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

Acknowledgements

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

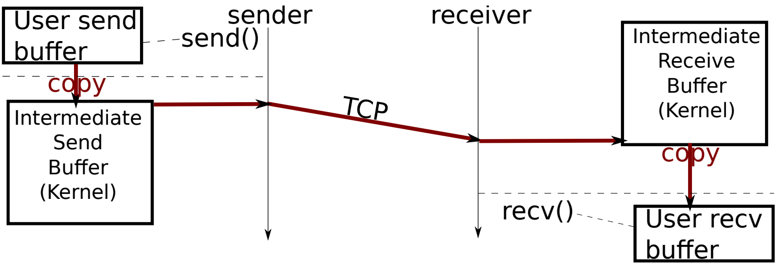
TCP (Transmission Control Protocol) Sockets

- Kernel involvement in all data transfers
- Buffered in kernel-space on both sides of connection
- Byte-stream oriented protocol
- Synchronous programming interface

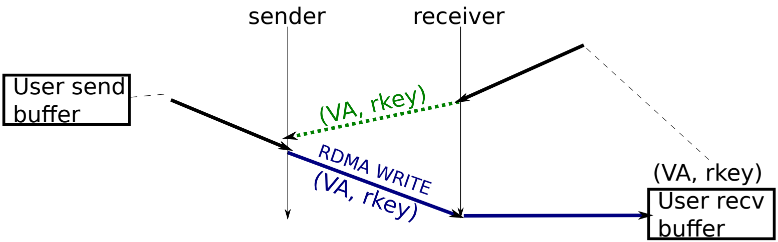
RDMA (Remote Direct Memory Access)

- "Kernel bypass": data transfers with no OS involvement
- "Zero-copy": Direct virtual memory to virtual memory transfers
- Message-oriented
- Asynchronous programming interface

TCP Sockets Data Transfer

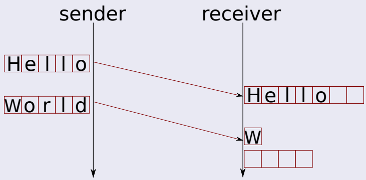


RDMA WRITE Data Transfer

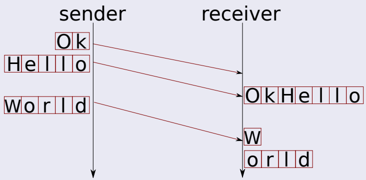


Message vs. Byte Stream Semantics

Message Transfer (UDP)



Byte Stream Transfer (TCP)



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]

UNH EXS (Extended Sockets)

- 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

UNH EXS Programming

Example

Example asynchronous send operation

```
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);
```

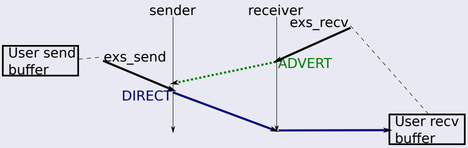
Dynamic Protocol

Motivation

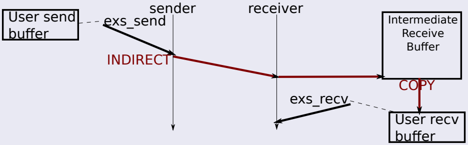
- Provide familiar **byte-stream** abstraction over RDMA
- **Dynamically** combine best aspects of zero-copy RDMA and fast send response benefit of TCP-style buffering.
- Deliver user data from sender to receiver **in order** with **no errors**
- Implementation of sender and receiver should be as **independent** as possible
- Work well over large **distance**
- Work **automatically** without user intervention

UNH EXS Dynamic Protocol

Direct Transfer (Streams and Messages)



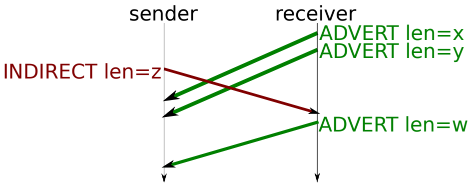
Indirect Transfer (Streams only)



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

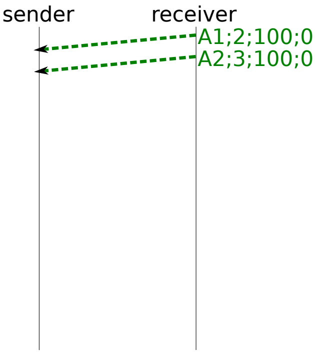


Dynamic Protocol

Late Advertisements: Solution

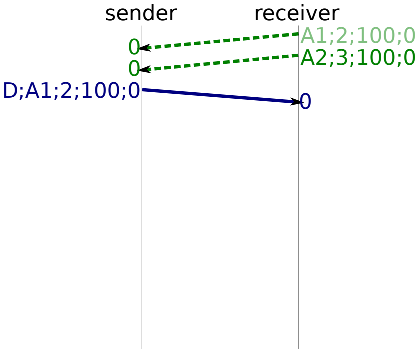
- Advertisement includes estimated **byte sequence number** and **phase number**
- Byte sequence number is estimated sequence number of first byte in stream satisfying associated `exs_recv()` request.
  - Each advertisement at receiver increases estimate by 1 (minimum transfer length)
  - Each transfer 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

Scenario



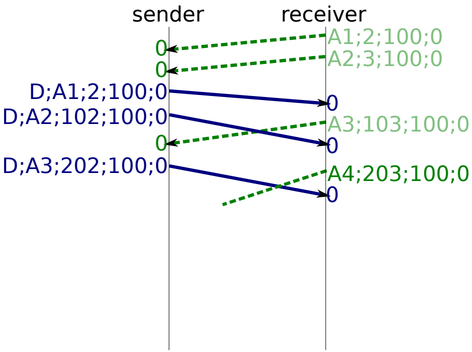
At start, sender has sent 2 bytes to receiver directly. Receiver issues two `exs_recv` requests.

Scenario

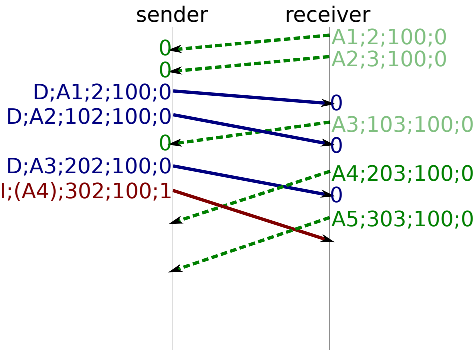


Sender issues single `exs_send` request.

Scenario

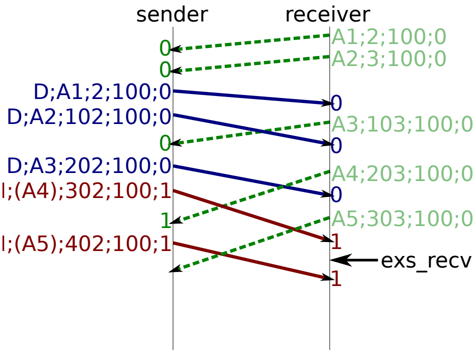


Sender and receiver continue to issue `exs_send` and `exs_recv` requests.



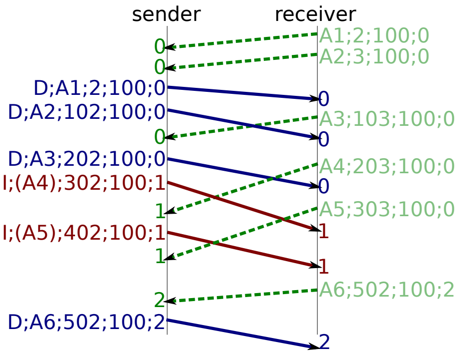
Advertisement A4 is delayed, so sender sends indirect transfer.

Scenario



Receiver **holds off** on sending off advertisements until all pending advertisements satisfied.

Scenario



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

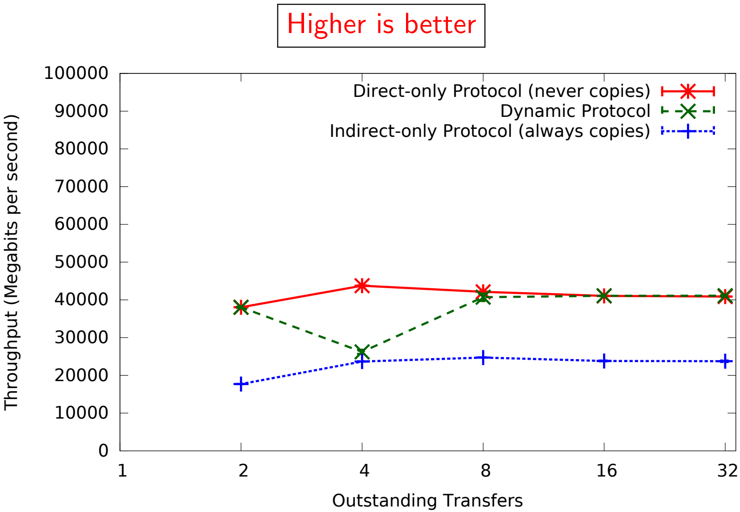
- Comparison of dynamic protocol with baseline protocols
  - **Direct-only** protocol: sender always waits for ADVERT (receiver never copies)
  - **Indirect-only** protocol receiver never sends ADVERTs (receiver always copies)
- Measure throughput, CPU usage, and percent of direct sends (average across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

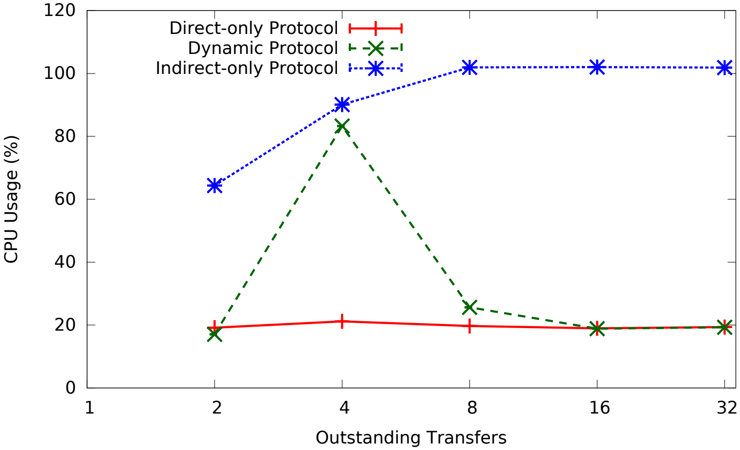
outstanding sends $= \frac{1}{2}$ outstanding receives



Dynamic Protocol CPU Comparison

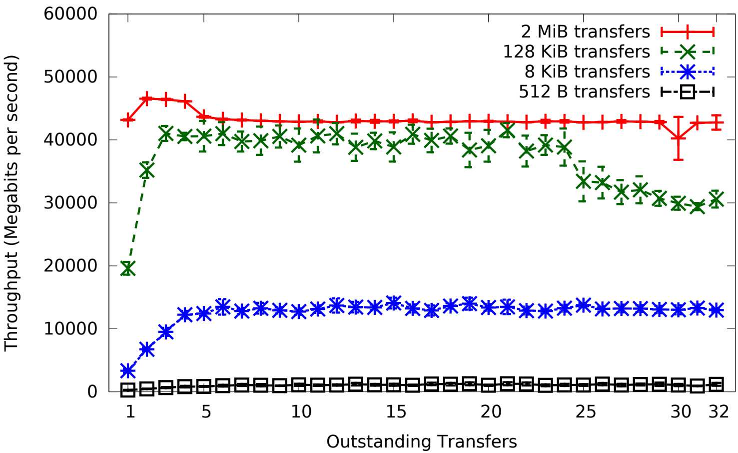
outstanding sends = $\frac{1}{2}$ outstanding receives

Lower is better



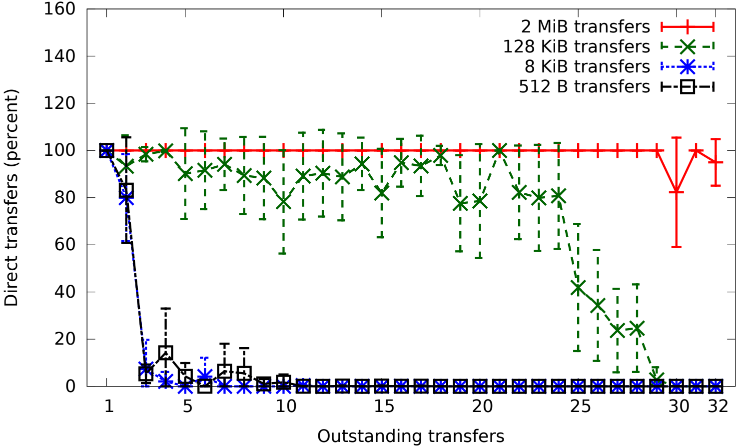
Dynamic Protocol Throughput
Outstanding receives fixed at 32

Higher is better



Dynamic Protocol Percent Direct Sends

Outstanding receives fixed at 32

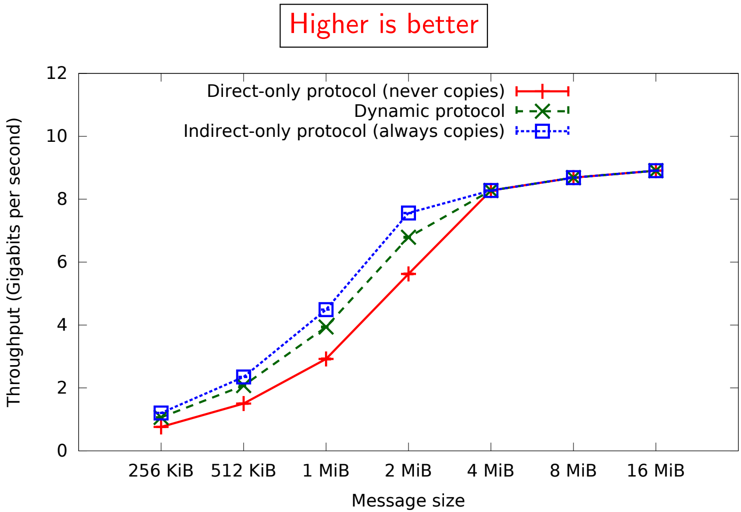


Distance test

- 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



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

Thanks!

Questions?

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

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

References

References I

Infiniband Trade Association,
"Supplement to Infiniband Architecture Specification Volume 1, Release 1.2.1: Annex A4: Sockets Direct Protocol (SDP),"
Oct. 2011.

D. Goldenberg, M. Kagan, R. Ravid, and M. S. Tsirkin,
"Zero copy sockets direct protocol over Infiniband—preliminary implementation and performance analysis,"
in *High Performance Interconnects, 2005. Proceedings. 13th Symposium on.* IEEE, 2005, pp. 128–137.

References II

P. Balaji, S. Bhagvat, H.-W. Jin, and D. K. Panda,
"Asynchronous zero-copy communication for synchronous sockets in the sockets direct protocol (SDP) over InfiniBand,"
in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International.* IEEE, 2006, pp. 8–pp.

Y. Lin, J. Han, J. Gao, and X. He,
"uStream: a user-level stream protocol over InfiniBand,"
in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on.* IEEE, 2009, pp. 65–71.

References III

S. Hefty,
Rsockets.
Available: https://www.openfabrics.org/ofa-documents/doc_download/495-rsockets.html

Interconnect Software Consortium in association with the Open Group,
"Extended Sockets API (ES-API) Issue 1.0,"
Jan. 2005.

——,
"A General-purpose API for iWARP and InfiniBand,"
in *the First Workshop on Data Center Converged and Virtual Ethernet Switching (DC-CAVES)*, Sep. 2009.

References IV

Interconnect Software Consortium in association with the Open Group,
"Extended Sockets API (ES-API) Issue 1.0,"
Jan. 2005.

Backup

Issue: Implementing "Zero-copy" Sockets

- Memory to be used for RDMA must (currently) be **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

while $\neg\text{EMPTY}(q_A)$ **do**
    $A \leftarrow \text{HEAD}(q_A)$
    if $\text{PHASE\_IS\_INDIRECT}(P_s) \wedge (P_A < P_s \vee S_A < S_s)$ **then**
        if $P_s < P_A$ **then**
            $P_s \leftarrow \text{NEXT\_PHASE}(P_A)$
        end if
        throw away ADVERT A
    else
        if $\text{PHASE\_IS\_INDIRECT}(P_s)$ **then**
            $P_s \leftarrow P_A$ ▷ P_s is now direct
        end if
        $S_s \leftarrow S_s + l_w$
        send direct transfer

exs_send() Matching Algorithms II

```
        return
    end if
end while
if ¬FULL(b_s) then
    if PHASE_IS_DIRECT(P_s) then
        P_s ← NEXT_PHASE(P_s)                ▷ P_s is now indirect
    end if
    S_s ← S_s + l_w
    b_s ← b_s − l_w
    send indirect transfer
    return
end if
```

Pre-Advertisement Sending Algorithm

if $b_r > 0 \vee k_a > 0 \vee k_b > 0$ **then**
    do not send ADVERT
    return
end if
if $\textsc{phase\_is\_indirect}(P_r)$ **then**
    $P_r \leftarrow \textsc{next\_phase}(P_r)$ ▷ P_r is now direct
end if
$P_A \leftarrow P_r$
$S_A \leftarrow S'_r$
if MSG_WAITALL is set **then**
    $S'_r \leftarrow S'_r + l_r$
else
    $S'_r \leftarrow S'_r + 1$
end if

Receive Transfer Processing Algorithm

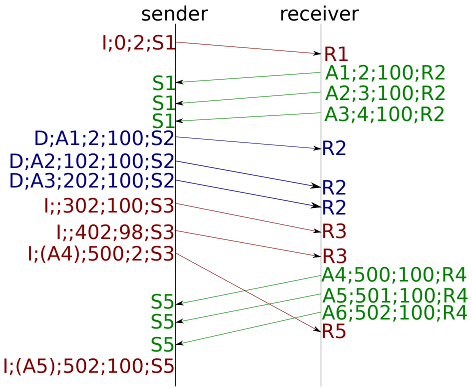
```
if incoming transfer is direct then
    S_r ← S_r + l_w
    if MSG_WAITALL was not set then
        S'_r ← S'_r + l_w − 1
    end if
    do normal processing
else                                ▷ incoming transfer is indirect
    if PHASE_IS_DIRECT(P_r) then
        P_r ← NEXT_PHASE(P_r)
    end if
    do normal processing
end if
```

Intermediate Buffer Copy Algorithm

copy data from stream buffer
send ACK to sender notifying of freed space
$b_r \leftarrow b_r - l_c$
$S_r \leftarrow S_r + l_c$
if advert sent and MSG_WAITALL was *not* set **then**
    $S'_r \leftarrow S'_r + l_c - 1$
end if

Scenario 2

Sender Phase Increment on Discarding Advertisement



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