Harmonic: Hardware-assisted RDMA Performance Isolation for Public Clouds

Jiaqi Lou[1*] Xinhao Kong[2*] Jinghan Huang[1] Wei Bai[3†] Nam Sung Kim[1] Danyang Zhuo[2]

University of Illinois Urbana-Champaign[1] *Duke University*[2] *Microsoft*[3]

Abstract

Performance isolation is essential for sharing resources in multi-tenant public clouds. Compared with traditional kernel-based networking, RDMA presents unique challenges especially because RDMA NIC's complex microarchitecture resources are often hidden from users. Current RDMA isolation methods overlook these microarchitecture resources, leading to insufficient performance isolation. Consequently, a faulty/malicious tenant can exploit these microarchitecture resources to compromise well-behaved tenants' network performance. In this paper, we introduce the first microarchitecture-resource-aware RDMA performance isolation solution for public clouds, Harmonic. It consists of two key components designed to be conscious of the RDMA NIC's microarchitectural resources: (1) a programmable intelligent PCIe switch (prototyped with FPGA) and (2) an RDMA-friendly rate limiter. At runtime, these two components allow us to accurately monitor and modulate the RDMA NIC resource usage per tenant. We evaluate Harmonic with a state-of-the-art RDMA performance isolation test suite (Husky) and a popular in-memory database application (Redis). We demonstrate that Harmonic can not only successfully pass Husky but also provide Redis with $1.4\times$ higher throughput than the best alternative isolation solution.

1 Introduction

The Remote Direct Memory Access (RDMA) technology has been widely deployed in modern clouds to improve network performance. First-party workloads in clouds, such as storage [11, 17], heavily rely on RDMA to achieve high throughput, low latency, and high CPU efficiency. A natural next step for cloud providers is to bring RDMA's benefits to their public cloud tenants. Unfortunately, this has not yet come true because RDMA was initially designed for high-performance computing, lacking adequate multi-tenancy support.

One of the key missing components for bringing RDMA to public clouds is *performance isolation*. Without proper performance isolation, a buggy or malicious tenant can affect the RDMA performance of other tenants, and even conduct side-channel attacks through the RDMA network [29, 52, 54, 55]. Although network performance isolation has been extensively studied in the past decades [10, 12, 19, 20, 25, 32, 53], recent work has highlighted that prior RDMA performance isolation solutions are insufficient for public clouds [29]. An RDMA NIC (RNIC) has microarchitecture resources, such as on-NIC cache and on-NIC processing units that significantly affect RDMA performance [26, 27, 29]. However, all existing performance isolation solutions are agnostic to the contention of these microarchitecture resources among tenants, providing insufficient performance isolation when the microarchitecture resources are exhausted. For example, RDMA traffic that keeps generating expensive ATOMIC requests can exhaust the on-NIC processing units and drastically reduce the RDMA performance of other tenants [29, 46].

The goal of this paper is to explore the possibility of building a microarchitecture-resource-aware solution for RDMA performance isolation. Our high-level approach is as follows: we monitor the usage of RDMA resources (including microarchitecture resources) per tenant, and then modulate it accordingly to provide isolation. Yet, realizing our approach faces two challenges:

(C1) Accurately measuring per-tenant RNIC resource usage. RDMA traffic bypasses the kernel, which makes it hard to intercept and monitor the RDMA traffic in system software. Moreover, RNICs today only expose limited aggregate statistics, such as RNIC cache miss rates and total PCIe bandwidth consumption, without the capability of identifying the specific tenant causing this resource usage.

(C2) Finding an appropriate rate limit enforcement entry point. System software is not a viable rate limit enforcement point because most RDMA operations bypass the control of cloud providers. Commodity RNICs also do not provide rich rate enforcement features. For example, no current RNIC provides a mechanism to limit a tenant's rate of specific RDMA operations (*e.g.*, ATOMIC), and cloud providers cannot feasi-

[*] Jiaqi Lou and Xinhao Kong contributed equally to this research.
[†] Wei Bai is now with NVIDIA.

bly modify existing RNICs to incorporate these new features. We also cannot simply drop excessive packets at the RNIC, because packet losses can significantly degrade RDMA performance [21, 31, 60, 64].

Our key approaches to addressing the above challenges are outlined below. First, we make a PCIe switch serve as a sweet spot for measuring the RDMA resource usage of tenants at runtime. This choice is motivated by the following reasons. All RDMA traffic goes through the PCIe bus, allowing us to intercept all RDMA behaviors. More importantly, RDMA pins all RDMA-related objects (*e.g.*, payloads and other metadata) in the host DRAM. Thus, the physical address to tenant/object mapping is fixed. This enables us to correlate a PCIe transaction with a specific tenant and associated RDMA behaviors by mapping the transaction's target physical memory address to the RDMA objects.

To tackle the second challenge, we repurpose the rate limiters in RNIC hardware for our performance isolation. Modern commodity RNICs employ many rate limiters for congestion control purposes. These rate limiters react to network congestion feedback and reduce rates accordingly. We therefore can proactively inject an appropriate amount of congestion feedback to targeted tenants, to limit their rates when we need to limit their RDMA resource usage.

Applying our insights above, we develop Harmonic, the first hardware/software co-design solution for RDMA performance isolation that takes RNIC microarchitecture resources into account without requiring changes to applications. To measure the RDMA resource usage of tenants at runtime, we implement an FPGA-based Programmable Intelligent PCIe Switch (PIPS) in Harmonic. We extend existing RNIC kernel drivers to a Harmonic kernel driver to obtain the aforementioned physical memory address to tenant/object mappings. PIPS connects the RNIC to the host, and monitors the RDMA traffic of each tenant using the mappings provided by the Harmonic kernel driver. We implement a Harmonic daemon to repurpose the rate limiters in the RNIC hardware. Most, if not all, commodity RNICs support DCQCN [64] as congestion control algorithm [13, 24, 40]. The Harmonic daemon therefore can generate and send Congestion Notification Packet (CNP), the congestion feedback in DCQCN, to rate-limit targeted tenants for our performance isolation purpose. The Harmonic daemon limits tenants' rates based on PIPS's monitoring results. To make performance isolation more practical for public RDMA clouds, we also extend the existing RDMA performance abstraction to include a set of RDMA-specific resources, such as the number of QPs and the RDMA request rate.

We use Harmonic to enhance an NVIDIA ConnectX-6 Dx 25 Gbps NIC and evaluate Harmonic with the state-of-the-art RDMA performance isolation test suite, Husky [29], and a popular in-memory database application, Redis over RDMA [63]. We compare Harmonic with other performance isolation solutions, including hardware Single Root I/O Virtualization (SR-IOV), separate hardware queues, and Justitia [62]. Our evaluation results show that Harmonic successfully provides stronger performance isolation under various types of resource contention. This results in improving the throughput of Redis by up to 1.4×, compared to the state-of-the-art isolation solutions. To the best of our knowledge, Harmonic is the first RDMA performance isolation solution that can pass the Husky test suite [29].

Lastly, current Harmonic supports 25 Gbps RNICs, limited by the speed of the PCIe physical layer (PCIe PHY) in our commodity FPGA development board[1]. A deployable solution for high-speed RNIC will require adopting our proposed techniques in the future RNIC design. While Harmonic serves as a prototype, it demonstrates the viability of RDMA performance isolation for public clouds and can act as a benchmark for future implementations. Our design presented in this paper is currently being integrated into one leading technology enterprise's next-generation RNIC design.

2 Background

2.1 Remote Direct Memory Access

RDMA enables user applications to directly interface with RNIC by offloading network stack processing to RNIC hardware. RDMA enables low-latency, CPU-efficient networking at high bandwidth, and it is increasingly deployed at datacenters [11, 17, 21]. For example, Bai et al. [11] demonstrated that more than 70% of traffic in Azure is RDMA.

Figure 1 shows the four key components (*i.e.*, userspace libraries, kernel drivers, RNIC firmware, and RNIC ASIC) in a modern commodity RDMA system from a top-down perspective. The first component that user applications interact with is *userspace libraries*. Applications invoke APIs provided by these libraries to issue data verb and control verb operations. For example, applications call control verbs to allocate necessary objects such as queue pair (QP), completion queue (CQ), and memory region (MR). Applications thereby issue data verbs to send RDMA network traffic, such as RDMA WRITE requests to directly write remote host's memory. In a typical RDMA system, control verbs are first processed by RDMA *kernel drivers*. Kernel drivers usually conduct a few checks (*e.g.*, parameter validation) and construct a command to send to the RNICs. In the RNIC, a small piece of software or microcode embedded into hardware device memory will process these commands and return results to the kernel drivers, such as the newly created QP [42]. This software on the RNIC is known as the *RNIC firmware*. When the RNIC firmware processes control verbs, *RNIC ASIC* is also involved since many hardware status may be updated. For example, RNIC

[1]We find that the state-of-the-art FPGA whose PCIe PHY can be configured in PCIe switch upstream/downstream mode only has 8-lane edge connector after several rounds of communication with our FPGA manufacturer, Xilinx. This is also confirmed by Xilinx's public information [3, 4], but it can support any RNICs with any speed offered by the PCIe PHY. We discuss the scalability of our solution to higher speed in §7.

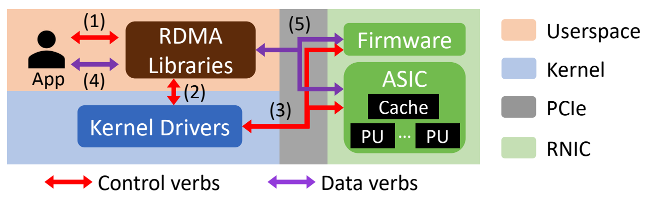


Figure 1: RDMA workflow.

has on-NIC cache to store QP contexts [27], which can be accessed and updated when the RNIC firmware handles QP creation or destruction.

Data verbs are directly passed to RNIC hardware without involving kernel drivers (or any system software), which is known as kernel bypass. For example, when applications call `ibv_post_send` to issue an RDMA SEND request, userspace libraries prepare work queue entries (WQEs) in send/recv queues. Each entry in the queues corresponds to a data verb. The libraries then notify the RNIC hardware that there is a WQE to process. Specifically, the libraries may ring the doorbell of the corresponding QP (*i.e.*, write a specific register) on the RNIC, triggering RNIC hardware to DMA read those WQEs from the host and start to process. When processing data verbs, RNIC firmware may also be involved under some scenarios, such as handling an error triggered by a data verb.

There are three types of resources in RNICs:

(R1) Traditional network resources. They include network bandwidth and packet processing capacity, indicated by bits per second (BPS) and packets per second (PPS), respectively.

(R2) RDMA-specific architectural resources. They comprise the number of QPs and request rates of different verbs (*e.g.*, ATOMIC, WRITE, and SEND) that applications can directly operate on.

(R3) RDMA-specific microarchitecture resources. They encompass the PCIe bandwidth, on-NIC cache and on-NIC processing units that are vendor-specific. These resources are not exposed to applications and can be neither monitored nor controlled precisely [31].

2.2 RDMA Performance Isolation

RDMA has already been successfully adopted in accelerating first-party workloads such as storage [11, 17]. The next question is whether these RDMA advantages can be extended to third-party workloads in the public cloud. RDMA performance isolation for public clouds is important, as customers primarily choose RDMA for workloads with demanding performance requirements. Without proper performance isolation, a faulty or malicious tenant could detrimentally impact the performance of other tenants [29].

To design a performance isolation solution, one key question is: *what's the abstraction of network performance?* The conventional wisdom is that a cloud provider should guarantee network bandwidth, measured by BPS, to a virtual machine (VM) or container. For example, Amazon Web Service (AWS) provides a 30 Gbps guarantee for its m7gd.16xlarge instance and Azure offers a 40 Gbps guarantee for its D96as_v5 VM series [7, 35]. This is done by limiting the available network bandwidth to the remaining VMs co-located on the same host.

In this paper, we argue that this conventional wisdom does not work for an RDMA network. The aforementioned microarchitecture resources make RDMA performance isolation different from that on traditional TCP/IP networks. In RDMA, most verb processing tasks are offloaded to the RNIC firmware and RNIC hardware. RNICs leverage their internal resources to support these offloaded functionalities. Not considering these resources results in performance isolation designs that are insufficient to be used in public clouds. One of the empirical evidences is that Husky [29], a prior work, has already shown that no mature RDMA performance isolation solution exists. Therefore, a comprehensive RDMA performance isolation solution for the public cloud has to consider various types of interference on RNIC's microarchitecture resources, which can occur when multiple tenants contend for access to these resources.

Static partitioning versus dynamic resource usage modulation. In general, there are two approaches to achieving performance isolation when sharing resources. Our paper explores the dynamic resource usage modulation approach, which is to monitor and control each tenant's resource usage. The other approach is to statically partition every resource and assign partitioned resources to each tenant. We did not explore the static partitioning approach for two reasons. First, RNIC microarchitecture resources (*e.g.*, NIC caches) are crucial for applications' performance. We have observed many prior works in RDMA application design to use these resources efficiently in order to avoid resource exhaustion [15, 26, 27, 31]. Static partitioning of these resources may cause catastrophic performance penalties for RDMA applications. Second, commodity RNICs currently do not support static resource partitioning, and exploring this approach thus requires building an RNIC from scratch, which is beyond the scope of our research. Our goal is to design a prototype that shows feasibility for deployment, and we thus choose to build our system around commodity RNICs.

2.3 Design Space for Monitoring and Controlling Tenant RDMA Resouce Usage

Two key questions arise for monitoring and controlling tenants' RDMA resource usage: (1) where should the cloud provider monitor per-tenant resource usage, and (2) where should the provider enforce resource usage?

The answers to these two questions depend on the deployment model of RDMA, *i.e.*, how RDMA is virtualized. Figure 2 shows the ownership (*i.e.*, owned by tenants or cloud providers) of RDMA system components in typical RDMA virtualization schemes. In the bare-metal scenario, tenants own the entire physical host, including userspace libraries and RDMA kernel drivers. They can even modify and upgrade RNIC firmware as needed [41]. Cloud providers have

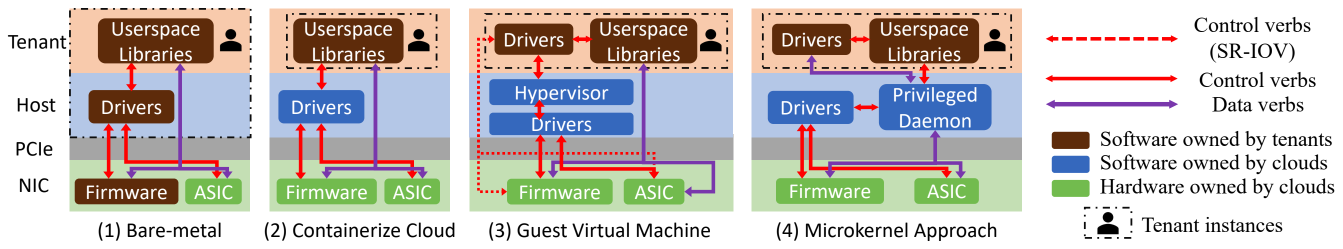


Figure 2: RDMA virtualization schemes.

limited observability and control over both data and control verbs in this scenario. However, RNIC isolation is not a pressing concern as one tenant exclusively occupies the entire machine.

In containerized clouds, each tenant owns a container, and the host OS manages all containers. In this setup, a tenant owns its container instance, including userspace libraries. The tenant's data verbs therefore fully bypass the cloud provider's control. However, drivers and hardware components are still controlled by the cloud provider, allowing them to implement management features. For instance, cloud providers can monitor and regulate RDMA control verbs by incorporating the necessary logic into kernel drivers.

In guest virtual machine (VM) clouds, each tenant owns a VM, running on top of the hypervisor. There are several approaches to exposing an RNIC to guest VM. A widely adopted approach is to use Single Root Input/Output Virtualization (SR-IOV). With SR-IOV, multiple virtual instances of the RNIC, referred as Virtual Functions (VFs), are allocated on a physical RNIC. These VFs can be attached to VMs, allowing applications within the VM to directly interact with and utilize the RNIC. The control verbs and data verbs generated by guest VM applications bypass the hypervisor completely. HyV [49] and MasQ [22] employ hybrid virtualization techniques to expose RDMA to guest VMs. They introduce backend drivers within the hypervisor, requiring guest VM drivers to communicate with these backend drivers for processing tenants' control verbs. The hypervisor operates control verbs on the RNICs on behalf of these tenants. Meanwhile, tenants within the guest VMs have the capability to directly transmit data verbs to the RNIC, bypassing the guest kernel and the hypervisor. This ensures native RDMA performance for tenant applications. In these guest VM scenarios, cloud providers typically retain ownership of the hardware components, while the ownership of kernel drivers may vary depending on the specific scheme being employed.

Another virtualization scheme adopts a microkernel-like approach. It forces all tenants to talk to a privileged daemon to use RDMA, such as Freeflow [28] and mRPC [14]. In this scenario, tenants send both control verbs and data verbs to this privileged daemon. The daemon, in turn, initiates the actual RDMA APIs to execute these verbs and subsequently provides the results back to the tenants. This design grants cloud providers comprehensive control over all aspects but comes with the trade-off of additional performance overhead.

Existing solutions' observability and enforcement entry point. To summarize, except for bare-metal environment and virtualization only using SR-IOV, control verbs can be monitored and controlled by cloud providers in kernel drivers, hypervisor backend or privileged daemon. However, data verbs cannot be easily observed or regulated. In containerized cloud (2) or guest VM (3) scenarios, data verbs completely bypass cloud provider's control. Justitia [62], an RDMA performance isolation solution, requires tenants to use its customized *userspace* libraries. However, a malicious tenant can easily bypass or alter the libraries, circumventing the intended isolation. For the microkernel approach, even if we add performance isolation features into a microkernel service, it is still challenging to accurately monitor and regulate data verbs, especially for one-sided operations. For example, RDMA one-sided operations (*e.g.*, WRITE and READ) completely bypass the responder's CPU and therefore cannot be intercepted by the privileged daemon easily.

3 Harmonic Overview

We develop Harmonic, the first RDMA performance isolation solution for public clouds that considers RDMA microarchitecture resources. Our design incorporates three key ideas.

We first introduce an RDMA-specific performance abstraction tailored for public clouds. Currently, cloud providers provide tenants with network abstractions based on BPS or PPS. Unfortunately, such metrics fall short of capturing the varied sets of resources RDMA operations use. RDMA supports various verbs as its primitives, and these verbs demand distinct resource usage. For example, let's consider an 8-byte RDMA ATOMIC compare-and-swap (CAS) request and an 8-byte RDMA SEND request. Both generate identical network traffic in terms of bits and packets, yet the ATOMIC request consumes more NIC processing cycles [27, 29], thus incurring a higher cost. Our performance abstraction considers the RDMA-specific architectural resource capacities allocated to each tenant, such as the number of QPs, CQs, MRs, and the total MR size. It is worthwhile to note that our abstraction does not include RDMA-specific microarchitecture resources, because these resources are vendor-specific and cannot be directly controlled by tenants.

The second pillar of our design ideas is to perform runtime hardware-based measurements of per-tenant RDMA resource consumption. Since RDMA data verbs bypass the kernel, resource measurement requires direct hardware involvement. In RDMA networks, a tenant's resource consumption is tightly coupled with its verb behaviors. Therefore, by intercepting and analyzing these verbs, we can gain precise insights into the resource consumption of that particular tenant. However, we cannot directly observe verb behaviors on the inter-host network, *i.e.*, Ethernet (for standard RoCEv2 deployment). This limitation arises because many RNIC resource usage behaviors would be opaque if we only monitor packets sent and received by the RNIC. For instance, the RNIC initiates PCIe transactions to retrieve entries from DRAM when its cache entries are exhausted. This RNIC activity incurs both cache miss and extra consumption of PCIe bandwidth—a crucial microarchitectural resource—but remains undetected on the Ethernet. We argue that we need to observe this *within* the host. We find PCIe switch as a sweet spot to enable this runtime measurement feature for two reasons. First, all RDMA traffic goes through PCIe bus, allowing us to capture all tenants' verb behaviors including the host memory address to be accessed in the PCIe Transaction Layer Packet (TLP) header. Second, RDMA requires all RDMA-related objects (*e.g.*, payloads, QPs, CQs) to be pinned in the host DRAM. This indicates the physical address to objects/tenants mapping is fixed and we can monitor tenant's verb behaviors by monitoring which addresses are accessed. Therefore, we can simply parse the TLP header to extract the address field and match it with the mapping, without looking into the large volume of PCIe TLP payloads. There is no existing PCIe switch supporting this functionality. We therefore build an FPGA-based Programmable Intelligent PCIe Switch to prototype this runtime measurement feature. The analogy of this PCIe switch is a programmable switch (*e.g.*, P4-based Tofino switch) in the traditional computer network. The difference is that we design the switch to run on PCIe bus instead of Ethernet. Observing verb behaviors directly allows us to not only measure the network resource consumption (*e.g.*, BPS) but also gauge the utilization of RDMA-specific microarchitecture resources, including PCIe and RNIC processing capacities.

Our third idea is to repurpose the RNIC's congestion control mechanism to facilitate RDMA-friendly rate limiting. Given the kernel and CPU bypass characteristics of RDMA, traditional software-based rate limiters are off the table due to the CPU overheads and the additional latency. Software-based rate limiters are also ineffective in limiting the data receiver side when one-sided operations are used. Moreover, RDMA deployment stems from a lossless network, and current RNICs cannot consistently ensure optimal retransmission performance across all scenarios [21, 31, 64]. Therefore, simply discarding excessive RDMA packets in hardware [16] can cause RDMA performance degradation and is not an option. Our key observation is that we already have a native hardware rate limiting mechanism implemented in modern commodity RNICs for congestion control purposes (*i.e.*, DCQCN [64]). These rate limiters react to network congestion feedback, known as congestion notification packets (CNPs) in DCQCN, to reduce the rate of RDMA connections. We can re-purpose these rate limiters for performance isolation purpose by proactively generating and sending CNPs to modulate the RDMA resource usage per tenant. While this method does consume some processing cycles (CPU cycles in our prototype), the overheads are considerably reduced compared to software-based rate limiters (§6.5).

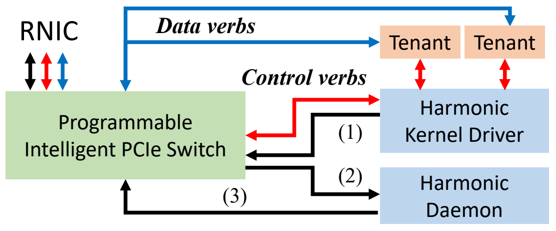


Figure 3: Harmonic overview.

Harmonic's deployment model and workflow. Harmonic assumes that the cloud provider owns the RDMA kernel drivers to intercept control verbs. This is standard for containerized RDMA clouds, para-virtualized VM clouds and microkernel virtualization clouds. We didn't consider the RDMA virtualization scheme that solely depends on SR-IOV, and we show SR-IOV itself is not enough to provide performance isolation (§6.3). In summary, Harmonic can handle the (2), (3) and (4) scenarios in Figure 2. We do not consider the virtualization scenario (1) because RNIC isolation is unnecessary on the bare-metal setting. We implement our prototype with a temporary focus on scenario (2), but it should be easily generalized to both (3) and (4) because we only rely on the modification to the RDMA kernel drivers without touching other system software components.

Figure 3 presents the system architecture of Harmonic. Harmonic has two main components: the Harmonic daemon and the Programmable Intelligent PCIe Switch (PIPS) with Harmonic kernel driver. Harmonic kernel driver is a modified version of the standard RDMA kernel driver that keeps track of control verbs issued per tenant and (1) generates the address-to-tenant/object mappings to PIPS. PIPS not only forwards RDMA traffic as a regular PCIe switch, but intercepts PCIe traffic to keep track of data verbs issued per tenant as well. Harmonic daemon is a privileged process that runs on the host OS or hypervisor. It (2) polls tenant's data verb behavior statistics from the PIPS and (3) sends congestion feedback packets to each tenant to modulate their RDMA resource usage. All these components are trusted and will not be tempered by the tenants.

Harmonic's benefits. Harmonic has several key benefits compared to existing RDMA performance isolation solutions. First and most important, Harmonic takes microarchitecture resource usage into account and thus provides stronger isolation. Harmonic observes both data and control verbs, in the meantime, restricts tenant resource usage correspondingly.

Table 1: An example for RDMA performance abstraction.

Name	# of QPs	# of WQEs	# of MRs	# of CQs	# of CQEs	MR Size	BPS	DRPS	CRPS	Prio
Alice	128	16384	128	16	8192	2 GB	10 Gbps	30 Mrps	1 Krps	0

This is different from simply observing network bandwidth usage. Second, Harmonic requires no modification of applications. There's no need to adjust application libraries, allowing for straightforward integration with application binaries. Third, our approach delivers native RDMA performance for public cloud usage. Applications' data verbs continue to bypass system software entirely, and the only latency overhead comes from the PCIe switch, which is minimal (§6.5).

Harmonic's performance abstraction. Our performance abstraction includes a set of metrics that enable tenants to accurately describe their expected RDMA network performance needs. At the same time, it allows us to design the performance isolation mechanisms to guarantee the metrics to tenants. In addition to the conventional BPS metric, our performance abstraction considers per-tenant RDMA-specific resources, including the number of QPs, CQs, MRs, and the total MR size. Application developers have direct control over these RDMA-specific architectural resources, because they directly interface these resources in the application source code. The resources in our abstraction are also vendor-agnostic: they are specified as part of the verb library [2], which work across different vendors' RNICs. It is important to note that our performance abstraction intentionally excludes explicit consideration of RNIC microarchitecture resources, such as on-NIC cache and NIC processing units. These components are vendor-specific and generally opaque to RDMA developers.

Moreover, our performance abstraction includes the typical resources other performance isolation solutions use, such as Request Per Second (RPS). We categorize RPS into data verbs RPS (DRPS) and control verbs RPS (CRPS) as they serve different purposes. While a more granular categorization of DRPS into sub-types such as ATOMIC RPS or SEND RPS is conceivable, we have chosen to opt for a normalized RPS, balancing precision with user-friendliness. The analogy is that CPU vendors use cycles instead of instructions per second as the performance metric because instructions can have variable lengths. To illustrate, Table 1 presents an example detailing the guaranteed metrics for a tenant within this framework. Let us assume one ATOMIC request consumes the resources equivalent to 3 SEND requests. Alice, with 30M DRPS, therefore can achieve up to either 10M ATOMIC requests per second or 30M SEND requests per second. It should be noted that DRPS and BPS guarantees are offered in a mutually exclusive "OR" fashion. For instance, a tenant consistently posting SEND requests with large message sizes will encounter BPS throttling before reaching the DRPS limit. Next, we present the design and implementation details of Harmonic that uses the above performance abstraction to provide RDMA performance isolation.

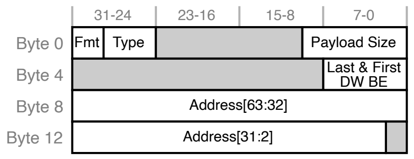


Figure 4: TLP header format where the gray blocks represent unused fields for PIPS. DW BE denotes dword byte enable.

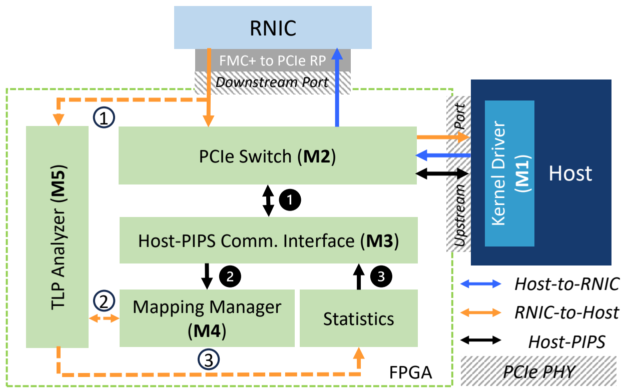


Figure 5: Programmable Intelligent PCIe Switch (PIPS) internal architecture. The dash line indicates asynchronous TLP analysis, decoupled with PCIe switch forwarding path.

4 Programmable Intelligent PCIe Switch

To monitor tenant's verbs behavior through PCIe, we develop a Programmable Intelligent PCIe Switch (PIPS) that can forward PCIe Transaction Layer Packets (TLPs) at line rate and perform real-time RDMA-centric inspections. Given address-to-object/tenant mappings captured in kernel driver, we extract the physical address of the host memory from the RNIC-issued DMA read/write TLP header (Address field in Figure 4) and utilize it to identify both the object and the tenant associated with this TLP. This capability enables us to accurately measure per-tenant RDMA resource utilization.

We build PIPS using AMD/Xilinx Versal VCK190 Evaluation FPGA board with 4K lines of RTL Verilog code and various AMD/Xilinx IPs (Intellectual Property Core). PIPS has five Modules (Figure 5): **(M1)** kernel driver, **(M2)** PCIe switch, **(M3)** host-PIPS communication interface, **(M4)** mapping manager, and **(M5)** TLP analyzer. The kernel driver maintains latest address-to-object/tenant mappings. The PCIe switch routes TLPs to their corresponding destinations. Host-PIPS communication interface and mapping manager handle the synchronization of address-to-object/tenant mappings between host and PIPS while collecting RDMA traffic statistics. The TLP analyzer inspects the TLP headers of RNIC-initiated DMA read/write requests and matches them with the address-to-object/tenant mappings.

4.1 PCIe Configuration and Routing Logic

The PCIe switch **(M2)** is the key component of Harmonic. It consists of routing logic and two instances of Xilinx Versal ACAP Integrated Block for PCI Express IPs [8]. The PCIe PHYs in the two instances are configured as PCIe switch's upstream port and downstream port, respectively. Figure 5 demonstrates Harmonic architecture: the upstream port is directly connected to the host using the PCIe edge connecter of FPGA, and the downstream port leverages the FMC+ expansion connector with a PCIe Root FMC+ plug-in module [23] to be connected to RNIC.

4.2 Address-to-Object/Tenant Mappings

Maintaining real-time address-to-object/tenant mappings in PIPS is essential for precisely monitoring RDMA resource usage per tenant. These mappings can change when applications create, delete, or modify objects. The change of the mappings is triggered by control verbs posted by RDMA applications, which are processed by the kernel driver **(M1)**. Therefore, we modify a legacy NVIDIA RNIC kernel driver (*e.g.*, `mlx5_ib.ko` and `ib_uverbs.ko`) to track address-to-object/tenant mappings. We use container's process ID as tenant ID. When a tenant calls a control verb, the Harmonic kernel driver first traverses the process tree in the kernel to find the tenant ID. It then records a mapping entry for this control verb behavior, including tenant ID, the type (*e.g.*, QP creation), the size and start physical address of the object. The RNIC kernel has already translated the virtual addresses for these RDMA objects to physical addresses for its DMA purpose, and we can directly use these translated physical addresses to populate our mapping entries. For application payloads, we also record the payload registered flags (*e.g.*, ATOMIC enabled). This information helps us determine the type of payload regions accessed by tenants in PIPS. The kernel driver is responsible for updating address-to-object/tenant mappings on PIPS by embedding an operation code in mapping entry to signal insert or delete operations to PIPS. We show detailed format and contents of both address-to-object/tenant mapping and statistics entry in Appendix B.

4.3 Mapping Synchronization and Management

We obtain address-to-object/tenant mappings from the kernel driver and then utilize the host-PIPS communication interface **(M3)** and the mapping manager **(M4)** to continuously update and manage the most up-to-date mappings in PIPS. This is crucial for later use by the TLP analyzer.

The host-PIPS communication interface receives and parses the MMIO write requests from host to update address-to-object/tenant mappings in the PIPS mapping manager (❶, ❷). Out of performance (*i.e.*, achieving real-time monitoring) and implementation complexity considerations, the mapping manager employs a hashing-based mechanism and maintains a hierarchical mapping storage system, consisting of a first-level (L1) direct-map scheme and a second-level (L2) linked-list slot pool. The mapping manager utilizes a double-hash strategy and leverages two distinct hash functions for calculating the hash values of the address field as the indexes to L1 and L2, respectively. Note that L2 is only used when collision happens in L1. In this case, each mapping entry in L1 is treated as the head of a linked list, with the remaining entries being stored in L2 linked-list slot pool. In addition to mapping management, the host-PIPS communication interface also generates completion TLPs with associated statistics as payload, when the host polls RDMA traffic statistics through MMIO read requests (❸).

4.4 Efficient TLP Analyzer

The TLP analyzer **(M5)** is responsible for extracting the target physical address in TLP headers from RNIC-issued DMA read/write requests (①). When a TLP arrives at PIPS, it duplicates the TLP and sends one copy to the TLP analyzer for analysis, while simultaneously forwarding the original TLP to its destination. In addition, the TLP analyzer implements an efficient search engine to collaborate with the mapping manager, which can perform `search` operation (②) in parallel with `insert` and `delete` operations, taking the hash value of physical address in TLP header as search key. Since the hash collision rate is low, the average search time is only 7 cycles including the latency for interconnection and updating statistics.Upon a mapping search hit, the TLP analyzer computes the statistics entry offset based on `TID`, `flags`, and `type` found in retrieved mapping entry, along with the direction of current TLP (*i.e.*, RNIC DMA read/write Host). Then it updates the statistics entry at this determined offset (③). With this approach, PIPS maintains an accurate record of both the access count and the volume of bytes accessed for each object and tenant, while simultaneously identifying the type and flag associated with the accessed memory.

5 RDMA-friendly Rate Limiting

Harmonic daemon is responsible for modulating per tenant's resource usage. It achieves this by employing two distinct rate limiting techniques for data verbs and control verbs.

5.1 Data Verbs Rate Limiting in Harmonic Daemon

The Harmonic daemon takes the proactive approach of creating and injecting Congestion Notification Packets (CNPs) to control tenants' rate. Because commodity RNICs automatically generate CNPs within the ASIC without providing an interface to users, the Harmonic daemon forges CNPs and sends them to the data sender side of tenants. Forging CNP needs the source and destination IP addresses as well as the remote QP number (QPN). Harmonic daemon obtains this information during the setup of connections. When tenants create or modify QPs, these control verbs are intercepted by Harmonic kernel driver. Subsequently, Harmonic kernel driver sends an event to notify Harmonic daemon that a new connection is set up, including both IP addresses and the QPN.

Harmonic daemon decides which tenant should be paced and at what specific rate. Harmonic daemon first keeps polling statistics collected by the PCIe switch through MMIO reads. These statistics include BPS and RPS of various types of RNIC-initiated DMA requests, such as fetching WQEs, fetching QP context, and writing payload into host memory. Then, Harmonic daemon calculates per tenant NIC BPS, PCIe BPS, DRPS consumption, as well as cache miss frequency based on the collected statistics. It sums up tenant's DMA accesses to various types of payloads (*e.g.*, WRITE) to calculate NIC BPS and DRPS, and sums up tenant's DMA accesses to various types of RDMA metadata (*e.g.*, QP contexts) to calculate cache miss frequency. For DRPS, we normalize different types of RDMA requests into the same unit, based on an estimated cost ratio for various types of data verbs. We conduct an offline profiling to estimate this cost ratio by running a set of micro-benchmarks. We run `perftest` [1] to send requests of minimal sizes in a batch to measure the maximum rate of different data verbs, and the 1/rate is the cost. In practice, we normalized WRITE and SEND operations to 1 unit, READ to 1.1 unit and ATOMIC to 3 units. Given the accurate resource usage per tenant, Harmonic daemon next compares each tenant's current usage and its allocation. Harmonic daemon directly uses NIC BPS and DRPS from tenant's profile (*e.g.*, Table 1), and calculates tenant's PCIe allocation using dominant resource fairness model [18]. Harmonic daemon analyzes guarantee profiles of all tenants on the same host and identifies the dominant resource among them. Then Harmonic daemon distributes the PCIe bandwidth based on the allocation of this dominant resource. For example, given a network capacity as 25 Gbps bandwidth and 30M DRPS, let us assume tenant A needs 15 Gbps bandwidth and 10M DRPS and tenant B needs 5 Gbps and 15M DRPS. The dominant resource therefore is bandwidth for tenant A ($\frac{2}{3}$) and DRPS for tenant B ($\frac{1}{2}$). We next allocate the available PCIe bandwidth to tenants A and B following the proportion of 4:3 (*i.e.*, $\frac{2}{3}/\frac{1}{2}$).

When a tenant uses more BPS/DRPS/PCIe bandwidth than its allocation, we send CNPs to data sender ends of this tenant's connections. Harmonic daemon currently applies a simple strategy to compute the CNP rate. Harmonic sends 1-4 CNPs in a batch after T_i intervals (in microseconds) to manage tenants' rate. Equation 1 shows how interval is updated based on the measured rate and target rate. We use two heuristic parameters T_{min} and T_{basic} in practice. T_{min} is a minimal interval threshold to avoid excessively frequent adjustments, which could lead to unstable rate or even cause performance anomaly. T_{basic} serves as a multiplier, reflecting the intrinsic response sensitivity to resource overuse. Tuning these values can adjust the strictness of policy, as a small T_{basic} punishes tenants that overuse resources more strictly.

$$T_i = max(T_{min}, T_{basic} * (1.0 - \frac{R_{current} - R_{target}}{R_{target}})) \quad (1)$$

We specially handle on-NIC cache resources due to their unique characteristics. While we can measure tenant's cache miss statistics by tracking the number of PCIe access to those metadata (*e.g.*, QP context), we do not set a cache miss threshold for each tenant. This decision is because a higher cache miss rate in one tenant does not necessarily indicate an excessive use of cache resources. Instead, we monitor overall RNIC cache contention and slow down tenants accordingly. When Harmonic daemon observes severe cache misses, Harmonic starts to slow down tenants with the lowest priority. For tenants with the same priority, we slow down them using the dominant resource fairness policy mentioned above. We acknowledge that there are alternative policies, such as monitoring a tenant's active QPs/MRs as the basis for rate-limiting decisions. However, we find that our straightforward policy is already effective in providing isolation when cache contention arises.

5.2 Control Verb Rate Limiting in Harmonic Drivers

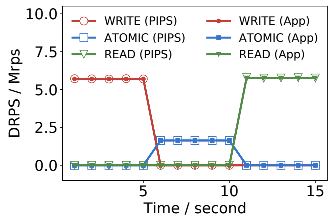
Control verbs rate limiter first needs to limit the capacity of control verbs (akin to in-flight packets) for each tenant, including the maximum number of QPs and MRs allowed per tenant. We record tenants' control verbs guarantee profiles as a linked list in Harmonic kernel driver. When a new tenant is created, we invoke Harmonic kernel driver to register a new control profile and insert it to the linked list. Whenever this tenant calls a control verb, Harmonic driver checks its current resource usage and the profile, determining if this control verb should be rejected or not.

We also need to limit the rate for control verbs to prevent tenants from excessively updating hardware status. Frequent updates have the potential to induce RNIC cache thrashing, as discussed in prior work [29]. We record timestamps for each tenant in our defined structure when they issue control verbs. When a tenant calls a control verb, we compare the current timestamp and the previously recorded timestamps. If the tenant is making control verb calls at a rate that exceeds their allocated rate, we introduce a sleep delay. We choose to slow down tenants through sleep instead of returning an explicit error. This way, Harmonic remains transparent to tenants. If we directly return errors to applications, it would necessitate error code checks and retries in applications.

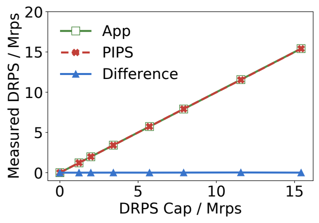
6 Evaluation

6.1 Testbed Setup

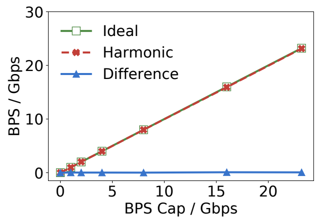
There are two servers in our testbed, each equipped with one NVIDIA ConnectX-6 Dx (CX-6) 25 Gbps RNIC. Our FPGA-based programmable PCIe switch supports up to PCIe Gen 4 with 8 lanes with up to 128 Gbps PCIe bandwidth. Nevertheless, there are no NVIDIA 100 Gbps RNICs that support PCIe Gen 4 with 8 lanes. We therefore use Harmonic to enhance our CX-6 25 Gbps RNIC. RNICs of two hosts are directly connected without a network switch. The BPS capacity of our RDMA endhost is 25 Gbps. We use the standard RDMA benchmark tool, `perftest` [1], to measure the DRPS capacity,



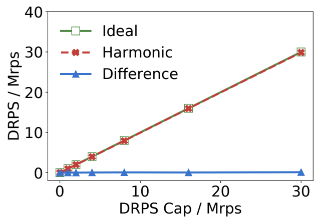
(a) Measurement of operation types



(b) Measurement of RPS



(c) Control of BPS



(d) Control of RPS

Figure 6: Measurement and control of RDMA traffic. App denotes the performance metrics as reported by `perftest`.

and the result is ∼30 M DRPS.

Both servers are running Ubuntu 20.04. Harmonic kernel driver is built upon MLNX_OFED-5.8.1.1.2.1 [42], with a total of 658 lines of C code modifications. Harmonic daemon is implemented in C/C++ with a total of 2537 lines of code.

6.2 Measurement and Control of RDMA Resources

We first use microbenchmarks to demonstrate that Harmonic can accurately measure tenants' verbs behaviors and limit their resource usage. We let a tenant run different data verbs workload from `perftest` in three time periods. It generates WRITE traffic, ATOMIC traffic, and READ traffic, each for 5 seconds. We record the DRPS measured by the `perftest` per second and compare it with the request rate measured by PIPS. As shown in Figure 6a, PIPS successfully identifies the types of data verbs and measures the request rates of each workload accurately. Figure 6b shows that Harmonic also accurately measures tenants' behaviors across different request rates.

Next, we evaluate our CNP-based RDMA-friendly rate limiter. We use `perftest` to generate workloads that extensively consume BPS and DRPS resources. We let Harmonic to set different capacity for these two resources. We measure the achieved BPS/DRPS and compare them with the capacity. Figure 6c and Figure 6d show that our rate limiter can accurately control a tenant's BPS and DRPS. It is worthwhile to note that we observe that Harmonic daemon can react to resource overuse within one millisecond. As discussed in Equation 1, a stricter T_{basic} or T_{min} leads to fast reaction (*i.e.*, a few hundreds of microseconds) while it may also hurt overall performance. In practice, we set T_{basic} to 500 us and T_{min} to 200 us, which we find already sufficient to enforce isolation.

6.3 Harmonic End-to-end Evaluation

We use the state-of-the-art RDMA performance isolation test suite, Husky [29], to perform end-to-end evaluation of Harmonic. Husky includes a set of victim traffic patterns that are sensitive to different types of resource contention, and four sets of attacker traffic patterns that exhaust four types of resources: RNIC BPS, RNIC processing capacity, RNIC cache, and RNIC PCIe bandwidth. We observe that the reliable connection retransmission attack described in Husky (Section 3.3) that exhausts RNIC processing capacity has already been fixed in the latest NIC firmware, and the RNIC control verbs cache attack only has a negligible effect on 25 Gbps RNIC. Harmonic passes all other Husky's tests with a tolerance level $\alpha = 20\%$, indicating a tenant's traffic will be no less than 80% of its guarantee in the worst case, which is substantially better than all existing solutions. For most tests, Harmonic effectively safeguards tenants to achieve their guarantees (*i.e.*, less than 5% difference). We next use a set of typical workloads from Husky as the case study to demonstrate why existing solutions fail and how Harmonic satisfies tenants' guarantee.

For each case study, we also compare our results with three baselines: (1) SR-IOV, which allocates individual virtual function (VF) for each tenant to use [47]; (2) Separate hardware traffic class (HW TC), which is supported by modern RNICs to dedicate a RNIC traffic class to specific tenant for quality-of-service (QoS) control and performance isolation [45]; (3) Justitia, a recent software-based isolation solution for RDMA networks [62]. Note that Justitia requires tenants to use specific userspace libraries, so a malicious tenant can circumvent Justitia's control by not using these Justitia's libraries. However, we still want to evaluate Justitia's isolation mechanism (which includes its rate limiter design).

We allocate two tenants, named Alpha and Beta, on the same pair of hosts. Our RDMA hosts support up to 25 Gbps and ∼30 M DRPS. We thus set our isolation goal to be that both Alpha and Beta are guaranteed with 12.5 Gbps and 15 M DRPS. For each attack, we let Alpha run a Husky victim traffic that is sensitive to a specific type of resource, and we let Beta run an attacker traffic targeting the specified resource. Results are shown in Figure 7 to Figure 10. *No Interference* means running Alpha or Beta alone with no isolation enabled, and *No Protection* means Alpha and Beta are running together without any isolation. For a fair comparison, we configure SR-IOV and HW TC to assign one virtual function or traffic class to each tenant, respectively. Justitia currently only supports fair share and does not provide any QoS guarantee. SR-IOV and HW TC only support RNIC bandwidth (*i.e.*, BPS) guarantee. We therefore configure each virtual function with 12.5 Gbps for SR-IOV. HW TC currently does not support floating-point rate configuration, and we thus configure both tenants with 12 Gbps guarantee and reserve 1 Gbps for potential traffic burst. For each figure, we use a *dashed red line* to denote the guarantee, a *gray dashed line* to de-

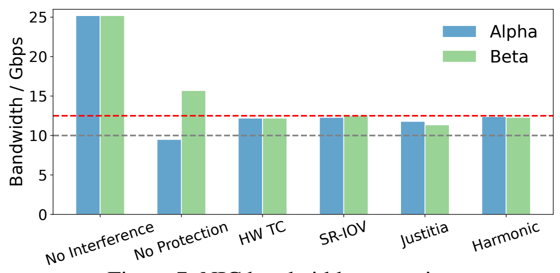


Figure 7: NIC bandwidth contention.

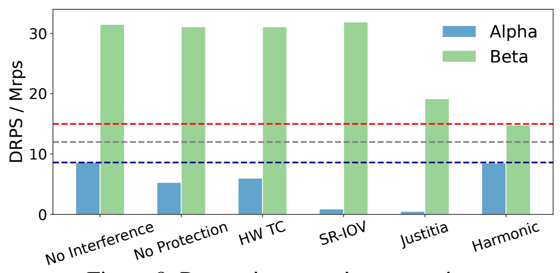


Figure 8: Processing capacity contention.

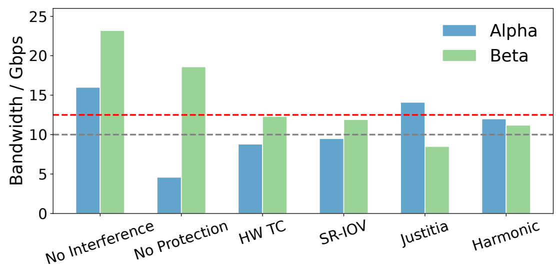


Figure 9: RNIC cache contention.

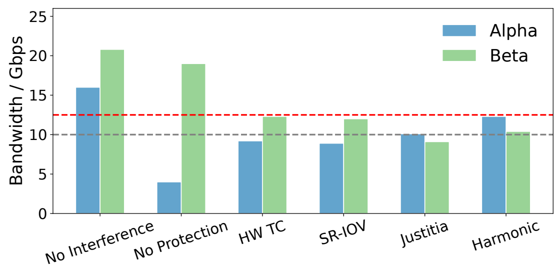


Figure 10: PCIe bandwidth contention.

note the tolerance bar (10 Gbps and 12 Mrps). We use a *blue dash line* in Figure 8 to show the victim performance since it is originally smaller than the guarantee and should not be affected.

RNIC BPS contention. We first conduct BPS contention experiment. Alpha sets up a single connection and keeps sending 64KB WRITE verbs, and Beta sets up 16 connections and keeps sending 4KB WRITE verbs in a batch. Both Alpha and Beta consume almost all the RNIC BPS when running alone. When running together without isolation, Beta occupies more BPS since it has more connections. When isolation is enabled, we observe that all existing solutions and Harmonic successfully satisfy all tenant's guarantees. This shows that RNIC BPS is accurately monitored and controlled by all the existing solutions and Harmonic.

Processing capacity contention. We let Alpha run a throughput-sensitive Husky victim, which uses 36 connections and keeps issuing 64-byte messages. Beta uses 64 connections to keep generating expensive 8-byte ATOMIC traffic to exhaust RNIC processing capacity. We normalize DRPS based on our profiling results, which show that a single ATOMIC operation costs roughly three times as WRITE operations. Figure 8 shows the DRPS for Alpha and Beta. Note that Alpha does not use all its traffic demand, so the isolation goal is that Alpha's performance should not be affected when Beta joins (shown as the blue dashed line). When no isolation is enabled, Beta's ATOMIC workloads exhaust the RNIC processing capacity and cause Alpha's performance to drop by 38%. HW TC does not react to this attack effectively because Beta only consumes a small amount of BPS (i.e., 6.7 Gbps), which is substantially lower than the rate limit. Beta's rate therefore is not paced by HW TC, and Beta exhausts the RNIC processing capacity. When SR-IOV is enabled, Beta's rate is not reduced as well, and we observe Alpha's rate drops. Since SR-IOV implementation details are not publicly available, our best guess is that this workload may cause some scheduler issues in SR-IOV implementation. Even though Justitia considers processing capacity in its design, it is agnostic to the type of verbs and thus performs even worse. It does reduce Beta's ATOMIC traffic, but Alpha's performance is even more severely degraded. This is because Justitia treats these verbs equivalently without accounting for the actual resource consumption of expensive ATOMIC verbs. Our observation of existing solutions is aligned with Husky's results. Harmonic carefully considers the expensive costs of ATOMIC requests and limits Beta's rate accordingly, reserving adequate processing capacities to achieve Alpha's guarantee while satisfying Beta's requirement.

RNIC cache contention. We let Alpha run a Husky victim that is sensitive to on-NIC cache contention, which keeps generating 8-byte WRITE requests in batches across 512 different memory regions. Beta runs a Husky attacker that uses 4 connections to repeatedly issue single 512-byte WRITE request to 16K different memory regions to exhaust the on-NIC cache resources. Figure 9 shows that when cache contention occurs, the available RNIC BPS is less than 25 Gbps. Even though both SR-IOV and HW TC reduce Beta's BPS consumption to less than 12 Gbps, Alpha's performance is only improved by a minimal extent. We suspect that under severe cache contention, the effectiveness of SR-IOV and HW TC is also affected. For example, the severe cache miss may also slow down the SR-IOV and HW TC scheduling process. Though the current design of Justitia is cache agnostic, it successfully satisfies Alpha's guarantee while leading to a drastic drop in Beta's performance, making it not satisfy the guarantee even with a 20% tolerance level. This is probably because

Justitia identifies both applications as throughput-sensitive applications and schedules them equivalently, while Alpha issues smaller messages in batch and therefore occupies more Justitia's tokens. Harmonic detects the cache contention and measures the available BPS. It then allocates the reduced available BPS to Alpha and Beta fairly. This makes both Alpha and Beta achieve the guarantee within the tolerance level. Note that the strict guarantee is impractical in this case because the bottleneck is the RNIC cache.

PCIe contention. Though our testbed supports up to ~ 64 Gbps PCIe bandwidth. We configure our FPGA-based PCIe switch to only support 32 Gbps PCIe bandwidth for our 25 Gbps RNIC. This $\frac{PCIe\ BW}{RNIC\ BW} = \frac{32}{25}$ ratio emulates scenarios for higher speed RNICs (*e.g.*, $\frac{128}{100}$ and $\frac{256}{200}$), where PCIe bandwidth can be one of the bottlenecked microarchitecture resources. We let Alpha run the same application as in the Cache contention case. We let Beta run the PCIe attack in Husky, which keeps sending 257-byte WRITE that triggers several DMAs to maximize PCIe consumption. As shown in Figure 10, the available RNIC BPS therefore is capped by the PCIe bandwidth and is substantially smaller than 25 Gbps. Both SR-IOV and HW TC successfully reduce Beta's rate and improve Alpha's performance, but to a limited extent. The key reason is that given the same amount of RNIC BPS consumption, Beta consumes more PCIe bandwidth than Alpha and should be paced more in this situation. Similar to what is observed in the above cache contention scenario, Justitia reduces Beta more because of its larger message size and only satisfies Alpha's guarantee. Harmonic's hardware monitor allows us to accurately track each tenant's PCIe bandwidth consumption and allocate PCIe bandwidth accordingly based on tenants' guarantee. For example, each tenant is allowed to consume half of the PCIe bandwidth (*i.e.*, roughly 16 Gbps) in this situation.

6.4 Performance Isolation for End-to-End Applications

We evaluate how Harmonic provides performance isolation for a real application. We use an RDMA-based Redis [63] as our tenant workload. We use the same Husky attack workloads described in the previous section as attackers. Similarly, our isolation goal is to enforce fair share resource allocation between the Redis application and the attackers. We also enable both SR-IOV and HW TC as a comparison. We do not evaluate Justitia's performance for two reasons: (1) Jusitia needs application modification to fully support its isolation and is not secured for real cloud deployments; (2) Justitia does not support READ operations in the latest drivers.

Redis over RDMA implements an RDMA backend transport to accelerate Redis key-value store and has been large-scale deployed in industry [63]. We use this redis-benchmark application to generate 1KB get and set workloads, and measure its average application QPS. This benchmark can achieve about 450K QPS, consuming 4.2 Gbps BPS and 1.2 Mrps DRPS. This is less than its performance guarantee, so the

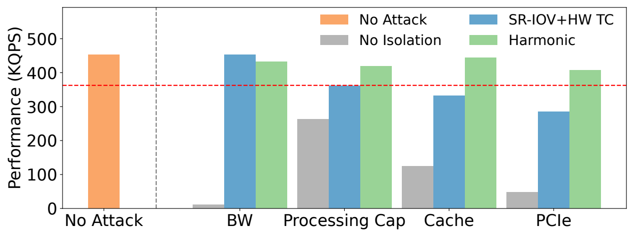


Figure 11: Performance of Redis over RDMA across different attack types and isolation schemes.

goal is that Redis's performance should not be affected by any attacker. We then run those four types of attacks without isolation, with SR-IOV + HW TC, and with Harmonic. As shown in Figure 11, all four types of attack successfully exhaust specific types of RDMA resources and cause a drastic Redis performance drop. When isolation is enabled, we observe that both SR-IOV + HW TC and Harmonic successfully provide protection against an attacker that tries to exhaust network bandwidth. Though SR-IOV + HW TC does not consider processing capacity, Redis achieves its guarantee under processing capacity contention with SR-IOV + HW TC. This is probably because Redis workload is more robust to the processing capacity contention. However, SR-IOV + HW TC fails to provide sufficient isolation when cache or PCIe bandwidth is contended. Harmonic proactively monitors these microarchitecture resource contentions and applies rate limit according to per tenant's usage. Harmonic therefore successfully maintains Redis's performance within the tolerance level when on-NIC cache or PCIe bandwidth is under contention performing 1.3x$\sim$1.4x better than the combination of two state-of-the-art isolation solutions.

6.5 Overhead Analysis

Hardware and PCIe costs. Our hardware cost analysis based on the implementation report from AMD Vivado [9] shows that PIPS, with an internal reference clock frequency set at 250 MHz, consumes 8,571 LUTs (*i.e.*, 0.95% of VCK190 FPGA LUT resources), and 554 BRAMs (*i.e.*, 57.29% of VCK190 FPGA BRAM resources) mainly used to store host mapping entries.

We also measure the cost of PCIe bandwidth for updating mappings and collecting statistics between the host and PIPS. RDMA application does not frequently invoke control verbs during data transmission, the mapping updates consumption is therefore negligible. During our evaluation of various Husky's attack workloads, we observe that the mapping updates only consume no more than 8 Mbps (0.025%) extra PCIe bandwidth. The extra PCIe bandwidth consumption caused by polling statistics is determined by the polling frequency. In practice, we poll these statistics every 100 us and we find it already sufficient to achieve an accurate rate control and enforce performance isolation. The per tenant PCIe bandwidth consumption is 64 Mbps for host-to-switch direction and 76.8 Mbps for switch-to-host direction, which

Table 2: Network performance overhead.

	Latency (us)		Max. Bandwidth	Max. Throughput
	64B	64KB	(Gbps)	(Mrps)
Baseline	3.3	50.4	23.0	28.1
Harmonic	5.6	52.6	22.8	28.1

can be comfortably accommodated by the bandwidth slack between PCIe and RNIC line rate. The detailed calculation and analysis is in Appendix C. Harmonic daemon currently only consumes 33.5% of a single CPU core and scales with negligible CPU usage increment. The CPU usage is determined by the frequency of polling statistics.

Network performance overheads. We run microbenchmarks using `perftest` to measure the latency, achieved bandwidth, and request throughput with and without Harmonic to analyze network overheads introduced by Harmonic. For brevity, we show the results of RDMA READ in Table 2. We demonstrate the latency penalty under different packet sizes in Figure 12. While there is a marginal increase in latency overhead with larger packet sizes, Harmonic adds less than 2 us to the round-trip latency across all packet sizes. This is mainly because our PCIe switch is implemented in FPGA, which is less performant than traditional ASIC-based PCIe switches on the host and SmartNICs. Furthermore, employing PCIe extender card and FMC+ to PCIe root module for the purpose of full-system operation can also incur additional latency. Note that our monitoring feature is decoupled with the PCIe switch forwarding functionality, so the monitoring feature does not contribute to this overhead at all. Besides, Harmonic introduces only negligible drops in network bandwidth or request throughput.

To summarize, Harmonic's overhead is negligible for high-speed RDMA networks. Additionally, we believe that the monitoring and rate limiting functions inherently should be integrated into future generations of RNICs. Overheads, such as the extra PCIe consumption and the FPGA's latency, will be further eliminated when these functions are implemented within RNIC's ASIC. For example, NVIDIA Bluefield-2 SmartNIC has an embedded PCIe switch that routes RDMA traffic among RNIC ASIC, embedded ARM CPUs, and host [44], and only introduces nanosecond-level latency overhead [59]. We therefore believe Harmonic's PIPS overhead can also be mostly eliminated by implementation in ASIC and being integrated into RNIC.

7 Discussion

Scaling to higher-speed network. We believe our solution is scalable to 100/200 Gbps RNICs because the overhead of Harmonic (*i.e.*, FPGA resources usage, extra PCIe bandwidth consumption) does not increase with higher network capacity. The concerns may fall on whether TLP analyzer can keep up with higher PCIe bandwidth and whether the mapping manager scales to store more mapping entries. Our TLP analyzer can handle higher PCIe bandwidth, as the average search and update time of our design for one mapping entry is 7 cycles at 250MHz frequency. This can be even further minimized

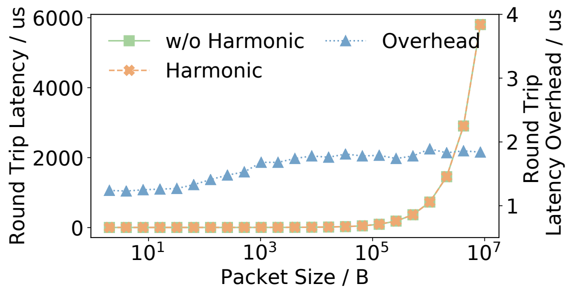


Figure 12: Latency overhead across different packet sizes. The green and orange lines present the absolute round trip latency with left y-axis when packet size differs. The blue line demonstrates the round trip latency overhead is less than 2 us using the right y-axis.

with increased parallelism. The architecture of the mapping manager can easily be extended to a multi-hashing hierarchy, thereby facilitating the storage of a greater number of mappings with only a marginal increase in search time. Additionally, concepts from match-action table of P4 switch and more advanced mapping management like binary search or Cuckoo hashing can be implemented on top of PIPS to further reduce memory overheads. In the meanwhile, the scalability concerns are notably mitigated when considering an ASIC implementation with optimized logic interconnections and resource utilization while running at a higher frequency.

Is our performance abstraction easy for users to understand? Our performance abstraction is more complex than traditional performance abstraction which only considers network bandwidth. We believe this is necessary because RDMA network is indeed more complex and application developers are already interacting with this performance abstraction when developing RDMA programs [26, 27]. We only extend the abstraction to include more architectural resources that users can directly control, such as the number of QPs. These extended metrics are no difference from the number of vCPUs or the size of memory in today's cloud VMs specifications. We believe developers should be aware of these resources in order to write performant and predictable RDMA applications.

Deployability of Harmonic. Harmonic requires both hardware and software modifications to existing clouds. From the perspective of hardware, Harmonic uses PIPS as a prototype to measure per-tenant RDMA resource consumption at runtime. In practice, the best implementation entry point should be within the RNIC regarding performance and hardware costs. One leading technology enterprise is currently integrating part of our designs into their next-generation RNIC. In terms of software, Harmonic needs to have the full control of the RDMA kernel drivers to manage control verbs for all tenants. Containerized clouds have already provided such control since all tenants are sharing the same kernel managed by cloud operators. Harmonic software therefore can be deployed in containerized clouds without any barriers. In

VM-based clouds, native SR-IOV does not support managing control verbs for tenants since guest kernel drivers can directly communicate with RNICs. Extra modifications to both guest kernel drivers and hypervisors are required to deploy Harmonic in these scenarios. Existing solutions such as HyV [49] and MasQ [22] have already virtualized all control verbs involving hypervisor in VM-based clouds. This provides feasible entry points to integrate Harmonic's software features into these solutions.

RDMA-friendly rate limiting. We currently repurpose RNIC's native rate limiters to modulate tenants' RDMA resource usage by sending CNPs. This achieves efficiency and is transparent to applications, but we acknowledge that sending CNPs from software may not be the best approach in the future. For example, a transient network congestion may affect the accuracy of such rate limiting mechanism. Emerging RNIC features such as programmable congestion control (PCC) [43] allow customized congestion control algorithms. This potentially provides a more straightforward and accurate way to leverage RNIC's rate limiter for performance isolation purposes. For example, a data receiver can send specialized packets to specify the maximal sending rate that the data sender can enforce, similar to TCP receive window.

Generality of Harmonic. Harmonic currently targets at RDMA performance isolation, focusing on the bottlenecked RNIC microarchitecture resources. We believe Harmonic can also be leveraged for other scenarios besides RDMA networks. For example, multiple I/O devices (*e.g.*, GPU and NIC) may be connected to the same PCIe switch and thereby contend on PCIe and memory bus resources [30]. Harmonic can also be adapted to isolate resources among different I/O devices and hence manage the complex intra-host network.

8 Related Works

Understanding microarchitecture resources in RNICs. Research community has already started to study the hardware resources in RNICs. Existing works focus on how to avoid certain performance anomalies caused by NIC resources from the application layer [15, 26, 27, 29, 38]. Husky [29] discusses the definition of RNIC microarchitecture and conducts a holistic study on how different RDMA operations make use of on-NIC microarchitecture resources. Kalia et al. [27] provide guidelines for writing efficient high-performance RDMA programs. These works target understanding or optimizing the RDMA programs and the usage of some specific RNIC microarchitecture resources, but they do not provide RDMA performance isolation.

RNIC design. Several works have been conducted to optimize RNIC design [34, 37, 57, 58]. SRNIC [58] modifies both protocols and RNIC architecture to improve on-NIC memory efficiency and utilization for better scalability. IRN [37] proposes to enable fast loss recovery on NIC to avoid reliance on lossless fabrics. These works contribute to improving RDMA performance. However, our work targets at providing performance isolation for multi-tenant RDMA clouds.

Understanding intra-host communication. Intra-host communication has received increasing attention in research communities [5, 6, 30, 33, 39, 61]. Breaking Band [61] leverages an expensive commercial PCIe analyzer to get a system-level PCIe latency breakdown. Min [36] implements a simple soft PCIe switch to obtain CPU-GPU communication patterns. Neugebauer et al. [39] analyze the PCIe theoretical model and study how PCIe affects network performance. Harmonic targets a different angle. It sniffers intra-host communication traffic to monitor RDMA network behaviors for RDMA performance isolation.

Performance isolation and QoS. Previous research [12, 20, 25, 32, 50, 51] has already provided software-based solutions implemented on the endpoints (hosts) and achieved performance isolation and QoS, by ensuring VM-pair level bandwidth guarantee. However, centering around the TCP/IP kernel network stack, they mainly focus on the bandwidth contention of the network fabric (*e.g.*, switch, router, etc.) and provide pure software solutions to the narrow problem. PicNIC [32] uses the number of CPU cycles spent on the packet processing as a criterion of NIC contention for TCP/IP networks. Harmonic is an orthogonal and complementary research work, with a focus on performance isolation on the RDMA-capable endhost. An end-to-end network performance isolation solution requires isolation mechanisms in different components of the network, including both inter-host network bandwidth and RDMA NIC resources on the endhost. Harmonic provides a microarchitecture-resource-aware solution for the RDMA NIC resource isolation in addition to traditional network bandwidth.

9 Conclusion

We propose the first RDMA performance isolation solution for public clouds, Harmonic, that is aware of microarchitecture resources. Harmonic consists of an FPGA-based programmable intelligent PCIe switch to measure per-tenant RDMA resource usage and an RDMA-friendly rate limiter to modulate RDMA resource per tenant. Harmonic requires no application modification. We evaluate Harmonic using the state-of-the-art test suite for RDMA performance isolation. Our evaluation results show that Harmonic delivers strong RDMA performance isolation in a multi-tenant public cloud setting, compared to all the existing solutions.

Acknowledgement

We thank Xilinx for their technical support. We also thank our shepherd Yizhou Shan and other anonymous reviewers for their insightful feedback. Our work is supported in part by grants from NSF (CNS-2238665 and CCRI-CISE 2213808) and IBM IIDAI, and gifts from Adobe, Amazon, Meta, Intel, and IBM.

References

[1] OFED perftest. https://github.com/linux-rdma/perftest.

[2] RDMA Core Userspace Libraries and Daemons. https://github.com/linux-rdma/rdma-core/.

[3] Advanced Micro Devices. How to implement pcie switch on Ultrascale. https://support.xilinx.com/s/question/0D54U00006cnZ2rSAE/how-to-implement-pcie-switch-on-ultrascale?language=en_US.

[4] Advanced Micro Devices. Use Ultrascale+ PCIe Integrated block as an endpoint PCI to PCI Bridge device. https://support.xilinx.com/s/question/0D54U00006hwlreSAA/use-ultrascale-pcie-integrated-block-as-an-endpoint-pci-to-pci-bridge-device?language=en_US.

[5] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, and Amin Vahdat. Understanding Host Interconnect Congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks (HotNets)*, pages 198–204, 2022.

[6] Saksham Agarwal, Arvind Krishnamurthy, and Rachit Agarwal. Host Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 275–287, 2023.

[7] Amazon. Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/.

[8] AMD/Xilinx. Versal Adaptive SoC Integrated Block for PCI Express LogiCORE IP Product Guide. https://docs.xilinx.com/r/en-US/pg343-pcie-versal.

[9] AMD/Xilinx. Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html.

[10] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, and Eno Thereska. End-to-end Performance Isolation Through Virtual Datacenters. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 233–248, 2014.

[11] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering Azure Storage with RDMA. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 49–67, 2023.

[12] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards Predictable Datacenter Networks. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 242–253, 2011.

[13] Broadcom. Introduction to Thor Congestion Control for RoCE. https://docs.broadcom.com/doc/NCC-WP1XX.

[14] Jingrong Chen, Yongji Wu, Shihan Lin, Yechen Xu, Xinhao Kong, Thomas Anderson, Matthew Lentz, Xiaowei Yang, and Danyang Zhuo. Remote Procedure Call as a Managed System Service. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 141–159, 2023.

[15] Youmin Chen, Youyou Lu, and Jiwu Shu. Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing. In *Proceedings of the 14th European Conference on Computer Systems (EuroSys)*, pages 1–14, 2019.

[16] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 51–66, 2018.

[17] 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 *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 519–533, 2021.

[18] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 323–336, 2011.

[19] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C Snoeren. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 681–693, 2020.

[20] 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 *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 1–12, 2010.

[21] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over Commodity Ethernet at Scale. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 202–215, 2016.

[22] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. MasQ: RDMA for Virtual Private Cloud. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 1–14, 2020.

[23] HiTech Global. PCI Express Gen4 Root FMC+ Module. https://hitechglobal.us/index.php?route=product/product&path=18_85&product_id=273.

[24] Intel. Intel® Ethernet 800 Series Linux Flow Control. https://edc.intel.com/content/www/us/en/design/products/ethernet/800-series-linux-flow-control-configuration-guide-for-rdma-use-c/congestion-management-tuning-parameters/.

[25] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Changhoon Kim, and Albert Greenberg. EyeQ: Practical Network Performance Isolation at the Edge. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 297–311, 2013.

[26] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be General and Fast. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1–16, 2019.

[27] Anuj Kalia, Michael Kaminsky, and David G Andersen. Design Guidelines for High Performance RDMA Systems. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, pages 437–450, 2016.

[28] 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 *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 113–126, 2019.

[29] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R Lebeck, and Danyang Zhuo. Understanding RDMA Microarchitecture Resources for Performance Isolation. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 31–48, 2023.

[30] Xinhao Kong, Jiaqi Lou, Wei Bai, Nan Sung Kim, and Danyang Zhuo. Towards a Manageable Intra-Host Network. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HotOS)*, pages 206–213, 2023.

[31] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding Performance Anomalies in RDMA Subsystems. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 287–305, 2022.

[32] 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 *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 351–366, 2019.

[33] Qiang Li, Qiao Xiang, Derui Liu, Yuxin Wang, Haonan Qiu, Xiaoliang Wang, Jie Zhang, Ridi Wen, Haohao Song, Gexiao Tian, Chenyang Huang, Lulu Chen, Shaozong Liu, Yaohui Wu, Zhiwu Wu, Zicheng Luo, Yuchao Shao, Chao Han, Zhongjie Wu, Jianbo Dong, Zheng Cao,

Jinbo Wu, Jiwu Shu, and Jiesheng Wu. From RDMA to RDCA: Toward High-Speed Last Mile of Data Center Networks Using Remote Direct Cache Access. *arXiv preprint arXiv:2211.05975*, 2023.

[34] 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 *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 44–58. 2019.

[35] Microsoft. Azure Virtual Machine series. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/.

[36] Seung Won Min. *Fine-grained memory access over I/O interconnect for efficient remote sparse data access*. PhD thesis, 2022.

[37] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting Network Support for RDMA. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 313–326, 2018.

[38] Sumit Kumar Monga, Sanidhya Kashyap, and Changwoo Min. Birds of a Feather Flock Together: Scaling RDMA RPCs with Flock. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP)*, pages 212–227, 2021.

[39] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W Moore. Understanding PCIe performance for end host networking. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 327–341, 2018.

[40] NVIDIA. DCQCN PARAMETERS. https://enterprise-support.nvidia.com/s/article/dcqcn-parameters.

[41] NVIDIA. Firmware Burning Tools (MFT). https://docs.nvidia.com/networking/category/mft.

[42] NVIDIA. MLNX_OFED InfiniBand/VPI. https://docs.nvidia.com/networking/category/mlnxofedib.

[43] NVIDIA. NVIDIA ConnectX-6 DX Datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectX-6-dx-datasheet.pdf.

[44] NVIDIA. NVIDIA MELLANOX BLUEFIELD-2 Datasheet. https://network.nvidia.com/files/doc-2020/pb-bluefield-2-smart-nic-eth.pdf.

[45] NVIDIA. Quality of Service (QoS). https://docs.nvidia.com/networking/pages/viewpage.action?pageId=107485812.

[46] NVIDIA. Security Bulletin: NVIDIA ConnectX - April 2023. https://nvidia.custhelp.com/app/answers/detail/a_id/5459/~/security-bulletin%3A-nvidia-connectx---april-2023.

[47] NVIDIA. Single Root IO Virtualization (SR-IOV). https://docs.nvidia.com/networking/pages/viewpage.action?pageId=107485951.

[48] PCI-SIG. PCI Express® Base Specification Revision 4.0. https://pcisig.com/specifications.

[49] 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 *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, page 17–30, 2015.

[50] 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 *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 351–362, 2013.

[51] Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C Snoeren. Cloud Control with Distributed Rate Limiting. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 337–348, 2007.

[52] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. ReDMArk: Bypassing RDMA Security Mechanisms. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security)*, pages 4277–4292, 2021.

[53] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the Data Center Network. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 309–322, 2011.

[54] Konstantin Taranov, Benjamin Rothenberger, Daniele De Sensi, Adrian Perrig, and Torsten Hoefler. NeVerMore: Exploiting RDMA Mistakes in NVMe-oF Storage Applications. In *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2765–2778, 2022.

[55] Shin-Yeh Tsai, Mathias Payer, and Yiying Zhang. Pythia: Remote Oracles for the Masses. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*, pages 693–710, 2019.

[56] VITA. FPGA Mezzanine Card Plus (FMC+) Standard. https://www.vita.com/fmc.

[57] Xizheng Wang, Guo Chen, Xijin Yin, Huichen Dai, Bojie Li, Binzhang Fu, and Kun Tan. StaR: Breaking the Scalability Limit for RDMA. In *Proceedings of the IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11, 2021.

[58] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong Guo. SRNIC: A Scalable Architecture for RDMA NICs. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1–14, 2023.

[59] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. Characterizing Off-path SmartNIC for Accelerating Distributed Systems. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 987–1004, 2023.

[60] Zhuolong Yu, Bowen Su, Wei Bai, Shachar Raindel, Vladimir Braverman, and Xin Jin. Understanding the Micro-Behaviors of Hardware Offloaded Network Stacks with Lumina. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 1074–1087, 2023.

[61] Rohit Zambre, Megan Grodowitz, Aparna Chandramowlishwaran, and Pavel Shamis. Breaking Band: A Breakdown of High-Performance Communication. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP)*, pages 1–10, 2019.

[62] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. Justitia: Software Multi-Tenancy in Hardware Kernel-Bypass Networks. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1307–1326, 2022.

[63] Zhenwei Pi. Redis Over RDMA Implementation. https://github.com/redis/redis/pull/11182.

[64] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 523–536, 2015.

A Harmonic Prototype Setup

We present the prototype setup of Harmonic in Figure 14. PIPS is implemented on an AMD Versal VCK190 FPGA board, connecting to the host system with a PCIe extender card. We connect RNIC with PIPS using an FMC+ expansion connector because the FPGA board does not contain a PCIe root connector interface. FMC+ is built upon FPGA Mezzanine Card (FMC) standard [56] which is a versatile and widely adopted standard for high-performance interfacing FPGAs with external devices.

B Entries for Mappings and Statistics

We illustrate the address-to-object/tenant mapping and statistics entry format in Figure 13. We explain how each field is derived to offer a comprehensive understanding of the mapping mechanism in Harmonic.

B.1 Address-to-Object/Tenant Mappings

There are mainly three types of objects (*i.e.*, memory regions, queue structures, and RDMA metadata) in RDMA. All these objects will be pinned in the host physical memory after creation, and the RNIC will maintain virtual-to-physical address mappings to DMA these objects.

The first type of object is application's memory region (MR). Applications register these MRs through `ibv_reg_mr`, which is processed by `mlx5_ib.ko` drivers in our NVIDIA testbed. We modify `mlx5_ib.ko` to record the starting physical address, the process ID (PID) of the caller, the size and the memory flags (*e.g.*, `IBV_ACCESS_REMOTE_WRITE`, which allows remote write) of this region. Note that we use container's process ID as tenant ID (TID).

The second type of objects is queue structures, including send/receive queues, completion queues, and the doorbell (memory mapped registers) for these queues. When an application initiates RDMA data verbs, the memory is accessed by the RNIC to fetch WQEs from send/receive queues or write completion queue entries (CQE) to completion queues. The memory for these objects is allocated and pinned during the allocation of these projects, such as `mlx5_ib_create_qp` in `mlx5_ib.ko`. Similarly, we record the PID of the caller, and the address and size of these objects.

The third type of object is RDMA metadata managed by RNIC driver and firmware, including QP contexts and memory translation/protection tables. When other two types of objects (*e.g.*, a QP) are created, the firmware reserves a few pinned pages and allocates metadata (*e.g.*, a QP context) in the pinned pages. We record the information on these pinned pages in a similar fashion as described above.

Figure 13a shows our unified entry to update such address-to-object/tenant mappings to our PCIe switch. Note that the

Opcode: 127-112	Rsvd: 111-96	Size: 95-65	Flags: 64-63	Addr: 62-15	TID: 14-3	Type: 2-0

(a) Address-to-Object/Tenant Mapping Entry Format.

# Byte: 63-31	# Access: 30-1	Direction: 0

(b) Statistics Entry Format.

Figure 13: Address-to-Object/Tenant Mapping and Statistics Entry Formats.

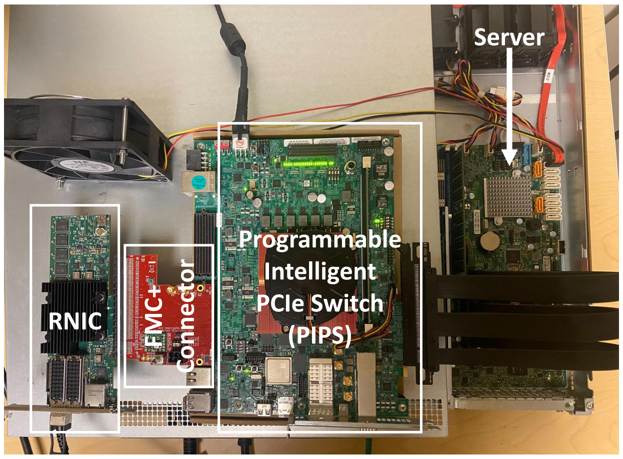


Figure 14: Programmable Intelligent PICe Switch (PIPS) Prototype.

most significant 16 bits of address-to-object/tenant mapping entry together serve as an operation code that notifies PIPS to either insert or delete the entry in PIPS. We reserve the second 16-bit field considering the possibility of other customization demands. Our modified drivers will fill in the remaining five fields and expose these entries to Harmonic daemon through system files.

B.2 Per-tenant RDMA Statistics

We store the monitored RDMA resource statistics in a 128-bit structured entry as shown in Figure 13b. As discussed in §4, the TLP analyzer leverages the physical address enclosed in TLP headers to search and retrieve the corresponding address-to-object/tenant mapping entry from which we identify the object and tenant associated with the TLPs. Then we collect and record PCIe bandwidth consumption, number of PCIe transactions, direction of TLPs, accessed memory type, and other information from TLP headers (Figure 4) in per-tenant statistics entries.

C Harmonic PCIe Overhead Computation

PIPS maintains 40 statistic entries per tenant and each entry is 8-byte. We issue PCIe read request to read these statistics from PIPS. For a PCIe read request, the minimum PCIe protocol overhead is 20 bytes [48]. Upon receiving the read request, PIPS responds with a completion packet (*i.e.*, Completion TLP), containing 8-byte payload and a 16-byte PCIe protocol overhead. Therefore, for a single statistics read, it consumes 800-byte for host-to-PIPS direction 960-byte for PIPS-to-host direction in total.

Assuming Harmonic daemon polls the statistics every N milliseconds. The extra PCIe bandwidth consumed therefore is $\frac{1000}{N} * 8 * 800 = \frac{6.4}{N}$ Mbps for the host-to-PIPS direction and $\frac{7.68}{N}$ Mbps for PIPS-to-host direction. Harmonic currently poll statistics every 100 us, which consumes 64 Mbps and 76.8 Mbps for these two directions. This overhead is less than 0.25% of the total PCIe bandwidth. Together with the extra PCIe bandwidth consumed by updating mappings, the overall PCIe bandwidth overhead of Harmonic is below 0.31% which can be comfortably accommodated by the existing 21.87% bandwidth slack between PCIe and RNIC line rate. Note that we assume our PCIe limit as 32 Gbps, which has the same network-to-PCIe capacity ratio as higher speed networks (*e.g.*, 100 and 200 Gbps). This means that the PCIe overhead of our solution remains negligible with a higher network speed. Not to mention that this PCIe overhead only depends on the number of tenants and the frequency of polling, independent on network bandwidth.