Migration to RDMA-Core

Exported on Jan/28/2022 08:00 AM

Table of Contents

Overview

As a leading RDMA-based interconnect provider, Mellanox strives to bring all its software drivers and features to the Linux Upstream in order to guarantee best out-of-the box experience.

The Mellanox Linux driver development methodology is “Upstream first”. Therefore, Mellanox contributes all its developed features to the RDMA subsystem before porting them to its own Linux driver, that is, MLNX_OFED.

Each generation of the ConnectX® network adapters includes dedicated hardware that unleashes cutting-edge RDMA features, delivering compute and data intensive applications with highest performance and scalability. The RDMA subsystem verbs library is a generic API library serving all vendors and all RDMA applications.

In order to enable the usage of new cutting-edge technologies as soon as they became available, Mellanox came up with an interim solution to work around the long wait periods before these features become supported and incorporated in Linux Upstream. As such, and until a faster acceptance process for the verbs API was implemented for the RDMA subsystem, Mellanox has developed and maintained a private API with the prefix “ibv_exp” (known as “experimental verbs”). This private API enabled Mellanox to expose early on the latest hardware offloads to its customers’ applications via the Mellanox Linux driver (MLNX_OFED).

Nowadays, the acceptance process has become faster. Therefore, the private verbs API has become redundant. Mellanox has set new APIs in Linux Upstream, which can be used to achieve most of the legacy experimental verbs’ functionalities and more. Going forward, future RDMA-related features will only be available through the Upstream RDMA subsystem.

The userspace RDMA subsystem (known as RDMA-Core) library introduces the following new libmlx5 API channels:

- mlx5dv - a channel for configuring hardware objects with Mellanox specific attributes.
- mlx5dv_devx - a channel for exposing raw Mellanox hardware objects, enabling the use of hardware features with minimal Kernel changes (some of the mlx5dv APIs use mlx5dv_devx at userspace level).

Application owners who wish to use the latest RDMA-based in-network computing acceleration engines and hardware offloads are kindly asked to use the RDMA-Core APIs and avoid using the experimental verbs APIs.

Application owners currently using the experimental verbs (ibv_exp) APIs are highly encouraged to begin porting their applications to use the RDMA-core APIs instead. The porting process is for the most part straight-forward.

Important Notes:

- Mellanox plans to deprecate the experimental verbs API in the near future, following an advance twelve-month notice.
- As of MLNX_OFED v4.7:
    - No new features will be added to the experimental verbs API.
    - mlx5dv and mlx5dv_devx APIs will become default APIs. However, the experimental verbs API will not be exposed.
    - Application owners can still choose to install the experimental verbs API and use the verbs (using a special installation flag).

For further assistance, please contact Mellanox Support.

MLNX_OFED Features Verbs and Capabilities

Accelerated Verbs

The accelerated verbs should be used to Post-Receive to the created WQs and to poll for completions.

Experimental Accelerated Verbs

- ibv_exp_query_intf
- ibv_exp_release_intf

Accelerated Verbs RDMA-Core Support

Accelerated verbs are replaced in RDMA-Core by the mlx5dv API in libmlx5.

AS Notify

AS Notify is a low-latency hardware-based thread wakeup mechanism. Instead of actively polling a Completion Queue (CQ), the user application can arm the CQ and issue a “wait” instruction to put the user thread to sleep. The user application will be woken up by AS_notify interrupt once a completion event takes place. Note that when AS_notify interrupt cannot be triggered, firmware will fall back into the traditional MSI interrupt.

AS Notify Experimental Verbs

- ibv_exp_create_cq
    - IBV_EXP_CQ_AS_NOTIFY

AS Notify Experimental Capabilities

IBV_EXP_CQ_AS_NOTIFY

AS Notify RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Atomic Operations

Atomic Operations execute a 64-bit operation at a specified address on a remote node. The operations atomically read, modify and write the destination address and guarantee that operations on this address by other QPs on the same CA do not occur between the Read and Write. The scope of the atomicity guarantee may optionally extend to other CPUs and HCAs.

Atomic Operations Experimental vs. RDMA-Core Verbs and Device Attributes

Experimental		RDMA-Core	
Verbs			
ibv_exp_query_device	exp_atomic_cap • IBV_EXP_ATOMIC_NONE • IBV_EXP_ATOMIC_HCA • IBV_EXP_ATOMIC_GLOB	**ibv_query_device_ex**	• IBV_ATOMIC_NONE • IBV_ATOMIC_HCA • IBV_ATOMIC_GLOB
	pci_atomic_caps		pci_atomic_caps

ibv_exp_reg_mr		**ibv_reg_mr**	
	IBV_EXP_ACCESS_REMOTE_ATOMIC		IBV_ACCESS_REMOTE_ATOMIC
Device Attributes			
IBV_EXP_DEVICE_ATTR_PCI_ATOMIC_CAPS			

Extended Atomics

Extended atomic operations are a set of operations beyond those defined by the IB Spec. Atomicity guarantees, Atomic Ack generation, ordering rules and error behavior for this set of extended atomic operations is the same as that for IB standard Atomic operations.

Extended Atomics Experimental Verbs

- Ibv_exp_post_send
  - IBV_EXP_WR_EXT_MASKED_ATOMIC_CMP_AND_SWP
  - IBV_EXP_WR_EXT_MASKED_ATOMIC_FETCH_AND_ADD
  - IBV_EXP_SEND_EXT_ATOMIC_INLINE
  - ext_op.masked_atomics
- ibv_exp_create_qp
  - max_atomic_arg
- ibv_exp_poll_cq
  - IBV_EXP_WC_MASKED_COMP_SWAP
  - IBV_EXP_WC_MASKED_FETCH_ADD
- ibv_exp_query_device
  - ext_atom

Extended Atomics Experimental Capabilities and Device Attributes

- IBV_EXP_DEVICE_EXT_ATOMICS
- IBV_EXP_DEVICE_EXT_MASKED_ATOMICS
- IBV_EXP_DEVICE_ATTR_EXP_CAP_FLAGS
- IBV_EXP_DEVICE_ATTR_EXT_ATOMIC_ARGS
- IBV_EXP_DEVICE_ATTR_MASKED_ATOMICS

RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Atomic Response Endianness

Atomic Operation in Connect-IB (MT27600) are fully supported on big endian (BE) machines (e.g. PPC). Their support is limited to little-endian machines (e.g. x86).

When using `ibv_exp_query_device` on little-endian machines with Connect-IB, the attr.exp_atomic_cap is set to IBV_EXP_ATOMIC_HCA_REPLY_BE, which indicates that if enabled, the atomic operation replied value is big-endian and contradicts the host endianness.

Enabling atomic operation with this endianness contradiction is by setting the IBV_EXP_QP_CREATE_QTOMIC_BE_REPLY flag in exp_create_flag in ibv_exp_create_qp.

RDMA-Core libmlx5 will not verify any atomic QP creation flag upon secnifn Atomic operations.

If an application swaps data between BE and host endianness over legacy library, it shall continue doing so.

For further information, please contact Mellanox Support.

Checksum Offload

The device supports calculation of checksum on transmitted packets and validation of received packets checksum. The adapter device offloads IPv4 checksum (L3) and TCP/UDP checksum (L4).

Checksum calculation is supported for TCP/UDP running over IPv4 and IPv6.

Checksum Offload Experimental Verbs

- ibv_exp_post_send
    - IBV_EXP_SEND_IP_CSUM

Checksum Offload Experimental Capabilities

- IBV_EXP_DEVICE_RX_CSUM_TCP_UDP_PKT
- IBV_EXP_DEVICE_RX_CSUM_IP_PKT
- IBV_EXP_SW_PARSING_CSUM

Checksum Offload RDMA-Core Verbs

- ibv_post_send
    - IBV_SEND_IP_CSUM
- ibv_query_device_ex
    - IBV_RAW_PACKET_CAP_IP_CSUM
- ibv_wr_post
    - IBV_SEND_IP_CSUM

Checksum Offload RDMA-Core Experimental Capabilities

- IBV_DEVICE_UD_IP_CSUM
- IBV_DEVICE_RC_IP_CSUM
- IBV_DEVICE_RAW_IP_CSUM

Contiguous Pages

Contiguous Pages improves performance by allocating user memory regions over physical contiguous pages. It enables a user application to ask low level drivers to allocate contiguous memory for it as part of `ibv_reg_mr`.

Contiguous Pages Environmental Variables

Environmental Variables	Value
MLX_QP_ALLOC_TYPE	CONTIG, PREFER_CONTIG
MLX_CQ_ALLOC_TYPE	CONTIG, PREFER_CONTIG
MLX_MR_MAX_LOG2_CONTIG_BSIZE	_MAX_LOG2_CONTIG_BLOCK_SIZE (23)
MLX_MR_MIN_LOG2_CONTIG_BSIZE	_MIN_LOG2_CONTIG_BLOCK_SIZE (12)
HUGE_CQ	
qp_huge_key	

Contiguous Pages Experimental Verbs

- ibv_exp_reg_mr
  - IBV_EXP_ACCESS_ALLOCATE_MR
- ibv_exp_rereg_mr
  - IBV_EXP_ACCESS_ALLOCATE_MR

Contiguous Pages RDMA-Core Support

Contiguous memory in RDMA-Core is achieved using huge pages.

CQE Compression

CQE compression saves PCIe bandwidth by compressing a few CQEs into a smaller amount of bytes on the PCIe.

CQE Compression Environmental Variables

- MLX5_ENABLE_CQE_COMPRESSION

CQE Compression Experimental Verbs

- ibv_exp_create_cq
    - IBV_EXP_CQ_COMPRESSED_CQE

CQE Compression Capabilities

- IBV_EXP_CQ_COMPRESSED_CQE

CQE Compression RDMA-Core Verbs

- mlx5dv_create_cq
    - cqe_comp_res_format
    - MLX5DV_CQ_INIT_ATTR_MASK_COMPRESSED_CQE

CQE Compression RDMA-Core Capabilities

- mlx5dv_query_device
    - mlx5dv_cqe_comp_caps

Relevant Man Pages

mlx5dv_create_cq: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_create_cq.3.md

CQ Moderation

CQ moderation enables the user to moderate completion events by specifying the number of completions that cause an event and the timeout in micro seconds to cause the event even if the number of completions was not reached.

CQ Moderation Experimental Verbs

- ibv_exp_modify_cq
    - cq_count
    - cq_period

CQ Moderation Experimental Capabilities

IBV_EXP_CQ_ATTR_MODERATION

CQ Moderation RDMA-Core Verbs

- ibv_modify_cq
    - ibv_moderate_cq

Relevant Man Pages

ibv_modify_cq: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_modify_cq.3

Cross Channel

Cross Channel is a Verbs API that enables one to define a list of communication tasks and synchronization points and post this list as a single WQE which the HCA progresses entirely. Once posted, only completion of the list is polled for by the CPU. In this way, one can create and schedule complex communication and coordination patterns among all nodes in a cluster.

Cross Channel Experimental Verbs

- ibv_exp_post_task
- ibv_exp_create_qp
  - IBV_EXP_QP_CREATE_CROSS_CHANNEL
  - IBV_EXP_QP_CREATE_MANAGED_SEND
  - IBV_EXP_QP_CREATE_MANAGED_RECV
  - IBV_EXP_QP_CREATE_IGNORE_SQ_OVERFLOW
  - IBV_EXP_QP_CREATE_IGNORE_RQ_OVERFLOW
- Ibv_exp_create_cq
  - IBV_EXP_CQ_CREATE_CROSS_CHANNEL
- Ibv_exp_modify_cq
  - IBV_EXP_CQ_IGNORE_OVERRUN
- Ibv_exp_post_send
  - union task
  - ibv_exp_calc_op
  - ibv_exp_calc_data_type
  - ibv_exp_calc_data_size
  - IBV_EXP_WR_SEND_ENABLE
  - IBV_EXP_WR_RECV_ENABLE
  - IBV_EXP_WR_CQE_WAIT
- ibv_exp_query_device
  - calc_cap

Cross Channel Experimental Capabilities and Device Attributes

- IBV_EXP_DEVICE_CROSS_CHANNEL
- IBV_EXP_DEVICE_ATTR_CALC_CAP

Cross Channel RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Device Memory

Device Memory is a verbs API that allows using on-chip memory, located on the device, as a data buffer for send/receive and RDMA operations. The device memory can be mapped and accessed directly by user and kernel applications, and can be allocated in various sizes, registered as memory regions with local and remote access keys for performing the send/ receive and RDMA operations. Using the device memory to store packets for transmission can significantly reduce transmission latency compared to the host memory.

Device Memory Experimental Verbs

- ibv_exp_alloc_dm
- ibv_exp_memcpy_dm
- ibv_exp_free_dm
- ibv_exp_reg_mr
    - ibv_exp_reg_mr_dm

Device Memory RDMA-Core Verbs

- Ibv_alloc_dm
- Ibv_reg_dm_mr
- ibv_memcpy_to_dm
- ibv_memcpy_from_dm
- ibv_free_dm

Relevant Man Pages

ibv_alloc_dm: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_alloc_dm.3

Example

See example in libibverbs/examples/rc_pingpong.c

Dynamically Connected (DC) QPs

A dynamically connected transport service is an extension to transport services that enables a higher degree of scalability while maintaining high performance for sparse traffic. Utilization of DC transport reduces the total number of QPs required system-wide, by having QPs of reliable type dynamically connect and disconnect from any remote node.

> ⚠ DC QP is only supported in mlx5 driver.

DC QPs Experimental vs. RDMA-Core Verbs and Capabilities

Experimental		RDMA-Core	
Verbs			
ibv_exp_create_dct		**mlx5dv_create_qp** • qp_type = IBV_QPT_DRIVER • dv_init_attr.dc_init_attr.dc_type = MLX5DV_DCTYPE_DCT	
	dc_key		dct_access_key
	access flags		access flags
	flow_label		ah_attr.grh.flow_label
	Inline_size		Not supported by RDMA-Core
	IBV_EXP_DCT_OOO_RW_DATA_PLACEMENT		Supported through opensm
ibv_exp_destroy_dct		**ibv_destroy_qp**	
ibv_exp_query_dct		**ibv_query_qp**	
	dc_key		Not supported by RDMA-Core
	port		Port
	access_flags		access_flags
	min_rnr_timer		min_rnr_timer
	tclass		tclass
	flow_label		flow_label
	mtu		mtu
	pkey_index		pkey_index
	gid_index		gid_index
	hop_limit		hop_limit
	key_violations		Not supported by RDMA-Core
	state		Not supported by RDMA-Core

ibv_exp_post_send	dct_access_key, dct_number	**mlx5dv_wr_post**	
ibv_exp_poll_cq	IBV_EXP_WC_DCT	**mlx5dv_wr_set_dc_addr**	
ibv_exp_modify_qp	dct_key, IBV_EXP_QP_DC_KEY		Not supported by RDMA-Core
Capabilities and Device Attributes			
	IBV_EXP_DEVICE_DC_TRANSPORT	No DC capabilities flags needed in RDMA-Core	
	IBV_EXP_DEVICE_DC_RD_REQ, IBV_EXP_DEVICE_DC_RD_RES		
	IBV_EXP_DEVICE_DC_INFO		
	IBV_EXP_DEVICE_ATTR_MAX_DCT		
	IBV_EXP_TM_CAP_DC		

Relevant Man Pages

- mlx5dv_create_qp: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_create_qp.3.md
- mlx5dv_wr_post: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_wr_post.3.md

Example

```
/**Create DC QP**/
struct mlx5dv_qp_init_attr dv_init_attr;
struct ibv_qp_init_attr_ex init_attr;


memset(&dv_init_attr, 0, sizeof(dv_init_attr));
memset(&init_attr, 0, sizeof(init_attr));


init_attr.qp_type = IBV_QPT_DRIVER;
init_attr.send_cq = send_cq;
init_attr.recv_cq = recv_cq;
init_attr.pd = pd;


if (initiator) {                                                    /** DCI **/
                init_attr.comp_mask |= IBV_QP_INIT_ATTR_SEND_OPS_FLAGS | IBV_QP_INIT_ATTR_PD;
                init_attr.send_ops_flags |= IBV_QP_EX_WITH_SEND;


                dv_init_attr.comp_mask |=
                                        MLX5DV_QP_INIT_ATTR_MASK_DC |
                                        MLX5DV_QP_INIT_ATTR_MASK_QP_CREATE_FLAGS;
                dv_init_attr.create_flags |=
                                        MLX5DV_QP_CREATE_DISABLE_SCATTER_TO_CQE;
                dv_init_attr.dc_init_attr.dc_type = MLX5DV_DCTYPE_DCI;
} else {                                                                /** DCT **/
                init_attr.comp_mask |= IBV_QP_INIT_ATTR_PD;
                init_attr.srq = srq;
                dv_init_attr.comp_mask = MLX5DV_QP_INIT_ATTR_MASK_DC;
                dv_init_attr.dc_init_attr.dc_type = MLX5DV_DCTYPE_DCT;
                dv_init_attr.dc_init_attr.dct_access_key = DC_KEY;
}
```

```
qp = mlx5dv_create_qp(context, &init_attr, &dv_init_attr);


if (initiator) {
                ex_qp = ibv_qp_to_qp_ex(qp);
                dv_qp = mlx5dv_qp_ex_from_ibv_qp_ex(ex_qp);
}
```

```
/**DCI post send**/
struct ibv_ah_attr ah_attr;
ah_attr.dlid = rem_dest->lid;
ah_attr.port_num = ib_port;

ah = ibv_create_ah(pd, &ah_attr);
if (ah) {
    return -1;
}

ibv_wr_start(ex_qp);
ex_qp->wr_id = SEND_WRID;
ex_qp->wr_flags = IBV_SEND_SIGNALED;
ibv_wr_send(ex_qp);
mlx5dv_wr_set_dc_addr(dv_qp, ah, rem_dest->dctn, DC_KEY);
ibv_wr_set_sge(ex_qp, mr->lkey, (uint64_t)mr->addr, size);
ibv_wr_complete(ex_qp);
return 0;
```

DC RDMA Atomic

DC RDMA Atomic Experimental Verbs

- ibv_exp_query_device
    - max_dc_req_rd_atom
    - max_dc_res_rd_atom

DC RDMA Atomic RDMA-Core Support

> This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Dynamically Connected On-Demand-Paging (DC ODP)

DC ODP Experimental Query Capabilities

- ibv_exp_query_device
    - dc_odp_caps

DC ODP RDMA-Core Query Capabilities

- mlx5dv_query_device
    - comp_mask: MLX5DV_CONTEXT_MASK_DC_ODP_CAPS
    - dc_odp_caps with enum ibv_odp_transport_cap_bits

Enabling ODP for DC is done, as in all other transports, by setting the IBV_ACCESS_ON_DEMAND flag in ibv_reg_mr access parameter.

Example

```
struct mlx5dv_context attr_out = {};
attr_out.comp_mask |= MLX5DV_CONTEXT_MASK_DC_ODP_CAPS;
mlx5dv_query_device(context, &attr_out);
if (attr_out.dc_odp_caps & IBV_ODP_SUPPORT_SEND)
    printf("support DC ODP send operation");
```

Key Violations

Key Violations Experimental Verbs

- ibv_exp_arm_dct
- ibv_exp_poll_dc_info

Key Violations RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Flow Entropy

Flow Entropy Experimental Verbs

- ibv_exp_modify_qp
    - flow_entropy

Flow Entropy Experimental Capabilities

IBV_EXP_QP_ATTR_FLOW_ENTROPY

Flow Entropy RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Flow Steering

Flow steering is a model which steers network flows based on flow specifications to specific QPs. Those flows can be either unicast or multicast network flows. In order to maintain flexibility, domains and priorities are used. Flow steering uses a methodology of flow attribute, which is a combination of L@-L4 flow specifications, a destination QP and a priority. Flow steering rules may be inserted either by using ethtool or by using InfiniBand verbs. The verbs abstraction uses a

different terminology from the flow attribute (ibv_flow_attr), defined by a combination of specifications (struct ibv_flow_spec_*).

Flow Steering Experimental vs. RDMA-Core Verbs and Capabilities

Experimental Verbs		RDMA-Core	
Verbs			
ibv_exp_create_flow		**ibv_create_flow**	
	IBV_EXP_FLOW_ATTR_*		IBV_FLOW_ATTR
	IBV_EXP_FLOW_ATTR_FLAGS_ALLOW_LOOP_BACK		IBV_FLOW_ATTR_FLAGS_ALLOW_LOOP_BACK
	IBV_EXP_FLOW_SPEC_ETH		IBV_FLOW_SPEC_ETH
	IBV_EXP_FLOW_SPEC_IB		Not supported by RDMA-CORE
	IBV_EXP_FLOW_SPEC_IPV4		IBV_FLOW_SPEC_IPV4
	IBV_EXP_FLOW_SPEC_IPV6		IBV_FLOW_SPEC_IPV6
	IBV_EXP_FLOW_SPEC_IPV4_EXT		IBV_FLOW_SPEC_IPV4_EXT
	IBV_EXP_FLOW_SPEC_IPV6_EXT		IBV_FLOW_SPEC_IPV6
	IBV_EXP_FLOW_SPEC_TCP		IBV_EXP_FLOW_SPEC_TCP
	IBV_EXP_FLOW_SPEC_UDP		IBV_FLOW_SPEC_UDP
	IBV_EXP_FLOW_SPEC_VXLAN_TUNNEL		IBV_FLOW_SPEC_VXLAN_TUNNEL
	IBV_EXP_FLOW_SPEC_INNER		IBV_FLOW_SPEC_INNER
	IBV_EXP_FLOW_SPEC_ACTION_TAG		IBV_FLOW_SPEC_ACTION_TAG
	IBV_EXP_FLOW_SPEC_ACTION_DROP		IBV_FLOW_SPEC_ACTION_DROP
	Ibv_exp_flow_spec_*		Ibv_flow_spec_*
	Ibv_exp_destroy_flow		Ibv_destroy_flow
Capabilities			
IBV_EXP_DEVICE_MANAGED_FLOW_STEERING		IBV_DEVICE_MANAGED_FLOW_STEERING	

Relevant Man Pages

ibv_create_flow: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_create_flow.3

Example

https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_create_flow.3

Hardware Offload for Erasure Code

HW Offload for Erasure Code Experimental Verbs

- ibv_exp_alloc_ec_calc
- ibv_exp_dealloc_ec_calc
- ibv_exp_ec_encode_async
- ibv_exp_ec_encode_sync
- ibv_exp_ec_decode_async
- ibv_exp_ec_decode_sync
- intibv_exp_ec_update_async
- intibv_exp_ec_update_sync
- ibv_exp_ec_poll
- ibv_exp_ec_encode_send

HW Offload for Erasure Code Experimental Capabilities and Device Attributes

- IBV_EXP_DEVICE_EC_OFFLOAD
- IBV_EXP_DEVICE_ATTR_EC_CAPS
- IBV_EXP_DEVICE_ATTR_EC_GF_BASE

HW Offload for Erasure Code RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Inline Receive

When Inline-Receive is active, the HCA may write received data in to the receive WQE or CQE. Using Inline-Receive saves PCIe read transaction since the HCA does not need to read the scatter list, therefore it improves performance in case of short receive -messages. On poll CQ, the driver copies the received data from WQE/CQE to the user's buffers. Therefore, apart from querying Inline-Receive capability and Inline-Receive activation the feature is transparent to user application.

Inline Receive Experimental Verbs

- ibv_exp_query_device
    - inline_recv_size
- ibv_exp_create_qp
    - max_inl_recv
- ibv_exp_create_dct
    - inline_size

Inline Receive Experimental Device Attributes

IBV_EXP_DEVICE_ATTR_INLINE_RECV_SZ

Inline Receive RDMA-Core Verbs

- mlx5dv_create_qp
    - MLX5DV_QP_CREATE_ALLOW_SCATTER_TO_CQE

Inline Receive RDMA-Core Environmental Variables

MLX5_SCATTER_TO_CQE

Relevant Man Pages

mlx5dv_create_qp: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_create_qp.3.md

Memory Window (MW)

Memory Window allows the application to have more flexible control over remote access to its memory. It is available only on physical functions or native machines. The two types of Memory Windows supported are: type 1 and type 2B.

Memory Window is intended for situations where the application wants to:

- Grant and revoke remote access rights to a registered region in a dynamic fashion with less of a performance penalty.
- Grant different remote access rights to different remote agents and/or grant those rights over different ranges within registered region.

MW Experimental vs. RDMA-Core Verbs, Variables and Capabilities

Experimental Verbs	RDMA-Core
Verbs	

ibv_exp_bind_mw		**ibv_bind_mw**	
ibv_exp_post_send	bind_mw		ibv_wr_bind_mw
ibv_exp_reg_mr	IBV_EXP_ACCESS_MW_BIND	**ibv_reg_mr**	IBV_ACCESS_MW_BIND
Capabilities and Device Attributes			
IBV_EXP_DEVICE_MEM_WINDOW		IBV_DEVICE_MEM_WINDOW	
IBV_EXP_DEVICE_MW_TYPE_2A		IBV_DEVICE_MEM_WINDOW_TYPE_2A	
IBV_EXP_DEVICE_MW_TYPE_2B		IBV_DEVICE_MEM_WINDOW_TYPE_2B	
		IBV_DEVICE_MEM_MGT_EXTENSIONS	
Environmental Variables			
MLX5_SHUT_UP_MW		In RDMA-Core, it is possible to not use MW feature with the new post send API (ibv_posr_wr)	

Relevant Man Pages

- ibv_wr_post: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_wr_post.3.md
- ibv_bind_mw: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_bind_mw.3

Multi-Packet RQ

Multi-Packet RQ is a receive queue where multiple packets are written to the same WR data buffer. The feature improves performance and reduces memory footprint.

Multi-Packet RQ Experimental vs. RDMA-Core Verbs and Capabilities

Experimental		RDMA-Core
Verbs		
ibv_exp_query_device	supported_qps: • IBV_EXP_MP_RQ_SUP_TYPE_SRQ_TM • IBV_EXP_MP_RQ_SUP_TYPE_WQ_RQ	Not supported by RDMA-Core.

	allowed_shifts: • IBV_EXP_MP_RQ_NO_SHIFT • IBV_EXP_MP_RQ_2BYTES_SHIFT		
ibv_exp_query_device • mp_rq_caps	min_single_wqe_log_num_of_strides	**mlx5dv_query_device** • mlx5dv_striding_rq_caps	min_single_wqe_log_num_of_strides
	max_single_wqe_log_num_of_strides		max_single_wqe_log_num_of_strides
	min_single_stride_log_num_of_bytes		min_single_stride_log_num_of_bytes
	max_single_stride_log_num_of_bytes		max_single_stride_log_num_of_bytes
ibv_exp_create_wq	mp_rq	**mlx5dv_create_wq**	• striding_rq_attrs • MLX5DV_WQ_INIT_ATTR_MASK_STRIDING_RQ

ibv_exp_poll_cq	• mp_wr
	• exp_wc_flags
	  • IBV_EXP_WC_MP_WR_MORE_IN_MSG
	  • IBV_EXP_WC_MP_WR_CONSUMED
	  • IBV_EXP_WC_RECV_NOP
		Not supported by RDMA-Core.	

Capabilities and Device Attributes			
IBV_EXP_DEVICE_ATTR_MP_RQ		Not supported by RDMA-Core.	
IBV_EXP_MP_RQ_SUP_TYPE_SRQ_TM			

No Operation (NOP)

NOP feature is used for cache optimization.

NOP Experimental Verbs

- ibv_exp_post_send
  - IBV_EXP_WR_NOP

NOP Experimental Capabilities

IBV_EXP_DEVICE_NOP

NOP RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further support, please contact Mellanox Support.

Out-of-Order (OOO)

In certain fabric configurations, InfiniBand packets for a given QP may take up different paths in a network from source to destination. This results into packets being received in an out-of-order manner. These packets can now be handled instead of being dropped, in order to avoid retransmission. Data will be placed into host memory in an out-of-order manner when out of order messages are received.

OOO Experimental Verbs

- ibv_exp_create_dct
    - IBV_EXP_DCT_OOO_RW_DATA_PLACEMENT

OOO Environmental Variables

MLX5_RELAXED_PACKET_ORDERING_ON

OOO Device Attributes

IBV_EXP_DEVICE_ATTR_OOO_CAPS

OOO Rdma-Core Support

OOO feature is enabled by default in RDMA-Core.

PeerDirect Sync

PeerDirect sync allows for attaching context to a peer client. The verb receives an attribute in addition to the ibv device, then it calls the original open device to create a context.

PeerDirect Sync Experimental Verbs

- ibv_exp_open_device
    - peer_info

RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

PeerDirect™ Async

Mellanox PeerDirect Async sub-system gives peerDirect hardware devices, such as GPU cards, dedicated AS accelerators, and so on, the ability to take control over HCA in critical path offloading CPU. To achieve this, there is a set of verb calls and structures providing application with abstract description of operation sequences intended to be executed by peer device.

PeerDirect™ Async Experimental Verbs

- ibv_exp_create_cq
  - ibv_exp_peer_direct_attr
- ibv_exp_create_qp
  - ibv_exp_peer_direct_attr
- ibv_exp_peer_commit_qp
- ibv_exp_rollback_qp
- ibv_exp_peer_peek_cq
- ibv_exp_peer_abort_peek_cq
- Any verb in the header file peer_ops.h

RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Physical Address Memory Allocation

Physical address memory region (PA_MR) allows for managing physical memory used for posting Send and Receive requests. This can benefit the performance of applications that register large memory regions with random access.

PA_MR Experimental Verbs

- Ibv_exp_reg_mr
  - IBV_EXP_ACCESS_PHYSICAL_ADDR

PA_MR Device Capabilities

- IBV_EXP_DEVICE_PHYSICAL_RANGE_MR

PA_MR RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

Prefetch Memory Region

Prefetch Memory Region (MR) enables prefetch pages of an On-Demand Paging (ODP) memory region.

Prefetch Memory Region Experimental vs. RDMA-Core Verbs

Experimental Verbs	RDMA-Core Verbs
ibv_exp_prefetch_mr • IBV_EXP_PREFETCH_WRITE_ACCESS	**ibv_advise_mr** • IBV_ADVISE_MR_ADVICE_PREFETCH • IBV_ADVISE_MR_ADVICE_PREFETCH_WRITE

Relevant Man Page

ibv_advise_mr: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_advise_mr.3.md

Example

See example in libibverbs/examples/rc_pingpong.c

Query GID Attributes

Query GID attributes, such as GID type.

Query GID Attributes Experimental Verbs

- ibv_exp_query_gid_attr

Query GID Attributes RDMA-Core Support

GID type can be extracted from the ibsysfs.

Example

Review ibv_query_gid_type, ibv_read_sysfs_file from rdma-core:
https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/verbs.c

Raw Ethernet

Raw Ethernet programming enables writing an application that bypasses the kernel stack. To achieve this, packet headers and offload options need to be provided by the application.

Raw Ethernet Experimental vs. RDMA-Core Verbs and Capabilities

Experimental		RDMA-Core	
Verbs			
ibv_exp_create_qp		ibv_create_qp / ibv_create_qp_ex	
	IBV_QPT_RAW_ETH		IBV_QPT_RAW_PACKET
Capabilities			
	IBV_EXP_DEVICE_WQ_DELAY_DROP		IBV_RAW_PACKET_CAP_DELAY_DROP

Relevant Man Pages

ibv_create_qp_ex: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_create_qp_ex.3

Registration and Re-registration of Memory Region (MR)

Re-registration MR allows the user to change attributes of the memory region. The user may change the PD, access flag or the address and length of the memory region.

MR Registration/Re-Registration Experimental vs. RDMA-Core Verbs

Experimental Verbs		RDMA-Core Verbs	
ibv_exp_reg_mr		ibv_reg_mr	

Experimental Verbs		RDMA-Core Verbs	
	IBV_EXP_ACCESS_*		IBV_ACCESS_*
	IBV_EXP_ACCESS_ALLOCATE_MR		Please refer to the Contiguous Pages section
	IBV_EXP_ACCESS_SHARED_MR_* IBV_EXP_ACCESS_NO_RDMA		Shared MR functionality is deprecated in OFED and not supported in RDMA-Core.
	IBV_EXP_ACCESS_RELAXED		*ODP relaxed* Replaced with *implicit ODP*
	IBV_EXP_ACCESS_PHYSICAL_ADDR		Please refer to the Physical Address Memory Allocation section.
	IBV_EXP_ACCESS_TUNNELED_ATOMIC		Not supported in RDMA-Core
	IBV_EXP_REG_MR_DM	**ibv_reg_dm_mr**	
ibv_exp_rereg_mr		**ibv_rereg_mr**	
	IBV_EXP_REREG_MR_CHANGE_TRANSLATION		IBV_REREG_MR_CHANGE_TRANSLATION
	IBV_EXP_REREG_MR_CHANGE_PD		IBV_REREG_MR_CHANGE_PD
	IBV_EXP_REREG_MR_CHANGE_ACCESS		IBV_REREG_MR_CHANGE_ACCESS
	IBV_EXP_ACCESS_*		IBV_ACCESS_*

Relevant Man Pages

- ibv_reg_mr: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_reg_mr.3
- ibv_rereg_mr: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_rereg_mr.3.md

Example

See example in libibverbs/examples/rc_pingpong.c

Resource Domain

Resource domain is a verb object that may be associated with a QP and CQ objects upon creation to enhance data-path performance.

Resource Domain Experimental Verbs

- Ibv_exp_create_res_domain
- Ibv_exp_destroy_res_domain
- Ibv_exp_create_qp
    - ibv_exp_res_domain
- ibv_exp_query_device
    - max_ctx_res_domain
- ibv_exp_create_cq
    - ibv_exp_res_domain
- ibv_exp_create_wq
    - ibv_exp_res_domain

Resource Domain Experimental Capabilities

- IBV_EXP_DEVICE_ATTR_MAX_CTX_RES_DOMAIN

Resource Domain RDMA-Core Verbs

- ibv_alloc_td
- ibv_alloc_parent_domain
- ibv_dealloc_pd

Relevant Man Pages

- ibv_alloc_parent_domain: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_alloc_parent_domain.3l
- ibv_alloc_td: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_alloc_td.3

RoCE Time-Stamping

RoCE Time-Stamping allows you to stamp packets when they are sent to the wire or when they are received from the wire. The time stamp is given in raw hardware cycles, but can easily be converted into hardware-referenced nanoseconds-based-time. Additionally, it enables you to query the hardware for the hardware time, thus stamp other application's event and compare time.

RoCE Time-Stamping Experimental vs. RDMA-Core Verbs and Device Attributes

Experimental		RDMA-Core	
Verbs			
ibv_exp_query_values		ibv_query_rt_values_ex	

	IBV_WXP_VALUES_HW_CLOCK_NS		ibv_query_rt_values_ex
	IBV_EXP_VALUES_HW_CLOCK	**ibv_query_rt_values_ex**	IBV_VALUES_MASK_RAW_CLOCK
	IBV_EXP_VALUES_CLOCK_INFO	**mlx5dv_get_clock_info**	
Ibv_exp_cqe_ts_to_ns		**mlx5dv_ts_to_ns**	
Ibv_exp_create_cq		**ibv_create_cq_ex**	
	IBV_EXP_CQ_TIMESTAMP		IBV_WC_EX_WITH_COMPLETION_TIMESTAM
	IBV_EXP_CQ_TIMESTAMP_TO_SYS_TIME		IBV_WC_EX_WITH_COMPLETION_TIMESTAMP_WALLCLOCK
Ibv_exp_poll_cq		**ibv_start_poll**	
	• Timestamp • IBV_EXP_WC_WITH_TIMESTAMP		• ibv_wc_read_completion_ts • ibv_wc_read_completion_wallclock_ns
Ibv_exp_query_device		**ibv_query_device_ex**	
	timestamp_mask		completion_timestamp_mask
	hca_core_clock		hca_core_clock
Device Attributes			
IBV_EXP_DEVICE_ATTR_WITH_TIMESTAMP_MASK		Device attributes are not needed in RDMA-Core.	
IBV_EXP_DEVICE_ATTR_WITH_HCA_CORE_CLOC			

Relevant Man Pages

- ibv_query_rt_values_ex: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_query_rt_values_ex.3
- mlx5dv_get_clock_info: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_get_clock_info.3
- mlx5dv_ts_to_ns: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_ts_to_ns.3
- ibv_create_cq_ex: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_create_cq_ex.3

Example

See example in libibverbs/examples/rc_pingpong.c

Shared Memory Region

This feature helps share a memory region (MR) between processes.

Shared MR Experimental Verbs

- ibv_exp_reg_shared_mr
- ibv_exp_reg_mr
  - IBV_EXP_ACCESS_SHARED_MR_*
  - IBV_EXP_ACCESS_NO_RDMA

Shared MR Experimental Capabilities

- IBV_EXP_DEVICE_SHARED_MR

Shared MR RDMA-Core Support

> ⚠ This feature is deprecated in MLNX_OFED driver and is not supported in RDMA-Core.

Tag Matching

Tag Matching and Rendezvous Offload is a technology employed by Mellanox to offload the processing of MPI messages from the host machine onto the network card. Employing this technology enables a zero copy of MPI messages, i.e. messages are scattered directly to the user's buffer without intermediate buffering and copies. It also provides a complete rendezvous progress by Mellanox devices. Such overlap capability enables the CPU to perform the application's computational tasks while the remote data is gathered by the adapter.

Tag Matching Experimental Verbs

ibv_exp_create_dct

- IB_EXP_SRQT_TAG_MATCHING
- MLX5_DCTC_OFFLOAD_TYPE_RNDV

Tag Matching Experimental Capabilities

- IBV_EXP_DEVICE_ATTR_TM_CAPS
- IBV_EXP_TM_CAP_DC

Tag Matching RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

TCP Segmentation Offload (TSO)

TCP Segmentation Offload (TSO) enables the adapter cards to accept a large amount of data with a size greater than the MTU size. The TSO engine splits the data into separate packets and inserts the user-specified L2/L3/L4 headers automatically per packet. With the usage of TSO, CPU is offloaded from dealing with a large throughput of data.

TSO Experimental vs. RDMA-Core Verbs and Capabilities

Experimental Verbs		RDMA-Core	
Verbs			
ibv_exp_create_qp		**ibv_create_qp_ex**	
	max_tso_header		max_tso_header
ibv_exp_query_device		**ibv_query_device_ex**	
	tso_caps		tso_caps
ibv_exp_post_send	tso	• **ibv_post_send** • **ibv_wr_send_tso**	tso
Capabilities and Device Attributes			
IBV_EXP_DEVICE_ATTR_TSO_CAPS			

Relevant Man Pages

- ibv_create_qp_ex: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_create_qp_ex.3
- ibv_query_device_ex: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_query_device_ex.3

Tunneled Atomic

Tunneled Atomic updates RDMA Write Operations so that when an RDMA Write operation is issued, the payload indicates which atomic operation to perform, instead of being written to the memory region (MR).

Tunneled Atomic Experimental Verbs

- ibv_exp_reg_mr
    - IBV_EXP_ACCESS_TUNNELED_ATOMIC
- ibv_exp_query_device

- tunneled_atomic_caps

Tunneled Atomic Experimental Capabilities

IBV_EXP_DEVICE_ATTR_TUNNELED_ATOMIC

Tunneled Atomic RDMA-Core Support

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

User-Mode Memory Registration (UMR)

UMR is a fast registration mode which uses Send queues. This feature enables the usage of RDMA operations and scatters data through appropriate memory keys on the remote side.

UMR Experimental Verbs

The following UMR and Non-Inline UMR experimental verbs are no longer the default verbs used for configuration of neither feature.

UMR Verbs:

- Ibv_exp_create_qp
  - exp_create_flags: IBV_EXP_QP_CREATE_UMR
  - max_inl_send_klms
- ibv_exp_query_device
  - umr_cap
  - umr_fixed_size_caps
- ibv_exp_post_send
  - exp_opcode: IBV_EXP_WR_UMR_FILL, IBV_EXP_WR_UMR_INVALIDATE
  - ext_op: umr (umr type, memory_objects, exp_access, modified_mr, base_addr, num_mrs, mem_reg_list, mem_repeat_block_list, repeat_count, stride_dim)
- ibv_exp_poll_cq
  - exp_opcode: IBV_EXP_WC_UMR
- ibv_exp_create_mr
- ibv_exp_query_mkey

Non-Inline UMR Verbs:

- environmental variable: MLX*_POST_SEND_PREFER_BF
- ibv_exp_dealloc_mkey_list_memory
- ibv_exp_alloc_mkey_list_memory

UMR Experimental Capabilities and Device Attributes

- IBV_EXP_DEVICE_UMR, IBV_EXP_DEVICE_UMR_FIXED_SIZE, IBV_EXP_DEVICE_MR_ALLOCATE
- IBV_EXP_DEVICE_ATTR_UMR , IBV_EXP_DEVICE_ATTR_UMR_FIXED_SIZE_CAPS

UMR RDMA-Core Verbs

UMR Verbs:

- Ibv_create_qp_ex
  - send_ops_flags: IBV_QP_EX_WITH_BIND_MW, IBV_QP_EX_WITH_LOCAL_INV
- mlx5dv_create_qp
  - send_ops_flags: MLX5DV_QP_EX_WITH_MR_INTERLEAVED, MLX5DV_QP_EX_WITH_MR_LIST
- mlx5dv_wr_post
  - mlx5dv_wr_mr_interleaved
  - mlx5dv_wr_mr_list
- mlx5dv_wc_opcode
  - MLX5DV_WC_UMR

Non-Inline UMR Verbs:

> ⚠ This feature is currently not supported by RDMA-Core.
> For further information, please contact Mellanox Support.

UMR RDMA-Core Capabilities and Device Attributes

No UMR capabilities flags are needed in RDMA-Core.

Relevant Man Pages

- mlx5dv_create_qp: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_create_cq.3.md
- mlx5dv_wr_post: https://github.com/linux-rdma/rdma-core/blob/master/providers/mlx5/man/mlx5dv_wr_post.3.md

Example

```
struct mlx5dv_qp_init_attr mlx5_qp_attr = {};
struct ibv_qp_init_attr_ex init_attr_ex = {};
struct ibv_qp     *qp;
struct ibv_qp_ex *qpx;
struct mlx5dv_qp_ex *dv_qp;
struct mlx5dv_mkey *dv_mkey;
struct mlx5dv_mkey_init_attr mkey_init_attr = {};
struct mlx5dv_mr_interleaved *array_interleaved;
int num_interleaved = 1;
int repeat_count = 2;
int message_size = 4096;
int skip_bytes_interleaved = 20;

mlx5_qp_attr.comp_mask = MLX5DV_QP_INIT_ATTR_MASK_SEND_OPS_FLAGS;
mlx5_qp_attr.send_ops_flags = MLX5DV_QP_EX_WITH_MR_INTERLEAVED;
init_attr_ex.cap.max_inline_data = 128;
init_attr_ex.send_ops_flags = IBV_QP_EX_WITH_SEND;

qp = mlx5dv_create_qp(context, &init_attr_ex, &mlx5_qp_attr);
qpx = ibv_qp_to_qp_ex(qp);
dv_qp = mlx5dv_qp_ex_from_ibv_qp_ex(qpx);
mkey_init_attr.create_flags = MLX5DV_MKEY_INIT_ATTR_FLAGS_INDIRECT;
mkey_init_attr.max_entries = 4;
mkey_init_attr.pd = pd;

dv_mkey = mlx5dv_create_mkey(&mkey_init_attr);
array_interleaved = calloc(1, num_interleaved * sizeof(struct mlx5dv_mr_interleaved));
```

```
qpx->wr_flags = IBV_SEND_INLINE | IBV_SEND_SIGNALED;
array_interleaved[0].addr = (uintptr_t)mr->addr;
array_interleaved[0].bytes_count = (message_size - skip_bytes_interleaved) / 2;
array_interleaved[0].bytes_skip = skip_bytes_interleaved / 2;
array_interleaved[0].lkey
= mr->lkey;
ibv_wr_start(qpx);
mlx5dv_wr_mr_interleaved (dv_qp, dv_mkey, access_flags,
                          repeat_count, num_interleaved, array_interleaved);
ret = ibv_wr_complete(ctx->qpx);
```

VLAN Offload (VLAN Insertion/Stripping)

The device offloads VLAN insertion and stripping for raw Ethernet frames.

VLAN insertion is performed by the driver, inlining the VLAN tag into the Ethernet frame headers in the WQE Eth Segment.

VLAN Stripping is configured in RQ through the VLAN Stripping Disable (vsd) bit. When configured to perform VLAN Stripping, the device removes the VLAN tag from the incoming frames and reports it in the CQE fields.

VLAN offload Experimental vs. RDMA-Core Verbs and Capabilities

	Experimental		RDMA-Core
Verbs			
ibv_exp_query_qp	wq_vlan_offloads_cap		Check caps through ibv_query_device_ex
ibv_exp_create_wq		**ibv_create_wq**	
	vlan_offloads • IBV_EXP_RECEIVE_WQ_CVLAN_STRIP • IBV_EXP_RECEIVE_WQ_CVLAN_INSERTION		• IBV_WQ_FLAGS_CVLAN_STRIPPING • VLAN insertion not supported by Rdma core
ibv_exp_modify_wq	vlan_offloads	**ibv_modify_wq**	flags IBV_WQ_FLAGS_CVLAN_STRIPPING
Capabilities and Device Attributes			
	IBV_EXP_DEVICE_ATTR_VLAN_OFFLOADS		IBV_RAW_PACKET_CAP_CVLAN_STRIPPING

Relevant Man Pages

- ibv_create_wq: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_create_wq.3
- ibv_modify_wq: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_modify_wq.3
- ibv_query_device_ex: https://github.com/linux-rdma/rdma-core/blob/master/libibverbs/man/ibv_query_device_ex.3

Additional Capabilities in MLNX_OFED

The following table lists additional experimental and RDMA-Core capabilities in MLNX_OFED.

Experimental	RDMA-Core
IBV_EXP_DEVICE_CAPI	NOT supported by RDMA-Core
IBV_EXP_DEVICE_VXLAN_SUPPORT	NOT supported by RDMA-Core
IBV_EXP_DEVICE_ATTR_PACKET_PACING_CAPS	ibv_packet_pacing_caps, ibv_modify_qp_rate_limit
IBV_EXP_DEVICE_ATTR_TUNNEL_OFFLOADS_CAPS	MLX5DV_CONTEXT_MASK_TUNNEL_OFFLOADS
IBV_EXP_DEVICE_ATTR_PCI_ATOMIC_CAPS	NOT supported by RDMA-Core
IBV_EXP_DEVICE_ATTR_RX_PAD_END_ALIGN	IBV_DEVICE_PCI_WRITE_END_PADDING

Applications Support in RDMA-Core

Rping Example

RTT Measurements

RTT measurements functionality in examples/rping.c is not supported by RDMA-Core. The functionality remains in the experimental rping.c example in the legacy mode using `rping -m`.

For further information, please contact Mellanox Support.