USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

BURST: Seeking High-performance, Interoperability and Scalability in Soft-RDMA

Huijun Shen, *Hunan University;* Zelong Yue, Jian Yang, Zhuo Jiang, Lang An, Yulin Chen, Yong Zhang, Luochangqi Ding, Xiaolong Zhong, Zhihong Wang, Jie Ding, Hongyu Wu, and Jianxi Ye, *ByteDance Inc.;* Xijin Yin, Xingyu Zhang, Xingyu Guo, and Guo Chen, *Hunan University*

https://www.usenix.org/conference/nsdi26/presentation/shen

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

BURST: Seeking High-performance, Interoperability and Scalability in Soft-RDMA

Huijun Shen[†], Zelong Yue[‡], Jian Yang[‡], Zhuo Jiang[‡], Lang An[‡], Yulin Chen[‡], Yong Zhang[‡], Luochangqi Ding[‡], Xiaolong Zhong[‡], Zhihong Wang[‡], Jie Ding[‡], Hongyu Wu[‡], Jianxi Ye[‡], Xijin Yin[†], Xingyu Zhang[†], Xingyu Guo[†], Guo Chen[†,*]

[†]*Hunan University* [‡]*ByteDance Inc.*

Abstract

Modern data centers deploy heterogeneous server pods, including a mix of commercial RDMA NICs (RNICs), legacy Ethernet NICs, and custom in-house hardware. This diversity creates significant interoperability challenges, particularly for Non-RNIC-to-RNIC (NR2R) communication, a scenario driven by emerging disaggregated workloads like LLM inference, large-scale infrastructure upgrades, and the integration of novel network protocols. Due to strict hardware dependencies, RNICs discard packets from non-compliant packets, forcing a costly fallback to TCP/IP and limiting RDMA network scaling. Existing software RDMA solutions, RXE (SoftRoCE implementation in Linux kernel), suffer from prohibitive CPU overhead, making them unsuitable for high-speed networks.

To address this, we present BURST, a high-performance, user-space software RDMA stack designed for high-speed networks. BURST operates as an independent process that maintains full compatibility with the standard RDMA Verbs API, allowing unmodified applications to run on Ethernet NICs. It integrates a lock-free DPDK data plane for line-rate packet processing, leverages Intel's DSA for reducing CPU, and features a kernel-bypass connection manager to accelerate setup. Experimental results show that BURST achieves 98.7% line-rate bandwidth on 400G NICs, delivering a 3.2-6.3x throughput improvement over kernel RXE. In production workloads, BURST accelerates LLM inference latency to 25.2% of TCP's and increases connection setup speeds by 12x compared to native RDMA CM, demonstrating its benefits for unifying communication in heterogeneous environments.

1 Introduction

Remote Direct Memory Access (RDMA) delivers high performance and low CPU overhead through hardware-offload capabilities, yet its implementation remains tightly coupled with vendor-specific designs in commercial RDMA NICs (RNICs). Although RDMA has been widely deployed in modern data centers [19, 21], most large-scale infrastructures rarely standardize their hardware on a single vendor, let alone a specific generation, to mitigate deployment costs and supply chain risks [29]. Consequently, commercial data centers accommodating high-speed network traffic are profoundly heterogeneous, typically comprising three distinct types of NICs: commercial RNICs from major vendors [25, 39, 40], legacy Ethernet NICs with high-performance software stacks [15, 33, 34, 59], and in-house customized RDMA NICs with self-developed features [17, 20, 35, 49, 50, 54].

This hardware diversification creates critical, yet often overlooked, interoperability demands, particularly in what we term Non-RNIC-to-RNIC (NR2R)[1] scenarios. Analysis of enterprise production environments reveals three key drivers for efficient NR2R communication.

First, the architectural disaggregation of modern applications creates a direct need for high-speed communication between non-RNIC and RNIC pods. The rise of distributed AI workloads, such as disaggregated LLM inference [23, 43, 45], typifies this trend. With the advent of architectures that require massive KV cache transfers, workloads are split: compute-intensive pre-fill stages run on cost-effective non-RNIC pods, while latency-sensitive decode stages run on high-performance commercial RNIC pods. This architectural shift creates a critical need for efficient data exchange between them. Second, scaling and upgrading the infrastructure necessitates accelerating the performance of the vast, pre-existing base of legacy hardware. In our storage clusters comprising hundreds of thousands of nodes, over 20% are legacy NICs limited to traditional software protocols. With increasing traffic volumes, their communication with newly deployed pods using commercial RNICs has become a significant bottleneck. Finally, the trend of hardware protocol innovation introduces a transitional interoperability challenge. To support domain-specific enhancements (*e.g.,* SACK), large operators increasingly develop custom in-house NICs. However, achieving full hardware-level RoCEv2 compatibility for

*: Guo Chen is the corresponding author.

[1]We define non-RNICs as all Ethernet NICs without the RoCE feature.

these new devices is a complex, multi-year process. During this prolonged evolution, emerging hardware must still interoperate with the vast installed base of commercial RoCEv2 pods, creating a pressing need for a software-based compatibility layer, for example, when integrating advanced transport protocols [2, 30, 46, 47, 54].

This incompatibility presents a challenging issue. Due to native RDMA's strict hardware dependencies, any non-standard RoCEv2 packets originating from a non-RNIC are simply discarded by the other end's RNIC hardware. This hardware-driven fragmentation creates an insurmountable barrier to direct communication, forcing a costly fallback to the kernel TCP/IP (KTCP) stack and sacrificing RDMA's performance benefits precisely when they are most needed.

Existing research on RDMA interoperability, while valuable, fails to adequately address this critical NR2R challenge. As a summary in Fig. 1, case (1), prior works focus mainly on the interoperability between distinct RNICs, utilizing RNIC features such as high-performance packet processing (*e.g.,* UC mode [29, 57]) and direct memory access (*e.g.,* GPU direct [57]). Case (2), other works focus on pure non-RNICs [33, 35, 57], which cannot directly talk to existing RDMA nodes. As such, case (3), prior works can only use software RDMA stacks to enable NR2R communication [34, 48]. Unfortunately, as we will detail in Sec. 2.2, existing solutions suffer from prohibitive CPU overhead and performance limitations in high-speed networks, making them unsuitable for our production systems.

As a global-scale Internet infrastructure provider, our goal is to unify the RDMA protocol stack across diverse hardware configurations especially in NR2R scenarios, ensuring maximum interoperability, performance, and generality. We aim to build this unification at a foundational layer that preserves compatibility with the standard RDMA Verbs API, allowing various applications to leverage it without modification.

In this paper, we propose BURST, a novel software RDMA network stack designed for high-performance and seamless interoperability in 400G networks. We deploy BURST as an independent user-space process. Unmodified applications can use BURST in this process service with standard Ethernet NICs. To achieve this, BURST accesses metadata registered by the user via *libibverbs*, such as Queue Pairs (QPs), Completion Queues (CQs), and Memory Regions (MRs), through inter-process shared memory. It then parallelly reads and processes Work Queue Entries (WQEs) using an enhanced shared queue model implemented in software. BURST integrates a lock-free Data Plane Development Kit (DPDK) [4] data plane for line-rate packet processing and leverages Intel's Data Streaming Accelerator (DSA) to support near zero-copy I/O. Beyond data plane performance, applications such as upgrading or restarting distributed storage systems now demand the rapid setup of tens of thousands of RDMA connections, a requirement also highlighted by prior work [31, 55]. However, hardware RDMA Connection Management (CM), constrained

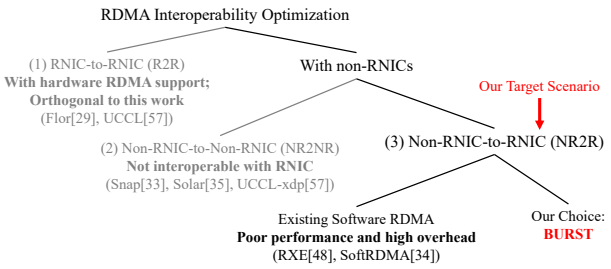


Figure 1: Classification of RDMA Interoperability Work.

by limited on-chip resources [54] , kernel-dependent [22] and I/O overhead, struggle to meet the increasingly strict, sub-millisecond service level agreements (SLAs) for this task. Furthermore, BURST introduces a pure user-space, kernel-bypass connection manager that leverages abundant host memory resources to accelerate connection setup.

Experimental results demonstrate that BURST achieves 98.7% line-rate bandwidth per 400G/NIC on NR2R networks with less than 4 CPU cores, achieving a 3.2-6.3 *times* throughput improvement compared to RXE [48] (also known as Soft-RoCE). Compared to KTCP (the hardware's fallback option), BURST increases throughput by approximately 47 Gbps and saves over 66% of CPU resources. By supporting GDR in software, BURST reduces the end-to-end average latency for LLM inference deployed in NR2R scenarios to 25.2% of that of KTCP. We also evaluated the user-space RDMA Connection Management (CM) connection setup approach provided by BURST in a high-concurrency, enterprise-level storage application scenario. The results show that with multi-threading support, BURST can achieve a connection setup speed of over 24,000 connections per second, which is 12 *times* that of the native RDMA CM.

Our main contributions are summarized as follows:

- We propose BURST, the first high-performance software RDMA network stack demonstrated to be deployable in large-scale production applications on 400G NICs, specifically addressing the NR2R interoperability challenge.
- We propose a pure user-space CM approach that decouples connection logic from the kernel, dramatically accelerating control plane operations for large-scale, dynamic applications.
- We evaluate and deploy BURST in demanding, real-world scenarios, including large-scale RPC services, distributed storage systems and LLM inference applications, showing that it brings significant performance benefits and has already replaced KTCP or RXE communication flows.

2 Background and Motivation

2.1 RDMA Interoperability Challenges: A Taxonomy

In modern data centers, hardware-based RDMA networks are progressively replacing traditional software-based TCP/IP stacks for high-speed applications. Fundamentally, RDMA offloads the entire network stack processing to specialized hardware (RNICs), thereby bypassing remote CPU and OS involvement in data transfers. This approach delivers low-latency and high-throughput communication. Through continuous iterations by major vendors, Ethernet RDMA bandwidth has evolved rapidly from an initial 10Gbps to 400Gbps and beyond in just a few years [38, 40]. This rapid pace of hardware evolution makes it impractical for large-scale data centers to maintain a homogeneous hardware infrastructure, creating a complex, hybrid environment.

As illustrated in Fig. 1, this hardware diversity gives rise to a taxonomy of interoperability challenges. While challenges exist within homogeneous pods, this work focuses on the most critical and unaddressed of these scenarios: the Heterogeneous Non-RNIC-to-RNIC (NR2R) connection (other scenarios discussed in Sec. 6). The difficulty in NR2R stems from a fundamental asymmetry between endpoints, where one side benefits from full hardware offload while the other relies entirely on software.

This asymmetry manifests as a critical hardware limitation. Due to native RDMA's strict hardware dependencies, commercial RNICs are designed to discard any non-standard RoCEv2 packets originating from a non-RNIC endpoint at the hardware level. Ultimately, this hardware-driven fragmentation creates an insurmountable barrier to direct communication, forcing a costly fallback to the TCP/IP stack and sacrificing RDMA's performance benefits precisely when they are most needed.

2.2 NR2R Work and Limitations

Performance Limitations of Existing Software RDMA Stacks. As mentioned before, the only viable approaches for NR2R interoperability are pure software RDMA stacks [34, 48]. However, our analysis reveals that these solutions suffer from critical weaknesses in both performance and functionality, rendering them unsuitable for deployment in modern, high-speed production environments. RXE [48], the sole open-source, kernel-space RoCE implementation, offers compatibility but introduces operational incompatibilities with standard OFED drivers [8]. It also creates excessive latency overhead because control signals must traverse multiple software layers within the kernel stack [24]. We confirm its performance deficiencies in Sec. 5.1. Another example, SoftRDMA [34] is a proof-of-concept validated only on 10G NICs. This approach, built on a user-space TCP/IP implementation, retains a memory copy from the application to the stack and lacks consideration for advanced RDMA transport features like congestion control and concurrent connections.

Functional Gap: Lack of Software GPU Direct RDMA (GDR). Beyond raw performance, a more critical failure of existing solutions is the complete lack of support for GPU Direct transport. This limitation is not trivial; even Snap [33], a state-of-the-art user-space network system from Google that delivers superior performance over kernel TCP/IP, does not provide a software path for GDR. This functional gap is critical for modern AI and scale-up applications.

While TCP-based alternatives for GPU transport exist, they are not viable in practice. The state-of-the-art solution, Device Memory TCP [9], suffers from three prohibitive deployment barriers: (1) **Hardware Dependency:** It requires NICs with specific header-split capabilities to separate packet headers and payloads into different buffers. (2) **Kernel Dependency:** It is built on a bleeding-edge Linux kernel (6.12+), necessitating extensive backporting efforts for enterprise-grade systems. (3) **Application Access Overhead:** It lacks native GPU transport APIs, forcing intrusive and friction-prone application code modifications.

Furthermore, attempting to use such TCP-based methods to connect with commercial RNICs creates a functional conflict, as the RNIC's own hardware-based GPU Direct capability cannot coexist with the software TCP stack, making optimal performance impossible to achieve.

Production Experience. Consequently, TCP/IP remains the de facto solution for large-scale data transfer in scenarios demanding ease of deployment and compatibility. As application scales expand in modern data centers, the TCP/IP stack increasingly fails to satisfy performance requirements, forcing endpoints to forfeit the hardware offload benefits of RDMA NICs. This reality underscores the imperative for a robust, high-performance, and fully-featured software solution for RDMA interoperability.

3 System Architecture and Design Principles

3.1 Goals and Principles

Our goal is to support RoCE for both host and GPU memory through software in 400+ Gbps/NIC networks, as shown in Fig. 2, maximizing network performance while minimizing CPU overhead (significantly outperforming kernel-based RXE and TCP) and maintaining application transparency to reduce upgrade/deployment costs.

To achieve these objectives, we adhere to four design principles: (1) Preserve standard RDMA library interfaces, ensuring applications use BURST identically to hardware RDMA stacks; (2) Saturate line-rate bandwidth to approach hardware RNIC performance; (3) Handle RDMA transport procedures (connection setup, WQE scheduling, etc.) with maximal efficiency to reduce latency overhead; (4) Eliminate CPU

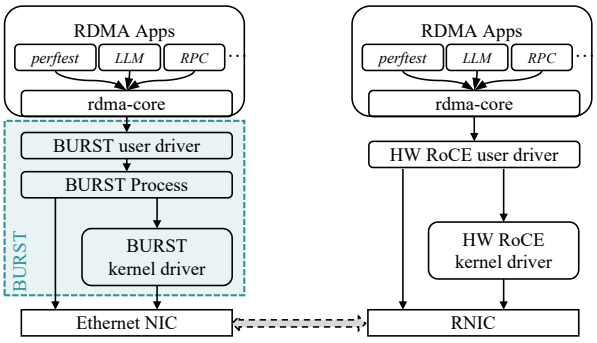


Figure 2: BURST in NR2R scenario.

overhead from memory copies to accelerate BURST 's data transfer capability.

3.2 Our Choice: An Independent Process

Guided by the above principles, we design BURST to operate as an independent user-space process. On one hand, existing kernel-centric solutions, such as RXE, are known to suffer from excessive software overhead due to system calls and context switching. This motivates migrating the bulk of the protocol stack's functionality into user-space. On the other hand, there are some network stack deploying as a LibOS model [13, 27, 28, 59], which presents significant challenges in general-purpose RDMA environments. First, the standard OFED drivers have inherent kernel dependencies for initialization and control path operations. Achieving compatibility with the standard RDMA Verbs API in a pure user-space LibOS is non-trivial and often forces application-level code modifications, which contradicts our goal of seamless migration. Second, when multiple RDMA applications run on the same host, each with its own LibOS instance, a mechanism is required to steer incoming packets to the correct stack. While some work [59] addresses this by partitioning ports, this feature is not universally supported by commercial RNICs, and implementing this steering in software introduces new logical overhead.

Consequently, we adopt the independent process model for BURST to balance high performance with general applicability, as shown in Fig. 5. Systems like Snap [33] also utilize a microkernel-like process but are typically tailored for a specific application ecosystem and its proprietary interfaces. However, BURST is designed for universal RDMA application scenarios, providing robust support for multi-process and multi-threaded environments while ensuring fairness and isolation between different workloads.

Following these principles and consideration, BURST faces four core challenges:

Challenge 1: Isolation of memory access. Before network communication, applications using RDMA protocols allocate memory and register MR via standard RDMA interfaces. However, this target address in user-space or GPU

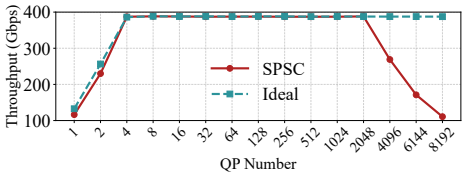


Figure 3: Impact of QP scalability with different DBQ models.

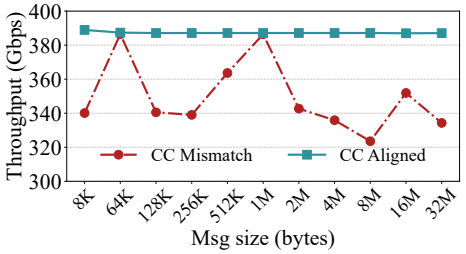


Figure 4: Throughput Comparison with Congestion Control Mismatch.

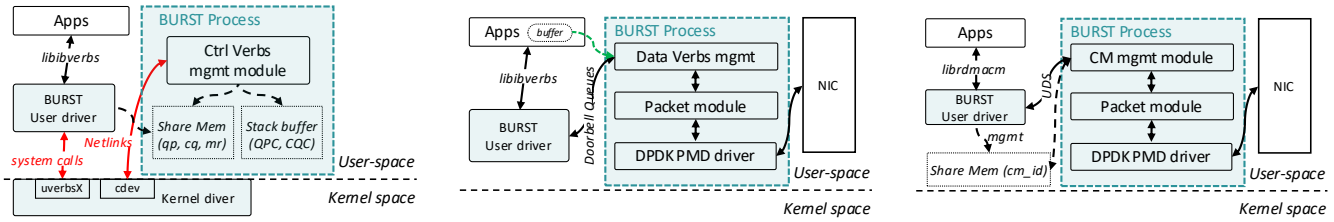
memory cannot be directly accessed by the kernel or non-RNICs. Directly using user-provided address information for memory access will fail to access the correct data.

Challenge 2: WQE perception overhead. A key challenge is efficiently notifying the user-space BURST process of new WQEs posted by applications. Straightforward notification methods often lack the required efficiency and scalability. For instance, relying on kernel-based mechanisms like the system call `eventfd` incurs high overhead (approx. 4μs per operation) [52]. The other intuitive approach of dedicating a simple doorbell queue (DBQ) for each QP (a Single-Producer-Single-Consumer (SPSC) model) fails to scale, as our evaluation in Fig. 3 shows its throughput collapsing sharply after the QP count reaches 4096. Therefore, designing a scalable shared-memory queue for WQE synchronization is non-trivial. Without a proper design, locking and access contention will bring extra latency overheads that significantly degrade performance.

Challenge 3: Congestion Control Divergence. Existing studies confirm that different congestion control (CC) between RDMA endpoints degrades performance [29]. In our NR2R scenario, we evaluate the performance when BURST running DCQCN communicates with NVIDIA BF3 series RNICs using ZTRCC [2] [41]. Fig. 4 reveals significant performance penalties, including reduced throughput convergence rate and lower peak bandwidth. For RNICs, functioning as black boxes, we possess limited information to probe their congestion control algorithms and operational mechanisms.

Challenge 4: Kernel-Dependent Connection Setup. After migrating the RDMA data path to software, the traditional connection setup path remains a bottleneck [6, 22, 55]. This process is heavily kernel-dependent, where complex system calls and time-consuming context switching create significant overhead, hindering end-to-end performance optimization.

[2] NVIDIA's Zero Touch RoCE (ZTR) RTT-based CC algorithm.



(a) Shared Resource Management (b) Efficient Data Plane (c) User-space Connection Management

Figure 5: The Architecture of BURST, illustrating its three core components: (a) Shared Resource Management, (b) Efficient Data Plane, and (c) User-space Connection Management.

3.3 BURST Overview

As shown in Fig. 5, the logical flow of an application communicating with a remote commercial RNIC through BURST highlights its three primary components:

Shared Resource Management: A framework that enables the application, the kernel driver, and the BURST process to share critical resources like MRs, QPs, and CQs (Fig. 5(a)). This is achieved through carefully orchestrated memory mapping, which allows for lock-free, zero-copy interactions between the application and the BURST stack.

Efficient Data Plane: A high-performance data plane, built upon DPDK, that manages the entire lifecycle of packet transmission and reception (Fig. 5(b)). This includes efficient WQE processing, packet encapsulation/decapsulation following the InfiniBand specification, and congestion control updates.

User-space Connection Management: A pure user-space RDMA CM approach that handles the entire connection setup process via the standard *librdmacm* API, eliminating kernel involvement and memory copies for faster establishment (Fig. 5(c)).

4 BURST Design

This section presents the complete design of BURST. RDMA communication is typically divided into a control path and a data path. We first explain our control path design. In Sec. 4.1, we describe how BURST manages network functions and resources in user-space while maintaining compatibility with standard RDMA control plane kernel modules. Following this, Sec. 4.2 presents our design for enabling the BURST process to efficiently perceive data transfer requests (WQEs). Accordingly, we shift to the data path, detailing our efforts to minimize software overhead and presenting the fundamental data transmission workflow in Sec. 4.3 and Sec. 4.4. Finally, We introduce our pure user-space connection setup, which is built upon the standard RDMA CM API in Sec. 4.7.

4.1 Shared Resources Access

To handle the RDMA control path in user-space (**Challenge 1**), our workflow aligns with the standard OFED model. Applications invoke standard *libibverbs* interfaces to register resources like MRs, QPs and CQs. We process these *Verbs* requests by introducing a BURST software device, which relays the calls to a corresponding kernel driver for privileged resource registration.

As shown in Fig. 5(a), BURST's kernel driver uses a netlink socket to pass the address information of the newly created resources (MRs, QPs, CQs) to the user-space BURST process. The BURST process then uses *mmap()* to map these memory regions into its own address space. This thread-safe mechanism not only grants the BURST process exclusive access, protecting resources from the user application, but also enables it to subsequently bypass the kernel on the data path, mirroring the behavior of the hardware RDMA. BURST runs as an independent user-space process, to scale with increasing traffic demands, it can spawn multiple worker threads, where each thread is responsible for processing WQEs from a set of application QPs. A single worker thread may manage multiple QPs, enabling flexible load distribution while preserving isolation between applications.

For managing the state of each connection, RDMA uses the Queue Pair Context (QPC) to record and manage. BURST allocates a region of host memory and manages these contexts as an object pool. In contrast to hardware RDMA, which must store and access these contexts on the NIC, maintaining them directly in host memory provides significantly larger capacity and faster access speeds.

4.2 Handling Doorbell Queues with an Enhanced Shared Queue Model

Hardware RDMA employs a *Doorbell mechanism* to notify the RNIC that new WQEs have been posted for processing. This is achieved by mapping a dedicated Doorbell register on the RNIC to the corresponding host memory. When the CPU writes to this register, it generates a hardware signal that is directly perceived by the hardware circuitry. This notification

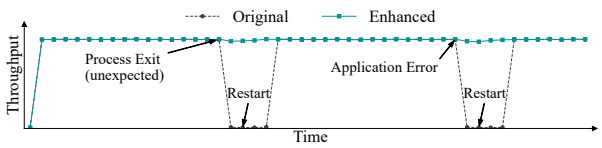


Figure 6: Crash-not-Safety Findings in Production Apps.

prompts the RNIC to automatically fetch and process the new WQEs from the corresponding queue without further software intervention.

In contrast, BURST must emulate this notification mechanism in software while coordinating work across threads, since it operates as an independent user-space process. BURST acknowledges and processed all WQEs from multiple applications. However, as a user-space process, it lacks the hardware's intrinsic ability to perceive these memory-write events. Thus, BURST provides an enhanced Multi-Producer Single-Consumer (MPSC) model. (**Challenge 2**)

Dedicated DBQ for Worker Threads to Enhance Parallelism. We instantiate a dedicated DBQ for each worker thread within the BURST process. In this model, the multiple QPs from various applications act as concurrent *producers* that are directed (e.g., via hashing) to a specific DBQ. The corresponding BURST worker thread acts as the single *consumer* for that queue. This architecture partitions the overall scheduling problem into multiple parallel MPSC systems, avoiding a single point of contention.

Crash-Safety with Timeout-based Reclamation. Inspired by state-of-the-art MPSC queue designs [53], we partition the DBQ buffer into fixed-size blocks. This design effectively mitigates the contention issues over shared head/tail pointers that are common in other concurrent queue implementations. However, we identified and addressed a critical crash-safety issue. As illustrated in Fig. 6, we observed in production that when an application forcibly exits or encounters unexpected errors, the link bandwidth for an entire worker thread would drop to zero due to the single application's failure. Specifically, the problem arises if a producer reserves a slot in the queue but crashes before it can write the data and confirm the operation (e.g., by updating the tail pointer). This would leave the slot in an unusable "zombie" state, effectively blocking the queue and causing data loss.

To ensure crash safety, we implemented a timeout-based reclamation mechanism. If the producer fails to commit the write within this predefined period, we assume the producer has failed. BURST then safely invalidates and reclaims the slot, ensuring the queue's integrity and robustly handling producer failures. (Appendix A.1.2)

BURST's DBQ model not only demonstrates superior performance but also eliminates the aforementioned crash-safety vulnerability. The detailed implementation and evaluation of this model are explained in Appendix A.1.

4.3 Packet Transmission

As mentioned in Sec. 3.3, BURST's data plane proposes efficient data transfer logic in user-space. Fig. 7 shows the detailed packet sending and receiving process, and the major components within BURST's *Packet module* are as follow: **Requester:** The Requester module handles all SEND/WRITE/READ WQEs from the send queue (SQ), encapsulating them with InfiniBand and UDP/IP headers for transmission via DPDK APIs; **Congestion Control:** This module updates congestion parameters based on received packets' acknowledgments (ACKs), explicit congestion notification (ECN), and congestion notification packet (CNP) values; **Responder:** The Responder module processes incoming data packets from a peer's SEND or WRITE operations. For SEND semantics, it copies the payload to the target address and sends an ACK. For WRITE semantics, it copies the payload without generating a completion queue element (CQE); **Completer:** The Completer module processes received ACK packets, verifies the packet sequence number (PSN), and generates a corresponding CQE for the SEND/WRITE WQE once all associated work requests (WRs) are fully acknowledged.

Data process. An RDMA SEND/RECV flow comprising the following steps: (1) The user posts a SEND request by putting a WQE into a SQ. The Requester parses the WQE to get the virtual address of the data buffer and translates it to a physical address using the Memory Translation Table (MTT). (2) The Requester retrieves the data from the host buffer, splits it into MTU-sized packets, encapsulates each packet with the required headers, and sends them to the NIC via the DPDK-based data plane. (3) The packets are transmitted over the Ethernet. (4) The Responder on the receiver side validates the incoming packet by checking its PSN and remote key (R_KEY). (5) If the packet is valid, the Responder copies the payload to the destination address specified by the RECV WQE. For SEND semantics, a CQE is then created for the RECV host, and (6) an ACK packet is sent back to the sender host. (7) The Completer on the sender side processes the received ACK packet. It verifies the PSN and, if all associated WRs are acknowledged, generates a final CQE to complete the operation.

4.4 Reduce CPU in Data Transmission

Extensive prior research on software protocol stacks has demonstrated that the CPU overhead caused by memory copies has long been a critical bottleneck hindering performance breakthroughs [13, 27, 34, 59]. As analyzed in Sec. 2.2, memory access in native software RDMA stacks is performed by the kernel, thus consuming significant CPU resources for memory copies between user-space and kernel-space.

To minimize the CPU costs introduced by memory copies, BURST eliminates memory copies in the transmit direction

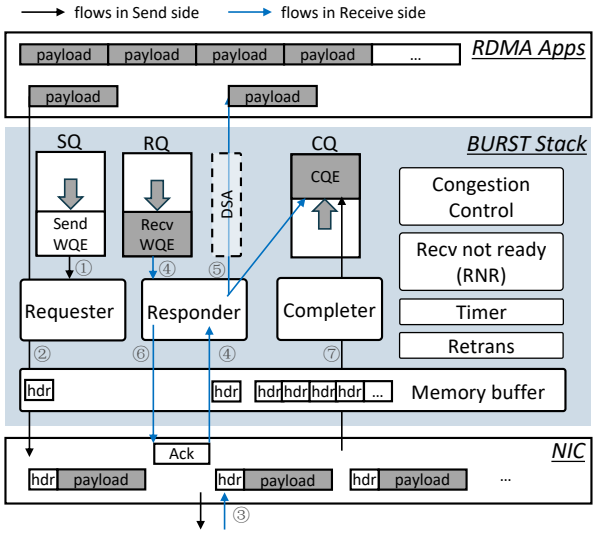


Figure 7: The data flows in BURST process.

via ingenious memory mapping and Memory Region (MR) address translation (Sec. 4.4.1). Although constrained by the requirement for user transparency on the receive side, BURST also supports hardware accelerators like the DSA to handle data copying, thereby offloading the CPU (Sec. 4.4.2).

4.4.1 Supporting Zero-copy in Send Side

In native hardware RDMA, when the user calls `ibv_reg_mr` to register a MR, the corresponding memory will be pinned, and the mapping between Virtual Address (VA) and Physical Address (PA) will be offloaded to the RNIC's Memory Translation Table (MTT).

As mentioned in Sec. 3.1, BURST needs to be deployed on non-RNIC hardware and must support user transparency. We resolve the correct PA and notify the NIC by maintaining new mapping relationships within the BURST process in user-space. Specifically, we design a system to map the User_VA (provided by applications) and the BURST _VA (controlled by the BURST stack) to the same PA. In this scenario, the User_VA-to-BURST _VA translation requires only a simple calculation. Subsequently, the translation BURST _VA to PA is handled by the IOMMU (Input-Output Memory Management Unit) maintained by the operating system, ensuring that the device can perform DMA operations on the memory. Finally, for the data transmission, BURST follows the steps as shown in Fig. 7: constructing the RDMA protocol header based on the WQE and translating the payload source address using the MTT. The packet send request and address information are subsequently passed to the NIC via DPDK.

Regarding GPU Direct RDMA (GDR) in send side, we make the following extensions for GPU memory access: (1) Since the GPU consists of several memory channels and stores data contiguously in physical memory, BURST leverages peer_mem kernel modules for getting the mapping between the User_VA and the GPU_PA. By working directly with the physical address, BURST can provide uniform DMA support for GPUs bypassing complex, vendor-specific GPU virtual memory systems and driver APIs. (2) After BURST delivers the PA to the NIC, the NIC directly DMAs the data payload from GPU memory using this PA. It then DMAs the header from host memory, similar to how BURST handles RDMA for host memory, completes the packet assembly, and transmits the packet.

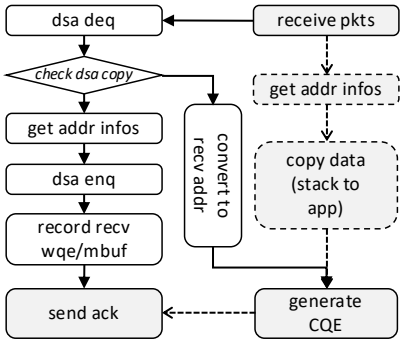


Figure 8: The BURST recv path with DSA.

4.4.2 DSA Acceleration in Receive Side

As depicted in Step 5 of Fig. 7, we integrate Intel DSA into the responder-side data path to mitigate CPU overhead from software-based data copies. Intel Data Streaming Accelerator (DSA) [14] is a hardware DMA engine that offloads memory copy operations from the CPU using asynchronous descriptors (Appendix A.5). Each thread binds to a dedicated DSA queue initialized at startup.

Fig. 8 illustrates the upgraded receive data path. While BURST leverages DPDK's scatter/gather capability on the send path to transmit headers and payloads without merge-copying, this does not provide end-to-end zero-copy semantics on the receive path. Upon receiving data, the responder enqueues a DSA descriptor that triggers an asynchronous copy from NIC-managed packet buffers to application-provided receive buffers (Fig. 7, Step 5). This copy is required to ensure data visibility before CQE notification and thus cannot be eliminated. After the copy completes, the corresponding WQE generates a CQE to notify the application, followed by an ACK sent to the remote host.

Single scatter/gather entry (sge). We restrict receive WQEs to a single sge to ensure atomicity of each DSA operation. Supporting multiple sges would require splitting a packet into multiple copy requests, complicating buffer management and CQE reporting.

4.5 Congestion Control Adaptability

As mentioned in Sec. 3.1 (**Challenge 3**), unified congestion control (CC) algorithms on both RDMA communication ends enable rapid convergence and better performance, especially under heavy traffic. However, for non-RNIC scenarios

where BURST operates, we face commercial RNICs as "black boxes", raising a design challenge: how to detect the CC algorithm used by commercial NICs before RDMA data transfer, or whether we can notify them to align with BURST's CC algorithm.

Through extensive experimentation and observation, we discovered that for commercial RNICs (e.g., NVIDIA BF3mini): When using CM interfaces for connection setup, ZTRCC is employed for congestion control; when using socket interfaces, DCQCN is detected. Based on this finding, we identified that the vendor_id parameter in RDMA CM packets can guide RNIC's CC selection.

Congestion Control Negotiation: As we optimized native RDMA connection setup with standard CM interfaces in Sec. 4.7, we can instruct RNICs to fall back to DCQCN during CM setup for efficient interoperability. Standard RDMA CM packets contain a 24-bit vendor_id field to coordinate protocol functionalities during connection setup [3]. BURST detects invalid vendor_id values, sets vendor_id=0 in CM packets, and switches to DCQCN. Upon receiving these packets, commercial RNICs detect vendor_id=0 and enforce DCQCN fallback to ensure baseline compatibility. This pre-transmission CC negotiation ensures efficient data transfer.

Other CC Algorithms Support: Leveraging software flexibility and programmability, BURST not only adapts to commercial RNICs' DCQCN [58] but also supports user-space congestion control algorithms. We have successfully implemented DCTCP [11] and provide a streamlined deployment platform for novel CC algorithm research [44]. Crucially, BURST enables dynamic selection among multiple CC algorithms based on network conditions.

4.6 Retransmission Considerations

Timeout Retransmission Mechanism in BURST. BURST supports a timeout retransmission mechanism to ensure efficient and reliable data transmission. The specific process is as follows: After a packet is sent, the timer is started and the `timer_run` event is placed into the task queue. Upon detecting a timeout, the corresponding QP is modified to the `need_retry` state. The requester enters the retransmission processing logic using a Go-Back-N (GBN) approach based on the QP state. Retransmission only handles inflight WQEs and does not process new WQEs. After completion, the timer is restarted. Upon receiving an ACK, the QP returns to its normal state. If the maximum number of retries is reached, the QP enters the `ERROR` state.

Error Acknowledgment Mechanisms in BURST. Consistent with the standard RDMA specification, BURST utilizes the `syndrome` field within acknowledgment (ACK) packets to signal specific error conditions. It generates NAK packets in response to various protocol violations, such as an out-of-order Packet Sequence Number (PSN), an invalid request, a remote key (R_Key) mismatch, or a remote operational error. Separately, if the responder detects insufficient receive buffer space (*e.g.,* a receive request has not been posted), it sends an Receive Not Ready (RNR) ACK packet. Upon receiving a NAK or RNR ACK, the completer on the peer side transitions the Queue Pair (QP) to the `need_retry` state and initiates a specific `rnr_timer`. Once this timer expires, the requester handles the retransmission identically to the timeout scenario described previously.

Design Choice: GBN over Selective Repeat. We adopt GBN primarily for broad compatibility with commercial RNICs, where GBN is the most widely supported retransmission strategy. While selective repeat offers higher efficiency, its vendor-specific implementations (e.g., Selective repeat in NVIDIA CX-6 Dx [37]) require hardware negotiation and lack uniform software interoperability across heterogeneous pods. However, if the peer RNIC supports a known selective repeat protocol (e.g., SRNIC [54] with explicit `seth` headers), BURST can leverage user-space packet headers to enable SACK-based retransmission(Appendix A.3).

4.7 User-space Connection Management

As discussed previously in Sec. 3.2 (**Challenge 4**), connection setup is an essential prerequisite for any RDMA data transmission. For native hardware RDMA protocol stacks, conventional connection mechanisms are fundamentally bottlenecked by kernel involvement, incurring high-latency overhead from system calls and context switching. To address this, BURST implements the entire connection setup process in user-space. This approach leverages the standard *librdmacm* API to maintain application compatibility while eliminating kernel overhead and costly memory copies, leading to significantly faster connection establishment.

As shown in Fig. 9, the RDMA CM handshake is similar to the familiar TCP three-way handshake, but with key advantages. The server first calls `rdma_listen` to await incoming requests. The client then initiates the connection by sending a `ConnectRequest` packet. The server, upon receiving this request, can choose to accept it by sending back a `ConnectReply` packet. However, a key distinction from traditional sockets lies in its inherently asynchronous, event-driven model; the application does not block on a function call but rather receives events (e.g., `CONNECT_REQUEST`, `ESTABLISHED`) from an event channel, similar to `epoll`. Crucially, these CM packets are standard transport packets that travel the same network path as subsequent data traffic (in-band signaling). This provides a significant benefit: a successful connection establishment implicitly validates the data path's connectivity, eliminating the need for separate data path probes.

To manage this user-space dialogue within BURST, Fig. 5(c) illustrates our interaction mechanism. To bridge the application and the BURST process, we co-design components in both the user-space driver and the process itself.

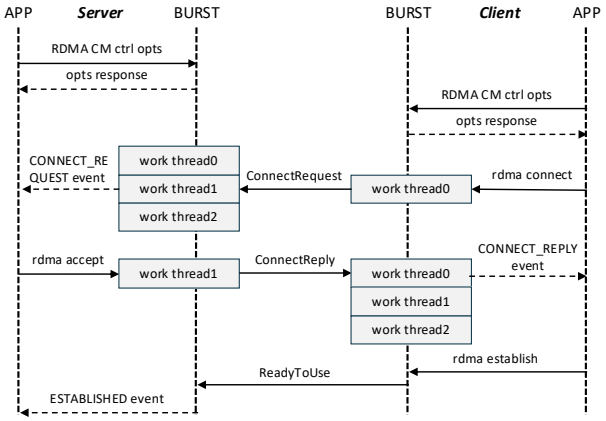


Figure 9: The BURST Connection Setup Process.

When an application calls a standard *librdmacm* function to manage a Connection Manager Identifier (`cm_id`), our user-space driver intercepts this call. It then forwards the request to the BURST process via a Unix Domain Socket (UDS). On the receiving end, a dedicated CM control thread within BURST listens on the UDS. This thread validates the incoming request and enqueues it into a Connection Management Work Queue (CM WQ). From there, worker threads in BURST's thread pool consume the request to perform network protocol processing. Once the connection operation is complete, BURST uses a semaphore to signal the user-space driver, which in turn unblocks the application's original API call. This decoupled design makes the entire user-space offload transparent to the application.

Once a connection request is dequeued by a worker thread, BURST assumes full responsibility for CM packet encapsulation. Bypassing the kernel means that BURST itself must construct the entire packet. It leverages DPDK to build the complete header stack, from the lower-level Ethernet and IP/UDP headers to the InfiniBand Base Transport Header (BTH). The RDMA CM payload, containing the specific connection parameters, is then appended according to the InfiniBand Architecture Specification. By directly implementing these standardized packet formats, BURST operates independently of the kernel's protocol logic, ensuring both protocol correctness and high performance. remark

5 Evaluation

In this section, we evaluate BURST through testbed experiments and enterprise CCL workloads, comparing it with BF3mini, RXE, and KTCP. We examine: (1) throughput, latency, and software overhead in 400G NR2R networks (Sec. 5.1); (2) LLM inference acceleration benefits (Sec. 5.2); (3) performance gains from DSA optimization (Sec. 5.3); and (4) the scalability and performance of BURST's UCM in large-scale applications (Sec. 5.4).

Environment setup: Our evaluation utilizes two testbeds. **Testbed1** consists of two Inspur NF5280M6 hosts (Intel Xeon Platinum 8457C 3.10GHz) connected via a 64-port 400G enterprise switch. The client host is equipped with four 400G Ethernet NICs, while the server host is equipped with four BlueField-3 E-series DPU 400GbE/NDR NICs. **Testbed2** (only for Sec. 5.3) comprises two Inspur NF5280M6 hosts (Intel Xeon Platinum 8457C 3.10GHz), each equipped with one 100G Mellanox ConnectX-6 Dx NIC, connected through a 48-port 100G enterprise switch. In both testbeds, the Maximum Transmission Unit (MTU) is set to 4200, and the driver version is OFED 23.10.

We evaluate BURST against three baselines: the kernel's software RDMA implementation (RXE from Linux kernel 5.15.120), kernel TCP (KTCP from Linux kernel 5.10.200), and native hardware RDMA. We use several standard tools for our workloads: *perftest* [5] to measure the performance of BURST, RXE, and hardware RDMA; *iperf* [26] for KTCP; and *cmtime* [7] to evaluate single-process RDMA CM connection setup time.

5.1 Performance on 400G Networks

Throughput. We compare BURST with the kernel RXE and kernel TCP (KTCP) in NR2R scenarios, and we also test hardware RDMA (BF3mini) in RNIC-to-RNIC scenarios as a point of reference. For these experiments, we use Testbed 1 with one network card at each end, and have 32 available CPU cores. We evaluate the throughput of RXE, BURST, and BF3mini using the *perftest* benchmark, configured with 16 QPs and a `post_send` batch size of 16, following its default configuration with unidirectional traffic from sender to receiver. We focus on unidirectional bandwidth and multi-qp in this evaluation, as it reflects common NR2R deployment scenarios where one endpoint primarily acts as a sender or receiver, and avoids ambiguity introduced by bidirectional traffic configurations. Additionally, our single-QP performance achieves up to 3 Mpps and approximately 110–120 Gbps, which is bounded by the single-queue performance limitation of DPDK (Appendix A.7).

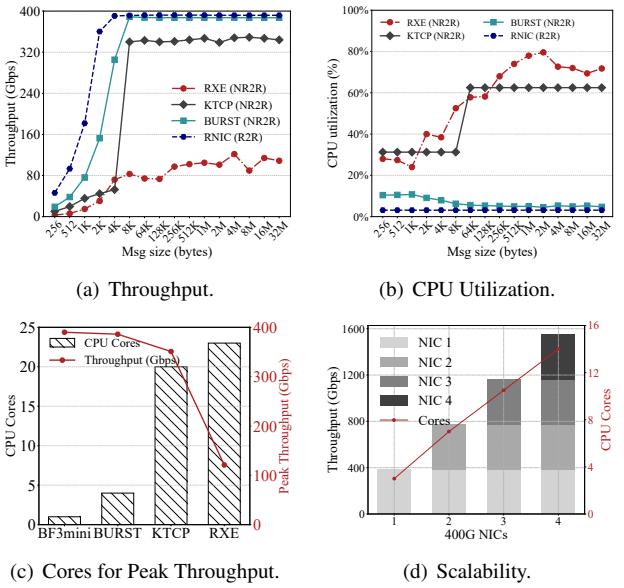
As shown in Fig. 10(a), BURST demonstrates significant throughput improvements over the kernel-based RXE and KTCP stacks across varying message sizes. When the message size exceeds 8 KB, BURST performs up to 4.69× higher bandwidth than RXE and up to 2.86× higher bandwidth than KTCP, achieving the theoretical line-rate (390Gbps) and has a small gap of 1.3% of the hardware RDMA (RNIC). In contrast, RXE's reliance on system call-intensive packet processing and memory copies limits its performance. Notably, our experiments show that RXE's performance is worse than KTCP under a 400G network, which we attribute to its kernel stack lacking optimizations for congestion control [24], inlining, and CPU cache misses.

As shown in Fig. 10(b), we monitor BURST's PMD threads, which show a modest 19%–42% CPU utilization during peak throughput, which is sufficient to achieve the peak bandwidth

of a single 400G NIC. All CPU usage results reported here correspond to unidirectional 400G traffic, where data transfers are issued from the sender to the receiver, consistent with the *perftest* default behavior. This represents a CPU utilization savings of 62.2%–93.1% compared to RXE and 82.8%–92.7% compared to KTCP. For the software stacks (BURST, RXE, and KTCP), with a message size of 1MB, we test the number of threads required to achieve peak throughput by increasing the number of work threads. Fig. 10(c) shows the peak bandwidth and the number of CPU cores required for each protocol stack. Given that most modern servers have a large number of cores (> 32), a CPU core overhead of less than 4 cores per NIC is a small price to pay for the significant performance benefits BURST provides in production.

Multi-NIC Scalability. We evaluate the multi-NIC scalability and total throughput of BURST by incrementally adding NICs and QPs on a single machine. We set the parameters for *perftest* with a message size of 1MB, 8 QPs per NIC, a sender queue depth of 128, and 16 available CPU cores. To prevent performance loss from cross-NUMA memory access, we bind all NICs to their corresponding NUMA node.

As shown in Fig. 10(d), BURST demonstrates near-linear scaling. With a single NIC and 8 flows, BURST achieves a throughput of 387.12 Gbps using 3 CPU cores. When we increase the configuration to four NICs and 32 flows, BURST maintains an average throughput of 387.6 Gbps per NIC, while utilizing a total of 14 CPU cores. This performance highlights BURST's ability to efficiently scale and saturate multiple modern NICs with a low CPU footprint.



(a) Throughput. (b) CPU Utilization.

(c) Cores for Peak Throughput. (d) Scalability.

Figure 10: Throughput and CPU Overheads.

Latency. We measure the latency of BURST, RXE, and hardware RDMA (BF3mini) under varying message sizes with *perftest*. As shown in Fig. 11, BURST performs a low network latency and has no significant long-tail latency, which is close to hardware RDMA in R2R scenario. When facing the same NR2R scenario, BURST saves 30% to 92.4% of time overhead compared to RXE. BURST transfers a 1M message in just 99.41μs, and its p999 latency only increases by 7.93μs compared to its average latency. In contrast, RXE needs 1118.89μs, and its tail latency increases by 1019.48μs.

Furthermore, all three stacks adhere to the unified maximum transmission unit (MTU) of the RDMA protocol (4KB). As observed from the latency curve of RXE, when a software stack handles a transmission request larger than the MTU, it faces packet processing delays due to fragmentation. As a result, RXE's latency shows a linear increase from the 4KB message size. In addressing this common challenge for software protocol stacks, we employed engineering optimization methods such as batching send during the implementation of BURST. Therefore, BURST still incurs a progressive latency increase of 9.3–71.8μs due to MTU fragmentation overhead.

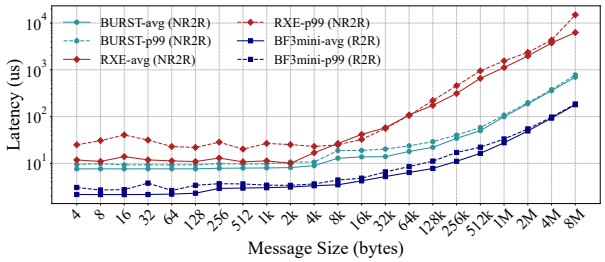


Figure 11: Latency Comparison.

5.2 Accelerating App-level Communication

Hardware RDMA is unavailable in the evaluated app-level NR2R scenario due to architectural constraints, and Sec. 5.1 has shown that RXE underperforms TCP in 400G network benchmark experiments; thus, we use TCP as the baseline.

LLM Inference. To demonstrate BURST's efficacy in real-world AI applications, we conducted an initial grayscale test within an enterprise Collective Communication Library (CCL), and focus on the *time to first token (TTFT)* metric, which is a critical latency indicator for LLM inference and captures the end-to-end delay from request submission to generating the first output token. Our experiments are conducted on a two-node NR2R setup: the RNIC side is equipped with 4 Bluefield-3 NICs connected to 8 high-performance GPUs (interconnected via NVLink), while the NR side deploys 4 400Gbps NICs paired with 8 high-performance GPUs, with both nodes connected to the same ToR switch.

As depicted in Fig. 12, we evaluate the communication latency of *send key-value cache* operations within TTFT, as described in Appendix A.4, with an average transfer size of 2.94K tokens' KVCache. The results show that BURST achieves a remarkable performance improvement in transfer latency. On average, BURST reduces KVCache transfer latency from 16.9 ms to 4.26 ms (3.97×), while lowering the P99 latency from 111 ms to 24.2 ms. In addition to KVCache transfer latency, we also report the overall TTFT improvement.

With BURST, the average TTFT is reduced from 1190 ms to 931 ms, achieving a 21.76% reduction, while the P99 TTFT decreases from 3720 ms to 3210 ms (13.7% reduction).

Storage System. We deploy BURST between storage nodes in a production enterprise distributed file system to accelerate network transport on commodity Ethernet NICs. Table 5.2 shows the performance of a representative *GetAttr* operation. In these scenarios, BURST achieves a 27.6% to 47.4% performance improvement compared to KTCP, demonstrating that BURST's communication efficiency directly translates into tangible application-level benefits.

Metric		KTCP	BURST	Gain
Req-lat (μs)	1st-hop-avg	42.47	30.73	27.6%
	E2E-avg	52.4	37.6	28.2%
	E2E-p99.9	747	393	47.4%
Sys-tpt (QPS)	I/O Depth = 1	18.6K	25.9K	39.0%
	I/O Depth = 2	34.6K	49.8K	43.0%
	I/O Depth = 8	48.7K	69.5K	43.0%

Table 1: Performance comparison in an enterprise storage system. *Evaluating request latency (Req-lat) at an I/O depth of 1 and system throughput (Sys-tpt), measured in Queue Pairs per second, for concurrent workloads (I/O depth > 1)*

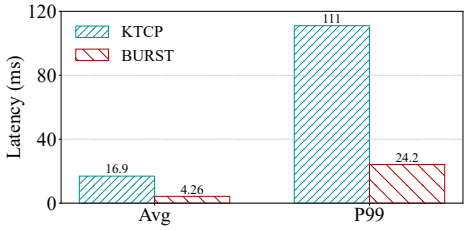
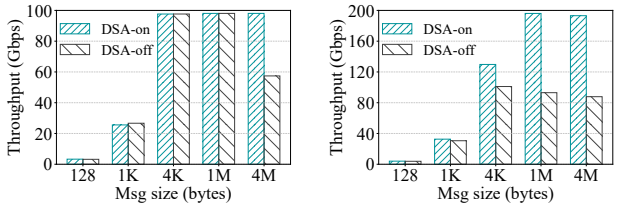


Figure 12: BURST for LLM Inference Workloads.

5.3 Performance with DSA

In this section, we evaluate BURST with DSA in Testbed2. As described in Sec. 4.4.2, DSA is used to offload memory copying on the RECV side. BURST runs on both client and server for unidirectional and bidirectional tests, with the *perftest* and BURST processes each pinned to one CPU core. Under this CPU-bound setup, CPU utilization differs by less than 1% with or without DSA; therefore, we use throughput as the primary metric to quantify DSA's performance benefits.

Fig. 13(a) and 13(b) illustrate that when the packet length is less than 4K, the CPU consumption on the client side (transmission direction) is greater than on the server side (reception direction). Since DSA is currently only applied to the RX direction, the performance improvement of DSA is not evident for packet lengths less than 4K. When the packet length is equal to 4K, the CPU consumption on the client side and server side is roughly balanced. During the actual test, increasing the test program to two processes did not improve throughput, which suggests that the bottleneck is in the forwarding CPU at this point. When the packet length is greater than 4K, the CPU bottleneck remains on the client side sending the test stream. Increasing the test program to two processes (resulting in higher CPU usage) leads to increased throughput, indicating that the bottleneck is in the test program at this point. From these observations, it can be concluded that after applying DSA, BURST can maintain stable throughput at the NIC line rate starting from a packet length of 4K.

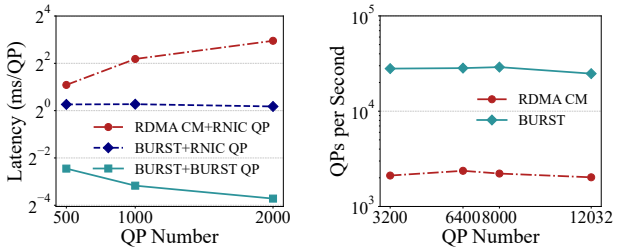


(a) Single-Direction Test. (b) Dual-Direction Test.

Figure 13: DSA-accelerated Throughput Comparison.

5.4 Connection Setup with User-space CM

Single-thread scenarios. In this part, we compare BURST with the original RDMA CM. We use the existing tools in *cmtime* to test connection setup speed, setting the affinity to a single CPU core. Our tests measured the total time required to establish 500, 1000, and 2000 connections, respectively, and calculated the average time to establish each connection.



(a) Single-Thread. (b) Multi-Thread.

Figure 14: Connection Setup Performance.

Fig. 14(a) shows the average time to establish each connection (ms/QP), where the performance trends diverge as the connection count grows. The average setup time for kCM increases, which can be attributed to more cache misses on the RNIC's limited on-chip resources. In contrast, BURST's average time per QP decreases because its initial software overhead is amortized over a larger number of connections, allowing it to benefit from ample host memory. This efficiency allows BURST to save 33%-36.7% of the setup time even when managing hardware RNIC QPs. When using its own software QPs, BURST further eliminates hardware interaction overhead, saving 95% of the connection setup time compared to RDMA CM.

Multi-thread scenarios. We evaluate BURST with an enterprise RPC library to assess its performance in large-scale, multi-threaded environments. We simulated tens of thousands of connections on two machines (detailed calculation in Appendix A.2). As shown in Fig. 14(b), when scaling from

thousands to tens of thousands of concurrent connection requests, BURST's connection rate (QPS) is consistently 12× that of RDMA CM. BURST maintains a stable rate of establishing 24,792 connections per second, even as the number of connections scales to tens of thousands.

6 Related Work

RNIC-to-RNIC (R2R) Solutions. Recent efforts have focused on enabling interoperability between commercial RNICs from different vendors and generations. These solutions often leverage the RDMA Unreliable Connection (UC) service type to implement advanced customizable network features such as congestion control and selective retransmission through a combination of software and hardware. For instance, Flor [29] and UCCL [57] demonstrate effective deployment across heterogeneous RNICs and support the UC service type. And another work from Azure [12] presents an industrial-scale study on deploying RDMA in large cloud storage systems, highlighting practical challenges in achieving interoperability across RNIC generations and the necessity of careful system-level engineering. BURST, which also supports the UC service type, is orthogonal to these existing solutions, as it focuses on enabling efficient NR2R communication between RNIC and non-RNIC endpoints. Flor often fails when attempting to interoperate with legacy or novel non-standard RoCE NICs. Furthermore, while some kernel-level solutions exist, they suffer from excessive kernel dependency, such as SMC-R [18] and L5 [16].

Non-RNIC-to-Non-RNIC (NR2NR) Solutions. Another line of research has explored high-speed communication among non-RNICs, which also demonstrates high performance but often struggles with cost-effective scalability in large-scale hybrid networks. Specifically, SNAP [33] implement an RDMA-like service, which is incompatible with the standard RDMA Verbs API, preventing interoperability with commercial RNICs and thus exacerbating network isolation. Solar [35] relies on a hardware-software co-design, which involves long hardware upgrade cycles. Moreover, UCCL-XDP [57], a plugin for the NCCL [42] library, incurs potential overhead by requiring data copies via `cudaMemcpy()` for packet reassembly when using its AF_XDP [51] backend. Furthermore, AF_XDP has been shown to underperform DPDK in small-message processing scenarios [36].

User-space TCP Solutions for NR2R. As discussed in Sec. 2.2, state-of-the-art kernel-level TCP implementations face significant limitations in Non-RDMA-to-RDMA (NR2R) scenarios. In recent years, a number of user-space TCP solutions have emerged, including mTCP [27], IX [13], LUNA [59], VMA [10], and Demikernel [56]. These works have shown remarkable performance improvements in traditional applications built upon the TCP protocol. However, for high-speed network applications developed with RDMA, the inherent interface inconsistencies between these user-space TCP stacks and RDMA verbs introduce substantial development and migration costs, which are critical concerns.

Connection Setup Acceleration. KRCore [55] leverages Dynamically Connected Transport (DCT) [6] to pre-establish shared connection pools, effectively mitigating connection setup latency and eliminating data plane overhead. However, its reliance on vendor-specific RDMA NICs (RNICs) and proprietary Dynamically Connected (DC) Verbs APIs [1] restricts its generalizability. SECM [22] employs multi-threading within applications to handle complex connection setup processes, but it lacks user transparency and still leaves room for performance optimization in kernel space.

7 Discussion

Deployment and Optimization of BURST in Diverse Scenarios. We have found that as a software protocol stack, BURST can be deployed in a wider range of scenarios with more comprehensive optimizations in a production environment. For instance, we have implemented SACK-based retransmission in software, following the retransmission rules consistent with [54] (Appendix A.3). Then, BURST also supports direct integration with RDMA standard interface-based applications through the LibOS mode, allowing hardware access directly from application threads. (Appendix A.6)

Evolution Potential of BURST. Unlike hardware with fixed on-chip resources, BURST's current CPU and cache bottlenecks can be mitigated through software techniques like advanced memory virtualization. This indicates that BURST faces no insurmountable limitations and holds significant potential for further optimization.

8 Conclusion

In this paper, we introduced BURST, a novel user-space software RDMA stack designed to address the critical interoperability challenge between non-RNIC and RNIC endpoints in modern heterogeneous data centers. Driven by application disaggregation, infrastructure evolution, and protocol innovation, the need for a high-performance NR2R communication fabric has become increasingly urgent. BURST bridges this gap by providing a solution that is deployable on Ethernet hardware without requiring any application modification.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported by the National Natural Science Foundation of China (U24B20150, 62222204), the Major Project of Changsha Science and Technology Plan (kh2401005), the Major Project of Shenzhen Science and Technology Plan (KJZD20240903100813017), and the collaborative project between Hunan University and ByteDance.

References

[1] Dynamically connected qps. https://docs.nvidia.com/networking/display/rdmacore50/dynamically+connected+(dc)+qps.

[2] Falcon: A reliable and low latency hardware transport. https://netdevconf.info/0x18/docs/netdev-0x18-paper43-talk-slides/Introduction%20to%20Falcon%20Reliable%20Transport.pdf. 2024.

[3] Infinibandtm architecture specification volume 1. https://www.afs.enea.it/asantoro/V1r1_2_1.Release_12062007.pdf. 2007.

[4] Intel dpdk. https://www.dpdk.org/.

[5] perftest: Rdma performance testing tools. https://github.com/linux-rdma/perftest. Accessed: 2023-10-10.

[6] Dynamically connected transport. https://www.openfabrics.org/images/eventpresos/workshops2014/DevWorkshop/presos/Monday/pdf/05_DC_Verbs.pdf, 2013.

[7] Cmtime - rdma cm connection steps timing test. https://man7.org/linux/man-pages/man1/cmtime.1.html, 2017.

[8] Howto configure softroce. https://enterprise-support.nvidia.com/s/article/howto-configure-soft-roce, 2022.

[9] *Device Memory TCP*, July 2023.

[10] ACCELERATOR, M. M. Mellanox messaging accelerator.

[11] ALIZADEH, M., GREENBERG, A., MALTZ, D., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Dctcp: Efficient packet transport for the commoditized data center.

[12] BAI, W., ABDEEN, S. S., AGRAWAL, A., ATTRE, K. K., BAHL, P., BHAGAT, A., BHASKARA, G., BROKHMAN, T., CAO, L., CHEEMA, A., ET AL. Empowering azure storage with {RDMA}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 49–67.

[13] BELAY, A., PREKAS, G., KLIMOVIC, A., GROSSMAN, S., KOZYRAKIS, C., AND BUGNION, E. {IX}: a protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), pp. 49–65.

[14] CORPORATION, I. Intel data streaming accelerator (intel dsa) architecture specification, 2020. Accessed: 2023-10-01.

[15] DPDK.ORG. Unleashing network performance with microsoft azure mana and dpdk. https://www.dpdk.org/unleashing-network-performance-with-microsoft-azure-mana-and-dpdk/, 2024. Accessed: 2024-10-27.

[16] FENT, P., VAN RENEN, A., KIPF, A., LEIS, V., NEUMANN, T., AND KEMPER, A. Low-latency communication for fast dbms using rdma and shared memory. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)* (2020), IEEE, pp. 1477–1488.

[17] FIRESTONE, D., PUTNAM, A., MUNDKUR, S., CHIOU, D., DABAGH, A., ANDREWARTHA, M., ANGEPAT, H., BHANU, V., CAULFIELD, A., CHUNG, E., ET AL. Azure accelerated networking:{SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 51–66.

[18] FOX, M., KASSIMIS, C. G., AND STEVENS, J. IBM's Shared Memory Communications over RDMA (SMC-R) Protocol. RFC 7609, Aug. 2015.

[19] GAO, Y., LI, Q., TANG, L., XI, Y., ZHANG, P., PENG, W., LI, B., WU, Y., LIU, S., YAN, L., FENG, F., ZHUANG, Y., LIU, F., LIU, P., LIU, X., WU, Z., WU, J., CAO, Z., TIAN, C., WU, J., ZHU, J., WANG, H., CAI, D., AND WU, J. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)* (Apr. 2021), USENIX Association, pp. 519–533.

[20] GIBSON, D., HARIHARAN, H., LANCE, E., MCLAREN, M., MONTAZERI, B., SINGH, A., WANG, S., WASSEL, H. M., WU, Z., YOO, S., ET AL. Aquila: A unified, low-latency fabric for datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (2022), pp. 1249–1266.

[21] GUO, C., WU, H., DENG, Z., SONI, G., YE, J., PADHYE, J., AND LIPSHTEYN, M. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference* (New York, NY, USA, 2016), SIGCOMM '16, Association for Computing Machinery, p. 202–215.

[22] GUO, X., CHEN, G., ZHAN, X., QU, T., AND HAN, Z. Secm: Securely and efficiently connections setup using rdma-cm. *Computer Networks 250* (2024), 110541.

[23] HU, C., HUANG, H., HU, J., XU, J., CHEN, X., XIE, T., WANG, C., WANG, S., BAO, Y., SUN, N., ET AL. Memserve: Context caching for disaggregated llm serving with elastic memory pool. *arXiv preprint arXiv:2406.17565* (2024).

[24] HUANG, H., ZHANG, J., CHEN, X., SONG, Z., QIN, J., AND WANG, Z. {SwCC}:{Software-Programmable} and {Per-Packet} congestion control in {RDMA} engine. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)* (2025), pp. 1243–1260.

[25] INTEL. Production brief for intel® ethernet controller e810-cam2/cam1/xxvam2, 2020. Accessed: 2023-10-03.

[26] IPERF2 PROJECT. iPerf2 Network Performance Measurement Tool. Available at https://sourceforge.net/projects/iperf2/, 2024. Accessed: 2024-10-27.

[27] JEONG, E., WOOD, S., JAMSHED, M., JEONG, H., IHM, S., HAN, D., AND PARK, K. {mTCP}: a highly scalable user-level {TCP} stack for multicore systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (2014), pp. 489–502.

[28] KAFFES, K., CHONG, T., HUMPHRIES, J. T., BELAY, A., MAZIÈRES, D., AND KOZYRAKIS, C. Shinjuku: Preemptive scheduling for {μsecond-scale} tail latency. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)* (2019), pp. 345–360.

[29] LI, Q., GAO, Y., WANG, X., QIU, H., LE, Y., LIU, D., XIANG, Q., FENG, F., ZHANG, P., LI, B., ET AL. Flor: An open high performance {RDMA} framework over heterogeneous {RNICs}. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)* (2023), pp. 931–948.

[30] LIAO, H., LIU, B., CHEN, X., GUO, Z., CHENG, C., WANG, J., CHEN, X., DONG, P., MENG, R., LIU, W., ZHOU, Z., ZHANG, Z., GAI, Y., QIAN, C., XIONG, Y., CHENG, Z., XIA, J., MA, Y., CHEN, X., DU, W., XIAO, S., LI, C., QIN, Y., XIONG, L., YU, Z., CHEN, L., CHEN, L., WANG, B., WU, P., GAO, J., LI, X., HE, J., YAN, S., AND MCCOLL, B. Ub-mesh: a hierarchically localized nd-fullmesh datacenter network architecture, 2025.

[31] MA, T., MA, T., SONG, Z., LI, J., CHANG, H., CHEN, K., JIANG, H., AND WU, Y. X-rdma: Effective rdma middleware in large-scale production environments. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)* (2019), pp. 1–12.

[32] MARHUBI, K. Some early linux ipc latency data, Jun 2015.

[33] MARTY, M., DE KRUIJF, M., ADRIAENS, J., ALFELD, C., BAUER, S., CONTAVALLI, C., DALTON, M., DUKKIPATI, N., EVANS, W. C., GRIBBLE, S., ET AL. Snap: A microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (2019), pp. 399–413.

[34] MIAO, M., REN, F., LUO, X., XIE, J., MENG, Q., AND CHENG, W. Softrdma. In *Proceedings of the First Asia-Pacific Workshop on Networking* (Aug 2017), p. 43–49.

[35] MIAO, R., ZHU, L., MA, S., QIAN, K., ZHUANG, S., LI, B., CHENG, S., GAO, J., ZHUANG, Y., ZHANG, P., ET AL. From luna to solar:

the evolutions of the compute-to-storage networks in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 753–766.

[36] MOLÈ, M., SHAHINFAR, F., TRANQUILLO, F. M., ZONI, D., PANDA, A., AND ANTICHI, G. Performance implications at the intersection of af_xdp and programmable nics. In *Proceedings of the 3rd Workshop on EBPF and Kernel Extensions* (New York, NY, USA, 2025), eBPF '25, Association for Computing Machinery, p. 1–7.

[37] NVIDIA. *ConnectX-6 Dx Datasheet*, 2023.

[38] NVIDIA. *ConnectX-7 Datasheet*, 2023. Accessed: 2023-10-03.

[39] NVIDIA. Nvidia bluefield.

[40] NVIDIA. *ConnectX-8 Datasheet*, 2025.

[41] NVIDIA CORPORATION. Scaling zero-touch roce technology with round-trip time congestion control, 2023. Accessed: 2023-10-10.

[42] NVIDIA CORPORATION. NVIDIA Collective Communications Library (NCCL). GitHub repository, 2025. Accessed: 2025-10-27.

[43] PATEL, P., CHOUKSE, E., ZHANG, C., SHAH, A., GOIRI, Í., MALEKI, S., AND BIANCHINI, R. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* (2024), IEEE, pp. 118–132.

[44] PENG, Y., WEI, H., ZHONG, X., HUANG, J., XU, H., WANG, Z., BAI, Y., JIANG, Z., YE, J., WANG, X., ET AL. Barre: Empowering simplified and versatile programmable congestion control in {High-Speed}{AI} clusters. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)* (2025), pp. 343–363.

[45] QIN, R., LI, Z., HE, W., ZHANG, M., WU, Y., ZHENG, W., AND XU, X. Mooncake: A kvcache-centric disaggregated architecture for llm serving, 2024.

[46] QUINNELL, E. Tesla transport protocol over ethernet (ttpoe): A new lossy, exa-scale fabric for the dojo ai supercomputer. In *2024 IEEE Hot Chips 36 Symposium (HCS)* (2024), IEEE Computer Society, pp. 1–23.

[47] RAJA, R. AMAZON ELASTIC FABRIC ADAPTER: ANATOMY, CAPABILITIES, AND THE ROAD AHEAD. Presented at the OpenFabrics Alliance Workshop 2019, 2019.

[48] ROCE INITIATIVE. SoftRoCE: A Practical Guide for Implementing RoCE in Software. Whitepaper, 2016.

[49] SHALEV, L., AYOUB, H., BSHARA, N., AND SABBAG, E. Supercomputing on nitro in aws cloud. *IEEE Micro, vol. PP* (2020), 1–1.

[50] SINGHVI, A., AKELLA, A., GIBSON, D., WENISCH, T. F., WONG-CHAN, M., CLARK, S., MARTIN, M. M., MCLAREN, M., CHANDRA, P., CAUBLE, R., 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* (2020), pp. 708–721.

[51] THE LINUX KERNEL DEVELOPMENT COMMUNITY. AF_XDP. Linux Kernel Documentation, 2025. Accessed: 2025-09-18.

[52] THE LINUX MAN-PAGES PROJECT. *eventfd – create a file descriptor for event notification*. The Linux Kernel development community, 2025. Accessed: 2025-09-19.

[53] WANG, J., BEHRENS, D., FU, M., OBERHAUSER, L., OBERHAUSER, J., LEI, J., CHEN, G., HÄRTIG, H., AND CHEN, H. BBQ: A block-based bounded queue for exchanging data and profiling. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)* (Carlsbad, CA, July 2022), USENIX Association, pp. 249–262.

[54] WANG, Z., LUO, L., NING, Q., ZENG, C., LI, W., WAN, X., XIE, P., FENG, T., CHENG, K., GENG, X., ET AL. {SRNIC}: A scalable architecture for {RDMA}{NICs}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 1–14.

[55] WEI, X., LU, F., CHEN, R., AND CHEN, H. Krcore: a microsecond-scale rdma control plane for elastic computing.

[56] YANG, J., YUE, Y., AND RASHMI, K. A large-scale analysis of hundreds of in-memory key-value cache clusters at twitter. *ACM Transactions on Storage (TOS) 17*, 3 (2021), 1–35.

[57] ZHOU, Y., CHEN, Z., MAO, Z., LAO, C., YANG, S., KANNAN, P. G., GAO, J., ZHAO, Y., WU, Y., YOU, K., REN, F., XU, Z., RAICIU, C., AND STOICA, I. An extensible software transport layer for gpu networking, 2025.

[58] ZHU, Y., ERAN, H., FIRESTONE, D., GUO, C., LIPSHTEYN, M., LIRON, Y., PADHYE, J., RAINDEL, S., YAHIA, M. H., AND ZHANG, M. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 523–536.

[59] ZHU, Y., ET AL. Deploying user-space tcp at cloud scale with luna. In *Proceedings of the 2023 USENIX Annual Technical Conference (ATC)* (2023), pp. 123–135.

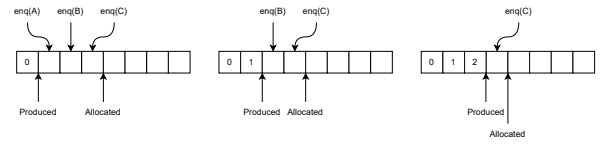
A Appendix

A.1 Details for Shared Doorbell Queue

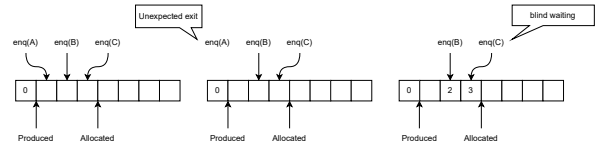
A.1.1 Different Implementation and Evaluation

This entire notification process, therefore, needs to be built in software using Inter-Process Communication (IPC). **Different Solutions and Our Analysis.** Drawing on the native RDMA Doorbell mechanism and recent work in IPC, we explored several solutions: (1) One straightforward approach we considered was **per-QP polling**. This method mirrors hardware RDMA by having a separate Doorbell Queue (DBQ) for each Queue Pair (QP). While simple to set up, we found that it does not scale well. As Fig. 3 shows, the software processing quickly becomes a bottleneck even with over than 4096 QPs. (2) A second option we looked at involved using **kernel-based notification**. This means using features like `eventfd` and `epoll` to tell the BURST process when something happens. However, this brings back the reliance on the operating system kernel, which adds overhead; a single `eventfd` operation can introduce up to 4μs of delay [32]. (3) Based on these findings, we chose our final design: a **global shared DBQ**. This solution involves creating one shared DBQ in memory. We found that this provides a scalable way for the independent BURST process to handle multiple threads and keep communication separate for different applications. Table 2 shows that we eliminate different implementations of shared DBQ, therefore choosing the most efficient one.

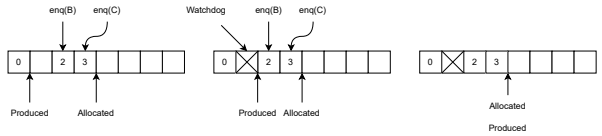
A.1.2 The Crash Not Safety issue



(a) Normal Multi-Producer Enqueue Process.



(b) Deadlock Scenario in MPSC Queue with Process Exit.



(c) Proposed Deadlock Recovery Mechanism.

Figure 15: Enhanced Doorbell Queue operations and our deadlock-free details.

We present the details of our proposed enhanced Doorbell queues, addressing the critical issue of deadlocks that can arise from process failures.

For instance, when three processes (A, B, and C) attempt to enqueue, they first each allocate a block and update the allocated pointer. These three processes then sequentially fill their data and move their produced pointer, as shown in Fig. 15(a), smoothly completing a parallel operation in the shared queue. However, a deadlock scenario will occur if a producer process exits abnormally. As shown in Fig. 15(b), process A crashes after allocating but before moving the produced pointer. Processes B and C are then unable to wait for the produced pointer to advance to their respective positions, and the queue enters a deadlock state. To mitigate this issue, we incorporate a watchdog thread with a timeout mechanism, as depicted in Fig. 15(c). The watchdog thread periodically checks for a deadlock by monitoring the produced and allocated pointers. If these pointers are unchanged and inconsistent for a prolonged period, the thread intervenes by invalidating the data from the failed process and advancing the produced pointer to its expected position. This proactive recovery mechanism restores the MPSC queue to a functional state, preventing the deadlock and maintaining system availability.

A.2 Experimental Setup for Multi-Threaded Connection Benchmark

The evaluation of our user-space Connection Manager in a large-scale, multi-threaded scenario, presented in Sec. 5.4, was conducted using a testbed based on an enterprise RPC framework.

The test involved two machines (a client and a server), with 8 worker threads configured on each. To simulate a high-density, many-to-one connection pattern common in production services, we created an N:1 incast scenario. This was achieved by varying the `server_num_per_thread` parameter on the server side. The data points shown in Fig. 14(b) correspond to `server_num_per_thread` values of 50, 100, 125, and 188.

We define the variables for calculating the total connections as follows: T_{conn} represents the total connections, S_{pt} is the number of server instances per thread (`server_num_per_thread`), N_{st} is the number of server threads, and N_{ct} is the number of client threads. The total is calculated using Equation 1:

$$T_{conn} = S_{pt} \times N_{st} \times N_{ct} \quad (1)$$

A.3 Supporting SACK with known selective repeat approach

BURST facilitates interoperability with RDMA NICs featuring known SACK implementations [54], with results confirming significantly superior performance compared to both

	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads	128 Threads
DPDK Ringbuf	2.4 ms	4.8 ms	9.8 ms	24.3 ms	126.8 ms	624 ms	679.6 ms	
Mutex	2.4 ms	4.2 ms	8.0 ms	21.8 ms	62.7 ms	152.0 ms	288.3 ms	
Multi-Producer Enqueue	4.8 ms	5.6 ms	6.8 ms	8.8 ms	12.7 ms	22.1 ms	37.5 ms	74.8 ms

Table 2: Performance Comparison of Different Implementations for a Shared Doorbell Queue. (*Multiple producer threads each enqueue 10,000 items, which are processed by a single consumer thread. We record the total completion latency.*)

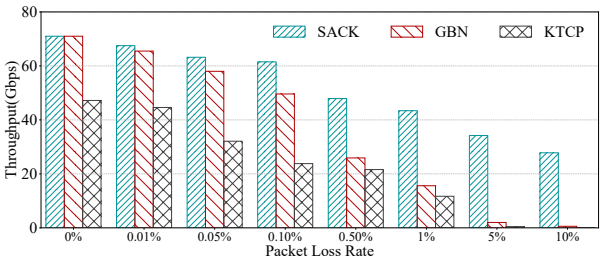


Figure 16: Throughput under different packet loss rates.

Go-Back-N (GBN) and TCP's fast retransmission under different packet loss rates, as shown in Fig. 16. Furthermore, in pure software deployments (e.g., both endpoints running BURST), we implement selective repeat via a lightweight `seth` extension, as described above, significantly improving retransmission efficiency.

A.4 KVCache Transfer and TTFT

The time to first token (TTFT) metric consists of multiple components, including prompt processing, key-value cache (KVCache) communication, and first-token generation. In our evaluation, the KVCache communication latency corresponds to the data transfer stage between distributed inference workers. Although KVCache communication does not dominate the entire TTFT, it lies on the critical path of LLM inference and directly affects the availability of the first generated token, especially under large-context workloads. As a result, the reduction in KVCache communication latency achieved by BURST contributes directly to the end-to-end TTFT improvements reported in Fig. 12.

A.5 Background on Intel DSA

Intel Data Streaming Accelerator (DSA) is a programmable hardware DMA engine designed to offload memory copy and data movement operations from the CPU. It supports asynchronous submission of copy descriptors and can overlap data movement with computation, thereby reducing CPU utilization for memory-intensive workloads. In BURST, DSA is used to offload unavoidable data copies on the receive path, where incoming packets stored in NIC-managed buffers must be copied into application-visible memory before completion notification. By delegating these copy operations to DSA, BURST reduces CPU overhead without changing RDMA semantics or relying on hardware-specific packet processing features.

A.6 LibOS mode for BURST

BURST also supports direct integration with RDMA standard interface-based applications through the LibOS mode, allowing hardware access directly from application threads. In this mode, BURST can achieve ultra-low latency; however, it typically relies on spin-polling application threads. Therefore, this mode is generally suitable for high-performance requirements, mostly in scenarios where a single application exclusively occupies the server. The above thread model pertains to the independent thread mode of BURST. For the LibOS mode, each PMD thread not only includes the BURST protocol stack logic but also runs the user's application logic within the same thread.

A.7 Single-QP Performance

To further understand the fundamental performance characteristics of BURST, we conduct the single-QP performance evaluation using the identical experimental setup detailed in Sec. 5.1. As shown in Table 3, while the BF3mini hardware NIC scales linearly to near line-rate (∼385 Gbps) with large message sizes, BURST saturates at approximately 121 Gbps. It is worth noting that in production environments, workloads typically multiplex traffic across a large number of QPs. The performance ceiling observed in this single-QP microbenchmark is primarily attributed to the architectural constraints of the underlying packet processing framework. Since BURST is built upon DPDK, it adheres to a run-to-completion model where a single QP is mapped to a specific hardware Rx/Tx queue. Due to the programming paradigm, this queue is strictly bound to a single CPU core. Consequently, the throughput is limited by the packet processing capability of that single core (which exhibits ∼60% utilization in our tests), rather than the total available PCIe or network bandwidth.

Message Size	BF3mini (Gbps)	BURST (Gbps)
16 Bytes	0.61	0.36
64 Bytes	2.43	1.43
256 Bytes	9.68	5.72
1 KB	38.8	22.9
4 KB	154.7	116.0
16 KB	351.4	119.6
1 MB	385.4	121.9
4 MB	385.5	121.5
8 MB	385.6	121.5

Table 3: Throughput comparison details (Single-QP).