Understanding RDMA Microarchitecture Resources for Performance Isolation

Xinhao Kong and Jingrong Chen, *Duke University;* Wei Bai, *Microsoft;* Yechen Xu, *Shanghai Jiao Tong University;* Mahmoud Elhaddad, Shachar Raindel, and Jitendra Padhye, *Microsoft;* Alvin R. Lebeck and Danyang Zhuo, *Duke University*

https://www.usenix.org/conference/nsdi23/presentation/kong

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Understanding RDMA Microarchitecture Resources for Performance Isolation

Xinhao Kong Jingrong Chen Wei Bai[†] Yechen Xu[#] Mahmoud Elhaddad[†]
Shachar Raindel[†] Jitendra Padhye[†] Alvin R. Lebeck Danyang Zhuo

Duke University [†]*Microsoft* [#]*Shanghai Jiao Tong University*

Abstract

Recent years have witnessed the wide adoption of RDMA in the cloud to accelerate first-party workloads and achieve cost savings by freeing up CPU cycles. Now cloud providers are working towards supporting RDMA in general-purpose guest VMs to benefit third-party workloads. To this end, cloud providers must provide strong performance isolation so that the RDMA workloads of one tenant do not adversely impact the RDMA performance of another tenant. Despite many efforts on network performance isolation in the public cloud, we find that RDMA brings unique challenges due to its complex NIC microarchitecture resources (e.g., the NIC cache).

In this paper, we aim to systematically understand the impact of RNIC microarchitecture resources on performance isolation. We present a model that represents how RDMA operations use RNIC resources. Using this model, we develop a test suite to evaluate RDMA performance isolation solutions. Our test suite can break all existing solutions in various scenarios. Our results are acknowledged and reproduced by one of the largest RDMA NIC vendors. Finally, based on the test results, we summarize new insights on designing future RDMA performance isolation solutions.

1 Introduction

Multiplexing workloads from different tenants on a shared computing infrastructure enables the modern cloud computing era. The global cloud infrastructure revenue has already surpassed 400 billion US dollars and is forecast to grow to reach around 1 trillion US dollars in the next decade [7].

It is well known that having different tenants' workloads share computing resources can lead to unpredictable application performance interference [12, 18, 66] and privacy leakage [32, 39]. This drives plenty of studies focusing on performance isolation in the cloud, especially for performance-critical applications that have stringent service-level objectives [11, 12, 18, 41, 63, 66, 70]. The state of the art in practice has also significantly advanced: CPU vendors even implement hardware mechanisms to control and isolate access to CPU caches [20]. Side channels through shared resources are being patched over time [39].

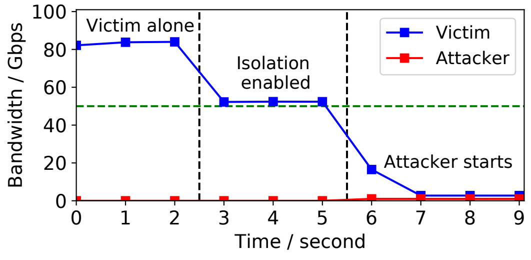

Figure 1: Violations of performance isolation under existing methods

In this paper, we visit one particular hardware device, the RDMA NIC (RNIC). RDMA offloads the network stack from OS kernel to NIC hardware to provide high throughput and ultra-low processing latency with near-zero CPU overhead. RDMA has been deployed in datacenters at scale to improve performance and free up CPU cores for first-party workloads like storage and ML [14, 17, 38, 51]. Now cloud providers are working towards supporting RDMA in general-purpose guest VMs to benefit third-party workloads. To this end, cloud providers must provide strong performance isolation for tenants sharing the same RNIC.

Many efforts have been made to improve network performance isolation in the public cloud, with a special focus on bandwidth and packet processing capacity [3, 15, 16, 25, 34, 62, 64]. However, RDMA brings new challenges due to its unique and complex NIC *microarchitecture resources* (e.g., NIC caches and processing units). Their existence and impact on performance are already known to the research community [29, 33]. To avoid performance anomalies, developers carefully design RDMA systems to avoid exhausting these microarchitecture resources [5, 9, 10, 27, 30, 50, 61]. Our study is from a different angle: we look at how these microarchitecture resources affect RDMA performance isolation from a public cloud provider's perspective. The cloud provider has no knowledge and control of tenants' RDMA applications, and tenants can consume RNIC microarchitecture resources in arbitrary manners.

To demonstrate RNIC microarchitecture resources' significant impact on performance isolation, we test the state-of-the-art approach: using SR-IOV with separated hardware traffic class (HW TC). Both SR-IOV and HW TC are hardware mechanisms available on commodity RNICs. HW TC leverages multiple hardware queues (usually 8 queues) in RNICs. We can assign each tenant application to use one queue. We run one victim traffic between two virtual machines using `ib_write_bw`, a standard RDMA bandwidth testing tool in Perftest [56]. Each virtual machine is on a different server, and the two servers are equipped with 100 Gbps NVIDIA ConnectX-5 RNICs. Figure 1 shows the bandwidth. The bandwidth test achieves 80 Gbps. We start one virtual machine on each server to represent an attacker (i.e., a buggy or malicious tenant application) and enable performance isolation to grant half of the total bandwidth to the victim and the attacker. The victim traffic reduces to 50 Gbps, which is expected. However, when we start a carefully designed attacker traffic of only 1 Gbps to intentionally exhaust one of the RNIC microarchitecture resources, the victim immediately drops to 2 Gbps, violating the performance isolation guarantee (i.e., 50 Gbps of guaranteed network bandwidth for the victim).

We develop a set of experiments to study how RNIC microarchitecture resources are used by different types of RDMA operations. Our experiments surface several interesting findings, including: (1) Exception or error handling pauses the RNIC's pipelines and causes other tenants' performance to drop drastically. (2) Control verbs cause a severe increase in cache misses and impair other tenants' performance. (3) Data verbs can exhaust different types of microarchitecture resources and violate performance isolation. To the best of our knowledge, we are the first to systematically study the impact of all types of control verbs and exceptions on RDMA microarchitecture resource consumption.

We leverage these findings to create an RDMA operation model to describe the relationship between the RDMA verb operations and the microarchitecture resources consumed. Our model allows us to understand how to exhaust each of the RNIC resources. Using the operation model, we create the first test suite, Husky, to systematically test and evaluate RNIC performance isolation solutions. Unfortunately, running our test suite on commodity RNICs reveals bad news: *there is currently no solution that can provide RNIC performance isolation.* We have already reported all of our findings to three major RNIC vendors, NVIDIA, Chelsio, and Intel. Our results are fully reproduced and acknowledged by NVIDIA, one of the largest RDMA NIC manufacturers. Finally, we present new insights on how future performance isolation solutions should be built. We hope these insights can benefit future RNIC design and RDMA software development.

This paper makes the following contributions:

- We identify multiple interactions between RDMA operations and the RNIC microarchitecture resources, including the previously unknown impact of error handling and control operations.
- We introduce the first RDMA operation model to describe how RNIC microarchitecture resources are consumed in verb operations (the standard RDMA programming API) and why these microarchitecture resources affect performance isolation.
- We build the first test suite to systematically test and evaluate RNIC performance isolation solutions. We show that none of the existing performance isolation solutions can pass our test suite. Husky test suite is available at `https://github.com/host-bench/husky`.

This work demonstrates that providing performance isolation for RDMA in the public cloud is much more difficult than one may think. There must be a higher standard for future RDMA performance isolation solutions: they should carefully consider RNIC microarchitecture resources and be evaluated by systematic benchmarks.

2 Background and Motivation

We first present the background knowledge of the network performance isolation in the public cloud. Then we introduce RDMA and discuss new challenges presented by the RDMA network performance isolation.

2.1 Network Performance Isolation in the Public Cloud

Tenants in the cloud mainly cause contention on two types of network resources. The first the most obvious one is the bandwidth in the network fabric. To mitigate bandwidth contention among tenants, one line of work [58, 60, 62] statically limits per-tenant bandwidth. Another line of work [1, 3, 4, 6, 16, 24, 25, 37, 58, 59, 68] gives each tenant a minimum bandwidth guarantee and allows tenants to use spare bandwidth capacity. The second type of resource is the packet processing resources at the end host. Per-packet processing costs depend on many factors, such as cache misses and operations to perform. Recently, PicNIC [34] provides isolation for such software packet processing. People also leverage specialized hardware to achieve the same goal [64].

It is worthwhile to note that network performance isolation is very different from network virtualization. Network virtualization orchestrates network resources to provide each tenant with an illusion of an independent network. A tenant should not impact the connectivity of the network of another tenant. The goal of network virtualization is to achieve low overhead [19, 31, 57]. In comparison, network performance isolation focuses on how to manage resource contentions to ensure that tenants can achieve guaranteed performance.

2.2 RDMA Overview

RDMA allows the NIC to directly transfer data between the wire and the application memory. The networking protocol is implemented in the NIC. Figure 2 presents the overview of

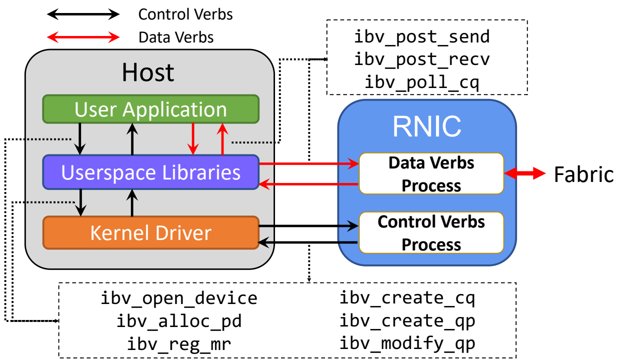

Figure 2: Overview of RDMA workflow. Verbs processing logics are heavily offloaded to the RNIC.

the RDMA workflow. It classifies standard RDMA programming interface, a.k.a., verbs, into two categories: control and data. An application first needs to call several *control verbs* to allocate necessary objects, such as queue pair (QP) and completion queue (CQ), to set up a reliable connection (RC), an unreliable connection (UC), or an unreliable datagram (UD) transmission endpoint. Then the application needs to register a memory region (MR). This registration essentially pins the memory in the host DRAM and obtains the mapping from virtual addresses to physical addresses, which enables the RNIC to directly read from or write to this memory region. All these control verbs are processed by the following procedure: RDMA's userspace libraries and kernel drivers process the verb request, generate a request command, put the command in a negotiated command queue, and ring the RNIC's doorbell (e.g., memory-mapped registers). The RNIC fetches the command from the command queue, processes it, and pushes the response back to the queue. The drivers then process the response and return the object to the application.

After the above initialization, the application can start data transmissions between local and remote memory. There are several types of operations that applications can use, such as SEND/RECV, WRITE, READ, and ATOMIC. We name these operations as *data verbs*. To issue a data verb, the application generally posts a request to its send queue and rings the RNIC's doorbell through userspace libraries. The RNIC then parses the request, reads data from the host memory, segments data into packets, and transmits packets. This procedure bypasses the kernel. There are certain differences in processing different types of requests. For example, for SEND/RECV messages, the receiver should post enough RECV requests before the sender issues SEND requests. Otherwise, the incoming SEND requests may be dropped or need retransmissions because the receiver RNIC lacks receive requests to process them, which is known as the receive not ready (RNR) error. For WRITE/READ data to/from the remote end or execute ATOMIC operations, the sender should specify correct remote virtual addresses and memory keys. An invalid address or a wrong key will trigger a memory protection error and cause the QP to transition into the error state.

2.3 Why RDMA Performance Isolation is Hard?

As shown above, RDMA offloads many host network functionalities to the RNIC, which has many invisible hardware components, and each component may individually become a performance bottleneck. Figure 3 shows the hardware components of a commodity RNIC. We draw this figure based on publicly available documents from NVIDIA [44,46,48]. In addition to the packet buffers (TX/RX Buffer), the RNIC also has multiple processing units (PU) and many types of internal caches. Each internal cache is used to store a specific type of metadata. For example, in NVIDIA RNICs, the Interconnect Context Memory (ICM) cache stores QP contexts; the Memory Translation Table (MTT) and Memory Protection Table (MPT) store entries for memory address translation and protection information; and the Work Queue Entry (WQE) cache stores prefetched send WQEs and posted receive WQEs. As these caches are derived from the design needs, other RNICs include similar components. We name these RNIC hardware components *microarchitecture resources* based on the analogy for CPU hardware. CPUs are designed to conform to a standard instruction set architecture (e.g., ARM, x86), but the CPU designers can make the microarchitecture-level decisions, such as how many levels of caches and the cache sizes. RNICs are similar because RNIC vendors have to provide the same programming interface for RDMA application developers, but the vendors can decide on these microarchitecture-level details, e.g., RNIC caches.

Many previous efforts have already identified some impacts of these microarchitecture resources on RDMA application performance. For example, [5,29,50] find that an RNIC caches QP contexts. A QP context cache miss can trigger an additional PCIe round trip for the RNIC to fetch the context from the host DRAM, thus degrading application performance. For example, 200 connections can cause an 90% request rate drop on NVIDIA ConnectX-3 NIC [5]. However, these efforts study microarchitecture resources from the perspective of an *application developer*. After a performance degradation, they identify the bottleneck resource, seek more efficient methods to use data verbs, and modify their applications correspondingly.

However, in public clouds, cloud providers have no control over tenants' applications. Tenants thus can consume RNIC's microarchitecture resources as they wish, even maliciously. Therefore, from the perspective of the *cloud provider*, we need to understand the microarchitecture resource consumption of most of (if not all) RDMA verbs, not just common data verbs. Only with this knowledge can we properly allocate RNIC's microarchitecture resources to different tenants to deliver predictable performance.

3 RNIC Microarchitecture Resources

In this section, we present a study on all the RNIC microarchitecture resources that we are currently aware of. Prior works have already identified several particular forms of resource

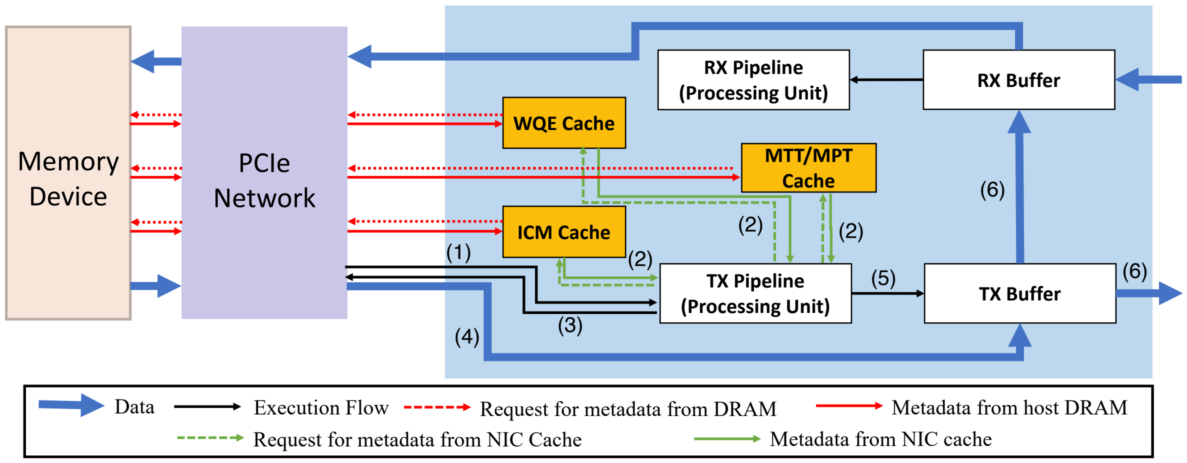

Figure 3: RDMA NIC microarchitecture hardware details: when the doorbell is rung, the RNIC first fetches the control/data verbs request from the host DRAM. (1) To fetch and process this request, the RNIC may need several metadata (e.g., QP contexts) and there are different types of caches inside the RNIC that can store this metadata. The RNIC can get the metadata directly from these caches, (2) or fetch them from DRAM if a cache miss happens (red lines in the figure). Then the RNIC processes the request and (3) sends the response back to the host DRAM for control verbs or issues DMA requests to read payload for data verbs. After (4) reading data from the host DRAM, the RNIC (5) processes the data into network packets and (6) sends them to the fabric. The symmetric receiver side is not shown for simplicity.

contention. But our goal here is to systematically study all possible types of resource contention. For each microarchitecture resource, we study how it is consumed by three categories of RDMA operations: (1) control verbs that allocate objects for applications (e.g., `ibv_create_qp`), (2) data verbs that initiate data transfer (e.g., `ibv_post_send`), and (3) exception handling operations that handle exceptions or errors (e.g., RNR errors). Due to space limitations, we first present a few *key findings* that have significant implications on RNIC performance isolation. After that, we summarize several other findings. We present a detailed analysis of NVIDIA's responses to these findings in Appendix B.

3.1 Methodology

Our findings center around how to exhaust RNIC microarchitecture resources through the verbs interface [21], the standard RDMA programming API. For each key finding, we demonstrate it with a concrete setting, which consists of a victim workload and an attacker workload. *Although we use the terminology attacker, the attacker tenant does not get unauthorized access to other tenants through vulnerabilities. Instead, the attacker is just a normal RDMA application that issues standard RDMA verbs.* Each tenant has one client and one server. The clients of the victim and the attacker locate on the same physical machine and share the same RNIC. The servers of the victim and the attacker are colocated on a different physical server. During the measurement, we do not enable any isolation mechanism. We will study existing performance isolation solutions in §5.

We focus on the performance interference between the victim and the attacker through the exhaustion of microarchitecture resources. We first run only the victim to saturate the link bandwidth capacity (bits per second) or the RNIC's maximum request rate (requests per second). We then start the attacker and measure the two metrics for both the victim and the attacker. If there is no microarchitecture resource contention, the sum of the performance metrics of the two tenants should match the RNIC's limit in the specification. Modern RNICs specify their bandwidth capacity and request rate limits. If the sum of the two tenants' performance metrics falls below both specified limits, we attribute this to the contention of microarchitecture resources. For example, assume there is no attacker, and the victim can achieve 100 Gbps. However, with an X Gbps attacker, the victim reduces to Y Gbps, and $X+Y<100$. Let us also assume the total request rate is below the RNIC specification. In this situation, we conclude that some microarchitecture resource is bottlenecked. The traffic is using RC connection unless otherwise noted.

We test four types of 100 Gbps RNICs: NVIDIA ConnectX-5 EN and ConnectX-6 Dx, Chelsio T62100-LP-CR, and Intel E810. NVIDIA NICs runs RoCE, and the Chelsio NIC runs iWARP. Intel E810 supports both RoCE and iWARP, but we currently only test its RoCE implementation. RoCE and iWARP are two standard ways to run RDMA over Ethernet-based networks. Our testbed consists of two servers, each equipped with an RNIC, and the two RNICs are connected via a 100 Gbps switch. For NVIDIA RNICs, we have access to their hardware counters, e.g., cache miss counters, through their network adapter management tool `NEO-Host` [44]. These hardware counters allow us to pinpoint which resource is oversubscribed. For example, when the ICM cache miss counter increases quickly with a certain application workload, we learn that this workload heavily uses this cache, making it oversubscribed. Since other RNICs do not expose such counters, we experiment other RNICs based on their end-to-end performance metrics (e.g., bandwidth).

Scenarios	Alone	Registration	Deregistration
BW / Gbps	96.6	95.9	**48.0**
Miss Rate	17.2%	22.9%	49.1%

Table 1: MR control verbs exhaust the MTT cache and reduce bandwidth.

3.2 NIC Caches

We are aware that an RNIC has at least three types of caches, as shown in Figure 3. The RNIC stores several types of metadata in these caches to accelerate the request processing, such as the QP contexts in the ICM cache. Prior works have identified some RNIC cache contention problems caused by data verbs with particular patterns. For example, transmitting small messages across many RC QPs simultaneously and random accesses to a large number of memory regions can cause certain types of severe cache misses (e.g., ICM and MTT/MPT) [29, 53]. ScaleRPC [5] found that this scalability problem can reduce the WRITE request rate by 90%.

In addition to these well-known problems, we observe a new, and even more severe way to exhaust caches:

Key finding #1: control verbs can cause excessive cache misses and a drastic performance reduction. Control verbs (e.g., `ibv_reg_mr`) are used to create and destroy objects like MRs and QPs, which will be used by data verbs to transfer data. To the best of our knowledge, there is no study on how control verbs consume RNIC microarchitecture resources. We find that control verbs can easily trigger excessive cache misses, thus degrading bandwidth and request rate.

We demonstrate this finding with a simple experiment on NVIDIA ConnectX-5 RNICs. We let the victim tenant use 6 cores, 16 connections per core, to issue 512B WRITE requests to exhaust the bandwidth capacity of the RNIC (i.e., 100 Gbps). Table 1 shows the results. The victim can achieve 96.6 Gbps with 17.2% MTT cache miss rate. The victim can still achieve line rate under such cache miss rate because QP multiplexing and the RNIC pipeline design can mask the overhead of cache misses to some degree. We let a single-threaded attacker keep registering memory regions (MRs) using `ibv_reg_mr` ($\sim$5K registration per second) on the victim's sender side. In this scenario, the victim's bandwidth is almost not affected, staying at 95.9 Gbps with the miss rate slightly increased to 22.9%. However, if the attacker keeps deregistering MRs, we can see a significant impact on the victim: the cache miss rate increases to 49.1%, and the bandwidth degrades to 48 Gbps. The overhead under such a high cache miss rate becomes significant and can no longer be masked by the RNIC processing pipeline. It is worthwhile to note that the attacker does not need to issue any data verbs, so the attacker consumes no network bandwidth or request rate at all. Fortunately, we observe that such interference is negligible at the receiver side.

Compared with data verbs, we find that control verbs are easier to cause performance interference. To overfill cache resources, we need to launch enough in-flight data verbs and force them to randomly access a large number of objects (e.g., MRs). For example, on NVIDIA ConnectX-5 RNIC, we find that it takes 6 threads to access more than 18K MRs with 96 QPs to cause serious enough MTT cache misses that can degrade bandwidth by 40.1%. We believe cache misses due to data verbs will become less serious since RNIC vendors keep increasing on-chip cache resources. In contrast, control verbs impact cache resources by their special semantics instead of simply consuming them, and thus the impact from control verbs can be hard to mitigate. For example, we speculate that the MR deregistration may invalidate the entire MTT/MPT cache to avoid accessing outdated MRs. This causes cache misses for accessing other MRs.

We also conduct the same experiments on Chelsio and Intel NICs, and we observe similar results.

3.3 Processing Units

The RNIC has several processing units (PUs) to process verbs requests. Due to the lack of public available counters to monitor the status of PUs, we use the request rate as the metric to measure how PUs are consumed by different verbs. We summarize the following two key findings:

Key finding #2: performance interference between different data verbs depends on the complexity of verbs. Different data verbs have different complexities. Simple verbs, like `send` and `read`, only copy data between machines. Complex verbs, such as `fetch_and_add`, atomically add a 64-bit value to the memory of a remote address. This operation leverages PCIe features (e.g., read-modify-write transactions), and may also acquire a lock on the target address. These complex verbs consume more PU resources, resulting in a lower request rate [29]. Our new discovery here is that this difference in resource consumption can also open a new pathway for performance interference through resource exhaustion: a victim's performance can be substantially penalized when colocated with an attacker that uses complex verbs intensively.

To understand this effect, we first measure the data verbs request rate when competing with other data verbs. We begin with the NVIDIA 100 Gbps ConnectX-5 RNIC. We set up two workloads for each test, and each workload runs 8 QPs across 8 dedicated CPU cores to saturate the RNIC's rate. To avoid RNIC severe cache misses, we only use 128 QPs in total and 16 MRs. We observe less than 1% cache miss in all the PU tests. To avoid reaching the bandwidth capacity limit, we use 8B as the request size of all data verbs. We first set up one workload (victim) using a particular type of data verbs, and then set up the attacker workload with different types of data verbs. We show their request rate results in Figure 4.

Our first takeaway is that in addition to the ATOMIC operations [29], the READ operations are also more expensive than SEND/RECV and WRITE. When they are running alone (as victim traffic), FAA and CAS only achieve 5.2 Mrps and 4.8

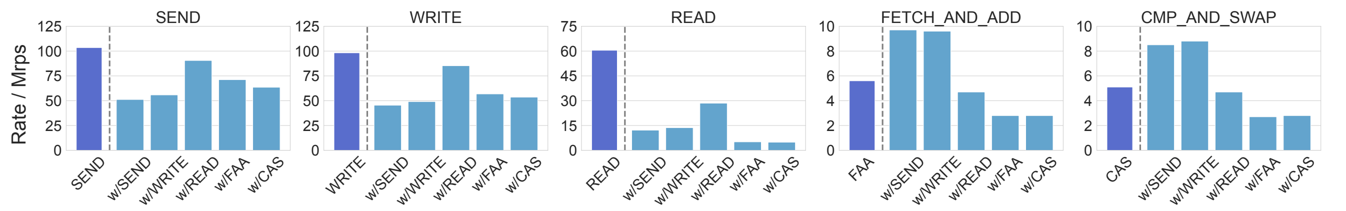

Figure 4: The contention of different data verbs on PU (NVIDIA). The leftmost bar on each subfigure is the request rate of running the victim only. The right 5 bars of each subfigure are the victim's rate when the attacker is running.

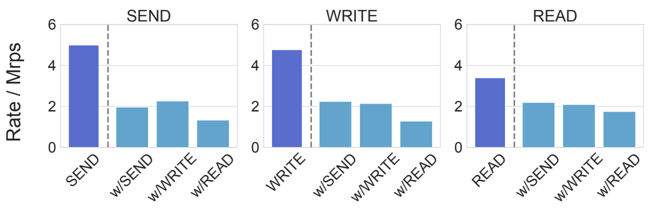

Figure 5: The contention of different data verbs on PU (Chelsio). The leftmost bar on each subfigure is the request rate of running the victim only. The right 3 bars of each subfigure are the victim's rate when the attacker is running.

Mrps respectively. READ achieves approximately 60 Mrps. SEND and WRITE can achieve more than 90 Mrps.

The second and the more important takeaway is that the contention behavior between different combinations of data verb operations can vary. For example, when the victim runs a READ workload alone, it can achieve 60 Mrps. If the attacker runs a CAS workload, the victim's request rate immediately drops to 3 Mrps. If the attacker runs a READ workload, the victim's request rate only drops to 30 Mrps. This means the complex verbs (e.g., CAS) can consume more resources and penalize other colocated verb workloads. One non-intuitive behavior we want to highlight is that the request rate of the victim running FAA or CAS can actually increase if the attacker runs a SEND or WRITE workload under this setting[1].

We also conduct similar tests on 100 Gbps Chelsio T62100-LP-CR RNIC, and the results are shown in Figure 5. This iWARP RNIC does not support ATOMIC operations. We observe that the iWARP RNIC's request rate for all types of data verbs is lower compared with RoCE RNICs, which matches findings from previous works [8, 49, 71]. We find that the contention among data verbs on Chelsio's RNIC also varies. For example, the victim with WRITE workload can achieve 4.76 Mrps without interference. The attacker can cause the victim's request rate to drop 55.0% with SEND workload and 73.1% with READ workload. The specific patterns are different from NVIDIA RNIC, but this result still demonstrates our key finding: the PU overhead of different data verbs varies.

Key finding #3: error handling can stall RNIC processing units and hang all the applications. RNICs need to handle a few types of errors, including transport timeout (the responder side does not send an ACK or NACK), Receive Not Ready

[1]We report this to the RNIC vendor and this observation is acknowledged. However, the root cause currently has not been figured out yet.

Scenario	Victim Bandwidth	SEND Bandwidth
Victim Only	97.07	-
w/o RNR	93.53	4.01
w/ RNR	**0.018**	**0**

Table 2: The impact of RNR errors on bandwidth. The unit is Gbps.

(RNR) error (the responder does not have enough receive requests for arriving send requests), local or remote protection error (the posted request does not reference a valid local or remote memory region), and local operation error (an opcode is operated on the wrong type of QP). Handling these errors require resources from RNIC processing units and some errors can be expensive for RNICs to handle.

On NVIDIA ConnectX-5 and ConnectX-6 RNICs, we find handling RNR errors can *completely* stall the RNIC processing units. For the victim, we use Perftest [56] to keep 128 outstanding 64KB WRITE requests on a single QP to saturate the bandwidth capacity. For the attacker, we only use a single QP (i.e., the SEND application in the table) to keep only one in-flight 4KB SEND request to consume a small amount of bandwidth. As shown in Table 2, if the SEND application generates traffic normally (e.g., the responder posts enough receive requests), it consumes 4 Gbps bandwidth, and the bandwidth for the victim only drops approximately 3.5 Gbps. However, when the SEND application triggers RNR errors (e.g., the responder side does not post any receive requests), both the SEND application and the victim are *stalled*. We test this RNR errors with both directions and see the same results. The reason is that the RNIC of the RNR receiver is stalled, and the RNIC cannot even process the ACK packet. The victim therefore is stalled even when they are sending traffics in the opposite direction.

We conduct the same experiments using both Intel and Chelsio NICs. We observe that the victim's QP connections are also terminated unexpectedly during data transfer for Intel E810. Fortunately, we do not see such RNR issue for Chelsio T62100-LP-CR. Our best guess is that the iWARP is designed on the top of TCP and aimed at running on a lossy fabric, so it may have a more effective error handling mechanism.

3.4 PCIe Bandwidth

The RNIC is connected to the PCIe controller and transfers data from/to the CPU using PCIe lanes. The impact of PCIe on the networking stacks has been studied by several prior works [29, 34, 52]. Based on existing PCIe models, we further study how RDMA verbs consume and even use up the PCIe bandwidth. Previous works have already identified how RDMA loopback traffic can exhaust PCIe bandwidth [26, 33]. We therefore focus on the normal RDMA TX and RX traffic. To transfer an RDMA message, PCIe introduces the following types of extra bytes: (1) an MMIO to ring the doorbell on the RNIC (64B, depending on cache line size), (2) a Work Queue Element (WQE) (36B or 64B), (3) the PCIe protocol overhead (e.g., TLP headers), and (4) extra PCIe operations triggered by cache misses. Our key observation for PCIe bandwidth is:

Key finding #4: PCIe bandwidth will only become the bottleneck when the request size is in a specific range. We only need a single tenant to demonstrate this key finding. We run the experiment on NVIDIA 100 Gbps ConnectX-5 RNIC. The PCIe bandwidth capacity is 128 Gbps (PCIe Gen 3.0 x16). We use 96 QPs across 6 cores to saturate the PCIe TX bandwidth. Each QP keeps 256 outstanding WRITE requests. We vary the request size and collect both the NIC and the PCIe bandwidth consumption by reading the RNIC's counters. The result is shown in Figure 6. We first observe that when the payload size is small, the commodity RNIC can mitigate the WQE overhead by embedding the small message in the WQE. As shown in the green rectangle, when the request size is smaller than 28B, increasing the request size does not cause more PCIe bandwidth consumption because the payload is embedded in the same MMIO operation with the WQE.

Our second observation is that PCIe TX bandwidth may only become the bottleneck when the payload size of the request is in a specific range. The reason is that short requests are first throttled by the request rate before exhausting PCIe bandwidth while large requests are always throttled by the RNIC's bandwidth capacity. We confirm this observation through a theoretical PCIe consumption model and we present two concrete examples. We assume the network MTU is 4096B and the maximum payload per PCIe transaction is 128B (the worst setting to maximize the PCIe overhead). The TLP overhead depends on the implementation [52] and we assume it as 20B, a typical size for a PCIe 3.0 device. Transmitting a 29-byte message will consume at most 127 network bytes and at least 189 PCIe bytes [29, 69]. Therefore, to saturate the link bandwidth (100 Gbps), we need at least 148.8 Gbps PCIe bandwidth, which is much larger than the PCIe 3.0x16 capacity. Appendix A includes the detailed computation. Our measurement shows that the actual consumption can be even higher, as shown in Figure 6. The consumption model for PCIe RX bandwidth (i.e., the RNIC to the host) is similar to that of TX. Additionally, too many cache misses may also cause high PCIe bandwidth consumption due to lots of PCIe reads to fetch metadata. However, in most scenarios, the large number of cache misses will first slow down the RNIC execution (e.g., introduce extra latency) and the PCIe bandwidth is therefore less consumed. In our measurement

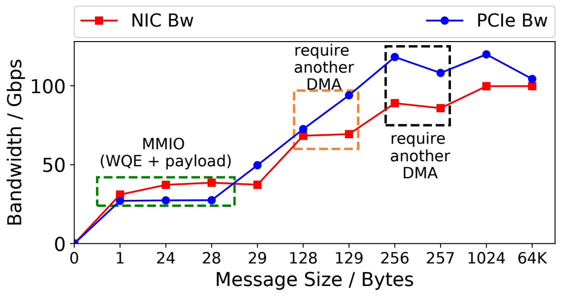

Figure 6: The PCIe bandwidth and RNIC bandwidth consumed by the application.

of cache misses, we do not observe cases where PCIe TX bandwidth is exhausted.

Both the theoretical model and our experimental results demonstrate that the PCIe bandwidth can become the bottleneck, but only for a particular request size range.

3.5 Other findings

We also have several other interesting findings. In the interest of space, we only briefly present them here. However, we do use these findings to guide our test suite design in §4.

Other finding #1: Data verbs contend for different RNIC caches. We conduct the scalability test using different data verbs, and observe different types of cache contention. For example, a large number of RC QPs that issue READ and WRITE will mainly cause ICM cache misses. A large number of UD QPs that issue SEND/RECV requests or many RC QPs that issue ATOMIC requests can cause severe RECV WQE cache misses. This observation indicates that data verbs contend for cache differently, similar to the contention on RNIC PUs.

Other finding #2: Wide range access across many objects (QP, CQ, MR) causes ICM cache misses. The scalability issue has been well studied, but our measurement reveals new observations. In addition to QP and MR, the context of the completion queue (CQ) is also stored in the ICM cache. Thus, accessing a large number of CQs can also trigger severe ICM cache misses. In addition, allocating a large number of these objects does not necessarily cause severe ICM cache misses. Wide range access across the objects (i.e., poor locality) is the key to triggering severe ICM cache misses and performance degradation.

Other finding #3: The impact of control verbs is restricted by its kernel involvement. We observe that all control verbs are first processed by the kernel drivers, thus causing expensive context switch. The execution rates of these control verbs are usually throttled by the kernel instead of RNIC processing. Therefore, control verbs have a limited impact on exhausting RNIC PUs. However, they can still cause significant performance interference and affect the other applications by triggering severe cache misses, as our key finding #1 shows.

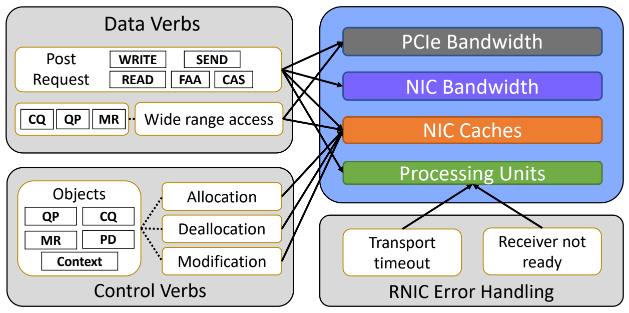

Figure 7: The relationship between verbs and microarchitecture resources. The arrow indicates heavy resource consumption.

3.6 The Resource Consumption Model

We summarize our findings in an RDMA operation model shown in Figure 7. This model describes which microarchitecture resource a verb operation consumes *heavily*. Note that a verb operation can also use other microarchitecture resources that are not captured by our experiments. This is because the usages of these resources are low and do not lead to resource contention. This model is *qualitative*: we do not try to understand the exact resource usage since we have no visibility into proprietary RNIC hardware. For example, we know a certain traffic pattern can trigger a certain type of cache misses, but we does not figure out the total size of the cache or how much of the cache an operation consumes. Even so, we show that this model is sufficiently powerful for us to create the first test suite for RNIC performance isolation, and it can capture a wide range of workloads that can break existing performance isolation solutions.

4 The Husky Test Suite

After we understand how different RDMA operations use these microarchitecture resources, we can design a test suite to evaluate performance isolation solutions. Our goal is the following: given an RNIC hardware and a performance isolation solution, we want to find a set of workloads combinations for an attacker and a victim that can break the performance isolation. We need to check different victim workloads for completeness because different victim workloads are sensitive to exhaustion of different microarchitecture resources.

Our test suite must be general: we will use it to test various RNIC performance isolation solutions on different RNICs. This means we cannot rely on tools and features from specific vendors, such as Mellanox Neo-Host [44]. In addition, different RNICs have different amounts of microarchitecture resources. And existing performance isolation solutions may only be able to mitigate contention on specific resources.

To this end, we build Husky to systematically test and evaluate RNIC performance isolation solutions. Husky targets at four types of resources: NIC bandwidth, PCIe bandwidth, NIC PU, and NIC cache. For each type of resource, we design synthetic workloads with different types of behaviors (e.g., control verbs) to exhaust this resource. More specifically, we exhaust NIC BW with long messages using different opcodes (e.g., WRITE); we exhaust PCIe bandwidth with loopback traffic and specific message patterns (from key finding #4); we exhaust NIC PU with expensive data verbs (key finding #2), small messages, or error handling behaviors (key finding #3); we exhaust different types of RNIC cache with intensive control verbs (key finding #1) and a wide range access of data verbs. We vary parameters (e.g., connection types) of some synthetic workloads to be more inclusive. In all, Husky includes 52 attacker synthetic workloads (6 for NIC BW, 4 for PCIe BW, 14 for NIC PU, and 28 for NIC cache) and 20 synthetic victim workloads. Many of the attacker workloads cannot be directly generated with existing RDMA traffic engines. We therefore extend Collie [33]'s traffic engine, the most flexible one to the best of our knowledge, to generate these synthetic RDMA traffics, including flexible control verbs workloads and error handling workloads.

Husky's framework can also easily allow running real applications as additional victim workloads. Husky currently contains two real applications, including the OSU benchmark [54] and eRPC-based Masstree key-value store [27, 40]. The OSU benchmark contains workloads such as allreduce and allgather. Note that we can integrate any RDMA applications into Husky. We test all the (victim, attacker) workload pair exhaustively from our test suite.

One key question is how to define a violation of performance isolation. Our definition of violation depends on the concrete isolation solution. Husky uses a user-specified predicate to compute the expected performance results when isolation is enabled. Husky compares the actual performance with the expected performance to identify violation. For example, most of existing performance isolation solutions only provide bandwidth guarantee. The expected performance for these isolation solutions therefore is a guaranteed bandwidth, B_g. We assume the application can consume bandwidth of B_a when running alone. The bandwidth of this application should be at least $(1-\alpha)\min(B_a, B_g)$ under *any* attacker workload, where α is a tolerance level. A lower α means stricter isolation. We use an example to demonstrate how this definition works: let us assume that attacker and the victim are configured to share the same 100 Gbps network and we set α to be 25%. If the victim can achieve 60 Gbps when running alone, it should be able to achieve at least $(1-25\%)\min(60, 50) = 37.5$ Gbps under the attacker's workload. If the victim can only achieve 10 Gbps when running alone, its consumed bandwidth should not be less than $(1-25\%)\min(10, 50) = 7.5$ Gbps. In practice, we find all existing performance isolation solutions for commodity RNICs are bandwidth guarantee or can be translated into bandwidth guarantee. We use this definition for performance isolation violation in §5 and set α to be 25%.

5 Evaluation

We use a NVIDIA testbed to evaluate existing RDMA performance isolation solutions. There are two servers in the testbed,

Resource	Processing Units			RNIC Cache		PCIe BW
Isolation Mechanism	Error Handling (RC)	Error Handling (UD & UC)	Data Verbs	Control Verbs	Data Verbs	Data Verbs
SR-IOV	✓	✗	✗	✗	✗	✗
HW TC	✗	✗	✗	✗	✗	✗
SR-IOV + HW TC	✓	✗	✗	✗	✗	✗
Justitia	✗	✗	✓	✗	✗	✗
Justitia + HW TC	✗	✗	✓	✗	✗	✗

Table 3: Performance isolation violation caused by exhausting microarchitecture resource. Justitia can only provide isolation among applications using the same function, so cannot be combined with SR-IOV. ✓ means performance isolation is properly enforced. ✗ means Husky can find a workload pair (attacker, victim) to violate performance isolation by exhausting microarchitecture resources.

and each is equipped with one 100 Gbps NVIDIA ConnectX-5 RNIC. The server is equipped with Intel Xeon Gold 5215 CPUs, and the RNICs are connected to the server through PCIe 3.0 x16. The RNICs are connected to a 100 Gbps NVIDIA switch. We use Ubuntu 20.04 and the kernel version is 5.11. For NVIDIA NICs, the kernel drivers and verbs libraries are both from 5.4-OFED. The firmware version is 16.31.1014. We also conduct all the experiments also on NVIDIA ConnectX-6 RNICs and the result is similar.

We evaluate 3 different isolation solutions provided by RNIC vendors and prior work: (1) *NVIDIA separate hardware traffic class (HW TC)*. Cloud operators can set separate TCs for different tenants to use, which separate the RNIC bandwidth and packet buffers [47] to enforce performance isolation. Modern RNICs typically only have 8 traffic classes. This means we cannot use HW TC when we want to colocate more than 8 tenants in a physical server. (2) *NVIDIA SR-IOV*. Though the SR-IOV technique is designed for hardware virtualization, it provides separate virtual functions with some separated resources to different tenants and actually achieves some degrees of performance isolation [45]. *(3) Justitia, a software-based performance isolation solution* [71]. Justitia implements data verbs rate-limiting and pacing in RDMA userspace libraries to enforce performance isolation. This means Justitia has no security: malicious applications can easily circumvent the userspace library. Although Justitia's software architecture does not target a multi-tenant public cloud environment, we still use Husky to evaluate the effect of its isolation policy (e.g., its token-based algorithm). We also evaluate all the possible combinations of the above solutions[2]. Unfortunately, though we have a testbed with Chelsio T62100-LP-CR and Intel E810 NICs, we did not enable their hardware-based isolation mechanisms. Justitia also does not support Chelsio or Intel drivers. We therefore are not able to conduct the same evaluation on Chelsio or Intel NICs. [3]

[2]We do not test Justitia with SR-IOV because Justitia only isolates traffic through the same device. When SR-IOV is enabled, tenants are using different devices (i.e., VF) and Justitia does not work for that scenario.

[3]We contact the NIC vendors and have multiple rounds of conversations with their experts. However, we still fail to enable any hardware isolation solution for RDMA on both NICs. In addition, we are not aware of any prior work that can set up such RDMA isolation.

5.1 Testing Existing Performance Isolation Solutions

Based on the types of verbs and the exhausted resources, we categorize the workloads generated by Husky into 6 groups. We distinguish the error handling of RC from UD & UC because they cause different behaviors of RNIC PU, and we observe some isolation solution (e.g., SR-IOV) provides different degrees of isolation on these PU behaviors.

We first take a look at the hardware-based isolation mechanism provided by NVIDIA. For *NVIDIA SR-IOV*, we enable two virtual functions (VF) and assign both the victim tenant and the attacker tenant with one VF. We also enable the VF-based rate limiter and restrict the maximal TX bandwidth of each tenant to be 50 Gbps, which is a typical fair sharing setting for the public multi-tenant environment. Given this configuration, we therefore define the isolation violation for *NVIDIA SR-IOV* as the victim's consumed bandwidth (in terms of bits per second) being reduced by the attacker to less than $(1-\alpha)min(50, B_a)$, where α is 25% and B_a is the victim's bandwidth without attack. For *NVIDIA HW TC*, we assign each tenant with a dedicated TC. For example, the victim exclusively uses TC 0 and the attacker exclusively uses TC 3. We configure TC 0 and TC 3 to equally share the RNIC bandwidth and the NIC buffer (which stores the packets, different from the cache). The violation definition for *NVIDIA HW TC* therefore is the same as that of *NVIDIA SR-IOV*.

The first three rows of Table 3 show the isolation effect provided by SR-IOV, HW TC, and the combination of them. Unfortunately, we find both SR-IOV and HW TC fail to provide enough isolation on RNIC's microarchitecture resources. For example, by exhausting RNIC's cache through either control verbs or data verbs, Husky can successfully affect the colocated victim's applications, even when both SR-IOV and HW TC are enabled. The key reason is that both SR-IOV and HW TC only isolate the architectural resources (e.g., link bandwidth) and do not restrict the cache usage of a single tenant. Husky therefore is able to use an attacker workload that exhausts RNIC cache, such as MTT/MPT cache. Other applications would suffer from severe cache miss and hence the performance drop. In addition, we find that although SR-IOV is mainly aimed at virtualization, it has indeed enforced some isolation, especially for RNIC PUs. The RC RNR error handling can cause RNIC PUs to pause and even hang the colocated applications if there is no performance isolation mechanism enabled. With SR-IOV, the RC RNR error does not affect tenants running on other VFs. However, the similar RNR exception handling process for UD and UC still violates the isolation of SR-IOV. Due to the RNIC's black box nature, we do not know the root cause of such a difference. Our best guess is that some part of the RNIC's PUs (e.g., that handles RC RNR) is isolated by different VFs, while other parts are not well isolated. These hardware-based solutions also cannot isolate PCIe bandwidth well. We observe that an attacker can consume substantial PCIe bandwidth and reduce the victim's usable bandwidth.

We then evaluate the software-based solution, Justitia. Justitia is not designed for the public cloud and requires the tenant to cooperate (e.g., using modified RDMA libraries). Husky can certainly break its isolation by bypassing the modified libraries, but this would defeat the purpose of testing Justitia. We therefore require all of Husky's traffics (both the victim and the attacker) to go through Justitia's modified drivers and be paced by Justitia. In addition, Justitia only supports limited types of data verbs on the latest drivers (i.e., mlx5), so we restrict the applications to only use the opcodes that Justitia supports. Justitia aims at providing each tenant a fair share of the NIC resource. We only set up two tenants, so we simply define the violation of Justitia as the victim's bandwidth is less than $(1-\alpha)min(B_a, 50)$, similar to the definition for SR-IOV. We also test the combination of Justitia and HW TC.

As shown in Table 3, Justitia does provide some PU isolation but to a limited extent. For example, Justitia takes the RNIC's request rate (i.e., execution throughput) into its isolation consideration. It therefore uses a pacer to control the request rate for each tenant and successfully prevent a single tenant from posting a large number of requests to exhaust the PUs. However, its isolation is violated when the attacker keeps posting requests that trigger error handling on the RNIC. The reason is that these errors are detected and handled by RNIC, which is out of Justitia's control. In addition, Justitia does not take cache and PCIe into consideration. The attacker tenant therefore can still exhaust the RNIC cache and PCIe bandwidth and cause other tenants to suffer from excessive cache misses or low usable PCIe bandwidth.

It is worthwhile to note that these solutions already provide more or less tolerable isolation for architectural resources, e.g., NIC bandwidth. Husky includes a set of workloads that only contend for NIC bandwidth, and we do not see such violation on those workloads when enabling these solutions. However, ignoring microarchitecture resources makes these solutions insufficient for real public cloud deployment.

5.2 Impact for Real Applications

Next, we conduct experiments on a larger testbed to study how microarchitecture resource exhaustion impacts real application workloads when using state-of-the-art performance isolation solutions. We use the allreduce workload [54] on an RDMA-based MPI implementation [55] and eRPC-based Masstree (a key-value store) [27,40] as two real victim applications. Our testbed consists of four physical servers. Each server is equipped with one 100 Gbps NVIDIA ConnectX-5 RNIC. The other settings are the same as §5.1. The victim applications run their VMs on all the four servers. The attacker tenant controls two VMs, each on a different server. We set up the testbed this way to emulate a real multi-tenant environment because an attacker may not have VMs colocated with all the victim's VMs. However, our results demonstrate that violation of performance isolation in a subset of the victim's VMs is already enough to substantially reduce the overall end-to-end performance of the real distributed applications.

For protection mechanisms, we enable either SR-IOV + HW TC or Justitia + HW TC to provide isolation for the collective communication application. For eRPC-based Masstree, we only enable SR-IOV + HW TC. This is because Justitia only supports high-performance RDMA WRITE on the latest NVIDIA drivers, but eRPC-based Masstree leverages UD SEND/RECV for its communication.

We use four types of attackers from the Husky test suite to demonstrate our results: (1) **BW attack** is the baseline. We use the standard Perftest [56] `ib_write_bw` to set up a bandwidth-hungry application. It uses 16 RC QPs and each QP keeps 128 outstanding 1 MB WRITE requests to saturate the link bandwidth (consuming ~50 Gbps when rate limiter is enabled). BW attack does not target any microarchitecture resources. (2) **PCIe attack** exhausts PCIe TX bandwidth. It runs 36 RC QPs on 6 cores and keeps 128 outstanding 257 B WRITE requests. It also consumes almost 50 Gbps link bandwidth (less than 20 Mrps) but causes more than 73 Gbps PCIe TX bandwidth consumption. This leaves only about 50 Gbps usable PCIe TX bandwidth (i.e., less than 50 Gbps usable network bandwidth) for the victim. (3) **Cache attack** exhausts RNIC cache. It runs 1536 RC QPs on 6 cores, uses 12288 MRs and each QP keeps only a single 256 B outstanding request. This attacker causes severe cache miss and only uses less than 7 Gbps link bandwidth (i.e., 3 Mrps). (4) **PU attack** pauses RNIC PUs. It runs 1 UC QP on a single core and keeps 128 outstanding SEND/RECV requests. Its receiver side does not post any receive requests, so the RNIC has to handle many receive not ready exceptions. It consumes less than 0.5 Gbps and less than 0.5 Mrps.

We begin with testing the RDMA-based allreduce workload. Allreduce is a collective communication operation widely used in distributed deep learning training. It aggregates a vector across all workers and propagates the result back to all workers. We set up 2 workers on each host (8 in total) to run allreduce. The allreduce buffer size is set to 1 MB. We run allreduce continuously and record the execution rate (allreduce operations per second). The raw rate without any isolation mechanism and interference is shown as the leftmost bars in the figure. The bar of no attack indicates the effect of

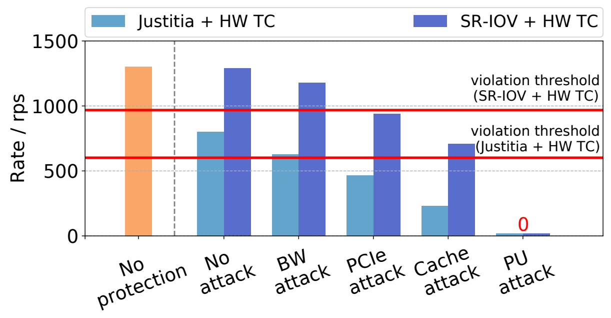

Figure 8: Allreduce results under exhaustion of different resources.

enabling these isolation solutions. When Justitia is enabled, the allreduce rate drops by 38.5%. One possible reason is that Justitia uses a shim layer (the pacer) to exert sender admission control, which introduces extra performance overheads compared to the hardware-based solutions. Since the allreduce workload only uses less than half of the NIC bandwidth (23 Gbps), its performance under attack should be at least $(1-\alpha)P_a$, where α is 25% and P_a is its performance without any attack. We can then compute a violation threshold in allreduce rate for each isolation solution based on the bandwidth the victim should consume.

The result for allreduce is shown in Figure 8. The horizontal red lines show the violation threshold. Bars under the red line indicate isolation violation. P_a for the application with Justitia + HW TC is 38.5% lower than that with SR-IOV + HW TC. This means the violation threshold is also 38.5% lower for Justitia + HW TC. We first observe that the BW attack only causes a negligible performance drop for SR-IOV + HW TC setting. And Justitia + HW TC also achieves the bandwidth isolation goal within the tolerance. We then observe that all the PCIe, Cache, and PU attacks successfully violate the isolation provided by either Justitia + HW TC or SR-IOV + HW TC. For example, the PCIe attack can cause the performance of the allreduce application to drop 27.3% for SR-IOV + HW TC and 42.1% for Justitia + HW TC. The impact of the Cache attack is more significant. Allreduce workload's performance drops more than half (71.3%) for Justitia + HW TC and almost half for SR-IOV + HW TC. We observe that the PU attack is the most powerful. It can directly stall the allreduce application by exhausting the RNIC PUs.

We use the same set of attackers to test the eRPC-based Masstree. We use the default setting of eRPC-based Masstree (e.g., key size and the number of threads). We set up the key-value server in one physical server and three clients each in a different physical server. We colocate one attacker VM with the key-value server and another attacker VM with one of the clients. We collect the execution rate (in terms of the number of GET requests per second) and the latency from all the clients. The Masstree server only uses 14 Mrps and less than 20 Gbps, so we define the isolation violation as the same as the violation of allreduce. Figure 9 and Figure 10 show the GET rates and the latency results. The SR-IOV + HW TC

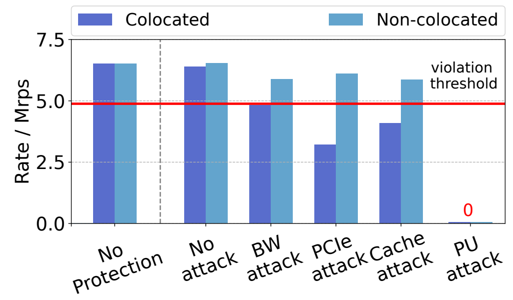

Figure 9: Mastree's GET rate under exhaustion of different resources. colocated means that the client and the attacker are on the same host. Non-colocated means that they are on different hosts.

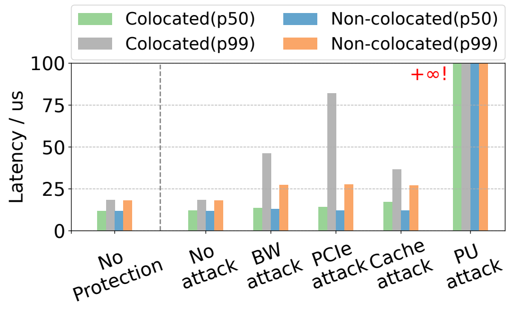

Figure 10: Mastree's latency under exhaustion of different resources.

more or less achieves the BW isolation goal within tolerance. We find that all microarchitecture resource exhaustion attacks successfully violate the isolation for the client that is colocated with an attacker VM. Similar to the allreduce workload, the PU attacker stalls the entire key-value store system. Worse still, it even pauses the clients that are not colocated with an attacker VM. This is because we stall the key-value server.

Another observation is that the performance of eRPC-based Masstree is impaired by the cache exhaustion attack but to a very limited extent. One possible reason is that the eRPC leverages UD transport. A UD QP does not need as much connection metadata as an RC QP does and therefore is less sensitive to the RNIC internal cache miss. In addition, we find that the Masstree is more sensitive to PCIe exhaustion. This is probably due to its small request size. According to our key finding #4, requests of a relatively small size cause more extra PCIe TX bandwidth consumption.

We have several high-level takeaways from the real application results.

Takeaway #1: targeting microarchitecture resources makes violating performance isolation easy. If we treat the RNIC as a black box, it is quite difficult to break performance isolation. The BW attack targets the bandwidth resource, and we observe that all the existing solutions provide good protection. However, once we know a few more details about how an RNIC works (e.g., the potential microarchitecture resources), breaking isolation becomes simple. Our attack is very efficient. For example, Cache Attack only needs 7 Gbps and 3 Mrps. PU Attack stalls victims with even less bandwidth and request rate. Note that these attacks are only targeting publicly disclosed microarchitecture components.

Takeaway #2: applications' sensitivity for resource contention is different. Applications' end-to-end performance drops can be quite different even for the same attack. The allreduce application is more sensitive to the cache exhaustion while the Masstree is more vulnerable to the PCIe exhaustion.

Takeaway #3: distributed applications need performance isolation on every single server. For both applications, the attacker only has two VMs, but why does the application-level performance drop substantially even if the application is running across four machines? Many modern distributed systems' performance is usually bottlenecked by a few slowest workers in the system. For example, in allreduce, each iteration requires synchronization of all workers. Thus, our attack on one or two workers can slow down the entire allreduce procedure.

5.3 Analysis for Existing Solutions

Our evaluation shows that all existing approaches fail to provide RDMA performance isolation.We now analyze the fundamental restrictions of these solutions and some potential improvements we may achieve.

SR-IOV and separate HW TC. These hardware based solutions already provide some hardware resource isolation (e.g., the hardware queue and the on-NIC packet buffer). Theoretically, RNIC vendors should be able to incorporate more hardware isolation features to these solutions. For example, to statically separate NIC PU or partition NIC cache for different VFs can help to build a better isolation mechanism for SR-IOV. However, these hardware modifications are nontrivial and can hardly be applied to existing hardware. RNIC vendors usually release these new features together with their new hardware products. Cloud providers thus cannot use these features in existing hardware.

Justitia. Justitia provides modified userspace libraries and uses sender admission control to enforce fair sharing of both bandwidth and execution rates for all tenants. Justitia does not work for a multi-tenant cloud because its policies are not enforceable: a malicious application can easily circumvent the modified user libraries. Putting the security aspect aside, it is worthwhile to ask whether a pure software solution like Justitia could in theory support RDMA performance isolation. We do not have a direct answer to this question, and we believe it is an interesting future research direction. We reckon that this can be quite difficult for the following reasons. First, it is challenging to track and control how much cache a tenant has occupied without hardware support. Second, it is challenging to establish a quantitative resource consumption model for verbs. Finally, error handling is deeply integrated into RDMA NIC hardware, and is opaque to software.

6 Guidelines

Our results show that, unfortunately, no existing RNIC performance isolation solution is sufficient. We analyze the failure of existing isolation solutions based on our key findings, and we present several design guidelines for potential future RDMA performance isolation work. These guidelines may also be helpful for RDMA application developers to write better RDMA applications under multi-tenant environments.

Hardware support for isolation is needed. Software approaches like Justitia [71] have a common problem. They only monitor architecture-level metrics, e.g., latency, bandwidth, and request rate. They cannot detect contention in microarchitecture resources, e.g., caches, let alone manage and fair share those resources. We believe future performance isolation solutions will have to leverage hardware support, similar to how modern hypervisors can use Intel Resource Director Technology (RDT) to monitor and manage access to the last-level cache and memory. NVIDIA RNICs expose several useful hardware counters, but they are still insufficient. For example, we can only observe cache misses, but we cannot manage the cache access or split the cache for different tenants.

A layer of indirection is needed. RDMA means kernel bypass for data verbs. This enables low latency and reduced CPU overheads. So where should performance isolation be enforced? We believe that future performance isolation solutions will require a layer of indirection either in NIC or in software. Having the enforcement point in the userland RDMA library (as Justitia) does not work, because it lacks security. Instead, a software indirection can have a microkernel-like design, with a set of cores running the isolation logic in a separate protection domain [43]. RDMA performance isolation should be enforced in such a central controller that takes over both control verbs and data verbs.

Programmer, compiler, and library support for RDMA applications. After a future performance isolation solution is invented, applications may need modification as well. If the future performance isolation solution requires strict partitioning of microarchitecture resources, this means each application has limited microarchitecture resources to use and can lead to substantially reduced performance. The amount of microarchitecture resource an application uses may also vary (depending on how many other tenants are on the same server or other configurations). Building high-performance RDMA applications will require additional effort for the programmer, compiler, and application library to efficiently use these limited resources. For CPU cache, these efforts occurred in the research community two decades ago [35, 36, 42].

7 Discussion

The impact of broken RDMA performance isolation. Our evaluation shows that a malicious tenant can cause other tenants' to suffer from drastic performance drop or even get stuck. In addition, a broken performance isolation exposes vulnerability for malicious users to conduct side-channel attacks. Since the tenant can affect others' performance on the same host, it can set up side channels that leak access patterns of victim nodes or deliver information by affecting the host's performance in a pattern [65]. RDMA performance isolation therefore is a critical feature for a secured RDMA public cloud.

What RDMA performance isolation solution should cloud providers use today? One good news is that we are not aware of any cloud provider that currently using commodity RNICs to provide RDMA-capable VMs with partitioned host resources. To rent an RDMA-capable VM, customers have to rent the entire physical machine. This means currently we do not need an RNIC performance isolation solution at all, because the RNIC only runs a single tenant's traffic. To move forward to multi-tenant usage of an RNIC, we believe performance isolation is still a major blocker, and multi-tenancy should not be enabled until a mature performance isolation solution is ready, one that can at least pass our test suite.

Generalizability to other kernel bypass host networking architectures. Our test suite design is based on the *verbs* interface, which is RDMA-specific. However, we believe our methodology should be generalizable to find violations of performance isolation in other kernel bypass architectures, e.g., DPDK [13], 1RMA [64], as these implementations commonly require RDMA-like mechanisms in the DMA portion of the design. The industry trend today is to offload functions to hardware accelerators. For example, RDMA is offloading congestion control and reliable message delivery into the hardware. Microarchitecture resources in hardware are critical to delivering these offloaded functions. Paying attention to these microarchitecture resources for performance isolation is going to be increasingly important.

8 Related Work

Microarchitecture resources in RNICs. The existence of RNIC microarchitecture resources is well-known in the networking community, and many studies focus on how to design RDMA applications to circumvent certain RNIC performance anomalies due to these resources. For example, HERD [28], FaSST [30], and eRPC [27] avoid using RDMA reliable connection to mitigate the QP context cache miss for better scalability. ScaleRPC [5] and Flock [50] multiplex reliable connections in a time-sharing manner to mitigate the scalability problem. Kalia et al. [29] studies the RNIC's PCIe behaviors and provides guidelines for writing efficient RDMA programs. Unfortunately, these works only focus on optimizing applications to fully utilize the limited resources in RNICs. However, public cloud providers cannot control the third-party tenants' applications. Collie [33] conducts a systematic search on RDMA performance anomalies, and the anomalies are mostly due to oversubscribed microarchitecture resources. However, since Collie only focuses on first-party traffic, it just builds a search space based on normal operations. It therefore only considers normal data verbs and fails to uncover findings related to other types of behaviors. For example, the key findings #1, #2, and #3 in §3 are fundamentally not covered by Collie's search space because Collie does not take control verbs, error handling, and expensive atomic verbs into consideration. In all, prior works focus more from the perspective of application developers. Our work is on a complementary aspect by looking from the public cloud provider's perspective: how these microarchitecture resources affect performance isolation. This requires us to be microarchitecture resource aware and take a look at all types of RDMA behaviors, including control verbs and error handling, because we need to deal with misbehaving and even malicious tenants.

Other NIC performance isolation solutions. PicNIC [34] provides isolation for both packet processing and bandwidth on NIC. This allows latency-bound workloads not to be affected by bandwidth-bound workloads. FairNIC [15] isolates resources in SoC-based SmartNICs. Compared with them, our work focuses on the RDMA-related resources on NICs.

Performance isolation in other contexts. Performance isolation problems are not limited to NICs. Other server hardware components also have this issue, and they already have corresponding solutions. There exist several partitioning techniques for CPU caches [11, 20] and memory bandwidth [22]. Network bandwidth in the network fabric is also a crucial resource to isolate [1, 3, 4, 6, 16, 24, 25, 37, 58–60, 62, 68] as well as the switch processing piplines [67].

9 Conclusion

RDMA is a promising networking technology to enable low latency and high CPU efficiency in datacenter networks. To enable RDMA in a multi-tenant environment, performance isolation is an important property, and RDMA NICs (RNICs) bring new challenges due to the existence of microarchitecture resources (e.g., RNIC cache, processing units). We present an RNIC operation model on how these resources are used by different RDMA operations. Using this model, we create Husky, the first test suite to evaluate RNIC performance isolation solutions. Our results show that none of the existing RNIC performance isolation solutions provides sufficient isolation against workloads that try to exhaust these microarchitecture resources. Our findings are acknowledged and reproduced by one of the largest RDMA NIC vendors. We believe that building a usable RNIC performance isolation solution will be a long battle.

Acknowledgement

We thank Chelsio, and Intel for their technical support. We especially thank NVIDIA, who gives us timely and insightful feedback, including the root causes of our findings and the corresponding solutions. We thank our shepherd Brent Stephens and other anonymous reviewers for their insightful feedback. Our work is partially supported by gifts from Adobe, Amazon, Meta, and IBM.

References

[1] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, and Eno Thereska. End-to-End Performance Isolation through Virtual Datacenters. In *OSDI*, 2014.

[2] Infiniband Trade Association. Rocev2, 2014.

[3] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards Predictable Datacenter Networks. In *SIGCOMM*, 2011.

[4] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. Chatty Tenants and the Cloud Network Sharing Problem. In *NSDI*, 2013.

[5] Youmin Chen, Youyou Lu, and Jiwu Shu. Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing. In *EuroSys*, 2019.

[6] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. HUG:Multi-Resource Fairness for Correlated and Elastic Demands. In *NSDI*, 2016.

[7] The Global Cloud Computing Market Size. https://www.yahoo.com/now/global-cloud-computing-market-size-081600295.html, 2021.

[8] Chelsio Communications. 100g network performance for illumos, 2018.

[9] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast Remote Memory. In *NSDI*, 2014.

[10] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *SOSP*, 2015.

[11] Nosayba El-Sayed, Anurag Mukkara, Po-An Tsai, Harshad Kasture, Xiaosong Ma, and Daniel Sanchez. KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores. In *HPCA*, 2018.

[12] Alexandra Fedorova, Margo Seltzer, and Michael D Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*, pages 25–38. IEEE, 2007.

[13] Linux Foundation. Data plane development kit (DPDK). http://www.dpdk.org, 2015.

[14] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *NSDI 21*, 2021.

[15] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C. Snoeren. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *SIGCOMM*, 2020.

[16] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *CoNEXT*, 2010.

[17] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over Commodity Ethernet at Scale. In *SIGCOMM*, 2016.

[18] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 342–362. Springer, 2006.

[19] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. MasQ: RDMA for Virtual Private Cloud. In *SIGCOMM*, 2020.

[20] Andrew Herdrich, Edwin Verplanke, Priya Autee, Ramesh Illikkal, Chris Gianos, Ronak Singhal, and Ravi Iyer. Cache QoS: From Concept to Reality in the Intel Xeon Processor E5-2600 v3 Product Family. In *HPCA*, 2016.

[21] Jeff Hilland. RDMA Protocol Verbs Specification. Technical report, Internet Engineering Task Force, 2003.

[22] Derek R. Hower, Harold W. Cain, and Carl A. Waldspurger. PABST: Proportionally Allocated Bandwidth at the Source and Target. In *HPCA*, 2017.

[23] IEEE. 802.3-2018 - ieee standard for ethernet. https://ieeexplore.ieee.org/document/8457469.

[24] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. Silo: Predictable Message Latency in the Cloud. In *SIGCOMM*, 2015.

[25] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert Greenberg, and Changhoon Kim. EyeQ: Practical Network Performance Isolation at the Edge. In *NSDI*, 2013.

[26] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *OSDI*, 2020.

[27] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be General and Fast. In *NSDI*, 2019.

[28] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using RDMA Efficiently for Key-Value Services. In *SIGCOMM*, 2014.

[29] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design Guidelines for High Performance RDMA Systems. In *USENIX ATC*, 2016.

[30] Anuj Kalia, Michael Kaminsky, and David G. Andersen. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *OSDI*, 2016.

[31] Daehyeok Kim, Tianlong Yu, Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Shachar Raindel, Chuanxiong Guo, Vyas Sekar, and Srinivasan Seshan. FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds. In *NSDI*, 2019.

[32] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *IEEE S&P*, 2019.

[33] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding Performance Anomalies in RDMA Subsystems. In *NSDI*, 2022.

[34] Praveen Kumar, Nandita Dukkipati, Nathan Lewis, Yi Cui, Yaogong Wang, Chonggang Li, Valas Valancius, Jake Adriaens, Steve Gribble, Nate Foster, and Amin Vahdat. PicNIC: Predictable Virtualized NIC. In *SIGCOMM*, 2019.

[35] Monica D. Lam, Edward E. Rothberg, and Michael E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. *ASPLOS IV*, 1991.

[36] A.R. Lebeck and D.A. Wood. Cache Profiling and the SPEC Benchmarks: a Case Study. *Computer*, 27(10):15–26, 1994.

[37] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. Application-Driven Bandwidth Guarantees in Datacenters. In *SIGCOMM*, 2014.

[38] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. HPCC: High Precision Congestion Control. In *SIGCOMM*, 2019.

[39] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading Kernel Memory from User Space. In *USENIX Security*, 2018.

[40] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. Cache Craftiness for Fast Multicore Key-Value Storage. In *EuroSys*, 2012.

[41] Artemiy Margaritov, Siddharth Gupta, Rekai Gonzalez-Alberquilla, and Boris Grot. Stretch: Balancing QoS and Throughput for Colocated Server Workloads on SMT Cores. In *HPCA*, 2019.

[42] M. Martonosi, A. Gupta, and T.E. Anderson. Tuning Memory Performance of Sequential and Parallel Programs. *Computer*, 28(4):32–40, 1995.

[43] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: A Microkernel Approach to Host Networking. In *SOSP*, 2019.

[44] Mellanox. Mellanox neo-host network adapter management software. https://support.mellanox.com/s/productdetails/a2v50000000N2OlAAK/mellanox-neohost.

[45] Mellanox Single Root IO Virtualization (SR-IOV). https://docs.nvidia.com/networking/pages/viewpage.action?pageId=12013542.

[46] Mellanox. Proprietary mellanox adapter diagnostics counters. https://docs.nvidia.com/networking/m/view-rendered-page.action?abstractPageId=12005244.

[47] Mellanox Quality of Service (QoS). https://docs.mellanox.com/pages/viewpage.action?pageId=19811934, 2018.

[48] Mellanox Adapters Programmer's Reference Manual. https://www.mellanox.com/related-docs/user_manuals/Ethernet_Adapters_Programming_Manual.pdf, 2021.

[49] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting Network Support for RDMA. In *SIGCOMM*, 2018.

[50] Sumit Kumar Monga, Sanidhya Kashyap, and Changwoo Min. Birds of a Feather Flock Together: Scaling RDMA RPCs with Flock. In *SOSP*, 2021.

[51] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, KR Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models, 2021.

[52] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. Understanding PCIe Performance for End Host Networking. In *SIGCOMM*, 2018.

[53] Stanko Novakovic, Yizhou Shan, Aasheesh Kolli, Michael Cui, Yiying Zhang, Haggai Eran, Boris Pismenny, Liran Liss, Michael Wei, Dan Tsafrir, and Marcos Aguilera. Storm: A Fast Transactional Dataplane for Remote Data Structures. In *SYSTOR*, 2019.

[54] OSU benchmarks. https://mvapich.cse.ohio-state.edu/benchmarks/, 2021.

[55] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. The MVAPICH project: Transforming Research into High-Performance MPI Library for HPC Community. *Journal of Computational Science*, 2021.

[56] OFED perftest. https://github.com/linux-rdma/perftest, 2021.

[57] Jonas Pfefferle, Patrick Stuedi, Animesh Trivedi, Bernard Metzler, Ionnis Koltsidas, and Thomas R. Gross. A Hybrid I/O Virtualization Framework for RDMA-Capable Network Interfaces. In *VEE*, 2015.

[58] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. FairCloud: Sharing the Network in Cloud Computing. In *SIGCOMM*, 2012.

[59] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C Mogul, Yoshio Turner, and Jose Renato Santos. Elasticswitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing. In *SIGCOMM*, 2013.

[60] Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C. Snoeren. Cloud Control with Distributed Rate Limiting. In *SIGCOMM*, 2007.

[61] Waleed Reda, Marco Canini, Dejan Kostić, and Simon Peter. RDMA Is Turing Complete, We Just Did Not Know It Yet! In *NSDI*, 2022.

[62] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the Data Center Network. In *NSDI*, 2011.

[63] David Shue, Michael J. Freedman, and Anees Shaikh. Performance Isolation and Fairness for Multi-Tenant Cloud Storage. In *OSDI*, 2012.

[64] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 1RMA: Re-Envisioning Remote Memory Access for Multi-Tenant Datacenters. In *SIGCOMM*, 2020.

[65] Shin-Yeh Tsai, Mathias Payer, and Yiying Zhang. Pythia: Remote oracles for the masses. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 693–710, Santa Clara, CA, August 2019. USENIX Association.

[66] Ben Verghese, Anoop Gupta, and Mendel Rosenblum. Performance isolation: Sharing and isolation in shared-memory multiprocessors. In *ASPLOS VIII*, 1998.

[67] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. Isolation mechanisms for High-Speed Packet-Processing pipelines. In *NSDI*, 2022.

[68] Di Xie, Ning Ding, Y Charlie Hu, and Ramana Kompella. The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers. In *SIGCOMM*, 2012.

[69] Understanding Performance of PCI Express Systems. https://docs.xilinx.com/v/u/en-US/wp350, 2018.

[70] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. CPI2: CPU Performance Isolation for Shared Compute Clusters. In *EuroSys*, 2013.

[71] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. Justitia: Software Multi-Tenancy in Hardware Kernel-Bypass Networks. In *NSDI*, 2022.

A Network v.s. PCIe

To transmit a payload through Ethernet-based IP-routed RDMA network (i.e., RoCEv2), the network protocol introduces the following overhead.

1. **Ethernet overhead**. Each Ethernet frame includes 14-byte Ethernet (exclude VLAN) header and 4-bytes CRC as L2 overhead. In addition, each Ethernet frame has L1 overhead - each frame is preceded by a 7-byte preamble and 1-byte start-of-frame delimiter. The frame is also followed by an inter-frame gap. The gap should be at least 12-byte. The total Ethernet overhead per frame therefore is 38-byte [23].

2. **IP overhead**. IP overhead comes from the IP header, with a least size 20-byte.

3. **UDP overhead**. UDP overhead comes from the 8-byte UDP header.

4. **Infiniband overhead**. The Infiniband protocol implements headers inside the UDP payload. A simple WRITE message through reliable connection (RC) needs 12-byte Base Transport Header (BTH), 16-byte RDMA Extended Transport Header (RETH), and 4-byte invariant CRC. Hence, the Infiniband protocol overhead is at least 32-byte [2].

To transmit the payload from the host DRAM to the RNIC, the RNIC PCIe behaviors include the following overhead.

1. **Ringing the doorbell**. To post a work request, users need to ring the RNIC's doorbell through memory-mapped IO (MMIO). Each MMIO has a fixed aligned size 64-byte.

2. **Work Queue Element**. The RNIC needs to fetch a work queue element (WQE) from host DRAM to the NIC. A WQE for RC/UC is 36-byte, and 68-byte for UD.

3. **TLP overhead**. Each PCIe transaction has PCIe Transaction Layer Packet (TLP) header, and the header size varies for different PCIe implementation. We assume its least size as 20-byte according to [29, 69].

We next shows the computation of the 29-byte payload example in §3. The 29-byte payload is obviously less than the MTU, and can be sent using a single network packet. Therefore, the network bytes consumed by this payload is:

$$\begin{aligned}\text{Bytes(network)} &= \text{Bytes(payload)} + \text{Bytes(Ethernet)}\\ &+\text{Bytes(IP)} + \text{Bytes(UDP)} + \text{Bytes(IB)}\\ &= 29 + 38 + 20 + 8 + 32\\ &= 127\text{(bytes)}\end{aligned}$$

For PCIe consumption, the 29-byte payload is larger than the maximal inline size (28-byte). So it cannot be delivered in the same PCIe transaction as the WQE. It therefore needs three PCIe transactions: (1) Doorbell, (2) WQE, and (3) payload, and consume the following bytes:

$$\text{Bytes(PCIe)} = \text{Bytes(payload)} + \text{Bytes(payload TLP)}$$

$$+\text{Bytes(WQE)} + \text{Bytes(WQE TLP)}$$

$$+\text{Bytes(Doorbell)} + \text{Bytes(DB TLP)}$$

$$= 29 + 20 + 36 + 20 + 64 + 20$$

$$= 189\text{(bytes)}$$

Therefore, the PCIe consumption for such payload when saturating the link capacity (100 Gbps) is:

$$\text{Bandwidth(PCIe)} = \text{Bandwidth(network)} * \frac{\text{Bytes(PCIe)}}{\text{Bytes(network)}}$$

$$= 100 * \frac{189}{127} = 148.8\text{(Gbps)}$$

B Response from NIC Vendors

We report our findings and results to the NIC vendors, including NVIDIA, Intel, and Chelsio. NVIDIA, one of the largest RDMA NIC vendors, has spent substantial effort on acknowledging and reproducing our experiments. They have successfully reproduced all of our findings in their own environment. In addition, NVIDIA provides us with detailed analysis and feedback. We would like to share them here.

Key finding #1: control verbs can cause excessive cache misses and a drastic performance reduction. NVIDIA provides a more accurate analysis of this finding: the deregistration control verbs can cause drastic performance reduction mainly because of the NIC internal QoS scheduling policy. The deregistration control verbs have higher priority than other types of operations and will be scheduled first. Consequently, these deregistration verbs trigger excessive cache misses and cause the performance to drop drastically. NVIDIA has already figured out a solution to address this issue. The high-level idea is to tune the NIC internal QoS policy so that deregistration does not have such a high priority. They are planning for a firmware upgrade to fix this issue.

Key finding #2: performance interference between different data verbs depends on the complexity of verbs. NVIDIA is familiar with this phenomenon and will roll out new firmware upgrades to address this issue.

Key finding #3: error handling can stall RNIC processing units and hang all the applications. NVIDIA provides a more accurate explanation of this phenomenon: for unreliable transport types (UC and UD), there is not the same specific RNR exception handling procedure as RC. Instead, they have other processing logic that involves firmware that handles out-of-order packets. This is the root cause of the performance interference when attacking using unreliable transport types. NVIDIA also provides a potential solution to mitigate such interference. NVIDIA Connect-X series NICs support monitoring per-VM consumption of the NIC resources. The cloud operators therefore can enforce VM capabilities policy based on the visibility of NIC resources consumption. Furthermore, NVIDIA is planning to introduce an additional layer of protection in the coming NIC firmware/hardware release to completely eliminate the attack vector for RC.

Key finding #4: PCIe bandwidth will only become the bottleneck when the request size is in a specific range. Though PCIe bandwidth contention is not a unique interference brought by RDMA, NVIDIA still acknowledged and confirmed our observation on the PCIe consumption for RDMA NIC.

We thank NVIDIA for their kind and great support. We believe the above understanding will benefit cloud operators and RDMA application developers. In addition, our collaboration with NVIDIA also demonstrates how Husky can help to improve existing RDMA solutions and build robust RDMA performance isolation in the future.