Exploring Low-latency Interconnect for Scaling Out Software Routers

Sangwook Ma
KAIST
Daejeon, Korea
Email: sangwook.ma@kaist.ac.kr

Joongi Kim
KAIST
Daejeon, Korea
Email: joongi@an.kaist.ac.kr

Sue Moon
KAIST
Daejeon, Korea
Email: sbmoon@kaist.edu

***Abstract*—We propose and evaluate RoCE (RDMA over Converged Ethernet) as a low-latency back-plane for horizontally scaled software router nodes. By exploring combinations of design choices in developing internal fabric for software routers, we select a set of parameters and packet I/O APIs that yield the lowest latency and highest throughput. Using the optimal settings derived, we measure and compare latency and throughput of an RoCE interconnect against Ethernet using a high-performance userspace network driver (Intel DPDK). Our comparison shows that RoCE keeps low latency in all packet sizes while it has throughput penalties for network workloads (e.g., small packet sizes). To mitigate throughput penalties imposed by guaranteeing low latency, we suggest a hardware-assisted, batched forwarding scheme based on scatter-and-gather functionality of RDMA-capable NICs. When forwarding ingress network packets, our scheme achieves comparable to or higher throughput then Ethernet at the cost of several microseconds of latency.**

I. Introduction

Software routers built on off-the-shelf commodity servers are attractive for their low cost and programmability. Recent advances in commodity hardware have further empowered off-the-shelf servers, allowing each server to process multi-10 Gbps workloads. In addition, approaches such as using GPU as co-processors [12], [18], [30] and processing packets in userspace [7], [26], [3] have also improved the performance of software router.

Clustering multiple servers via an interconnect scales out a software router beyond a single-box configuration. In a scaled-out software router, traffic flows through multiple nodes between an ingress and an egress. The number of hops and passes through the interconnect depends on the scaled-out router configuration, workload distribution, and forwarding policy [6], [32]. The drawback is that these additional hops increase the overall latency of packet processing to the multiple of the single-box latency.

A natural question to raise at this point is: how can we reduce this additional overhead in latency? In previous work of scaled-out software routers, the focus has been on the single-box latency improvement [6], [11], [23], [32]. Packet I/O batching decreases the PCIe transaction delay between a NIC and a host and we see great improvement in the amortized per-packet latency. However, it comes at a significant latency penalty as the batch size grows. For a poll-mode userspace Ethernet driver [7], we observe a 10 μsec single-hop latency for a 1500-byte packet. The latency increases to 73 μsec when the batch size increases to 32 (§III-D).

The other component in the overall latency is the interconnect. So far, standard Ethernet has been the norm de rigueur as an interconnect. Yet, between the nodes within a router cluster, there is no restriction on the physical layer technology. We should review other high-speed technologies to see if there is room for latency improvement.

In this work, we propose RDMA over Converged Ethernet (RoCE) as an internal fabric for building clustered software routers [2]. RDMA (Remote Direct Memory Access) allows a machine to access the application memory of a remote host with minimal involvement of the remote CPU [25]. RoCE is an RDMA implementation over commodity Ethernet links. RDMA is widely used in High Performance Computing (HPC) applications as it provides high throughput and low latency by avoiding extra data copying and processing overheads in the kernel stack. However, using RDMA has additional complexities compared to the traditional socket interface because it provides many API options such as RDMA transport types and transfer operation types.

The contribution of the paper is twofold. First, we measure and compare throughput and latency of RoCE and Ethernet as a router interconnect. We find that RoCE maintains one-way latency below 3 μsec but shows lower throughput when the packet size is equal to or less than 1500 bytes (§III). Second, we propose hardware-assisted batching to enhance the throughput of RoCE interconnect. It makes NIC hardware to perform batched I/O on packets at different locations in the memory, eliminating the need to call software functions for I/O batching. This method exploits scatter-and-gather mechanisms provided by RDMA-capable NIC. Our measurement shows that the throughput increases up to 4.7 times with cost of several μsec when we apply the batching (§IV).

II. Background

A. Topologies of scaled-out software routers

There are three candidate architectures proposed to scale out software routers. The first model connects server nodes directly in a full mesh topology as in Figure 1(a). RouteBricks follows this model [6] and Valiant Load Balancing in traffic forwarding guarantee fairness in distributing packet processing workloads [29]. The second model connects all servers to a central hardware switch as in Figure 1(b). Use of a hardware

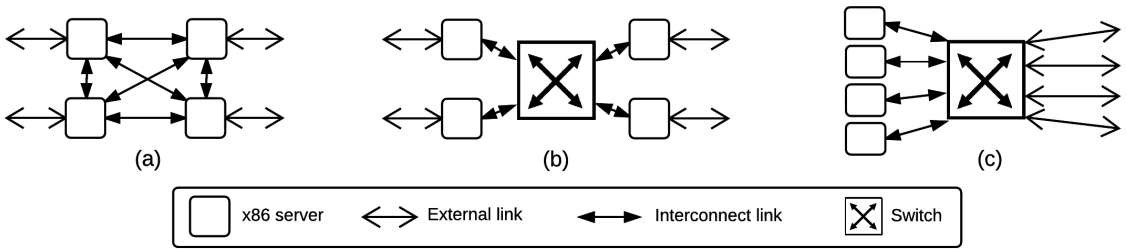

Fig. 1: Three models of scaling out a software router

switch reduces the number of hops and enables a simple topology. ScaleBricks implements this model [32]. RouteBricks has chosen the model Figure 1(a) over Figure 1(b), for it is cost-effective to use additional servers instead of an expensive hardware switch. ScaleBricks argues that the overall cost of the Figure 1(b) model is now cheaper than the full mesh model due to the recent switch cost reduction.

Greenhalgh et al. has presented a sketch of Figure 1(c) in their Flowstream platform [11]. E2 also chooses the model in Figure 1(c) [23]. It is similar to Figure 1(b) in the sense that a switch is used as an interconnect. The difference is that not all ports are for internal connection. Some switch ports directly connect to external networks, and the other ports are used for internal connection among x86 servers (Figure 1(c)).

There are two similarities in these models. First, they use Ethernet to interconnect software router nodes. Second, they assume that packets might go through several nodes to complete required processing. Although scaling out increases processing capacity of the whole system, increase in processing latency by these hops is inevitable.

B. Remote Direct Memory Access (RDMA)

RDMA is a transfer mechanism allowing a machine to access the application memory of a remote host [25]. It bypasses the kernel stack at the remote host and thus reduces user-kernel context switches and data copies. It has been closely coupled with InfiniBand and seen limited use in in high-performance computing due to its incompatibility with Ethernet and the high price of InfiniBand cards [1]. Recently, commodity Ethernet cards have started to support RoCE (RDMA over Converged Ethernet) and enabled users to run RDMA over Converged Enhanced Ethernet links [2], [9], [10].

In RDMA data transfer, user applications and RDMA-capable network cards (RNICs) work asynchronously. A user application posts *Work Requests* to a *Work Queue* using non-blocking functions, and checks if a *Completion Queue* contains any *Work Completions*. A RNIC takes out a request from the *Work Queue* and executes a data transfer. When it completes the transfer, it posts a *Work Completion* to the *Completion Queue* and takes out the next request. The pair of a *Work Queue* at the sender and a *Work Queue* at the receiver is called a *Queue Pair*.

RDMA provides three transport types for *Queue Pairs*: Reliable Connection (RC), Unreliable Connection (UC) and Unreliable Datagram (UD). RC and UC are connected transport types and they require a fixed connection between the send *Work Queue* and the receive *Work Queue*. They support message sizes up to 2GB by breaking a message into MTU-sized frames and reassembling them. In unconnected transport of UD, a *Work Queue* communicates with any other peer *Work Queue* without an explicit connection but the maximum size of a message is limited to the MTU size. Transport types are either reliable or unreliable. RC, which is a reliable transport type, guarantees lossless packet transfer by using ACKs and NACKs. Unreliable transports do not guarantee lossless transfer, but they consume less link bandwidth because they do not generate ACK and NACK packets.

In each *Work Request*, the application sets an operation code to designate the transfer operation the RNIC should perform. Transfer operations supported are: SEND, RECV, WRITE, READ, atomic fetch-and-add, and atomic compare-and-swap operations. Atomic operations act on 64-bit values and they are too small for packet transfer. Thus we do not consider them in this work. SEND and RECV operations work in pairs. First, a receiver posts a RECV work request specifying a memory region where the received message will be stored. A sender then makes the RNIC post a SEND work request to a *Work Queue* in its local memory. On the other hand, WRITE and READ operations are one-sided; a local host specifies a memory region on a remote host and the CPU of the remote host is not involved in data transfers. A WRITE operation "pushes" the contents of a local memory region to the memory region on the remote host. In a READ operation, a host "pulls" the contents of a remote host's memory region and puts it into a local memory region.

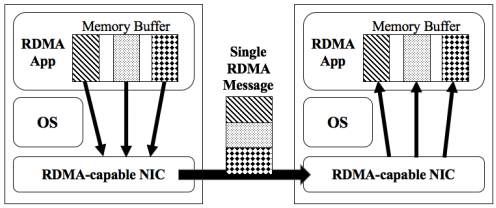

Fig. 2: An example of RDMA scatter-and-gather data transfer with three scatter-and-gather elements.

RNICs support the scatter-and-gather functionality to access discontinuous memory segments in a single *Work Request*. Applications post *Work Requests* with multiple scatter-and-

gather elements, each of which containing the address and the length of a local memory buffer segment. In SEND and WRITE operations, the RNIC assembles contents of multiple memory regions into a single RDMA message and sends it to a remote host. In RECV and READ operations, the RNIC separates the payload of a gathered RDMA message and writes each part into the designated memory regions (Figure 2).

WRITE and READ operations do not involve the remote CPU and are a popular means of data transfer in high-throughput key-value storage systems in today's datacenters. They are not suitable for latency-sensitive applications such as our packet processing routers. Thus we only consider SEND and RECV in our evaluation of RoCE.

III. RoCE as an Interconnect

In this section, we examine RoCE as an alternative interconnect technology to Ethernet for software routers. First, we decide on the transport type and transfer operation that are appropriate to packet transfer and achieves the highest throughput and the lowest latency (§III-C). Based on the chosen type and operation, we compare the throughput and the one-way latency of RoCE and Ethernet. (§III-D).

Item	Specification
CPU	Intel Xeon E5-2670v3
RAM	DDR4 32GB (2,133Mhz)
NIC	Mellanox ConnectX-3 (single-port 40GbE)

TABLE I: Server hardware specification.

A. Experiment Setup

We evaluate RoCE on a lab-scale setup consisting of two servers (Table I) and a switch in the middle. To compare Ethernet and RoCE under the same conditions, we use Mellanox ConnectX-3 network cards and the Mellanox SX1036 switch, where line rates are 40 Gbps. To run Ethernet and RoCE applications, we use Intel DPDK (Data Plane Development Kit) v2.1.0 [7] and Mellanox OFED (OpenFabrics Enterprise Distribution) v3.0.2[1] on Ubuntu 14.04.

Figure 3 shows details of the software stacks including drivers and userspace libraries. Both RoCE and Ethernet use `libmlx4` and `libibverbs` libraries to bypass the kernel protocol stack. A major difference lies in the way they configure network cards to transfer packets using the `libibverbs` library. DPDK's userspace poll-mode driver uses an internal transport type named IBV_QPT_RAW_PACKET to send Ethernet packets one by one. On the other hand, our RoCE application uses either IBV_QPT_RC or IBV_QPT_UC service type to transfer RDMA messages using RC and UC transport types. Although the maximum RDMA MTU supported by `libibverbs` is 4096 bytes, our RoCE application has chosen to use 2048 bytes as MTU for technical difficulties on the Mellanox NIC.

[1]http://www.mellanox.com/page/products_dyn?product_family=26&mtag=linux_sw_drivers

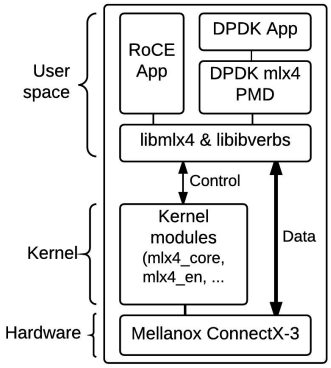

Fig. 3: Software stacks to use RoCE and DPDK Ethernet on a Mellanox ConnectX-3 NIC. PMD stands for a poll-mode driver.

Application programs that run on ConnextX-3 with the MTU size of 4096 cannot set up a queue pair and crash.

B. Measurement Methodology

We implement and use simple micro-benchmark programs to measure throughput and latency. They run symmetric tasks on each side of servers, sending and receiving RoCE/Ethernet messages in both directions that flow through the switch in the middle. To generate traffic and calculate throughput and latency, we use the pspgen-dpdk traffic generator [24]. All programs in the experiment run on a single CPU core to measure and compare isolated performance numbers.

The throughput reports in this paper include the protocol header sizes, 72 bytes for RoCE and 14 bytes for Ethernet, to demonstrate the actual number of bytes transmitted through the wire. The latency numbers are from application-level timestamps since RoCE and Ethernet expose different TX/RX APIs to applications. Specifically, RoCE applications use `ibv_post_send()` to transmit messages and `ibv_poll_cq()` to receive messages. Those APIs correspond to `rte_eth_tx_burst()` and `rte_eth_rx_burst()` of DPDK. To get the one-way latency, we use symmetric timestamping on both sender and receiver sides. First, the sender writes a timestamp (t_1) to the message just before calling the TX API. Then the receiver adds a timestamp (t_2) to the message just after returning from the RX API, and it again adds another timestamp (t_3) before sending the message back to the sender using the TX API. Finally, when the sender retrieves the message payload via the RX API, it takes a timestamp (t_4) and calculates the one-way latency by $((t_4-t_1)-(t_3-t_2))/2$.

C. Transport Type Selection

As mentioned in II-B, we only use SEND and RECV out of six operations. We exclude WRITE and READ operations since they are for one-sided communication without notifying the remote side while we want the receiver side to be notified immediately after WRITE in favor of low latency transports and forwarding. We also drop atomic operations due to their message size limits.

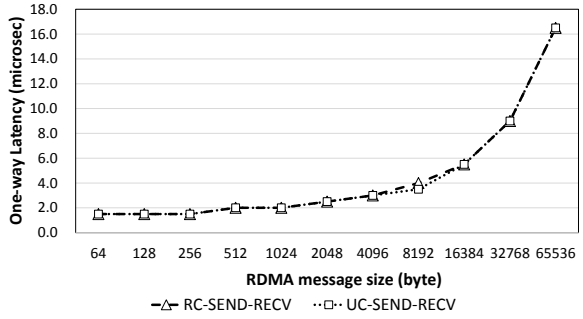

Fig. 4: Median one-way latency of RoCE RC and UC transport types.

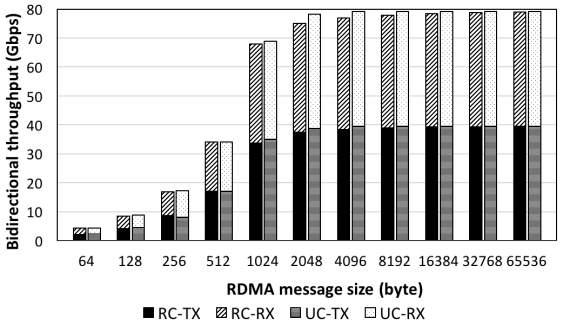

Fig. 5: Average throughput of RoCE RC and UC transport types. RX throughput is stacked up on top of that of TX for easy viewing.

Among the three RoCE transport types, RC guarantees reliable transfer and thus we include it in our evaluation. Kalia et al. report that there is no message loss even in unreliable transport types because RDMA runs on top of lossless link-level technologies such as Infiniband and Converged Enhanced Ethernet [16]. Hence we include UC in our evaluation. We exclude UD for its limitation on the maximum message size to an MTU. As the result, we compare the performance of RC and UC transport types using the SEND-RECV operation pair.

Figure 4 shows the latency measurement results. We vary the RDMA message size from 64 to 65,536 bytes and plot the median of measured latencies to mitigate the impact of sporadic outliers. We have observed little difference between average and median latencies. Latencies increase linear to the message size. We observe no significant difference between the two transport types.

Next, we plot the average throughput of RC and UC with the same range of message sizes in Figure 5. Although the difference is small, the throughput of the UC type is higher than that of RC in all message sizes. The source of throughput penalty in RC is the overhead of 74-byte-long ACK and NACK messages. The plot also shows that there is little asymmetry of throughput depending on the directions.

From the throughput and latency measurement experiments, we conclude that UC transport type combined with the SEND-RECV operation pair is most suitable for interconnects as UC offers higher throughput without latency and reliability compromises. From now on, we use it as the base configuration for performance comparisons against Ethernet.

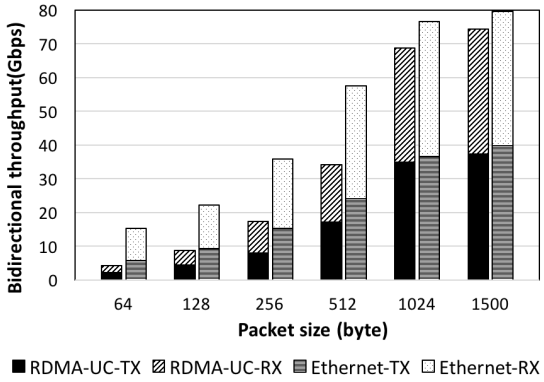

Fig. 6: Average throughput of Ethernet with I/O batch size of 32 packets and RoCE. RX throughput is stacked up on top of that of TX for easy viewing.

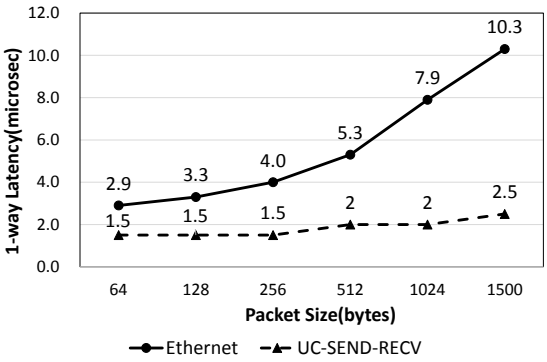

Fig. 7: Median one-way latency of Ethernet without I/O batching and RoCE.

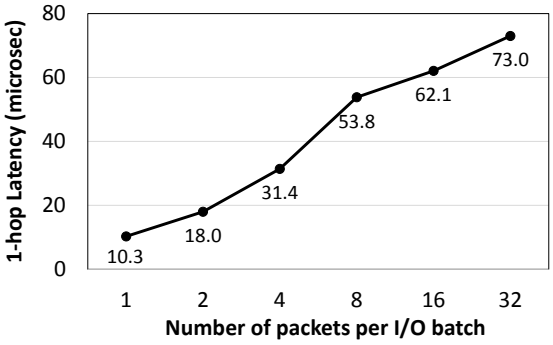

Fig. 8: Effect of I/O batching on median one-way latency of Ethernet. Packet size is 1500 bytes.

D. Performance Comparison against Ethernet

Now we compare the performance of RoCE and Ethernet in the range of typical network packet sizes (64 to 1500 bytes).

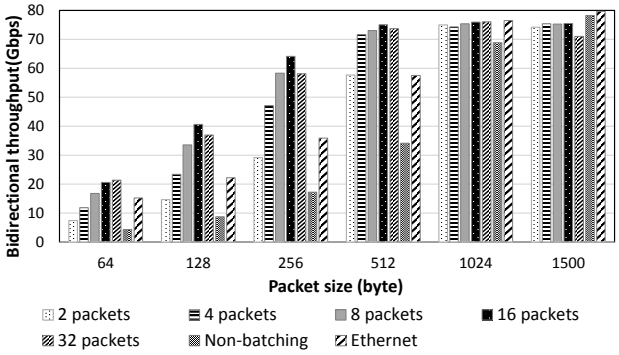

Fig. 9: Average bidirectional throughput of HW-assisted batching by packet size and the number of packets per batch.

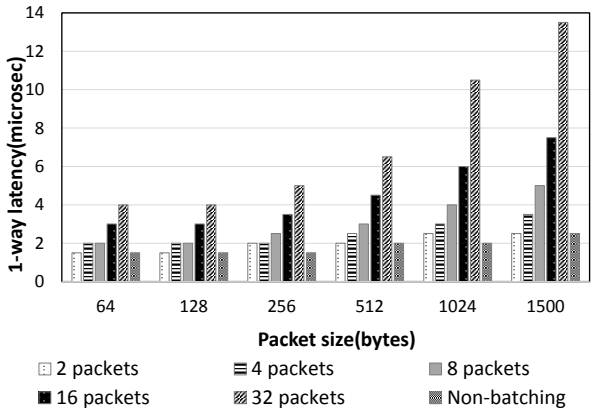

Fig. 10: Median one-way latency of HW-assisted batching by packet size and the number of packets per batch.

We measure the Ethernet throughput with packet I/O batching applied, since it is a de-facto standard technique to achieve high throughput in modern packet I/O libraries and packet processing applications [7], [12], [17], [18], [26]. Figure 6 shows that Ethernet performs better than RoCE for all packet sizes with the batch size of 32 packets. When the packet size is 256 bytes or smaller, the throughput of RoCE falls to less than half of that of Ethernet. The combined throughput of Ethernet TX and RX goes up to 80 Gbps with 1500-byte packets, reaching the physical limit of line rates.

Figure 7 shows the median one-way latency of RoCE and Ethernet. For fair comparison, we set the I/O batch size to 1 in Ethernet. RoCE keeps its median latency less than 3 μsec in all cases while Ethernet shows median latency of 3$\sim$10 μsec. Although the latency of Ethernet looks comparable to RoCE, we should not overlook the impact of I/O batching to latency. Figure 8 shows that in Ethernet the median one-way latency rapidly increases as the batch size grows, up to 73 μsec with 32 packets per batch, which is 7$\times$ of the case without batching and about 30$\times$ of RoCE that has no batching (Figure 7) when offered the same MTU-sized packets. As the latency of typical single-box hardware routers ranges below 29 μsec [5], such latency penalty imposed by batching is prohibitively expensive and must be eliminated when we build multi-node routers with an interconnect.

In summary, RoCE keeps lower latency compared to Ethernet but has lower throughput. If we could find a way to increase the throughput of RoCE without compromising the latency, RoCE could be an alternative interconnect technology in multi-node scaled-out routers.

IV. Hardware-assisted Batching

The remaining question from §III-D is: could we find a way to improve throughput without compromising the low latency for RoCE for small-sized packets (< 1500 bytes)? Since RoCE achieves near-line rates with large messages (Figure 6), can we send multiple small messages *inside* a single huge message.

There are multiple ways to implement such "packing" on a large message. The first method is to use copying functions such *memcpy()* to collect multiple packets into a single memory buffer. This method, however, adds latency due to the extra function calls and memory accesses by the CPU.

The second is to use the hardware-supported scatter-and-gather function (Figure 2) to split and merge multiple memory buffers from/to a single large message. We exploit it by making the NIC hardware to "pack" small messages into a one large frame, and name this method *hardware-assisted batching*. Specifically, user applications post work requests with multiple scatter-and-gather elements that each describes the start address and the length of Ethernet packets in the main memory. The potential disadvantage is that we loose some bandwidth due to an extra Ethernet header added to the merged message, but we expect that this bandwidth loss to be negligible in light of the throughput gain.

Using the same experiment setup and measurement methods as in §III, we evaluate the throughput and one-way latency while increasing the number of packets per batch (i.e. the number of scatter-and-gather elements per work request) up to 32. This is the maximum number of scatter-and-gather elements supported by Mellanox ConnectX-3 RNIC.

In Figure 9, we find that hardware-assisted batching enhances the throughput by the maximum of 4.7$\times$, reaching near-line rates even in packet sizes less or equal to 512 bytes. This is higher or close to the throughput of Ethernet with I/O batching in all packet sizes. Even before reaching the line rates in packet sizes equal to or less than 256 bytes, we observe that hardware-assisted batching consistently outperforms Ethernet by 3.7 to 4.8$\times$. In 1024 and 1500 byte packets, there is a slight drop compared to Ethernet with I/O batching. We suspect that the reason for this degradation is the increased memory access rates; RNIC has to access more memory regions when there are multiple scatter-and-gather elements in a work request, causing more PCIe bus traffic. In all batch sizes tested, the batch size of 16 delivers the highest throughput except for the packet size of 1500.

Figure 10 shows the one-way latency of RoCE with hardware-assisted batching. The latency linearly increases as the batch size and packet size grow. Note that the batch and

packet sizes increases by the power of two in our experiments. The latency goes up to 13.5 μsec with 1500 byte packets and the batch size of 32. This is still 5.4$\times$ lower than Ethernet with the same packet size and the batch size, and comparable to the latency of Ethernet with no I/O batching (Figure 7).

The above results support our claim that RoCE with the hardware-assisted batching is a viable alternative to Ethernet as an interconnect for high throughput and low latency.

V. Related Work

RDMA-based networked systems One major application of RDMA is accelerating the performance of existing systems or mechanisms where data to move is much larger than the Ethernet packet size. RDMA-based MPI implementations ([27], [19], [20]) show a significant improvement in performance and has been widely used in the HPC community. Other projects have proposed design and implementation of HBase [13], NFS [4], memcached [15] and HDFS [14] replacing the communication layer with RDMA.

New approaches are emerging to build a high performance key-value store optimized for RDMA from scratch. Pilaf uses RDMA READ operation for `GET` and SEND/RECV operation for `PUT` [21]. It proposes a self-verifying hash table structure to ensure consistency. HERDimproves throughput and latency by using WRITE operations over UC transport for requests and SEND operations over UD transport for responses, eliminating the overhead of posting RECV operation at the receiver side [16] . HydraDB provides high availability by supporting data replication using RDMA WRITE [31]. It brings the cache-friendly hash table to its NUMA-aware design and exploit the benefits of multicore systems. FaRM is a general-purpose distributed computing platform using RDMA that provides main memory of a serve cluster as shared address space [8] . It demonstrates its effectiveness by implementing key-value store and graph store on top of it.

RDMA-like interconnects for rack-scale computing Scale-out NUMA introduces a new hardware architecture and communication protocol to integrate multiple NUMA nodes into a single System-on-Chips [22]. Each node has a custom hardware called remote memory controller and a on-chip router and inter-node communication is basically a cache-to-cache data transfer. Applications use an RDMA-inspired programming model to access the memory of remote NUMA nodes. A send-and-receive simulation based on the architecture shows one-way latency lower than 6 μsec.

Marlin proposes a PCIe-based rack interconnect to combine the main memory of each server into a single global shared memory [28]. It exploits an RDMA-like inter-machine transfer mechanism and implements an Ethernet interface on top of it. Its prototype shows 8.5 μsec one-way latency and 19.6 Gbps TCP throughput.

Our work is different from existing body in that we focus on the use case of a router interconnect. Also we evaluate the scatter-and-gather feature for network packet batching.

VI. Conclusion

In this work, we evaluate RoCE (RDMA over Converged Ethernet) as an interconnect technology for a software router cluster in place of Ethernet. By comparing the performance of RoCE and Ethernet, we find that RoCE reduces the one-way latency by 30 times compared to Ethernet. However, we have seen throughput penalty when the packet size is 1500 bytes or smaller.

To improve the throughput without compromising the latency, we propose hardware-assisted batching, which utilizes the scatter-and-gather feature of RDMA-capable NICs to ”pack” multiple packets in a single large message. Our performance evaluation shows that the batching increases throughput by 3.7$\sim$4.7 times in packets smaller than 1500 bytes. We have also shown that it maintains latency under 14 μsec, which is still 5.4 times lower than the Ethernet configurations. In summary we conclude RoCE is a viable technology to replace Ethernet as a multi-node router interconnect.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2014007580). We thank Junhyun Shim for review on the structure of the paper and anonymous reviewers for their precious feedbacks.

References

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

[2] Infiniband Trade Association et al. Supplement to Infiniband architecture specification volume 1, Release 1.2. 1: Annex A16: RDMA over Converged Ethernet (RoCE), 2010.

[3] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast userspace packet processing. In *Proceedings of the 11th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE Computer Society, 2015.

[4] Brent Callaghan, Theresa Lingutla-Raj, Alex Chiu, Peter Staubach, and Omer Asad. NFS over RDMA. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, 2003.

[5] Performance-comparison testing of IPv4 and IPv6 throughput and latency on key Cisco router platforms. http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/enterprise-ipv6-solution/IPv6perf_wp1f.pdf. Page 25. Average latency of Cisco 7606 router.

[6] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP)*. ACM, 2009.

[7] Intel DPDK: Data Plane Development Kit. http://dpdk.org.

[8] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. Farm: fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2014.

[9] IEEE. 802.11Qau. Congestion notification, 2010.

[10] IEEE. 802.11Qbb. Priority based flow control, 2011.

[11] Adam Greenhalgh, Felipe Huici, Mickael Hoerdt, Panagiotis Papadimitriou, Mark Handley, and Laurent Mathy. Flow processing and the rise of commodity network hardware. *SIGCOMM Computer Communication Review*, 2009.

[12] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review*, 2011.

[13] Jian Huang, Xiangyong Ouyang, Jithin Jose, Md Wasi-ur Rahman, Hao Wang, Miao Luo, Hari Subramoni, Chet Murthy, and Dhabaleswar K Panda. High-performance design of HBase with RDMA over Infiniband. In *International Conference on Parallel & Distributed Processing Symposium (IPDPS)*. IEEE, 2012.

[14] Nusrat S Islam, Mohammad Wahidur Rahman, Jithin Jose, Raghunath Rajachandrasekar, Hao Wang, Hari Subramoni, Chet Murthy, and Dhabaleswar K Panda. High performance RDMA-based design of HDFS over InfiniBand. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012.

[15] Jithin Jose, Hari Subramoni, Miao Luo, Minjia Zhang, Jian Huang, Md Wasi-ur Rahman, Nusrat S Islam, Xiangyong Ouyang, Hao Wang, Sayantan Sur, et al. Memcached design on high performance RDMA capable interconnects. In *International Conference on Parallel Processing (ICPP)*. IEEE, 2011.

[16] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using RDMA efficiently for key-value services. In *Proceedings of the Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2014.

[17] Joongi Kim, Seonggu Huh, Keon Jang, KyoungSoo Park, and Sue Moon. The power of batching in the Click modular router. In *Proceedings of the Asia-Pacific Workshop on Systems (APSys)*. ACM, 2012.

[18] Joongi Kim, Keon Jang, Keunhong Lee, Sangwook Ma, Junhyun Shim, and Sue Moon. NBA (network balancing act): a high-performance packet processing framework for heterogeneous processors. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*. ACM, 2015.

[19] Jiuxing Liu, Weihang Jiang, Pete Wyckoff, Dhabaleswar K Panda, David Ashton, Darius Buntinas, William Gropp, and Brian Toonen. Design and implementation of MPICH2 over InfiniBand with RDMA support. In *Proceedings of the IEEE Parallel and Distributed Processing Symposium (PDPS)*, 2004.

[20] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K Panda. High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming*, 2004.

[21] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using one-sided RDMA Reads to build a fast, CPU-Efficient key-value store. In *USENIX Annual Technical Conference (ATC)*, 2013.

[22] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. Scale-out NUMA. *ACM SIGARCH Computer Architecture News*, 2014.

[23] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: a framework for NFV applications. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*. ACM, 2015.

[24] PSPGEN-DPDK packet generator. https://github.com/ANLAB-KAIST/pspgen-dpdk.

[25] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A remote direct memory access protocol specification. Technical report, 2007.

[26] Luigi Rizzo. netmap: A novel framework for fast packet I/O. In *USENIX Annual Technical Conference (ATC)*, 2012.

[27] Galen M Shipman, Tim S Woodall, Rich L Graham, Arthur B Maccabe, and Patrick G Bridges. Infiniband scalability in Open MPI. In *Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2006.

[28] Cheng-Chun Tu, Chao-tang Lee, and Tzi-cker Chiueh. Marlin: a memory-based rack area network. In *Proceedings of the 10th ACM/IEEE symposium on Architectures for Networking and Communications systems (ANCS)*. ACM, 2014.

[29] Leslie G Valiant and Gordon J Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 1981.

[30] Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis. GASPP: a GPU-accelerated stateful packet processing framework. In *USENIX Annual Technical Conference (ATC)*, 2014.

[31] Yandong Wang, Li Zhang, Jian Tan, Min Li, Yuqing Gao, Xavier Guerin, Xiaoqiao Meng, and Shicong Meng. HydraDB: a resilient RDMA-driven key-value middleware for in-memory cluster computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015.

[32] Dong Zhou, Bin Fan, Hyeontaek Lim, David G Andersen, Michael Kaminsky, Michael Mitzenmacher, Ren Wang, and Ajaypal Singh. Scaling Up clustered network appliances with ScaleBricks. In *Proceedings of the Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015.