Towards Large-Scale RDMA Networks without Performance Anomalies

by

Xinhao Kong

Department of Computer Science
Duke University

Defense Date: November 14, 2024

Approved:

Danyang Zhuo, Supervisor

Xiaowei Yang

Matthew Lentz

Michael Reiter

Alvin R. Lebeck

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2024

ABSTRACT

Towards Large-Scale RDMA Networks without Performance Anomalies

by

Xinhao Kong

Department of Computer Science
Duke University

Defense Date: November 14, 2024

Approved:

Danyang Zhuo, Supervisor

Xiaowei Yang

Matthew Lentz

Michael Reiter

Alvin R. Lebeck

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2024

Abstract

Remote Direct Memory Access (RDMA) has become a popular networking solution in modern data centers for its capability of delivering high bandwidth, low latency, and high CPU efficiency. Applications, such as machine learning training and inference, as well as remote storage, heavily rely on RDMA networks for their inter-host communication. Entering the new era of artificial intelligence, RDMA is becoming, if not has already become, one of the core components in modern datacenter network infrastructure.

Although RDMA can deliver extremely high performance, fully realizing this potential at large scale remains challenging: (1) specific workloads can trigger slow paths on the host or inside the RDMA NIC (RNIC), leading to unexpected RDMA performance degradation and even threatening the entire data center network. (2) Severe performance interference caused by RDMA-specific resource contention prevents applications from efficiently sharing the network infrastructure. We name these two types of issues as RDMA performance anomalies. These anomalies can lead to catastrophic consequences, such as applications' performance drop, Service Level Agreement (SLA) violation, head-of-line blocking, and even deadlocking the entire datacenter. Therefore, they have to be systematically uncovered and effectively addressed before a large-scale RDMA network starts to serve critical workloads.

Unfortunately, no existing approach can uncover and prevent these anomalies. The root cause lies in the fact that current methods adopt a traditional network perspective, overlooking critical aspects unique to RDMA networks, which possess a highly complex microarchitecture. For instance, RNICs integrate on-NIC processing units and caches to support their hardware offloading capabilities. The lack of microarchitecture-awareness limits the effectiveness and efficiency of existing solutions. Moreover, due to the invisibility of RNIC internals, prior work only has limited, if not none, understanding of the complex RDMA microarchitecture.

This raises an important question: is it possible for cloud operators to gain insight into RDMA's microarchitecture and develop microarchitecture-aware solutions to effectively and efficiently uncover and prevent performance anomalies? In this dissertation, I argue that this is indeed feasible and practical, and being microarchitecture-aware is crucial to achieving these goals. I propose, design, and implement three software systems to support this thesis argument from various aspects:

(i) Collie, a performance anomaly detection system that is the first to use qualitative microarchitecture information exposed by RDMA hardware counters to efficiently uncover RDMA performance anomalies. (ii) Husky, an end-to-end test suite that reveals RDMA microarchitecture resource consumption model and identifies unique performance interference caused by microarchitecture resource contention in RDMA networks. (iii) Harmonic, a first microarchitecture-aware solution that monitors and modulate per application's RDMA microarchitecture resource usage to prevent performance interference and mitigate performance anomalies.

These systems have been comprehensively evaluated across various testbeds, and the results strongly support the proposed thesis statement. These systems and their evaluation results have also been successfully transferred to multiple industry collaborators, making a significant impact on the broader community.

Dedication

To my parents for their constant trust and support;

to my advisor for his invaluable guidance and encouragement;

and to my younger self for his courage and perseverance.

Contents

List of Tables

List of Figures

Acknowledgements

As I sit by the window at home, looking out at the typical North Carolina forest, it feels like everything happened just yesterday. The past three years and a half have been the most dynamic period of my life so far - my Ph.D. journey is filled with uncertainty and moments of self-doubt, yet also with immense growth and joy. I feel incredibly fortunate to have received so much support from so many people. All these fortunes allow me to look back on this time with a sense of happiness and fulfillment. I cannot make this journey without the help and support I receive from my advisor, my mentors, my friends, and my family.

First and foremost, I must express my deepest gratitude to my advisor, Professor Danyang Zhuo. I could easily write a 12-page, double-column paper (like my usual NSDI papers) to elaborate why Danyang is the best advisor. For now, I will keep it brief with an abstract. Danyang has not only provided me with invaluable research vision, insights, and technical knowledge, but he has also put tremendous effort into helping me grow as an independent researcher. Beyond focusing on project progress or publications, Danyang genuinely cares about the personal growth I gain through every effort I make. Whenever I have to make decisions (project, internship, etc.), he always stands by my side and fully supports me to choose whatever I believe is the best for me. He has made me believe that he will always have my back.

What matters even more is that Danyang cares deeply about my happiness as a Ph.D. student. It's inevitable that I occasionally feel overwhelmed by the challenges of research, criticism from reviewers, or just the peer pressure, leading to moments of confusion and frustration. Danyang seizes every opportunity to provide positive feedback. He never lets unmet project expectations result in any negativity, but instead, he always works with me to analyze the situation and explore solutions. Our communication therefore is always clear and transparent, and he consistently offers me trust and encouragement. I often feel that his belief in me surpasses that of anyone else — even myself. I have learned a lot from him how to collaborate effectively with others and how to be a great mentor. All of these lessons will be invaluable to me for the rest of my life. Thank you for everything, Danyang.

I would like to thank my Ph.D. committee members, Professor Alvin R. Lebeck, Professor Matthew Lentz, Professor Xiaowei Yang, and Professor Michael Reiter. My gratitude goes beyond your role on my committee and our collaboration - each conversation we've had during this journey has been a source of profound inspiration. Whether it was refining small details, like how to improve titles on each slide, or larger discussions about research vision and future career choices, your guidance has been invaluable. Your support has shaped my journey in countless ways, and for that, I am deeply grateful.

I would like to express my sincere thanks to Wei Bai, who introduced and brought me to Microsoft, where I happily spent three summers as an intern. Through our collaboration, I have learned many valuable research techniques and insights. Wei consistently recognizes my technical abilities and shows genuine concern for my career development, frequently recommending me for exciting opportunities, which has led to fruitful experiences. He has been incredibly supportive, much like an unofficial co-advisor outside of Duke, offering me valuable career guidance and practical support. Constantly having big dinner and discuss research together at his home is one of my fondest memory back in Seattle. His mentorship and support has been truly instrumental in shaping my career and expanding the impact of my research projects, for which I am truly thankful.

I am especially grateful to my managers at Microsoft, Mahmoud Elhaddad and Ranveer Chandra, who offered essential support and thoughtful advice on my career path as well. Their mentorship has greatly shaped my professional development, and I genuinely hope to have the opportunity to collaborate with them again in the future. I would also like to thank many great and talented people I met at Microsoft Azure: Shachar Raindel, Marina Lipshteyn, Jitendra Padhye, Xiakun Lu, Konstantin Taranov, Qiongwen Xu, Zhuolong Yu; as well as at Microsoft Research: Waleed Reda, Fred Chase, Guo Hu, Stephen Dahl, Kartik Sharma, Pantea Karimi, Liangcheng Yu, Fengchen Gong, Mingran Yang, Yajie Zhou, Weiyang Wang, Isaac Ju, Xuting Liu, Yifei Xu, Zheng Yang, Yuhong Zhong, Chenhao Ye. Working with you has been both productive and joyful, and I am truly grateful for all your

guidance, support, and encouragement.

I would also like to express great thanks to my amazing collaborators: Jiaqi Lou, Jinghan Huang, Professor Nam Sung Kim, Yibo Zhu, Chuanxiong Guo, Jianxi Ye, Zhuo Jiang, Huaping Zhou, and Zhenwei Pi. Your support and collaboration have made this journey an amazing experience, and I am deeply thankful for the time we have worked together. I would like to especially thank Jiaqi. She is diligent, meticulous, responsible, and incredibly kind. I am truly grateful for the knowledge she has shared and the productive partnership we've built. She has taught me so much about hardware, and our collaboration on the Harmonic system has been both enjoyable and highly efficient.

I am grateful to all other members, besides my defense committee, at Duke System Research Group: Professor Jeffrey Chase, Professor Bruce Maggs, Professor Lisa Wills, Jingrong Chen, Entropy Xu, Shihan Lin, Yicheng Jin, Yechen Xu, Jianxing Qin, Shujun Qin, Jiyao Hu, Xiao Zhang, Mason Ma, Yongji Wu, Luka Duranovic, Jack Snyder, Chris Kjellqvist, Avery Blanchard. Working and learning alongside all of you, and growing together as both researchers and friends, has been an incredibly joyful experience. I would like to especially thank Jingrong. We knew each other before we began our Ph.D. journeys, and it was he who introduced me to research in networking and systems. He encouraged me to apply for a Ph.D. and recommended Danyang to me. During my time at Duke, he helped me overcome countless technical challenges — "If you have a technical problem, just ask Jingrong" became one of our slogans. We have also become great friends, sharing everything with each other, often traveling together, and staying up late talking when we shared a room at conferences. No matter what I wanted to share, Jingrong was always willing to listen and enthusiastically offer help whenever I needed it. Jingrong, I am always in your debt.

I would like to extend my heartfelt thanks to the faculty members and staff at Duke who have provided me with invaluable support and guidance throughout my journey. In particular, I am grateful to Professor Xiaoyue Ni from the Department of Mechanical Engineering and Materials Science; and from the Department of Computer Science, Professor

Boyuan Chen, Professor Tingjun Chen, Elizabeth Labriola, Pamela Spencer, Joe Shamblin, Howard Goldsmith, and Jeff Wright. Their assistance and support have been instrumental to my growth and success.

I am fortunate to have been accompanied by so many other close friends in North Carolina: Zonghao Huang, Ziang Chen, Yuxi Liu, Yihao Hu, Haibo Xiu, Yingfan Wang, Yongkang Li, Kaiping Yin, Chenhang Li, Chaorui Qiu, Guozhen She, Yi Zhou, Zhenzhou Qi, Zhengwei Tong, Jeffrey Ching, Sixuan Dang, Feiyang Yu, Yujie Zhang, Jiaxun Liu, Junyu Ding, Shuyi Wang, Shiwei Que, Xinyue Liu, Xiaoquan Liu, Siyu Dong, Henry Lu; and across the world: Bowen Tian, Xintong Xie, Yuande Lin, Junyou Chen, Xinzhi Chen, Hui Shen, and Ruiting Li; Dayou Li, Pinyan Chen, Ruge Chen, Moxin Li, Jiani Wang, Zeyuan Li, Shishu Qiu, Zekun Li, Xuejia Yang, Yuanlin Wu; Yi Cui, Yibo Huang, Yiren Zhao, Yang Zhou, Zhiyao Ma, Peiqing Chen, Yinda Zhang, and Ziqi Pang. Their companionship and support have brightened and enriched my otherwise routine PhD life.

I am most grateful to my parents and all my beloved family members. Growing up in a large, loving family has been one of my greatest fortunes. The love they instilled in me during my upbringing has filled me with a deep sense of kindness and faith toward the world, which has served as a constant source of strength throughout this challenging journey.

Lastly, I want to thank my younger self, who finished this PhD degree in 40 months at the cost of losing more than 40 pounds (for fitness actually). He was brave when feeling afraid, steadfast in moments of confusion, resilient in times of exhaustion, and tough in the face of sadness. You've worked hard, courageous young man, and you deserve this.

1. Introduction

In the dynamic realms of cloud computing and artificial intelligence (AI), demands on underlying computational and network infrastructures have increased exponentially. Data-intensive applications require extremely high network throughput for efficiency. For example, modern large language model (LLM) training tasks involve tens of thousands of GPUs, and each GPU pair needs to transfer hundreds of gigabytes of data per iteration [47]. This requires constant high network bandwidth for the inter-host GPU communication. Latency-sensitive applications, such as disaggregated storage and LLM serving tasks, require ultra-low latency to meet applications' Service Level Agreements (SLA). These increasing demands introduce the critical need for network technologies that can offer both high throughput and low latency.

The Remote Direct Memory Access (RDMA) technology emerges as a compelling solution to these increasing demands, if not the only viable solution. RDMA offloads the network protocol stack from software (*i.e.*, kernel stack) to the network interface card (NIC) hardware. In this way, RDMA achieves kernel bypassing and eliminates redundant context switches to achieve ultra-low latency, such as single-digit microseconds round trip delay for servers within the same rack. In addition, with hardware offloading, an RDMA NIC (RNIC) is able to directly read/write host DRAM and GPU RAM in a peer-to-peer communication. This enables zero-copy transmission and helps to deliver extremely high bandwidth. For example, RDMA can easily saturate hundreds of Gbps bandwidth using only a single core and a single connection. These performance benefits are well-suited to the emerging demands for modern datacenter applications. Therefore, the role of RDMA as a core component for future datacenter network infrastructure is becoming increasingly pronounced.

1.1 Performance Anomalies in RDMA Networks

Although RDMA networks are designed to deliver high throughput and low latency, achieving these benefits in practice can be challenging. The primary challenge lies in the prevalence of performance anomalies in RDMA networks, which can be categorized into two main types: unexpected performance degradation and severe RDMA-specific performance interference.

1.1.1 Unexpected RDMA Performance Degradation

Modern data center applications heavily rely on RDMA for efficient inter-host communication. However, the performance of RDMA networks does not always meet their potential for high bandwidth and low latency. In some instances, RDMA networks exhibit unexpected performance degradation that results in reduced bandwidth or increased latency for applications. In more extreme cases, such degradation can completely pause applications or even the entire data center network. For instance, RDMA typically depends on lossless network fabrics to function optimally. In Ethernet-based data centers, RDMA over Converged Ethernet (RoCEv2) protocol is commonly deployed by leveraging pause frames to ensure lossless network fabrics. However, specific workloads may cause a receiver RNIC to send an excessive number of pause frames, leading to head-of-line blocking and potentially deadlocking the entire data center network [39, 144, 49, 90].

Although individual components like RNICs and GPUs have been extensively tested by their manufacturers, the complete RDMA subsystems, including the RNIC as well as other hardware components that it interacts with, may still experience various types of unexpected performance degradation. These issues often arise from the dynamics between the RNIC and other elements of the server hardware. For instance, a specific type of performance anomaly that triggers pause frame storms only occurs under a particular workload on a server with a certain NUMA configuration and a specific RNIC model. Despite the complexity, this combination of hardware and configuration is very common in real data center production environments.

Data center operators therefore have to conduct holistic tests on the entire RDMA networks to ensure the networks are anomaly-free and can provide desired performance for various applications [12]. However, existing test solutions only include microbenchmarks (*e.g.*, Perftest [108]) and running existing representative applications in a small-scale test cluster. Unfortunately, these tests are insufficient and cannot uncover the aforementioned anomalies that cause performance degradation. The main reason is that they only test simple or existing workloads while datacenter applications' workloads change over time. For example, when the configuration for a machine learning training job is changed, the RDMA communication patterns can change and trigger a new type of performance degradation that is not covered by any existing workloads. How to effectively and efficiently uncover these performance anomalies remain challenging.

1.1.2 RDMA-specific Performance Interference

RDMA was originally developed under the assumption of a solitary, trusted user operating within a single cluster (*e.g.*, HPC clusters). This scenario allowed for optimized performance without the need for robust isolation mechanisms between multiple users. However, modern computing has dramatically shifted towards cloud environments, where infrastructure is often shared among multiple users, or tenants, to maximize resource efficiency. In such multi-tenant environments, particularly in public clouds, tenants can range from well-behaved to buggy, and potentially even malicious.

This evolution raises concerns about the second type of performance anomaly, RDMA-specific performance interference in clouds. In a multiplexing environment, multiple applications may share the same RDMA networks and contend on the same type of resources. Without proper isolation, one application may suffer from significantly degraded performance or even denial-of-service attack due to resource exhaustion. Furthermore, the shared nature of the RDMA infrastructure can lead to inadvertent privacy leaks, where the activities of one tenant can reveal confidential information about another. Unfortunately, existing RDMA networks lack the necessary mechanisms to effectively prevent such interference. The main reason is that RDMA's performance interference is very different from traditional networks because of RDMA's unique and complex microarchitecture resources. RNIC has many special hardware components (*e.g.*, on-NIC processing units and on-NIC cache) for its hardware offloading capability. These hardware components are used to process almost all types of RDMA operations (*e.g.*, set up connections or send/receive messages). Therefore, when requests issued by different applications heavily consume the same type of microarchitecture resource, specific types of RNIC internal resources may become the bottleneck and cause severe performance interference.

1.1.3 Challenges to Finding and Preventing Performance Anomalies

Existing solutions fail to effectively and efficiently find or prevent the aforementioned performance anomalies. The main reason is that all existing solutions adopt traditional network perspectives and neglect RDMA's unique microarchitecture characteristics. For example, cache misses occurring on the RNIC can heavily impact the performance and cause either unexpected degradation or interference. Additionally, RDMA workloads are highly diverse, with varying microbehaviors that affect the cache differently. However, current solutions, whether for uncovering or preventing anomalies, fail to consider and leverage such microarchitecture-specific information.

This limitation affects both effectiveness and efficiency: testing methods that ignore microarchitecture information cannot easily identify which workloads are more likely to trigger bottlenecks on the RNICs, limiting their efficiency and effectiveness; similarly an anomaly prevention solution (*e.g.*, performance isolation) that neglects RNIC cache cannot effectively address interference caused by cache contention. Worse still, since RNICs typically provide only limited visibility into their internal details, existing literature has only limited, if not none, understanding of the RDMA microarchitecture. To the best of our knowledge, none of the existing literature has systematically studied RDMA microarchitecture, and no prior work is microarchitecture-aware in either finding or preventing these performance anomalies.

1.2 Thesis Contributions

The gaps in existing solutions raise an important question: is it possible for cloud operators to understand RDMA microarchitecture and integrate microarchitecture awareness into their solutions to effectively and efficiently uncover and prevent performance anomalies? In this dissertation, I argue that *understanding RDMA microarchitecture is feasible and practical, and being microarchitecture-aware is crucial to find and prevent performance anomalies effectively and efficiently.* To support this thesis statement, I have designed and implemented three microarchitecture-aware software systems to identify and prevent RDMA performance anomalies. I have systematically evaluated these systems under various settings to demonstrate their effectiveness and efficiency, and the evaluation results have provided strong evidence in support of the proposed thesis.

This dissertation is structured as follows:

Chapter 2: Background

In this chapter, we provide an in-depth overview of RDMA technology, its key mechanisms, and the emerging challenges it faces in modern datacenter environments. We explore how RDMA has evolved from high-performance computing (HPC) applications to becoming a critical component in cloud infrastructures. Additionally, we examine the current limitations, particularly focusing on performance anomalies that disrupt RDMA's promise of high throughput and low latency. These limitations highlight the necessity of innovative solutions to ensure RDMA continues to meet the growing demands of modern datacenter applications.

Chapter 3: Finding RDMA Performance Anomalies

In this chapter, we take the first step to explore the available microarchitectural information within modern RDMA networks. We then delve into how to utilize this information effectively in practice. Notably, we observe that modern RNICs expose a variety of hardware counters designed for monitoring and diagnostics, including performance and diagnostic counters. Performance counters typically reflect general RNIC performance metrics (e.g., bits per second), while diagnostic counters are often associated with specific unexpected events, such as on-NIC cache miss rates. These hardware counters reveal critical insights into the microarchitectural state of the RNIC and can serve as guidance for identifying performance anomalies. For example, we assume workloads that can drive these counters to extreme value regions are more likely to cause a performance anomaly.

Based on the above observation, we develop Collie, a performance anomaly detection system. Collie constructs an RDMA workload space using standard RDMA programming abstraction (*i.e.*, verbs) and leverages the microarchitectural information provided by these counters to efficiently detect anomalies in the workload space.

Our experimental results highlight the significant impact of microarchitecture-awareness on the effectiveness and efficiency of Collie. We evaluate effectiveness using the number of new anomalies uncovered by the system, as this directly reflects its ability to detect issues overlooked by existing solutions. Efficiency is evaluated based on search speed and the required resources (*e.g.*, the number of hosts needed).

We tested Collie across eight different RDMA subsystems, including six types of RNICs from various manufacturers. Collie operates with minimal hardware requirements, as all our experiments are conducted in an environment utilizing only two hosts. Collie has identified 15 previously unknown performance anomalies within only 10 hours, and all the anomalies have been subsequently confirmed and reproduced by the respective hardware manufacturers. Moreover, Collie provides valuable guidance for developers to build RDMA applications free of anomalies, including an RDMA RPC library and an RDMA-based distributed machine learning framework. The numerous new performance anomalies identified by Collie highlights the practical value of leveraging microarchitectural information in anomaly detection, reinforcing the central argument of this thesis.

Chapter 4: Understanding RDMA Microarchitecture

In this chapter, we take one step further towards demonstrating that a comprehensive understanding of RDMA microarchitecture is both feasible and essential. To achieve this, we design and implement a set of microbenchmarks to investigate how different RDMA microbehaviors uniquely consume RNIC microarchitecture resources. These tests help to reveal a qualitative model of RDMA resource consumption and yield several key insights. This model and these insights deepen our understanding of the root causes of performance interference in RDMA networks. Building on this, we construct a targeted set of workloads comprising both attacker and victim types. The attacker workloads attempt to exhaust one specific type of RDMA resources at a time using specific RDMA operations, and the victim workloads are fine-tuned to be sensitive to different types of resource contention. This attacker-victim test therefore can cause contention on specific types of resources and serve as a fine-grained test suite to identify performance interference and evaluate RDMA performance isolation. We name this entire test suite as Husky.

We apply Husky to study existing commodity RNICs as well as evaluate state-of-the-art performance isolation solutions. Husky presents three findings that can cause severe security vulnerabilities for some commodity RNICs and lead to security bulletin tickets and firmware upgrades [100]. More importantly, Husky victim-attacker test expose weaknesses across all state-of-the-art performance isolation solutions. These experimental results underscore that a systematic understanding of RDMA microarchitecture is both feasible and essential. The effectiveness of Husky is demonstrated in a manner similar to that of Collie. By leveraging awareness of the RDMA microarchitecture, Husky identifies numerous new types of interference previously unrecognized by the entire community, even including the RNIC manufacturers. In addition, Microarchitecture-aware testing not only uncovers impactful findings but also highlight the limitations of existing solutions that are agnostic to RDMA microarchitecture. Furthermore, these results also strongly demonstrate that a microarchitecture-aware approach is necessary to mitigate performance interference and prevent performance anomalies.

Chapter 5: Enabling RDMA Performance Isolation

With a thorough understanding of RDMA microarchitecture in place, we develop Harmonic, a microarchitecture-aware solution designed to prevent performance interference and mitigate anomalies. Harmonic leverages hardware-assisted mechanisms to provide robust performance isolation through two key components: a programmable intelligent PCIe switch (PIPS) based on FPGA and an RDMA-friendly rate limiter. By utilizing the FPGA-based PIPS to monitor PCIe traffic between the RNIC and the host, Harmonic captures all physical address access information. We extend the existing RDMA kernel driver into a Harmonic kernel driver to obtain the mappings between physical memory addresses to RDMA related objects of different applications. In this way, Harmonic is able to map PCIe traffic to per application RDMA behaviors, and therefore accurately monitor per application's RDMA resource usage, including those RDMA-specific microarchitecture resources. Harmonic then leverages a daemon that repurposes the congestion control rate limiters in RNIC hardware for our rate modulation. Based on the monitoring results, the Harmonic daemon generates and sends congestion feedback signals to targeted tenants to modulate its resource usage, preventing performance interference and performance anomalies.

We apply Harmonic to enhance a commodity RNIC and evaluate its efficacy using the Husky test suite and real-world applications. In Harmonic, we evaluate its effectiveness by evaluating its robustness in providing isolation using the Husky test suite. For efficiency, we measure its performance overheads from various dimensions. Our results demonstrate that Harmonic significantly strengthens performance isolation across various types of resource contention and is the first RDMA performance isolation solution to pass the Husky test suite. Moreover, Harmonic incurs minimal performance overhead. This success underscores the central thesis of this work: a microarchitecture-aware approach is essential for effectively preventing performance interference and mitigating anomalies in RDMA networks.

Chapter 6: Conclusion

In this chapter, we summarize our key findings and contributions, review the lessons we have learned, and share the broader impacts of this dissertation. We revisit and validate the thesis statement and conclude this dissertation by discussing a few future work directions.

2. Background

Remote Direct Memory Access (RDMA) technology has been widely deployed in high performance computing clusters for years. RDMA offloads the entire network protocol stack to the network interface card (NIC) hardware, and provides a new set of abstractions (*e.g.*, verbs API) to applications, enabling them to directly access the RDMA NIC (RNIC) and conduct data transfer with minimal involvement of the operating systems.

2.1 RDMA Abstraction

```
 1 // Socket API: Server side
 2 int fd = socket(...);
 3 // Bind and listen
 4 bind(fd, ...);
 5 listen(fd, ...);
 6 // Accept client connection
 7 int clientfd = accept(fd,
       ...);
 8 // Receive and send data
 9 recv(clientfd, buffer, ...);
10 send(clientfd, buffer, ...);
11 // Close connection
12 close(clientfd);
```

```
 1 // Verbs API: Server side
 2 // Create an initial queue pair
 3 struct ibv_qp *qp = ibv_create_qp(...);
 4 // Connect to the remote end
 5 ibv_modify_qp(qp, ...);
 6 // Allocate and register memory region
 7 struct ibv_mr *mr = ibv_reg_mr(..., buffer, ...);
 8
 9 // Post receive and send requests
10 struct ibv_recv_wr recv_wr = {..., buffer, ...};
11 ibv_post_recv(qp, &recv_wr, ...);
12 struct ibv_send_wr send_wr = {{..., buffer, ...};
13 ibv_post_send(qp, &send_wr, ...);
```

FIGURE 2.1: Comparison between Socket API and RDMA Verbs API.

We use Figure 2.1 to compare the RDMA programming abstraction and the traditional socket API, illustrating the overall RDMA workflow. When an application wants to perform RDMA communication, similar to traditional kernel socket-based networks, it first needs to create a connection. In RDMA networks, the connection exists in the form of a queue pair (QP), which consists of a send queue and a receive queue. QP allocation and connection setup still require RNIC kernel driver involvement.

However, RDMA differs from traditional sockets in that applications must register memory before it can be accessed (either read or write) by the RNIC, as shown in line 7. During memory registration, RNIC kernel drivers pin the memory to ensure it resides in the host DRAM, translate the virtual addresses to physical addresses, and send these address mappings to the RNIC hardware. Next, the application can post receive requests to the receive queue (lines 10 and 11) and send requests to the send queue (lines 12 and 13), including the target `buffer` address. Since the RNIC already has the address mapping information, it can translate the virtual `buffer` address to a physical address and conduct DMA operations without involving the kernel or host CPUs. Therefore, subsequent data transfer operations fully bypass the kernel and avoid excessive memory copies. This enables RDMA to achieve extremely high bandwidth, ultra-low latency, and high CPU efficiency.

2.2 RDMA Protocols

There are three major protocols used by commodity RDMA networks, and each protocol is suited to specific network environments.

Infiniband. InfiniBand is a high-speed, low-latency networking standard primarily used in supercomputers, HPC clusters, and data centers. It runs completely different protocols from link layer to transport layer and is typically incompatible with the well-known Ethernet-based networks. It provides the best performance in terms of bandwidth and latency but requires specialized hardware and infrastructure, such as specialized Infiniband switch.

RoCE (RDMA over Converged Ethernet). RoCE is a protocol that allows RDMA to run over Ethernet-based networks. It extends RDMA's high-throughput, low-latency benefits to widely used Ethernet infrastructures, making it accessible for a broad range of applications without requiring specialized hardware like InfiniBand. RoCE comes in two main versions, RoCEv1 and RoCEv2. RoCEv1 is an Ethernet link layer protocol and RoCEv2 extends RoCE to run on top of UDP/IP based network, enabling RDMA traffic to leverage existing IP-addressing infrastructure.

iWARP. iWARP is another RDMA protocol that supports the standard TCP/IP networks. Therefore, iWARP can be easily deployed over traditional TCP/IP network infrastructure and is suitable for environments where minimal changes to the network stack are preferred. However, since iWARP offloads the full standard TCP/IP stack to the hardware, it typically has slightly higher overhead in terms of latency and reduced throughput.

2.3 RDMA Transport

To the best of our knowledge, RoCEv2 is the major deployed protocol in modern datacenters. Therefore, we primarily focus on RoCEv2 protocol in this dissertation. We next introduce a few core features of RoCEv2 transport.

2.3.1 Lossless Fabrics

RDMA implements all transport capabilities in the hardware. For example, to ensure reliable communication, RNIC hardware needs to conduct packet retransmission when packet loss happens. However, due to the limited available resources on the hardware, even the state-of-the-art RNICs today suffer from poor retransmission performance [12, 137]. Therefore, lossless network fabrics are typically required to avoid congestion loss and realize high performance RDMA networks. Priority-based Flow Control (PFC) [109, 39] is the prevalent mechanism to achieve lossless fabrics. PFC works as follows: the RNICs and the switches enabled with PFCs have PFC thresholds for the corresponding ingress queues. When the packets accumulating in the ingress queue exceed certain thresholds, the switch/NIC sends a PFC pause frame to the upstream egress queue, pausing the upstream port for a duration to avoid packet drops in its ingress queue. PFC uses either Priority Code Point (PCP) or Differentiated Services Code Point (DSCP) to identify and manage the priority to pause. In modern data centers, DSCP-based PFC is more prevalent because it leverages layer 3 information and is easier to manage [39]. It is worthwhile to note that though typical RDMA deployment relies on lossless fabrics [12, 31, 55], many efforts have also been made to explore towards empowering RDMA in lossy environments [91, 11].

2.3.2 Congestion Control

Though RDMA networks are typically enabled with PFC to ensure lossless, many congestion control algorithms have been designed to mitigate the triggering of PFCs to prevent high tail latency, head-of-line blocking, or PFC pause frame storms. The prevalent ones include DCQCN [144], TIMELY [90], and HPCC [71]. DCQCN leverages Explicit Congestion Notification (ECN) to indicate congestion, and reaction points (*i.e.*, receiver) send Congestion Notification Packets (CNP) to senders when seeing ECN-marked packets. When receiving CNP, a sender throttles its sending rate based DCQCN parameters set on the host. TIMELY uses round-trip delay as the signal to detect congestion. It adjusts the transmission rate based on RTT variations, making it responsive to changes in the network without relying on ECN. HPCC builds on network telemetry to make more accurate and proactive congestion control decisions. It monitors precise packet-level information using the in-network telemetry, allowing it to react to congestion faster and with finer granularity compared with other methods. However, it relies on advanced features (*e.g.*, INT) on switches, which may not be always available in the underlying network infrastructure.

2.4 RDMA Performance Anomalies

While RDMA (Remote Direct Memory Access) offers promising performance improvements by enabling low-latency, high-throughput communication, achieving this ideal performance is often hindered by various performance anomalies. These anomalies can emerge from several factors, ranging from hardware-specific issues to network congestion, misconfigurations, or incompatibilities between RDMA subsystems.

There are two broad categories of RDMA performance anomalies: performance degradation and performance interference. Each presents unique challenges to RDMA networks and must be addressed to fully realize RDMA's benefits.

Unexpected Performance Degradation. RDMA networks are often deployed with the expectation of delivering ultra-low latency and high throughput. However, real-world performance frequently falls short of these expectations due to various anomalies. For example, specific workloads may trigger hardware issues such as NIC bugs or misaligned hardware configurations, slowing down the NIC processing pipelines. Consequently, applications may suffer from reduced throughput, increased latency, or even get completely paused. Such degradation can be difficult to diagnose, as they are highly dependent on the specifics of the RDMA hardware and workload characteristics. We provide an in-depth description and analysis of this category of anomalies in §3.2.

Performance Interference. RDMA heavily relies on the hardware offloading capability to achieve high performance. Interference therefore arises when multiple applications contend on the same hardware resource on the RNIC. Unfortunately, modern RDMA networks provide neither robust isolation mechanism to mitigate such interference among applications nor clear internal resource model for people to understand the contention. Therefore, unexpected contention can happen and lead to unpredictable performance drops for some applications. For example, our Husky project shows that an application can cause another application's performance to drop more than 50% by simply calling some control verbs (*e.g.*, register memory), without sending any network traffic. The detailed description and analysis of this category of anomalies can be found in §4.2 and §4.3.

2.5 Microarchitecture in RDMA Networks

Compared with the traditional kernel network stack, RDMA networks have complex but critical microarchitecture. There are several core components.

On-NIC Processing Unit (PU). RNICs have their own on-NIC PUs as the core components for the hardware offloading capability. These PUs handle most of the RDMA requests, including processing the work requests, initiate DMA operations, run the protocol stack, etc. In addition, these PUs also process the control requests, such as setting up connections (*i.e.*, QPs), memory registration, as well as error handling. The processing capacities of these on-NIC PUs and how different operations consume such capacities are critical to the end-to-end RDMA performance. For example, specific RDMA operations may trigger the slow paths of these PUs and lead to performance anomalies. Some RDMA operations may consume more PU cycles than others and cause head-of-line blocking.

On-NIC Cache. RNICs also equip with on-NIC cache to accelerate request processing. To process a request, RNIC needs to access various types of metadata, including per-connection metadata (*e.g.*, sending rate as congestion control status), memory translation metadata (*i.e.*, the mapping between applications' virtual address to the actual host DRAM physical address), etc. These metadata are typically stored on the host DRAM, and cached on the NIC to accelerate processing. Given the limited size of the cache on the RNICs, the cache can be full under heavy workloads (*e.g.*, too many connections or wide memory access range) and the RNIC has to access host DRAM to fetch the necessary metadata when a cache miss happens. Such cache misses introduce extra PCIe operations for RDMA operations, leading to resource waste (*e.g.*, PCIe bandwidth) and overheads (*e.g.*, increased delays).

PCIe Switch and PCIe Root Complex. These PCIe components also serve as crucial roles in RDMA network's microarchitecture. On the one hand, the increasing RNIC bandwidth demands more available PCIe bandwidth for its payload and metadata communication. This makes PCIe bandwidth on either PCIe switch or root complex a potential bottleneck for the end-to-end performance. On the other, the configurations on these components heavily affect RDMA's behaviors and have huge impacts on the performance. For example, Access Control Service (ACS) configurations on PCIe switch and root complex affect how RNIC's traffic is forwarded during peer-to-peer communication. When enabled, all RNIC traffic has to be forwarded to the root complex even though it may only need to go through PCIe switch to the other peer (*e.g.*, another GPU connected to the same PCIe switch). This can lead to drastic performance degradation and even harm the entire datacenter.

Unfortunately, prior to the work presented in this dissertation, there was a notable absence of research that carefully studied and analyzed the microarchitecture of RDMA networks. As a result, many related performance anomalies remained largely unexplored, with existing solutions lacking the microarchitecture-awareness necessary to uncover them effectively. More critically, no existing solution could prevent RDMA performance interference or the resulting anomalies, as RDMA microarchitecture was neglected. This gap motivates the development of a systematic approach to understanding RDMA microarchitecture, identifying performance anomalies, and designing microarchitecture-aware solutions to mitigate and prevent these issues. Through this work, we offer tools and frameworks that not only advance academic understanding but have also been successfully adopted by industry collaborators to enhance the performance reliability of RDMA systems.

3. Finding RDMA Performance Anomalies

In this chapter, we introduce Collie, the first core component of this dissertation. In this project, we take the initial step to explore the RDMA microarchitecture. We observe that modern RNICs expose microarchitectural information through hardware counters. We investigate how to apply this information in anomaly detection and build Collie. Collie supports the thesis argument that understanding and utilizing RDMA microarchitecture is essential for effective and efficient performance anomaly detection. For effectiveness, Collie uncovers many new anomalies that have never been found by prior efforts. For efficiency, Collie only needs two hosts and can find many more anomalies with much less given time than random search and other approaches. This work was completed in collaboration with Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo.

3.1 Introduction

Data center applications relentlessly demand low packet latency and high CPU efficiency. That makes Remote Direct Memory Access (RDMA) an appealing solution for cloud providers and other data center operators. Today, many top companies have already adopted RDMA in their data centers [144, 71, 39]. RDMA has been integrated into many application domains, such as graph processing [17, 120], data stores [23, 57], and deep learning [136, 54].

To deploy RDMA in production, i.e., using RoCEv2 for Ethernet-based data center network, we need to make sure that the RDMA network performance can meet our expectations, free of performance anomalies like low throughput and pause frame storm [39, 144, 96, 49]. This is important because applications require high-performance RDMA networks to deliver their service-level objectives (SLO). Furthermore, some abnormal behaviors, like pause frame storms, can cause catastrophic consequences including deadlocking the entire data center network [39, 49, 32, 113].

We have encountered the following anomalies in our RoCEv2 production environment:

- A particular application workload's performance of the same RDMA NIC (RNIC) varies substantially on servers with only a slight difference in their PCIe specifications.
- A specific application workload only triggers pause frame storms with certain NUMA settings on a particular RNIC combined with particular server hardware.
- A particular application workload triggers pause frame storms with only a single connection on a particular RNIC from a particular vendor.

Although we collaborate with the most reliable vendors and they have conducted extensive tests on individual devices, the entire RDMA subsystem still has anomalies. The RDMA subsystem consists of RNICs and other server hardware that interacts with the RNICs. Our observation is that most of the anomalies are highly related to the interactions between RNICs and rest of the server hardware. Additional integration tests are thus critical, and we usually conduct these tests on our own because of two reasons. First, vendors cannot access our highly customized hardware, system configurations, and applications. Second, anomalies are too critical for the reliability and performance of the entire data center network, and we cannot completely rely on third parties for testing.

Currently, there are two approaches to conduct tests over the entire subsystem. The first approach is to run simple test benchmarks (e.g., `Perftest` [108]) to conduct basic throughput and latency tests. The second approach is to run a set of representative RDMA applications. Unfortunately, these two approaches are not able to comprehensively uncover RDMA subsystem anomalies. The fundamental problem is that these approaches only test simple or existing workloads. They therefore fail to capture anomalies comprehensively because real application workloads change over time. In addition, even if an anomaly is found with an application workload, application developers do not know how to modify the workload to avoid the anomaly.

Our goal for this paper is to explore the possibility of systematic search for application workloads that can trigger performance anomalies in RDMA subsystems. Finding these anomalies for the vendors can help them improve their hardware and thus improve the reliability and the performance of the entire data center network. Besides, the systematic approach can help developers understand the conditions to trigger such anomalies and how to avoid them by changing application workloads.

To realize this goal, the first question is how to formally define an anomaly? Having such a definition is difficult because application performance highly depends on the workload and the hardware. In this paper, we focus on two types of performance anomalies that can be precisely defined: no PFC pause frames if the network is not congested and throughput should be bottlenecked either by bits/second or packets/second as in RNIC specification.

Given this definition, we still need to address three challenges. The first challenge is how to build a comprehensive workload search space. An ideal approach for testing with the entire RDMA subsystem is to exactly modeling each component and then construct the search space. However, this is extremely hard for us, given the black-box nature of RNIC and other hardware components. The second challenge is even after we successfully construct a comprehensive enough search space, how can we search efficiently? The search space is inherently very large because RDMA subsystems are complicated. For example, traffics within an RDMA subsystem can be from/to different memory devices (e.g., main memory and GPU memory) and the transportation setting for a given workload is various (e.g., number and type of connections). Conducting tests blindly in such a large space is inefficient. The third challenge is how to find the complicated triggering conditions of such anomalies? This is important both during the search and after the search. During the search, we need the triggering condition to avoid testing similar application workloads for the same anomaly to speed up the search. After the search, we need to use these conditions to help developers avoid anomalies.

To this end, we design and implement Collie, the first tool to systematically uncover RDMA subsystem performance anomalies, with the following three ideas.

Our first idea is to construct the search space from a developer's perspective. Though the underlying hardware is various and opaque to us, the narrow-waist RDMA programming abstractions (i.e., verbs) are clearly defined and stable. All application workloads can be interpreted as a combination of verbs operations. We carefully analyze the standard verbs library and the design decisions developers are allowed to make (the request pattern, how RDMA buffers are allocated, etc.). Moreover, to cover the entire RDMA subsystem, we analyze all the potential data flows within a given server configuration. In this way, Collie constructs a comprehensive search space for application workloads in the domain of RDMA subsystem, including the host of the network traffic (e.g., GPU connected to a different PCIe bridge from the RNIC, DRAM from a different CPU socket), message sizes, number of connections, and memory region configurations.

Our second idea is that we can use two sets of counters to guide the search. The first set is the performance counters (e.g., bits per second), which are provided by all commodity RNICs and other hardware components. In addition, modern commodity RNICs and other hardware components provide diagnostic counters (e.g., PCIe backpressure). Diagnostic counters are mapped to particular unexpected events that happen to the hardware components. These counters are currently only used for debugging and monitoring purposes. Collie uses search algorithms based on simulated annealing to maximize/minimize counter values to uncover anomalies.

Our third idea is to find the minimal area in the search space that covers the found anomalies. We call this area (i.e., the conditions to trigger the anomaly) the minimal feature set (MFS). Collie includes a MFS algorithm to test each feature that an anomaly has (e.g., number of connections) and generate the necessary conditions set. With the MFS algorithm, Collie can further improve search efficiency by avoiding redundant tests of the same area. Also, finding the triggering conditions of an anomaly allows developers to avoid the anomaly by breaking one of the provided conditions.

We evaluate Collie on 8 different RDMA subsystems, including 6 types of RNICs from NVIDIA Mellanox and Broadcom, with speeds between 25 Gbps and 200 Gbps. Before we build Collie, we already know 3 existing performance anomalies by testing with existing RDMA applications. Collie successfully reproduces all of them and has found 15 new anomalies. We report these anomalies to the vendors, and all of them are acknowledged.

7 of them are already fixed by firmware upgrade or detailed configuration following our vendors' instructions. We also describe our experience in using Collie to guide an RDMA RPC library and an RDMA distributed machine learning framework to avoid these anomalies. These experiences show Collie can help data center operators to uncover anomalies and assist RDMA application developers to implement better applications.

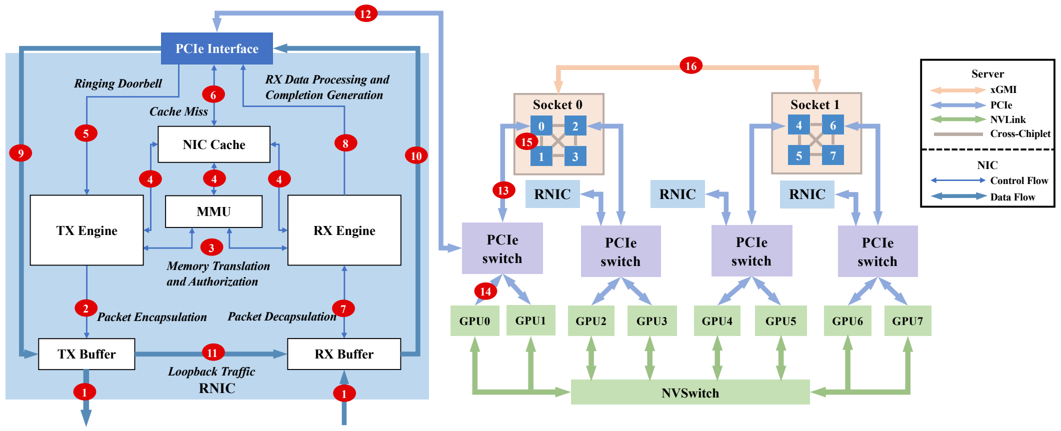


FIGURE 3.1: An example of an RDMA subsystem (RNIC internal design and its deployment environment in a server). Red circles mean potential performance bottlenecks that can trigger performance anomalies.

This work makes the following contributions:

- We design a developer-oriented approach to systematically construct a search space of application workloads to find performance anomalies in RDMA subsystems.
- We propose the first work to leverage hardware counters to guide the search for performance anomalies. These counters do not have proprietary hardware knowledge. This makes Collie general and useful for all types of RDMA subsystems.
- We develop a simulated annealing based search algorithm and MFS algorithm. These algorithms speed up search and help developers avoid anomalies.
- We implement Collie, the first tool to help data center operators to uncover and avoid RDMA subsystem performance anomalies. Collie has found 18 anomalies (3 known

ones and 15 new ones). We present these anomalies, their mitigation strategies, and their implications.

3.2 Background

3.2.1 RDMA Subsystem Performance Anomalies

RDMA is increasingly deployed in data centers for applications to achieve high throughput and high CPU efficiency. An application process can directly communicate through an RNIC with a remote process without involving either side's CPUs. RDMA requires a lossless network to achieve high performance. The default technology to deploy RDMA for Ethernet-based data centers is RoCEv2 [39, 144]. It relies on Priority-based Flow Control (PFC) [109] mechanism to guarantee a lossless network: once an ingress queue length exceeds a threshold, the switch/NIC sends out a PFC pause frame to the upstream egress queue, asking the egress queue to pause for a duration to avoid ingress queue overflow.

RDMA subsystem performance does not always meet user expectations and can have severe performance anomaly. According to our production experience, specific application workloads can trigger hardware bottlenecks of a particular type of RDMA subsystem and cause the entire subsystem performance to drop drastically. Applications of the same subsystem will be affected (e.g., throughput drop) and miss the service level agreement. Worse still, an anomalous RDMA subsystem can send out a large amount of PFC pause frames, which pauses the priority queue of the corresponding switch port and may threaten the entire data center network, such as causing head-of-line blocking and PFC deadlocks [39, 49, 90]

Though the vendors of RNICs and other hardware components (e.g., GPU, motherboard) have conducted extensive tests on their products, we still find many anomalies in our RoCEv2 production environment. The fundamental reason is that RDMA performance is highly related to the entire RDMA subsystem, consisting of both RNIC internals and other hardware components. Figure 3.1 shows the complexity of an RDMA subsystem.

This figure is based on public resources [83, 123, 96] and does not expose proprietary information. Our conversation with Mellanox indicates that a real RNIC is much more complex than our figure shows. To the best of our understanding, an RNIC has at least 6 components: (1) a TX engine that receives doorbells (a signal mechanism for the server to notify RNIC to send a request), fetches and processes requests, and initiates transmission; (2) an MMU that translates the virtual address to physical address for RDMA memory regions; (3) an SRAM-based NIC cache that caches per-connection metadata and memory translation table; (4) a RX engine that processes incoming data and generates completion to notify server; (5)(6) buffers that hold packets to transmit and received packets. An RNIC is connected to a server via PCIe. The server has two CPU sockets and each CPU socket has four CPU chiplets (Only AMD CPUs and new-generation Intel CPUs have cross-chiplet communication, otherwise all the cores inside a CPU socket share the last-level cache.) RNICs and GPUs are all connected to PCIe switches.

There are many potential performance bottlenecks inside the RNIC and between the RNIC and other hardware components within the RDMA subsystem. We use red circles to show such potential bottlenecks (in Figure 3.1). When these bottlenecks are triggered, the network performance may drop and the RNIC can even send out pause frames to reduce the amount of traffic going through the RNIC. We find that many anomalies only occur when multiple bottlenecks or the bottlenecks between different components are triggered. For example, when the RNIC receives a packet, it will store the packet in RX buffer, process the packet (circle 7), and finally DMA the content to main memory or GPU memory (circles 10, 12, 13 or circles 10, 12, 14). Normally, the RX buffer won't accumulate much because the PCIe bandwidth is larger than RNIC's line rate (circle 1). However, once there exists loopback traffic (e.g., the client and server are collocated on the same host), the loopback traffic (circle 11) may drain the PCIe bandwidth and cause RX buffer accumulation. It depends on both the RNIC and the PCIe slot. The worst consequence is that the RNIC keeps sending a large amount of PFC pause frame and threatens the entire data center network. Vendors' individual tests are not able to uncover this anomaly because it depends on the combination of circles 1, 11, 12 (even more) from different components. Further, data center operators like us may use highly customized hardware or specific system configurations that are not accessible to vendors. This makes it necessary and crucial for us to conduct our own independent tests before deploying RDMA hardware in production, especially for anomalies that can potentially generate pause frame storms.

3.2.2 Existing Approaches

Data center operators' tests are integration tests: instead of testing individual hardware components, these tests focus on the performance of the entire RDMA subsystems. There are two existing approaches. The first approach is to run a set of test traffic, such as `Perftest` [108] and `OSU micro-benchmarks` [105]. The second approach is to run a representative set of real applications. However, these two approaches can not uncover RDMA subsystem performance anomalies comprehensively. For example, we deploy 200 Gbps RNICs in our clusters to support a performance-critical distributed machine learning framework. We test the machine learning framework on the cluster of these RNICs, and there is no performance anomaly found. We also have done extensive testing both with synthetic testing workloads and other real applications before deployment. However, months after deployment, our developers find that the performance of the framework has reduced significantly, even worse than just using 100 Gbps RNICs. At the same time, a substantial amount of pause frames are generated from these 200 Gbps RNICs. This is strange because pause frames usually appear with hundreds of connections that trigger congestion, but our machine learning framework only creates a few connections between each server pair. We stopped the machine learning framework and ran our performance tests again, and everything is normal. After several weeks of careful debugging, we finally realize that the case only happens when the application (1) use one-sided RDMA operations with Reliable Connection, (2) has bidirectional traffic, (3) uses a particular workload including a mixture of small and large messages, (4) with 200 Gbps RNIC on particular AMD servers. We find that the developers for our machine learning framework slightly modified their code after

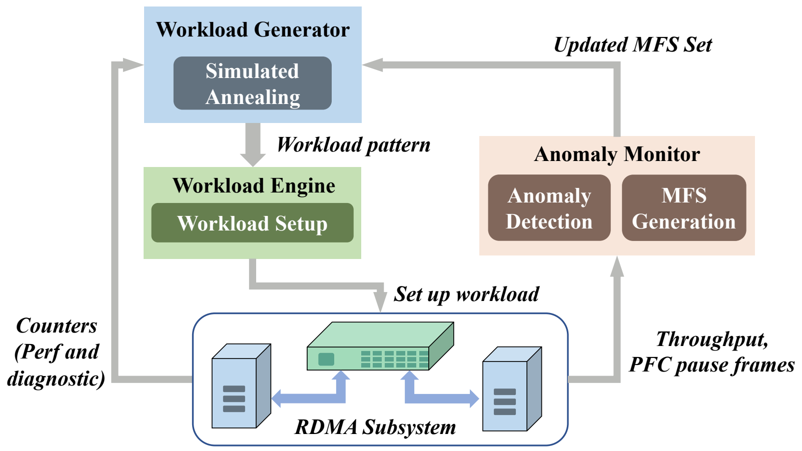


FIGURE 3.2: Collie system overview. The workload engine sets up RDMA traffic. The anomaly monitor detects performance anomalies and their minimal feature sets. The workload generator fetches hardware counters and decides the workload pattern to test.

passing our application tests. The new workload contains messages of mixed lengths (i.e., a large message followed by a small message followed by a large message in bidirectional traffic), which triggers a performance bottleneck between the RNIC and the PCIe switch. This is not a problem with 100 Gbps RNICs from the same vendor or on other types of servers.

The fundamental reason why current approaches fail to uncover such anomalies is that they only test existing workloads and inherently are not able to capture anomalies triggered by unknown workloads. However, real application workloads are various and will change over time. Besides, even current approaches have found such anomalies, it is hard and time-consuming to locate the triggering conditions. Capturing the triggering conditions of performance anomalies allows data center operators to work with vendors to fix potential hardware/firmware bugs, and improve the reliability and performance of the data center network. When fixes to the anomalies are not immediately available (e.g., firmware upgrade, hardware replacement), application developers can build high-performance RDMA applications by avoiding workload that can trigger anomalies.

3.3 Overview

We build Collie, the first tool to help data center operators systematically search for application workloads that can trigger performance anomalies.

The first question we need to answer is how to define an anomaly? Unfortunately, today there does not exist such a definition. Having such a definition is fundamentally hard because application performance (e.g., latency) can be highly dependent on the workload and the hardware. Instead of trying to capture the entire set of anomalous behaviors, we focus on two types of performance anomalies that are of great importance in production environment and can be precisely defined: when applications keep injecting RDMA traffic into the network, (1) no PFC pause frames should be generated if the network is not congested; (2) throughput should be bottlenecked either by total bits/second or total packets/second as in RNIC specifications. The first definition ensures that an RDMA subsystem will not threaten the entire data center network and the second ensures that an RDMA subsystem's capability matches user expectation. [1] We discussed this definition with several hardware vendors, and they all agree with our definition. Even though some anomalies may be due to system limitations rather than bugs, it is also important for both vendors and us to be aware of them. We report all the anomalies we found using this definition to the hardware vendors, and they acknowledged all the reported anomalies. We believe that this definition naturally matches the application developer's mental model of RDMA and thus allows developers to roughly estimate the network performance.

Given this definition of anomaly, we still need to overcome three major research challenges.

Challenge #1: How to design a comprehensive workloads search space for a given RDMA subsystem? An ideal solution is to carefully analyze and modeling the entire RDMA subsystem, and then construct the search space from the perspective of hardware. This

[1] We do not use latency as a metric to define anomalies. The only latency specification for RNICs is the latency under zero load. We did not observe any anomaly in this way, probably because the RNIC is not stressed. However, when RNIC is stressed, it is hard to accurately define the correctness of latency or tail latency due to queuing delay.

complete white-box approach allows us to test all bottlenecks and the combinations of them givens an RDMA subsystem. However, it is impractical for data center operators like us due to the black-box nature of RNICs and other hardware components. Our key observation is that though the components of RDMA subsystems are black boxes and there are diverse RDMA applications, the abstractions between the hardware and applications are clearly defined and stable. All application workloads are essentially composed of a series of basic verbs operations, a narrow waist of the RDMA programming. With this observation, we carefully analyze this RDMA programming abstraction and design a general search space (§3.4).

Challenge #2: How to search efficiently? Due to the complexity of RDMA subsystems and the variety of workloads, the size of search space is very large. Unfortunately, none of existing heuristic search algorithms can be directly applied due to the lack of a search signal (e.g., direction for the next workload to test). We observe that there are two sets of counters commodity RDMA subsystem provide can be leveraged to guide the search. The first set is known as performance counters. For example, all modern RNIC provide the counter of bits sent per second for monitoring purpose. The second is known as diagnostic counters. Modern RNICs and other hardware components expose diagnostic counters for debugging purpose (e.g., the counter indicates PCIe backpressure and NIC internal cache miss) [99, 104]. Diagnostic counters are more informative. For example, when some bottlenecks of the RDMA subsystem are triggered, the performance may not drop while the corresponding diagnostic counter has increased. However, using diagnostic counters typically requires vendor's assistance, and the number of diagnostic counters customers can access depends on vendors. For Collie to be general, we use both performance counters and optionally diagnostic counters as search signals. We conduct the efficient search by using a simulated annealing based algorithm to drive these counters to extreme value regions (§3.5.1).

Challenge #3: How to find the set of conditions to trigger anomalies? Some anomalies are complicated and only occur when many features co-exist, such as a certain type of transportation, particular message pattern, lots of connections, and specific batching operations. We invent a minimal feature set (MFS) algorithm to detect each factor's contribution to the uncovered anomaly and construct the necessary conditions set. To search efficiently, we use MFS to avoid testing similar workloads that map to the same anomalies. After the search, developers use the MFS to understand the triggering conditions for each anomaly and bypass them accordingly when the fixes are temporarily unavailable (§3.5.2).

Figure 3.2 shows our system design. Collie consists of three core components: (1) a workload engine that conducts experiments on RDMA subsystems by setting up RDMA traffic; (2) an anomaly monitor that detects performance anomaly and MFS to reproduce the observed anomaly; and (3) a workload generator that decides the next workload pattern to experiment based on the counters collected in the RDMA subsystem and the current search space. All the experiments Collie does are on the RDMA subsystem with two servers with RNICs, connected with a commodity switch.

3.4 Search Space and Workload Engine

There are two types of factors that can affect an RDMA subsystem performance in deployment. First, we need to consider the application workloads. These include host topology (i.e., where does traffic come from inside a server), how many memory regions the application registered, what transport applications choose to use, and the message patterns. Second, we need to consider the network behavior, for example, congestion on switch and packet loss rate. Currently, our paper focuses on constructing a comprehensive search space for the first factor. For the network behavior, we consider a simplified environment: two RNICs connected by a single switch, and there is no packet drop on the switch. Collie can be easily generalized to test more complicated environments.

We take the bottom-up approach to construct a comprehensive search space for various application workloads. We decompose application workloads into combinations of basic RDMA operations and construct the search space based on these combinations. Figure 3.3

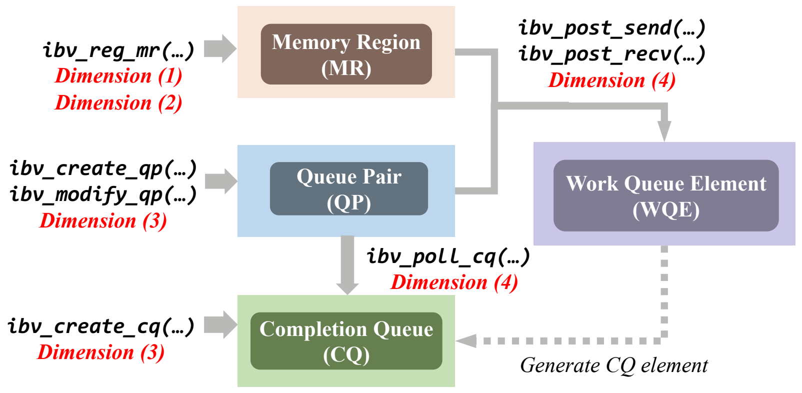


FIGURE 3.3: RDMA programming abstractions.

shows the key abstractions and operations of RDMA programming. These are only high-level software abstractions exposed by standard verbs API and we do not need to know how these high-level abstractions are implemented in the RNIC. In this way, the search space is more comprehensive and general because it does not rely on either extra proprietary RDMA subsystem hardware knowledge or specific application features. In addition, the combinations of verbs operations are inherently able to describe workloads of both single application and co-existing scenarios.

We examine the RDMA programming model at first and extract four search dimensions that jointly describe the application workloads of the entire subsystem. To send a message through RDMA networks, applications first need to register a set of memory regions (MRs), using `ibv_reg_mr`. Once registered, an RNIC has the right to directly access these MRs without CPU involvement. Second, applications create a set of queue pairs (i.e., connections in traditional networking terminology), using `ibv_create_qp` and `ibv_modify_qp`. Applications need to choose a transport type for each queue-pair (QP). There are three standard types of QPs: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). Applications can use `ibv_post_send` or `ibv_post_recv` to post a list of Work Queue Element (WQE). Each WQE represents a work request and has a scatter-gather (SG) list. Each SG list contains a list of entries and each entry designates a contiguous memory region that is within the registered memory regions. A WQE can notify the RNIC to READ/WRITE remote memory (1-sided operation) or SEND/RECV local memory to/from a remote server (2-sided operation). To know that a WQE is complete, the application can register a completion queue (CQ) using `ibv_create_cq`, and the application can call `ibv_poll_cq` to poll on the CQ to get completion queue elements (CQE). Given this RDMA programming model, we extract the following search dimensions.

Dimension 1. Host Topology. Host topology describes how traffic flows to/from an RNIC to/from other server hardware components. Individual component tests are hard to cover this dimension while the topology has a huge impact on RDMA subsystem performance. For example, traffic can be from NUMA-affinitive DRAM or from a GPU that needs to traverse both PCIe and SMP interconnect between NUMA nodes. The latter will have a longer data path and therefore higher average DMA latency. This will trigger PCIe backpressure to the RNIC and may induce performance anomalies under some specific application workloads. We list all accessible memory devices for this dimension.

Dimension 2. Memory Allocation Settings. Traditional RDMA testing is not comprehensive for this dimension, while the memory allocation settings are crucial for RDMA subsystem performance testing. First, the number of MRs affects RDMA subsystem performance because RNIC has an MMU that translates virtual addresses of memory regions to DMA-capable physical addresses and handles memory protection (e.g., authorization). RNIC only caches a fixed size of entries of the memory address translation table. If too many MRs are registered, it is then likely that the RNIC encounters cache miss and needs to access memory address translation tables on server DRAM via extra PCIe operations. These interactions have an impact on the performance. Second, MRs can have different sizes. This also affects RDMA performance because the size also affects the number of translation table entries. Moreover, many RNICs use Intel Data Direct IO (DDIO) to directly access the CPU's last-level cache. If the access range of an MR is large, it can cause severe cache misses in the CPU's last-level cache [19]. This dimension is bounded because we can set a reasonable upper bound on the number of MRs (200K), and the MR size is bounded by the total amount of memory that can be registered (pinned) in the physical server.

Dimension 3. Transport Setting. Transportation setting is crucial for RNIC performance, and this is well known in the research community [58, 89, 56]. We use the following factors to compose the transport setting: (1) QP type (RC, UC, UD), (2) the number of QPs, (3) the opcode type (SEND/RECV, WRITE, READ), and (4) the usage of SG and WQE. Different QP type with different opcode creates different pressure for the RNIC. For example, UD does not require ACK for each packet, which lessens the RNIC packet processing pressure. However, the SEND/RECV requires pre-posted receive buffers, which puts more pressure on the RNIC cache. The number of QPs also has a great impact on RNIC performance because of the limited RNIC cache. This is known as the scalability problem [56, 96, 19]. How SG list and WQE affect RNIC performance is a bit tricky. RNICs have to consume extra PCIe bandwidth to fetch WQE from the host DRAM [58]. The PCIe bandwidth consumed by WQE becomes substantial under some particular application workloads and can even be the performance bottleneck. We enumerate all the transport types and the opcodes (e.g., RC WRITE, UD SEND). It is practical and reasonable to set an upper bound (e.g., 20K) for the number of QPs because data center operators will hardly set up more connections. The SG list and WQE can be parameterized by this formula: $\sum_{i=1}^{n} m_i = k$, where k denotes the number of messages to send, n denotes the number of WQE and m_i denotes the number of SG elements within the i^{th} WQE.

Dimension 4. Message Pattern. Existing RDMA testing approaches lack flexibility and comprehensiveness, especially for this dimension. `Perftest` [108] only repeatedly send messages of a fixed size and other collective communication benchmarks (e.g., `OSU benchmark` [105]) test RDMA similarly. These simple benchmark traffic are inadequate for RDMA subsystem testing because they ignore the interaction among different requests (i.e., WQE) in a sequence.

Our ideal goal is to construct this dimension that can represent any application message pattern. However, it is impractical because application traffic can be very different and the interaction among different requests is unknown given the black-box nature of RNIC. We therefore construct this dimension in the following way. We build a request vector with n elements, where each element describe the request attribute (e.g., size of the message to send). We assume that the 1^{st} request affects the 2^{nd}, the 3^{rd}, ..., the n^{th} requests but won't affect the request after the n^{th}. The larger n we set the larger search space we can cover, but we also need to consume more time. This kind of trade-off is similar to the approach when testing file systems [79], where researchers test fixed-length file system operation sequences. Modern RNIC has limited Processing Units (PU) and pipeline stages [116], restricting the number of outstanding requests an RNIC can process. We thus set n to be the product of the number of PUs and the pipeline stages. We discretize request size into multiple discrete value regions based on MTU and the burst size of the RNIC. The RNIC splits a long request into multiple bursts and processes each burst at one time to avoid Head-of-Line (HoL) blocking. The granularity can be easily modified. With more search time, we can discretize request size in a more fine-grained way. Message inter-arrival time is usually considered as a parameter for application workloads. However, adding the inter-arrival time will substantially extend our search space, so we temporarily only consider the pattern without such inter-arrival time.

Workload engine. We build a flexible workload engine to conduct tests in our search space. Compared to traditional traffic generation tools, e.g., `Perftest`[2], our workload engine is more flexible and has a holistic view. It can send and receive traffic with particular pre-defined patterns (e.g., a large WRITE request followed by a small SEND request). Besides, it supports various memory and transport settings, which can test the entire subsystem holistically. To test with a point in our search space, Collie first translates a test point's settings into a set of input parameters of the workload engine. For example, the setting of dimension 1 and 2 are translated into memory allocation parameters (i.e., which GPU or NUMA DRAM to use and how many MR to register) of the engine. Then,

[2] Existing tools, e.g., `Perftest`, are arguably not designed for this type of testing. They are performance benchmark tools. However, we are not aware of any other tools can that test RDMA subsystems.

the workload engine will take these input parameters to set up connections and generate traffic.

3.5 Search for Performance Anomalies

The total size of our search space (i.e., the combination of parameters) is on the order of 36. Each experiment we do requires 20-60 seconds, mostly depending on the number of QPs to create and the number of MRs to register. This means we cannot exhaust the search space. One naive approach is to generate random input in the search space. This approach is already much better than existing tests because the design of our search space is more comprehensive than that in existing tools (§3.7). However, similar to typical black-box fuzz testing on software, random inputs can only find few anomalies and cannot efficiently uncover complicated anomalies that require multiple conditions to hold simultaneously.

3.5.1 Workloads Generation

We leverage two types of counters to guide the search. The high-level approach is to use an optimization algorithm to drive counters to extreme value regions by keeping mutating the test workloads. For performance counters, we drive the counters to low-value regions. For diagnostics counters (which map to unexpected events), we drive the counters to high-value regions.

Our algorithm is based on simulated annealing (SA). SA is a probabilistic algorithm to find the global minimum of a given function. The idea is to keep mutating the input in the direction of minimizing a given function. SA calls the function value energy. To avoid getting stuck at a local minimum, SA maintains a temperature value. At the beginning of the algorithm, the temperature is high and SA allows mutating input in the direction of increasing the energy. As temperature decreases during the search, SA is less likely to move the input in the direction of increasing the energy. Finally, when the temperature is below a certain threshold, every mutation of the input must decrease energy. SA finishes when there is no way to mutate the input to make the energy lower.

Algorithm 1 Search for Performance Anomalies

```
Require: $S$: initial anomaly set; $T_0$: a high enough initial temperature; $T_{min}$: the lowest limit of
    temperature; $n$: the number of times SA runs for a certain temperature;
Ensure: $S$: An updated anomaly set;
 1: $P_{old}, M_{old} = MeasureRandomPoint()$; pick a random point, setup traffic and collect metrics
    as $M$
 2: while $T > T_{min}$ do
 3:     for $i = 0$; $i < n$; $i++$ do
 4:         mutate $P_{old}$ for a new application workload $P_{new}$;
 5:         if $MatchMFS(S, P_{new})$ then continue;
 6:         $M_{new} = MeasurePoint(P_{new})$;
 7:         $\Delta E = CompareMetric(M_{new}, M_{old})$;
 8:         if $\Delta E < 0$ then
 9:             $P_{old} = P_{new}$
10:         else
11:             the probability $prob = exp(-\Delta E/T_{(i)})$;
12:             if $rand(0,1) < prob$ then $P_{old} = P_{new}$;
13:         end if
14:         if $IsAnomaly(M_{new})$ then
15:             $new\_mfs = ConstructMFS(P_{new})$;
16:             Put $new\_mfs$ into $S$;
17:             $P_{old}, M_{old} = MeasureRandomPoint()$; pick another random point when a new
    anomaly is found
18:         end if
19:     end for
20:     $T = T * \alpha$; where $\alpha$ is decay factor
21: end while
22: return $S$
```

Algorithm 1 shows our algorithm that is based on SA. We maintain a list of performance anomalies. Each anomaly is an MFS (e.g., an area in the search space) that contains workloads to reproduce the performance anomaly. The search starts from a random workload in the search space, and our algorithm measures the counter values. In each iteration of SA, we mutate the workload in one of our search dimensions (line 4). We test whether the new workload causes a performance anomaly with our anomaly monitor. If so, we run our MFS algorithm to determine the entire area in the search space that belongs to this anomaly. We add the new anomaly to the set and change the current workload to a random one. If the new workload does not trigger a performance anomaly, we measure the point by comparing counter values and decide whether to move the current workload to the new one. We always skip workloads that belong to an existing performance anomaly for efficient search.

Table 3.1: Testbed RDMA subsystems configurations. We use numbers in the name of concrete CPU types for confidentiality.

	RNIC	Speed	CPU	PCIe	NPS	Memory	GPU
A	CX-5 DX	25 Gbps	Intel(R) Xeon(R) 1	3.0 x 16	1	128 GB	-
B	CX-5 DX	100 Gbps	Intel(R) Xeon(R) 2	3.0 x 16	1	768 GB	-
C	CX-5 DX	100 Gbps	Intel(R) Xeon(R) 2	3.0 x 16	1	384 GB	V100
D	CX-6 DX	100 Gbps	Intel(R) Xeon(R) 2	3.0 x 16	1	768 GB	-
E	CX-6 DX	200 Gbps	AMD EPYC 1	4.0 x 16	1	2 TB	A100
F	CX-6 DX	200 Gbps	Intel(R) Xeon(R) 3	4.0 x 16	1	2 TB	A100
G	CX-6 VPI	200 Gbps	AMD EPYC 1	4.0 x 16	2	2 TB	-
H	P2100G	100 Gbps	Intel(R) Xeon(R) 2	3.0 x 16	1	384 GB	-

Our algorithm extends the standard SA algorithm in several important ways to adapt it for our context. First, we compute the energy in the following way: assuming the counter value changes from A to B, we set the different in energy (ΔE) to be $\frac{B-A}{A}$ for performance counters and $\frac{A-B}{B}$ for diagnostic counters because we are minimizing performance counters and maximizing diagnostic counters to trigger potential anomalies. This also allows us to avoid value region problem (e.g., the value regions of diagnostic counters are sometimes opaque). Second, we do not require SA algorithm to find the actual global optimum because we care about all potential anomalies. We therefore always set a more relaxed temperature and α that enable the algorithm to jump out of a certain stage even when it has already run lots of iterations. In addition, we maintain a set of performance anomalies (i.e., MFS). When mutating the point, we compare the mutated point with our existing MFS (line 5). Each MFS contains a list of parameters ranges. If the mutated point matches all parameters ranges of an MFS (i.e., the parameter value of this point is in the MFS's range), we claim this point belongs to the MFS and skip testing it. This ensures that the future search does not redundantly test workload already covered by the existing set of anomalies.

3.5.2 Anomaly Monitor

Our anomaly monitor detects performance anomalies and computes the MFS of them.

Anomaly Detection Condition. We use two conditions to detect anomalies. First, if any pause frame is generated. Here we use a metric called pause duration ratio. If the pause duration ratio is 1%, this means for every second, transmission is paused by 10 ms. We set our threshold to be 0.1%. The reason is our experiment platform only has two servers and our switch that connects the servers support line rate traffic, so there is no network congestion to begin with. We set the threshold to be above 0, because RNIC may generate a few pause frames when the memory bus or PCIe bus is busy temporarily, especially when connections are just set up. Second, each RNIC has its maximum bits per second and maximum packets per second in its specification that can be easily verified by running simple benchmarks. Without performance anomalies, network traffic should be restricted by either one of these upper bounds. If a workload's throughput (in terms of both metrics) is 20% lower than the upper bounds, it means that the performance is likely to be restricted by some other bottlenecks of the RDMA subsystem. Collie reports this and runs the MFS algorithm below.

Minimal Feature Set (MFS). After we detect an anomalous workload, we need to know what features of this workload actually trigger the anomaly. For example, if we currently find a new anomaly that has 5 features. It may be the case that 3 features are already sufficient to reproduce this anomaly. One approach is to use machine learning based algorithms to generate decision trees or deep neural networks to locate the area in the search space for the anomaly. However, machine learning approaches usually require much more training data and thus many more hardware experiments.

We instead use a heuristic approach. Since we only have 4 search dimensions with few factors, we just do a few tests on each dimension to determine whether a factor belongs to the MFS. For example, if our search algorithm finds a certain workload using UD can cause a performance anomaly. We test whether the same workload with RC and UC can cause performance anomalies. If not, UD belongs to the MFS because it is necessary to reproduce the anomaly. To determine the MFS of a dimension that is continuous (e.g., number of connections), we discretize them manually into a set of value regions and test each of them. Finer-granularity discretization is acceptable because MFS algorithm only runs when uncovering a new anomaly and the number of anomalies is relatively small compared to the entire search space.

We report all the anomalies to RNIC vendors and we can wait for their fixes. Unfortunately, the solutions to these anomalies are case by case. Some anomalies require vendors to spend a substantial amount of time on coming up with solutions and the solutions may not be applicable for data center operators immediately, such as hardware replacement. Hence, developers need to avoid such anomalies instead of waiting for a fix. Collie provides MFS to help developers avoid such anomalies by changing application workload to break the conditions in the MFS.

MFS helps developers to avoid anomalies in two areas. The first one is anomaly prevention. Before an application is implemented, Collie lets developers restrict the search space using their knowledge of their applications to represent all the possible workloads. Then, Collie outputs whether the restricted search space contains performance anomalies. If not, assuming the developers' understanding of their applications is correct, the application won't encounter any performance anomaly found by Collie. The second one is debugging. When an existing application unfortunately encounters anomalies, we can run Collie on the RDMA subsystem and generate all the MFS. Comparing the application with the generated MFS, Collie provides several suggestions that help to break the triggering conditions. We present two real cases to show how MFS helps developers in §3.7.3.

One caveat of our approach is that we are not able to know the root causes of these anomalies given the black-box nature of the RNICs and other hardware components in the RDMA subsystem. This means it may be the case that multiple MFS are actually due to the same anomaly (i.e., the same hardware bug). This is acceptable because the goal of MFS is to accelerate the search algorithm by eliminating redundant test cases and help developers understand what features of the workloads can trigger the anomaly. We anyway need to report all the anomalies (i.e., all the MFS) we found to the vendors and that is also the best we can do given the RNIC black-box hardware nature.

Table 3.2: Performance anomalies found on subsystem F and H with the necessary conditions to trigger them. Anomalies marked with green color are new anomalies found by Collie. Rest are the anomalies we know before building Collie.

	RNIC	Direc.	Transport	MTU	WQE	SGE	WQ depth	Message Pattern	# of QPs	Symptom
🟩 #1	CX-6	-	UD SEND	-	⩾64	-	⩾ 256	-	-	pause frame
🟩 #2	CX-6	-	UD SEND	-	⩽8	-	⩾1024	⩽1KB	⩾≈16	low throup.
🟩 #3	CX-6	-	RC READ	1K	-	-	-	⩾16KB	-	pause frame
🟩 #4	CX-6	Bi-	RC READ	-	⩾32	⩾4	-	-	⩾≈160	pause frame
🟩 #5	CX-6	-	RC SEND	1K	⩾64	-	⩾1024	⩾2KB and ⩽8KB	-	pause frame
🟩 #6	CX-6	-	RC SEND	1K	⩽16	⩾2	⩾1024	⩽1KB	⩾≈32	low throup.
🟩 #7	CX-6	-	RC WRITE	-	No	-	-	⩽1KB and ⩾≈12K MRs	-	low throup.
🟩 #8	CX-6	-	RC WRITE	-	No	-	⩽16	⩽1KB	⩾≈500	low throup.
🟧 #9	CX-6	Bi-	-	-	-	⩾3	-	mix of ⩽1KB & ⩾64KB	-	pause frame
🟩 #10	CX-6	Bi-	RC WRITE	-	⩾64	-	-	mix of ⩽1KB & ⩾64KB	⩾≈320	pause frame
🟩 #11	CX-6	Bidirectional cross-socket traffic on particular AMD servers								pause frame
🟧 #12	CX-6	Particular GPU-Direct RDMA traffic on particular servers								pause frame
🟧 #13	CX-6	Co-existence of loop traffic and receiving traffic								pause frame
🟩 #14	P2100	Bi-	RC	4K	-	⩾4	-	-	⩾≈1300	low throup.
🟩 #15	P2100	-	UD SEND	-	-	-	⩾64	-	⩾≈32	pause frame
🟩 #16	P2100	-	RC READ	1K	⩾8	-	-	-	⩾≈500	pause frame
🟩 #17	P2100	-	RC SEND	-	⩽16	-	⩾128	⩽1KB	⩾≈64	pause frame
🟩 #18	P2100	Bi-	RC	1K	⩾32	-	-	⩽64KB	⩾≈30	pause frame

3.6 Implementation

The workload generator and the anomaly monitor are written in ˜2100 lines of Python. The workload engine is implemented with ˜2000 lines of C/C++. We directly use monitor tools from vendors to collect hardware counters (both performance and diagnostic counters) from the RDMA subsystem.

The workload engine is implemented with the verbs API and rdma-core-34.0 libraries[115]. In deployment, the Mellanox RNIC uses mlx5 driver (OFED 5.2-1.0.4.0) and the Broadcom RNIC uses bnxt driver (1.10.1.216.2.89.0). The workload engine set up connections by TCP out-of-band transmission. When all connections are set up, the engine starts to generate workload.

The anomaly monitor collects primary metrics, such as throughput and pause frame duration, four times per iteration. It first decides whether the traffic is stable and then compares the primary metrics (e.g., bits per second, packets per second) with the predefined thresholds.

The workload generator collects counters using monitors provided by vendors. These monitors provide counters every second. Collie fetches these counters four times per iteration and uses the average results.

3.7 Evaluation and Experience

We evaluate Collie on 8 different RDMA subsystems. Table 3.1 shows the hardware and related configurations. We use the same anomaly detect conditions as described in §3.5.2

3.7.1 Performance Anomalies Found

Before we build Collie, we already know 3 existing anomalies. Collie can find all the existing ones and find 15 new anomalies. All of them are reported to our vendors and are acknowledged by them. Table 3.2 shows the 18 anomalies. We only present those found on subsystem F and H because anomalies found on other subsystems are subsets of those found on F. §3.11 provides details about these anomalies, including the exact workload, as well as the explanations and solutions from vendors. Here we choose two tricky anomalies to show the importance of Collie's systematic search.

Anomaly #4: Bidirectional RC READ with large WQE batch size, long SG list, and a few connections causes PFC pause frames. Our vendors have successfully reproduced this anomaly in their environment using Collie's traffic generator and acknowledged it, but currently there is no fix. This anomaly cannot be found by existing approaches such as using `Perftest` to generate workloads, because `Perftest` does not support flexible WQE and SG list batching strategies. Though `Perftest` is not designed for this purpose, it is the prevalent tool to uncover performance anomalies. To the best of our knowledge, we don't see any other state-of-the-art work address this problem, which also shows that Collie is the first work to fill this vacancy.

Anomaly #10: Bidirectional RC WRITE with large WQE batch size, particular message pattern, and a few connections causes PFC pause frames. This anomaly is not captured by existing approaches (e.g., running current applications) but we successfully reproduce it by slightly modifying our production RDMA RPC library: when users call the library to send a message, it will try to send as many messages as possible in a WQE batch. The batch size is highly dependent on the timeout value. If the application is throughput sensitive rather than latency sensitive, the timeout value can be set high, which allows a larger batch size. Currently the timeout value is set small because most applications supported by this library are latency sensitive. However, by changing this value we successfully enlarge the WQE batch size and the conditions of #10 are all met. This shows the importance of the anomalies found by Collie, as well as how Collie can capture those anomalies missed by existing solutions.

We try our best to reproduce the anomalies found by Collie using existing workload generators (e.g., `Perftest`), only 4 of them (#3, #8, #13, #15) can be reproduced with very careful parameters tuning. Rest anomalies are all outside the search space of existing approaches.

3.7.2 Running Time for Anomaly Search

To evaluate the efficiency of performance anomaly search, we compare Collie with two baselines: (1) random input generation in our search space and (2) Bayesian Optimization (BO), a widely used method in search problem [95]. We implement the BO approach based on [95]. We set the counter values as BO's optimization target. Our vendors provide us with 9 diagnostic counters. For Collie and BO, we first generate 10 random points. We then compute the standard deviation over the mean of the counter values collected in the first 10 run and use the result to rank these diagnostic counters in decreasing order. Both Collie and BO optimize each diagnostic counter in this order. For a fair comparison, we use MFS to enhance BO as well. In this section, we use subsystem F as an example. We run each search for 10 hours.

Figure 3.4 shows the running time to find performance anomalies. Random input (i.e., fuzzing) can already find 7 anomalies that only require simple conditions to trigger. BO does improve efficiency but to a very limited extent. BO can speed up the search process but only find 8 anomalies with the given time. We analyze the optimization process of BO and find that it is not able to optimize the corresponding counters. Our guess is that

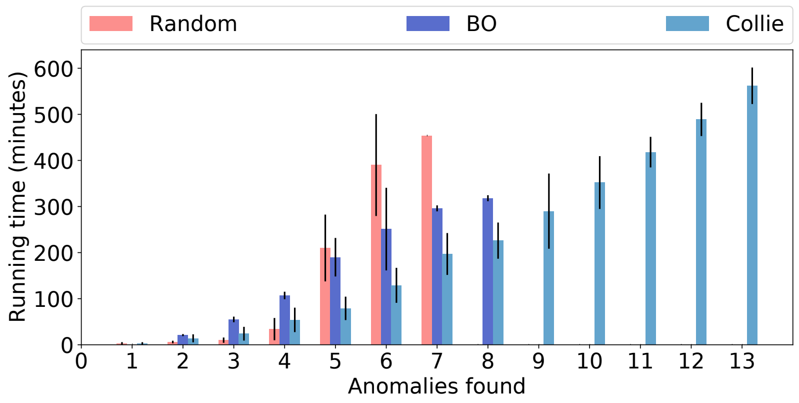


FIGURE 3.4: Mean time to find anomalies with random input generation, BO, and Collie. Error bars denote standard deviations. There is no red bar starting from 8, and no purple bar starting from 9, because random input generation and BO can only find 7 and 8 anomalies, respectively.

BO works well when counter values are smooth in the search space. However, the counter values in our search space can have sudden changes, because some discrete dimensions have a huge impact on the counter values (e.g., QP type). Collie uses a simulated annealing based algorithm to optimize the counter values and successfully speed up the search process. Given limited time, it can find all the performance anomalies of this RDMA subsystem. We believe this improvement comes from the optimization process: driving counters to extreme regions is more likely to trigger performance anomalies. It is possible that a more efficient search algorithm (e.g., a fine-tuned BO, reinforcement learning) can perform better, and it is worth future exploration. However, our goal here is to demonstrate that existing simple optimization algorithms, such as simulated annealing, can search efficiently with these hardware counters.

Collie uses diagnostic counters and MFS to further speed up the search. Now we break down their contribution to our overall search speed. Figure 3.5 shows the result.

The value of diagnostic counters. Figure 3.5 shows that with performance counters, Collie(Perf) has already found 11 of the 13 anomalies, including the 3 existing ones. This proves that the performance counters are informative and can be used to improve search efficiency. It shows the generality of Collie because performance counters are general and provided by all commodity RDMA subsystems. Figure 3.5 also shows that using diagnostic counters can further improve the speed. Given limited time, Collie(Diag) can uncover more anomalies and is faster. For example, Anomalies #7 and #8 are not captured by Collie(Perf) because there is no performance change during the search, but Collie(Diag) can observe the increase of RNIC internal cache miss and uncover them.

The value of minimal feature set (MFS) The main difference between SA and Collie is whether MFS is applied. With MFS, the efficiency of all approaches (both using diagnostic counters and using performance counters) is significantly improved. For example, Collie(Diag) only uses about half of the time to uncover all the anomalies found by SA(Diag). MFS improves efficiency by eliminating redundant tests from the search space. Otherwise, approaches without MFS may be stuck in the area of an uncovered anomaly.

To understand why increasing diagnostic counter values can help to find anomalies and how MFS works, here we use Receive WQE Cache Miss counter as an example. We do not rely on the meaning of these diagnostic counters during the search. To the best of our knowledge, the counter means the number of times that RNICs need to issue extra DMA operations to fetch receive WQE from host DRAM.

Figure 3.6 shows the diagnostic counter values during the search. The random input generation approach (the orange line) does not increase the diagnostic counter value and thus cannot find many performance anomalies. Collie w/o MFS (the green line) can drive the diagnostic counter value very high, but it cannot find many distinct performance anomalies because further increasing the counter value in the neighboring regions of existing performance anomalies wastes time. Collie (the blue line) is effective in finding performance anomalies, because it can both increase the diagnostic counter value to find application workloads that cause anomalies and also do not need to test application workloads that belong to the same anomaly. Figure 3.6 shows that most anomalies are found when the diagnostic counter value is high. This also supports the intuition that it is likely to trigger performance anomalies when the diagnostic counter value is driven to extreme regions, which indicates the RDMA subsystem is under pressure. Some anomalies in Figure 3.6 do

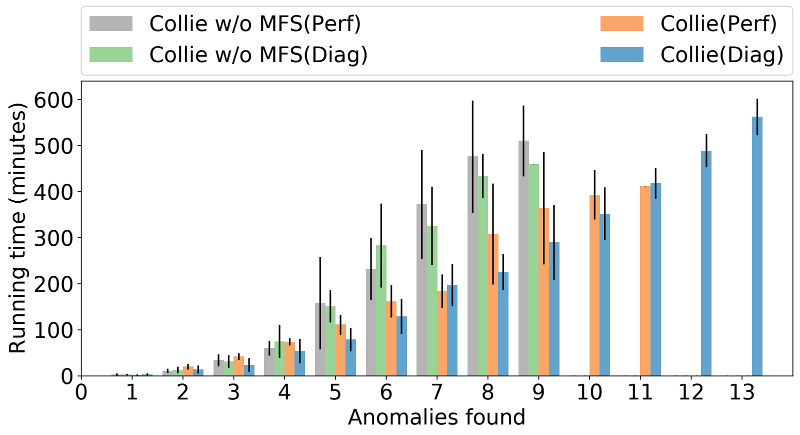


FIGURE 3.5: Mean time to find anomalies. (Diag) means diagnostic counters, and (Perf) means performance counters. Error bars denote standard deviations.

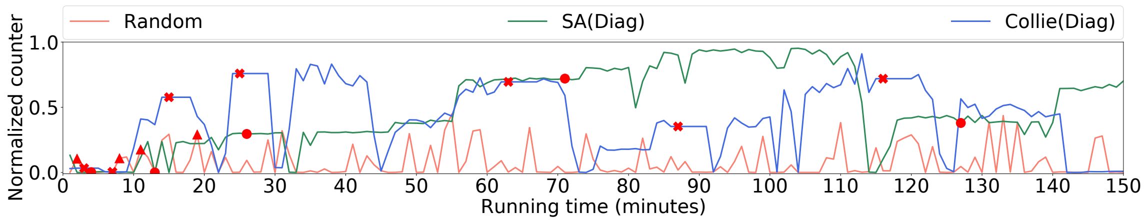


FIGURE 3.6: Diagnostic counter values (Receive WQE Cache Miss) during the search. Counter values are normalized based on the maximum value we observed in the search. Red crossings denote the performance anomalies found by Collie. Red triangles denote the performance anomalies found by random input generation. Red squares denote the performance anomalies found by Collie without MFS. Collie (the blue line) is flat for a few minutes after finding a new performance anomaly. This is to represent the time needed for extracting the MFS.

not show a high value of this counter. This is mainly due to that they are anomalies that can be easily triggered. They are usually triggered at the beginning of the search process (left corner of Figure 3.6) and another corresponding diagnostic counter value is high. For example, Anomaly #13 has simple triggering conditions and is usually found very soon. It does not increase the Receive WQE Cache Miss counter but will increase another counter, the counter of PCIe Internal Back Pressure.

3.7.3 Using Collie for Application Design

We use Collie in the development and performance debugging of two key RDMA applications.

First, Collie provides design suggestions for our self-developed efficient RDMA RPC library during its design and implementation. The library needs to be CPU-efficient, and we thus only consider RC as the transport because it is the only transport that supports all one-sided RDMA operations (i.e., READ, WRITE) and ensures reliable messages. In addition, major services that use this RPC library will mainly be deployed on subsystem B and C. Given the search space, Collie provides two suggestions to the developers. (1) Anomaly #4 is in the restricted search space if the RDMA RPC library uses READ, large WQE batch size, and a long SG list to improve throughput and shape the message format. (2) The library needs to use SEND/RECV to deliver small control messages and generally keeps a large receive queue in case of receive-not-ready error. This can potentially trigger Anomaly #5. Unfortunately, both #4 and #5 temporarily have no fix, so Collie suggest developers (1) use RDMA WRITE to transmit data in a batch and (2) configure receive queue depth carefully in SEND/RECV for small control messages transmission. This RDMA based RPC library achieves expected performance and is currently supporting three major services in production.

Second, Collie helps a distributed machine learning (DML) application based on BytePS [54] bypass anomalies during its further development in our production environment. Our DML application encountered anomaly #9 when deploying on our new subsystem E. We worked with multiple vendors (RNIC, server, CPU), but for several weeks we didn't find the root cause or the fix for this anomaly. During this time, we ran Collie and compared the anomalous application with the MFS we got. We found that the application's behaviors matched one of the MFS: (1) use a long SG list to send tensors with several meta data and (2) the message pattern of tensors and meta data is a typical pattern that contains mix of short and long messages. Collie suggested the developers to avoid these conditions. The developers hence bypassed this anomaly before vendors' fix is ready.

3.7.4 Implications of the Performance Anomalies Found

After careful analysis of the anomalies found by Collie, we have several interesting and important observations.

Holistic performance testing/tuning over entire RDMA subsystems is important. With our vendors' help, we try our best effort to present the root causes of these anomalies in §3.11. The root causes can be bottlenecks from RNIC internals, PCIe controllers, and host topologies (cross socket communication). This is because the RDMA network performance is highly related to the entire subsystem and the holistic test is thus important. Besides, we need to configure systems carefully (MTU, PCIe, NUMA, IOMMU, etc.) to fully leverage RDMA's performance [94, 58]. Collie shows that it is sometimes difficult to choose what configuration to use. For example, comparing the Anomaly #14 with other cases related to the MTU setting (e.g., #6), we observe there is no optimal MTU setting for all types of RDMA subsystems. This also indicates that data center operators have to test various RDMA subsystem configurations and tune the system carefully before deploying them.

Opaque resource limitation of the RDMA subsystems. RDMA virtualization, especially performance isolation is important for deploying RDMA to the public cloud environment. Researchers have spent a lot of effort and proposed several solutions [127, 60, 142, 42, 110]. However, anomalies found by Collie suggest that there are new challenges. Existing approaches mainly focus on the isolation of visible resources like verbs structures (e.g., QP, MR, CQ), pinned memory, and bandwidth. However, there exist resources that are opaque for developers and data center operators. For example, the RNIC has limited caches that store many data structures, including connection context (well known as QPC) and receive WQE. Anomalies #1, #3, #4, #5 show that severe WQE cache miss can have a huge impact on performance. Hence, it is possible that a connection with a specific message pattern affects another connection by triggering cache misses, even when the bandwidth and other resources are well isolated. We therefore believe it is necessary to take these invisible resources into consideration when enforcing RDMA performance isolation, especially in public clouds.

Does Ethernet-based RDMA need end-to-end flow control? Currently there is no end-to-end flow control mechanism (e.g., the sliding window for TCP) for production Ethernet-based RDMA deployment (i.e., RoCEv2). Collie shows that this is a major barrier for RDMA subsystems to achieve high-performance and reliability. For example, many anomalies (e.g., #9 and #12) show that the host limitation can slow down RNIC's outbound rate (dispatching received data to host memory). This makes the receiver cannot consume packets as fast as the sender sends. Without end-to-end flow control, the RoCEv2 now can only rely on PFC, the hop-by-hop flow control mechanism. PFC helps to avoid such overflow packet drop but can cause catastrophic consequences [39, 49]. Note that RDMA congestion control [144, 90, 71] mainly targets in-network congestion, so it is orthogonal. A similar observation has been shown in IRN [91], but they mainly focus on in-network behaviors. Collie shows that, in addition to switches, the hosts can also generate PFC pause frames, which requires attention when deploying RDMA in production.

3.8 Discussion and Future Work

Search space. Collie mainly focuses on how specific application workloads can stress the RDMA subsystems and trigger performance anomalies. We therefore focus on a simple setting of two RNICs and assume the network is free of anomaly. In addition, we temporarily ignore control path behaviors and the inter-arrival time between requests of a connection. The main reason is that adding these factors substantially enlarge the size of our search space. How to efficiently expand Collie's search space is an interesting direction for exploration.

Search algorithm. Collie uses simulated annealing based algorithm with minimal feature set (MFS) to search efficiently. Though powerful data centers can run Collie on multiple machines for a longer time, the search algorithm is also important. According to the MFS found by Collie, the expected time for a random approach is tens of days to find some anomalies that require complicated triggering conditions. There are many other search algorithms alternatives that can be leveraged, such as reinforcement learning. Integrating more search algorithms into Collie is another interesting direction to explore.

Generality of Collie. We believe that Collie can be used for any type of RDMA subsystem or even subsystems with other types of NICs. For example, though the link/transport protocols are different for Infiniband and RoCEv2, the NIC internal structures should be similar (e.g., both can use Mellanox CX-6 VPI RNIC). Collie only relies on non-proprietary counters that expose NIC internal status. Therefore, this methodology should be generalizable to any NIC in any deployment environment if similar counters are available.

Analysis of Performance Anomalies. Collie is designed to uncover anomalies and help to bypass them from the perspective of data center operators, so it assumes minimal hardware knowledge of RDMA subsystems for generality and does not directly analyze the underlying causes. However, since the anomalies found by Collie can be severe (e.g., triggering PFC pause storms), we believe to fully understand them is also an important direction to explore. For example, as mentioned in §3.7.4, many anomalies are due to bottlenecks on some opaque resources. Both RNIC vendors and data center operators hence need to understand what extra resources should be considered if they want to provide performance isolation for RDMA in a public cloud.

3.9 Related Work

Hardware bottlenecks in host networking. With the fast growth in NIC performance, researchers have noticed several potential hardware bottlenecks in host networking. Neugebauer et al. [94] study the implication of PCIe performance in host networking. Farshin et al. [26] examine when and when not Intel Data Direct I/O technology can speed up host networking by allowing NIC to access CPU's last-level cache directly. Kalia et al. [56] observe the scalability bottlenecks of caching per-connection metadata in RNIC. Stanko et al. [96] study how the number of connections and memory regions affect performance. These works have raised our attention to RNIC hardware behaviors. Our work is on a different angle: we systematically uncover the performance anomalies that can be triggered by specific application workload due to hardware bottlenecks.

Fuzz testing. Our techniques are in the broader category of fuzz testing. There are three types of fuzz testing: black-box [87, 86], white-box [35, 36, 30], and gray-box fuzzing [15, 119]. Black-box fuzzing is to generate random inputs to test a program, and usually black-box fuzzing can only uncover shallow bugs. In our context, this is also true that using randomly generated application workload can only uncover a small set of anomalies (§3.7). White-box fuzzing is to use symbolic execution on source code to guide the fuzzer to generate inputs that can have high coverage. We do not have the internal designs of the various components within an RDMA subsystem, so we cannot use white-box approaches. Gray-box fuzzing in the software context is to use the coverage in the control flow graph to guide the fuzzer to incrementally generate inputs that can lead to larger coverage. Our approach is similar to gray-box fuzzing that we both use simulated annealing and mutation-based test case generation. However, the key difference is that we use hardware counters in the RDMA subsystem to guide the search rather than the coverage on the control flow graphs of the source code.

Application design on top of RDMA. Many RDMA application designs leverage specific RDMA performance characteristics, and some already try to circumvent certain RNIC performance anomalies. HERD [57] uses UD SEND and UC Write to implement an RPC library for reduced RNIC packet processing overheads and better scalability. FaSST [59] and eRPC [56] uses UD to further mitigate RNIC scalability bottlenecks in RPC libraries. Kalia et al. [58] provide guidelines to optimize HERD's transport by considering PCIe bottlenecks. FaRM [23, 24] uses RC to access remote in-memory key-value stores, so that it can use RDMA 1-sided READ/WRITE operation for reduced CPU overheads. Our goal is complementary: we systematically uncover the set of performance anomalies of RDMA subsystems that application developers need to be aware of. We show that for RDMA developers, in reality, there is no optimal choice for a particular design decision (e.g., all transport types have certain performance anomalies). Developers therefore need to have a holistic view of all the design decisions and the entire RDMA subsystem before designing and implementing RDMA applications.

3.10 Summary

RDMA has been increasingly used in the industry for its low latency and reduced CPU overheads. Performance anomalies hurt application performance and can lead to catastrophic consequences (e.g., deadlocking the data center network). We build Collie, a tool to help RDMA users to find performance anomalies of the entire RDMA subsystems, without the need for access to any hardware internals design. Collie constructs a comprehensive search space for RDMA application workloads and finds performance anomalies by using simulated annealing to optimize two types of vendor-provided counters. We evaluate Collie on 8 commodity RDMA subsystems and Collie found 15 new performance anomalies that are all acknowledged by the vendor. 7 of them are already fixed under vendors' guidance. We also present our experience in using Collie to guide our development of an RDMA RPC library and help our distributed machine learning applications bypass performance anomalies before vendor fix is ready. Collie is available at `https://github.com/host-bench/collie`.

3.11 Appendix: Performance Anomalies

More details of these anomalies and the lesson we learn are included in this section. We present a concrete example of each anomaly and try our best to simplify each anomaly so that they can be reproduced easier. It is possible to find milder or stricter conditions that trigger the anomaly. We, to the best of our knowledge, also categorize these performance anomalies to their root causes based on our observation and conversations with our vendors.

3.11.1 Subsystem F with Mellanox 200 Gbps CX-6 VPI

Root cause #1: Receive WQE cache misses bottleneck RNIC receiving rate.

(New) Anomaly #1: UD with large WQE batch size and long WQ causes PFC pause frames and drastic throughput drop.

Collie observes that the pause duration ratio can be up to $\approx 20.0\%$ with only a single UD QP. The pause duration ratio means that RNIC is asking the corresponding switch port to pause for ≈ 200 milliseconds within one second on average. We share the NIC vendor with our traffic engine tool and the running command. They have reproduced the anomaly in their environments, but the root cause is still not clear yet. Therefore, we claim this anomaly not fixed yet. To the best of our knowledge, it is likely due to the cache miss triggered by the pre-fetch mechanism for the receive WQE. This bottlenecks the receiver from receiving traffic.

Here is a simplified concrete trigger setting of Anomaly #1: There is 1 connection of UD QP using SEND/RECV Opcode. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 256 (i.e., max_send/recv_wr = 256). The MTU is 2KB. The sender keeps sending 64 requests in a batch. Each request only has one SG element and a fixed size of 2KB.

(New) Anomaly #2: UD with small WQE batch size, long WQ, small messages, and a few connections causes throughput to drop without pause frames.

This anomaly is similar to #1 but more tricky and has a different end-to-end symptom.

Unlike #1, Collies does not observe PFC pause frames when the setting is slightly different from #1: if the sender does not post sending requests in batch or the batch size is small (e.g., less than 8) and the messages are relatively small (e.g., 512B, 1KB), the throughput will drop by more than 20% without any PFC pause frame triggered when the receiver has an extremely long work queue. If we set a smaller work queue for the receiver, the throughput returns to the line rate. This anomaly is also reproduced and acknowledged by NIC vendor. We conjecture that it has a similar root cause to #1, but due to unknown RNIC bottlenecks, it behaves differently that the throughput drops without pause frame.

Here is a simplified concrete trigger setting of Anomaly #2: There are 16 connections of UD QP using SEND/RECV Opcode. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 1024. The MTU is 1KB. The sender keeps sending 4 requests in a batch. Each request only has one SG element of 1KB.

(New) Anomaly #3: RC READ with large messages causes PFC pause frames when MTU is under 1500 (the default MTU for Ethernet).

We observe the throughput drops drastically once we use RDMA READ opcode with 1500 MTU (1024 for RDMA), the default value for our data centers. The pause duration can be up to 10% and throughput drops to less than half. We report this to our NIC vendor and they tell us the low MTU may trigger the RNIC internal packet processing bottleneck for this 200 Gbps NIC. We carefully survey the potential effect of MTU modification in our deployment and modify the MTU from 1500 to 4200, which supports 4096 as RDMA MTU. This anomaly is successfully fixed in this way.

Here is a simplified concrete trigger setting of Anomaly #3: There are 8 connections of RC QP using Read opcode. Each QP has 1 sending MR of 4MB and 1 receiving MR of 4MB. Each QP has a work queue of length 128. The MTU is 1KB. The sender keeps sending RDMA READ requests. Each request only has one SG element and a fixed size of 4MB.

(New) Anomaly #4: Bidirectional RC READ with large WQE batch size, long SG list, and a few connections causes PFC pause frames, even when MTU is set to 4200 (4096 for

RDMA).

This anomaly is tricky but severe. Even with 4200 MTU (Anomaly #3 is solved), Collie observes about 30% PFC pause duration ratio that when bidirectional RDMA READ happens and both sides post a large number of requests in a batch (e.g., 32), each request consists of multiple scatter gather element (e.g., 4) and there are a few connections (e.g., $\approx$ 160). As usual, this newly found anomaly is reported to the vendor and they have reproduced and confirmed the anomaly. For now, the root cause of this anomaly is still unknown. Therefore, we claim this anomaly not fixed yet.

Here is a simplified concrete trigger setting of Anomaly #4: There are 80 connections of RC QP using Read opcode for each direction. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 128. The MTU is 4KB. The sender keeps sending 128 requests in a batch. Each request has 4 SG elements and a fixed size of 128B.

(New) Anomaly #5: RC SEND with small MTU, large WQE batch, long WQ, and long messages causes PFC pause frames and drastic throughput drop.

(New) Anomaly #6: RC SEND with small MTU, small WQE batch, large SG list batch, long WQ, small messages, and a few connections causes reduced throughput without any pause frame.

They are similar to UD ones (Anomaly #1 and #2) but have a more complex and stricter trigger. For example, Collie observes such anomaly only when MTU is small (e.g., 1024 for RDMA), work depth exceeds 1K for each QP as well as post multiple receive WQE in a batch. These anomalies are different because they have different QP types and stricter trigger conditions. For example, those anomalous application workloads in #1 and #2 won't trigger anomalies if we only switch the type of QP from UD to RC. Several discussion with our vendors tells us that the Reliable Connection type contains some subtle variance inside the RNIC that result in such difference. These two are currently not fixed yet.

Here is a simplified concrete trigger setting of Anomaly #5: There is 1 connection of

RC QP using SEND/RECV opcode. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 1024. The MTU is 1KB. The sender keeps sending 64 requests in a batch. Each request has 2 SG elements and a fixed size of 2KB.

Here is a simplified concrete trigger setting of Anomaly #6: There are 32 connections of RC QP using SEND/RECV opcode. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 1024. The MTU is 1KB. The sender keeps sending 8 requests in a batch. Each request has 2 SG elements and a fixed size of 1KB.

Root cause #2: Interconnect Context Memory cache misses reduce RNIC sending rates.

(New) Anomaly #7: RC WRITE with many QPs, small messages, small WQ depth, and small WQE batch size causes reduced throughput.

(New) Anomaly #8: RC WRITE with many MRs, small messages, and small WQE batch size causes reduced throughput.

Though these two anomalies are well-known as the RDMA scalability problem, our real applications do not meet them even when the number of QPs exceeds 10K and the number of MRs exceeds 100K. However, Collie uncovers these two so we classified them into New anomalies. We take a deep look into how Collie discovers them and have many discussions with our vendors. We find our experience interesting and worthy of sharing: RNIC caches many necessary structures on its cache (e.g., memory translation table and connection context). When a request triggers cache miss, the RNIC has to issue extra PCIe operation to fetch them from the host DRAM. This will certainly induce extra PCIe latency for processing this request (victim request). However, RNIC is highly pipelined, so even when the victim request has finished the PCIe operation, it may still have to wait for the other pipeline stages to get ready (e.g., a previous long egress request blocks this short egress request). Therefore, if the request size is relatively large enough, the cache miss will not have a large effect on end-to-end performance because the overhead is hidden due to the pipeline.

Here is a simplified concrete trigger setting of Anomaly #7: There are 480 connections of RC QP using RDMA WRITE opcode. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 16. The MTU is 1KB. The sender keeps sending requests without WQE batch. Each request has 1 SG element and a fixed size of 512B.

Here is a simplified concrete trigger setting of Anomaly #8: There are 24 connections of RC QP using RDMA WRITE opcode. Each QP has 1024 sending MR of 64KB and 1024 receiving MR of 64KB. Each QP has a work queue of length 128. The MTU is 1KB. The sender keeps sending requests without WQE batching. Each request has 1 SG element and a fixed size of 512B.

Root cause #3: PCIe controller blocks RNIC from reading host memory.

(Old) Anomaly #9: Bidirectional traffic with a mixture of small and large messages in an SG list on particular AMD servers causes PFC pause frames and drastic throughput drop.

This anomaly is found by one of our production applications that keeps sending such message patterns (described in 3.2). The root cause of this anomaly is due to PCIe ordering issue. If the RNIC on the AMD server is not configured as PCIe relaxed ordering device, a DMA request may be blocked by the previous one. Therefore, when bidirectional traffic with a mix of short and long requests. The ingress short requests, together with the completion of egress traffic, blocks the ingress long requests. This results in RNIC buffer accumulation and triggers a large amount of PFC pause frames. The throughput can only achieve 60 Gbps with 25% pause frame duration ratio on average. With much effort from our appreciative vendors, we finally fix this by configuring RNIC as a forced relaxed ordering PCIe device.

Here is a simplified concrete trigger setting of Anomaly #9: There are 8 connections of RC QP using RDMA WRITE opcode for each direction. Each QP has 1 sending MR of 4MB and 1 receiving MR of 4MB. Each QP has a work queue of length 128. The MTU is 4KB. The sender keeps sending 8 requests in a batch. Each request has 3 SG elements and the pattern is [128B, 64KB, 1KB].

Root cause #4: RNIC packet processing bottleneck.

(New) Anomaly #10: Bidirectional RC Write with large WQE batch size, a mixture of long messages and lots of short messages, and a few connections causes PFC pause frames.

Collie finds that when several RC QPs keep posting multiple short requests (e.g., 64B, 128B) in batch and a few long requests for both directions, a large amount of pause duration is triggered. This RNIC of the RDMA subsystem has already been configured as forced relaxed ordering PCIe device (Anomaly #8 is solved). Our vendors have confirmed this anomaly and announce it fixed in their upcoming firmware release. The lengthy discussion with our vendor shows us the rough root cause: some component for packet processing inside the RNIC is not fully bidirectional, and our bidirectional reliable traffic (requires packet-level ACK) pattern with a huge amount of short requests, trigger that component's bottleneck. This results in long requests blocked and then many PFC pause frames are generated.

Here is a simplified concrete trigger setting of Anomaly #10: There are 320 connections of RC QP using RDMA WRITE opcode for each direction. Each QP has 1 sending MR of 64KB and 1 receiving MR of 64KB. Each QP has a work queue of length 128. The MTU is 1KB. The sender keeps sending 64 requests in a batch. Each request has 1 SG element and the pattern is [64KB, 128B, 128B, 128B].

Root cause #5: Host topology causes PCIe latency to increase, and this bottlenecks RNIC receiving rate.

(New) Anomaly #11: On specific types of AMD servers, Bidirectional cross-socket traffic causes pause frame storm and drastic throughput drop.

Collie outputs the minimal feature set with only source/destination NUMA set and bidirectional traffic, indicating these two are the dominant factors. With this bidirectional (A to B and B to A) cross-socket NUMA setting (e.g., NUMA 0 from socket 0 for A and NUMA 2 from socket 1 for B, where socket 0 is the affinitive node for RNIC), even mild traffic with only a single connection can trigger up to 15.7% pause frame duration ratio.

After several conversations with our RNIC and server vendors, we conjecture the root cause lies in these particular servers' cross-socket performance because we run the same traffic with the same NIC on different servers but do not observe the same phenomenon. We consider this anomaly as fixed because the vendor helps us roughly understand the root cause and suggest we use 2x100 Gbps NIC (each for a socket) to reduce cross-socket traffic, and we follow this guidance.

Here is a simplified concrete trigger setting of Anomaly #11: There is 1 connection of RC QP using RDMA WRITE opcode for each direction. Each QP has 32 sending MR of 4MB and 32 receiving MR of 4MB. Each QP has a work queue of length 128. The MTU is 4KB. The sender keeps sending 16 requests in a batch. Each request has 1 SG element with a fixed size of 256KB. The QP on host A is using the memory of socket 0 and the QP on host B is using the memory of socket 1.

(Old) Anomaly #12: GPU-direct RDMA causes pause frame storm and drastic throughput drop on particular AMD servers.

We observe a huge amount of pause frames and drastic throughput drop only on some servers in our clusters. The pause duration ratio can be up to 15% and throughput can drop to less than 20% (i.e., 40 Gbps) in this scenario. After careful debugging with our NIC vendor's strong support, we find out that there is a slight difference in PCIe bridge configuration (PCIe ACSCtl) between the anomalous server and normal ones. The anomalous configuration will forward GPU traffic to the root complex rather than directly to the RNIC. We fix this anomaly by adopting the correct configuration.

Here is a simplified concrete trigger setting of Anomaly #12: There are 8 connections of RC QP using RDMA WRITE opcode for each direction. Each QP has 1 sending MR of 4MB and 1 receiving MR of 4MB. Each QP has a work queue of length 128. The MTU is 4KB. The sender keeps sending 8 requests in a batch. Each request has 3 SG elements and the pattern is [128B, 64KB, 1KB]. All MRs are allocated from GPU memory and we use the GPU under the same PCIe bridge (i.e., shown as PIX/PXB in nvidia-smi result).

Root cause #6: RDMA NIC has potential in-NIC incast/congestion.

(Old) Anomaly #13: Co-existence of receiving traffic and loopback traffic causes PFC pause frames.

This anomaly is found in our real applications and can also be uncovered by Collie. Our machine learning system runs workers and servers, and they use RDMA to accelerate the communication. However, once a worker and a server are scheduled on the same physical machine, there will be loopback traffic: the worker will send RDMA traffic to the server on the same host. Meanwhile, the server is receiving traffic from workers on other physical machines. The combination of receiving and loopback traffic triggers congestion/incast inside the NIC. And this RNIC lacks a mechanism to limit the loopback traffic rate, which makes the problem worse. After several discussions with our vendor, we bypass this anomaly by identifying the loopback communication and using other IPC mechanisms (e.g., shared memory). We do not consider this anomaly fixed because we cannot fully rely on other IPC mechanisms, especially for the virtualization environment. This anomaly exposes that a proper design of RNIC needs to consider NIC incast and we are glad to see that some latest RNIC have done so.

Here is a simplified concrete trigger setting of Anomaly #13: There are 16 connections of RC QP using RDMA WRITE opcode. 16 receivers are 8 senders are on the same host A and the other 8 senders are on the host B. Each QP has 32 sending MR of 4MB and 32 receiving MR of 4MB. Each QP has a work queue of length 128. The MTU is 4KB. The sender keeps sending 16 requests in a batch. Each request has 1 SG element with a fixed size of 256KB.

3.11.2 Subsystem H with Broadcom 100 Gbps P2100G

(New) Anomaly #14: Bidirectional RC traffic with lots of connections and the large MTU causes reduced throughput without PFC pause frame.

Collie observes that a large MTU is necessary to trigger this anomaly. Once we switch the MTU from 4096 (for RDMA) to 1024, both directions can achieve the line rate. This is unusual because most cases show that large MTU improves the performance and small

MTU triggers performance anomalies. We don't observe the same phenomenon on any other type of RNICs.

Here is a simplified concrete trigger setting of Anomaly #14: There are 1024 connections of RC QP using RDMA WRITE opcode for each direction. Each QP has 81 sending MR of 256KB and 83 receiving MR of 256KB. Each QP has a work queue of length 128. The MTU is 4KB. The sender keeps sending 1 request in a batch. Each request has 4 SG element with a fixed size of 64KB.

(New) Anomaly #15: UD with long WQ and lots of connections causes PFC pause frames.

This anomaly is similar to the Mellanox anomaly #1 but has a slightly different trigger. Collie successfully trigger #1 with only a single connection, but for P2100 RNIC our multiple runs show that a few connections are necessary.

Here is a simplified concrete trigger setting of Anomaly #15: There are 32 connections of UD QP using SEND/RECV opcode. Each QP has 1 sending MR of 4KB and 1 receiving MR of 4KB. Each QP has a work queue of length 64. The MTU is 2KB. The sender keeps sending 1 request in a batch. Each request has 1 SG element. The message pattern is like [256B, 1KB, 64B, 1KB].

(New) Anomaly #16: RC READ with lots of connections, large WQE batch size, and small MTU causes PFC pause frames.

This anomaly is similar to the Mellanox anomaly #4 and it shows that for the same RNIC and other hardware components, the best MTU choice can be different when workloads change.

Here is a simplified concrete trigger setting of Anomaly #16: There are 500 connections of RC QP using RDMA READ opcode. Each QP has 1 sending MR of 256KB and 1 receiving MR of 256KB. Each QP has a work queue of length 128. The MTU is 1KB. The sender keeps sending 8 requests in a batch. Each request has 1 SG element with a fixed size of 64KB.

(New) Anomaly #17: RC SEND with lots of connections, small WQE batch size, small

MTU, short messages, and long WQ causes PFC pause frames.

We have reported this anomaly to our vendor. To the best of our knowledge, we conjecture this anomaly is related to some corresponding WQE cache component inside RNIC.

Here is a simplified concrete trigger setting of Anomaly #17: There are 80 connections of RC QP using SEND/RECV opcode. Each QP has 1 sending MR of 1MB and 1 receiving MR of 1MB. Each QP has a work queue of length 128. The MTU is 1KB. The sender keeps sending 1 request per batch. Each request has 1 SG element of fixed size 1KB.

(New) Anomaly #18: Bidirectional RC WRITE with a few connections, large WQE batch, and small messages causes PFC pause frames.

Our vendor has confirmed anomalies #17 and #18. They have reproduced these two anomalies and help us fix them. The solution is to configure some specific registers of the RNIC, and these two anomalies disappear.

Here is a simplified concrete trigger setting of Anomaly #18: There are 16 connections of RC QP using RDMA WRITE for each direction. Each QP has 1 sending MR of 12KB and 1 receiving MR of 12KB. Each QP has a work queue of length 64. The MTU is 1KB. The sender keeps sending 16 requests in a batch. Each request has 1 SG element of fixed size 64KB.

4. Understanding RDMA Microarchitecture

In this chapter, we introduce Husky, the second core component of this dissertation. Building on our initial exploration of RDMA microarchitecture, we present our efforts towards comprehensively understanding RDMA microarchitecture. We develop a systematic test suite to reveal RDMA resource consumption model and use this model to construct a set of workloads to identify RDMA performance interference. Husky reinforces the thesis that a deep, microarchitecture-aware approach is critical for identifying performance interference and anomalies in RDMA networks. Husky has identified numerous new types of interference that are previously unknown, even for the RNIC manufacturers, highlighting its effectiveness. Husky also show that none of existing solutions provide robust isolation against such interference because they all neglect RDMA's microarchitecture resource contention. This work was completed in collaboration with Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R. Lebeck, and Danyang Zhuo.

4.1 Introduction

Multiplexing workloads from different tenants on a shared computing infrastructure enables the modern cloud computing era. The global cloud infrastructure revenue has already surpassed 400 billion US dollars and is forecast to grow to reach around 1 trillion US dollars in the next decade [21].

It is well known that having different tenants' workloads share computing resources can lead to unpredictable application performance interference [128, 40, 27] and privacy leakage [72, 61]. This drives plenty of studies focusing on performance isolation in the cloud, especially for performance-critical applications that have stringent service-level objectives [128, 122, 140, 40, 27, 78, 25]. The state of the art in practice has also significantly advanced: CPU vendors even implement hardware mechanisms to control and isolate access to CPU caches [43]. Side channels through shared resources are being patched over time [72].

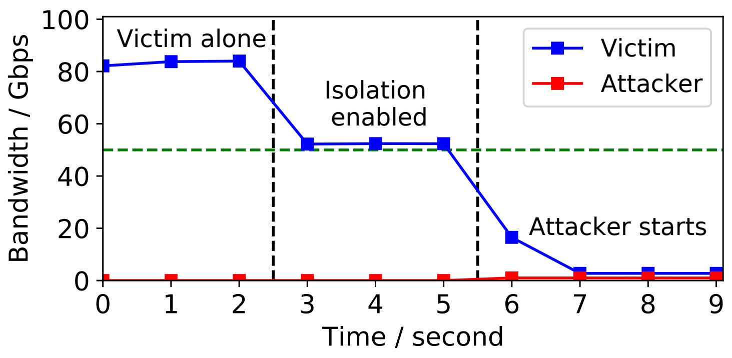


FIGURE 4.1: Violations of performance isolation under existing methods.

In this paper, we visit one particular hardware device, the RDMA NIC (RNIC). RDMA offloads the network stack from OS kernel to NIC hardware to provide high throughput and ultra-low processing latency with near-zero CPU overhead. RDMA has been deployed in datacenters at scale to improve performance and free up CPU cores for first-party workloads like storage and ML [39, 71, 32, 93]. Now cloud providers are working towards supporting RDMA in general-purpose guest VMs to benefit third-party workloads. To this end, cloud providers must provide strong performance isolation for tenants sharing the same RNIC.

Many efforts have been made to improve network performance isolation in the public cloud, with a special focus on bandwidth and packet processing capacity [13, 37, 38, 53, 65, 121, 123]. However, RDMA brings new challenges due to its unique and complex NIC microarchitecture resources (e.g., NIC caches and processing units). Their existence and impact on performance are already known to the research community [64, 58]. To avoid performance anomalies, developers carefully design RDMA systems to avoid exhausting these microarchitecture resources [56, 59, 23, 24, 116, 19, 92]. Our study is from a different angle: we look at how these microarchitecture resources affect RDMA performance isolation from a public cloud provider's perspective. The cloud provider has no knowledge and control of tenants' RDMA applications, and tenants can consume RNIC microarchitecture resources in arbitrary manners.

To demonstrate RNIC microarchitecture resources' significant impact on performance isolation, we test the state-of-the-art approach: using SR-IOV with separated hardware traffic class (HW TC). Both SR-IOV and HW TC are hardware mechanisms available on commodity RNICs. HW TC leverages multiple hardware queues (usually 8 queues) in RNICs. We can assign each tenant application to use one queue. We run one victim traffic between two virtual machines using `ib_write_bw`, a standard RDMA bandwidth testing tool in Perftest [108]. Each virtual machine is on a different server, and the two servers are equipped with 100 Gbps NVIDIA ConnectX-5 RNICs. Figure 4.1 shows the bandwidth. The bandwidth test achieves 80 Gbps. We start one virtual machine on each server to represent an attacker (i.e., a buggy or malicious tenant application) and enable performance isolation to grant half of the total bandwidth to the victim and the attacker. The victim traffic reduces to 50 Gbps, which is expected. However, when we start a carefully designed attacker traffic of only 1 Gbps to intentionally exhaust one of the RNIC microarchitecture resources, the victim immediately drops to 2 Gbps, violating the performance isolation guarantee (i.e., 50 Gbps of guaranteed network bandwidth for the victim).

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

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

This paper makes the following contributions:

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

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

4.2 Background and Motivation

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

4.2.1 Network Performance Isolation in the Public Cloud

Tenants in the cloud mainly cause contention on two types of network resources. The first the most obvious one is the bandwidth in the network fabric. To mitigate bandwidth contention among tenants, one line of work [121, 111, 114] statically limits per-tenant bandwidth. Another line of work [13, 111, 14, 38, 134, 9, 69, 53, 112, 20, 52] gives each tenant a minimum bandwidth guarantee and allows tenants to use spare bandwidth capacity. The second type of resource is the packet processing resources at the end host. Per-packet processing costs depend on many factors, such as cache misses and operations to perform. Recently, PicNIC [65] provides isolation for such software packet processing. People also leverage specialized hardware to achieve the same goal [123].

It is worthwhile to note that network performance isolation is very different from network virtualization. Network virtualization orchestrates network resources to provide each tenant with an illusion of an independent network. A tenant should not impact the connectivity of the network of another tenant. The goal of network virtualization is to achieve low overhead [60, 110, 42]. In comparison, network performance isolation focuses on how to manage resource contentions to ensure that tenants can achieve guaranteed performance.

4.2.2 RDMA Overview

RDMA allows the NIC to directly transfer data between the wire and the application memory. The networking protocol is implemented in the NIC. Figure 4.2 presents the overview of the RDMA workflow. It classifies standard RDMA programming interface, a.k.a., verbs, into two categories: control and data. An application first needs to call several control verbs to allocate necessary objects, such as queue pair (QP) and completion queue (CQ), to set up a reliable connection (RC), an unreliable connection (UC), or an unreliable datagram (UD) transmission endpoint. Then the application needs to register a memory region (MR). This registration essentially pins the memory in the host DRAM and obtains the mapping from virtual addresses to physical addresses, which enables the RNIC

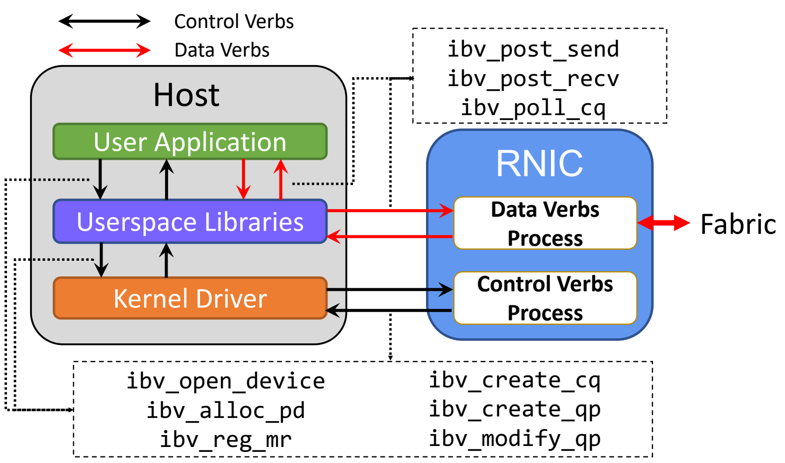


FIGURE 4.2: Overview of RDMA workflow. Verbs processing logics are heavily offloaded to the RNIC.

to directly read from or write to this memory region. All these control verbs are processed by the following procedure: RDMA's userspace libraries and kernel drivers process the verb request, generate a request command, put the command in a negotiated command queue, and ring the RNIC's doorbell (e.g., memory-mapped registers). The RNIC fetches the command from the command queue, processes it, and pushes the response back to the queue. The drivers then process the response and return the object to the application.

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

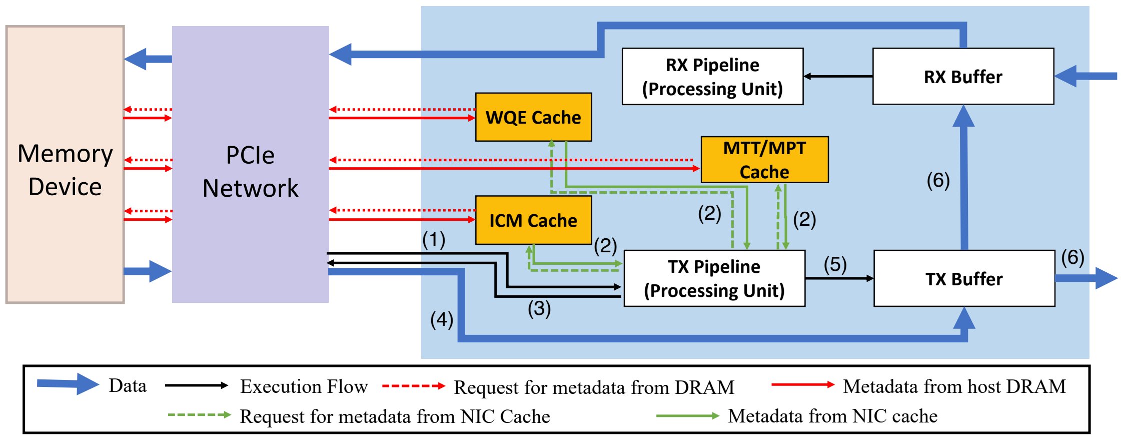


FIGURE 4.3: RDMA NIC microarchitecture hardware details: when the doorbell is rung, the RNIC first fetches the control/data verbs request from the host DRAM. (1) To fetch and process this request, the RNIC may need several metadata (e.g., QP contexts) and there are different types of caches inside the RNIC that can store this metadata. The RNIC can get the metadata directly from these caches, (2) or fetch them from DRAM if a cache miss happens (red lines in the figure). Then the RNIC processes the request and (3) sends the response back to the host DRAM for control verbs or issues DMA requests to read payload for data verbs. After (4) reading data from the host DRAM, the RNIC (5) processes the data into network packets and (6) sends them to the fabric. The symmetric receiver side is not shown for simplicity.

data to/from the remote end or execute ATOMIC operations, the sender should specify correct remote virtual addresses and memory keys. An invalid address or a wrong key will trigger a memory protection error and cause the QP to transition into the error state.

4.2.3 Why RDMA Performance Isolation is Hard?

As shown above, RDMA offloads many host network functionalities to the RNIC, which has many invisible hardware components, and each component may individually become a performance bottleneck. Figure 4.3 shows the hardware components of a commodity RNIC. We draw this figure based on publicly available documents from NVIDIA [83, 99, 104]. In addition to the packet buffers (TX/RX Buffer), the RNIC also has multiple processing units (PU) and many types of internal caches. Each internal cache is used to store a specific type of metadata. For example, in NVIDIA RNICs, the Interconnect

Context Memory (ICM) cache stores QP contexts; the Memory Translation Table (MTT) and Memory Protection Table (MPT) store entries for memory address translation and protection information; and the Work Queue Entry (WQE) cache stores prefetched send WQEs and posted receive WQEs. As these caches are derived from the design needs, other RNICs include similar components. We name these RNIC hardware components *microarchitecture resources* based on the analogy for CPU hardware. CPUs are designed to conform to a standard instruction set architecture (e.g., ARM, x86), but the CPU designers can make the microarchitecture-level decisions, such as how many levels of caches and the cache sizes. RNICs are similar because RNIC vendors have to provide the same programming interface for RDMA application developers, but the vendors can decide on these microarchitecture-level details, e.g., RNIC caches.

Many previous efforts have already identified some impacts of these microarchitecture resources on RDMA application performance. For example, [58, 19, 92] find that an RNIC caches QP contexts. A QP context cache miss can trigger an additional PCIe round trip for the RNIC to fetch the context from the host DRAM, thus degrading application performance. For example, 200 connections can cause an 90% request rate drop on NVIDIA ConnectX-3 NIC [19]. However, these efforts study microarchitecture resources from the perspective of an *application developer*. After a performance degradation, they identify the bottleneck resource, seek more efficient methods to use data verbs, and modify their applications correspondingly.

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

4.3 RNIC Microarchitecture Resources

In this section, we present a study on all the RNIC microarchitecture resources that we are currently aware of. Prior works have already identified several particular forms of resource contention. But our goal here is to systematically study all possible types of resource contention. For each microarchitecture resource, we study how it is consumed by three categories of RDMA operations: (1) control verbs that allocate objects for applications (e.g., `ibv_create_qp`), (2) data verbs that initiate data transfer (e.g., `ibv_post_send`), and (3) exception handling operations that handle exceptions or errors (e.g., RNR errors). Due to space limitations, we first present a few key findings that have significant implications on RNIC performance isolation. After that, we summarize several other findings. We present a detailed analysis of NVIDIA's responses to these findings in §4.11.

4.3.1 Methodology

Our findings center around how to exhaust RNIC microarchitecture resources through the verbs interface [44], the standard RDMA programming API. For each key finding, we demonstrate it with a concrete setting, which consists of a victim workload and an attacker workload. Although we use the terminology attacker, the attacker tenant does not get unauthorized access to other tenants through vulnerabilities. Instead, the attacker is just a normal RDMA application that issues standard RDMA verbs. Each tenant has one client and one server. The clients of the victim and the attacker locate on the same physical machine and share the same RNIC. The servers of the victim and the attacker are colocated on a different physical server. During the measurement, we do not enable any isolation mechanism. We will study existing performance isolation solutions in §4.5.

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

We test four types of 100 Gbps RNICs: NVIDIA ConnectX-5 EN and ConnectX-6 Dx, Chelsio T62100-LP-CR, and Intel E810. NVIDIA NICs runs RoCE, and the Chelsio NIC runs iWARP. Intel E810 supports both RoCE and iWARP, but we currently only test its RoCE implementation. RoCE and iWARP are two standard ways to run RDMA over Ethernet-based networks. Our testbed consists of two servers, each equipped with an RNIC, and the two RNICs are connected via a 100 Gbps switch. For NVIDIA RNICs, we have access to their hardware counters, e.g., cache miss counters, through their network adapter management tool `NEO-Host` [104]. These hardware counters allow us to pinpoint which resource is oversubscribed. For example, when the ICM cache miss counter increases quickly with a certain application workload, we learn that this workload heavily uses this cache, making it oversubscribed. Since other RNICs do not expose such counters, we experiment other RNICs based on their end-to-end performance metrics (e.g., bandwidth).

4.3.2 NIC Caches

We are aware that an RNIC has at least three types of caches, as shown in Figure 4.3. The RNIC stores several types of metadata in these caches to accelerate the request processing, such as the QP contexts in the ICM cache. Prior works have identified some RNIC cache contention problems caused by data verbs with particular patterns. For example, transmitting small messages across many RC QPs simultaneously and random accesses to

Table 4.1: MR control verbs exhaust the MTT cache and reduce bandwidth.

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

a large number of memory regions can cause certain types of severe cache misses (e.g., ICM and MTT/MPT) [58, 96]. ScaleRPC [19] found that this scalability problem can reduce the WRITE request rate by 90%.

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

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

We demonstrate this finding with a simple experiment on NVIDIA ConnectX-5 RNICs. We let the victim tenant use 6 cores, 16 connections per core, to issue 512B WRITE requests to exhaust the bandwidth capacity of the RNIC (i.e., 100 Gbps). Table 4.1 shows the results. The victim can achieve 96.6 Gbps with 17.2% MTT cache miss rate. The victim can still achieve line rate under such cache miss rate because QP multiplexing and the RNIC pipeline design can mask the overhead of cache misses to some degree. We let a single-threaded attacker keep registering memory regions (MRs) using `ibv_reg_mr` ($\sim$5K registration per second) on the victim's sender side. In this scenario, the victim's bandwidth is almost not affected, staying at 95.9 Gbps with the miss rate slightly increased to 22.9%. However, if the attacker keeps deregistering MRs, we can see a significant impact on the victim: the cache miss rate increases to 49.1%, and the bandwidth degrades to 48 Gbps. The overhead under such a high cache miss rate becomes significant and can no longer be masked by the RNIC processing pipeline. It is worthwhile to note that the attacker does not need to issue any data verbs, so the attacker consumes no network bandwidth or request rate at all. Fortunately, we observe that such interference is negligible at the receiver side.

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

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

4.3.3 Processing Units

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

Key finding #2: performance interference between different data verbs depends on the complexity of verbs. Different data verbs have different complexities. Simple verbs, like `send` and `read`, only copy data between machines. Complex verbs, such as `fetch_and_add`, atomically add a 64-bit value to the memory of a remote address. This operation leverages PCIe features (e.g., read-modify-write transactions), and may also acquire a lock on the

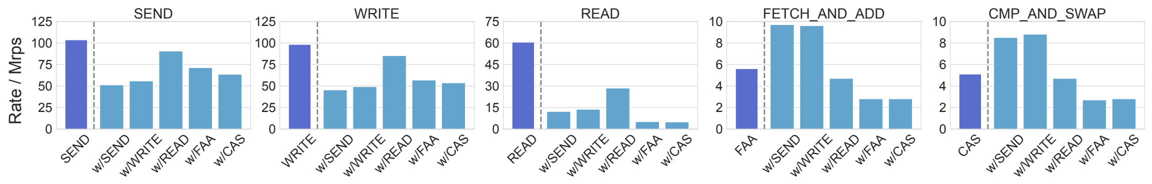


FIGURE 4.4: The contention of different data verbs on PU (NVIDIA). The leftmost bar on each subfigure is the request rate of running the victim only. The right 5 bars of each subfigure are the victim's rate when the attacker is running.

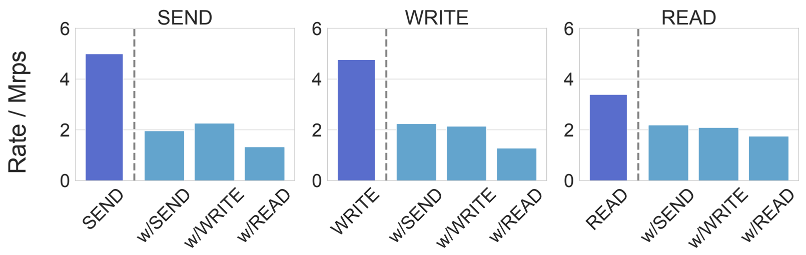


FIGURE 4.5: The contention of different data verbs on PU (Chelsio). The leftmost bar on each subfigure is the request rate of running the victim only. The right 3 bars of each subfigure are the victim's rate when the attacker is running.

target address. These complex verbs consume more PU resources, resulting in a lower request rate [58]. Our new discovery here is that this difference in resource consumption can also open a new pathway for performance interference through resource exhaustion: a victim's performance can be substantially penalized when colocated with an attacker that uses complex verbs intensively.

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

Figure 4.4.

Our first takeaway is that in addition to the ATOMIC operations [58], the READ operations are also more expensive than SEND/RECV and WRITE. When they are running alone (as victim traffic), FAA and CAS only achieve 5.2 Mrps and 4.8 Mrps respectively. READ achieves approximately 60 Mrps. SEND and WRITE can achieve more than 90 Mrps.

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

We also conduct similar tests on 100 Gbps Chelsio T62100-LP-CR RNIC, and the results are shown in Figure 4.5. This iWARP RNIC does not support ATOMIC operations. We observe that the iWARP RNIC's request rate for all types of data verbs is lower compared with RoCE RNICs, which matches findings from previous works [142, 91, 22]. We find that the contention among data verbs on Chelsio's RNIC also varies. For example, the victim with WRITE workload can achieve 4.76 Mrps without interference. The attacker can cause the victim's request rate to drop 55.0% with SEND workload and 73.1% with READ workload. The specific patterns are different from NVIDIA RNIC, but this result still demonstrates our key finding: the PU overhead of different data verbs varies.

Key finding #3: error handling can stall RNIC processing units and hang all the applications. RNICs need to handle a few types of errors, including transport timeout

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

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

On NVIDIA ConnectX-5 and ConnectX-6 RNICs, we find handling RNR errors can *completely* stall the RNIC processing units. For the victim, we use Perftest [108] to keep 128 outstanding 64KB WRITE requests on a single QP to saturate the bandwidth capacity. For the attacker, we only use a single QP (i.e., the SEND application in the table) to keep only one in-flight 4KB SEND request to consume a small amount of bandwidth. As shown in Table 4.2, if the SEND application generates traffic normally (e.g., the responder posts enough receive requests), it consumes 4 Gbps bandwidth, and the bandwidth for the victim only drops approximately 3.5 Gbps. However, when the SEND application triggers RNR errors (e.g., the responder side does not post any receive requests), both the SEND application and the victim are stalled. We test this RNR errors with both directions and see the same results. The reason is that the RNIC of the RNR receiver is stalled, and the RNIC cannot even process the ACK packet. The victim therefore is stalled even when they are sending traffics in the opposite direction.

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

4.3.4 PCIe Bandwidth

The RNIC is connected to the PCIe controller and transfers data from/to the CPU using PCIe lanes. The impact of PCIe on the networking stacks has been studied by several prior

Table 4.2: The impact of RNR errors on bandwidth. The unit is Gbps.

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

works [58, 94, 65]. Based on existing PCIe models, we further study how RDMA verbs consume and even use up the PCIe bandwidth. Previous works have already identified how RDMA loopback traffic can exhaust PCIe bandwidth [54, 64]. We therefore focus on the normal RDMA TX and RX traffic. To transfer an RDMA message, PCIe introduces the following types of extra bytes: (1) an MMIO to ring the doorbell on the RNIC (64B, depending on cache line size), (2) a Work Queue Element (WQE) (36B or 64B), (3) the PCIe protocol overhead (e.g., TLP headers), and (4) extra PCIe operations triggered by cache misses. Our key observation for PCIe bandwidth is:

Key finding #4: PCIe bandwidth will only become the bottleneck when the request size is in a specific range. We only need a single tenant to demonstrate this key finding. We run the experiment on NVIDIA 100 Gbps ConnectX-5 RNIC. The PCIe bandwidth capacity is 128 Gbps (PCIe Gen 3.0 x16). We use 96 QPs across 6 cores to saturate the PCIe TX bandwidth. Each QP keeps 256 outstanding WRITE requests. We vary the request size and collect both the NIC and the PCIe bandwidth consumption by reading the RNIC's counters. The result is shown in Figure 4.6. We first observe that when the payload size is small, the commodity RNIC can mitigate the WQE overhead by embedding the small message in the WQE. As shown in the green rectangle, when the request size is smaller than 28B, increasing the request size does not cause more PCIe bandwidth consumption because the payload is embedded in the same MMIO operation with the WQE.

Our second observation is that PCIe TX bandwidth may only become the bottleneck when the payload size of the request is in a specific range. The reason is that short requests are first throttled by the request rate before exhausting PCIe bandwidth while large requests are always throttled by the RNIC's bandwidth capacity. We confirm this

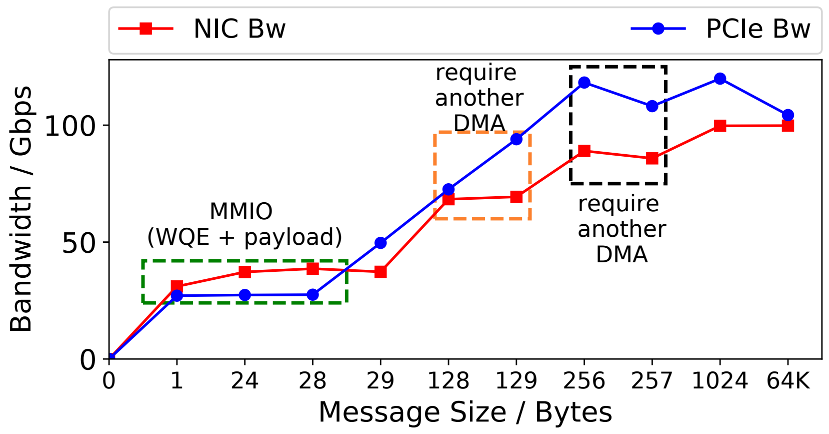


FIGURE 4.6: The PCIe bandwidth and RNIC bandwidth consumed by the application.

observation through a theoretical PCIe consumption model and we present two concrete examples. We assume the network MTU is 4096B and the maximum payload per PCIe transaction is 128B (the worst setting to maximize the PCIe overhead). The TLP overhead depends on the implementation [94] and we assume it as 20B, a typical size for a PCIe 3.0 device. Transmitting a 29-byte message will consume at most 127 network bytes and at least 189 PCIe bytes [135, 58]. Therefore, to saturate the link bandwidth (100 Gbps), we need at least 148.8 Gbps PCIe bandwidth, which is much larger than the PCIe 3.0x16 capacity. §4.10 includes the detailed computation. Our measurement shows that the actual consumption can be even higher, as shown in Figure 4.6. The consumption model for PCIe RX bandwidth (i.e., the RNIC to the host) is similar to that of TX. Additionally, too many cache misses may also cause high PCIe bandwidth consumption due to lots of PCIe reads to fetch metadata. However, in most scenarios, the large number of cache misses will first slow down the RNIC execution (e.g., introduce extra latency) and the PCIe bandwidth is therefore less consumed. In our measurement of cache misses, we do not observe cases where PCIe TX bandwidth is exhausted.

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

4.3.5 Other findings

We also have several other interesting findings. In the interest of space, we only briefly present them here. However, we do use these findings to guide our test suite design in §4.4.

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

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

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

4.3.6 The Resource Consumption Model

We summarize our findings in an RDMA operation model shown in Figure 4.7. This model describes which microarchitecture resource a verb operation consumes heavily. Note that

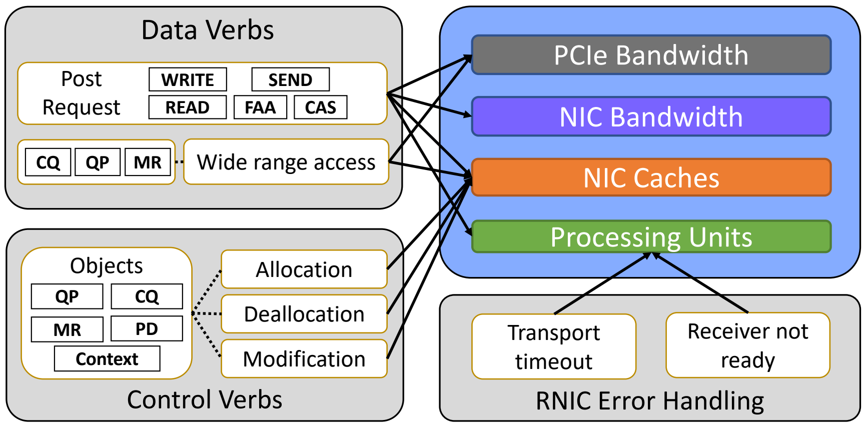


FIGURE 4.7: The relationship between verbs and microarchitecture resources. The arrow indicates heavy resource consumption.

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

4.4 The Husky Test Suite

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

Our test suite must be general: we will use it to test various RNIC performance isolation solutions on different RNICs. This means we cannot rely on tools and features from specific vendors, such as Mellanox Neo-Host [104]. In addition, different RNICs have different amounts of microarchitecture resources. And existing performance isolation solutions may only be able to mitigate contention on specific resources.

To this end, we build Husky to systematically test and evaluate RNIC performance isolation solutions. Husky targets at four types of resources: NIC bandwidth, PCIe bandwidth, NIC PU, and NIC cache. For each type of resource, we design synthetic workloads with different types of behaviors (e.g., control verbs) to exhaust this resource. More specifically, we exhaust NIC BW with long messages using different opcodes (e.g., WRITE); we exhaust PCIe bandwidth with loopback traffic and specific message patterns (from key finding #4); we exhaust NIC PU with expensive data verbs (key finding #2), small messages, or error handling behaviors (key finding #3); we exhaust different types of RNIC cache with intensive control verbs (key finding #1) and a wide range access of data verbs. We vary parameters (e.g., connection types) of some synthetic workloads to be more inclusive. In all, Husky includes 52 attacker synthetic workloads (6 for NIC BW, 4 for PCIe BW, 14 for NIC PU, and 28 for NIC cache) and 20 synthetic victim workloads. Many of the attacker workloads cannot be directly generated with existing RDMA traffic engines. We therefore extend Collie [64]'s traffic engine, the most flexible one to the best of our knowledge, to generate these synthetic RDMA traffics, including flexible control verbs workloads and error handling workloads.

Husky's framework can also easily allow running real applications as additional victim workloads. Husky currently contains two real applications, including the OSU benchmark [105] and eRPC-based Masstree key-value store [56, 77]. The OSU benchmark contains workloads such as allreduce and allgather. Note that we can integrate any RDMA applications into Husky. We test all the (victim, attacker) workload pair exhaustively from our test suite.

One key question is how to define a violation of performance isolation. Our definition of violation depends on the concrete isolation solution.

Husky uses a user-specified predicate to compute the expected performance results when isolation is enabled. Husky compares the actual performance with the expected performance to identify violation. For example, most of existing performance isolation solutions only provide bandwidth guarantee. The expected performance for these isolation solutions therefore is a guaranteed bandwidth, B_g. We assume the application can consume bandwidth of B_a when running alone. The bandwidth of this application should be at least $(1-\alpha)\min(B_a, B_g)$ under any attacker workload, where α is a tolerance level. A lower α means stricter isolation. We use an example to demonstrate how this definition works: let us assume that attacker and the victim are configured to share the same 100 Gbps network and we set α to be 25%. If the victim can achieve 60 Gbps when running alone, it should be able to achieve at least $(1-25\%)\min(60, 50) = 37.5$ Gbps under the attacker's workload. If the victim can only achieve 10 Gbps when running alone, its consumed bandwidth should not be less than $(1-25\%)\min(10, 50) = 7.5$ Gbps. In practice, we find all existing performance isolation solutions for commodity RNICs are bandwidth guarantee or can be translated into bandwidth guarantee. We use this definition for performance isolation violation in §4.5 and set α to be 25%.

4.5 Evaluation

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

We evaluate 3 different isolation solutions provided by RNIC vendors and prior work:

(1) NVIDIA separate hardware traffic class (HW TC). Cloud operators can set separate TCs for different tenants to use, which separate the RNIC bandwidth and packet buffers [82] to enforce performance isolation. Modern RNICs typically only have 8 traffic classes. This means we cannot use HW TC when we want to colocate more than 8 tenants in a physical server. (2) NVIDIA SR-IOV. Though the SR-IOV technique is designed for hardware virtualization, it provides separate virtual functions with some separated resources to different tenants and actually achieves some degrees of performance isolation [84]. (3) Justitia, a software-based performance isolation solution [142]. Justitia implements data verbs rate-limiting and pacing in RDMA userspace libraries to enforce performance isolation. This means Justitia has no security: malicious applications can easily circumvent the userspace library. Although Justitia's software architecture does not target a multi-tenant public cloud environment, we still use Husky to evaluate the effect of its isolation policy (e.g., its token-based algorithm). We also evaluate all the possible combinations of the above solutions[2]. Unfortunately, though we have a testbed with Chelsio T62100-LP-CR and Intel E810 NICs, we did not enable their hardware-based isolation mechanisms. Justitia also does not support Chelsio or Intel drivers. We therefore are not able to conduct the same evaluation on Chelsio or Intel NICs. [3]

4.5.1 Testing Existing Performance Isolation Solutions

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

We first take a look at the hardware-based isolation mechanism provided by NVIDIA.

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

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

Table 4.3: Performance isolation violation caused by exhausting microarchitecture resource. Justitia can only provide isolation among applications using the same function, so cannot be combined with SR-IOV. ✓ means performance isolation is properly enforced. ✗ means Husky can find a workload pair (attacker, victim) to violate performance isolation by exhausting microarchitecture resources.

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

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

The first three rows of Table 4.3 show the isolation effect provided by SR-IOV, HW TC, and the combination of them. Unfortunately, we find both SR-IOV and HW TC fail to provide enough isolation on RNIC's microarchitecture resources. For example, by exhausting RNIC's cache through either control verbs or data verbs, Husky can successfully affect the colocated victim's applications, even when both SR-IOV and HW TC are enabled. The key reason is that both SR-IOV and HW TC only isolate the architectural resources (e.g., link bandwidth) and do not restrict the cache usage of a single tenant. Husky therefore is able to use an attacker workload that exhausts RNIC cache, such as MTT/MPT cache. Other applications would suffer from severe cache miss and hence the performance drop. In addition, we find that although SR-IOV is mainly aimed at virtualization, it has indeed enforced some isolation, especially for RNIC PUs. The RC RNR error handling can cause RNIC PUs to pause and even hang the colocated applications if there is no performance isolation mechanism enabled. With SR-IOV, the RC RNR error does not affect tenants running on other VFs. However, the similar RNR exception handling process for UD and UC still violates the isolation of SR-IOV. Due to the RNIC's black box nature, we do not know the root cause of such a difference. Our best guess is that some part of the RNIC's PUs (e.g., that handles RC RNR) is isolated by different VFs, while other parts are not well isolated. These hardware-based solutions also cannot isolate PCIe bandwidth well. We observe that an attacker can consume substantial PCIe bandwidth and reduce the victim's usable bandwidth.

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

As shown in Table 4.3, Justitia does provide some PU isolation but to a limited extent. For example, Justitia takes the RNIC's request rate (i.e., execution throughput) into its isolation consideration. It therefore uses a pacer to control the request rate for each tenant and successfully prevent a single tenant from posting a large number of requests to exhaust the PUs. However, its isolation is violated when the attacker keeps posting requests that trigger error handling on the RNIC. The reason is that these errors are detected and handled by RNIC, which is out of Justitia's control. In addition, Justitia does not take cache and PCIe into consideration. The attacker tenant therefore can still exhaust the RNIC cache and PCIe bandwidth and cause other tenants to suffer from excessive cache misses or low usable PCIe bandwidth.

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

4.5.2 Impact for Real Applications

Next, we conduct experiments on a larger testbed to study how microarchitecture resource exhaustion impacts real application workloads when using state-of-the-art performance isolation solutions. We use the allreduce workload [105] on an RDMA-based MPI implementation [106] and eRPC-based Masstree (a key-value store) [56, 77] as two real victim applications. Our testbed consists of four physical servers. Each server is equipped with one 100 Gbps NVIDIA ConnectX-5 RNIC. The other settings are the same as §4.5.1. The victim applications run their VMs on all the four servers. The attacker tenant controls two VMs, each on a different server. We set up the testbed this way to emulate a real multi-tenant environment because an attacker may not have VMs colocated with all the victim's VMs. However, our results demonstrate that violation of performance isolation in a subset of the victim's VMs is already enough to substantially reduce the overall end-to-end performance of the real distributed applications.

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

We use four types of attackers from the Husky test suite to demonstrate our results: (1) BW attack is the baseline. We use the standard Perftest [108] `ib_write_bw` to set up a bandwidth-hungry application. It uses 16 RC QPs and each QP keeps 128 outstanding 1 MB WRITE requests to saturate the link bandwidth (consuming ∼50 Gbps when rate limiter is enabled). BW attack does not target any microarchitecture resources. (2) PCIe attack exhausts PCIe TX bandwidth. It runs 36 RC QPs on 6 cores and keeps 128 outstanding 257 B WRITE requests. It also consumes almost 50 Gbps link bandwidth (less than 20 Mrps) but causes more than 73 Gbps PCIe TX bandwidth consumption. This leaves only about 50 Gbps usable PCIe TX bandwidth (i.e., less than 50 Gbps usable network bandwidth) for the victim. (3) Cache attack exhausts RNIC cache. It runs 1536 RC QPs on 6 cores, uses 12288 MRs and each QP keeps only a single 256 B outstanding request. This attacker causes severe cache miss and only uses less than 7 Gbps link bandwidth (i.e., 3 Mrps). (4) PU attack pauses RNIC PUs. It runs 1 UC QP on a single core and keeps 128 outstanding SEND/RECV requests. Its receiver side does not post any receive requests, so the RNIC has to handle many receive not ready exceptions. It consumes less than 0.5 Gbps and less than 0.5 Mrps.

We begin with testing the RDMA-based allreduce workload. Allreduce is a collective communication operation widely used in distributed deep learning training. It aggregates a vector across all workers and propagates the result back to all workers. We set up 2 workers on each host (8 in total) to run allreduce. The allreduce buffer size is set to 1 MB. We run allreduce continuously and record the execution rate (allreduce operations per second). The raw rate without any isolation mechanism and interference is shown as the leftmost bars in the figure. The bar of no attack indicates the effect of enabling these isolation solutions. When Justitia is enabled, the allreduce rate drops by 38.5%. One possible reason is that Justitia uses a shim layer (the pacer) to exert sender admission control, which introduces extra performance overheads compared to the hardware-based solutions.

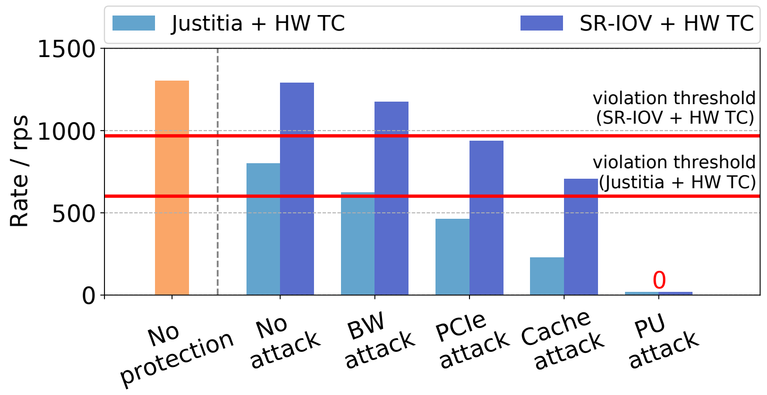


FIGURE 4.8: Allreduce results under exhaustion of different resources.

Since the allreduce workload only uses less than half of the NIC bandwidth (23 Gbps), its performance under attack should be at least $(1-\alpha)P_a$, where α is 25% and P_a is its performance without any attack. We can then compute a violation threshold in allreduce rate for each isolation solution based on the bandwidth the victim should consume.

The result for allreduce is shown in Figure 4.8. The horizontal red lines show the violation threshold. Bars under the red line indicate isolation violation. P_a for the application with Justitia + HW TC is 38.5% lower than that with SR-IOV + HW TC. This means the violation threshold is also 38.5% lower for Justitia + HW TC. We first observe that the BW attack only causes a negligible performance drop for SR-IOV + HW TC setting. And Justitia + HW TC also achieves the bandwidth isolation goal within the tolerance. We then observe that all the PCIe, Cache, and PU attacks successfully violate the isolation provided by either Justitia + HW TC or SR-IOV + HW TC. For example, the PCIe attack can cause the performance of the allreduce application to drop 27.3% for SR-IOV + HW TC and 42.1% for Justitia + HW TC. The impact of the Cache attack is more significant. Allreduce workload's performance drops more than half (71.3%) for Justitia + HW TC and almost half for SR-IOV + HW TC. We observe that the PU attack is the most powerful. It can directly stall the allreduce application by exhausting the RNIC PUs.

We use the same set of attackers to test the eRPC-based Masstree. We use the default setting of eRPC-based Masstree (e.g., key size and the number of threads). We set up the

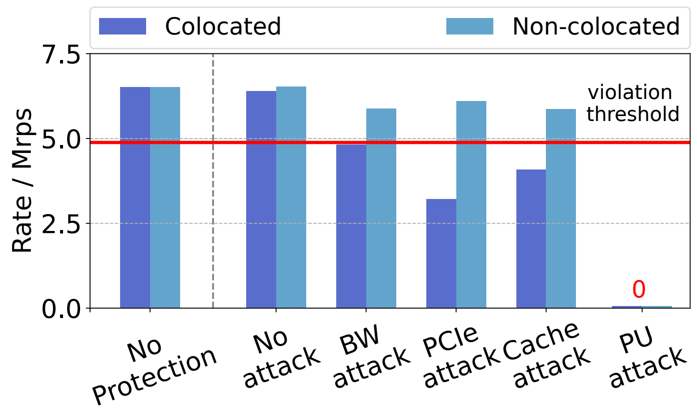


FIGURE 4.9: Mastree's GET rate under exhaustion of different resources. colocated means that the client and the attacker are on the same host. Non-colocated means that they are on different hosts.

key-value server in one physical server and three clients each in a different physical server. We colocate one attacker VM with the key-value server and another attacker VM with one of the clients. We collect the execution rate (in terms of the number of GET requests per second) and the latency from all the clients. The Masstree server only uses 14 Mrps and less than 20 Gbps, so we define the isolation violation as the same as the violation of allreduce. Figure 4.9 and Figure 4.10 show the GET rates and the latency results. The SR-IOV + HW TC more or less achieves the BW isolation goal within tolerance. We find that all microarchitecture resource exhaustion attacks successfully violate the isolation for the client that is colocated with an attacker VM. Similar to the allreduce workload, the PU attacker stalls the entire key-value store system. Worse still, it even pauses the clients that are not colocated with an attacker VM. This is because we stall the key-value server.

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

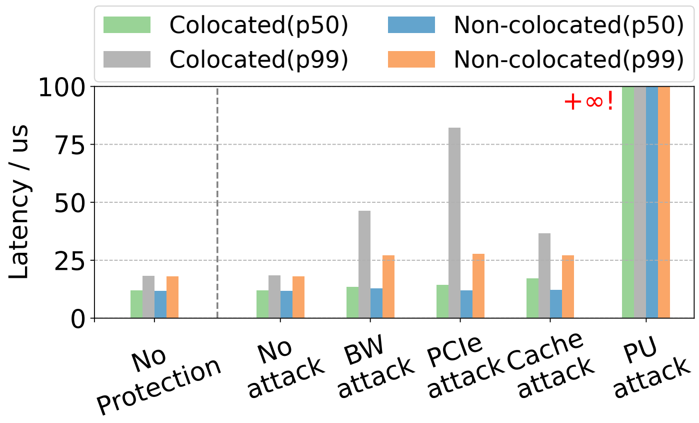


FIGURE 4.10: Mastree's latency under exhaustion of different resources.

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

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

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

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

4.5.3 Analysis for Existing Solutions

Our evaluation shows that all existing approaches fail to provide RDMA performance isolation.

We now analyze the fundamental restrictions of these solutions and some potential improvements we may achieve.

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

Justitia. Justitia uses delay to track resource contention and paces RDMA data verbs to allocate network bandwidth and executing rate among different tenants. As originally designed, Justitia does not provide isolation for control verbs or some RNIC resources (e.g,. RNIC cache). It is possible that Justitia could be modified to control more resources, but this requires more investigation. For example, the delay-based verbs pacing approach taken by Justitia could possibly detect RNIC cache resource contention as increases in latency and pace tenants' rate accordingly. However, it is unclear if this approach could accurately detect contention and if this approach would be responsive enough to prevent SLO violations from interference. Further, it is likely that having the RNIC provide per-tenant cache usage statistics could lead to a simpler and more accurate solution.

4.6 Guidelines

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

Hardware support for isolation is needed. Software approaches like Justitia [142] have a common problem. They only monitor architecture-level metrics, e.g., latency, bandwidth, and request rate. They cannot detect contention in microarchitecture resources, e.g., caches, let alone manage and fair share those resources. We believe future performance isolation solutions will have to leverage hardware support, similar to how modern hypervisors can use Intel Resource Director Technology (RDT) to monitor and manage access to the last-level cache and memory. NVIDIA RNICs expose several useful hardware counters, but they are still insufficient. For example, we can only observe cache misses, but we cannot manage the cache access or split the cache for different tenants.

A layer of indirection is needed. RDMA means kernel bypass for data verbs. This enables low latency and reduced CPU overheads. So where should performance isolation be enforced? We believe that future performance isolation solutions will require a layer of indirection either in NIC or in software. Having the enforcement point in the userland RDMA library (as Justitia) does not work, because it lacks security. Instead, a software indirection can have a microkernel-like design, with a set of cores running the isolation logic in a separate protection domain [81]. RDMA performance isolation should be enforced in such a central controller that takes over both control verbs and data verbs.

Programmer, compiler, and library support for RDMA applications. After a future performance isolation solution is invented, applications may need modification as well. If the future performance isolation solution requires strict partitioning of microarchitecture resources, this means each application has limited microarchitecture resources to use and can lead to substantially reduced performance. The amount of microarchitecture resource an application uses may also vary (depending on how many other tenants are on the same server or other configurations). Building high-performance RDMA applications will require additional effort for the programmer, compiler, and application library to efficiently use these limited resources. For CPU cache, these efforts occurred in the research community two decades ago [68, 80, 66].

4.7 Discussion

The impact of broken RDMA performance isolation. Our evaluation shows that a malicious tenant can cause other tenants' to suffer from drastic performance drop or even get stuck. In addition, a broken performance isolation exposes vulnerability for malicious users to conduct side-channel attacks. Since the tenant can affect others' performance on the same host, it can set up side channels that leak access patterns of victim nodes or deliver information by affecting the host's performance in a pattern [126]. RDMA performance isolation therefore is a critical feature for a secured RDMA public cloud.

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

Generalizability to other kernel bypass host networking architectures. Our test suite design is based on the verbs interface, which is RDMA-specific. However, we believe our methodology should be generalizable to find violations of performance isolation in other kernel bypass architectures, e.g., DPDK [29], 1RMA [123], as these implementations commonly require RDMA-like mechanisms in the DMA portion of the design. The industry trend today is to offload functions to hardware accelerators. For example, RDMA is offloading congestion control and reliable message delivery into the hardware. Microarchitecture resources in hardware are critical to delivering these offloaded functions. Paying attention to these microarchitecture resources for performance isolation is going to be increasingly important.

4.8 Related Work

Microarchitecture resources in RNICs. The existence of RNIC microarchitecture resources is well-known in the networking community, and many studies focus on how to design RDMA applications to circumvent certain RNIC performance anomalies due to these resources. For example, HERD [57], FaSST [59], and eRPC [56] avoid using RDMA reliable connection to mitigate the QP context cache miss for better scalability. ScaleRPC [19] and Flock [92] multiplex reliable connections in a time-sharing manner to mitigate the scalability problem. Kalia et al. [58] studies the RNIC's PCIe behaviors and provides guidelines for writing efficient RDMA programs. Unfortunately, these works only focus on optimizing applications to fully utilize the limited resources in RNICs. However, public cloud providers cannot control the third-party tenants' applications. Collie [64] conducts a systematic search on RDMA performance anomalies, and the anomalies are mostly due to oversubscribed microarchitecture resources. However, since Collie only focuses on first-party traffic, it just builds a search space based on normal operations. It therefore only considers normal data verbs and fails to uncover findings related to other types of behaviors. For example, the key findings #1, #2, and #3 in §4.3 are fundamentally not covered by Collie's search space because Collie does not take control verbs, error handling, and expensive atomic verbs into consideration. In all, prior works focus more from the perspective of application developers. Our work is on a complementary aspect by looking from the public cloud provider's perspective: how these microarchitecture resources affect performance isolation. This requires us to be microarchitecture resource aware and take a look at all types of RDMA behaviors, including control verbs and error handling, because we need to deal with misbehaving and even malicious tenants.

Other NIC performance isolation solutions. PicNIC [65] provides isolation for both packet processing and bandwidth on NIC. This allows latency-bound workloads not to be affected by bandwidth-bound workloads. FairNIC [37] isolates resources in SoC-based SmartNICs. Compared with them, our work focuses on the RDMA-related resources on NICs.

Performance isolation in other contexts. Performance isolation problems are not limited to NICs. Other server hardware components also have this issue, and they already have corresponding solutions. There exist several partitioning techniques for CPU caches [25, 43] and memory bandwidth [46]. Network bandwidth in the network fabric is also a crucial resource to isolate [114, 121, 111, 13, 14, 38, 134, 9, 69, 53, 112, 20, 52] as well as the switch processing piplines [130].

4.9 Summary

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

4.10 Appendix 1: Network v.s. PCIe

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

1. Ethernet overhead. Each Ethernet frame includes 14-byte Ethernet (exclude VLAN) header and 4-bytes CRC as L2 overhead. In addition, each Ethernet frame has L1 overhead - each frame is preceded by a 7-byte preamble and 1-byte start-of-frame delimiter. The frame is also followed by an inter-frame gap. The gap should be at least 12-byte. The total Ethernet overhead per frame therefore is 38-byte [50].

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

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

4. Infiniband overhead. The Infiniband protocol implements headers inside the UDP payload. A simple WRITE message through reliable connection (RC) needs 12-byte Base Transport Header (BTH), 16-byte RDMA Extended Transport Header (RETH), and 4-byte invariant CRC. Hence, the Infiniband protocol overhead is at least 32-byte [10].

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

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

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

3. TLP overhead. Each PCIe transaction has PCIe Transaction Layer Packet (TLP) header, and the header size varies for different PCIe implementation. We assume its least size as 20-byte according to [135, 58].

We next shows the computation of the 29-byte payload example in §4.3. The 29-byte payload is obviously less than the MTU, and can be sent using a single network packet. Therefore, the network bytes consumed by this payload is:

$$\text{Bytes(network)} = \text{Bytes(payload)} + \text{Bytes(Ethernet)}$$

$$+\text{Bytes(IP)} + \text{Bytes(UDP)} + \text{Bytes(IB)}$$

$$= 29 + 38 + 20 + 8 + 32$$

$$= 127\text{(bytes)}$$

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

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

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

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

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

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

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

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

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

4.11 Appendix 2: Response from NIC Vendors

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

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

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

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

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

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

5. Enabling RDMA Performance Isolation

In this chapter, we introduce Harmonic, the final core component of this dissertation. Building upon our understanding of RDMA microarchitecture and the insights gained from Collie and Husky, Harmonic is designed as a hardware-assisted solution to mitigate performance interference and ensure robust performance isolation in RDMA networks. We primarily use the Husky test suite to evaluate Harmonic. We evaluate its effectiveness based on whether Harmonic can successfully pass Husky's tests and measure its efficiency based on its performance overhead. Our evaluation results show that Harmonic is the first microarchitecture-aware solution and, to the best of our knowledge, is the first solution that can prevent all interference and anomalies uncovered by Husky. This work was completed in collaboration with Jiaqi Lou[1], Jinghan Huang, Wei Bai, Nam Sung Kim, Danyang Zhuo.

5.1 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 [12, 32], 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 [62, 118, 126, 124]. Although network performance isolation has been extensively studied in the past decades [13, 38, 53, 65, 121, 9, 37], recent work has highlighted that prior RDMA performance isolation solutions are insufficient for public clouds [62]. An

[1] Jiaqi and I contribute equally to this work and share co-first authorship. She focused on the hardware design and implementation, and I focused on the software design and implementation.

RDMA NIC (RNIC) has microarchitecture resources, such as on-NIC cache and on-NIC processing units that significantly affect RDMA performance [62, 56, 58]. 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 [100, 62].

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 feasibly 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 [64, 39, 144, 137].

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 [144] as congestion control algorithm [101, 16, 51]. 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 [62], and a popular in-memory database application, Redis over RDMA [143]. We compare Harmonic with other performance isolation solutions, including hardware Single Root I/O Virtualization (SR-IOV), separate hardware queues, and Justitia [142]. 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 [62].

Lastly, current Harmonic supports 25 Gbps RNICs, limited by the speed of the PCIe physical layer (PCIe PHY) in our commodity FPGA development board[2]. 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.

5.2 Background

5.2.1 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 [39, 12, 32]. For example, Bai et al. [12] demonstrated that more than 70% of traffic in Azure is RDMA.

Figure 5.1 shows the four key components (*i.e.*, userspace libraries, kernel drivers, RNIC

[2] 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, 2], 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 §5.7.

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 [103]. 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 has on-NIC cache to store QP contexts [58], 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. Specifially, 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.

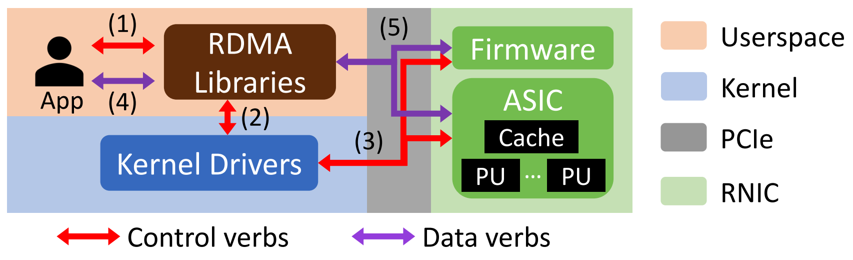


Figure 5.1: RDMA workflow.

(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 [64].

5.2.2 RDMA Performance Isolation

RDMA has already been successfully adopted in accelerating first-party workloads such as storage [12, 32]. 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 [62].

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 [6, 85]. 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 [62], 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 [56, 58, 64, 19]. 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.

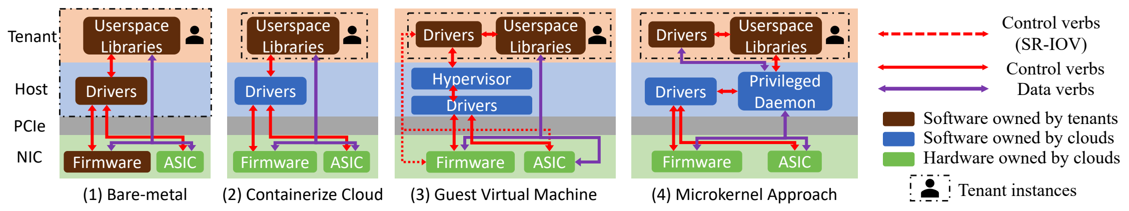


FIGURE 5.2: RDMA virtualization schemes.

5.2.3 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 5.2 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 [102]. Cloud providers have 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 [110] and MasQ [42] 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 [60] and mRPC [18]. 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 [142], 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.

5.3 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 [58, 62], 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 [64, 144, 39]. Therefore, simply discarding excessive RDMA packets in hardware [28] can cause RDMA performance degradation and is not an

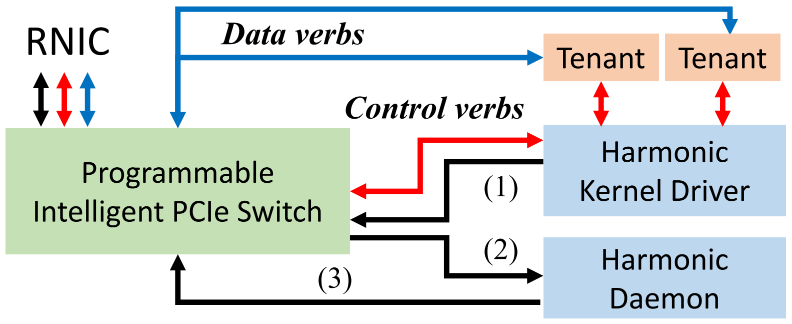


Figure 5.3: Harmonic overview.

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 [144]). 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 (§5.6.5).

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 (§5.6.3). In summary, Harmonic can handle the (2), (3) and (4) scenarios in Figure 5.2. 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 5.3 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. 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 (§5.6.5).

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 [34], which work across different vendors' RNICs. It is important to note that our performance abstraction intentionally

Table 5.1: 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

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 5.1 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.

5.4 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

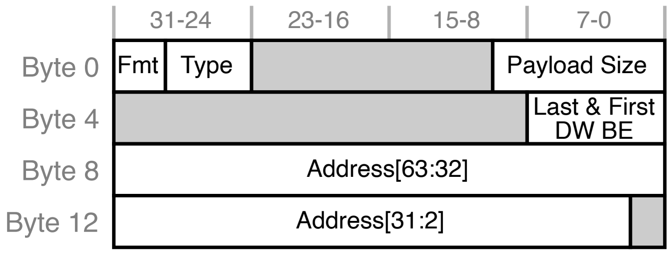


FIGURE 5.4: TLP header format where the gray blocks represent unused fields for PIPS. DW BE denotes dword byte enable.

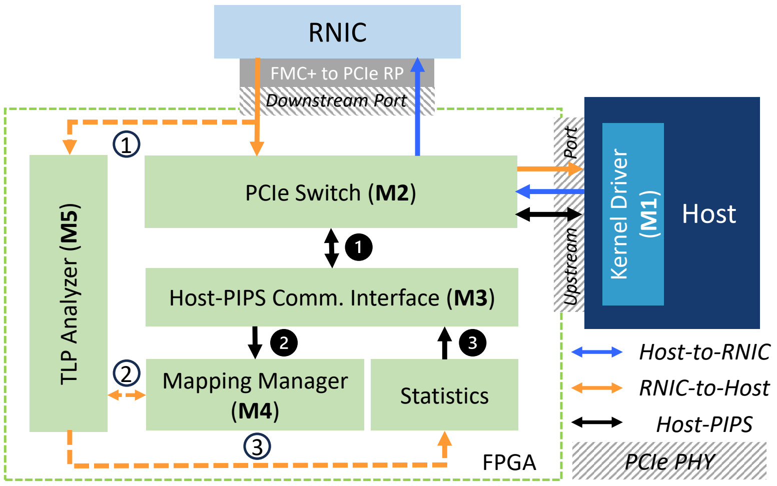


FIGURE 5.5: Programmable Intelligent PCIe Switch (PIPS) internal architecture. The dash line indicates asynchronous TLP analysis, decoupled with PCIe switch forwarding path.

from the RNIC-issued DMA read/write TLP header (Address field in Figure 5.4) 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 5.5): (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.

5.4.1 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 [7]. The PCIe PHYs in the two instances are configured as PCIe switch's upstream port and downstream port, respectively. Figure 5.5 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 [45] to be connected to RNIC.

5.4.2 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 §5.11.

5.4.3 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).

5.4.4 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.

5.5 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.

5.5.1 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` [108] 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 5.1), and calculates tenant's PCIe allocation using dominant resource fairness model [33]. 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

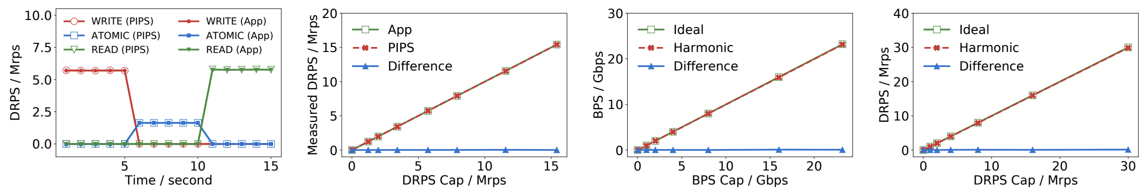
T_i intervals (in microseconds) to manage tenants' rate. Equation 5.1 shows how interval is updated based on the measured rate and target rate. We use two heuristic parameters T_{min} and T_{basic} in practice. T_{min} is a minimal interval threshold to avoid excessively frequent adjustments, which could 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.

$$T_i = max(T_{min}, T_{basic} * (1.0 - \frac{R_{current} - R_{target}}{R_{target}})) \tag{5.1}$$

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.

5.5.2 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



(a) Measurement of operation types (b) Measurement of RPS (c) Control of BPS (d) Control of RPS

FIGURE 5.6: Measurement and control of RDMA traffic. App denotes the performance metrics as reported by `perftest`.

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[62]. 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.

5.6 Evaluation

5.6.1 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` [108], to measure the DRPS capacity, and the result is $\sim$30 M DRPS.

Both servers are running Ubuntu 20.04. Harmonic kernel driver is built upon MLNX_OFED-5.8.1.1.2.1 [103], 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.

5.6.2 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 5.6a, PIPS successfully identifies the types of data verbs and measures the request rates of each workload accurately. Figure 5.6b 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 5.6c and Figure 5.6d 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 5.1, 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.

5.6.3 Harmonic End-to-end Evaluation

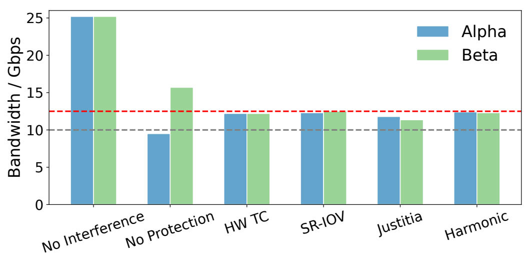
We use the state-of-the-art RDMA performance isolation test suite, Husky [62], to perform end-to-end evaluation of Harmonic. Husky includes a set of victim traffic patterns that are sensitive to different types of resource contention, and four sets of attacker traffic patterns that exhaust four types 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 [84]; (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 [82]; (3) Justitia, a recent software-based isolation solution for RDMA networks [142]. 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).

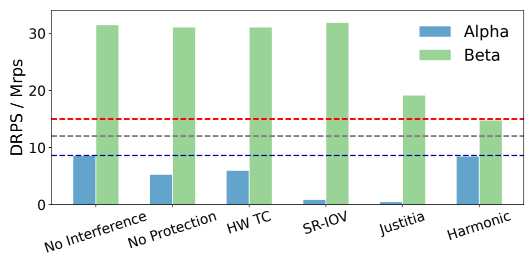
We allocate two tenants, named Alpha and Beta, on the same pair of hosts. Our RDMA hosts support up to 25 Gbps and ~30 M DRPS. We thus set our isolation goal to be that both Alpha and Beta are guaranteed with 12.5 Gbps and 15 M DRPS. For each attack, we let Alpha run a Husky victim traffic that is sensitive to a specific type of resource, and we let Beta run an attacker traffic targeting the specified resource. Results are shown in 5.7a to 5.7d. 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 denote the tolerance bar (10 Gbps and 12 Mrps). We use a blue dash line in 5.7b 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.

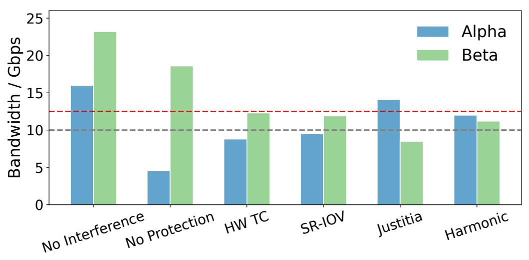
Processing capacity contention. We let Alpha run a throughput-sensitive Husky victim, which uses 36 connections and keeps issuing 64-byte messages. Beta uses 64 connections to keep generating expensive 8-byte ATOMIC traffic to exhaust RNIC processing capacity. We normalize DRPS based on our profiling results, which show that a single ATOMIC operation costs roughly three times as WRITE operations. 5.7b 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



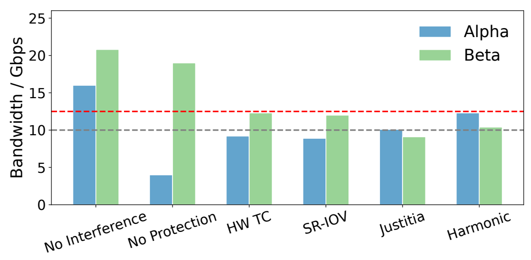
(a) NIC bandwidth contention.



(b) Processing capacity contention.



(c) RNIC cache contention.



(d) PCIe bandwidth contention.

FIGURE 5.7: Contention in various components of the system, including NIC bandwidth, processing capacity, RNIC cache, and PCIe bandwidth.

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. 5.7c 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 5.7d, 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.

5.6.4 Performance Isolation for End-to-End Applications

We evaluate how Harmonic provides performance isolation for a real application. We use an RDMA-based Redis [143] 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 [143]. 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 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 5.8, 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

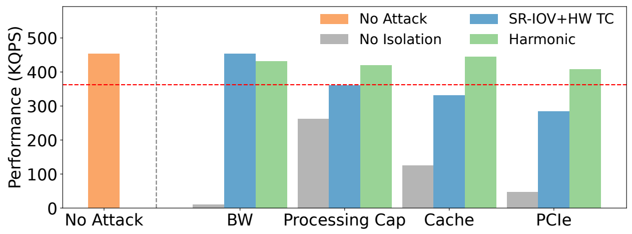


FIGURE 5.8: Performance of Redis over RDMA across different attack types and isolation schemes.

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∼1.4x better than the combination of two state-of-the-art isolation solutions.

5.6.5 Overhead Analysis

Hardware and PCIe costs. Our hardware cost analysis based on the implementation report from AMD Vivado [8] 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 can be comfortably accommodated by the bandwidth slack between PCIe and RNIC line rate. The detailed calculation and analysis is in §5.12. 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 5.2. We demonstrate the latency penalty under different packet sizes in Figure 5.9. 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 [97], and only introduces nanosecond-level latency overhead [133]. We therefore believe Harmonic's PIPS overhead can also be mostly eliminated by implementation in ASIC and being integrated into RNIC.

Table 5.2: 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

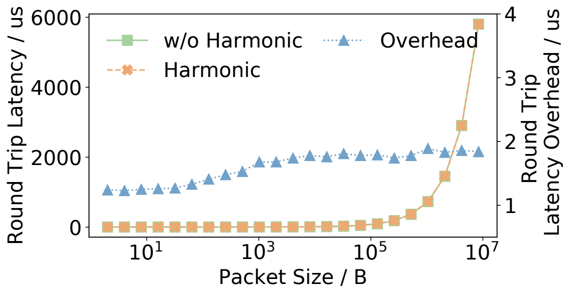


FIGURE 5.9: Latency overhead across different packet sizes. The green and orange lines present the absolute round trip latency with left y-axis when packet size differs. The blue line demonstrates the round trip latency overhead is less than 2 us using the right y-axis.

5.7 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 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 [56, 58]. 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 [110] and MasQ [42] 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) [98] 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 [63]. Harmonic can also be adapted to isolate resources among different I/O devices and hence manage the complex intra-host network.

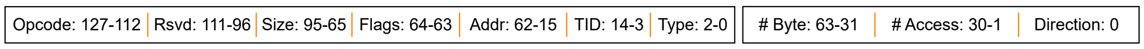
5.8 Related Works

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 [58, 56, 92, 19, 62]. Husky [62] 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. [58] 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 [132, 91, 131, 71]. SRNIC [132] modifies both protocols and RNIC architecture to improve on-NIC memory efficiency and utilization for better scalability. IRN [91] 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 [63, 94, 70, 138, 4, 5]. Breaking Band [138] leverages an expensive commercial PCIe analyzer to get a system-level PCIe latency breakdown. Min [88] implements a simple soft PCIe switch to obtain CPU-GPU communication patterns. Neugebauer et al. [94] 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 [13, 38, 53, 112, 114, 65, 141, 139, 48] 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 [65] 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.



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

FIGURE 5.10: Address-to-object/tenant mapping and statistics entry formats.

5.9 Summary

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.

5.10 Appendix 1: Harmonic Prototype Setup

We present the prototype setup of Harmonic in Figure 5.11. 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 [129] which is a versatile and widely adopted standard for high-performance interfacing FPGAs with external devices.

5.11 Appendix 2: Entries for Mappings and Statistics

We illustrate the address-to-object/tenant mapping and statistics entry format in Figure 5.10. We explain how each field is derived to offer a comprehensive understanding of the mapping mechanism in Harmonic.

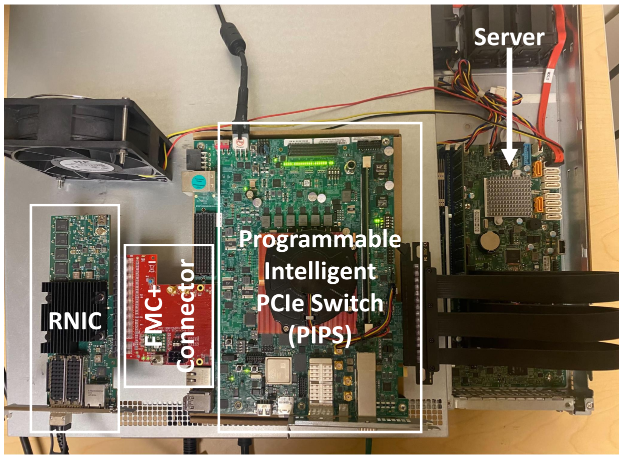


Figure 5.11: Programmable Intelligent PCIe Switch (PIPS) Prototype.

5.11.1 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 5.10a shows our unified entry to update such address-to-object/tenant mappings to our PCIe switch. Note that the 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.

5.11.2 Per-tenant RDMA Statistics

We store the monitored RDMA resource statistics in a 128-bit structured entry as shown in Figure 5.10b. As discussed in §5.4, 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 5.4) in per-tenant statistics entries.

5.12 Appendix 3: 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 [107]. 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.

6. Conclusion

In modern datacenter networks landscape, RDMA performance anomalies pose significant challenges, leading to degraded network reliability and unpredictable application performance. How can we proactively uncover and prevent these performance anomalies in RDMA networks? This dissertation reveals that the key is to delve into RDMA's microarchitecture. This dissertation presents three systems and demonstrates the feasibility of developing a systematic understanding of the complex RDMA microarchitecture. More importantly, we show that by leveraging critical insights gained from this understanding, we can effectively and efficiently identify and prevent RDMA performance anomalies.

We next discuss the lessons we have learned, the broader impacts of this work, and a few research directions left for future exploration.

6.1 Lessons

Besides the thesis statement, the most important lesson, we share a few other lessons we have learned throughout this dissertation.

RDMA Performance Is Critical in Modern Datacenter Networks. In modern datacenters, the scale of jobs has grown significantly. For example, a single LLM training job may use hundreds or even thousands of machines together as a cluster. This makes the performance of underlying RDMA network a pivotal factor. The reliability and performance of the entire cluster can drastically degrade if any individual node or single point-to-point RDMA communication underperforms. For instance, in widely-used collective operations like allreduce (*e.g.*, ring-based), a single slow receiver can slow down the completion time of the entire collective operation. Moreover, RDMA's inherent characteristics can even amplify these issues. For example, RDMA typically requires lossless fabrics, and even a performance anomaly at one node can have catastrophic consequences, such as triggering Priority Flow Control (PFC) deadlocks, threatening the entire cluster. In this dissertation, the testing results of Collie and Husky strongly demonstrate the severity of these performance anomalies.

As a result, the reliability requirements for datacenter networks have significantly increased. Operators must rigorously test and validate RDMA networks to ensure they are free from these anomalies. Collie and Husky of this dissertation serve as a systematic test suite to uncover these potential threats.

RDMA Microbehaviors Beyond Data Transmission. In traditional networks, performance issues are typically associated solely with data transmission behaviors. However, in RDMA networks, due to the nature of hardware offloading, numerous other microbehaviors, such as control path operations (*e.g.*, connection setup) and error handling (*e.g.*, handle receiver not ready errors), also consume or even exhaust significant microarchitecture resources on RDMA NICs, impacting overall performance. This makes it essential to include all these behaviors within the scope of our study to uncover and prevent performance anomalies. In developing Husky, we systematically study all these behaviors and analyze their impacts on the microarchitecture resources usage. We were the first to reveal that control paths operations and error handling can have substantial impact on RDMA performance, even pausing the entire RNIC. In Harmonic, we carefully modulate all these behaviors for each application to ensure performance isolation and prevent anomalies.

Hardware and Software Co-Design Is Essential for RDMA Performance. Another lesson from our work is that hardware assistance is crucial when optimizing RDMA performance. On the one hand, the understanding of hardware (especially those microarchitecture) is crucial. For example, without thoughtful software design that understands and accommodates these hardware features, the performance benefits of RDMA may be underutilized or even negated (*e.g.*, a workload that triggers performance anomaly). Collie and Husky has delivered a set of design guidelines for application developers and cloud operators to write high-performance and anomaly-free RDMA programs.

On the other, hardware assistance makes performance monitoring, diagnosis, and management much easier for RDMA networks. Since the data transmission of RDMA completely bypasses the OS kernel and the host CPU, cloud operators do not have sufficient visibility and control over RDMA traffics through traditional approaches (*e.g.*, through OS kernels or hypervisors). For example, none of existing approaches can accurately monitor per application's RDMA microbehaviors, such as how many on-NIC cache miss an application triggers. With hardware assistance, such as hardware-assisted monitoring proposed in Harmonic, we can develop a solution to accurately monitor and modulate per application's RDMA microarchitecture resource usage, enforcing performance isolation. It is worthwhile to note that such hardware assistance does not need to be enforced through a separate component. These capabilities can be also integrated into future generations of RNICs to be directly accessed by users.

6.2 Impacts

The broader impacts of this dissertation include both intellectual merits that motivate a line of work to continue to improve RDMA networks reliability in modern data centers, and real world impacts that can directly enhance the robustness and performance of existing infrastructures.

6.2.1 Intellectual Merits

Novel Insights into RDMA Microarchitecture and Microbehaviors. This dissertation is the first to systematically reveal the complex microarchitecture of modern RDMA networks. In addition, we not only have studied data verbs, but also have comprehensively analyzed all types of RDMA microbehaviors, including control verbs and error handling. We investigate how they impact resource usage and overall RDMA performance, and introduce a qualitative microarchitecture resource consumption model for modern RDMA networks. These novel findings have provided a theoretical foundation for many research in RDMA domain [73, 117, 137, 132, 1, 74, 67, 76, 75, 41, 125].

Design Guidelines for RDMA Application Development. Besides the understanding of RDMA microarchitecture and microbehaviors, we have also provided a set of design guidelines for developers to write anomaly-free applications. For example, Collie suggest an

RDMA RPC library developer to use RDMA WRITE and configure queue depth appropriately, and suggest a DML training framework developer to avoid scatter-gather batching to avoid performance anomalies in particular clusters [54]. Many of the guidelines summarized in Collie and Husky have also been adopted by a line of recent works [18, 117, 55, 125].

Inspiration for Next-Generation RNIC Design. This dissertation also provides many inspirations for future RNIC design. Collie and Husky demonstrates that the visibility (*e.g.*, hardware counters) provided by RNIC can be valuable for cloud providers to better operate their RDMA networks. The analysis of modern RDMA microarchitecture also reveal its limitations, such as the lack of fine-grained isolation. Besides, Harmonic strongly proves that adding only a small set of features (*e.g.*, monitoring) will be sufficient to build robust solutions to prevent RDMA performance interference and anomalies on commodity RNICs. Some of these inspirations are also applied and further explored in recent works [132].

6.2.2 Real-world Impacts

Open-Source Software. Collie and Husky projects are fully open-sourced and available at `https://github.com/host-bench/`. This enables groups from both academia and industry to use these systems to understand RDMA microarchitecture, uncover performance anomalies, and continue to research on improving the reliability and performance of RDMA networks.

Security Bulletins and Firmware Upgrades. We use Collie and Husky to test various types of testbeds and have uncovered numerous new performance anomalies. All these unseen anomalies have been acknowledged and reproduced by corresponding RNIC manufacturers. Many of the anomalies have raised huge concerns because of their potential security vulnerabilities, leading to release of security bulletins and firmware upgrades [100]. This strongly demonstrates the huge impacts of this dissertation that it has helped to improve the reliability and performance of commodity RNICs, which benefits many RNIC manufacturers and major cloud providers.

Wide Deployment in Industry Production. The practical impacts of this dissertation also extends beyond the anomalies themselves. Collie and Husky have been adopted by many industrial collaborators. Collie is used in ByteDance, Microsoft, and NVIDIA to test their RDMA networks. Husky has been applied by many RNIC manufacturers and cloud providers to identify performance interference in their RNICs, including Microsoft, NVIDIA, Intel, and DreamBig. In addition, parts of Harmonic are also being integrated into the design of next-generation RNICs by one of the major cloud providers to enable robust performance isolation.

6.3 Future Works

We next introduce a few directions for future research exploration.

6.3.1 Performance Anomalies of Various I/O Devices.

Besides RNICs, various types of I/O devices, such as GPUs, SSDs, and FPGAs, have been widely deployed in today's datacenters. The performance of these high-speed I/O devices is crucial for modern datacenter applications. For example, the LLM training job heavily rely on GPU for its computation and frequently access storage devices for data loading or saving checkpoints. In this dissertation, we primarily focus on performance anomalies in RDMA networks. However, performance anomalies that happen to these other I/O devices can also lead to catastrophic consequences, including slowdown and even impairing the training accuracy. Therefore, we believe to systematically uncover and prevent performance anomalies of various types of I/O devices is increasingly important in modern datacenters. Besides, most of these I/O devices encompass hardware offloading nature and have complex microarchitecture, similar to RNICs. Therefore, we believe the methodology and the key idea behind this dissertation can be insightful when conducting research on other I/O devices.

6.3.2 Towards Complex Intra-Host Networks.

As datacenter networks evolve, the complexity of the intra-host network is growing, particularly with the integration of the aforementioned accelerators. This complex interconnect also presents unique challenges for traditional network solutions. For example, the resource bottleneck within a host (*e.g.*, memory bus, and PCIe) can cause new type of congestion, the host congestion. Such bottlenecks may also lead to severe performance interference among applications. This dissertation presents a set of solutions to identify and address problems for the end-to-end RDMA network. However, the complex intra-host fabrics has not been fully explored yet. There are still many interesting and crucial problems to address for the increasingly important intra-host networks. For example, a few recent works have studied the host congestion problems [63, 4, 74], but how to enforce a holistic solution for fine-grained monitoring and control over the heterogeneous intra-host fabrics remains unexplored.

6.3.3 Enhancing Fault Tolerance in RDMA Networks.

RDMA networks typically require fine-tuning of both the host systems and network fabrics (*e.g.*, lossless) to achieve desired performance. This makes RDMA performance highly susceptible to hardware faults or misconfigurations. Our dissertation provides effective tools for identifying performance anomalies in offline situations and preventing performance interference at runtime. However, certain dynamically occurring problems, like link flapping, are beyond its scope. These types of faults can also drastically degrade RDMA networks' performance, and RDMA's unique characteristics (*e.g.*, hardware offloading) make many traditional approaches ineffective. These problems become increasingly critical for modern datacenter applications, especially those with high performance demand and substantial failure overheads, such as LLM training. Developing RDMA networks with robust fault tolerance mechanisms is thus a crucial and compelling research direction, offering significant benefits for the reliability of datacenter applications.

6.4 Final Remarks

In this dissertation, I propose a novel thesis, arguing that it is both practical and essential to understand the complex microarchitecture of modern RDMA networks. With this systematic understanding, we can effectively and efficiently identify and prevent critical performance anomalies. While we do not claim that the designs and implementations based on this idea are the only solutions, these designs and implementations strongly support our proposed statement and methodology. We believe that the insights into and utilization of RDMA microarchitecture contributed by this dissertation will enable the community to further enhance the reliability and performance of RDMA, laying a solid foundation for building future high-speed network infrastructure.

Bibliography

[1] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. Reverie: Low pass {Filter-Based} switch buffer sharing for datacenters with {RDMA} and {TCP} traffic. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 651–668, 2024.

[2] Advanced Micro Devices. How to implement pcie switch on Ultrascale. `https://support.xilinx.com/s/question/0D54U00006cnZ2rSAE/`, 2023.

[3] Advanced Micro Devices. Use Ultrascale+ PCIe Integrated block as an endpoint PCI to PCI Bridge device. `https://adaptivesupport.amd.com/s/question/0D54U00006hwlreSAA`, 2023.

[4] 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.

[5] 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.

[6] Amazon. Amazon EC2 Instance Types. `https://aws.amazon.com/ec2/instance-types/`, 2024.

[7] AMD/Xilinx. Versal Adaptive SoC Integrated Block for PCI Express LogiCORE IP Product Guide. `https://docs.xilinx.com/r/en-US/pg343-pcie-versal`, 2024.

[8] AMD/Xilinx. Vivado Design Suite. `https://www.xilinx.com/products/design-tools/vivado.html`, 2024.

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

[10] Infiniband Trade Association. Rocev2. `https://cw.infinibandta.org/document/dl/7781`, 2014.

[11] Aviv Barnea, Itay Ozery, and Barbara Claman. Scaling Zero Touch RoCE Technology with Round Trip Time Congestion Control. `https://developer.nvidia.com/blog/https://forums.developer.nvidia.com/t/198048`, 2021.

[12] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. Empowering azure storage with {RDMA}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49–67, 2023.

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

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

[15] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed Greybox Fuzzing. In *CCS*, 2017.

[16] Broadcom. Introduction to Thor Congestion Control for RoCE. `https://docs.broadcom.com/doc/NCC-WP1XX`, 2023.

[17] Chiranjeeb Buragohain, Knut Magne Risvik, Paul Brett, Miguel Castro, Wonhee Cho, Joshua Cowhig, Nikolas Gloy, Karthik Kalyanaraman, Richendra Khanna, John Pao, et al. A1: A Distributed In-Memory Graph Database. In *SIGMOD*, 2020.

[18] 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.

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

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

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

[22] Chelsio Communications. 100g network performance for illumos. `https://www.chelsio.com/wp-content/uploads/resources/t6-100g-nic-illumos.pdf`, 2018.

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

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

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

[26] Alireza Farshin, Amir Roozbeh, Gerald Q. Maguire Jr., and Dejan Kostić. Reexamining Direct Cache Access to Optimize I/O Intensive Applications for Multi-hundred-gigabit Networks. In *USENIX ATC*, 2020.

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

[28] 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.

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

[30] Vijay Ganesh, Tim Leek, and Martin Rinard. Taint-Based Directed Whitebox Fuzzing. In *ICSE*, 2009.

[31] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 57–70, 2024.

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

[33] 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.

[34] GitHub. RDMA Core Userspace Libraries and Daemons. `https://github.com/linux-rdma/rdma-core/`, 2024.

[35] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. Grammar-Based Whitebox Fuzzing. In *PLDI*, 2008.

[36] Patrice Godefroid, Michael Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. In *NDSS*, 2008.

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

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

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

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

[41] Zhiqiang He, Yuxin Chen, and Bei Hua. Roud: Scalable rdma over ud in lossy data center networks. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 36–46. IEEE, 2023.

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

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

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

[45] HiTech Global. PCI Express Gen4 Root FMC+ Module. `https://hitechglobal.us/index.php?route=product/product&path=18_85&product_id=273`, 2024.

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

[47] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, and Tianwei Zhang. Characterization of Large Language Model Development in the Datacenter. https://arxiv.org/abs/2403.07648, 2024.

[48] Shuihai Hu, Wei Bai, Kai Chen, Chen Tian, Ying Zhang, and Haitao Wu. Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud. *IEEE Transactions on Cloud Computing*, 9(2):763–776, 2018.

[49] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Deadlocks in Datacenter Networks: Why Do They Form, and How to Avoid Them. In *HotNets*, 2016.

[50] IEEE. Ieee standard for ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pages 1–5600, 2018.

[51] Intel. Intel® Ethernet 800 Series Linux Flow Control. `https://www.intel.com/content/www/us/en/support/articles/000088691/ethernet-products/800-series-network-adapters-up-to-100gbe.html`, 2023.

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

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

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

[55] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

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

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

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

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

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

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

[62] 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 *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 31–48, 2023.

[63] 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.

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

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

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

[67] Yanfang Le, Jeongkeun Lee, Jeremias Blendin, Jiayi Chen, Georgios Nikolaidis, Rong Pan, Robert Soulé, Aditya Akella, Pedro Yebenes Segura, Yuliang Li, et al. Sfc: Near-source congestion signaling and flow control. *arXiv preprint arXiv:2305.00538*, 2023.

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

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

[70] Qiang Li, Qiao Xiang, Derui Liu, Yuxin Wang, Haonan Qiu, Xiaoliang Wang, Jie Zhang, Ridi Wen, Haohao Song, Gexiao Tian, Chenyang Huang, Lulu Chen, Shao-zong 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.

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

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

[73] Kefei Liu, Zhuo Jiang, Jiao Zhang, Shixian Guo, Xuan Zhang, Yangyang Bai, Yong-bin Dong, Feng Luo, Zhang Zhang, Lei Wang, et al. R-pingmesh: A service-aware roce network monitoring and diagnostic system. In *Proceedings of the ACM SIG-COMM 2024 Conference*, pages 554–567, 2024.

[74] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. Hostping: Diagnosing intra-host network bottlenecks in {RDMA} servers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 15–29, 2023.

[75] Kefei Liu, Jiao Zhang, Zhuo Jiang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. Diagnosing end-host network bottlenecks in rdma servers. *IEEE/ACM Transactions on Networking*, 2024.

[76] Xiaoxiao Ma, Fan Yang, Zhan Wang, Ning Kang, and Xunjun An. Understanding the scalability problem of rnic cache at the micro-architecture level. In *ICC 2023-IEEE International Conference on Communications*, pages 6059–6065. IEEE, 2023.

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

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

[79] Ashlie Martinez and Vijay Chidambaram. CrashMonkey: A Framework to Automatically Test File-System Crash Consistency. In *HotStorage*, 2017.

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

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

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

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

[84] Mellanox Single Root IO Virtualization (SR-IOV). `https://docs.nvidia.com/networking/display/mlnxofedv53100143/single+root+io+virtualization+(sr-iov)`, 2023.

[85] Microsoft. Azure Virtual Machine series. `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/`, 2024.

[86] Barton Miller, Mengxiao Zhang, and Elisa Heymann. The Relevance of Classic Fuzz Testing: Have We Solved This One? *IEEE Transactions on Software Engineering*, page 1–1, 2020.

[87] Barton P. Miller, Louis Fredriksen, and Bryan So. An Empirical Study of the Reliability of UNIX Utilities. *Commun. ACM*, 33(12):32–44, December 1990.

[88] Seung Won Min. *Fine-grained memory access over I/O interconnect for efficient remote sparse data access.* PhD thesis, University of Illinois Urbana-Champaign, 2022.

[89] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *USENIX ATC*, 2013.

[90] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: RTT-Based Congestion Control for the Datacenter. In *SIGCOMM*, 2015.

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

[92] 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.

[93] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, KR Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models. https://arxiv.org/abs/2104.05158, 2021.

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

[95] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python. `https://github.com/fmfn/BayesianOptimization`, 2014.

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

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

[98] NVIDIA. NVIDIA ConnectX-6 DX Datasheet. `https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectX-6-dx-datasheet.pdf`, 2021.

[99] NVIDIA. Nvidia mellanox winof vpi documentation - performance tuning and counters. `https://docs.nvidia.com/networking/display/winofv55053000/performance+tuning+and+counters`, 2023.

[100] NVIDIA. Security bulletin: Nvidia connectx - april 2023. `https://nvidia.custhelp.com/app/answers/detail/a_id/5459`, 2023.

[101] NVIDIA. DCQCN PARAMETERS. `https://enterprise-support.nvidia.com/s/article/dcqcn-parameters`, 2024.

[102] NVIDIA. Firmware Burning Tools (MFT). `https://docs.nvidia.com/networking/category/mft`, 2024.

[103] NVIDIA. MLNX_OFED InfiniBand/VPI. `https://docs.nvidia.com/networking/category/mlnxofedib`, 2024.

[104] NVIDIA. Nvidia® mellanox® end-to-end network management solutions. `https://www.nvidia.com/en-us/networking/management-software/`, 2024.

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

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

[107] PCI-SIG. PCI Express® Base Specification Revision 4.0. `https://pcisig.com/specifications`, 2024.

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

[109] IEEE DCB. 802.1Qbb - Priority-based Flow Control. `https://1.ieee802.org/dcb/802-1qbb/`, 2021.

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

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

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

[113] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. Gentle Flow Control: Avoiding Deadlock in Lossless Networks. In *SIGCOMM*, 2019.

[114] 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.

[115] Linux rdma-core. `https://github.com/linux-rdma/rdma-core`, 2021.

[116] Waleed Reda, Marco Canini, Dejan Kostić, and Simon Peter. RDMA is Turing complete, we just did not know it yet!, 2021.

[117] Feng Ren, Mingxing Zhang, Kang Chen, Huaxia Xia, Zuoning Chen, and Yongwei Wu. Scaling up memory disaggregated applications with smart. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 351–367, 2024.

[118] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. ReDMArk: Bypassing RDMA Security Mechanisms. In *USENIX Security*, 2021.

[119] Sergej Schumilo, Cornelius Aschermann, Robert Gawlik, Sebastian Schinzel, and Thorsten Holz. kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels. In *USENIX Security*, 2017.

[120] Jiaxin Shi, Youyang Yao, Rong Chen, Haibo Chen, and Feifei Li. Fast and Concurrent RDF Queries with RDMA-Based Distributed Graph Exploration. In *OSDI*, 2016.

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

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

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

[124] 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.

[125] Tu Tran, Goutham Kalikrishna Reddy Kuncham, Bharath Ramesh, Shulei Xu, Hari Subramoni, Mustafa Abduljabbar, and Dhabaleswar K DK Panda. Ohio: Improving rdma network scalability in mpi_alltoall through optimized hierarchical and intra/inter-node communication overlap design. In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 47–56. IEEE, 2024.

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

[127] Shin-Yeh Tsai and Yiying Zhang. LITE Kernel RDMA Support for Datacenter Applications. In *SOSP*, 2017.

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

[129] VITA. FPGA Mezzanine Card Plus (FMC+) Standard. `https://www.vita.com/fmc`, 2024.

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

[131] 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.

[132] 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.

[133] 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.

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

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

[136] Jilong Xue, Youshan Miao, Cheng Chen, Ming Wu, Lintao Zhang, and Lidong Zhou. Fast Distributed Deep Learning over RDMA. In *EuroSys*, 2019.

[137] 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.

[138] 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.

[139] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. Guaranteeing deadlines for inter-datacenter transfers. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–14, 2015.

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

[141] Yinda Zhang, Peiqing Chen, and Zaoxing Liu. Octosketch: Enabling real-time, continuous network monitoring over multiple cores. In *NSDI 2024*. USENIX Association, 2024.

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

[143] Zhenwei Pi. Redis Over RDMA Implementation. `https://github.com/redis/redis/pull/11182`, 2022.

[144] 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 *SIGCOMM*, 2015.

Biography

Xinhao KONG is a Ph.D. candidate in the Department of Computer Science at Duke University, advised by Prof. Danyang Zhuo, since August 2021. His research interests include datacenter networks and high speed networks, with a focus on RDMA. He leads the project of RDMABench, a testing framework for RDMA/Infiniband software/hardware stack. The ambition of this project is to assist cloud providers and hardware vendors in building more reliable, efficient, and secure next generations of RDMA networks.

Before Duke, Xinhao received his bachelor's degree in Computer Science from Peking University in 2020.