Performance Isolation Anomalies in RDMA

Yiwen Zhang
University of Michigan

Juncheng Gu
University of Michigan

Youngmoon Lee
University of Michigan

Mosharaf Chowdhury
University of Michigan

Kang G. Shin
University of Michigan

ABSTRACT

To meet the increasing throughput and latency demands of modern applications, many operators are rapidly deploying RDMA in their datacenters. At the same time, developers are re-designing their software to take advantage of RDMA's benefits for individual applications. However, when it comes to RDMA's performance, many simple questions remain open.

In this paper, we consider the performance isolation characteristics of RDMA. Specifically, we conduct three sets of experiments – three combinations of one throughput-sensitive flow and one latency-sensitive flow – in a controlled environment, observe large discrepancies in RDMA performance with and without the presence of a competing flow, and describe our progress in identifying plausible root-causes.

CCS CONCEPTS

• **Networks → Network performance analysis**;

KEYWORDS

RDMA, performance isolation, fairness

ACM Reference format:
Yiwen Zhang, Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G. Shin. 2017. Performance Isolation Anomalies in RDMA. In *Proceedings of KBNets '17, Los Angeles, CA, USA, August 21, 2017,* 6 pages.
https://dx.doi.org/10.1145/3098583.3098591

1 INTRODUCTION

To deal with the growing application demands of ultra-low latency (e.g., in key-value stores [8, 13] and remote paging [9]) as well as very high bandwidth (e.g., in cloud storage [11, 16] and memory-intensive workloads [9, 19, 23]), cloud operators are aggressively deploying RDMA in their datacenters [10, 17, 25]. The intuition behind this is simple: RDMA can simultaneously provide both low latency and high bandwidth *without* noticeable CPU overhead.

Indeed, RDMA-based implementations of latency-sensitive applications have experienced order-of-magnitude improvements [8, 13] in latency (<10 μs) and throughput (tens of millions ops/second). Similarly, throughput-sensitive applications have been scaled to many customers without bottlenecking the CPU demands of the TCP/IP stack [9, 11, 25]. However, attaining these benefits is not always straightforward. Compared to TCP/IP, RDMA exposes a wider interface to applications with many parameters that require careful tuning on a case-by-case basis. For example, sending the same data from machine *A* to machine *B* can be accomplished by any of READ, WRITE, or SEND/RECEIVE operations over three different transport types, each combination with its own advantages, drawbacks, and constraints [14]. Consequently, the state-of-the-art in RDMA usage is primarily limited to optimizing individual applications.

Given the proliferation of cloud computing and large-scale RDMA deployments, RDMA-enabled latency- and throughput-sensitive applications are unlikely to run in vacuum; they must share the network with each other. Therefore, we ask a simple yet fundamental question: *what happens when multiple RDMA-enabled applications must share the network?*

Traditionally, RDMA has been used by the HPC community in special-purpose clusters over InfiniBand (IB). Because the typical application deployment model in these environments closely resembles static partitioning (e.g., all MPI processes are often gang scheduled), fine-grained sharing of RDMA networks or even a single link has not received as much attention from the HPC community [21, 22] as link and network sharing have received in commodity environments [4–7, 12, 20]. Although some recent work on deploying RDMA over IP/Ethernet-based datacenters have explored RDMA sharing between large flows [10, 17, 25], their primary focus has been on dealing with the vagaries of Priority-based Flow Control (PFC) in large networks.

In this paper, we summarize our key findings from a simple set of experiments. Specifically, we focus on three sharing scenarios on a single link: bandwidth sharing between two large flows, latency characteristics of two competing latency-sensitive flows, and sharing characteristics of a latency-sensitive application and a throughput-sensitive one. At a high level, performance isolation is relatively good among throughput-sensitive flows but poor among latency-sensitive flows. More specifically, fair sharing between throughput-sensitive flows depends primarily on the CPU; small messages (~1MB) fail to gain fair bandwidth if the CPU is not posting requests to the RNIC fast enough. Furthermore, in the latency-sensitive regime (i.e., flows using small messages), a smaller message size begets relatively better latency; however, two flows using the same message sizes can result in unpredictability. Finally, latency-sensitive flows lose both in terms of latency and throughput (messages per second) in the presence of large background flows.

KBNets '17, August 21, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5053-2/17/08…$15.00
https://dx.doi.org/10.1145/3098583.3098591

Table 1: RDMA Transport Types and Supported Verbs.

	SEND	RECEIVE	WRITE/WIMM	READ	ATOMIC
RC	✓	✓	✓	✓	✓
UC	✓	✓	✓		
UD	✓	✓			

2 BACKGROUND AND RELATED WORK

2.1 RDMA and Underlying Networks

RDMA allows direct access to the remote machine's memory without involving the CPU, cache, or operating system of either host. It is supported by multiple interconnects, including InfiniBand and RoCE (RDMA over Converged Ethernet). In this paper, we focus on InfiniBand-based RDMA.

RDMA Verbs. RDMA NICs expose the *verbs* API to applications for directly accessing the network interface using queue pairs (QP). Each QP consists of a send queue and a receive queue. A work request places a work queue element (WQE) on the appropriate work (send/receive) queue. The completion signal of verb operation can insert a work completion (WC) at the tail of the completion queue (CQ) associated with the QP. There are two transfer semantics for verbs: SEND and RECEIVE follow the channel semantic (similar to traditional TCP/IP); in contrast, READ, WRITE/WRITE-with-immediate (WIMM), and ATOMICs use the memory semantic.

Transport Types. Current RDMA implementations support three transport types: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). RC supports guaranteed-order packet delivery by enforcing packet acknowledgement. Connected transports (RC, UC) require one-to-one QP, while single QP can communicate with multiple QPs in connectionless transport (UD). Therefore, UD can offer better scalability to some applications [13–15]. However, not all transport types support all verbs (Table 1).

Network Layer in InfiniBand. The InfiniBand network layer uses Global Route Header (GRH) to route packets between subnets. The source and destination of a packet is identified by a 128-bit global ID (GID), which has a unique subnet ID and a globally unique device ID (GUID). The GRH of a packet contains both its source and destination GIDs. The packet is forwarded by routers until it reaches the target subnet.

Link Layer in InfiniBand. The link layer in InfiniBand handles packet forwarding and switching within a subnet. Each device is assigned with a 16-bit local ID (LID), which is used for addressing within a subnet. The destination LID of a packet is specified by its Local Route Header (LRH).

The link layer also supports QoS to traffic flowing through virtual lanes (VL) [2]. These VLs are separate logical links sharing a single physical link. The service level (SL) field in a packet is defined to ensure its QoS. Upcoming packets are mapped to the output VLs based on the SL-to-VL map set by the switch. Arbitration between different VLs is determined by a *two-priority-level weighted round robin* arbiter [1].

Flow control in InfiniBand is handled on a per VL basis. Contrary to TCP/IP, InfiniBand (also RoCE) applies a credit-based approach for lossless flow control [2]. Packets are not transmitted unless the receiver indicates it has enough buffer space, and no packet loss happens due to buffer overflow. Lossless flow control brings deterministic traffic flow and efficient bandwidth usage to InfiniBand.

2.2 RDMA-Enabled Applications

Due to its potential for ultra-low latency and high throughput to applications, RDMA [10, 17, 25] has received considerable attention in recent years. While the bulk of the work in this category focused on latency-sensitive applications such as key-value stores [8, 13] that require O(1) μs latency, RDMA-based communication layers are increasingly being used in throughput-sensitive applications as well [9, 11, 16, 19, 24]. However, regardless of target domains, they focus on optimizing independent applications' latency or throughput. We focus on performance isolation in scenarios where multiple applications must coexist and share RDMA networks.

2.3 Network Sharing

Sharing individual links is a classic networking problem with max-min fairness [12] being a well-established solution. Many mechanisms have been proposed over the years to achieve this goal – most notably weighted fair queueing (WFQ) [7] and its variants – that ensure at least $\frac{1}{n}$-th of a link's capacity to each of the n flows. While max-min fair sharing works well for throughput-sensitive elephant flows, latency-sensitive mice flows must rely on some form of prioritization to achieve performance isolation [4].

With the advent of cloud computing, the focus on link sharing has expanded to network sharing between multiple tenants with solutions ranging from static allocation to a variety of network-wide sharing mechanisms [5, 6, 20]. Mogul and Popa [18] have recently summarized many of them. Almost all of them – except for static allocation – deal with bandwidth isolation and ignore latency-sensitive flows.

Finally, in recent years, large-scale RDMA deployment over RoCEv2 [10, 17, 25] have received considerable attention because of its effectiveness in providing low-latency communication. The resulting RDMA congestion control algorithms primarily deal with issues arising from Priority-based Flow Control (PFC) and ensure fair sharing between throughput-sensitive elephant flows.

3 PERFORMANCE ISOLATION IN RDMA

Our focus in this paper is establishing a baseline understanding of link-level sharing in RDMA networks instead of considering more complicated tenant- or network-level sharing scenarios. We perform a set of experiments to highlight performance isolation characteristics of RDMA in three scenarios: (1) between two throughput-sensitive flows; (2) between two latency-sensitive flows; and (3) between a throughput-sensitive and a latency-sensitive flow. Throughout this section, we refer to large, throughput-sensitive flows as elephants and small, latency-sensitive flows as mice.

We performed our primary experiments on two machines with 56 Gbps InfiniBand NICs, where both are connected to the same InfiniBand switch. This allows us to avoid issues arising from path length asymmetry [25]. We also verified the results on newer generation hardware, where machines have 100 Gbps InfiniBand NICs

Table 2: Testbed Hardware Specifications.

(a) Primary Setup

Component	Specification
CPU	2× Xeon E5-2650v2 (8 cores, 2.6 Ghz)
DRAM	8× 8GB DDR-3 RDIMMs, 1.86 Ghz
IB NIC	Mellanox FDR ConnectX-3, 56 Gbps
IB Switch	Mellanox SX6036G, 4.032 Tbps

(b) Secondary Setup

Component	Specification
CPU	2× POWER8NVL (10 cores, 4.02 Ghz)
DRAM	16× 16GB DDR-3 DIMMs, 1.33 Ghz
IB NIC	Mellanox EDR ConnectX-4, 100 Gbps
IB Switch	Mellanox SB7700, 7 Tbps

(Mellanox ConnectX-4), and highlighted the differences in §3.7. Table 2 shows the hardware specifications of our testbeds.

3.1 Benchmarking Tool

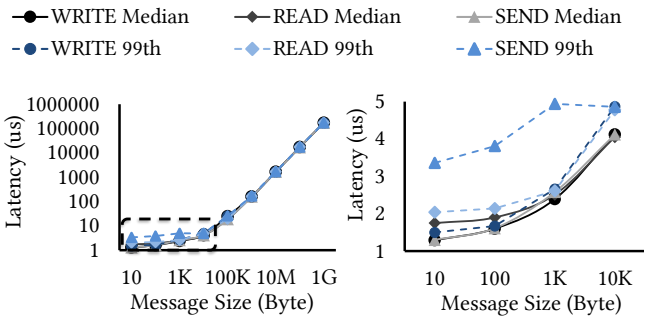
We implemented a benchmark tool[1] – based on *Mellanox Perftest* [3] – that can simultaneously run two RDMA flows with different configurations. Specifically, it creates two flows, each transferring a continuous stream of messages of a given size using RDMA verbs, and measures their bandwidth allocation and latency characterisitics when *both* flows are active. Across experiments, we consider the following RDMA design parameters.

RDMA Verbs and Transport. We compared WRITE, READ, WIMM, and SEND/RECEIVE verbs in our experiments. Consequently, we performed all experiments using RC connections to give all verbs a level playing field. Different types of verbs may impact fair sharing since it impacts on RDMA performance for different application scenarios [13, 14].

Event-Triggered Polling vs. Busy Polling. An application can choose to be notified of the completion of its RDMA verb operation either via interrupt/event or using busy polling. While busy polling occupies an entire CPU core, events cause context switching and hurt performance of latency-sensitive applications. However, as shown in Figure 3, event-triggered polling leads to significant CPU savings. In throughput-sensitive applications with large message sizes, event-triggered polling leads to negligible CPU usage. In latency-sensitive applications with smaller messages, the relatively higher CPU usage comes from frequent request posting instead of polling.

Message Acknowledgement. While SEND and WIMM require the receiver to explicitly post a receive request, other verbs use different acknowledgement methods such as busy polling of a specified memory address and zero-byte acknowledge messages. To focus on performance isolation between two RDMA flows, we remove acknowledgements and memory copy in different verbs for fair comparison. In our experiments, the next request will not be posted

[1]https://github.com/Infiniswap/frdma_benchmark



(a) Latency vs. message size **(b) Zoomed in for small sizes**

Figure 1: Median latencies for different RDMA verbs and message sizes.

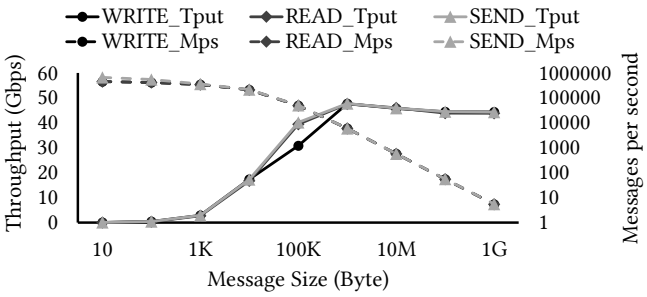


Figure 2: Throughput in terms of bandwidth (for elephants) and messages per second (for mice) for different RDMA verbs and message sizes.

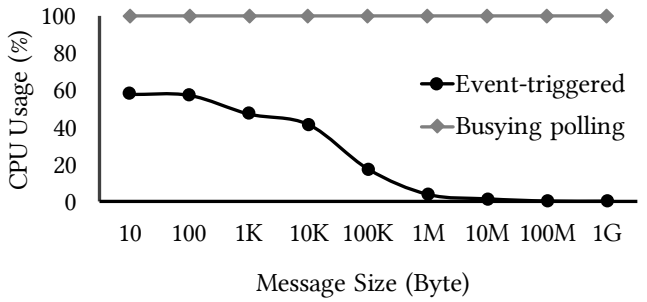


Figure 3: CPU usage of a RDMA WRITE flow for different message sizes.

to the send or receive queue using a single QP until the WC of the previous request is polled from CQ.

INLINE Message. The payload of an INLINE message can be inlined in its WQE buffer, eliminating the need to perform extra read for the payload. INLINE messages can be beneficial for latency-sensitive applications. We enabled INLINE messages for 10B and 100B in all latency-sensitive flows unless specified otherwise. The maximum INLINE data on the devices we can send is around 900 bytes. When INLINE-d data is too big, the CPU needs to write the send request onto the NIC using several rounds of memory-mapped

Table 3: Bandwidth Ratio (Left w.r.t. Top) of Two Event-triggered Elephants for Different Message Sizes.

		SEND		WIMM		READ		WRITE	
		1MB	1GB	1MB	1GB	1MB	1GB	1MB	1GB
WRITE	1GB	**1.41**	1.00	**1.44**	1.00	**1.39**	1.00	**1.40**	1.00
	1MB	1.02	**0.71**	1.00	**0.72**	0.99	**0.71**	1.00	
READ	1GB	**1.40**	1.00	**1.43**	1.00	**1.37**	1.00		
	1MB	1.08	**0.71**	1.04	**0.71**	1.00			
WIMM	1GB	**1.40**	1.00	**1.44**	1.00				
	1MB	1.00	**0.70**	1.00					
SEND	1GB	**1.41**	1.00						
	1MB	1.00							

I/O (MMIO), and that extra overhead becomes more expensive than the Direct Memory Access (DMA) read issued by the NIC [13]. In summary, using INLINE messages does not always reduce latency. Users need to verify the optimal INLINE data size in practice [14].

Request Pipelining. Pipelining SEND or RECEIVE requests can be useful when senders or receivers have a clear sense of how many messages they want to transfer. Unless otherwise specified, we disabled pipelining in the experiments.

Service Level (SL). Flows with a higher SL have higher priority. In theory, this can be useful to ensure QoS of starved flows or to prioritize latency-sensitive flows. However, SL is often not configurable by users in multi-tenant infrastructure; we failed to configure the service level from applications in three different RDMA clusters.

3.2 Defining an Elephant and a Mouse

The size of a message transferred in each RDMA operation determines whether the flow is throughput- or latency-sensitive. To clearly identify throughput- and latency-sensitive flows, we measured throughput, median and 99th-percentile latencies, and messages per second (MPS) for different message sizes (Figures 1 and 2). We observe that the latency for flows using message sizes smaller or equal to 1KB is similar, and flows using messages larger than 1MB are bandwidth-limited; message sizes in between 1KB and 1MB are harder to clearly classify.

Consequently, in this paper, we consider flows with individual messages up to 1KB to be latency-sensitive, mice flows, and those with individual messages larger than 1MB to be throughput-sensitive, elephant flows.

3.3 Elephant vs. Elephant

We compared two elephant flows by varying their verb types, message sizes, as well as the polling mechanism. Each flow transfers 1 TB data. All flows are running using our default setting with one exception: for SEND, the receiver needs to pre-post a small number of RECEIVE requests (we used 5 in this paper) to make SEND compatible with others.[2] We first explored the impact of message sizes on two elephant flows when both rely on event-triggered polling. Ratios of the two flows' bandwidth allocation – one using 1MB message size and the other 1GB – are shown in Table 3.

[2]However, increasing the number of pre-post RECEIVE requests does not help the SEND flow gain more bandwidth against others.

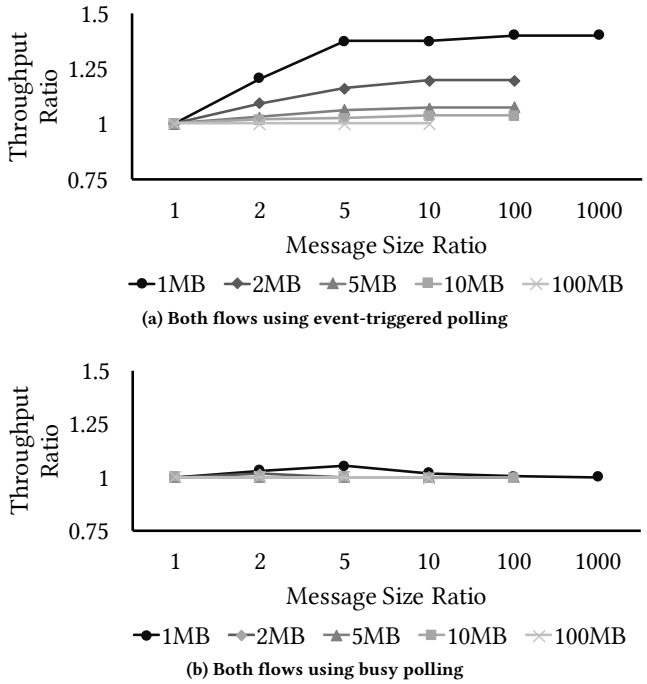


Figure 4: Ratio of bandwidth allocations of two elephant flows for increasing gaps between their message sizes for varying base flow sizes (1MB to 100MB).

We observe that when both flows are event-triggered, the message size has a large impact on bandwidth sharing between elephant flows. More precisely, when two flows use the same message size, the link bandwidth is equally shared; otherwise, the flow using the larger message size receives more bandwidth. In contrast, recall that the flow using 1MB messages achieved relatively higher bandwidth when running alone (Figure 2).

However, the verb type does not appear to affect fairness. As long as the message sizes equal, the bandwidth is equally shared no matter which verb we pick, assuming users would prepare a small receive pipeline when using SEND.

To further explore how differences in message sizes affect sharing, we measured the throughput ratio of different sizes of small base flow (from 1M to 100M) to competitors using varying message sizes (Figure 4a). Both flows use WRITE and transfer 1TB with event-triggered polling. When the base flow is small (e.g., 1MB), the ratio increases with message size ratio and saturates at a message size ratio of 5× and above. This implies that although bandwidth sharing is not fair, the amount of unfairness is perhaps bounded and relatively predictable. Also, this unfairness is alleviated for base flows with larger message sizes (5MB and above).

Next, we repeated the same experiments with busy polling for both flows (Figure 4b) and found it to eliminate unfairness. Moreover, we discovered that when a 1MB flow with busy polling competed with a 1GB flow with event-triggered pollling, their bandwidth was equally shared. When two 1MB flows – one using busy

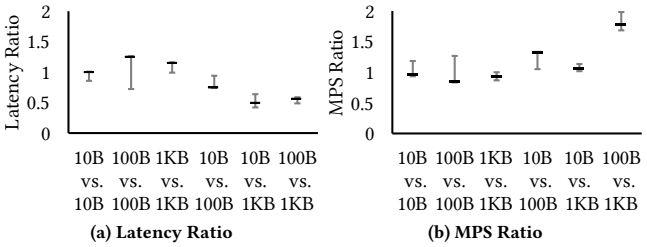


Figure 5: Ratio of median latencies and MPS between two mouse flows with different message sizes.

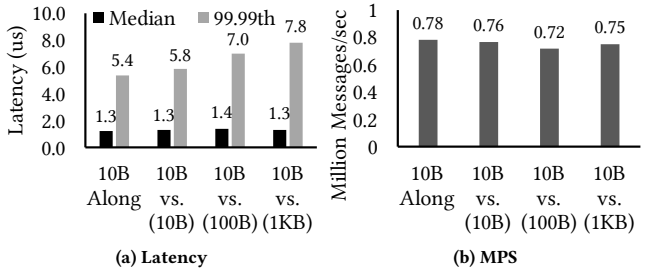


Figure 6: Latency and throughput of a 10B mouse flow when it is running alone and with another background mouse flow with different message sizes.

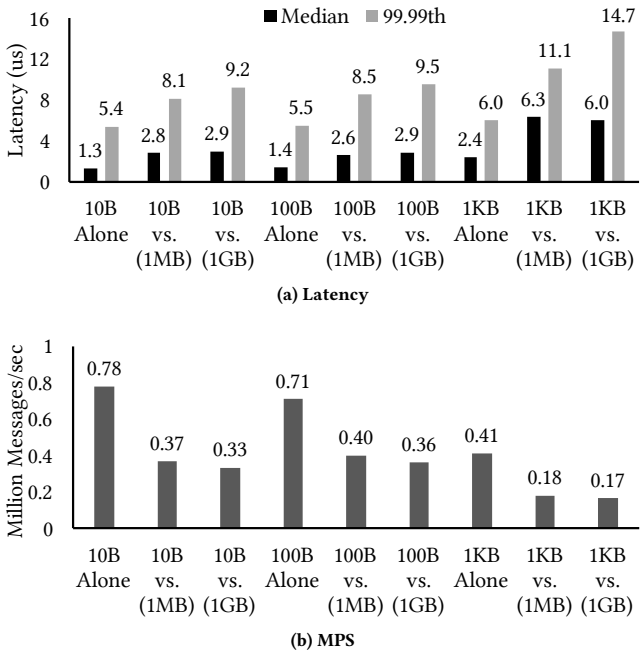


Figure 7: Latency and throughput of mouse flows w.r.t. elephant flows.

polling and the other event-triggered – competed, the former received 20% more bandwidth.

Summary of Observations

- When both flows are event-triggered, the larger-message flow receives more bandwidth; unfairness subsides when both use messages bigger than 5MB.
- When both flows use busy polling, isolation is maintained. It can also help gain more bandwidth when both flows use small messages (~1MB).
- RDMA verb type does not affect isolation as long as the receiver pre-posts a few RECEIVE requests when the sender transfers large data using SEND.

3.4 Mouse vs. Mouse

Next, we focused on latency-sensitive applications – specifically, on flows with three message sizes 10B, 100B, and 1KB. For each comparison, we took 5 runs and recorded the median latency and MPS. All flows used busy polling. Figure 5 shows the latency ratio and MPS ratio of two latency-sensitive flows sharing the same link. The bars in the graph indicate the range of median latency ratio out of the 5 runs. We observe that when both flows are less than 1KB, the sharing pattern is unpredictable. Even when both flows are using the same message size, the latencies are no longer similar; they are not predictable either. When there is a difference in the message size, all ratios appears to be below 1, indicating the flow with smaller message size runs faster.

Figure 6 compares the latency and MPS of a base flow of 10B message size running with a background mouse flow. In this plot, the target 10B flow was not stopped running until it sent 10 million messages using WRITE. All data points in the plot are taken from the median value among 5 runs. When the size of the competitor increases, the median latency of the 10B flow does not vary greatly, but the tail latency increases by up to 1.5×.

Summary of Observations

- Performance isolation is not guaranteed when two latency-sensitive flows with the same message size share a link (unlike throughput-sensitive flows).
- Latencies and MPS are stable when flows run alone but unpredictable when there is competition.

3.5 Elephant vs. Mouse

Finally, we focused on the performance variations of a latency-sensitive flow running with a background throughput-sensitive flow. All mouse flows are sending 10 million messages. We used busy polling for latency-sensitive flows and event-triggered polling for throughput-sensitive flows. Figure 7 plots the median and tail latencies as well as the MPS for a latency-sensitive flow running alone or accompanied by a 1MB or 1GB flow. From our measurements, the median latency of the latency-sensitive flow increases by around 2× and the MPS gets roughly halved when a throughput-sensitive flow is running in the background. Increasing the message size of the background flow further hurts the performance of the mouse, but the impact is relatively small. In contrast, the bandwidth

allocation of the background elephant flow remains unaffected in the presence of the latency-sensitive flow.

Summary of Observations

- Running small latency-sensitive flows under a big background flow loses *both* latency and MPS.
- Mice flows cannot be isolated from an elephant flow regardless of the message size the latter uses.
- The elephant flow is never affected by the mouse.

3.6 Dedicating CPU Cores Does Not Help

To determine whether the isolation issues are due to CPU contentions, we pinned each flow-generating process to its own core and re-ran our experiments. We confirmed that dedicated cores do not mitigate RDMA performance isolation anomalies.

3.7 Hardware is Not Enough for Isolation

We re-ran the same experiments on machines with 100 Gbps ConnectX-4 NICs to evaluate performance isolation on the newer generation hardware. The isolation problem in the elephant vs. elephant case still exists with a throughput ratio of 1.32. In the elephant vs mouse scenario, mouse flows are still affected by large background flows, where the median latency increases by up to 5×. In the mouse vs. mouse case the problem appears to be mitigated; we did not observe large tail-latency variations when two mouse flows compete.

4 DISCUSSION AND NEXT STEPS

It is too early to draw conclusions from the anomalies observed in this paper. Instead, we summarize our observations and hypothesize on the plausible reasons that may have caused those anomalies.

For throughput-sensitive elephants, we observed that the polling mechanism dictates bandwidth allocation: *how fast an application can post RDMA requests onto the RNIC is the only thing that matters in a throughput-sensitive environment.* This poses a tradeoff. On the one hand, to achieve high bandwidth and application-level pipelining (e.g., in streaming applications) with a reasonably large message size (1MB), one must dedicate CPU to achieve high bandwidth against other elephants. On the other hand, saving CPU via event-triggered polling can result in bandwidth loss.

For latency-sensitive mice, we observed little predictability between flows using equal-sized messages. However, when one is using small messages against another using a relatively larger mouse flow, the median latency of the smaller mouse appears to be stable. The tail latency, however, increases by up to 1.5×, and the MPS can sharply drop. One hypothesis is that in the latency-sensitive case, requests are posted fast and packets are not so easily blocked, thus causing more random behaviors.

Finally, in the presence of both types of flows, latency-sensitive flows suffer. In this scenario, the RNIC will be doing continuous DMA reads from the client's memory. The mouse flow posts requests more often, and its own packets may queue up in RNIC's queue buffer, thus causing degradation in latency and MPS.

Next Steps. We are currently studying the driver code for our RNICs to better understand how `libibverbs` interact with the RNIC and how RNIC interacts with the host's PCI Express and memory subsystems. We hope to be able to fix or mitigate some of the highlighted anomalies using software techniques. We also plan to expand our scope to larger scales – for example, in the presence of more than two flows, in scenarios where flows have different path lengths, and in RoCE environments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and feedback. This work was supported by National Science Foundation grants CNS-1617773, CCF-1629397, and CNS-1563095.

REFERENCES

[1] 2008. Infiniband Technology Overview. https://goo.gl/tyszb4. (2008).
[2] 2015. Infiniband architecture specification volume 1. https://cw.infinibandta.org/document/dl/7859. (2015).
[3] 2017. Mellanox Perftest Package. https://community.mellanox.com/docs/DOC-2802. (2017).
[4] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick Mckeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: Minimal Near-Optimal Datacenter Transport. In *SIGCOMM*.
[5] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards predictable datacenter networks. In *SIGCOMM*.
[6] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. 2016. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In *NSDI*.
[7] A. Demers, S. Keshav, and S. Shenker. 1989. Analysis and Simulation of a Fair Queueing Algorithm. In *SIGCOMM*.
[8] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. 2014. FaRM: Fast Remote Memory. In *NSDI*.
[9] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *NSDI*.
[10] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *SIGCOMM*.
[11] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure Coding in Windows Azure Storage. In *USENIX ATC*.
[12] Jeffrey M Jaffe. 1981. Bottleneck flow control. *IEEE Transactions on Communications* 29, 7 (1981), 954–962.
[13] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2014. Using RDMA efficiently for key-value services. In *SIGCOMM*.
[14] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. Design guidelines for high performance RDMA systems. In *USENIX ATC*.
[15] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. FaSST: fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *OSDI*.
[16] Haoyuan Li, Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. 2014. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *SoCC*.
[17] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *SIGCOMM*.
[18] Jeffrey C Mogul and Lucian Popa. 2012. What we talk about when we talk about cloud network performance. *SIGCOMM CCR* 42, 5 (2012), 44–48.
[19] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. 2015. Latency-tolerant software distributed shared memory. In *USENIX ATC*.
[20] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. 2012. FairCloud: Sharing the Network in Cloud Computing. In *SIGCOMM*.
[21] Adit Ranadive, Ada Gavrilovska, and Karsten Schwan. 2010. FaReS: Fair resource scheduling for VMM-bypass Infiniband devices. In *CCGRID*.
[22] Sayantan Sur, Matthew J Koop, Dhabaleswar K Panda, and others. 2007. Performance analysis and evaluation of Mellanox ConnectX InfiniBand architecture with multi-core platforms. In *IEEE Hot Interconnects*.
[23] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*.
[24] Fang Zheng, Hongbo Zou, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Jai Dayal, Tuan-anh Nguyen, Jianting Cao, Hasan Abbasi, Scott Klasky, Norbert Podhorszki, and Hongfeng Yu. 2013. FlexIO : I/O Middleware for Location-Flexible Scientific Data Analytics. In *IPDPS*.
[25] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *SIGCOMM*.