AGENDA

- Motivation for a new transport
- Introducing: Dynamically-connected transport (DC)
- A bit about Direct Verbs (DV)
- API Proposal
- Examples and use cases
### MOTIVATION

- UD doesn’t support RDMA (but is very scalable).
- RD doesn’t scale well (but supports RDMA).

<table>
<thead>
<tr>
<th>Feature</th>
<th>UD</th>
<th>UC</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send/Recv</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>RDMA Write</td>
<td>X</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>RDMA Read / Atomic</td>
<td>X</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>Max. send size</td>
<td>MTU</td>
<td>2GB</td>
<td>2GB</td>
</tr>
<tr>
<td>Reliability, Ordering</td>
<td>X</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>Scalability (per-process, for N processes)</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
**MOTIVATION**

- UD doesn’t support RDMA (but is very scalable).
- RD doesn’t scale well (but supports RDMA).
- Enter Dynamically-connected transport (DC).
  - The best of both worlds
  - Supports RDMA, RC-like capabilities
  - Scalable, single QP object with multiple destinations (via AD, UD-like)

<table>
<thead>
<tr>
<th></th>
<th>UD</th>
<th>UC</th>
<th>RC</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send/Recv</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>RDMA Write</td>
<td>X</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>RDMA Read / Atomic</td>
<td>X</td>
<td>X</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Max. send size</td>
<td>MTU</td>
<td>2GB</td>
<td>2GB</td>
<td>2GB</td>
</tr>
<tr>
<td>Reliability, Ordering</td>
<td>X</td>
<td>X</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Scalability (per-process, for N processes)</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>1*</td>
</tr>
</tbody>
</table>
DYNAMICALLY-CONNECTED TRANSPORT

- **DC (Dynamically Connected) Scalable Transport Service**
  - Reduces #QPs per node
  - RC-like reliability semantics

- **DC has asymmetric API:**
  - On the recv-side we have DC Target, or DCT
    - How many? One is enough
  - On the send-side we have DC Initiators, or DCI
    - How many? Less than #targets, but >1

- **Internally, DC forms “temporary connections”:**
  - First send-WR on a DCI, connects this DCI to a remote DCT
  - Second send-WR uses this open connection
  - DCI disconnects after some idle period without sends
  - What if the second send-WR is to a different destination? next slide…
A DCI can “switch destinations”
- If the next send-WR has a different destination specified

A DCT has a pool of “responders” (DCRs)
- Each incoming DC connection is allocated a DCR
DCI RECYCLING TRADEOFFS

- **Too few DCIs**
  - Same DCI switches back-and-forth between destinations
  - Redundant connect/disconnect flows (worst case: per-send)
  - Hurts latency

- **Too many DCIs**
  - Still not as bad as $N^2$ RC QPs…
  - Consumes resources and is bad for caching

- **Best practice**
  - Maintain a <DCI dest> hash-table, reducing connection re-establishment
  - LRU recycling policy, to increase the odds of picking a disconnected DCI to send on
DC HANDSHAKE TYPES

- **Half Handshake**
  - “speculative” data sent right after the CONNECT message
  - Improves latency, especially of small messages
  - *seen in previous slides*

- **Full Handshake**
  - Like a 3-way TCP handshake
  - Prevent potential race conditions…

![Diagram of DC Full Handshake Communication](image-url)
A DC connection request could be denied:
- If the DCT does not have enough resources (DCRs) to honor it
- All DC Responders are currently in use…
- In this case, the target may send a connection NAK (CNAK) to the DCI

API lets you set pool size
- Maximum DC Responders
  - Size of the pool
- Minimum DC Responders
  - Below this number of available DCRs – DCRs send CNAKs and release
  - No DC Responders?
    - No Response (timeout).
    - Only happens if you bombard a ‘small’ DCT…
API
**DIRECT VERBS**

- **Direct Verbs (DV) is a new place for vendor-specific API**
  - Distributed as part of the provider (libmlx5): `providers/mlx5/mlx5dv.h`
  - Exposes HW registers/definition
  - Good for non-IB-spec, configuration flags and objects, like DC

- **DV contains (today):**
  - DC
  - Advanced CQ
  - “Bare-metal” Data Path Access

- **Verbs/DV overlapping**
  - Both have QP object
  - DV needs to “bypass” verbs
  - QP state checks
  - “Vendor QP” passes control

```c
@@ -1069,7 +1069,9 @@ enum ib_qp_type {
    IB_QPT_RAW_PACKET = 8,
    IB_QPT_XRC_INI = 9,
    IB_QPT_XRC_TGT,
+   IB_QPT_VENDOR = 0xFFF,
    IB_QPT_MAX,
}
@@ -1196,6 +1196,9 @@ int ib_modify_qp_is_ok(...
    enum ib_qp_attr_mask req_param, opt_param;
+   if (type >= IB_QPT_MAX)
+      return 0;
```
struct ibv_qp *mlx5dv_create_qp(struct ibv_context *context,
    struct ibv_qp_init_attr_ex *qp_attr,
    struct mlx5dv_qp_init_attr *mlx5_qp_attr);

struct mlx5dv_qp_init_attr {
    uint64_t comp_mask;     /* Use enum mlx5dv_qp_init_attr_mask */
    uint32_t create_flags;  /* Use enum mlx5dv_qp_create_flags */
    struct mlx5dv_dc_init_attr dc_init_attr;
};

struct mlx5dv_dc_init_attr {
    enum mlx5dv_dc_type dc_type;
    uint64_t dct_access_key;
};

enum mlx5dv_dc_type {
    MLX5DV_DCTYPE_DCT     = 1,
    MLX5DV_DCTYPE_DCI,
};

enum mlx5dv_qp_init_attr_mask {
    MLX5DV_QP_INIT_ATTR_MASK_QP_CREATE_FLAGS        = 1 << 0,
    MLX5DV_QP_INIT_ATTR_MASK_DC                     = 1 << 1,
};
WHERE’S THE REST OF THE API?

- mlx5dv_create_qp() returns a standard struct ibv_qp*
  - Verbs API applies, e.g. ibv_modify/destroy_qp

- DCI Post Send
  - Requires extension of struct ibv_send_wr to get both:
    - RDMA/Atomic
    - DC AH
  - Refactor the QP Post Send API:
    - Registered provider functions per QP Type
    - Separate operations according to SendOpCode
    - Allow send DV extensions: setter for DC AH

- Poll CQ
  - Additional DV getters based on struct ibv_cq_ex

- Other DV API allows a user to create his own ibv_post_send()
  - Reference implementation is available in UCX, an open-source P2P library
    https://github.com/openucx/
THANK YOU

Alex Rosenbaum
Mellanox Technologies