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, Beijing, China

ABSTRACT

Recent academic and industrial work is exploring the challenges of using RDMA over Ethernet, to support highly reliable, latency-sensitive services in today's datacenters. Previous work on the high-speed packet I/O like netmap, DPDK, etc., and high-performance user-level stacks like mTCP, IX etc., rekindles our inspirations to implement a high-performance software RDMA over commodity Ethernet devices.

This paper summarizes and explores the much-overlapped design philosophy between RDMA and the high-performance user-level stacks. Inspired by these, we design SoftRDMA, which is a user-level iWARP stack, based on One-Copy and deliberate threading model design. SoftRDMA's system implementation includes user-level iWARP/TCP/IP protocols and the DPDK packet I/O. No special hardware or software is required beyond. It provides the basic verbs of iWARP for RDMA communication. In our evaluation, SoftRDMA demonstrates comparable latency and throughput performance against the hardware-supported iWARP scheme. It achieves microsecond latency for short message and nearly full line rate for long message transfer.

CCS CONCEPTS

• **Networks** → **Network performance evaluation**; **Data center networks**; • **Hardware** → *Networking hardware*;

KEYWORDS

RDMA; Userspace stacks; High-speed packet I/O; DPDK

ACM Reference format:
Mao Miao, Fengyuan Ren, Xiaohui Luo, Jing Xie, Qingkai Meng, Wenxue Cheng. 2017. SoftRDMA: Rekindling High Performance Software RDMA over Commodity Ethernet. In *Proceedings of APNet'17, Hong Kong, China, August 03-04, 2017,* 8 pages.

APNet'17, August 03-04, 2017, Hong Kong, China

ACM ISBN 978-1-4503-5244-4/17/08…$15.00
https://doi.org/10.1145/3106989.3106995

https://doi.org/10.1145/3106989.3106995

1 INTRODUCTION

Remote Direct Memory Access (RDMA) technology has been introduced in some industrial datacenters [18], to provide ultra-low latency and high throughput for applications like FaRM [16] and Pilaf [24]. It provides direct memory-to-memory data transfer between servers [28]. The networking protocol is implemented entirely on NICs. This offloading significantly reduces CPU overhead and overall latency. Besides, the data transfer is done directly from userspace without trapping into the kernel, avoiding the system call and context switching overhead [5]. This allows data to be transferred in a Zero-Copy manner, avoiding the intermediate copies done by the kernel in traditional stacks. To simplify the stack, the protocol assumes a lossless networking fabric.

Today, there are three main network protocols which support RDMA: InfiniBand (IB) [9], Internet Wide Area RDMA Protocol (iWARP) [1], RDMA Over Converged Ethernet (RoCE) [10]. However, all these protocols face their own challenges on both technique and deployment aspects. IB is expensive and incompatible with Ethernet infrastructure. RoCE is complex and needs restrictive configuration. Priority-based Flow Control (PFC), the RoCE relied on, will lead to head-of-line blocking, unfairness, spreading congestion problems, etc [18]. iWARP seems better which is Ethernet-compatible and acceptable in performance. But the offloading schemes make all these RDMA protocols inflexible in protocol upgrades, limited in hardware resources for large connections, and needs extra support from OS vendors.

Recently, commodity hardware and software evolvements provide the capability and opportunity for RDMA to solve these challenges in a software way, which is performance-guaranteed, Ethernet-compatible, commodity NIC in hand being utilized without customized devices. For hardware: 10 GbE NICs are widely deployed in datacenters, which guarantee fairly low round-trip latencies. 40 GbE and 100 GbE technologies are right around the corner [12]. For software: recent works like mTCP [20], IX [11], Arrakis [25], etc., utilizes these hardware capabilities to innovate high performance packet I/O framework, including PacketShader's packet I/O engine (PSIO) [19], netmap [29], DPDK [3], etc. They all drive the important technical changes on packet processing:

(i) *Zero-Copy/One-Copy*; (ii) *Kernel bypass*; (iii) *Memory pooling, pre-allocation and re-use*; (iv) *Batch processing*; (v) *Affinity and prefetching*, etc. Benefited from the technical improvements and architectural optimizations, they achieve both high throughput and low latency in software approaches.

In this paper, we are inspired to rekindle the high-performance software RDMA implementation over commodity Ethernet (called SoftRDMA). We choose the dedicated user-space stack design for kernel bypass, which is different from the historical software iWARP implementations [14, 15, 22, 27]. In the threading model design, we compare the recent popular multithreaded model which was taken by mTCP and the traditional Linux kernel stack, the run-to-completion model IX took and finally propose our own threading model, which is considered to better trade off between throughput, latency, and the ease of implementations. We argue for One-Copy rather than Zero-Copy, analyze the working process of Two-Copy and sum up the suitable use cases for One-Copy and Zero-Copy. In Section 4, we implement the SoftRDMA prototype system, which includes iWARP (RDMAP/DDP/MPA), user-level TCP/IP and DPDK packet I/O. SoftRDMA is kernel-bypass, One-Copy, memory buffer pre-allocation and reuse of packets and metadata. In the implementation and deployment of SoftRDMA protocol on commodity servers, no special hardware or software are required beyond. It provides the basic verbs for RDMA communication.

Our tests in Section 5 show that SoftRDMA can achieve comparable latency and throughput performance to hardware-support RDMA scheme. SoftRDMA achieves 6.63 μs latency for 64B message transfer, comparing to 3.59 μs of RNIC. SoftRDMA keeps the same order of magnitude on latency transferring different sized messages, and over 8 Gbit/s throughput compared to 9 Gbit/s of RNIC when transferring > 100 KB sized of messages.

2 BACKGROUND AND MOTIVATION

This section first enumerates the technical details of domain RDMA frameworks and summarize the common design philosophy. Then, we list the much similar technical improvements in recent commodity hardware and software evolvements, which motivate us to rekindle the high-performance software RDMA over commodity Ethernet.

2.1 Domain RDMA Network Protocols

IB: Historically, IB supports RDMA natively from the beginning [9], which requires custom NICs and purpose-built switches. The link layer (L2) uses credit-based, hop-by-hop flow control to prevent packet loss, allowing the transport protocol (L4) to be quite simple and efficient. Obviously, IB network is not compatible with IP and Ethernet technologies, which makes it difficult to be deployed in modern datacenters. Besides, deploying and managing two separate networks cost much. Thus, the RoCE standard [10], and its successor RoCEv2 [2] have been proposed.

RoCE: RoCE is indeed IB over Ethernet, where the transport and network layers of IB is replaced by raw Ethernet encapsulation [6]. Long claimed to be routable, RoCEv2 currently includes UDP and IP to provide that capability. RoCEv2 relies on Priority-based Flow Control (PFC) to guarantee lossless networks. However, PFC is a coarse-grained mechanism. It operates at the port level, and does not distinguish between flows, resulting in problems like the potential for deadlock [21], head-of-line blocking, and unfairness[30], etc., which hinders its large scale deployment.

iWARP: iWARP enables RDMA over the existing TCP/IP infrastructure, which means it is fabric consolidation. Only NICs should be specially built (called RNIC). No other changes are required for the Ethernet equipment. Network administrators can use standard IP tools to manage traffic in an iWARP network, taking advantage of existing skill sets and processes to reduce overall cost and complexity. This is different from the other special-purpose interconnects like IB, RoCE and RoCEv2 above. Testing conducted shows that an iWARP-enabled 10GbE network is a credible competitor on performance against IB and RoCE fabric [4, 7, 26].

All three protocols share the common design philosophy to accelerate the protocol processing and reduce CPU involvement: (i) custom *protocol offloading* to NIC; (ii) *kernel bypass*; (iii) *Zero-Copy*; (iv) *memory buffer pre-allocation and pre-registering* at both ends. Although these three protocols could provide high throughput and ultra-low latency for data transfer, they all require customized hardware devices and have their respective challenges as mentioned above, when coming to be deployed in today's Ethernet-based datacenters.

2.2 Hardware and Software Evolvements

Recently, commodity hardware (HW) and software (SW) have evolved greatly, which brings important technical improvements on packet processing.

Hardware: 10 GbE devices are widely deployed in datacenters, which guarantee fairly low round-trip latencies. 40 GbE and 100 GbE technologies are right there [12]. NICs also bring the hardware support for multi-processor, multi-core servers [13], such as RSS/Flow Director, DCA, etc.

Software: Recent work like mTCP [20], IX [11], Arrakis [25], etc., utilizes these hardware capabilities to innovate high performance packet I/O framework, including PacketShader's packet I/O engine (psio) [19], netmap [29], DPDK [3], etc. They drive the common important technical changes on packet processing: (i) *Zero-Copy/One-Copy*; (ii) *kernel bypass*; (iii) *Memory pooling, pre-allocation and re-use*; (iv)

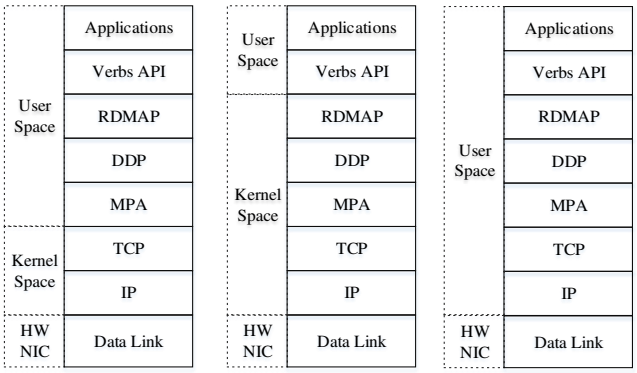


Figure 1: Different software iWARP design options based on TCP/IP protocol: (a) User-level iWARP + Kernel-level TCP/IP; (b) Kernel-level iWARP + Kernel-level TCP/IP; (c) User-level iWARP + User-level TCP/IP

Batch processing; (v) *Affinity and prefetching*, etc. Benefited from the technical improvements and architectural optimizations, they can achieve both high throughput and low latency in software approaches.

These HW/SW evolvements mostly share the similar design philosophy with RDMA frameworks, which inspire us to rekindle the design of software RDMA implementation over commodity Ethernet devices. We are motivated to design and implement SoftRDMA system, in order to explore this opportunity and capability. Our SoftRDMA should follow these design goals: (i) Providing comparable latency and throughput performance to hardware schemes; (ii) No customized hardware devices required; (iii) Be compatible with the commodity IP-based infrastructures.

3 SOFTRDMA DESIGN SPACE

This section describes the design of SoftRDMA in details. We explain why we choose the userspace iWARP as our approach, and why we take One-Copy instead of Zero-Copy. Besides, we pay more attention to the threading model design for better performance balance.

3.1 Dedicated Userspace iWARP Stack

Historically, research work on different software iWARP design combinations includes : building user-level iWARP (including RDMAP, DDP and MPA layers of iWARP) based on the kernel-level TCP/IP sockets as Fig. 1(a) [14] presents, building kernel-level iWARP on kernel-level TCP/IP sockets shown in Fig. 1(b) [15] and even building user-level iWARP on SCTP and UDP sockets [22, 27]. But the in-kernel stack comes with significant costs. Applications have to spend much time executing within the kernel: delivering interrupts, demultiplexing and copying packets [20]. The system calls also lead to context switching overheads which happen at every kernel crossing.

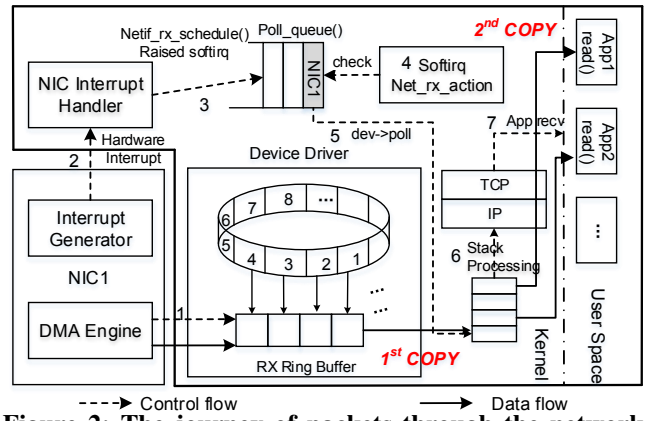


Figure 2: The journey of packets through the network stack: from the NICs to server's applications

In this work, we are the first to build the user-level iWARP based on the user-level TCP/IP stacks as Fig. 1(c) shows. Of course, user-level stacks are not a novel concept. The TCP/IP stacks in mTCP [20] and Sandstorm [23] entirely run in userspace, in order to eliminate kernel crossing overheads, and optimize packet processing without incurring the complexity of kernel modifications. Accordingly, we choose the user-level stacks since they provide us lower overheads, higher performance, more free space and convenient for novel stack design. We build our SoftRDMA on our own user-level TCP/IP stack to bypass the kernel for higher performance.

3.2 One-Copy versus Zero-Copy

As Fig.2 presents, the commodity NIC and the device driver process input data as following steps [31]: 1) Packets are transferred from NIC to ring buffer through DMA; 2) NIC raises hardware interrupt; 3) Hardware interrupt handler schedules RX Softirq; 4) Softirq checks its corresponding CPU's NIC device poll-queue; 5) Softirq polls the corresponding NIC's ring buffer; 6) Packets are copied from its RX ring buffer for the stack processing. The corresponding slot in the ring buffer is reinitialized and refilled; 7) After the stack processing, the payloads of input packets are copied to different applications' buffer in an ordered way. During the 7 steps, the traditional stack finishes Two-Copy: the first copy in step (6) and the second in step (7). Through memory mapping and careful memory management, the second copy from kernel space to user space could be eliminated. Could the DMA memory region be shared to remove the first copy?

To answer this, we revisit the interaction process between NIC and driver. During the device initialization, the driver allocates RX buffer as NIC's DMA region. The driver writes addresses of each RX entries to NIC, and treats those addresses as a ring buffer. Every time NIC receives a packet, it writes to the next free entry. Because the ring buffer has a finite and fixed capacity, entries can be overridden if NIC

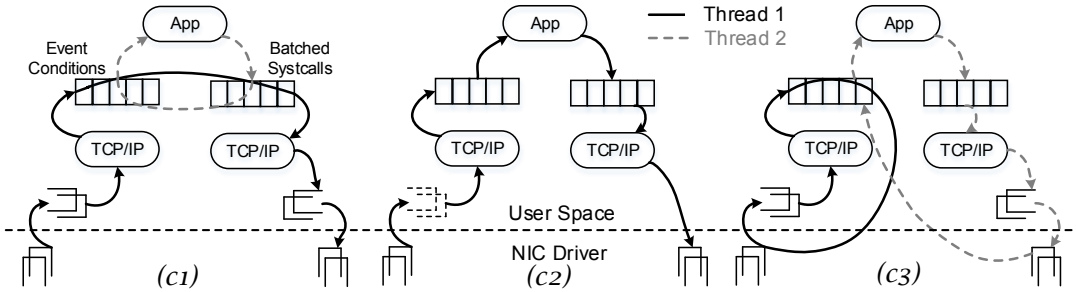


Figure 3: Different threading models that previous work employs

suffers from a packet flood (e.g. micro-burst). If the driver doesn't copy packets out of the ring buffer fast enough, NIC will lose packets due to lack of free DMA entries.

Thus, sharing the DMA region to support Zero-Copy should carefully consider two points: 1) Where to put the input packets for different applications and how to manage them? 2) Whether the DMA region is large enough or could be reused as fast as possible to hold input packets? During the interaction of the commodity NIC and driver above, NIC is unaware about the application-appointed place to store the input packet before stack processing. The DMA-enable region is also finite and fixed, which could only store up to thousands of input packets (e.g. The maximum ring length for Intel 82599 is 4096 [13]). Thus, the 1^{st} copy in Fig.2 is necessary, and should be done as quickly as possible.

For a fully offloading iWARP RNIC, the memory is preregistered by applications to notify the memory management modules, and after RNIC's protocol processing, NIC knows the place to copy the input packets to. Thus, Zero-Copy is achievable for RNIC. Among the recent high-performance software packet I/O, most frameworks like PSIO, netmap support One-Copy. But some others like DPDK, PF_RING DNA provide the Zero-Copy implementation. These Zero-Copy frameworks may accelerate the processing of some middleboxes or monitoring systems, which quickly operates on the input packets without complex processing. Actually, complicated middlebox functions or stack processing on commodity NICs still need the unavoidable 1^{st} copy [17].

For these reasons, SoftRDMA stack is One-Copy, which is the same with mTCP [20] design, but different from IX [11] which takes Zero-Copy for ultra-low latency.

3.3 Threading Model Design

The threading model design of stacks closely relates to the system's throughput and latency. We list some models that previous work employs and analyze their features.

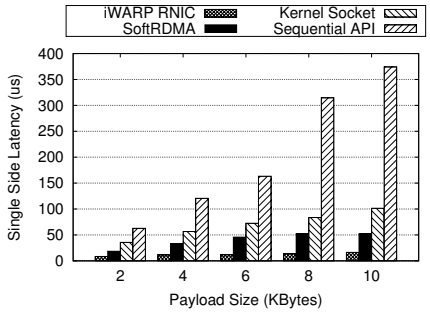
As Fig.3(c1) shows, a pair of threads are used in this model. One is responsible for packet processing in the application. The other is dedicated for packet receiving and sending. This model is good for throughput since it can process packets in batches. However, there are trade-offs between throughput and latency in this model. It incurs higher latency because of the thread switching and the communication cost via the aggressive batching. This model is adopted by mTCP [20], which has been demonstrated to handle millions of short message transactions per second. Besides, the similar multithreading model is also applied to the kernel stack and lwIP sequential API [8].

Fig.3(b) presents the run-to-completion thread model. It runs to completion all stages needed to receive and transmit a packet, interleaving protocol processing and application processing at certain transition points. This indeed improves the latency of packet processing. But if the application processing is slightly more sophisticated and consumes more time, the iteration time for the run-to-completion thread may be too long to receive input packets. IX [11] adopts this thread model and its dataplane is optimized for both bandwidth and latency. It is designed around a native, Zero-Copy API that supports processing of bounded batches of packets. However, since IX supports Zero-Copy, the DMA memory region may be overflowed in this case, causing a lot of packet losses as Section 3.2 described.

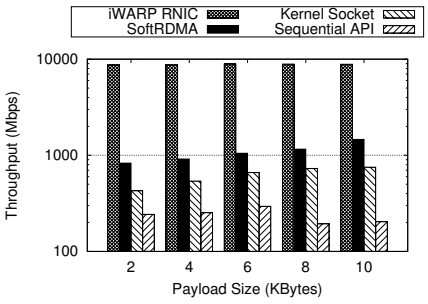
Summarizing previous work, we take the multithreading mode with a different design as shown in Fig.3(c3). One thread is responsible for packet receiving, which includes the 1^{st} copy of the input packets from DMA memory region to kernel memory, pushing packets into TCP/IP stack for processing and producing receiving events into the event queue. The other thread is responsible for application processing and the packet sending. The receiving thread could accelerate the packet receiving process, since packet receiving is more complex compared to sending. This allows more input packets to be fetched into the server's memory timely. The application processing and the sending run within a thread to improve the efficiency and reduce the processing latency. This is a better trade-off between throughput and latency.

4 SOFTRDMA IMPLEMENTATION

We implement SoftRDMA's prototype system with roughly 20K lines of C code (LoC), among which about 7.8K LoC are new and modified by us, which include DPDK-accelerated TCP/IP stack based on lwIP, MPA/DDP/RDMAP layer of

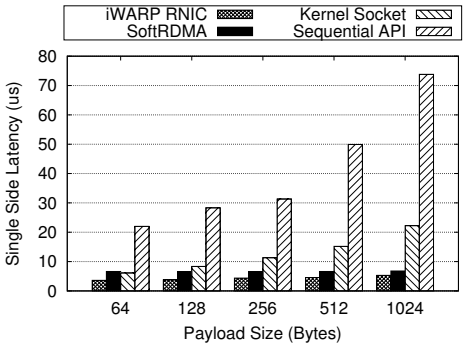


(a) $RTT/2$



(b) Throughput

Figure 4: Performance of four RDMA schemes to transfer $\leqslant$ 10K messages



(a) $RTT/2$

Figure 5: Latency of four RDMA schemes to transfer $\leqslant$ 1K messages

iWARP and the glue code among these layers. The whole software architecture is shown in Fig.1.

Verbs is the only interface into the iWarp stack that is accessible to the application. We highlight some of them. All the information is organized in the abstract RNIC structure, which could be allocated by `iwarp_rnic_open()`. The `iwarp_nsmr_register()` function is used to pre-register memory buffers for the connections. During the connection establishment period, `iwarp_qp_passive_connect()` and `iwarp_qp_active_connect()` functions will negotiate and decide whether to use marker or CRC for MPA layers based on initiator's criteria or responder's reply.

RDMA protocol (RDMAP) layer supplies communication primitives for verbs layer, including Send/Recv/RDMA Write/RDMA Read. Verbs layer work requests are delivered in order from RDMAP to low layers. The Send and RDMA Write operations require a single message for data transfer, while the RDMA Read needs a request by the consumer, followed by a response from the supplier. RDMAP is designed as a stream-based layer.

Direct Data Placement (DDP) layer allows the direct message transfer between user buffers in the application and the RNIC without intermediate buffering. DDP does this by segmentation. On sends, DDP splits the outgoing messages into small chunks that fit lower-level transport frames. On the receiving side, DDP reassembles these frames and places the data at the appropriate offset in the destination buffer. DDP places a small header in each outgoing message to specify the queue number, message number and offset.

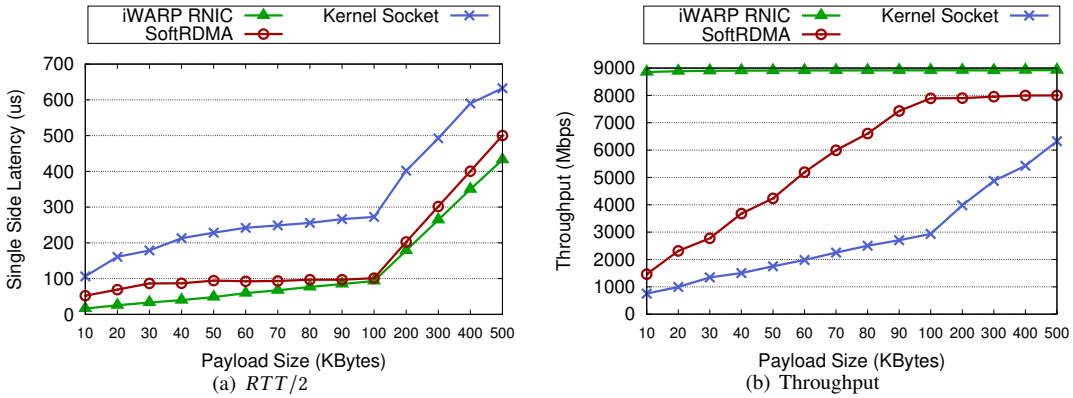
Marker PDU Aligned (MPA) protocol inserts markers into DDP data units before passing them to the TCP layer. It also reassembles marked data units from the TCP stream and removes the markers before passing them to the DDP layer. This new segment is known as Framing Protocol Data Unit (FPDU). The FPDU format has three essential changes: i) adding markers which point to the DDP header in case of middlebox fragmentation ii) adding CRC at the end of FPDU iii) sometimes adding segment pad bytes.

TCP/IP layer is based on the lwIP source code. We tried the sequential API and the raw API of lwIP separately, and finally chose the latter in the implementation. The sequential API provides an ordinary, sequential way to use the lwIP stack, which is similar to the BSD socket API. It is multithreaded like Fig.3(c1). The raw API allows the program execution to be event based by having callback functions called from within the TCP/IP code, which makes them run in the same thread. However, to program directly with the raw API is inconvenient, because functions are called from the bottom up which is quite different from the top-down software design and implementation way. But the threading model in Section 3.3(c3) provides a nice integration point to integrate the low-layer raw APIs and the upper-layer application program.

DPDK is a software development kit produced by Intel that allows direct userspace access to standard NICs. It uses the UIO modules to map the device I/O memory and interrupts into userspace in Linux. In SoftRDMA implementation, we utilize the DPDK library to implement *One-Copy* and bypass the kernel. It applies memory pre-allocation and re-usage, specifically, two memory regions are allocated: one for the packet data, and the other for its metadata. This reduces the memory allocation and deallocation costs for SoftRDMA. Besides of these, DPDK uses poll mode operations: the user application polls for new messages from the NIC rather than waiting for an interrupt. Integrating with the receiving loop thread of SoftRDMA in Section 3.3, the poll mode will further reduce the latency of packets receiving.

5 PERFORMANCE EVALUATION

Our experiment setup consists of two DELL PowerEdge R430 servers: one running as the SoftRDMA server; the other one as the client, connected via a direct cable through Intel 82599ES 10 Gbit NIC. We also use two Chelsio T520-SO-CR 10GbE



(a) $RTT/2$ (b) Throughput

Figure 6: Performance of four RDMA schemes to transfer 10K – 500K messages

RNICs to measure the hardware-supported iWARP's performance. These cards use PCIe v3.0 with 8 lanes, which provide enough bandwidth in both directions. Each server machines run on dual Intel Xeon E5-2620 v3 CPUs (6 cores per CPU), 64 GB of RAM. The clock speed is fixed at 2.3 GHz, and power conserving mechanisms and Hyper-Threading are disabled to make the measurements consistent and repeatable. CentOS 7.3 with kernel version 3.10.0 is used on all servers.

In the experiments, we compare the performance of four RDMA implementation schemes to transfer the messages with different sizes, including hardware-supported RNIC (denoted by ***iWARP RNIC*** legend), user-level iWARP based on kernel-socket (denoted by ***Kernel Socket*** legend), user-level iWARP based on lwIP sequential API (denoted by ***Sequential API*** legend) and user-level iWARP based on lwIP raw API which is our SoftRDMA implementation (denoted by ***SoftRDMA*** legend).

Short Message: We use the RDMA verbs to transfer $\leqslant$ 10K messages between the client and server, which is a Ping-Pong test. Fig.4 shows the throughput and latency to transfer short messages. From Fig.4(a), we could see SoftRDMA keeps the close latency metric (< 50 μs) with RNIC (< 20 μs) in the same order of magnitude, which is far less than the other two schemes. While the throughput in Fig.4(b) of SoftRDMA falls far behind compared to RNIC, as latency is the key performance index for short message delivering, we think this is acceptable. In details, Fig.5 shows the latency to transfer messages within 1K sizes. SoftRDMA achieves 6.63 μs to transfer 64B messages and 6.80 μs for 1024B messages, comparing to RNIC's 3.59 μs for 64B and 5.29 μs for 1024B. This demonstrates that SoftRDMA's One-Copy really consumes a little more than RNIC's Zero-Copy. Moreover, the latency to transfer 1024B-size message (1 copy-operation) only increases 2.5% compared to the one for 64B-size message transfer (1 copy-operation). However, the latency to transfer 10KB-size message (8 copy-operations) increases 86.28% compared to the one for 2KB-size message transfer (2 copy-operations). This illustrates that the cost for 1 copy-operation is fairly constant.

Long Message: In the experiments to transfer 10KB-500KB sized messages, we can see that SoftRDMA achieves both the low latency and increasingly higher throughput as Fig.6 shows, which is closer to the performance of RNIC. Fig.6(a) reflects that SoftRDMA always keeps the close latency with RNIC throughout the experiment, which is 52.19 μs for 10KB, 101.36 μs for 100KB, 500.06 μs for 500KB comparing to RNIC's 16.64 μs for 10KB, 93.45 μs for 100KB, 432.50 μs for 500KB separately. On the other hand, the throughput achieved by SoftRDMA increases from 1461.71 Mbps for 10KB messages transfer to 7893.31 Mbps for 100KB ones, and stays around 8000 Mbps afterward. Meanwhile, the throughput of RNIC only increases from 8854.16 Mbps for 10KB messages to 8917.44 Mbps for 100KB ones. Obviously, SoftRDMA keeps the close performance with RNIC for the long message delivering.

6 DISCUSSION

This section discusses three main things needed to be done next. Firstly, SoftRDMA would be based on a more stable and robust user-level stack instead of lwIP, which is designed for embedded devices, rather than server systems. Secondly, some common hardware features of NICs would be utilized to accelerate protocol processing furtherly, such as TCP checksum offload (TSO), large segmentation offload (LSO), and large receive offload (LRO). For NICs with memory based gather/scatter to/from the main memory, a truely *Zero-Copy* SoftRDMA version would be implemented and evaluated. We believe SoftRDMA can easily incorporate these features. Thirdly, more comparison experiments would be conducted between SoftRDMA, iWARP NIC and RoCE NIC. We also plan to deploy SoftRDMA on 40GbE/50GbE devices to check how does it scale with link rates of NIC.

7 CONCLUSION

In this work, we revisit the historical RDMA technologies including InfiniBand, iWARP and RoCE, compare their characteristics and challenges, and summarize the shared technologies these stacks adopt. Considering the struggle of RDMA in Ethernet integration and the overlapped design philosophy between RDMA and recent work on high-speed packet I/O frameworks and user-level network stacks embodied, we are motivated to rekindle the high-performance software RDMA implementation over commodity Ethernet.

We design SoftRDMA which takes the dedicated userspace iWARP stack roadmap to bypass the kernel, One-Copy to decrease memory traffic, the carefully designed threading model to make a better balance between throughput and latency. SoftRDMA achieves micro-second latency in the short message delivering tests and keeps both high throughput and low latency for long message transfer, which is comparable to the hardware-supported iWARP scheme.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the anonymous reviewers for their constructive comments. This work is supported in part by National High-Tech Research and Development Plan of China (863 Plan) under Grant No.2015AA020101, and Suzhou-Tsinghua Special Project for Leading Innovation.

REFERENCES

[1] Architectural specifications for RDMA over TCP/IP. http://www.rdmaconsortium.org.

[2] Infiniband Trade Association. RoCEv2. September 2014. https://cw.infinibandta.org/document/dl/7781.

[3] Intel DPDK. http://www.dpdk.org.

[4] Intel Ethernet 10 Gigabit iWARP Performance: PAM-CRASH, from ESI Group's Virtual Performance Solution. http://download.intel.com/support/network/sb/ethernetpamcrashwhitepaper_10_07_2015.pdf.

[5] Introduction to Remote Direct Memory Access (RDMA). http://www.rdmamojo.com/2014/03/31/remote-direct-memory-access-rdma/.

[6] iWARP: From Clusters to Cloud RDMA. http://www.chelsio.com/wp-content/uploads/resources/iWARP_Then_and_Now.pdf.

[7] iWARP Update: RDMA Over 40Gb Targets Data Center and Cloud Applications. http://www.chelsio.com/wp-content/uploads/resources/T5-iWARP-40Gb-Update.pdf.

[8] lwIP. http://savannah.nongnu.org/projects/lwip/.

[9] InfiniBand Trade Association et al. 2000. *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association.

[10] Infiniband Trade Association et al. 2010. RDMA over converged Ethernet. *Press release* (2010).

[11] Adam Belay, George Prekas, and Ana et al. Klimovic. 2014. IX: A protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 49–65.

[12] Gautam Chanda. 2012. The market need for 40 gigabit ethernet. *White Paper, Cisco* (2012).

[13] Intel 82599 10 GbE Controller Datasheet Revision 3.3. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82599-10-gbe-controller-datasheet.pdf.

[14] Dennis Dalessandro, Ananth Devulapalli, and Pete Wyckoff. 2005. Design and Implementation of the iWarp Protocol in Software.. In *IASTED PDCS*. 471–476.

[15] Dennis Dalessandro, Ananth Devulapalli, and Pete Wyckoff. 2006. iWarp protocol kernel space software implementation. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 8–pp.

[16] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. 2014. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 401–414. http://dl.acm.org/citation.cfm?id=2616448.2616486

[17] Brice Goglin. 2008. Design and implementation of Open-MX: High-performance message passing over generic Ethernet hardware. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 1–7.

[18] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 202–215. https://doi.org/10.1145/2934872.2934908

[19] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2011. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 195–206.

[20] E Jeong, Shinae Woo, Muhammad Jamshed, Haewon Jeong, and et al. Ihm. 2014. mTCP: a highly scalable user-level TCP stack for multicore systems. *Proc. 11th USENIX NSDI* (2014).

[21] Mark Karol, S Jamaloddin Golestani, and David Lee. 2003. Prevention of deadlocks and livelocks in lossless backpressured packet networks. *IEEE/ACM Transactions on Networking (TON)* 11, 6 (2003), 923–934.

[22] Patrick MacArthur. 2016. USIW: Design and Implementation of Userspace Software iWARP using DPDK. (2016).

[23] Ilias Marinos, Robert N.M. Watson, and Mark Handley. 2014. Network Stack Specialization for Performance. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. 175–186.

[24] Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Using One-sided RDMA Reads to Build a Fast, CPU-efficient Key-value Store. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. USENIX Association, Berkeley, CA, USA, 103–114. http://dl.acm.org/citation.cfm?id=2535461.2535475

[25] Simon Peter, Jialin Li, Irene Zhang, and et al. Ports. 2014. Arrakis: The operating system is the control plane. In *Proceedings of the 11th Symposium on Operating System Design and Implementation (OSDI'14)*.

[26] Mohammad J Rashti and Ahmad Afsahi. 2007. 10-Gigabit iWARP Ethernet: comparative performance analysis with InfiniBand and Myrinet-10G. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 1–8.

[27] Mohammad J Rashti, Ryan E Grant, Ahmad Afsahi, and Pavan Balaji. 2010. iWARP redefined: Scalable connectionless communication over high-speed Ethernet. In *High Performance Computing (HiPC), 2010 International Conference on*. IEEE, 1–10.

[28] Ro Recio, P Culley, D Garcia, J Hilland, and B Metzler. 2005. *An RDMA protocol specification*. Technical Report. IETF Internet-draft draft-ietf-rddp-rdmap-03. txt (work in progress).

[29] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O.. In *USENIX Annual Technical Conference*. 101–112.

[30] Brent Stephens, Alan L Cox, Ankit Singla, John Carter, Colin Dixon, and Wesley Felter. 2014. Practical DCB for improved data center networks. In *INFOCOM, 2014 Proceedings IEEE*. IEEE, 1824–1832.

[31] Wenji Wu, Matt Crawford, and Mark Bowden. 2007. The performance analysis of Linux networking–packet receiving. *Computer Communications* 30, 5 (2007), 1044–1057.