An Efficient Method for Stream Semantics over RDMA

Patrick MacArthur <pio3@cs.unh.edu>
Robert D. Russell <rdr@cs.unh.edu>

Department of Computer Science
University of New Hampshire
Durham, NH 03824-3591, USA

28th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2014)
May 21, 2014
The authors would like to thank the University of New Hampshire InterOperability Laboratory for the use of their RDMA cluster for the development, maintenance, and testing of UNH EXS. We would also like to thank the UNH-IOL and Ixia for the use of an Anue network emulator for performance testing.

This material is based upon work supported by the National Science Foundation under Grant No. OCI-1127228 and under the National Science Foundation Graduate Research Fellowship Program under award number DGE-0913620.
Background
### Differences Between RDMA and TCP Sockets

<table>
<thead>
<tr>
<th>TCP (Transmission Control Protocol) Sockets</th>
<th>RDMA (Remote Direct Memory Access)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Kernel involvement in all data transfers</td>
<td>- “Kernel bypass”: data transfers with no OS involvement</td>
</tr>
<tr>
<td>- Buffered in kernel-space on both sides of connection</td>
<td>- “Zero-copy”: Direct virtual memory to virtual memory transfers</td>
</tr>
<tr>
<td>- Byte-stream oriented protocol</td>
<td>- Message-oriented</td>
</tr>
<tr>
<td>- Synchronous programming interface</td>
<td>- Asynchronous programming interface</td>
</tr>
</tbody>
</table>
An Efficient Method for Stream Semantics over RDMA

Background
RDMA vs. TCP
Related Work
UNH EXS
Dynamic Protocol
Motivation
Overview
Scenario
Performance Evaluation
Simple Distance
Conclusions

TCP Sockets Data Transfer
An Efficient Method for Stream Semantics over RDMA

MacArthur and Russell

Background
- RDMA vs. TCP

Related Work
- UNH EXS

Dynamic Protocol

Motivation

Overview

Scenario

Performance Evaluation
- Simple Distance

Conclusions

RDMA WRITE Data Transfer

User send buffer

(sender)

receiver

(VA, rkey)

RDMA WRITE (VA, rkey)

User recv buffer

User send buffer
Message vs. Byte Stream Semantics

**Message Transfer (UDP)**

sender

```
Hello
World
```

receiver

```
Hello
W
```

**Byte Stream Transfer (TCP)**

sender

```
Ok
Hello
World
```

receiver

```
OkHello
W
world
```
Prior Implementations of Sockets over RDMA

- Sockets Direct Protocol (SDP) (defined by InfiniBand specification [InfiniBand 2011])
  - BCopy (buffering on both sides)
  - ZCopy (zero-copy, send() blocks) [Goldenberg 2005]
  - AZ-SDP (asynchronous, zero-copy, segfault handler) [Balaji 2006]
- uStream (asynchronous but not zero-copy) [Lin 2009]
Current Implementations of Sockets over RDMA

- SMC-R (100% compatibility with TCP/IP and sockets)
- rsockets (high-performance sockets replacement) [Hefty 2012]
- UNH EXS (extended sockets) [ISC 2005, Russell 2009]
Based on ES-API (Extended Sockets API) published by the Open Group [ISC 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
Example

Example 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/%d bytes complete with errno=%d\n",
        bufsize, ev.exs_evt_union.exs_evt_xfer.exs_evt_length,
        ev.exs_evt_errno);
```
Dynamic Protocol
Motivation

- Provide familiar **byte-stream** abstraction over RDMA
- **Dynamically** combine best aspects of zero-copy RDMA and fast send response benefit of TCP-style buffering.
- Deliver user data from sender to receiver **in order with no errors**
- Implementation of sender and receiver should be as **independent** as possible
- Work well over large **distance**
- Work **automatically** without user intervention
**Key idea:** UNH EXS automatically uses direct or indirect transfer based on current conditions.
Dynamic Protocol Challenge
Late Advertisements

- `exs_recv(fd, buf, n, ...)` may actually receive between 1 and `n` bytes (not known in advance by receiver)
- `exs_send(fd, buf, n, ...)` will transfer `n` bytes in absence of network errors (due to asynchronous nature of EXS)
- Advertisements may arrive late (i.e., after sender has already sent corresponding data indirectly)
- Sender must distinguish between “fresh” and “stale” advertisements
Dynamic Protocol
Late Advertisements: Solution

- Advertisement includes estimated **byte sequence number** and **phase number**
- Byte sequence number is estimated sequence number of first byte in stream satisfying associated `exs_recv()` request.
  - Each advertisement at receiver increases estimate by 1 (minimum transfer length)
  - Each transfer completion at receiver increases estimate by \( n - 1 \) (adjusts for actual number of bytes transferred)
  - Exact value known at sender
- Phase number is monotonically nondecreasing identifier of transfer phase
  - Incremented on each switch between direct and indirect transfers on both sides
  - At sender, always \( \geq \) that of last processed advertisement
At start, sender has sent 2 bytes to receiver directly. Receiver issues two `exs_recv` requests.
Scenario

Sender issues single `exs_send` request.
Scenario

Sender and receiver continue to issue `exs_send` and `exs_recv` requests.
Advertisement A4 is delayed, so sender sends *indirect* transfer.
Scenario

Receiver **holds off** on sending off advertisements until all pending advertisements satisfied.
Receiver starts sending advertisements again once caught up. Sender matches advertisement once sequence number in ADVERT matches and phase number in ADVERT is greater.
Performance Evaluation
Performance Evaluation

- Comparison of dynamic protocol with baseline protocols
  - **Direct-only** protocol: sender always waits for ADVERT (receiver never copies)
  - **Indirect-only** protocol receiver never sends ADVERTs (receiver always copies)
- Measure throughput, CPU usage, and percent of direct sends (average across entire run)
Simple Tests

- Systems used: Intel Xeon 2.40 GHz E5-2609 CPUs, 64 GB RAM, PCI-e Gen 3
- HCAs: Mellanox ConnectX-3 54.54 Gbps FDR InfiniBand HCAs
- Connection: Mellanox SX6036 FDR InfiniBand switch
Dynamic Protocol Throughput Comparison
outstanding sends = $\frac{1}{2}$ outstanding receives

- Direct-only Protocol (never copies)
- Dynamic Protocol
- Indirect-only Protocol (always copies)

Higher is better

Throughput (Megabits per second)

Outstanding Transfers

0 1 2 4 8 16 32

Throughput Comparison Graph

- Direct-only Protocol (never copies)
- Dynamic Protocol
- Indirect-only Protocol (always copies)
Dynamic Protocol CPU Comparison

outstanding sends = \( \frac{1}{2} \) outstanding receives

![Dynamic Protocol CPU Comparison Graph](image)

Lower is better
Dynamic Protocol Throughput
Outstanding receives fixed at 32

Higher is better

Throughput (Megabits per second)

Outstanding Transfers

2 MiB transfers
128 KiB transfers
8 KiB transfers
512 B transfers

Higher is better
Dynamic Protocol Percent Direct Sends
Outstanding receives fixed at 32
Systems used: Intel Xeon 2.93 GHz X-5670 CPUs, 64 GB RAM, PCI-e Gen 2

HCAs: Mellanox ConnectX-2 10 Gbps RoCE HCAs

Distance simulated using Ixia ANUE Network Emulator introducing 48 ms round-trip delay
Throughput Over Distance
Outstanding sends and receives fixed at 32

Throughput vs. Message Size

Higher is better
Conclusions
Contributions

- Design of a set of algorithms for dynamically choosing between direct and indirect transfers in a byte-stream protocol
- Proof of correctness
- Implementation and testing of these algorithms within UNH EXS
- Performance evaluation
Questions?

Patrick MacArthur <pio3@cs.unh.edu>
Robert D. Russell <rdr@cs.unh.edu>

https://www.iol.unh.edu/services/research/unh-exs
References

Infiniband Trade Association,

D. Goldenberg, M. Kagan, R. Ravid, and M. S. Tsirkin,
“Zero copy sockets direct protocol over Infiniband—preliminary implementation and performance analysis,”


——, “A General-purpose API for iWARP and InfiniBand,” in the First Workshop on Data Center Converged and Virtual Ethernet Switching (DC-CAVES), Sep. 2009.
Interconnect Software Consortium in association with the Open Group,
“Extended Sockets API (ES-API) Issue 1.0,”
Backup
Memory to be used for RDMA must (currently) be \textit{registered}.

Sockets programmers make assumptions about memory used in I/O operations:
- Memory can be reused/freed as soon as send() returns
- Alignment of messages does not matter

The reality when using RDMA:
- `ibv_post_send()` only queues send operation; buffer still in use when it returns
- Not respecting adapter’s natural alignment can cause severe performance degradation, especially on FDR adapters.
exs_send() Matching Algorithms I

1:  while ¬EMPTY(qA) do
2:      A ← HEAD(qA)
3:      if PHASE_IS INDIRECT(Ps) \& (PA < Ps \lor SA < Ss) then
4:          if Ps < PA then
5:              Ps ← NEXT_PHASE(PA)
6:          end if
7:          throw away ADVERT A
8:      else
9:          if PHASE_IS INDIRECT(Ps) then
10:             Ps ← PA \triangleright Ps is now direct
11:          end if
12:      Ss ← Ss + lw
13:      send direct transfer
exs_send() Matching Algorithms II

14: \textbf{return}
15: end if
16: end while
17: if \neg \text{FULL}(b_s) then
18: if \text{PHASE\_IS\_DIRECT}(P_s) then
19: \hspace{1em} P_s \leftarrow \text{NEXT\_PHASE}(P_s) \hspace{1em} \triangleright \text{P}_s \text{ is now indirect}
20: end if
21: S_s \leftarrow S_s + l_w
22: b_s \leftarrow b_s - l_w
23: send indirect transfer
24: \textbf{return}
25: end if
Pre-Advertisement Sending Algorithm

1: if \( b_r > 0 \lor k_a > 0 \lor k_b > 0 \) then
2:     do not send ADVERT
3:     return
4: end if
5: if \( \text{PHASE\_IS\_INDIRECT}(P_r) \) then
6:     \( P_r \leftarrow \text{NEXT\_PHASE}(P_r) \) \( \triangleright \) \( P_r \) is now direct
7: end if
8: \( P_A \leftarrow P_r \)
9: \( S_A \leftarrow S'_r \)
10: if \( \text{MSG\_WAITALL} \) is set then
11:     \( S'_r \leftarrow S'_r + l_r \)
12: else
13:     \( S'_r \leftarrow S'_r + 1 \)
14: end if
Receive Transfer Processing Algorithm

1: if incoming transfer is direct then
2: \[ S_r \leftarrow S_r + l_w \]
3: if MSG_WAITALL was not set then
4: \[ S'_r \leftarrow S'_r + l_w - 1 \]
5: end if
6: do normal processing
7: else \(\triangleright\) incoming transfer is indirect
8: if PHASE_IS_DIRECT\((P_r)\) then
9: \[ P_r \leftarrow \text{NEXT}_\text{PHASE}(P_r) \]
10: end if
11: do normal processing
12: end if
Intermediate Buffer Copy Algorithm

1. copy data from stream buffer
2. send ACK to sender notifying of freed space
3. $b_r \leftarrow b_r - l_c$
4. $S_r \leftarrow S_r + l_c$
5. if advert sent and MSG_WAITALL was not set then
6. $S'_r \leftarrow S'_r + l_c - 1$
7. end if
Scenario 2
Sender Phase Increment on Discarding Advertisement

If sender phase were not incremented after processing A4-A6, would incorrectly match A6.