisting after disconnect.
- dcgmConnect_v2() was added as a method for specifying additional connection options when connecting to the host engine.
- dcgmUnwatchFields() was added as a method of unwatching fields that were previously watched with dcgmWatchFields()
- dcgmActionValidate_v2() was added to be able to pass more parameters to the DCGM GPU Diagnostic.
- dcgmDiagResponse_t was increased from v2 to v3. See dcgmDiagResponse_v3 for details

1.2.3

- No API changes in this version.

1.1.1

- dcgmGetAllSupportedDevices() was added as a method to get DCGM-supported GPU Ids. dcgmGetAllDevices() can still be used to get all GPU Ids in the system.

1.0.0

- Initial Release.

Chapter 2. Modules

Here is a list of all modules:

- Compatibility Management
- Deep packet inspection
- NetFlow
- Version Management

2.1. Compatibility Management

Lib to define compatibility with current version, define experimental Symbols.

To set a Symbol (or specifically a function) as experimental:

__DOCA_EXPERIMENTAL int func_declare(int param1, int param2);

To remove warnings of experimental compile with "-D DOCA_ALLOW_EXPERIMENTAL_API"

#define __DOCA_EXPERIMENTAL __attribute__((deprecated("Symbol is defined as experimental"), section(".text.experimental")))

To set a Symbol (or specifically a function) as experimental.

2.2. Deep packet inspection

DOCA Deep packet inspection library. For more details please refer to the user guide on DOCA devzone.

struct doca_dpi_config_t

DPI init configuration.

struct doca_dpi_parsing_info

L2-L4 flow information.

struct doca_dpi_result

Dequeue result.

struct doca_dpi_sig_data

Extra signature data.

struct doca_dpi_sig_info

Signature info.

struct doca_dpi_stat_info

DPI statistics.

enum doca_dpi_dequeue_status_t

Status of dequeue operation.

Values

DOCA_DPI_DEQ_NA
No DPI enqueued jobs done, or no packets to dequeue

DOCA_DPI_DEQ_READY
DPI Job and result is valid

enum doca_dpi_enqueue_status_t

Status of enqueue operation.

Values

DOCA_DPI_ENQ_PROCESSING
Packet enqueued for processing

DOCA_DPI_ENQ_PACKET_EMPTY
No payload, packet was not queued

DOCA_DPI_ENQ_BUSY
Packet cannot be enqueued, queue is full

DOCA_DPI_ENQ_INVALID_DB
load_signatures failed, or was never called

DOCA_DPI_ENQ_INTERNAL_ERR

enum doca_dpi_flow_status_t

Status of enqueued entry.

Values

DOCA_DPI_STATUS_LAST_PACKET = 1<<1
Indicates there are no more packets in queue from this flow.

DOCA_DPI_STATUS_DESTROYED = 1<<2
Indicates flow was destroyed while being processed

DOCA_DPI_STATUS_NEW_MATCH = 1<<3
Indicates flow was matched on current dequeue

__DOCA_EXPERIMENTAL int doca_dpi_dequeue (doca_dpi_ctx *ctx, uint16_t dpi_q, doca_dpi_result *result)

Dequeues packets after processing.

Parameters

ctx
The DPI context.

dpi_q
The DPI queue from which to enqueue the flows.

result
Output, matching result.

Returns

doca_dpi_dequeue_status_t if successful, error code otherwise

Description

Only packets enqueued for processing will be returned by this API. Packets will return in the order they were enqueued.

__DOCA_EXPERIMENTAL void doca_dpi_destroy (doca_dpi_ctx *ctx)

Free the DPI memory and releases the regex engine.

Parameters

ctx
DPI context to destroy.

__DOCA_EXPERIMENTAL int doca_dpi_enqueue (doca_dpi_flow_ctx *flow_ctx, rte_mbuf *pkt, bool initiator, uint32_t payload_offset, void *user_data)

Enqueue a new DPI job for processing.

Parameters

flow_ctx
The flow context handler.
pkt
The mbuf to be processed.
initiator
Indicates to which direction the packet belongs. Typically, the first packet will arrive from the initiator.
payload_offset
Indicates where the packet's payload begins.
user_data
Private user data to b returned when the DPI job is dequeued.

Returns

doca_dpi_enqueue_status_t or other error code.

Description

This function is thread-safe per queue. For best performance it should always be called form the same thread/queue on which the flow was created. See Multithreading section of the DPI Programming Guide for more details.

Once a packet is enqueued, the DPI engine will increase ref count in the mbuf. User must not change or reuse the mbuf while it is being processed. See "Packet Ownership" section of the DPI Programming Guide for more details.

The injected packet has to be stripped of FCS. A packet will not be enqueued if:

- Payload length = 0

__DOCA_EXPERIMENTAL doca_dpi_flow_ctx *doca_dpi_flow_create (doca_dpi_ctx *ctx, uint16_t dpi_q, const doca_dpi_parsing_info *parsing_info, int *error, doca_dpi_result *result)

Creates a new flow on a queue.

Parameters

ctx
The DPI context.

dpi_q
The DPI queue on which to create the flows

parsing_info
L3/L4 information.

error
Output, Negative if error occurred.

result
Output, If flow was matched based on the parsing info, result->matched will be true.

Returns

NULL on error.

Description

Must be called before enqueuing any new packet. A flow must not be created on 2 different queues.

__DOCA_EXPERIMENTAL void doca_dpi_flow_destroy (doca_dpi_flow_ctx *flow_ctx)

Destroys a flow on a queue.

Parameters

flow_ctx
The flow context to destroy.

Description

Should be called when a flow is terminated or times out

__DOCA_EXPERIMENTAL int doca_dpi_flow_match_get (const doca_dpi_flow_ctx *flow_ctx, doca_dpi_result *result)

Query a flow's match.

Parameters

flow_ctx
The flow context of the flow to be queried.

result
Output, latest match on this flow.

Returns

0 on success, error code otherwise.

__DOCA_EXPERIMENTAL doca_dpi_ctx *doca_dpi_init (const doca_dpi_config_t *config, int *error)

Initialize the DPI library.

Parameters

config
See doca_dpi_config_t for details.

error
Output error, negative value indicates an error.

Returns

doca_dpi_ctx - dpi opaque context, NULL on error.

Description

This function must be invoked first before any function in the API. It should be invoked once per process. This call will probe the first regex device it finds (0).

__DOCA_EXPERIMENTAL int doca_dpi_load_signatures (doca_dpi_ctx *ctx, const char *cdo_file)

Loads the cdo file.

Parameters

ctx
The DPI context.

cdo_file
CDO file created by the DPI compiler.

Returns

0 on success, error code otherwise.

Description

The cdo file contains signature information. The cdo file must be loaded before any enqueue call.

Database update: When a new signatures database is available, the user may call this function again. The newly loaded CDO must contain the signatures of the previously loaded CDO or result will be undefined.

__DOCA_EXPERIMENTAL int doca_dpi_signature_get (const doca_dpi_ctx *ctx, uint32_t sig_id, doca_dpi_sig_data *sig_data)

Returns a specific sig info.

Parameters

ctx
The DPI context.

sig_id
The DPI queue on which the flow was created.

sig_data
Output of the sig metadata.

Returns

0 on success, error code otherwise.

__DOCA_EXPERIMENTAL int doca_dpi_signatures_get (const doca_dpi_ctx *ctx, doca_dpi_sig_data **sig_data)

Returns all signatures.

Parameters

ctx
The DPI context.

sig_data
Output of the sig data.

Returns

0 on success, error code otherwise.

Description

It is the responsibility of the user to free the array. Because this function copies all the sig info, it is highly recommended to call this function only once after loading the database, and not during packet processing.

__DOCA_EXPERIMENTAL void doca_dpi_stat_get (const doca_dpi_ctx *ctx, bool clear, doca_dpi_stat_info *stats)

Returns DPI statistics.

Parameters

ctx
The DPI context.

clear
Clear the statistics after fetching them.

stats
Output struct containing the statistics.

2.3. NetFlow

Doca lib for export a netflow packet to a netflow collector.

This lib simplifies and centralizes the formatting and exporting of netflow packets. Netflow is a protocol for exporting information about the device network flows to a netflow collector that will aggregate and analyze the data. After creating conf file and invoke init function, the lib send function can be called with netflow struct to send a netflow packet with the format to the collector of choice specified in the conf file. The lib uses the netflow protocol specified by cisco.

See also:

https://netflow.caligare.com/netflow_v9.htm

Conf File structure:

doca_netflow.conf

[doca_netflow_conf]

target = <hostname = name/ipv4/ipv6>:<port = integer>

source_id = <ID = integer>

version = <version = 9>

doca_netflow_default.conf

[doca_netflow_conf]

target = 127.0.0.1:2055

source_id = 10

version = 9

Limitations:

The lib supports the netflow V9 format. The lib is not thread safe.

struct doca_netflow_default_record

Flow record, represent a flow at specific moment, usually after a flow end or after some timeout. Each one is a data record that will appear in the collector. This template is based on V5 fields with additional V9 fields.

struct doca_netflow_flowset_field

One field in netflow template, please look at doca_netflow_types for type macros.

struct doca_netflow_template

Template for the records. struct record_exmaple { uint32_t src_addr_V4; uint32_t dst_addr_V4; } struct doca_netflow_flowset_field fields[] = { {.type = DOCA_NETFLOW_IPV4_SRC_ADDR, .length = DOCA_NETFLOW_IPV4_SRC_ADDR_DEFAULT_LENGTH}, {.type = DOCA_NETFLOW_IPV4_DST_ADDR, .length =

DOCA_NETFLOW_IPV4_DST_ADDR_DEFAULT_LENGTH} }; struct doca_netflow_template template = { .field_count = 2; .fields = fields; };.

struct doca_netflow_default_record ::packed

Flow record, represent a flow at specific moment, usually after a flow end or after some timeout. Each one is a data record that will appear in the collector. This template is based on V5 fields with additional V9 fields.

__DOCA_EXPERIMENTAL void doca_netflow_exporter_destroy (void)

Free the exporter memory and close connection.

__DOCA_EXPERIMENTAL int doca_netflow_exporter_init (const char *netflow_conf_file)

Init exporter memory, set configs and open connection.

Parameters

netflow_conf_file

Doca netflow configure file pointer including a section marked as [doca_netflow_conf], if a NULL pointer is given then look at default path in DOCA_NETFLOW_CONF_DEFAULT_PATH. This function can be called again only after doca_netflow_exporter_destroy was called.

Returns

0 on success, error code otherwise.

__DOCA_EXPERIMENTAL int doca_netflow_exporter_send (const doca_netflow_template *template, const void **records, size_t length, int *error)

Sending netflow records. Need to init first.

Parameters

template

Template pointer how the records are structed. for more info reffer to doca_netflow_template.

records
Array of pointers to the flows structs to send, must be packed. strings must be a direct array in the struct not a pointer.

length
Records array size.

error
If return value is -1 populate this field with the error.

Returns

Number of records sent, -1 on error.

Description

Note:

- if the return value is positive but not equal to length then just some of the records have sent. the send function sould run again with the remaining records. please reffer to the exmaple.
- When sending more then 30 records the lib split the records to multiple packets because packet can send up to 30 records (Netflow protocol limit)

doca_netflow_template *doca_netflow_template_default_get (void)

Return a default doca_netflow_template for use in send function, if using default template use doca_netflow_default_record struct for records.

Returns

pointer containing the default template

#define DOCA_NETFLOW_CONF_DEFAULT_PATH "/etc/doca_netflow.conf"

default conf path to look for

2.4. Version Management

Define function to get doca version and version compare.

const char *doca_version (void)

Function returning version string.

Returns

string version number of a format major.minor.patch

#define DOCA_CURRENT_VERSION_NUM DOCA_VERSION_NUM(DOCA_VER_MAJOR, DOCA_VER_MINOR, DOCA_VER_PATCH)

Macro of current version number for comparisons.

#define DOCA_VER_MAJOR 0

Major version number 0-255.

#define DOCA_VER_MINOR 1

Minor version number 0-255.

#define DOCA_VER_PATCH 0

Patch version number 0-255.

#define DOCA_VERSION_EQ_CURRENT (DOCA_VERSION_NUM(major, minor, patch) == DOCA_CURRENT_VERSION_NUM)

Return 1 if the version specified is equal to current.

#define DOCA_VERSION_LTE_CURRENT (DOCA_VERSION_NUM(major, minor, patch) <= DOCA_CURRENT_VERSION_NUM)

Return 1 if the version specified is less then or equal to current.

#define DOCA_VERSION_NUM ((major) << 16 | (minor) << 8 | (patch))

Macro of version number for comparisons.

Chapter 3. Data Structures

Here are the data structures with brief descriptions:

doca_dpi_config_t
DPI init configuration

doca_dpi_parsing_info
L2-L4 flow information

doca_dpi_result
Dequeue result

doca_dpi_sig_data
Extra signature data

doca_dpi_sig_info
Signature info

doca_dpi_stat_info
DPI statistics

doca_netflow_default_record
Flow record, represent a flow at specific moment, usually after a flow end or after some timeout. Each one is a data record that will appear in the collector. This template is based on V5 fields with additional V9 fields

doca_netflow_flowset_field
One field in netflow template, please look at doca_netflow_types for type macros

doca_netflow_template
Template for the records. struct record_exmaple { uint32_t src_addr_V4; uint32_t dst_addr_V4; } struct doca_netflow_flowset_field fields[] = { {.type = DOCA_NETFLOW_IPV4_SRC_ADDR, .length = DOCA_NETFLOW_IPV4_SRC_ADDR_DEFAULT_LENGTH}, {.type = DOCA_NETFLOW_IPV4_DST_ADDR, .length = DOCA_NETFLOW_IPV4_DST_ADDR_DEFAULT_LENGTH} }; struct doca_netflow_template template = { .field_count = 2; .fields = fields; };

3.1. doca_dpi_config_t Struct Reference

DPI init configuration.

uint32_t doca_dpi_config_t::max_packets_per_queue

Number of packets concurrently processed by the DPI engine.

uint32_t doca_dpi_config_t::max_sig_match_len

The minimum required overlap between two packets for regex match

uint16_t doca_dpi_config_t::nb_queues

Number of DPI queues

3.2. doca_dpi_parsing_info Struct Reference

L2-L4 flow information.

uint16_t doca_dpi_parsing_info::ethertype

Ethertype of the packet

in_addr doca_dpi_parsing_info::ipv4

Ipv4 destination address

in6_addr doca_dpi_parsing_info::ipv6

Ipv6 destination address

uint16_t doca_dpi_parsing_info::l4_dport

Layer 4 destination port

uint8_t doca_dpi_parsing_info::l4_protocol

Layer 4 protocol

uint16_t doca_dpi_parsing_info::l4_sport

Layer 4 source port

3.3. doca_dpi_result Struct Reference

Dequeue result.

struct doca_dpi_sig_info doca_dpi_result::info

Signature information

bool doca_dpi_result::matched

Indicates flow was matched

rte_mbuf *doca_dpi_result::pkt

Pkt provided on enqueue

int doca_dpi_result::status_flags

doca_dpi_flow_status flags

void *doca_dpi_result::user_data

User data provided on enqueue

3.4. doca_dpi_sig_data Struct Reference

Extra signature data.

char doca_dpi_sig_data::name

Signature name

uint32_t doca_dpi_sig_data::sig_id

Signature ID as provided in the signature

3.5. doca_dpi_sig_info Struct Reference

Signature info.

uint32_t doca_dpi_sig_info::sig_id

Signature ID as provided in the signature

3.6. doca_dpi_stat_info Struct Reference

DPI statistics.

uint32_t doca_dpi_stat_info::nb_http_parser_based

Total number of http signature matches

uint32_t doca_dpi_stat_info::nb_matches

Total number of signature matches

uint32_t doca_dpi_stat_info::nb_other_l4

Total number of other l5 signature matches

uint32_t doca_dpi_stat_info::nb_other_l7

Total number of other l7 signature matches

uint32_t doca_dpi_stat_info::nb_scanned_pkts

Total number of scanned packets

uint32_t doca_dpi_stat_info::nb_ssl_parser_based

Total number of ssl signature matches

uint32_t doca_dpi_stat_info::nb_tcp_based

Total number of tco signature matches

uint32_t doca_dpi_stat_info::nb_udp_based

Total number of udp signature matches

3.7. doca_netflow_default_record Struct Reference

Flow record, represent a flow at specific moment, usually after a flow end or after some timeout. Each one is a data record that will appear in the collector. This template is based on V5 fields with additional V9 fields.

char doca_netflow_default_record::application_name

Name associated with a classification

uint32_t doca_netflow_default_record::d_octets

Octets sent in Duration.

uint32_t doca_netflow_default_record::d_pkts

Packets sent in Duration

uint32_t doca_netflow_default_record::dst_addr_v4

Destination IP Address

uint16_t doca_netflow_default_record::dst_as

originating AS of destination address

uint8_t doca_netflow_default_record::dst_mask

destination address prefix mask bits

uint16_t doca_netflow_default_record::dst_port

TCP/UDP destination port number or equivalent

uint32_t doca_netflow_default_record::first

SysUptime at start of flow

uint64_t doca_netflow_default_record::flow_id

This identifies a transaction within a connection

uint16_t doca_netflow_default_record::input

Input interface index

uint32_t doca_netflow_default_record::last

and of last packet of flow

uint32_t doca_netflow_default_record::next_hop

Next hop router's IP Address

uint16_t doca_netflow_default_record::output

Output interface index

uint8_t doca_netflow_default_record::protocol

IP protocol type (for example, TCP = 6; UDP = 17)

uint32_t doca_netflow_default_record::src_addr_v4

Source IP Address

uint16_t doca_netflow_default_record::src_as

originating AS of source address

uint8_t doca_netflow_default_record::src_mask

source address prefix mask bits

uint16_t doca_netflow_default_record::src_port

TCP/UDP source port number or equivalent

uint8_t doca_netflow_default_record::tcp_flags

Cumulative OR of tcp flags

uint8_t doca_netflow_default_record::tos

IP Type-of-Service

3.8. doca_netflow_flowset_field Struct Reference

One field in netflow template, please look at doca_netflow_types for type macros.

int doca_netflow_flowset_field::length

field len in bytes (see link) - will be converted to uint16

int doca_netflow_flowset_field::type

field number id (see link) - will be converted to uint16

3.9. doca_netflow_template Struct Reference

Template for the records. struct record_exmaple { uint32_t src_addr_V4; uint32_t dst_addr_V4; } struct doca_netflow_flowset_field fields[] = { {.type = DOCA_NETFLOW_IPV4_SRC_ADDR, .length = DOCA_NETFLOW_IPV4_SRC_ADDR_DEFAULT_LENGTH}, {.type = DOCA_NETFLOW_IPV4_DST_ADDR, .length = DOCA_NETFLOW_IPV4_DST_ADDR_DEFAULT_LENGTH} }; struct doca_netflow_template template = { .field_count = 2; .fields = fields; };.

int doca_netflow_template::field_count

number of fields in 'fields' array - will be converted to uint16

doca_netflow_flowset_field *doca_netflow_template::fields

array of field info

Chapter 4. Data Fields

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

A

application_name
doca_netflow_default_record

D

d_octets
doca_netflow_default_record
d_pkts
doca_netflow_default_record
dst_addr_v4
doca_netflow_default_record
dst_as
doca_netflow_default_record
dst_mask
doca_netflow_default_record
dst_port
doca_netflow_default_record

E

ethertype
doca_dpi_parsing_info

F

field_count
doca_netflow_template
fields
doca_netflow_template
first
doca_netflow_default_record

flow_id
doca_netflow_default_record

I

info
doca_dpi_result
input
doca_netflow_default_record
ipv4
doca_dpi_parsing_info
ipv6
doca_dpi_parsing_info

L

l4_dport
doca_dpi_parsing_info
l4_protocol
doca_dpi_parsing_info
l4_sport
doca_dpi_parsing_info
last
doca_netflow_default_record
length
doca_netflow_flowset_field

M

matched
doca_dpi_result
max_packets_per_queue
doca_dpi_config_t
max_sig_match_len
doca_dpi_config_t

N

name
doca_dpi_sig_data
nb_http_parser_based
doca_dpi_stat_info
nb_matches
doca_dpi_stat_info
nb_other_l4
doca_dpi_stat_info

nb_other_l7
doca_dpi_stat_info
nb_queues
doca_dpi_config_t
nb_scanned_pkts
doca_dpi_stat_info
nb_ssl_parser_based
doca_dpi_stat_info
nb_tcp_based
doca_dpi_stat_info
nb_udp_based
doca_dpi_stat_info
next_hop
doca_netflow_default_record

O

output
doca_netflow_default_record

P

pkt
doca_dpi_result
protocol
doca_netflow_default_record

S

sig_id
doca_dpi_sig_info
doca_dpi_sig_data
src_addr_v4
doca_netflow_default_record
src_as
doca_netflow_default_record
src_mask
doca_netflow_default_record
src_port
doca_netflow_default_record
status_flags
doca_dpi_result

T

tcp_flags
doca_netflow_default_record

tos
doca_netflow_default_record
type
doca_netflow_flowset_field

U

user_data
doca_dpi_result

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 (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. 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