Verbs RSS Arch & API

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

Agenda

- Introduction
- Arch Overview
- RSS libibverbs API
- Application Initialization Flow Example
- What next?

Purpose

- RSS (Receive Side Scaling) technology allows to spread incoming traffic between different receive descriptor queues.
- Assigning each queue to different CPU cores allows to better load balance the incoming traffic and improve performance.
- This PPT introduces RSS arch and libibverbs API that comes to allow verbs based solutions to utilize the RSS offload capability, widely supported today by many modern cards.
- Notice: currently we cover only “RSS” for IBV_QPT_RAW_PACKET QP.

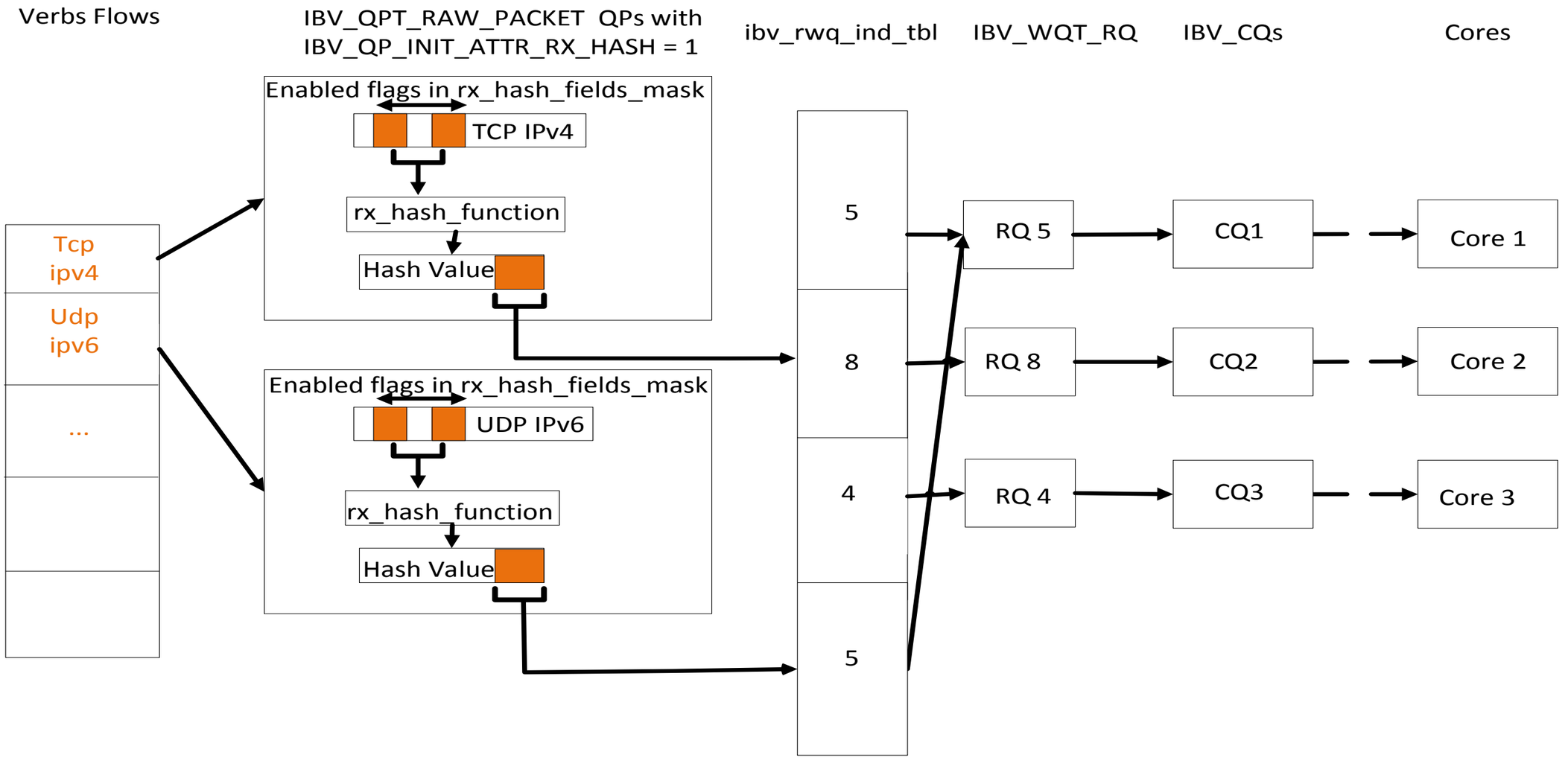
RSS Flow Overview

- Steering rules classify incoming packets and deliver a specific traffic types (e.g. TCP/UDP, IP only) to dedicated QP.
- QPs are responsible to spread the traffic they handle between RQs using RX hash and Indirection Table
- RQ can point to different CQs that can be associated with different cores
- This allows to workload the traffic between different cores
- In addition different traffic types can be subject to different spreading modes

RSS Flow Diagram

Verbs Steering Classifies the traffic

IBV_QPT_RAW_PACKET QPs distributes traffic type between RQs/Cores



WQ

```
/*
 * Work Queue. QP can be created without internal WQs "packaged" inside it,
 * this QPs can be configured to use "external" WQ object as its
 * receive/send queue.
 * WQ associated (many to one) with Completion Queue it owns WQ properties
 * (PD, WQ size etc).
 * WQ of type IBV_RQ contains receive WQEs
 * WQ of type IBV_SRQ is associated (many to one) with IB_SRQT_BASIC SRQ,
 * in which case it does not hold receive WQEs.
 * QPs can be associated with IBV_S/RQ WQs via WQ Indirection Table
 * (many to many).
 */
struct ibv_wq {
        struct ibv_context      *context;
        void                    *wq_context;
        uint32_t                 handle;
        struct      ibv_pd      *pd;
        struct      ibv_cq      *cq;
        /* SRQ handle if WQ is to be associated with an SRQ, otherwise NULL */
        struct      ibv_srq     *srq;
        uint32_t                 wq_num;
        enum ibv_wq_state        state;
        enum ibv_wq_type         wq_type;
};
```

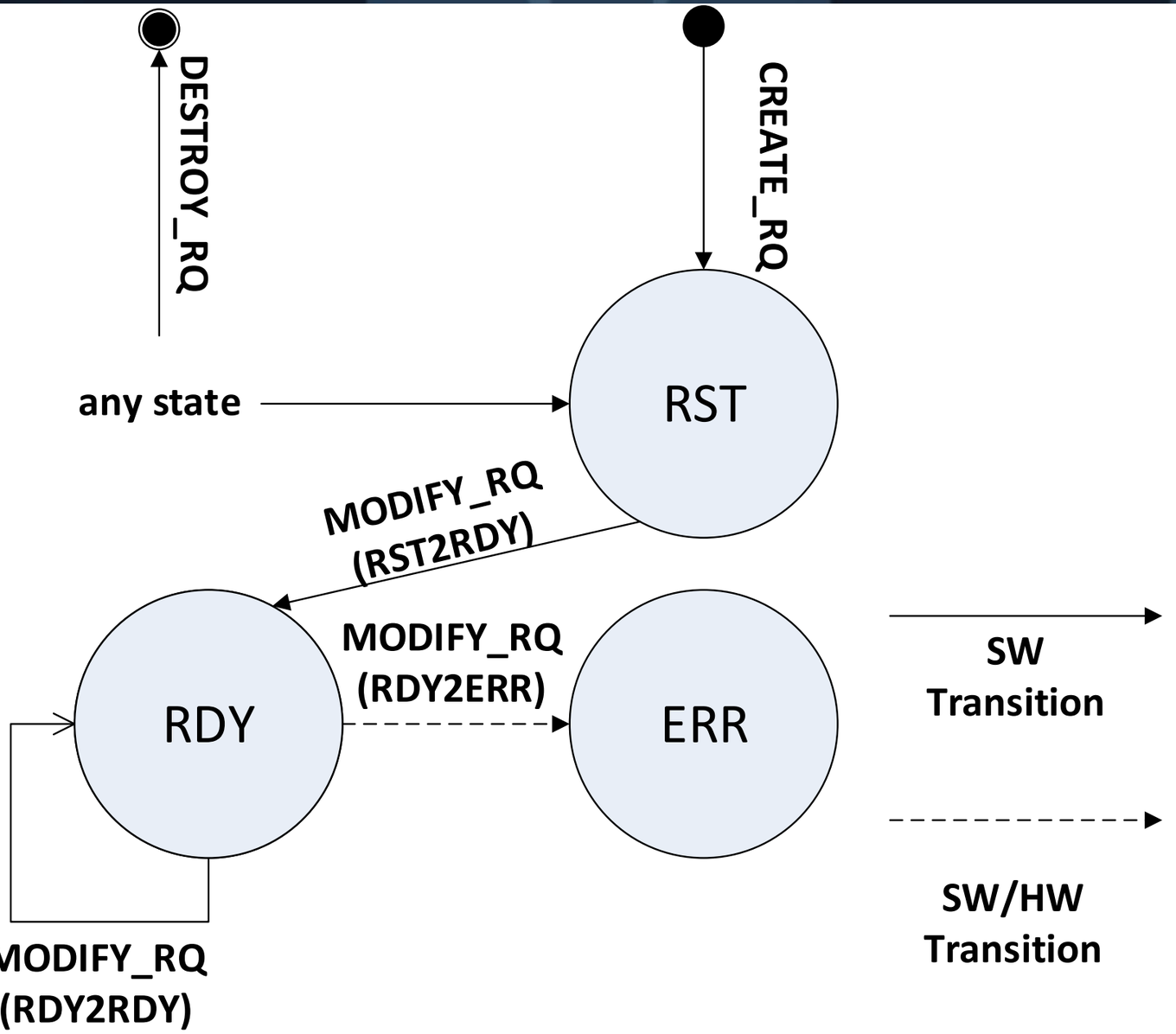
WQ Attributes

```
enum ibv_wq_type {
        IBV_WQT_RQ,
        IBV_WQT_SRQ
};

struct ibv_wq_init_attr {
        void                    *wq_context;
        enum ibv_wq_type         wq_type;
        uint32_t                 max_wr;  /* Valid for non IBV_WQT_SRQ WQ */
        uint32_t                 max_sge; /* Valid for non IBV_WQT_SRQ WQ */
        struct    ibv_pd        *pd;
        struct    ibv_cq        *cq;
        struct    ibv_srq       *srq;
};

struct ibv_wq_attr {
        uint32_t                 attr_mask;
        enum    ibv_wq_state     wq_state; /* Move the RQ to this state */
        enum    ibv_wq_state     curr_wq_state; /* Assume this is the current RQ state */
};
```

WQ Routines

- struct ibv_wq *ibv_create_wq(struct ibv_context *context,
  struct ibv_wq_init_attr *wq_init_attr);
- int ibv_modify_wq(struct ibv_wq *wq, struct ibv_wq_attr *wq_attr);
- int ibv_query_wq(struct ibv_wq *wq, struct ibv_wq_attr *wq_attr);
- int ibv_destroy_wq(struct ibv_wq *wq);
- static inline int ibv_post_wq_recv(struct ibv_wq *wq,
  struct ibv_recv_wr *recv_wr,
  struct ibv_recv_wr **bad_recv_wr);

IBV_WQT_RQ States Diagram



RQ Indirection Table

```
/*
 * Receive Work Queue Indirection Table.
 * QPs with IBV_QP_INIT_ATTR_RX_HASH flag enabled use Indirection Table
 * in order to distribute incoming packets between different
 * Receive Work Queues. Associating Receive WQs with different CPU cores
 * allows to workload the traffic between different CPU cores.
 * The Indirection Table can contain only WQs of type IBV_RQ/IBV_SRQ.
 * Notice: Multiple QP can point to the same Indirection Table.
*/
struct ibv_rwq_ind_table {
        struct ibv_context     *context;
        struct ibv_pd        *pd;
        int                     ind_tbl_num;
        uint32_t                comp_mask;
};
```

Indirection Table Attributes

```
/*
 * Receive Work Queue Indirection Table attributes
*/
struct ibv_rwq_ind_table_init_attr {
        struct ibv_pd        *pd;
        uint32_t              log_rwq_ind_tbl_size;
        struct ibv_wq      **rwq_ind_tbl;
};

/*
 * Receive Work Queue Indirection Table attributes mask
*/
enum ibv_rwq_ind_table_attr_mask {
        IBV_RWQ_IND_TABLE_ATTR_TABLE            = 1 << 0,
        IBV_RWQ_IND_TABLE_ATTR_TABLE_SIZE       = 1 << 1,
        IBV_RWQ_IND_TABLE_ATTR_RESERVED         = 1 << 2
};

/*
 * Receive Work Queue Indirection Table attributes
*/
struct ibv_rwq_ind_table_attr {
        uint32_t                attr_mask;
        uint32_t                log_rwq_ind_tbl_size;
        struct ibv_wq           **rwq_ind_tbl;
};
```

Indirection Table Routines

- struct ibv_wq_ind_tbl *ibv_create_rwq_ind_table(struct ibv_context *context,
  struct ibv_rwq_ind_table_init_attr*
  wq_ind_table_init_attr);
- int ibv_modify_rwq_ind_table(struct ibv_rwq_ind_table *wq_ind_table);
- int ibv_query_rwq_ind_table (struct ibv_rwq_ind_tbl*,
  struct ibv_rwq_ind_table_attr* rwq_ind_table_attr);
- int ibv_destroy_rwq_ind_table(struct ibv_rwq_ind_table *wq_ind_table);

“RSS” QP

- Add a new create flag supported for IBV_QPT_RAW_PACKET (and in the future for UD QP): IBV_QP_INIT_ATTR_RX_HASH.
- Enabling IBV_QP_INIT_ATTR_RX_HASH flag in QP has the following implications:
  - QP is Stateless
  - SWQ/RWQ size parameters in QP attributes must be set to 0 and no WQs are created inside that QP.
  - Post_recv and post_send can’t be done on that QP
  - The QP must be connected to RQ Indirection Table.
  - On packet reception the QP chooses to which RQ to deliver an incoming packet using RX hash result.
- RX hash function and packet fields to use in RX hash calculation are initialized on QP creation by updating ibv_qp_init_attr.ib_rx_hash_conf and can be modified by calling to ibv_modify_qp and updating ibv_qp_attr.ib_rx_hash_conf structure.
- Initialization Attributes: IBV_QP_INIT_ATTR_PD, IBV_QP_INIT_ATTR_RX_HASH, IBV_QP_INIT_ATTR_PORT

RX Hash Configuration

```
/*
 * RX Hash QP configuration. Sets hash function, hash types and
 * Indirection table for QPs with enabled IBV_QP_INIT_ATTR_RX_HASH flag.
*/
struct ibv_rx_hash_conf {
        /* enum ibv_rx_hash_fnction */
        uint8_t                         rx_hash_function;
        /* valid only for Toeplitz */
        uint8_t                        *rx_hash_key;
        /* enum ibv_rx_hash_fields */
        uint64_t                        rx_hash_fields_mask;
        struct ibv_rwq_ind_table       *rwq_ind_tbl;
};
/*
 * RX Hash Function.
*/
enum ibv_rx_hash_function_flags {
        IBV_EX_RX_HASH_FUNC_TOEPLTIZ            = 1 << 0,
        IBV_EX_RX_HASH_FUNC_XOR                 = 1 << 1
};
```

RX Hash Fields

```
/*
 * RX Hash flags, these flags allows to set which incoming packet field should
 * participates in RX Hash. Each flag represent certain packet's field,
 * when the flag is set the field that is represented by the flag will
 * participate in RX Hash calculation.
 * Notice: *IPV4 and *IPV6 flags can't be enabled together on the same QP
 * and *TCP and *UDP flags can't be enabled together on the same QP.
*/
enum ibv_rx_hash_fields {
        IBV_RX_HASH_SRC_IPV4            = 1 << 0,
        IBV_RX_HASH_DST_IPV4            = 1 << 1,
        IBV_RX_HASH_SRC_IPV6            = 1 << 2,
        IBV_RX_HASH_DST_IPV6            = 1 << 3,
        IBV_RX_HASH_SRC_PORT_TCP        = 1 << 4,
        IBV_RX_HASH_DST_PORT_TCP        = 1 << 5,
        IBV_RX_HASH_SRC_PORT_UDP        = 1 << 6,
        IBV_RX_HASH_DST_PORT_UDP        = 1 << 7
};
```

Verbs Application Initialization Flow Example

- N X Create CQ
- N X Create RQ
- Create and populate RQ Indirection Table with previously created RQs
- Create 2 X IB_QPT_RAW_PACKET QPs with enabled IB_QP_CREATE_RX_HASH_QP
  - Associate QPs with previously created Indirection Table
  - Configure QPs RX Hash properties:
    - QP1
      - Hash function: Toeplitz
      - Hash types: TCP source port, TCP destination port, IPv4 source address, IPv4 destination address
    - QP2
      - Hash function: Toeplitz
      - Hash types: UDP source port, UDP destination port, IPv4 source address, IPv4 destination address
- N X post receive to RQ
- Create appropriate flow rules
  - Configure Steering to deliver only TCP/IPv4 packets to “RSS” QP1
  - Configure Steering to deliver only UDP/IPv4 packets to “RSS” QP2

Plans for “RSS” UD Support

- Add support IBV_QP_INIT_ATTR_RX_HASH flag also for UD QP
  - “RSS” UD QP is connected to Indirection table of receive WQs
  - UD QP deals with the UD transport: pkey, qkey checks…
  - The wire destination QPN that appears in incoming packets is QPN of the “RSS” UD QP

Plans for TSS

- IBV_SQ WQ type
  - Send Queue.
  - This is where send WQEs are placed
  - Associated (many to one) with Completion Queue.
  - Associated (may to one) with “TSS” QP

- “TSS” QP
  - Stateless
  - SWQ/RWQ size parameters in QP attributes must be set to 0 and no WQs are created inside that QP.
  - Post_recv and post_send can’t be done on that QP
  - If this is UD QP:
    - It defines UD transport properties: pkey, qkey…
    - The wire source QPN that appears in sent packets is QPN of the “TSS” UD QP
  - The same QP can serve as both “RSS” QP and as “TSS” QP

Questions?

Thank You

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™