SoftRDMA: Rekindling High Performance Software RDMA over Commodity Ethernet

Mao Miao, Fengyuan Ren, **Xiaohui Luo**, Jing Xie, Qingkai Meng, Wenxue Cheng

Dept. of Computer Science and Technology, Tsinghua University,
Background

• **Remote Direct Memory Access (RDMA)**
  - *Protocol offload*
    • reduce CPU overhead and *bypass kernel*
  - *Memory pre-allocation* and pre-registering
  - *Zero-Copy*
    • Data transferred directly from userspace

• Domain RDMA Network Protocols
  - InfiniBand (IB)
  - RDMA Over Converged Ethernet (RoCE)
  - Internet Wide Area RDMA Protocol (iWARP)
Background

• InfniBand (IB)
  - Custom network protocol and purpose-built HW
  - Lossless L2 uses hop-by-hop, credit-based flow control to prevent packet drops

• Challenges
  - Incompatible with Ethernet infrastructure
  - Cost much
    • DC operators need to deploy and manage two separate networks
Background

• RDMA Over Converged Ethernet (RoCE)
  - Indeed IB over Ethernet
  - Routability
    • RoCEv2 currently includes UDP and IP layers to actually provide that capability
  - Lossless L2
    • Priority-based Flow Control (PFC)

• Challenges
  - Complex and restrictive configuration
  - Perils of using PFC in a large-scale deployment
    • head-of-line blocking, unfairness, spreading congestion problems, etc
Background

• Internet Wide Area RDMA Protocol (iWARP)
  - Enables RDMA over the **existing TCP/IP**
    • Leverages TCP/IP for reliability and congestion control mechanisms to ensure scalability, routability and reliability
  - **Only NICs** (RNIC) should be specially built, no other changes required

• Challenges
  - Specially-built RNIC
Motivation

• Common *challenges* for RDMA deployment
  - Specific and non-backward compatible devices
    • Ethernet non-compatibility
    • Inflexibility of HW devices
  - Expensive and high cost
    • Equipment replacement
    • Extra burden of operation management

• *Is it possible to design software RDMA (SoftRDMA)* over commodity Ethernet devices?
Motivation

• Software Framework Evolution
  - High-performance packet I/O
    • Intel DPDK, netmap, PacketShader I/O (PSIO)
  - High-performance user-level stack
    • mTCP, IX, Arrakis, Sandstorm
  - Drive the technical changes
    • Memory resources pre-allocation and re-use
    • Zero-copy
    • Kernel bypassing
    • Batch processing
    • Affinity and prefetching
Motivation

• Much similar design philosophy between RDMA and novel SW evolvements
  - Memory resources pre-allocation
  - Zero-copy
  - Kernel bypassing

• Can we design SoftRDMA based on the high-performance packet I/O?
  - Comparable performance to RDMA schemes
  - No customized devices required
  - Compatible with Ethernet infrastructures
SoftRDMA Design: Dedicated Userspace iWARP Stack

- User-level iWARP + Kernel-level TCP/IP
- Kernel-level iWARP + Kernel-level TCP/IP
- User-level iWARP + User-level TCP/IP
SoftRDMA Design: Dedicated Userspace iWARP Stack

• In-kernel stack
  - Mode switching overhead
  - Complexity of kernel modification

• User-level stack
  - Eliminate mode switching overhead
  - More free space for stack design
One-Copy versus Zero-Copy

- **Seven steps** for Pkts from NIC to App
- **Two-Copy**
  - Step 6: copied from RX ring buffer for stack processing
  - Step 7: copied to different Apps’ buffer after processing
One-Copy versus Zero-Copy

• One-Copy
  - Memory mapping between kernel and user space to remove the copy in Step 7

• Zero-Copy
  - Sharing the DMA region to remove the copy in Step 6
One-Copy versus Zero-Copy

• Two obstacles for Zero-Copy in stack processing
  - Where to put the input Pkts for different Apps and how to manage them?
    • *Unaware* about the application-appointed *place* to store the input Pkts before stack processing
  - Whether the DMA region is large enough or could be reused fast to hold input packets?
    • The *DMA region* is *finite and fixed*, which could only store up to thousands of input packets

• SoftRDMA adopts **One-Copy**
SoftRDMA Threading Model

- Traditional multi-threading model \((c1)\)
  - One thread for App processing, the other for Pkts’ RX/TX
  - Good for throughput as batch processing
  - Higher latency as the thread switching and communication cost
SoftRDMA Threading Model

• Run-to-completion threading model \((c2)\)
  - Run all stages (Pkt RX/TX, APP processing...) into completion
  - Indeed improve the latency
  - Sophisticated processing may make the Pkt loss
SoftRDMA Threading Model

- SoftRDMA threading model \((c3)\)
  - One thread for Pkts’ RX, including One-Copy, the other for App processing and Pkts’ TX
  - Accelerate the Pkt receiving process
  - App processing and Pkts’ TX run within a thread to improve the efficiency and reduce the latency
## SoftRDMA Implementation

- 20K lines of code, **7.8K** are new
  - DPDK I/O
  - User-level TCP/IP based on lwIP raw API
  - MPA/DDP/RDMAP layer of iWARP

## RDMA Verbs

<table>
<thead>
<tr>
<th>Function</th>
<th>Key Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iwar_rnic_open()</td>
<td>index, mode, context, rnic_hdl</td>
<td>Open access to the abstract RNIC, initialize the memory management, QP/WR region and DDP/MPA layer</td>
</tr>
<tr>
<td>iwar_pd_allocate()</td>
<td>rnic_hdl, prot_id</td>
<td>Allocate a protection domain, attaching it to rnic_hdl</td>
</tr>
<tr>
<td>iwar_cq_create()</td>
<td>rnic_hdl, num_evt, cq_hdl</td>
<td>Create a completion queue with length num_evts</td>
</tr>
<tr>
<td>iwar_qp_create()</td>
<td>rnic_hdl, qp_attr, qp_id</td>
<td>Create the QP with QP attribute and insert it into the QP array of the RNIC</td>
</tr>
<tr>
<td>iwar_qmr_register()</td>
<td>rnic_hdl, buffer, length, pd_id, stag_index, mem_region</td>
<td>Register a region of memory in protection domain pd_id and create an STag (assumes the memory has already been allocated by users)</td>
</tr>
<tr>
<td>iwar_qp_passive_connect()</td>
<td>rnic_hdl, port, ipaddr, qp_id</td>
<td>Server passively waits for connection from remote side</td>
</tr>
<tr>
<td>iwar_qp_active_connect()</td>
<td>rnic_hdl, port, servername, qp_id</td>
<td>Client actively connects to remote server</td>
</tr>
<tr>
<td>iwar_qp_post_rq()</td>
<td>rnic_hdl, qp_hdl, rq_wr</td>
<td>Put a receive work request onto the receive queue</td>
</tr>
<tr>
<td>iwar_qp_post_sq()</td>
<td>rnic_hdl, qp_hdl, sq_wr</td>
<td>Post a send work request onto the send queue</td>
</tr>
<tr>
<td>iwar_cq_poll()</td>
<td>rnic_hdl, cq_hdl, wc</td>
<td>Poll the completion queue and return in the wc struct the status and type</td>
</tr>
</tbody>
</table>
SoftRDMA Performance

• Experiment config
  - DELL PowerEdge R430
  - Intel 82599ES 10 GbE NIC
  - Chelsio T520-SO-CR 10GbE RNIC

• Four RDMA implementation schemes
  - Hardware-supported RNIC (iWARP RNIC)
  - User-level iWARP based on kernel-socket (Kernel Socket)
  - User-level iWARP based on DPDK-based lwIP sequential API (Sequential API)
  - User-level iWARP based on DPDK-based lwIP raw API (SoftRDMA)
SoftRDMA Implementation

• Short Message (≤ 10KB) Transfer

- The close latency metric
  • SoftRDMA: 6.63us/64B  6.80us/1KB  52.20us/10KB
  • RNIC:   3.59us/64B  5.29us/1KB  16.27us/10KB

- The throughput falls far behind
  • Acceptable for short message delivering
SoftRDMA Implementation

• Long Message (10KB-500KB) Transfer

- The close latency metric
  • SoftRDMA: 101.36us/100KB  500.06us/500KB
  • RNIC: 93.45us/100KB  432.50us/500KB

- The close throughput performance
  • SoftRDMA: 1461.71Mbps/10KB  7893.31Mbps/100KB
  • RNIC: 8854.16Mbps/10KB  8917.44Mbps/100KB
Next work

• A more **stable and robust** user-level **stack**
• NICs’ **HW features utilized** to accelerate the protocol processing
  - TSO/LSO/LRO
  - Memory based scatter/gather for Zero-Copy
• More comparison experiments
  - Tests among SoftRDMA, iWARP NIC, RoCE NIC
  - Tests on 40GbE/50GbE devices
Conclusion

• **SoftRDMA**: a high-performance software RDMA implementation over commodity Ethernet
  - The dedicated userspace iWARP stack based on high-performance network I/O
  - One-Copy
  - The carefully designed threading model
Thanks!

Q&A