PeRF: Preemption-enabled RDMA Framework

Sugi Lee and Mingyu Choi, *Acryl Inc.;* Ikjun Yeom, *Acryl Inc. and Sungkyunkwan University;* Younghoon Kim, *Sungkyunkwan University*

https://www.usenix.org/conference/atc24/presentation/lee

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Open access to the Proceedings of the 2024 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

PeRF: Preemption-enabled RDMA Framework

Sugi Lee[1], Mingyu Choi[1], Ikjun Yeom[1,2], and Younghoon Kim[2*]

[1]Acryl Inc.
[2]Sungkyunkwan University

Abstract

Remote Direct Memory Access (RDMA) provides high throughput, low latency, and minimal CPU usage for data-intensive applications. However, RDMA was initially designed for single-tenant use, and its application in a multi-tenant cloud environment poses challenges in terms of performance isolation, security, and scalability. This paper proposes a Preemption-enabled RDMA Framework (PeRF), which offers software-based performance isolation for efficient multi-tenancy in RDMA. PeRF leverages a novel RNIC preemption mechanism to dynamically control RDMA resource utilization for each tenant, while ensuring that RNICs remain busy, thereby enabling work conservation. PeRF outperforms existing approaches by achieving flexible performance isolation without compromising RDMA's bare-metal performance.

1 Introduction

Remote Direct Memory Access (RDMA) is a highly promising network technology that provides high throughput and ultra-low latency with minimal CPU usage via zero-copy read/write operations from/to applications' virtual memory spaces [12, 18, 34]. Various data-intensive applications, such as big-data analysis [4, 5], machine learning [2, 11, 15, 33], distributed storage [21, 27, 28, 41], and key-value stores [20, 25, 27, 38], have demonstrated significant performance improvements with RDMA. However, RDMA and its applications were originally designed for single-tenant use, which makes it challenging to deploy RDMA in multi-tenant cloud environments. According to recent studies, multiple tenants using RDMA can confront various problems such as security [19, 32, 39], scalability [16, 22, 36, 37], and performance isolation [22, 23, 40].

Among these problems, performance isolation is a particularly critical concern because, without adequate performance isolation, some tenants may fail to achieve the expected performance, thereby compromising the very purpose of utilizing RDMA. Several studies have been presented to provide RDMA performance isolation in multi-tenant environments. The basic policy of these studies is to regulate the data transmission rate of each tenant for efficient resource utilization, exploiting either hardware (HW)-based or software (SW)-based technologies. In [14, 19, 35], several RDMA virtualization frameworks have been proposed to leverage SR-IOV [6] or Virtual Lanes (VLs) [1] supported by RDMA HWs such as switches or RNICs. The advantage of these HW based regulation frameworks is that they can provide strict performance isolation, but they may not properly handle dynamic changes of tenants' requirements. SW-based solutions, meanwhile, focus on intercepting data transmission requests generated by applications to an RNIC, and controlling the request rate for each tenant in the user space [22, 30, 40]. They can achieve more flexible performance isolation against dynamic changes in network resource sharing compared to HW-based approaches. However, as they rely on reservation-based resource allocation to regulate transmission requests without help from HW, performance degradation is inevitable.

In this paper, we introduce a Preemption-enabled RDMA Framework (PeRF), a novel approach that offers SW-based performance isolation without compromising the bare-metal performance of RDMA. Previous SW-based solutions have attempted to regulate tenants' transmission request rates in order to control the packet scheduling of an RNIC. However, there is a significant limitation in these solutions such that inaccurate estimation of available network resources can lead to serious performance degradation. If available resources are underestimated, sufficient requests may not be issued to the RNIC, resulting in a loss of throughput and processing rate. Conversely, if overestimated, accumulated requests in the RNIC can break performance isolation.

In designing PeRF, our goal is to control the packet scheduling of an RNIC without requirement of accurate estimation of network resources. Unlike a reservation-based, non-work-conserving scheduling fashion, which demands accurate estimation of network resources, we employ a preemption-based, work-conserving scheduling approach in order to maximize

*Younghoon Kim is the corresponding author.

network utilization while providing performance isolation to multi-tenants. Our goal could be achieved using specific verbs (such as *IB_WR_WAIT* and *IB_WR_ENABLE* available in RDMA user-level APIs [8]), which can prompt an RNIC to temporarily pause packet processing for one connection, allowing another active connection to take the processing opportunity, similar to preemptive job scheduling in an OS. PeRF interleaves this preemption between requests generated by tenants and succeeds in elastically controlling the packet transmission of an RNIC. Moreover, by selectively preempting applications that use large messages or multiple connections, PeRF can be particularly beneficial for applications with small messages or a single connection. PeRF provides them with additional processing opportunities without impacting the performance of the applications currently occupying the communication channel by using large packets or multiple connections. Notably, our RNIC preemption mechanism may introduce some overhead to the CPU (for splitting large messages) and the RNIC (for managing larger request buffers). However, our investigation reveals that this overhead is minimal and does not adversely affect the performance of PeRF, as demonstrated in our extensive evaluation.

With the advantages of work-conserving scheduling with our novel preemption mechanism, PeRF properly isolates RDMA resource utilization of each tenant against dynamically changing network and tenant states, while maintaining high performance close to HW-based solutions. Furthermore, PeRF can be implemented entirely on existing RDMA APIs without requiring any specialized hardware or modifications to applications. Our evaluation demonstrates that PeRF delivers significantly higher throughput ($\sim$2.04$\times$) with similar latency compared to previous SW-based schemes. We also evaluate PeRF with real-world applications including Apache Crail [27], HERD [20], and rping [10]. Our framework transparently works on these applications, ensuring efficient performance isolation when they share an RNIC. This transparency also enables PeRF to complement other existing RDMA solutions, particularly those focused on security, scalability or network congestion control, enhancing its practicality in multi-tenant cloud environments.

2 Background and Motivation

2.1 RDMA Overview

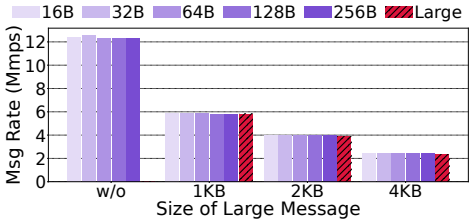
In RDMA, the entire network stack is offloaded on RNICs, and applications interact with them by using user-level RDMA APIs, IB Verbs [8]. To initiate data communication, an application creates a Queue Pair (QP) and a Completion Queue (CQ). The QP includes a Send Queue (SQ) and a Receive Queue (RQ). To send or receive a message, the application posts a Work Request (WR) to its QP (SQ or RQ) and rings a doorbell on an RNIC, prompting the RNIC to fetch the WR. This process fully offloads the message transmission to the RNIC, allowing the application to perform other tasks freely. Upon completing the message transmission, the RNIC pushes a Completion Queue Element (CQE) into the CQ, and then the application can recognize the completion of the message transmission by polling the CQE from its CQ.

RDMA supports two types of primitives: one-sided primitives, such as ***READ*** and ***WRITE*** operations, and two-sided primitives, such as ***SEND/RECV*** operations. All primitives are driven by the local application, while the remote application passively works. When a local application posts a WR for one-sided primitives, a local RNIC sends a message (data for WRITE or request for READ) that includes a remote virtual address and a memory key to a remote RNIC. This enables the remote RNIC to directly transmit data to or from the remote memory space without using the remote CPU. For two-sided primitives, a local application posts a SEND WR to a local RNIC, which then performs the SEND operation by transferring a message without any information related to the remote memory space. A remote RNIC processes the received message referring to a previously posted RECV WR to an RQ by a remote application. It is noted that performance advantages of RDMA are achieved mainly by the one-sided primitives, while the two-sided primitives more focus on flexibility.

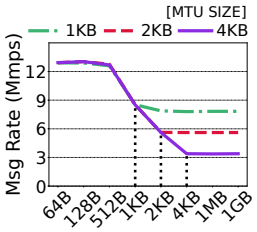
2.2 Micro-scale Analysis of RNIC

To understand how RNICs handle heterogeneous WRs posted to multiple QPs in parallel, we conduct micro-scale benchmark on a similar test-bed described in Section 5. The first benchmark aims to understand how an RNIC schedules multiple QPs. In this benchmark, each application uses its own single QP to send batches of messages with different sizes. Fig. 1(a) illustrates that, when five applications send small messages (< 1 KB), they achieve the RNIC's maximum message rate for a single QP. However, adding another application which sends larger messages (1, 2, or 4 KB) limits the message rates of other applications. This observation suggests that the RNIC employs a round-robin QP scheduling method, processing WRs from different QPs equitably, irrespective of their message sizes.

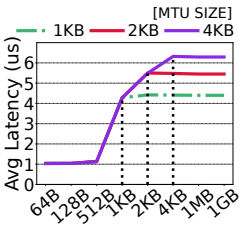
The second benchmark test investigates an RNIC's packet scheduling when transmitting messages larger than the MTU. Two applications share an RNIC, and each application uses a single QP: one sends messages in batches with sizes from 64 B to 1 GB as background traffic, while the other generates 16 B messages, and we measure the message rate (shown in Fig. 1(b)) and the latency (in Fig. 1(c)). To observe the impact of the MTU, we vary the MTU from 1 KB to 4 KB. As the size of the background messages increases from 64 B to the MTUs, the message rate and latency progressively worsen until the message size reaches the MTU. This occurs because the RNIC processes all packets with uniform priority, irrespective of their size. This makes a smaller packet wait for the transmission of a larger packet of the background traffic



(a) Round-Robin QP Scheduling of an RNIC



(b) Variation of Message Rate (by MTU-based Packet Scheduling)



(c) Variation of Avg. Latency (by MTU-based Packet Scheduling)

Figure 1: Micro-scale Analysis of RNIC's Scheduling

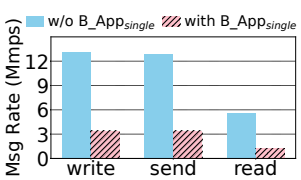
and causes extreme stalls for smaller ones. On the other hand, when the background message size surpasses the MTU, the rate of performance decline stabilizes. This stability arises as the RNIC, upon encountering packets larger than the MTU, segments these messages into packets of MTU size. Thus, the delay experienced by smaller packets does not exceed the time taken to deliver an MTU-sized packet.

Consequently, our observations reveal two distinct behaviors of RNICs during data transmission: (1) RNICs process WRs from multiple QPs with a round-robin scheduling; and (2) RNICs evenly schedule variously sized packets, which are segmented from messages originating from different QPs based on the MTU. Understanding these behaviors is crucial for developing optimized performance isolation schemes in RDMA, particularly in contexts where an RNIC is shared among multiple tenants.

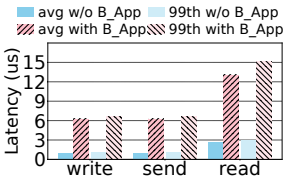
2.3 Challenges of Multi-tenancy in RDMA

In a multi-tenant cloud environment, system resources including processing power, storage and communication should be properly managed by the cloud administrator. Among these resources, however, RDMA networks are challenging for administrators to manage effectively due to their kernel-bypassing feature, as discussed in [23, 24, 40]. The straightforward and greedy packet scheduling mechanism of RDMA induces unfair opportunities for packet transmission among tenants, leading to performance anomalies and resource starvation. In this section, we analyze these anomalies in detail through micro-scale experiments.

In our experiments, we examine three types of RDMA applications classified by the message size and rate: delay-sensitive, message-intensive, and bandwidth-intensive appli-



(a) M_App_{single} vs. B_App_{single}



(b) D_App_{single} vs. B_App_{single}

Figure 2: Anomalies from Different Message Sizes

cations. Delay-sensitive applications (D_Apps) sparsely generate small messages ($\ll$ MTU) that require a short completion time. Message-intensive applications (M_Apps) generate batches of small messages to achieve a high message rate. Bandwidth-intensive applications (B_Apps) send large messages ($\gtrsim$ MTU) to achieve high throughput. Each type of applications is further divided into two subtypes based on whether it exploits multiple QPs or not. To sum up, we use six types of applications (B_App_{single}, B_App_{multi}, M_App_{single}, M_App_{multi}, D_App_{single}, and D_App_{multi}) in the experiments. The details of these applications are illustrated in Section 5.

2.3.1 Anomalies from Different Message Sizes

First, we look at how an application with large messages impacts on small message transmissions. As shown in Fig. 2(a), when an M_App_{single} shares an RNIC with a B_App_{single}, the message rate of the M_App_{single} is drastically reduced while the B_App_{single} achieves high throughput (94.77∼96.89 Gbps) for all operations. Similarly, in Fig. 2(b), a B_App_{single} can harm the performance of a D_App_{single} by increasing its average and the 99^{th} percentile (99%) tail MCT (Message Completion Time) while the B_App_{single} achieves 97.94 Gbps. We note that no performance anomaly occurs when an M_App_{single} and a D_App_{single} share an RNIC since their message sizes are similar. We also have confirmed a performance anomaly between two B_App_{single}s generating different sizes of messages (discussed in Appendix B).

2.3.2 Anomalies from Different QP Numbers

The QP-level round-robin scheduling of an RNIC may cause serious fairness anomalies by providing more packet-transmission opportunities to the multi-QP application in proportion to its number of QPs. Fig. 3(a) depicts the throughput comparison between a B_App_{single} and a B_App_{multi} when they compete for resources on the same RNIC. The throughput of the B_App_{single} decreases as the QP number of the B_App_{multi} increases. It has been also confirmed that the performance degradation of an M_App_{single} and a D_App_{single} is even more severe when they share the RNIC with a B_App_{multi}.

An M_App_{multi} can also degrade the performance of a D_App_{single} and an M_App_{single}. An RNIC has several Pro-

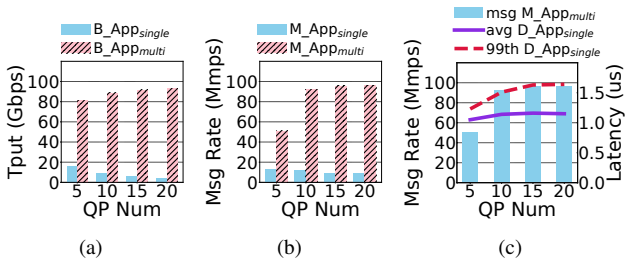


Figure 3: Anomalies from Different Numbers of QPs: (a) shows the throughput reduction of a B_App_{single} by a B_App_{multi}. (b) and (c) describe the performance degradation of an M_App_{single} and a D_App_{single} due to an M_App_{multi}.

cessing Units (PUs), and each PU independently processes WRs posted in a QP [24, 31]. This enables an RNIC to transmit packets of multiple QPs in parallel and to achieve its maximum message rate. For example, the RNIC in our experiments achieves approximately 105 Mmps with 9 QPs, while a single QP can achieve only about 13 Mmps at most. Fig. 3(b) and 3(c) illustrate that the message rate of an M_App_{multi} increases as the number of its QPs grows. However, when the RNIC reaches its maximum capacity, the M_App_{multi} begins to degrade performance for both an M_App_{single} and a D_App_{single}. We have conducted similar experiments with different RNICs and obtained similar results (illustrated in Section 5.2.5).

3 PeRF Design

3.1 Overview

PeRF, a Preemption-enabled RDMA Framework, is a SW-based performance isolation framework to rapidly adapt to dynamic network conditions and varying tenant demands without compromising resource utilization. Our analysis of RDMA in a multi-tenant environment (discussed in Section 2.3) highlights the necessity of prioritizing the performance of applications using small messages or a single QP over those monopolizing RNIC resources with large messages or multiple QPs. To address this, we establish three rules:

1. Isolate Small message transmission from large message transmission.
2. Isolate an application with a single QP from a bandwidth or message-intensive application with multiple QPs.
3. Ensure uninterrupted message transmission for delay-sensitive application.

Rules 1&2 enable PeRF to prevent the earlier-mentioned performance anomalies. To implement these rules efficiently, PeRF introduces a novel preemption mechanism for work-conserving performance isolation. Rule 3 necessitates segregated handling of delay-sensitive applications to ensure their

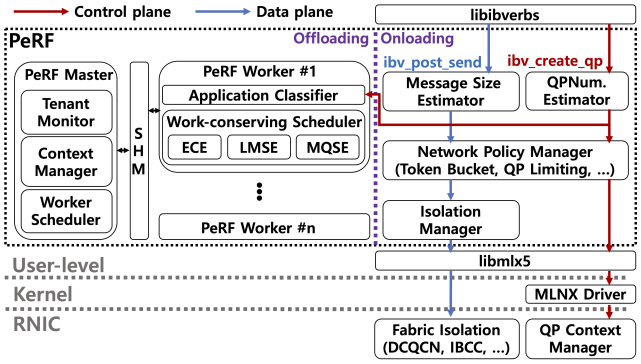


Figure 4: PeRF Architecture

ultra-low latency. The overall architecture of PeRF, designed to comply with these rules, is described in Fig. 4.

3.2 Splitting Isolation Process

Before delving into detailed design of PeRF, we describe how to minimize processing overhead for performance isolation. Implemented as an user-level library, PeRF requires the temporary blocking of an application during the execution of its performance isolation logic. In order to minimize the duration of this blocking, PeRF splits its process into two parts: onloading and offloading. The onloading process, executed on the main thread, specifically manages lightweight tasks, while more complex tasks are delegated to the offloading process running on a background thread.

The onloading process intercepts application's verb calls, and estimates the average message size and the number of created QPs. It also enforces network policies specified by the system administrator, including request rate limit or QP creation limit, for each tenant. PeRF's isolation manager in the onloading process dynamically triggers the isolation process depending on the application type.

The offloading process of PeRF involves two types of background threads: the PeRF master and PeRF workers. The PeRF master thread oversees the global states of tenants and orchestrates multiple worker threads to ensure optimal CPU utilization without contention. Whenever a tenant initiates a new RDMA application, PeRF spawns a new PeRF worker thread dedicated to manage the complex isolation procedures. The PeRF worker shares memory space with the main thread, and it can freely post WRs to the application's QP for PeRF's isolation process.

3.3 Design of PeRF Worker

Performance isolation of PeRF begins with identifying the requirements of applications executed by multiple tenants. The application classifier determines the application type based on the average message size and the number of created QPs estimated by the onloading part which intercepts user-level

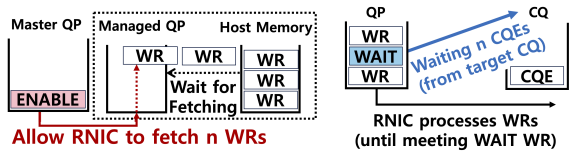
RDMA verbs (*ibv_create_qp* and *ibv_post_send*). If the average size is equal to or larger than 1 KB, the application is classified as a *B_App*. For applications with smaller messages, potentially classified as a *D_App* or an *M_App*, PeRF incorporates an additional classification step. It monitors the number of uncompleted WR in a SQ (*SQ_Len*) at 5 ms intervals, and an application is labeled as an *M_App* if the highest *SQ_Len* within the last 1,000 ms exceeds a specific threshold (Th_{M_App}), which we set to five in this study. If an application utilizes multiple QPs, scheduling messages of varying sizes and rates across different QPs can pose a challenge for PeRF in classifying the application type. In such cases, the application classifier considers the largest message size or the fastest rate among all QPs, and utilizes that value to categorize the application.

The work-conserving scheduler dynamically performs preemption processes to adhere to PeRF's isolation rules. These preemption processes utilize the PeRF preemption mechanism, incorporating PAUSE/RESUME operations on QPs and transmission interrupts. Detailed information about the mechanism are presented in the subsequent section. The scheduler is composed of three engines as outlined below:

- Large Message Scheduling Engine (LMSE): The LMSE is responsible for message-level isolation for small message transmission. It consistently interrupts large message transmissions via the PeRF preemption mechanism, thereby creating opportunities for the transmission of small messages.
- Multi-QP Scheduling Engine (MQSE): The MQSE provides QP-level isolation by managing Multi-QP applications. It employs PAUSE/RESUME operations within the PeRF preemption mechanism to regulate the number of QPs utilized by App_{multi}, and this facilitates performance isolation among applications with different numbers of QPs.
- Early Completion Engine (ECE): PeRF generates extra WRs within its preemption mechanism to enforce performance isolation. However, this may saturate the SQ for applications and potentially degrade their performance. The ECE is responsible for periodically polling these additional WRs, creating room for application WRs and mitigating potential performance issues.

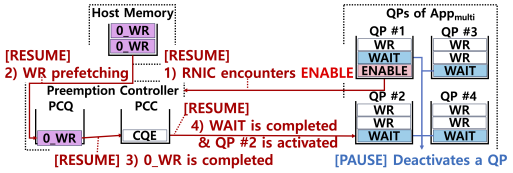
3.4 Selective Isolation Process in PeRF

Based on our extensive analysis, we have concluded that a *D_App* and an M_App_{single} have minimal impact on other applications even when utilizing multiple QPs concurrently. However, even a small overhead can result in significant performance degradation for them. To mitigate this issue, we have designed the PeRF isolation manager to allow WRs generated by a *D_App* and an M_App_{single} to bypass all isolation processes and be passed directly to the RNIC. For the case of an M_App_{multi}, the isolation manager either posts its WRs to the available QP or passes them to the offloading part to be posted later. All WRs of a *B_App* are offloaded.

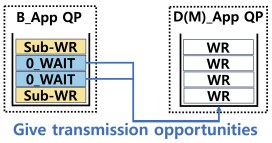


(a) Managed QP and ENABLE_WR (b) WAIT_WR

Figure 5: RNIC Preemption Mechanism of PeRF



(a) PAUSE/RESUME



(b) Transmission Interrupt

Figure 6: Performance Isolation of PeRF

For an application sending both large and small messages, PeRF permits bypassing the isolation process for small message requests if there is no ongoing transmission of a large message on a QP, regardless of the application's type. If a QP of any App_{multi} has a high message rate similar to *M_App*, PeRF applies the *M_App*'s isolation process to that QP. Together, these strategies allow PeRF to maintain ultra-low latency for delay-sensitive messages (whether for a *D_App* or not), while still providing performance isolation for many other RDMA applications.

3.5 PeRF Preemption Mechanism

We devise an innovative RNIC preemption mechanism that can temporarily pause or resume the RNIC resource utilization of each QP. This solution operates smoothly without the need for network resource estimation and without compromising the packet processing performance of the RNIC. As a result, PeRF efficiently isolates message transmissions for multiple tenants, offering a notable improvement compared to existing RDMA isolation solutions.

Our preemption mechanism uses a *managed* QP and specialized WRs (*ENABLE* and *WAIT*), which are provided by the user-level RDMA APIs [8]. When an application creates a QP in *managed* mode, any WR posted to the QP by the application is stored in host memory instead of being directly sent to the RNIC as shown in Fig. 5(a). To prompt the RNIC to fetch that WR, the application needs to issue an *ENABLE* WR

to another regular QP specifying the target *managed* QP's ID in the *ENABLE* WR. The *WAIT* WR posted to a QP causes the RNIC to pause processing subsequent WRs in the QP until it completes the *WAIT* WR as in Fig. 5(b). In the *WAIT* WR, the application must record the *wait_cqe_num* and *wait_cq*, indicating the number of CQEs to poll and the CQ from which to poll those CQEs. The *WAIT* WR can only be completed if the specified number of CQEs (*wait_cqe_num*) are polled from the *wait_cq*. Following the *WAIT* WR completion, the RNIC resumes processing subsequent WRs of the QP.

3.5.1 PAUSE/RESUME Operations for MQSE

PeRF initiates the process by creating an additional *managed* QP called Preemption Control QP (PCQ), and a corresponding CQ called Preemption Control CQ (PCC). Subsequently, PeRF posts a sufficient number of WRs in a host memory space to fill up the PCQ. These WRs (0_*WR*s) request zero-byte messages, containing only a header without a payload. The PCQ is configured to facilitate loopback communication. To manage individual QPs of an App_{multi}, PeRF utilizes these components in conjunction with *ENABLE* and *WAIT* WRs as shown in Fig.6(a). During the PAUSE operation, PeRF posts *WAIT* WRs to selected QPs of the App_{multi}, excluding $QNum_{allow}$ randomly chosen QPs. The *WAIT* WRs are configured to be completed when one CQE in the PCC is polled, and this ensures that the RNIC processes only the WRs from $QNum_{allow}$ of the non-paused (active) QPs. For the RESUME operation, PeRF posts an *ENABLE* WR to one of the active QPs to prompt the RNIC to resume processing WRs from one of the paused (non-active) QPs, specifically the oldest one. Notably, by posting a *WAIT* WR after the *ENABLE* WR, PeRF guarantees that the number of active QPs remains consistent with $QNum_{allow}$.

3.5.2 Transmission Interrupts for LMSE

Since an RNIC does not consider packet sizes during packet handling, it requires the interruption of large message transmissions to isolate small message transmissions. PeRF achieves this through transmission interrupts implemented via a PAUSE operation with a 0_*WAIT* WR. A 0_*WAIT* WR can be created by setting its *wait_cqe_num* to zero. When an RNIC encounters a 0_*WAIT* WR posted to a QP, it immediately completes it, halts message transmission of the QP, and proceeds to process WRs posted by another QP. Upon intercepting a WR requesting large message transmission, PeRF divides it into several sub-WRs, each requesting transmissions of equal-sized sub-messages (*SUB_MSG_SIZE*). Between these sub-WRs, PeRF inserts a few 0_*WAIT* WRs. This mechanism prevents the overuse of RNIC resources for transmitting large messages by *B_App*s and allocates reserved resources for transmitting small messages by *M_App*s or *D_App*s as in Fig.6(b).

4 PeRF Implementation

PeRF's implementation consists of approximately 400 lines of C++ code and 3,000 lines of C code added to the RDMA user-level driver (*libmlx*5) in the OpenFabrics Enterprise Distribution (OFED 4.9-4.1.7.0) [8]. Additionally, we utilize the *khash* library [3] to manage the context of multiple QPs created by an application. PeRF code is available via https://github.com/acryl-aaai/perf.

4.1 Large Message Scheduling Engine

Achieving message-level isolation between *B_App*s transmitting large messages and other applications such as *M_App*s and *D_App*s is facilitated by employing transmission interrupts. The allocation of RNIC resources to applications using small messages is determined by the number of 0_*WAIT* WRs interposed between sub-messages (0_*WAIT_NUM*). This value is calculated by dividing *SUB_MSG_SIZE* by 0_*WAIT_UNIT*. In this paper, we set the default values of *SUB_MSG_SIZE* and 0_*WAIT_UNIT* to be 16 KB and 1 KB, respectively. With these values, QPs with small messages have 16 times more transmission opportunities than a QP with large messages. Reducing 0_*WAIT_UNIT* results in more 0_*WAIT* WRs being inserted between sub-WRs, allowing for a greater allocation of RNIC resources to small message transmission. Note that large messages ($\leq$ *SUB_MSG_SIZE*) that do not need to be split are processed in the onloading part instead of being passed to the offloading part. This prevents throughput degradation due to communication overhead between the main and background threads. Although this may require the main thread to post several 0_*WAIT* WRs, we anticipate that it will have minimal impact on the application's performance because the number of 0_*WAIT* WRs will not be large.

4.2 Multi-QP Scheduling Engine

When an RNIC is shared between a multi-QP application (App_{multi}) and a single-QP application (App_{single}), PeRF initiates the QP-level isolation process. It can flexibly restricts the number of active QPs for each tenant with a combination of PAUSE and RESUME operations. If needed, it can activate the QPs of an App_{multi} one by one in a round-robin fashion by setting $QNum_{allow}$s for all tenants to one. This greatly helps PeRF isolate App_{multi}'s traffic from App_{single}'s traffic. However, it is also necessary for MQSE to support the maximum message rate for M_App_{multi}. M_App_{multi} needs parallel utilization of multiple QPs for that. Therefore, considering $QNum_{capa}$, which indicates the minimum number of QPs needed to fully utilize the RNIC's maximum message rate[1], PeRF adjusts $QNum_{allow}$ for each tenant starting from the default value, one.

[1]The $QNum_{capa}$ is a unique parameter to each RNIC.

When an application attempts to post WRs through multiple QPs surpassing $QNum_{allow}$, the MQSE oversees each QP, ensuring an equal opportunity for activation in a round-robin fashion. In cases where an application tries to post WRs to a non-active QP, these WRs are buffered in an MQSE queue rather than being directly forwarded to an RNIC until the QP becomes active. When a non-active QP becomes active, the MQSE initially posts the buffered WRs and subsequently permits the application to directly post WRs to the RNIC. This strategy enables PeRF to equitably manage both B_App_{multi} and M_App_{multi}, allowing them to utilize their multiple QPs without adversely affecting other applications.

4.3 Early Completion Engine

In RDMA, WRs are stored in an SQ until their corresponding CQEs are polled by an application. If the SQ becomes full, preventing the posting of additional WRs, the application must periodically poll CQEs to create space for future WRs. However, a PeRF worker posts sub-WRs, 0_*WAIT*, *WAIT*, or *ENABLE* WRs, which are transparent to an application and contribute to filling up the SQ. The ECE is responsible for polling CQEs corresponding to these transparent WRs. Since the ECE attempts to poll CQEs from the application's CQ, it may also poll normal CQEs that need to be passed to the application. Therefore, the ECE must store them in its buffer to deliver them to the application later.

4.4 Improving Practicality and Scalability

1) Minimization of Preemption Overhead: The PeRF preemption mechanism can result in overhead that affects PeRF's performance. To mitigate this, our scheme selectively activates the isolation process based on the global tenant distribution managed by the PeRF master. The process is triggered only when the problematic situation arises. For the message-level isolation, PeRF divides large messages into chunks larger than or equal to 16 KB and inserts batches of 0_*WAIT* WRs rather than posting one sub-WR for each small chunk alternated with a 0_*WAIT* WR. This approach significantly decreases the CPU overhead of the message-level isolation of PeRF (described in Section 5.2.1). Note that the QP-level isolation incurs minimal overhead, as only a few *WAIT* and *ENABLE* WRs are posted.

2) Scalable Management of Worker Threads: For CPU efficiency, we assign the offloading part to a single CPU core. To facilitate efficient communication between threads, we have meticulously designed a lock-free data structure in SHM. This data structure ensures that each element is updated exclusively by individual threads. Additionally, we leverage atomic operations to manage information shared between QPs for multi-threaded applications that utilize multiple threads for posting WRs to their QPs in parallel. To avoid CPU contention between PeRF workers, the PeRF master schedules the worker threads based on condition variables, enabling PeRF to support thousands of tenants and QPs in parallel without performance degradation as seen in Section 5.2.5.

3) Fabric Isolation: PeRF focuses on resolving issues for tenant-level RNIC resource sharing rather than flow-level fabric sharing. In PeRF, fabric isolation can be achieved using existing congestion control mechanisms implemented on commercial RNICs, such as DCQCN [42], TIMELY [29], and IBCC [7]. PeRF works well with DCQCN (for RoCEv2) or IBCC (for Infiniband), as confirmed in Section 5.2.4.

4) Support for Various RDMA Operations: We have mainly explained the design and implementation of PeRF, concentrating on the WRITE operation, but PeRF can support the SEND/RECV and READ operations as well. The SEND operation is similar to WRITE, and implementing the RECV operation is straightforward because PeRF's isolation process is driven by a sender. For the RECV operation, we add a mechanism that divides WRs for receiving large messages and early polls corresponding CQEs in a PeRF worker. We also develop a simple RPC-based READ operation similar to [20], allowing PeRF to manage bidirectional data transmission of the READ operation. The adoption of this RPC-based method is motivated by the unique challenge posed by READ operations, where the message generation is executed by a remote RNIC. In such situations, PeRF's isolation process must be carried out remotely. However, keeping the CPU consumption low on the remote RDMA application is a critical aspect in RDMA contexts. To address this, we employ the RPC mechanism to send READ requests to PeRF's offloading part on the remote side using the SEND/RECV operation. This enables the offloading part to efficiently handle the transfer of READ data using the WRITE operation and initiate the isolation process.[2]

5 Evaluation

5.1 Experiment Setup

Throughout the experiments, we set up a rack-scale cluster of five identical machines. Each machine has a 16-core Intel(R) Core(TM) i7-11700K 3.6GHz CPU and 32GB DRAM, connected via a Mellanox Open Ethernet 100 Gbps switch (SN-2100). We have conducted evaluations with various commercial RNICs described in Table 1, and ConnectX-6 for RoCEv2 is used as a default one unless specified otherwise.

Our experiments involve three types of applications: *B_App*, *M_App* and *D_App*. They post WRITE WRs with a single QP unless otherwise noted. Specifically, a *B_App* continuously sends 1 MB messages while an *M_App* and a *D_App* send 16 B messages. Only the *M_App* transmits 32 messages in batches. We utilize *perftest* [9], a widely used RDMA benchmarking tool. Each tenant is assigned with a

[2]Engines for exchanging READ requests are incorporated into the PeRF worker.

RNIC	Protocol	Speed	Num. of PUs [23]
ConnectX-5	RoCEv2	40Gbps	8
ConnectX-6	RoCEv2	100Gbps	16
ConnectX-6	Infiniband	100Gbps	16

Table 1: RNICs Specifications

Isolation Type	CPU	RAM
Message-level (*D_App* vs. *B_App*)	0.94%	8.8 MB
QP-level (*D_App* vs. B_App_{multi})	0.56%	12.2 MB
QP-level (*D_App* vs. M_App_{multi})	3.62%	14.4 MB

Table 2: The CPU and RAM Usage of PeRF

dedicated core pinned to their processes. For tenants with multiple QPs, however, we modify the application to support multi-threading and allocate threads in a way that minimizes sharing threads between QPs.

To demonstrate the efficiency and practicality of PeRF, we have conducted a comparison with Justitia [40]. We use Justitia's default chunk sizes and the token batch of 1,100 for mitigating an *M_App*'s overuse. The value is calculated following the literature, and it is sufficient to fully utilize one QP (about 13 Mmps). Justitia detects network congestion by measuring the 99% tail MCT of its monitoring packets. We also evaluate a relaxed version of Justitia, $Justitia_R$, with a more generous latency threshold of 15 μs [3] (compared to the default 2 μs). By increasing the threshold, $Justitia_R$ is less likely to perceive the network as congested, allowing more throughput to *B_App*s.

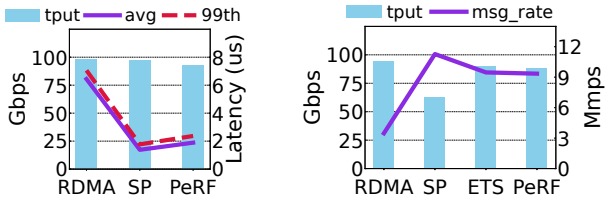
5.2 Baseline Benchmark

We analyze the overhead in the operation of PeRF and how effectively it isolates tenants in a rack-scale environment. We construct a single-hop network within a rack, generating traffic between two nodes. In the result plots below, we use *tput* to represent throughput, *avg* for average latency, and *99th* for 99^{th} percentile tail latency.

5.2.1 PeRF Overhead

This section presents experiments designed to evaluate the overhead of PeRF. The first experiment assesses the CPU and RAM usage involved in PeRF's Message-level and QP-level isolation processes. In this experiment, a *D_App* shares an RNIC with either a B_App_{single}, a B_App_{multi} and an M_App_{multi}, and we measure the corresponding resource usages. Given that PeRF's offloading part is configured to operate on a single core, as mentioned earlier, Table 2 illustrates the additional resource usage by applications when executing

[3]We have identified the optimal threshold value that allows Justitia to maintain an acceptable level of low latency (average of 1.5 μs) while achieving a much higher throughput (86 Gbps) in our test-bed.



(a) *B_App* vs. *D_App* (b) *B_App* vs. *M_App*

Figure 7: Evaluation of HW-based Solutions and PeRF

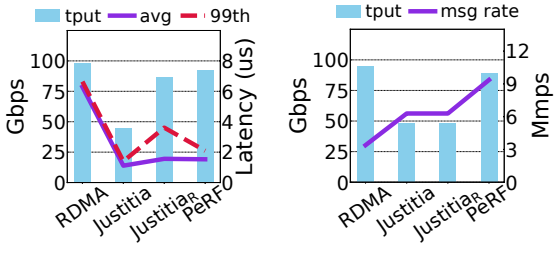
PeRF's onloading part. Notably, isolating M_App_{multi} necessitates slightly more CPU usage due to the high number of WRs for batched small message transmission, resulting in more frequent calls to PeRF's isolation mechanism. However, all isolation processes, including those for M_App_{multi}, add less than 5% CPU usage. It implies that PeRF achieves its performance isolation benefits almost at the cost of only a single CPU. Regarding RAM usage, approximately 9 MB is required by PeRF to isolate an App_{single}. However, for an App_{multi}, each additional QP in the isolation process necessitates about an extra 500 KB of memory.

Subsequently, we compare PeRF's overhead with HW-based schemes. Fig.7(a) presents the throughput of a *B_App* and the average and tail latency of a *D_App* when they share an RNIC. Strict Policy (SP) in the figure is a HW-based isolation scheme that strictly prioritizes small messages using priority queues in the RNIC. PeRF achieves comparable throughput and latencies with SP. Additionally, PeRF's efficacy is further evaluated with a *B_App* and an *M_App* in Fig.7(b). In this experiment, SP's throughput is severely degraded since the *M_App* dominates the RNIC with prioritized small messages. Thus, we compare with another HW-based scheme, Enhanced Transmission Selection (ETS) [13], which schedules message transmissions from both apps based on weighted round-robin (WRR) queues. For ETS, we configure two priorities, one each for a *B_App* and an *M_App*, with the weights set at 16 and 1, respectively. This configuration allows ETS to deliver efficient performance, ensuring high throughput for the *B_App* and a high message rate for the *M_App*. Remarkably, PeRF demonstrates performance similar to that of ETS.

During our evaluation, it was observed that ETS is less effective when a *B_App* and a *D_App* coexist.[4] This limitation arises from the inherent nature of WRR-based QoS schemes, like ETS, which schedule data transmissions primarily based on bandwidth utilization. While these schemes effectively manage applications requiring substantial bandwidth, such as *B_App* and *M_App*, they struggle to adequately isolate a *D_App*, which requires little (almost zero) bandwidth, from other RDMA applications.

To sum up, the evaluations in this section underscore

[4]We omit the graph for this experiment due to the space limit of the paper.



(a) *B_App* vs. *D_App* (b) *B_App* vs. *M_App*

Figure 8: Message-level Isolation of Various Schemes

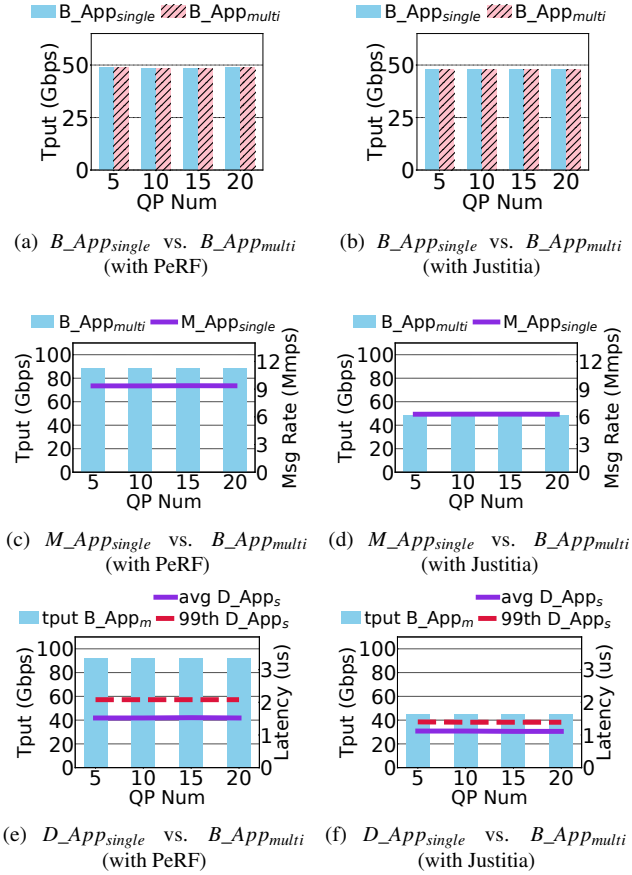
PeRF's proficiency in utilizing its SW-based preemption mechanism to dynamically manage data transmissions across various tenants. This mechanism enables the RNIC to differentially schedule packets with negligible overhead, on par with HW-based schemes, thereby ensuring optimal performance for various RDMA applications sharing an RNIC.

5.2.2 Message-level Isolation

In this section, we evaluate PeRF's message-level isolation between two tenants using a single QP. Native RDMA (*RDMA*) is used as a base case, and the performance is compared with Justitia and Justitia$_R$. As shown in Fig. 8, *RDMA* fails to control a *B_App*, which leads to higher latency of a *D_App* and lower message rate of an *M_App*.

In contrast, Justitia aims to ensure a fair distribution of RNIC resources among multiple tenants. To achieve this, Justitia calculates an equitable available bandwidth and message rate for each tenant and adjusts their WR posting rate not so as to exceed these defined limits. When a *B_App* operates alongside a *D_App* and an increase in the tail latencies of the *D_App* is observed, Justitia halves the WR posting rate of the *B_App* to promote fair bandwidth sharing, as illustrated in Fig.8(a). Similarly, when an *M_App* and a *B_App* share an RNIC, Justitia reduces the WR posting rates of both applications, considering the calculated fair bandwidth and message rate for each. This adjustment leads to an approximate 50% decrease in performance for each application, as demonstrated in Fig.8(b). Justitia$_R$, an adjusted version of Justitia, is less sensitive to the increase in tail latencies of the *D_App* due to a higher threshold setting, allowing for higher throughput for the *B_App* while maintaining similar performance to Justitia in scenarios where the *M_App* and the *B_App* coexist.

Our analysis indicates that while Justitia effectively allocates RNIC resources equitably, it introduces significant overhead by pausing WR postings of RDMA applications. This pause prevents the RNIC from continuously transmitting packets, leading to performance degradation. In contrast, our proposed scheme, PeRF, utilizes the novel preemption mechanism that allows RDMA applications to post WRs without interruption, enabling the RNIC to perform work-conserving scheduling for packet transmission. As a result, PeRF achieves



(a) B_App_{single} vs. B_App_{multi} (with PeRF) (b) B_App_{single} vs. B_App_{multi} (with Justitia)

(c) M_App_{single} vs. B_App_{multi} (with PeRF) (d) M_App_{single} vs. B_App_{multi} (with Justitia)

(e) D_App_{single} vs. B_App_{multi} (with PeRF) (f) D_App_{single} vs. B_App_{multi} (with Justitia)

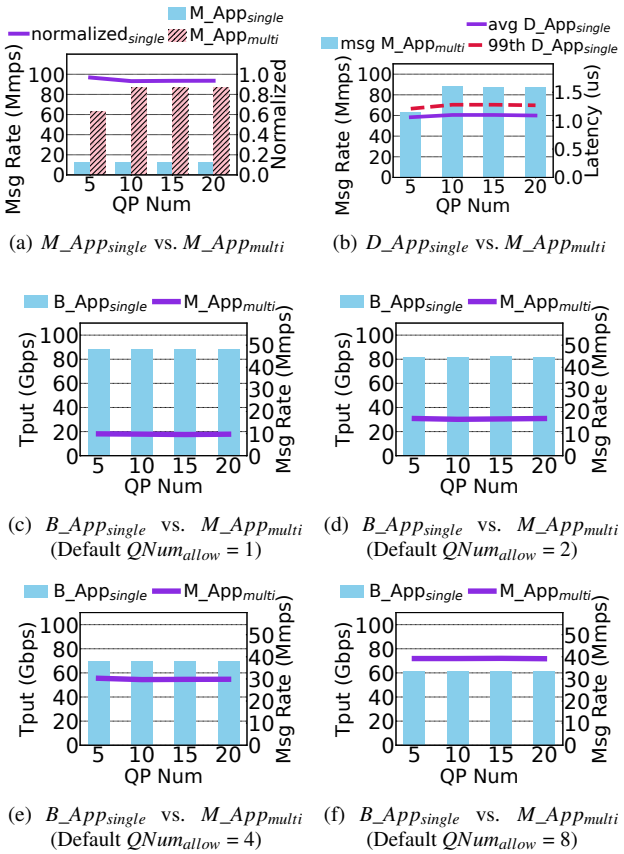
Figure 9: QP-level Isolation for B_App_{multi}

high throughput for a *B_App*, while maintaining low latency and high message rate for a *D_App* and an *M_App*, respectively, in our evaluations.

5.2.3 QP-level Isolation

We also investigate how PeRF addresses performance anomalies resulting from an App_{multi} in this section. A B_App_{multi} and an M_App_{multi} have distinct sources of performance anomalies, such as message sizes and the number of WQEs. Given that, we design two different sets of experiments where a B_App_{multi} (or an M_App_{multi}) coexist with a B_App_{single}, an M_App_{single} and a D_App_{single}, respectively.

***B_App$_{multi}$* cases:** PeRF successfully isolates tenants with different numbers of QPs even when a *B_App* utilizes multiple QPs, as depicted in Fig. 9. It maintains the throughput of *B_App*s regardless of the numbers of QPs while preserving low latencies of *D_App*s and high message rates of *M_App*s. On the other hand, as shown in Fig. 9(d), even though Justitia issues equal amounts of tokens to a B_App_{multi} and an M_App_{single}, it achieves only half of throughput and message rate due to its working mechanism. As shown in Fig. 9(e) and (f), PeRF exhibits a marginally higher latency for D_App_{single} compared to Justitia. However, this increased latency is a rea-



(a) M_App_{single} vs. M_App_{multi}

(b) D_App_{single} vs. M_App_{multi}

(c) B_App_{single} vs. M_App_{multi} (Default $QNum_{allow}$ = 1)

(d) B_App_{single} vs. M_App_{multi} (Default $QNum_{allow}$ = 2)

(e) B_App_{single} vs. M_App_{multi} (Default $QNum_{allow}$ = 4)

(f) B_App_{single} vs. M_App_{multi} (Default $QNum_{allow}$ = 8)

Figure 10: PeRF's QP-level Isolation for M_App_{multi}

sonable trade-off for the benefits of work-conserving scheduling.

***M_App_{multi}* cases:** Fig. 10(a) shows the message rates of an M_App_{single} and an M_App_{multi}, and the normalized rate of the M_App_{single}, which is normalized by the maximum achievable message rate with a single QP. In Fig. 10(b), the average and 99% tail MCT of a D_App_{single} are described with M_App_{multi}'s message rate. In both scenarios, by adjusting a $QNum_{allow}$ for each application, PeRF is successful in segregating the M_App_{multi} and achieves up to 35% higher message rate for the M_App_{single} and up to 22% lower 99% tail latency (5% for the average) for the D_App_{single} compared to *RDMA* (as described in Fig. 3(b) and 3(c)).

We have observed that an M_App_{multi} also degrades the performance of a B_App_{single} in *RDMA* by taking more chances to transmit its small messages. Even in this case, PeRF maintains a consistent throughput of 88 Gbps for the B_App and shows the message rate of 9 Mmps regardless of the number of QPs in the M_App_{multi} (Fig. 10(c)). We would like to highlight that PeRF perceives this situation as network resources being fully utilized, and limits the number of QPs used by an M_App_{multi} to a default $QNum_{allow}$ (statically set to one). It can achieve a comparable message rate by simply changing the default $QNum_{allow}$ as shown in Fig. 10(d), (e) and (f). Note that we do not evaluate Justitia for these cases since it does not consider performance anomalies caused by an M_App_{multi}.

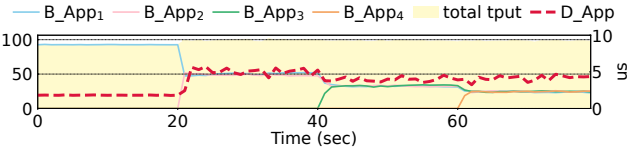


Figure 11: Evaluation of PeRF in Congested Networks

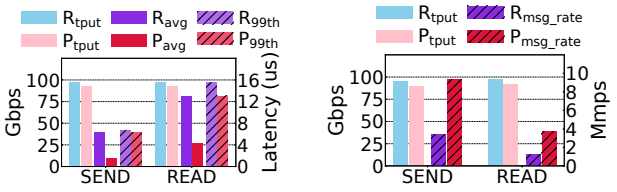
5.2.4 PeRF in Congested Networks

PeRF primarily focuses on the tenant-level RNIC resource allocation while leveraging existing congestion control mechanisms, as referenced in [7, 26, 29, 42], implemented on commercial RNICs for flow-level network link sharing (or fabric isolation). This unique approach allows PeRF to dynamically respond to network changes without the need for additional network state estimation. To assess PeRF's compatibility with DCQCN, one of the most widely deployed congestion control mechanism, we set up a 2-tier topology comprising three 100 GbE switches and two racks, each containing three nodes. Rack-1 includes Node-A, Node-B, and Node-C, while Rack-2 houses Node-a, Node-b, and Node-c. We execute a B_App on each node in Rack-1, transmitting 1 MB WRITE messages to their corresponding nodes in Rack-2. These B_Apps run sequentially with a 20-second interval. Concurrently, a D_App runs on Node-A from the outset to evaluate PeRF's performance isolation in congested networks.

The evaluation results, depicted in Fig. 11, illustrate PeRF's compatibility with DCQCN. PeRF effectively enables DCQCN to equitably distribute network bandwidth among the B_Apps while maintaining low latency for the D_App. Notably, with native RDMA, the MCT for the D_App increased by approximately 10 microseconds, but the corresponding graph is excluded in this paper due to space constraints. We also observed an increase in the MCT after the initiation of additional B_Apps, a result of congestion from the traffic of two B_Apps, leading to heightened network queuing delay. However, minimizing this delay is primarily the responsibility of congestion control mechanisms, and thus falls outside the scope of this paper. Based on these results, we conclude that PeRF's tenant-level performance isolation seamlessly integrates with existing fabric isolation schemes found in various commercial RNICs, making PeRF a promising solution for RDMA-based cloud environments.

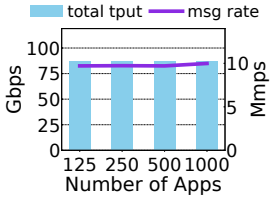
5.2.5 Practicality of PeRF

1) Support for other operations: PeRF addresses performance anomalies for SEND and READ operations as well. We demonstrate this by comparing PeRF's performance to that of *RDMA* when running a B_App alongside a D_App or



(a) *B_App* vs. *D_App* (b) *B_App* vs. *M_App*

Figure 12: Support for SEND and READ Operations



Num Apps (Num QPs)	Tput stdev (σ)
125 (1000)	0.01093
250 (2000)	0.00676
500 (4000)	0.00538
1000 (8000)	0.00537

(a) Scalability of PeRF (b) Fair Share of Bandwidth with PeRF

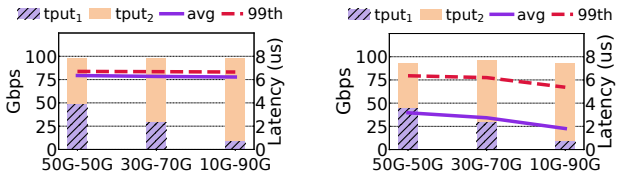
Figure 13: Evaluation of PeRF's Scalability

an *M_App* for both operations, as shown in Fig.12. As depicted in Fig.12(a), PeRF (marked as 'P' in the figure) compromises only 5% of *B_App*'s throughput compared to *RDMA* (marked as 'R' in the figure), while reducing the average latency of a *D_App* by 75% for SEND and by 67% for READ. Similarly, PeRF's message rate of an *M_App* reached 2.7× for SEND and 3× for READ compared to *RDMA* without sacrificing *B_App*'s throughput, as illustrated in Fig. 12(b). It is worth noting that these results are comparable to performance without the *B_App*, as shown in Fig. 2.

2) Scalability of PeRF: In order to evaluate PeRF's scalability, we gradually increase the number of *B_App*s with 8 QPs up to 1,000 while measuring the message rate of a single *M_App* in a single hop environment. As shown in Fig. 13(a), the message rate of the *M_App* remains at 9.7 Mmps, while *B_App*s achieves a throughput of 87 Gbps. The standard deviation of the throughput between *B_App*s is very low (Fig. 13(b)), indicating that the *B_App*s are perfectly isolated and fairly share the resources.

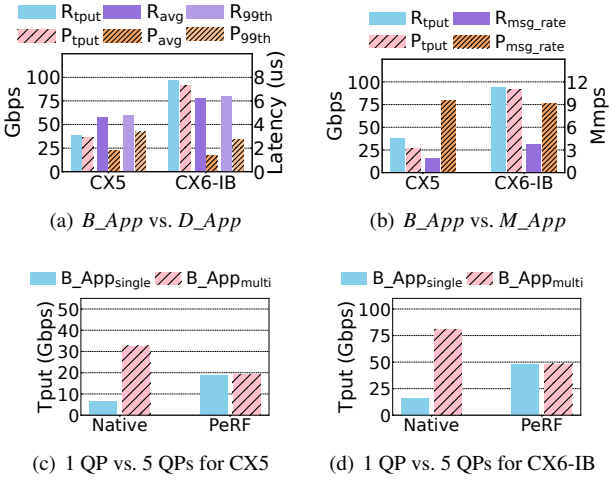
3) Support for Differentiation: In a cloud environment, it is sometimes necessary to adopt a weighted policy that differentiates resource allocation between tenants. We run two *B_App*s under a weighted policy based on the static token bucket (used in Freeflow [22]) and add a single *D_App*. Both *RDMA* and PeRF distribute throughput proportional to the weights. *RDMA* achieves the maximum throughput but worsens the overall latency of the *D_App* (Fig.14(a)). On the other hand, PeRF reduces the average latency by up to 70% compared to *RDMA* (Fig. 14(b)). PeRF's latency decreases as it gets close to the 10G-90G range, because the *B_App* with lower weight has fewer chances to request to the RNIC, opening up opportunities for the *D_App*.

4) Support for Other RNICs: We conduct an investigation into the anomalies that other RNICs in Table 1 encounter



(a) Native RDMA (b) PeRF

Figure 14: Weighted Policy Based on Token Bucket



(a) *B_App* vs. *D_App* (b) *B_App* vs. *M_App*

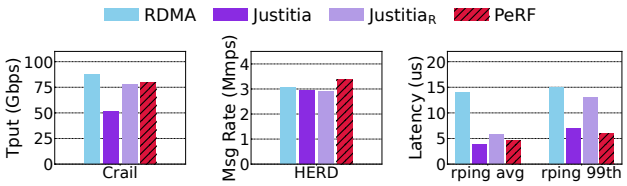
(c) 1 QP vs. 5 QPs for CX5 (d) 1 QP vs. 5 QPs for CX6-IB

Figure 15: Evaluation with Other Commercial RNICs

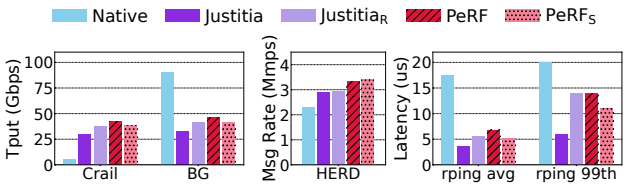
and evaluate how PeRF handles them. Our experiments, as shown in Fig.15, revealed that both the ConnectX-5 40GbE and ConnectX-6 100G Infiniband RNICs exhibit similar performance anomalies to those of CX6. PeRF significantly improves the performance of both a *D_App* and an *M_App* when coexisting with a *B_App*. Specifically, PeRF reduces the *D_App*'s average latency by up to 80% as shown in Fig.15(a) and increases the *M_App*'s message rate up to 5× as shown in Fig.15(b). Additionally, We evaluate the performance of a B_App_{single} and a B_App_{multi} using five QPs when they share both RNICs, and PeRF provides both applications with an equal share of the link bandwidth, as in Fig. 15(c) and 15(d).

5.3 Real-World Applications with PeRF

We apply PeRF in a practical scenario with three real-world applications: Apache Crail [27], HERD [20], and rping [10]. Crail is a distributed storage system sending large messages (*B_App*). HERD is a key-value store where clients send requests in batches (*M_App*). rping is a basic ping application for RDMA (*D_App*). In our experiment, we use four physical machines: three separate server nodes (Crail datanode, HERD server and rping server) and one client node with all client applications (including Crail namenode). Specifically, the Crail client continuously writes 1 GB files (to be divided into 1 MB chunks) to the Crail datanode. We configure HERD to create



(a) Without Background Traffic



(b) With Background Traffic

Figure 16: Evaluation of PeRF with Real-World Application

one client thread and densely send a mix of 5% PUT and 95% GET requests to eight workers. For rping, we simply ping from the client to the server.

In the first scenario, we execute Crail, HERD and rping simultaneously. Fig. 16(a) demonstrates that PeRF improves the throughput of Crail by 55% compared to Justitia, while compromising less than 1μs in rping latencies. The throughput of Justitia$_R$'s Crail is close to PeRF's, but PeRF still has lower average latency and 99% tail MCT than Justitia$_R$. PeRF also outperforms both Justitia and Justitia$_R$ in the message rate of HERD.

Next, in the same environment above, we generate background traffic from the client node to an additional node, transmitting 1 MB messages with 16 QPs. Fig. 16(b) demonstrates the impact of background traffic. Compared to Justitia and Justitia$_R$, PeRF not only increases the message rate by 14% but earns higher throughput in both Crail and background flow, utilizing 10∼30% more bandwidth in total. However, Justitia$_R$'s rping gets better latency than PeRF's. Thus, we also evaluate the small-message-friendly version of PeRF(PeRF$_S$) whose 0_*WAIT_UNIT* is set to 900, allowing more processing opportunities for small messages in RNIC. Consequently, PeRF$_S$ reduces its latency below Justitia$_R$'s while maintaining higher throughput and message rate. Notably, certain parameters in PeRF, including 0_*WAIT_UNIT* and default $QNum_{allow}$, are designed as configurable options. This allows cloud providers the flexibility to adjust these parameters to fit their specific operational strategies and requirements. Details of such parameters are illustrated in Appendix A.

6 Related Works

Various studies have been conducted to improve the efficiency of RDMA data transmission in multi-tenant cloud environments. Collie [24] introduces an automatic tool for detecting performance anomalies due to varying application workloads. Husky [23] goes further by analyzing RDMA performance issues related to RNIC's micro-architecture, such as PU and cache, and highlights a new anomaly type associated with atomic operations in RDMA. While PeRF's specific response to this anomaly isn't detailed here, its preemption mechanism is potentially adaptable for isolating such operations.

In addressing RDMA anomalies, HW or SW-based virtualization solutions have been proposed. Frameworks utilizing SR-IOV [6], as mentioned in [14, 17, 19], allocate Virtual Functions (VFs) to manage packet rates, while LITE [35] introduces a Linux kernel indirection layer, employing Virtual Lanes (VLs) [1] for data plane separation. These HW-based solutions offer effective isolation without compromising RDMA performance but necessitate static pre-configuration, which may not only lead to inefficiencies in dynamic cloud environments but also result in performance anomalies among tenants sharing the same group, such as VFs or VLs.

To overcome these limitations, SW-based solutions like Freeflow [22] and Justitia [40] have been developed. Freeflow uses a token bucket for WR rate control, though its static nature limits flexibility. Justitia, on the other hand, categorizes RDMA applications into three types and uses an adaptive token bucket to manage WR post rates, ensuring low latency for delay-sensitive applications. However, both these SW solutions face the risk of underutilizing network bandwidth or RNIC resources. This underutilization stems not only from potential inaccuracies in bandwidth estimation but also from their tendency to induce non-work-conserving packet scheduling in the RNIC. In contrast, PeRF overcomes these issues by incorporating an innovative preemption mechanism. This approach ensures efficient and work-conserving packet scheduling, thereby optimizing the use of RNIC and network resources.

7 Conclusion

In this paper, we introduce PeRF that offers software-based performance isolation while maintaining the high performance inherent to RDMA. Through micro-scale analysis, we have identified the RNIC's packet/QP scheduling mechanism and observed how various traffic types from applications can lead to significant performance anomalies. PeRF addresses these challenges by utilizing an RNIC preemption mechanism, built upon RDMA user-level verbs, and implementing work-conserving packet/QP scheduling. As a result, PeRF effectively delivers performance isolation without compromising RNIC performance in a range of experimental setups.

Acknowledgments

This work was supported in part by Institute for Information & communications Technology Promotion(IITP) and the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIP). (IITP-RS-2021-II210875 and NRF-2021R1C1C1009778)

References

[1] Infiniband trade association. infiniband architecture specification volume 1. https://cw.infinibandta.org/document/dl/7859, 2015.

[2] CNTK. https://github.com/Microsoft/CNTK/wiki, 2018.

[3] Klib. https://github.com/attractivechaos/klib, 2018.

[4] RDMA-based apache hadoop. http://hibd.cse.ohio-state.edu/, 2018.

[5] RDMA-based apache spark. http://hibd.cse.ohio-state.edu/, 2018.

[6] PCI special interest group. https://pcisig.com, 2021.

[7] Infiniband architecture specification. https://www.infinibandta.org/ibta-specification, 2022.

[8] MLNX_OFED drivers. https://network.nvidia.com/products/infiniband-rivers/linux/mlnx_ofed, 2023.

[9] perftest. https://github.com/linux-rdma/perftest, 2023.

[10] rping. https://github.com/linux-rdma/rdma-core/tree/master/librdmacm, 2023.

[11] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. OSDI'16, page 265–283, USA, 2016. USENIX Association.

[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] Motti Beck and Michael Kagan. Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure. In *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*, pages 9–15, 2011.

[14] Davda Bhavesh and Josh Simons. Rdma on vsphere: Update and future directions. *In Open Fabrics Workshop*, 2012.

[15] Rajarshi Biswas, Xiaoyi Lu, and Dhabaleswar K. Panda. Accelerating tensorflow with adaptive rdma-based grpc. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 2–11, 2018.

[16] Youmin Chen, Youyou Lu, and Jiwu Shu. Scalable rdma rpc on reliable connection with efficient resource sharing. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–14, 2019.

[17] 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 *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, April 2018. USENIX Association.

[18] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. When cloud storage meets rdma. In *NSDI*, pages 519–533, 2021.

[19] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. Masq: Rdma for virtual private cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Anuj Kalia, Michael Kaminsky, and David G Andersen. Design guidelines for high performance rdma systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, 2016.

[21] Daehyeok Kim, Amirsaman Memaripour, Anirudh Badam, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Shachar Raindel, Steven Swanson, Vyas Sekar, and Srinivasan Seshan. Hyperloop: Group-based nic-offloading to accelerate replicated transactions in multi-tenant storage systems. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 297–312, New York, NY, USA, 2018. Association for Computing Machinery.

[22] 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 *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 113–126, Boston, MA, February 2019. USENIX Association.

[23] 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, Boston, MA, April 2023. USENIX Association.

[24] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding performance anomalies in RDMA subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 287–305, Renton, WA, April 2022. USENIX Association.

[25] Pengfei Li, Yu Hua, Pengfei Zuo, Zhangyu Chen, and Jiajie Sheng. Rolex: A scalable rdma-oriented learned key-value store for disaggregated memory systems. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 99–114, 2023.

[26] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Bing Liu, Fang Liu, Nong Xiao, and Zhiguang Chen. Accelerating spark shuffle with rdma. In *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–7. IEEE, 2018.

[28] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an RDMA-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 773–785, Santa Clara, CA, July 2017. USENIX Association.

[29] 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 *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 537–550, New York, NY, USA, 2015. Association for Computing Machinery.

[30] Jonas Pfefferle, Patrick Stuedi, Animesh Trivedi, Bernard Metzler, Ionnis Koltsidas, and Thomas R. Gross. A hybrid i/o virtualization framework for rdma-capable network interfaces. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '15, page 17–30, New York, NY, USA, 2015. Association for Computing Machinery.

[31] Waleed Reda, Marco Canini, Dejan Kostić, and Simon Peter. RDMA is turing complete, we just did not know it yet! In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 71–85, Renton, WA, April 2022. USENIX Association.

[32] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F Wenisch, Monica Wong-Chan, Sean Clark, Milo MK Martin, Moray McLaren, Prashant Chandra, Rob Cauble, et al. 1rma: Re-envisioning remote memory access for multi-tenant datacenters. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 708–721, 2020.

[33] Feng Tian, Yang Zhang, Wei Ye, Cheng Jin, Ziyan Wu, and Zhi-Li Zhang. Accelerating distributed deep learning using multi-path rdma in data center networks. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, SOSR '21, page 88–100, New York, NY, USA, 2021. Association for Computing Machinery.

[34] Animesh Trivedi, Bernard Metzler, and Patrick Stuedi. A case for rdma in clouds: Turning supercomputer networking into commodity. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, New York, NY, USA, 2011. Association for Computing Machinery.

[35] Shin-Yeh Tsai and Yiying Zhang. Lite kernel rdma support for datacenter applications. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 306–324, New York, NY, USA, 2017. Association for Computing Machinery.

[36] Xizheng Wang, Guo Chen, Xijin Yin, Huichen Dai, Bojie Li, Binzhang Fu, and Kun Tan. Star: Breaking the scalability limit for rdma. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

[37] 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 *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1–14, Boston, MA, April 2023. USENIX Association.

[38] Xingda Wei, Rong Chen, and Haibo Chen. Fast rdma-based ordered key-value store using remote learned cache. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 117–135, 2020.

[39] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. Bedrock: Programmable network support for secure RDMA systems. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2585–2600, Boston, MA, August 2022. USENIX Association.

[40] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. Justitia: Software Multi-Tenancy in hardware Kernel-Bypass networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1307–1326, Renton, WA, April 2022. USENIX Association.

[41] Bohong Zhu, Youmin Chen, Qing Wang, Youyou Lu, and Jiwu Shu. Octopus+: An rdma-enabled distributed persistent memory file system. *ACM Trans. Storage*, 17(3), aug 2021.

[42] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery.

APPENDIX

We note that configurations used for evaluations in the appendices are the same as those presented in Section 5, unless otherwise stated.

A Configurable Parameters of PeRF

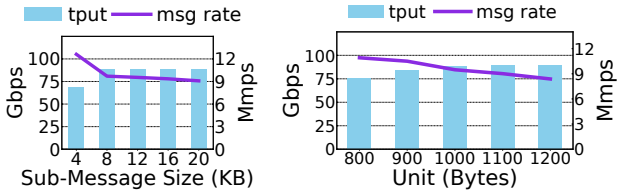
PeRF provides message-level isolation by 1) splitting WRs for large messages into multiple sub-WRs requesting *SUB_MSG_SIZE* of messages and 2) inserting 0*_WAIT_NUM* of 0*_WAIT* WRs between the sub-WRs. 0*_WAIT_NUM* is calculated by dividing *SUB_MSG_SIZE* by 0*_WAIT_UNIT*. In this paper, we set *SUB_MSG_SIZE* and 0*_WAIT_UNIT* to 16 KB and 1 KB by default, respectively. However, the performance and CPU efficiency of PeRF's message isolation are highly dependent on these parameters. Therefore, PeRF allows cloud providers to configure these parameters flexibly to achieve their own policies and objectives. To assist cloud providers in making informed decisions, this section presents several evaluations that illustrate the trade-offs between those parameters.

Impact of *SUB_MSG_SIZE*: In Fig. 17(a), we execute both a *B_App* and an *M_App* under various *SUB_MSG_SIZE*s. In this evaluation, we have observed that PeRF's message isolation has minimal impact on the throughput and message processing rate when *SUB_MSG_SIZE* is larger than 8KB, thanks to the improved CPU efficiency achieved by using larger sub-messages. On the other hand, too large *SUB_MSG_SIZE* can harm the performance of applications using small messages. As *SUB_MSG_SIZE* increases, a larger number of 0*_WAIT* WRs are generated at longer intervals, which may cause oscillations for *M_App*'s message rate or *D_App*' latency, leading to degradation of their stability and reliability.

Impact of *0_WAIT_UNIT*: Fig. 17(b) illustrates the performance of a *B_App* and an *M_App* with different 0*_WAIT_UNIT*s. A larger 0*_WAIT_UNIT* value indicates fewer 0*_WAIT* WRs issued between each sub-message, which leads to PeRF allocating more resources to the *B_App* while reducing the message rate of the *M_App*. As a result, we recommend cloud providers use a smaller 0*_WAIT_UNIT* if they want to prioritize message rate or latency over link utilization, otherwise, use a larger one.

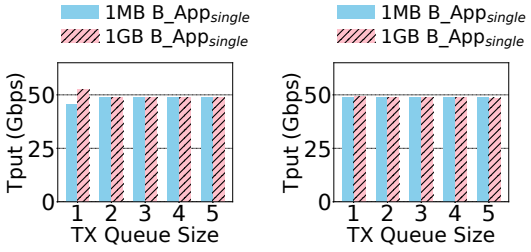
B Performance Anomaly Between Bandwidth-intensive Applications

In this paper, we extensively analyze performance anomalies that arise when different types of RDMA applications are in conflict. Additionally, we also have confirmed the existence of a performance anomaly between the same types of applications, specifically B_App_{single}s, when their message sizes



(a) *SUB_MSG_SIZE* (b) 0*_WAIT_UNIT*

Figure 17: *SUB_MSG_SIZE* and 0*_WAIT_UNIT*: PeRF parameters involve tradeoffs between the performance of different applications.

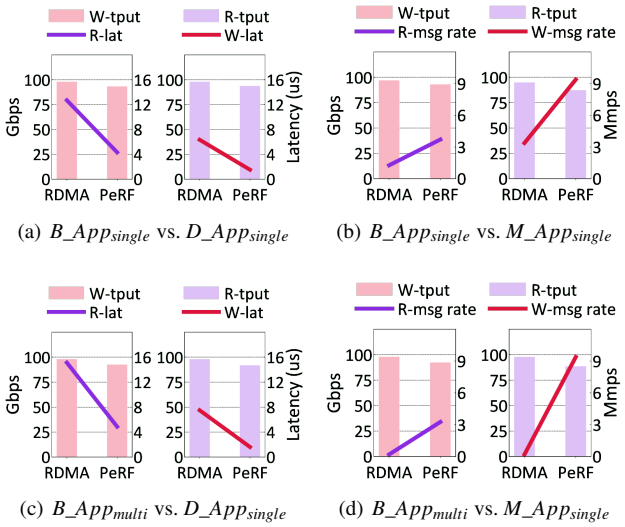


(a) Naitve RDMA (b) PeRF

Figure 18: Performance Anomaly between *B_App*s: In contrast to *RDMA*, PeRF properly manages *B_App*s using different message sizes.

are different. Based on our analysis illustrated in Section 2.2, network resources should be equally distributed between applications that use large messages, with the help of the granularity of MTU. However, as depicted in Fig. 18(a), when the SQ sizes are set to one (we varied the size of the SQ for both applications from one to five), the throughput for 1 MB message transmission is lower than that for 1 GB message transmission. This is because 1 MB message delivery requires more frequent posting of WRs and polling of CQEs than 1 GB message delivery, resulting in more processing or PCIe overhead. This often causes the SQ of the application using a 1 MB message to become empty, which pauses the message delivery.

Fortunately, this degradation can be easily mitigated by increasing the size of the SQ, which keeps the message delivery from being stalled while a new WR is being posted. Furthermore, Fig.18(b) shows PeRF addresses the performance anomaly between *B_App*s with different message sizes even when SQ is set to one. PeRF divides a large message WR into smaller sub-WRs, allowing them to be treated as identical to an application sending *SUB_MSG_SIZE* of messages.



(a) B_App_{single} vs. D_App_{single} (b) B_App_{single} vs. M_App_{single}

(c) B_App_{multi} vs. D_App_{single} (d) B_App_{multi} vs. M_App_{single}

Figure 19: Performance Anomalies between Inbound READ and Outbound WRITE Operations: PeRF provides isolation between tenants in situations in which inbound READ(R-) and outbound WRITE(W-) traffic coexist on the same host. In (c) and (d), B_App_{multi} utilized 15 QPs.

C Deep Consideration of READ Isolation in PeRF

When a local node initiates a READ operation, it requires data transmission from a remote node, which can cause contention with outbound WRITE operations at the remote node. However, PeRF appropriately isolates WRITE and READ operations on the same RNIC. We evaluate PeRF in eight different scenarios, combining different cases of a B_App_{single} competing with a B_App_{multi}, a D_App_{single}, or an M_App_{single}. In each scenario, a local node runs a B_App_{single} performing inbound READ operations [5] while executing other applications performing outbound WRITE operations, and vice versa. Our results, shown in Fig. 19, demonstrate that PeRF reduces average latency by at least 65% for the D_App and improves message rates by 3∼50× for the M_App, with only a 5% compromise on the throughput of the B_App_{single}.

D Future Works in PeRF Implemetation

This paper provides a systematic illustration of PeRF's design and implementation, accompanied by extensive evaluations. However, there are additional implementation issues that need to be addressed to ensure PeRF's perfect compatibility with applications utilizing various features of RDMA. We leave these issues as our future work and explain their details in the following.

[5] A read request is transmitted from a local node to a remote node while read data is sent from the remote to the local.

1) Support for UD Connection: RDMA supports three types of connections for message transmission: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). Our novel preemption mechanism, which leverages experimental verbs like *WAIT* and *ENABLE*, operates effectively with RC and UC. However, these experimental verbs are not supported in UD, making it necessary to develop a new isolation mechanism for PeRF to be compatible with UD. Fortunately, we have discovered that it is easy to provide QP-level isolation in UD, since a single UD QP can communicate with multiple destinations based on network information recorded in WRs by applications. Thus, we can modify PeRF to dynamically regulate the number of active UD QPs, providing QP-level isolation in UD. Additionally, 0_*WR* can be used instead of 0_*WAIT* to achieve message-level isolation in UD. This approach may introduce some network overhead, but it is negligible in networks with rich bandwidth.

2) Support for Event-driven Message Transmission: RDMA provides an event-handling mechanism that allows an application to poll CQEs only after they are generated. This event-driven approach provides RDMA applications with great CPU efficiency for transmitting large messages, as it eliminates the need for busy polling CQEs. Since PeRF's completion process based on the ECE is perfectly transparent to applications, implementing event-driven WR completion in PeRF is straightforward.

3) Support for Atomic Operations: RDMA supports two types of atomic operations: fetch and add (FAA) and compare and swap (CAS). As seen in [23], the atomic operations also cause performance anomalies in multi-tenant environments. Consequently, we plan to analyze these anomalies and explore the possibility of applying our preemption mechanism based on 0_*WAIT* WRs.