An Introduction to Remote Direct Memory Access

Patrick MacArthur
pio3@wildcats.unh.edu

Department of Computer Science
University of New Hampshire

CS 980
Advanced Topics in Network Research
October 28, 2013
1. Background

2. RDMA Concepts

3. InfiniBand

4. Performance Study

5. UNH EXS

6. Summary
Traditional Networks

TCP
- Reliable, connected
- Byte stream-oriented protocol
- Buffered in kernel-space on both sides
- Flow control via window size
- Synchronous: data transfer operations block until they complete

IP/Ethernet
- Each switch and router completely independent, autonomous
- Datagram-oriented
- Lossy—relies on upper layers to retransmit lost packets
• Message-oriented protocol
• "Zero-copy": direct application virtual memory to application virtual memory transfers
• Kernel bypass: userspace application talks directly with the hardware to do data transfers
• Flow control via credits at link layer—no loss due to congestion
• Asynchronous: data transfer operations placed onto queue
• Message latencies on the order of microseconds
RDMA Technologies

**InfiniBand [InfiniBand 2007]**
- Most popular RDMA implementation
- Defines entire network stack from top to bottom
- Speeds: SDR (10 Gbps), DDR (20 Gbps), QDR (40 Gbps), FDR (56 Gbps)

**iWARP (Internet Wide-Area RDMA Protocol) [RFC 5040, RFC 5041, RFC 5044]**
- Implements RDMA on top of TCP with 3 layers
- Defined by IETF

**RoCE (RDMA over Converged Ethernet) [InfiniBand 2010]**
- Implements upper layers of InfiniBand on top of Ethernet
- Defined by InfiniBand Trade Association
A local-area RDMA network is usually referred to as a **fabric**

A **channel adapter** is the hardware component that connects a system to the fabric

- iWARP refers to it as an RNIC (RDMA Network Interface Card)
InfiniBand Verbs

InfiniBand defines a set of “verbs” to communicate with RDMA hardware, but not an API

- Semantic details but no specific function calls, data structures, ...

OFA Verbs

The OpenFabrics Alliance (OFA) created a vendor-independent C API to access the InfiniBand verbs

- Very low-level API, involving direct manipulation of data structures
- Lots of code needed to write a very simple data transfer

OFED

The OFA periodically releases OFED (OpenFabrics Enterprise Distribution) which contains drivers and userspace libraries/tools for RDMA
Outline

1. Background
2. RDMA Concepts
3. InfiniBand
4. Performance Study
5. UNH EXS
6. Summary
Current RDMA hardware has specific requirements for memory used in data transfers:

- Memory must not be modified by application during data transfer
- Memory cannot be paged out by operating system—physical to virtual mapping must be fixed

Registration

To deal with the second requirement, applications must register memory to be used in data transfers

- Returns a local and remote key pointing to memory area
- Registration keys are supplied as part of data transfer request
Queue Pairs
- A queue pair is associated with each side of an RDMA transfer
- Assigned an integer identifier that is sent to the other communication endpoint(s)
- A queue pair consists of a send and a receive request queue
- Post Send Request and PostRecv Request verbs used to add work requests to the queue

Completion Queues
- A send and receive completion queue is associated with a queue pair (at creation time)
- May be same queue for both, or two separate queues
- Poll Completion Queue verb used to remove completion events from the queue
**Data Transfer Operations**

**SEND/RECV**
- Analogous to socket send/recv, but using direct transfers
- **RECV** must be ready prior to arrival of **SEND**
- Both sides of transfer receive notification of completion

**RDMA_WRITE**
- Push of local data to remote memory area
- Sender must supply virtual address and remote key of destination memory area
- Sender initiates and receives notification; receiver is completely **passive** and receives no notification
Data Transfer Operations (continued)

**RDMA_READ**
- Pull of remote memory area to local memory area
- Receiver must supply virtual address and remote key of destination memory area
- Receiver initiates and receives notification; sender is completely passive and receives no notification
- Inefficient since request moves in opposite direction from data

**RDMA_WRITE_WITH_IMM**
- Like RDMA_WRITE, but receiver receives notification as well
- Request includes four bytes of out-of-band immediate data
- Immediate data included in receive completion event
- Supported by InfiniBand and RoCE, but not current version of iWARP
One master **subnet manager** (SM) is elected for the entire fabric
Can have other SMs, but for failover only
Controls switching tables for entire fabric
Switching tables must be recomputed every time a node added/removed to/from fabric
Global Identifier (GID)
- Every InfiniBand node has an assigned **EUI-64 identifier**
  - 24 bit company identifier assigned by IEEE
  - 40 bit extension identifier assigned by manufacturer
- **Global Identifier (GID)** is a 128-bit identifier built from **EUI-64 identifier** and 64-bit **subnet prefix** assigned by SM
- Used for routing between subnets

Local Identifier (LID)
- Local ID (LID) is a 16-bit identifier dynamically assigned by SM
- Used for switching within a subnet
- **may change during switching table recalculation**
InfiniBand specifies four transport protocols:

- Reliable Connected (analogous to TCP)
- Reliable Datagram
- Unreliable Connected
- Unreliable Datagram (analogous to UDP)
<table>
<thead>
<tr>
<th>Local Route Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Transport Header</td>
</tr>
<tr>
<td>Extended Transport Headers <em>(if needed)</em></td>
</tr>
<tr>
<td>Packet Payload</td>
</tr>
<tr>
<td>Invariant CRC</td>
</tr>
<tr>
<td>Variant CRC</td>
</tr>
</tbody>
</table>

This is the packet format for packets using InfiniBand transport protocols.
## Local Route Header

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual lane</td>
<td>version</td>
<td>service level</td>
<td>reserved</td>
</tr>
<tr>
<td>destination LID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>packet length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>source LID</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Virtual Lane and Service Level fields used in QoS
- LNH is Link Next Header—1 bit global/local, 1 bit IB/raw
- Packet Length is in 4-byte words
### Base Transport Header

<table>
<thead>
<tr>
<th></th>
<th>opcode</th>
<th>SE</th>
<th>M</th>
<th>PadCnt</th>
<th>Version</th>
<th>Partition Key (P_KEY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>7-8</td>
<td>15-16</td>
<td>23-24</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SE</td>
<td>M</td>
<td>PadCnt</td>
<td>Version</td>
<td>Partition Key (P_KEY)</td>
</tr>
<tr>
<td></td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>destination QPN</td>
</tr>
<tr>
<td>A</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>packet sequence number</td>
</tr>
</tbody>
</table>

- SE (Solicited Event) bit indicates that CQ event handler should fire
- M is related to Automatic Path Migration
- PadCnt is number of bytes of padding in the Packet Length
- Partitions are analogous to VLANs on Ethernet
- Note that source Queue Pair Number is not part of this header
<table>
<thead>
<tr>
<th>0</th>
<th>3B2</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote Key (R_KEY)</td>
<td>DMA Length</td>
<td></td>
</tr>
<tr>
<td>Syndrome</td>
<td>Message Sequence Number</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
<td></td>
</tr>
</tbody>
</table>

0 7 8 15 16 23 24 31
Outline

1. Background
2. RDMA Concepts
3. InfiniBand
4. Performance Study
5. UNH EXS
6. Summary
Simultaneous Operations

Due to the asynchronous nature of RDMA, a key part of using RDMA effectively is having as many simultaneously outstanding transfer operations as possible.

The usual strategy is:

- Post as many RECV operations as possible
- Post a new RECV as soon as a RECV completion event fires
- Post send operations as soon as the data is ready to send

More simultaneous operations are needed over distance
### Completion Event Detection Strategies

<table>
<thead>
<tr>
<th>Event Notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>User requests notification when a completion event arrives</td>
</tr>
<tr>
<td>Requires kernel involvement to put thread/process to sleep and wake it up when event arrives</td>
</tr>
<tr>
<td>Benefit: Very low CPU usage</td>
</tr>
<tr>
<td>Cost: Lower throughput and higher latency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Busy Polling</th>
</tr>
</thead>
<tbody>
<tr>
<td>User polls in a tight loop for new completion events</td>
</tr>
<tr>
<td>Get events as soon as they arrive with no kernel involvement</td>
</tr>
<tr>
<td>Benefit: High throughput and low latency</td>
</tr>
<tr>
<td>Cost: 100% CPU usage</td>
</tr>
</tbody>
</table>

[MacArthur and Russell 2012]
Completion Event Detection Strategies

Throughput (Megabits per second)

Message size

1KiB 8KiB 64KiB 512KiB 4MiB 32MiB 256MiB

RDMA_WRITE, busy
RDMA_WRITE_WITH_IMM, busy
SEND/RECV, busy
RDMA_READ, busy
SEND/RECV, notify
RDMA_WRITE, notify
RDMA_WRITE_WITH_IMM, notify
RDMA_READ, notify
Inline Send Requests

- The **inline** feature causes an RDMA adapter to copy a small message into a buffer on the CA at the time that the work request is posted.
- Subject to channel adapter limit.
- Avoids need for memory registration on sending side.
- Performance gain for very small messages, but hurts for larger messages.

[MacArthur and Russell 2012]
One-way Time for Inline Send Requests

- 912-byte messages
- 256-byte messages
- 64-byte messages
- 16-byte messages
- 4-byte messages
- 1-byte messages

Buffer count

One-way time per message (Microseconds)

Patrick MacArthur
RDMA
CS 980; 2013-10-28
Outline

1. Background
2. RDMA Concepts
3. InfiniBand
4. Performance Study
5. UNH EXS
6. Summary
UNH EXS (Extended Sockets)

- Based on ES-API (Extended Sockets API) published by the Open Group [Interconnect 2005]
- Extensions to sockets API to provide asynchronous, zero-copy transfers
  - Memory registration (exs_mregister(), exs_mderegister())
  - Event queues for completion of asynchronous events (exs_qcreate(), exs_qdequeue(), exs_qdelete())
  - Asynchronous operations (exs_send(), exs_recv(), exs_accept(), exs_connect())
- UNH EXS supports SOCK_SEQPACKET (reliable message-oriented) and SOCK_STREAM (reliable stream-oriented) modes
Sample asynchronous send operation

```c
exs_mhandle_t mh = exs_mregister(buf, bufsize, EXS_ACCESS_READ);
exs_qhandle_t qh = exs_qcreate(10);

if (exs_send(fd, buf, bufsize, 0, mh, 0, qh) < 0) {
    perror("Could not start send operation");
    /* bail out */
}

/* do work in parallel with data transfer */

exs_event_t ev;
if (exs_qdequeue(qh, &ev, 1, NULL) < 0) {
    perror("Could not get send completion event");
    /* bail out */
}

fprintf(stderr, "Send of %d bytes complete with errno=%d; actual byte count %d\n",
        bufsize, ev.exs_evt_errno,
        ev.exsEvtUnion.exsEvtXfer.exsEvtLength);
```
For SOCK_SEQPACKET:

- Receiver sends ADVERT containing address, length, and remote key of receive buffer
- Sender uses RDMA_WRITE_WITH_IMM to write data directly into destination buffer

For SOCK_STREAM, same as above, except there is also an intermediate receive buffer that is used if no advertisements are available
**Key idea:** allow sender to use direct or indirect transfer based on current conditions

**Direct Transfer**

- Sender sends to receiver using direct transfer
- User sends to buffer
- Exs_send
- Direct

**Indirect Transfer**

- Sender sends to intermediate receive buffer
- Exs_send
- Indirect
- User sends to buffer
- Exs_recv
- Intermediate receive buffer
- User recv buffer
Key challenge: Advertisements may arrive late (i.e., after sender has already sent corresponding data), and sender must distinguish between “fresh” and “stale” advertisements

Here, the sender does not know how many exs_recv operations (corresponding to ADVERTs) were consumed by its INDIRECT send. The solution is to assign each advertisement a phase number and increment the phase number at the sender when an indirect transfer is sent.
UNH EXS Dynamic Stream Protocol Performance

Throughput (Megabits per second)

Direct transfers (percent)

Simultaneous outstanding operations

Outstanding Receives

Outstanding Sends

Indirect-only Protocol

Dynamic Protocol

Direct-only Protocol

Outstanding Receives

Outstanding Sends

Simultaneous outstanding operations

Patrick MacArthur

RDMA

CS 980; 2013-10-28 34 / 40
Outline

1. Background
2. RDMA Concepts
3. InfiniBand
4. Performance Study
5. UNH EXS
6. Summary
- RDMA is used all over high-performance computing due to its high throughput and low latency
- Asynchronous, kernel bypass, and “zero-copy”
- Three standard protocol stacks: InfiniBand, iWARP, and RoCE
Open Issues

- OFED verbs library
  - Too complex for casual network programmers
  - Does not match semantics of consumers such as MPI
- Applications not written with RDMA in mind
  - Messages that are large for TCP/IP are *small* for RDMA
  - Applications often use only double buffering if they have any support at all for multiple outstanding transfers
- Subnet manager scalability
- Performance over distance
- Error handling
P. Cully, U. Elzur, R. Recio, S. Bailey, J. Carrier,  
“Marker PDU Aligned Framing for TCP Specification,”  
RFC 5044 (Standards Track), Oct. 2007. [Online]. Available:  
http://www.ietf.org/rfc/rfc5044.txt

P. Grun,  
*Introduction to InfiniBand for End Users*,  

InfiniBand Trade Association,  
“InfiniBand Architecture Specification version 1.2.1,”  
Nov. 2007.

—,  
“Supplement to Infiniband Architecture Specification Volume 1,  
version 1.2.1: Annex A16: RDMA over Converged Ethernet (RoCE),”  
Apr. 2010.
Interconnect Software Consortium in association with the Open Group,
“Extended Sockets API (ES-API) Issue 1.0,”

P. MacArthur, R. Russell,
“A Performance Study to Guide RDMA Programming Decisions,”

R. Recio, B. Metzler, P. Culley, J. Hilland, D. Garcia,