OPENFABRICS ALLIANCE

13th ANNUAL WORKSHOP 2017

# USER SPACE IPOIB PACKET PROCESSING

Tzahi Oved, Alex Rosenbaum

Mellanox Technologies

[ March, 2017 ]

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

# AGENDA

- Motivation for extending IPoIB user space processing
- Progress of Eth user space processing
- Status update on Eth user space processing
- IPoIB Stack
- IPoIB address resolution
- User mode IPoIB QPn Addressing
- User Verbs and “IPoIB QP”
- Verbs API extensions
- IPoIB and RSS
- IPoIB and TSS
- IPoIB TSO
- IPoIB and overlay networking
- Summary

# EXTENDING THE USER LEVEL NETWORKING API

- **Over the last year the RDMA stack has been extended to support packet processing applications and user-level TCP/IP stacks**

- **This allowed delivering of low latency and high message-rate to these applications. We'll provide an extensive introduction to both current and upcoming packet processing Verbs, such as checksum offloads, TSO, flow steering, and RSS**

- **2016 focus was on Ethernet**
- **In 2017 we want to expand to IPoIB**

# FOLLOW UP ON 2016 OFA PRESENTATION: USER MODE ETHERNET VERBS

- Presentation of features to enable higher rate user space Ethernet implementation for packet processing
  https://www.openfabrics.org/images/eventpresos/2016presentations/205EthernetVerbs.pdf

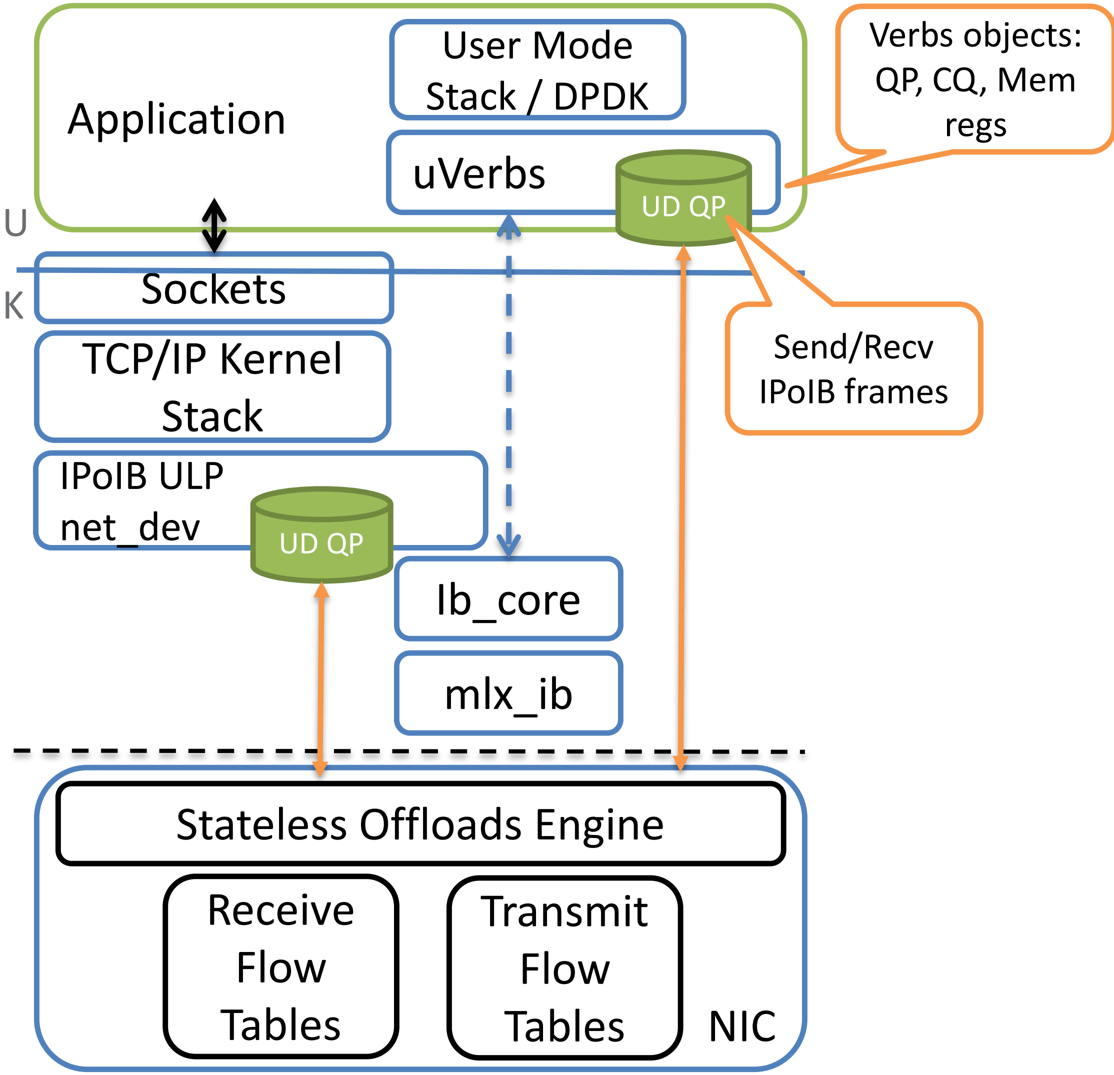
- **Status:**
  - Done: Receive Side Scaling (RSS)
    - `ibv_create_rwq_ind_table()`
    - `ibv_create_qp_ex(IBV_QP_INIT_ATTR_RX_HASH | IBV_QP_INIT_ATTR_IND_TABLE)`
  - Done: Work Queue’s
    - `ibv_create_wq(IBV_WQT_RQ)`
  - Done: TSO
    - `ibv_create_qp_ex(IBV_QP_INIT_ATTR_MAX_TSO_HEADER)`
  - Done: Tunneling (Kernel part)
    - `IB_FLOW_SPEC_INNER & IB_FLOW_SPEC_VXLAN_TUNNEL`
  - Done: Capture (Sniffer) for RDMA and Eth
    - `IBV_FLOW_ATTR_SNIFFER`
  - Done: CQ iterator
    - `ibv_start_poll(), ibv_next_poll(), ibv_end_poll()`, and many getter()’s

# USER MODE ETHERNET VERBS - NEXT

- Interrupt vector (CQ) binding to CPU core – In progress
- Interrupt Moderation (CQ Moderation) – In progress
- Extend Tunneling: to user space, VXLAN, NVGRE, GENEVE – In progress
- LRO support
- Support Non-Privileged Ethernet QP types
  - Kernel to control send headers L2/L3/L4 (vs RAW_PACKET)
  - Kernel to control receive filters a process can assign

# IPOIB VERBS – CURRENT STATUS

- **IPoIB (datagram mode) is UD QP**
- **Today user space application can:**
  - Create a UD QP
    *ibv_create_qp_ex(IBV_QPT_UD)*
  - Join multicast as full member with the SM
    *rdma_join_multicast()*
  - Receive multicast by entire MGID steering
    *ibv_attach_mcast()*
  - Send multicast and unicast with respectful AH
    *ibv_post_send()*
- **But there are many limitations… (next slide)**
- **Motivation:**
  - Run user mode TCP/IP stack, DPDK or other socket accelerations solutions over Infiniband/IPoIB



# IPOIB AND ARP

- **IPoIB Specification**
  - Defined by IETF group in RFCs 4391, 4392
  - Covers IPoIB HW addressing, IPoIB ARP and DHCP
- **IPoIB RFC Defines the following L2 Hardware address format:**

| | | |
|---|---|---|
| Reserved [7:0] | QP Number [23:0] | GID = SubnetPrefix:GUID<br>[127:0] = [63:0][63:0] |

- **IPoIB ARP packet format payload uses the above Hardware address format**

<table>
<tr><td colspan="2">HW Type (0x20)</td><td>Protocol : IPv4</td></tr>
<tr><td>HW Addr Len</td><td>Proto Addr Len</td><td>ARP Operation</td></tr>
<tr><td colspan="3">Sender HW Address</td></tr>
<tr><td colspan="3">Sender Protocol Address</td></tr>
<tr><td colspan="3">Receiver HW Address</td></tr>
<tr><td colspan="3">Receiver Protocol Address</td></tr>
</table>

- **Following address resolution process (IPoIB ARP), network stack is familiar with remote peer IPoIB QPN**

# CHALLENGES FOR IPOIB PACKET PROCESSING

- **IPoIB QPN**
  - Receive steering is preformed by match of packet->dst_QPN to my_ibv_qp->qp_num
    - Address resolution result is IPoIB QPN and not “My QPN”
  - Sends will use my_ibv_qp->qp_num as src_QPN in DETH header over the wire
  - In order to send/recv based on a different QPN (L2 IPoIB address) Verbs should support:
    - Define wire (DETH) QPN
    - Ibv_create_flow() to allow steering based on IPoIB netdev QPN (only as CAP_NET_RAW) to “My QPN”
  - Learning the IPoIB QPN value:
    - Part of the link layer hardware address definition: <GID, QPN>
      - defined in IPoIB RFC: https://tools.ietf.org/html/rfc4391
    - Exposed on net_dev as L2 of interface
- **To support selective flows packet processing:**
  - Ibv_create_flow() to support L3 & L4 header attributes specs for IPoIB as well
- **Enabling stateless offloads:**
  - Checksum
  - TSO, LRO
  - RSS, TSS
  - Also for tunneled IPoIB (VXLANoIPoIB)
- **Reuse existing APIs and Verbs objects (ibv_cq, ibv_qp, ibv_wq, ibv_rwq_ind_tbl, ibv_mr, ibv_flow, …)**

# USER SPACE UD QP ASSOCIATION WITH IPOIB QP

- **Extend UD QP to be ASSOCIATED with another QPN**
  - RX: Allow steer ingress traffic flows from another QPN to application UD QP’s RecvQ (e.g.: steer IPoIB QPN traffic)
  - TX: Application post_send from “My UD QP” (SQ) to send with separately defined src QPN on the wire
    - Send with well known IPoIB QPN as DETH.sQPN
    - For TSS, all Send queues (SQ) will use same pre-defined QPN

- **Transport properties are defined by the Associated QP owner (IPoIB):**
  - Port, Pkey, State

- **my_ibv_qp->qp_num is a handle with local scope only**
  - Has no meaning on the wire

- **Data Path is UD/IB:**
  - Tx requires <AH, remote_pkey, remote_qpn>
  - Rx might hold GRH header following by IPoIB, IP, TCP/UDP…
    - Requires flow steering to steer flows out of associated QP

# VERBS API EXTENSIONS

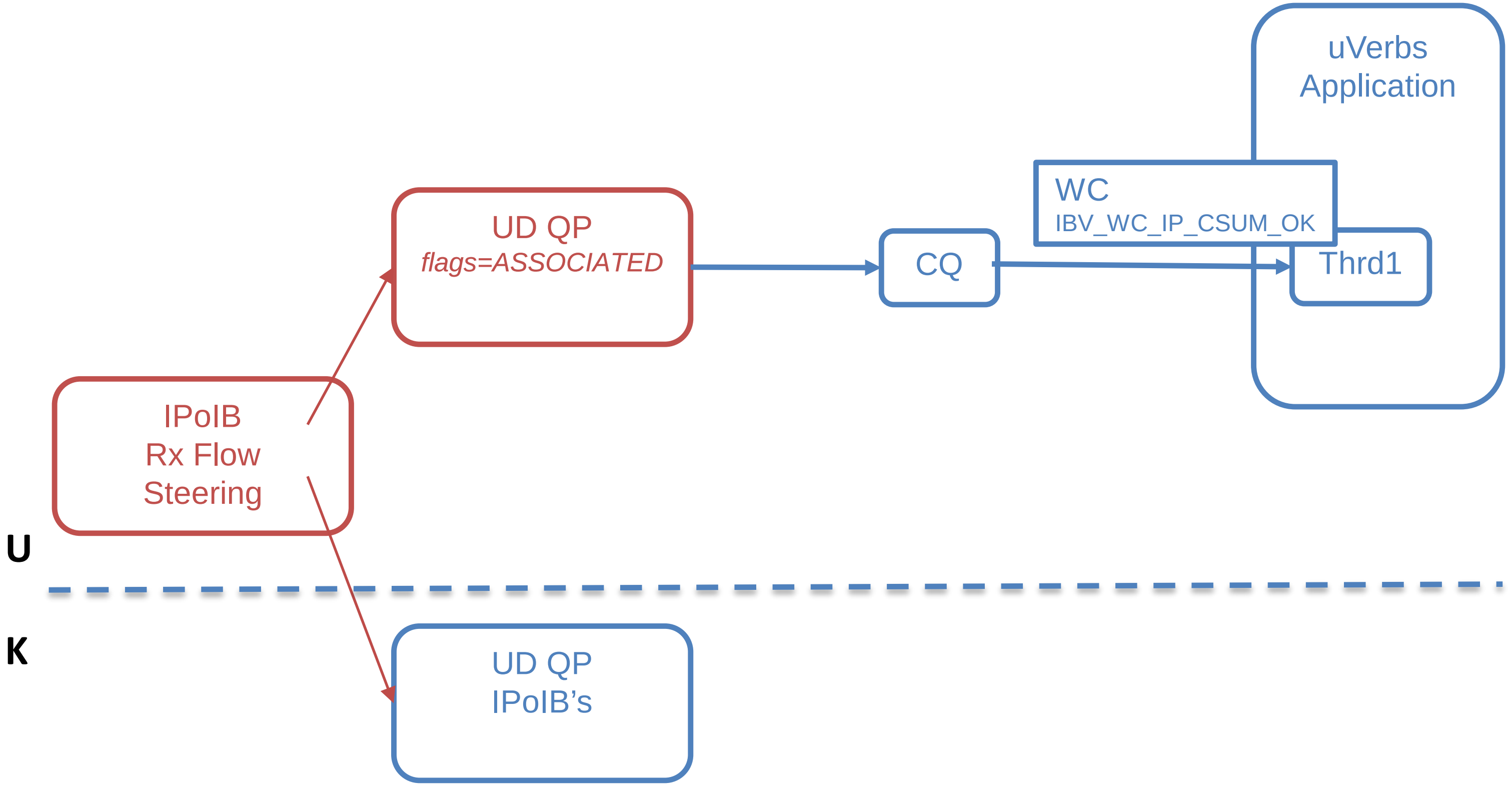
- **Create the Associated UD QP (overlay)**
  - `ibv_create_qp_ex( comp_mask |= IBV_QP_INIT_ATTR_ASSOCIATED_QPN)` and provide the QPN of the associated IPoIB UD QP
- **Checks & Failures**
  - If QPN is not part of the callers ibv_context then check for CAP_NET_RAW or fail with errno=EPERM
  - If requested ibv_device provider does not support the ASSOCIATED mask then fail with errno=ENOSYS
  - If requested QPN is not found, or QPN is not in a healthy state (RTS), fail with errno=EINVAL
- **Multi-Queue**
  - RSS with: `IBV_QP_INIT_ATTR_IND_TABLE | IBV_QP_INIT_ATTR_RX_HASH`
  - TSS with multiple ibv_wq of type IBV_SQ
- **TSO**
  - `With IBV_QP_INIT_ATTR_MAX_TSO_HEADER`
- **Modify**
  - Only state transitions are allowed (prepare the RQ and SQ)
  - No transport definitions for ibv_modify_qp() for the overlay QP
- **Device Cap**
  - `IBV_DEVICE_MANAGED_FLOW_STEERING`
  - `IBV_DEVICE_UD_IP_CSUM`
  - `struct ibv_rss_caps     rss_caps;`
  - `struct ibv_tso_caps     tso_caps;`

```
enum ibv_qp_init_attr_mask {
    IBV_QP_INIT_ATTR_PD             = 1 << 0,
    IBV_QP_INIT_ATTR_XRCD           = 1 << 1,
    IBV_QP_INIT_ATTR_CREATE_FLAGS   = 1 << 2,
    IBV_QP_INIT_ATTR_MAX_TSO_HEADER = 1 << 3,
    IBV_QP_INIT_ATTR_IND_TABLE      = 1 << 4,
    IBV_QP_INIT_ATTR_RX_HASH        = 1 << 5,
    IBV_QP_INIT_ATTR_ASSOCIATED_QP  = 1 << 6,
    IBV_QP_INIT_ATTR_RESERVED       = 1 << ~~6~~7
};

struct ibv_qp_init_attr_ex {
    ...
    uint32_t        associated_qp_num;
};
```

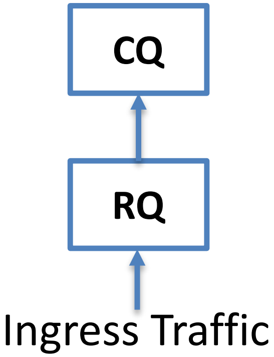
# VERBS USE CASE – RECEIVE FLOW STEERING

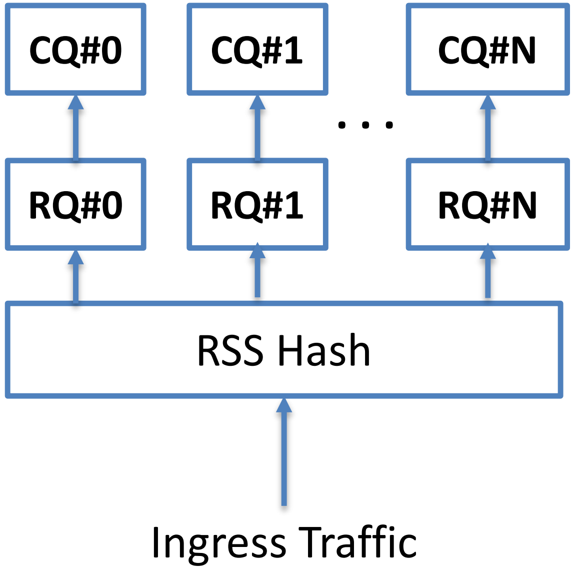
- **Use existing Ibv_create_flow()**
- **Ibv_flow_spec to support:**
  - IPoIB flow_spec_type
  - Associated QPN
- **Use existing ibv_flow_spec types to steer UDP/TCP 3/5-tuple flows**



# RSS - INTRODUCTION

- **Receive Side Scaling (RSS) technology enables spreading incoming traffic to multiple receive queues**
- **Each receive queue is associated with a completion queue**
- **Completion Queues (CQ) are bound to a CPU core**
  - CQ is associated with interrupt vector and thus with CPU
    - For polling, user may run polling for each CQ from associated CPU
  - In NUMA systems, CQ and RQ may be allocated on close memory to associated CPU
- **Spreading the receive queues to different CPU cores allows spreading receive workload of incoming traffic**





# RSS - FLOW OVERVIEW

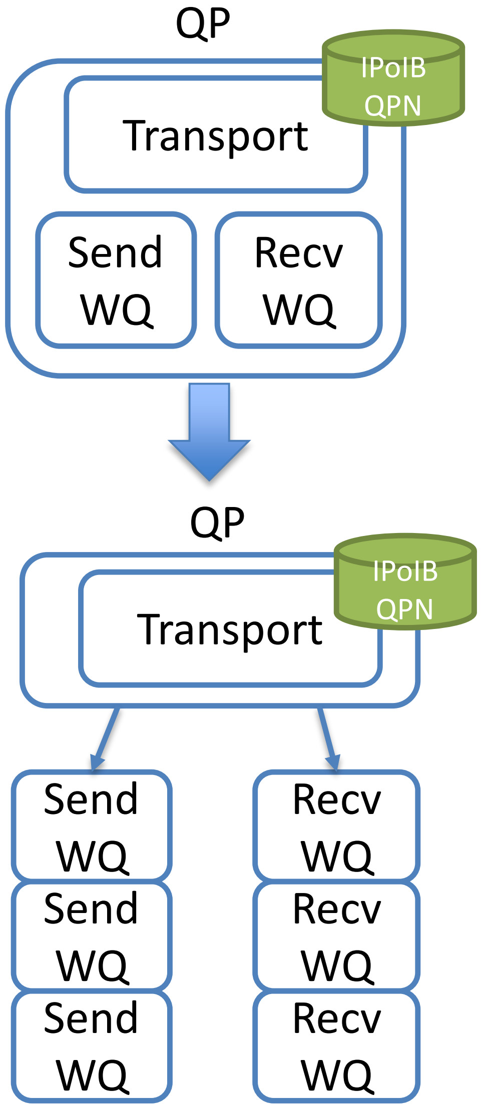
## Classify first, distribute after

- **Begin with classification**
  - Using Steering (ibv_create_flow()) classify incoming traffic
  - Classification rules may be any of the packet L3/4 header attributes
    - e.g. TCP/UDP only traffic, IPv4 only traffic, ..
  - Classification result is transport object - QP
- **Continue with spreading**
  - Transport object (QPs) are responsible for spreading to the receive queues
  - QPs carry RSS spreading rules and receive queue indirection table
- **RQs are associated with CQ**
  - CQs are associated with CPU core

- **Different traffic types can be subject to different spreading**

# RSS - WORK QUEUE (WQ)

- **Typically QPs (Queued Pairs) are created with 3 elements**
  - Transmit and receive Transport
  - Receive Queue
    - Exception is QPs which are associated with SRQ
  - Send Queue
- **Verbs were extended to support separate allocation of the above 3 elements**
  - Transport – ibv_qp with no RQ or SQ
    - Ibv_qp_type of IBV_QPT_UD with ASSOCIATED QPN
    - Ibv_init_qp_attr_ex->ibv_rx_hash_conf
  - Work Queue
    - Using ibv_qp_init_attr_ex-> ibv_rwq_ind_table
    - Where ibv_rwq_ind_table includes list of ibv_wq with IBV_RQ type



# RSS - WORK QUEUE (WQ) CONT.

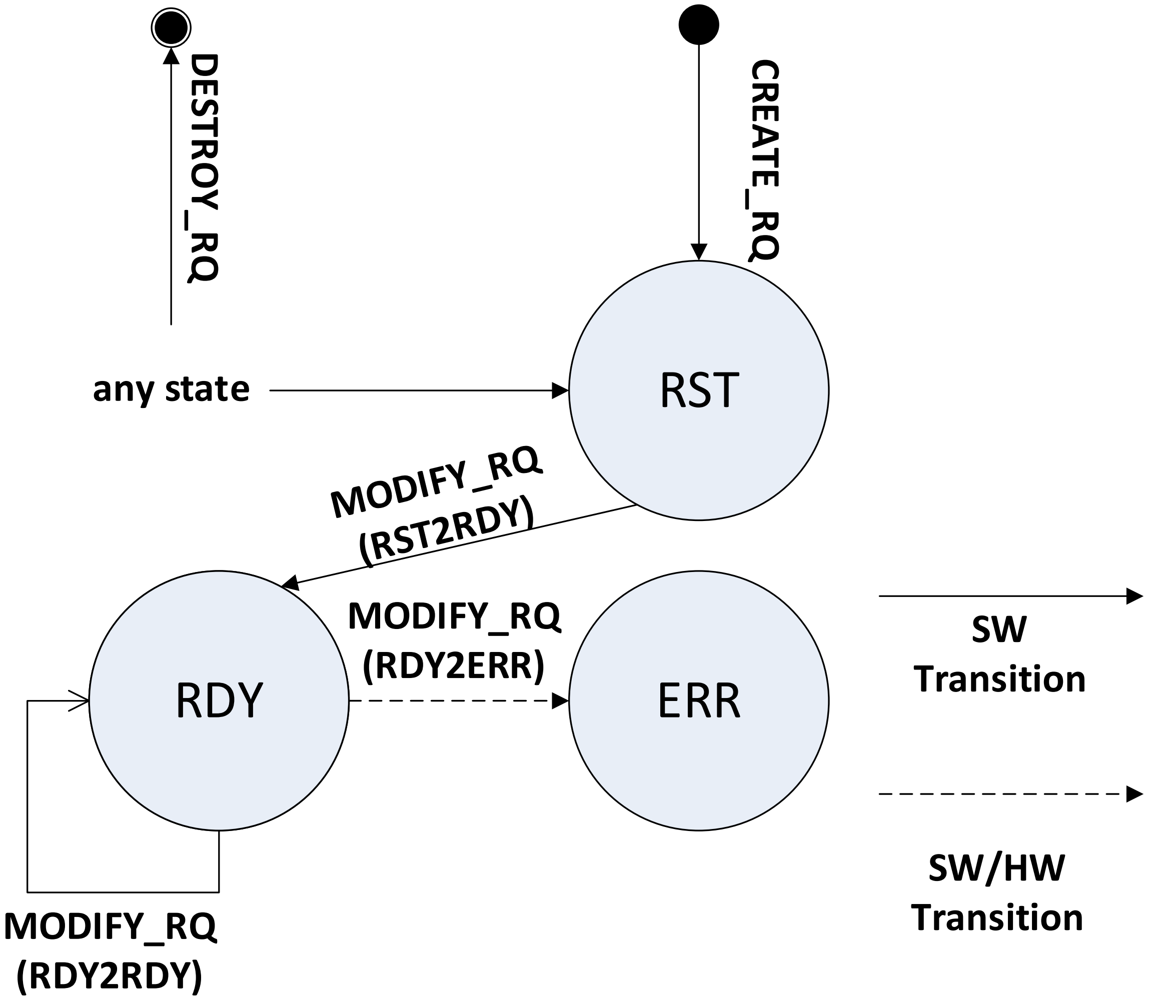
- **Use existing Work Queue object – ibv_wq**
- **Managed through following calls:**
  - ibv_wq *ibv_create_wq(ibv_wq_init_attr)
  - ibv_modify_wq(ibv_wq , ibv_wq_attr)
  - ibv_destory_wq(ibv_wq)
  - ibv_post_wq_recv(ibv_wq, ibv_recv_wr)
- **Work Queues (ibv_wq) are associated with Completion Queue (ibv_cq)**
  - Multiple Work Queues may be mapped to same Completion Queue (many to one)

```
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;
     uint32_t       comp_mask;
};
```

- **Work Queues of type Receive Queue (IBV_RQ) may share receive pool**
  - By associating many Work Queues to same Shared Receive Queue (the existing verbs ibv_srq object)

- **QP (ibv_qp) can be created without internal Send and Receive Queues and associated with external Work Queue (ibv_wq)**
- **QP can be associated with multiple Work Queues of type Receive Queue**
  - Through Receive Queue Indirection Table object

# RSS - WQ OF TYPE RQ STATE DIAGRAM



# RSS - RECEIVE WORK QUEUE INDIRECTION TABLE

- **Use existing Receive Work Queue Indirection Table object – ibv_rwq_ind_table**
- **Managed through following new calls:**
  - ibv_wq_ind_tbl *ibv_create_rwq_ind_table(ibv_rwq_ind_table_init_attr)
  - ibv_modify_rwq_ind_table(ibv_rwq_ind_table)*
  - ibv_query_rwq_ind_table(ibv_rwq_ind_tbl, ibv_rwq_ind_table_attr)*
  - ibv_destroy_rwq_ind_table(ibv_rwq_ind_tbl)
- **QPs may be associated with an RQ Indirection Table**
- **Multiple QPs may be associated with same RQ Indirection Table**

*** Not upstream yet**

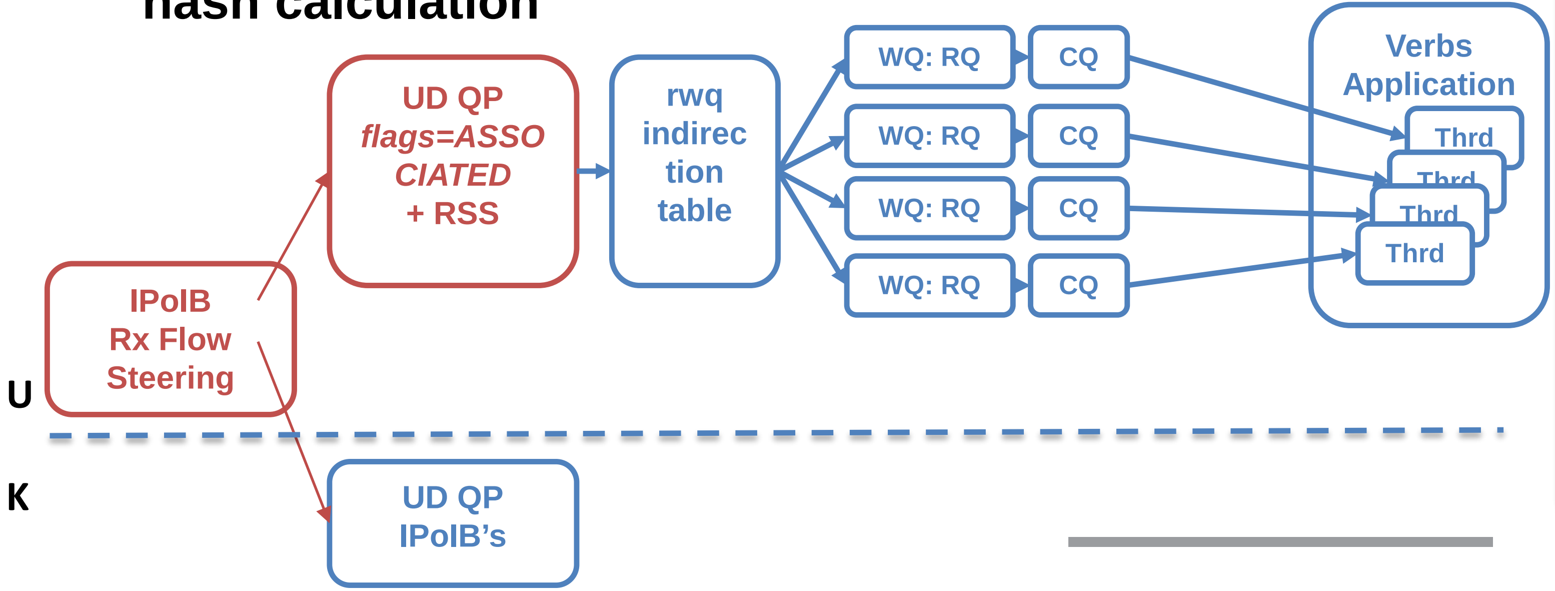
```
struct ibv_rwq_ind_table {
     struct ibv_context  *context;
     uint32_t             handle;
     int                 ind_tbl_num;
     uint32_t            comp_mask;
};

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

/*
 * 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;
     uint32_t            comp_mask;
};
```

# RSS - TRANSPORT OBJECT (QP)

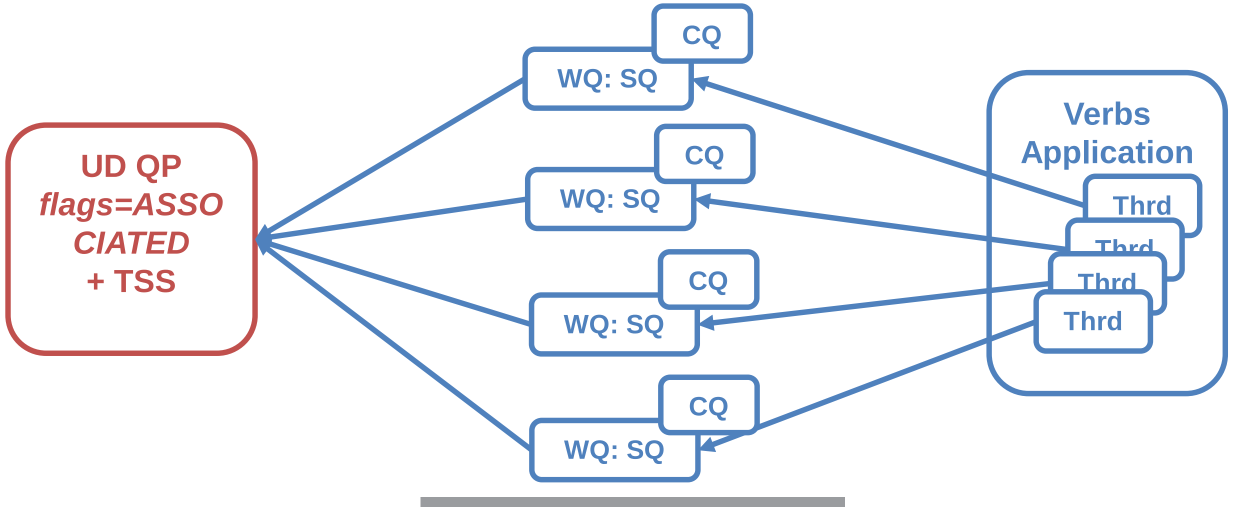
- **“RSS” QP**
  - ibv_qp_init_attr_ex->ibv_rx_hash_conf to define RSS hash params
    - Hash key
    - Packet headers
  - ibv_qp_init_attr_ex->ibv_rwq_ind_table to define RQ list
  - ibv_post_wq_recv to post receive WQE
- **On Receive, traffic is steered to the QP according Ibv_create_flow() spec and ASSOCIATED QPN**
- **Following, matching RQ is chosen according to QPs hash calculation**



```
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_RX_HASH_FUNC_TOEPLTIZ   = 1 << 0,
      IBV_RX_HASH_FUNC_XOR        = 1 << 1
};
/*
 Field represented by the flag will be
 used in RSS Hash calculation.
*/
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
};
```

# TSS

- **Work Queue to support new IBV_SQ type**
  - Ibv_wq_init_attr->wq_type
- **Multiple ibv_wq of type IBV_SQ can be associated with same IPoIB UD QP**
  - All SQs share same transport properties
    - QPN – use single s.QPN on the wire for all SQs
    - Pkey
    - Qkey
- **New ibv_post_wq_send() for posting send WQE on an SQ**
- **ibv_wq->cq of type IBV_SQ is associated with send CQ**
- **Same QP may be used for both RSS and TSS operations**

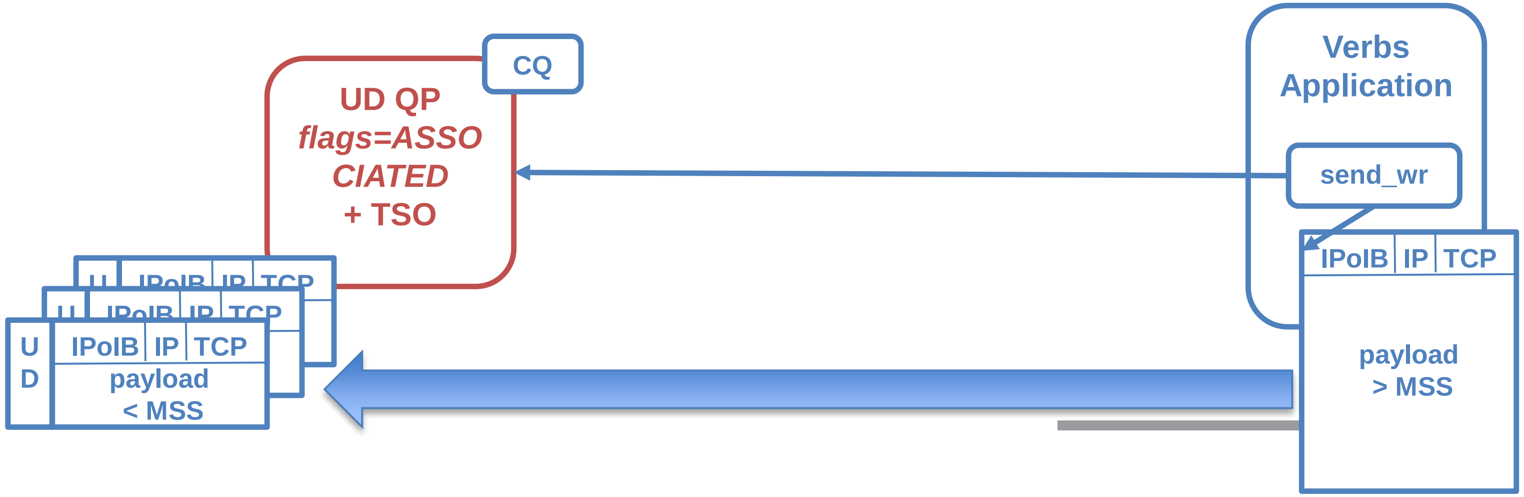


# TSO – USAGE FLOW

- **Check device capabilities with ibv_query_device_ex()**
  - ibv_is_qpt_supported (ibv_device_attr_ex->**tso_cap**.supported_qpts, IBV_QPT_UD)
  - ibv_device_attr_ex->**tso_cap**.max_tso
- **Create UD QP with TSO through ibv_create_qp_ex() :**
  - ibv_qp_init_attr_ex->comp_mask = IBV_QP_INIT_ATTR_MAX_TSO_HEADER
  - ibv_qp_init_attr_ex->max_tso_header = 44; // IPoIB/IPv4/TCP headers
- **Send the large TSO frame with ibv_post_send():**
  - Send opcode: ibv_send_wr->opcode = IBV_WR_TSO
  - IPoIB TSO packet setting: ibv_send_wr->tso->hdr, hdr_sz, mss
  - UD send addressing: ibv_send_wr->ud->ah, remote_qpn, remote_qkey
- **Next – Allow SQ support for TSO**
  - ibv_post_wq_send() to support IBV_WR_TSO operation

```
struct ibv_tso_caps {
    uint32_t max_tso;
    uint32_t supported_qpts;
};

struct ibv_send_wr {
    uint64_t                wr_id;
    struct ibv_send_wr     *next;
    struct ibv_sge         *sg_list;
    int                     num_sge;
    enum ibv_wr_opcode      opcode;
    int                     send_flags;
    __be32                  imm_data;
    union {
        struct {
            struct ibv_ah  *ah;
            uint32_t        remote_qpn;
            uint32_t        remote_qkey;
        } ud;
    } wr;
    union {
        struct {
            void                   *hdr;
            uint16_t                hdr_sz;
            uint16_t                mss;
        } tso;
    };
};
```



# OVERLAY NETWORKING

- **Tunneling technologies like VXLAN, NVGRE, GENEVE were introduced for solving cloud scalability and security challenges**
- **Allow tunneling over IPoIB – outer L2 is the IPoIB 4 bytes header**
- **Require extensions of traditional NIC stateless offloads**
  - TX and RX inner headers checksum
    - ibv_qp_attr to control inner csum offload
    - Ibv_send_wr, ibv_wc to request and report inner csum
  - Inner TCP Segmentation and De-segmentation (LSO/LRO)
    - ibv_send_wr to support inner MSS settings
  - Inner Ethernet header VLAN insertion and stripping
    - Ibv_qp_attr to control inner VLAN insert/strip
    - Ibv_send_wr to indicate VLAN
    - Ibv_wc to report strip VLAN
  - Steering to QP according to outer and inner headers attributes
    - Ibv_create_flow(ibv_flow_attr) to support inner headers
  - Perform RSS based on inner and/or outer header attributes
    - Ibv_qp_attr.ibv_rx_hash_conf to support inner header attributes
  - Inner packet parsing and reporting its properties in Completion Queue Entry (CQE)
    - Ibv_wc to support inner headers extraction

# SUMMERY

- **User Verbs generic object model to enable user space IPoIB Packet Processing**
- **Reuse existing Verbs infrastructure**
  - Ibv_qp, ibv_cq, ibv_flow, ibv_mr
  - Ibv_wq, ibv_rwq_ind_table
- **Control and data path infrastructure**
  - Use OS services for control path and allow bypass for data path
  - Can answer performance requirements for both high PPS, BW and low latency
- **Create association between application UD QP and underlying IPoIB net_dev QPN**
  - Allow app UD QP to receive selected flows of the ingress traffic
  - Allow send from application UD QP with wire QPN of the net_dev UD QPN
- **Support all packet processing stateless offloads**
  - CSUM, RSS, TSS, TSO, LSO
  - Many are already available in verbs for Ethernet RAW PACKET QP – ***reuse, Yeh*** ☺

OPENFABRICS ALLIANCE

13th ANNUAL WORKSHOP 2017

# THANK YOU

Tzahi Oved, Alex Rosenbaum

Mellanox Technologies

Mellanox® TECHNOLOGIES

Connect. Accelerate. Outperform.™