Efficient and Flexible Datapaths for Fine-Grained Rack-Scale Interconnects with *Elastic QP*

Chenxingyu Zhao
University of Washington
Seattle, Washington, USA
cxyzhao@cs.washington.edu

Yibo Wu*
University of Wisconsin-Madison
Madison, Wisconsin, USA
wu668@wisc.edu

Hongtao Zhang
University of Washington
Seattle, Washington, USA
hongtaoz@cs.washington.edu

Jaehong Min
University of Washington
Seattle, Washington, USA
jaehongm@cs.washington.edu

Ming Liu
University of Wisconsin-Madison
Madison, Wisconsin, USA
mgliu@cs.wisc.edu

Arvind Krishnamurthy
University of Washington
Seattle, Washington, USA
arvind@cs.washington.edu

Abstract

Rack-scale interconnects serve as critical datapaths for emerging communication-intensive systems to scale up. Innovative solutions for this datapath are rising at a rapid pace, especially those based on Ethernet. However, existing hardware-based solutions, such as RDMA, face performance issues, particularly for small-message memory access, and suffer from the inflexibility of hardware-fixed processing. The community is actively pursuing efficient, flexible, and cost-effective rack-scale datapaths.

In this work, we propose Software-Interposed Datapath (SID), an efficient, software-flexible, and low-cost solution for rack-scale interconnects, particularly optimized for fine-grained memory access. Improving small-message efficiency is a well-known challenge, and software involvement for flexibility seems to amplify the performance hurdle further. SID boosts performance by exploiting one insight: existing NICs primarily rely on Queue Pair (QP)-level parallelism, but underutilize intra-QP Work Queue Element (WQE)-level parallelism. Harnessing parallelism is non-trivial, especially at the WQE-level, due to ordering semantics and request dispatching. The key technique is our Elastic QP data structure built atop the on-NIC datapath processors, which realizes ordered intra-QP parallelism while minimizing coordination overhead. Regarding flexibility, SID supports extensible operation sets that comply with the OpenSHMEM model for ML/HPC workloads and the Message Queue model for cloud service workloads. Regarding cost efficiency, SID is built on top of commodity components such as Ethernet, PCIe, and datapath cores of NVIDIA ConnectX-8 and BlueField-3 NICs. Evaluation shows that SID achieves up to 11.03x higher rates for small messages than RDMA-based baselines and supports both CPU and GPU-Direct operations.

*Work done while at the University of Washington.

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/2026/08
https://doi.org/10.1145/3789240.3829160

CCS Concepts

• **Hardware → Networking hardware**; • **Networks → Network adapters**; *Data center networks*; • **Computer systems organization → Interconnection architectures**.

Keywords

Scale-Up Networks, Memory-Semantic Interconnects, SmartNICs, RDMA, GPU Communication

ACM Reference Format:
Chenxingyu Zhao, Yibo Wu, Hongtao Zhang, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2026. Efficient and Flexible Datapaths for Fine-Grained Rack-Scale Interconnects with *Elastic QP*. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3789240.3829160

1 Introduction

Rack-scale interconnects, as essential datapaths, empower distributed communication-intensive systems to scale up, which benefits emerging workloads such as big data analytics [1–3], HPC [4, 5], and ML training and inference [6–10]. Fundamentally, rack-scale computing sits at a *sweet middle spot* in networked systems: It is larger than a single server, thereby enabling the composition of more devices (*e.g.,* GPUs); Meanwhile, it is smaller than cluster-scale systems, with a simpler interconnect topology that makes intensive fine-grained communication more feasible (*e.g.,* memory-semantic OpenSHMEM [11, 12]).

As rack-scale datapaths gain importance, innovative solutions are blooming, especially Ethernet-based ones, including ongoing initiatives such as OCP Ethernet for Scale-Up Networking (ESUN/SUE) [13] and UEC [14], as well as existing transports such as RDMA RoCEv2 [15–18], Falcon [19, 20], and SRD [21]. Among these, RDMA is commodity-ready and widely deployed [7, 22, 23]; however, it faces two key limitations when applied to rack-scale interconnects. First, RDMA suffers from inflexibility because its logic is baked into ASICs. Second, and more critically, RDMA is not efficient for fine-grained, small-message communication, as measured in §3.1. Notably, our measurements suggest that the root cause of RDMA's limitation is *not* insufficient NIC processing power (*e.g.,* upcoming 800GbE), but rather a lack of an efficient manner to exploit the *intra-Queue Pair (QP)* parallelism to fully harness the Ethernet capability.

The opportunity of intra-QP parallelism arises from a common pattern in fine-grained rack-scale workloads, which we refer to as *Remote Memory Access (RMA) Queue Buildup*. As noted in the ESUN/SUE specification (§2.2), local memory bandwidth typically exceeds network bandwidth, so accessing remote memory with local/remote-agnostic intensity leads to significant queue buildup. Beyond this, several workload factors contribute to the rapid buildup of this queue (§4.1). In this context, queue buildup is not a drawback, as commonly thought. Instead, it creates opportunities for parallel processing as discussed later.

Inspired by classical techniques in CPU architecture, instruction-level parallelism (ILP) and thread-level parallelism (TLP) [24], we propose intra-QP Work Queue Element **(WQE)-level parallelism (WLP)**. In current RDMA systems, applications typically use multiple QPs to achieve high throughput, that is, utilizing QP-level parallelism (QLP). The motivation of ILP is that applications can transparently benefit from parallelism without explicitly managing multiple instruction streams, as hardware absorbs the complexity of parallelism and ordering. In contrast, TLP pushes applications to manage multiple threads. Similarly, we advocate for WLP to shift ordering and parallelism management to the NIC, enabling transparent parallel execution within a single QP, unlike QLP, which requires applications to manage multiple QPs and reason about ordering across them in a stateful manner. Just as ILP and TLP coexist in the CPU, WLP and QLP are orthogonal and can be jointly leveraged. However, current NICs primarily rely on QLP, leaving WLP largely underutilized.

Along with the motivation of WLP, another key enabler is the on-NIC *interposition processor*. The NIC serves as a unique chokepoint bridging intra-host and inter-host interconnects. Moreover, emerging RDMA NICs (RNICs) and SmartNICs embed on-path programmable processors. For instance, NVIDIA ConnectX-8 and BlueField-3 integrate a Datapath Accelerator (DPA), a multi-core processor with the low-profile RISC-V [25]. We highlight three key capabilities of these interposition processors: 1) load/store access to host-side memory, 2) programmable cores coupled with on-NIC memory, and 3) tight integration with the Ethernet MAC. These capabilities lay a foundation for enabling new datapath designs, particularly for exploiting WLP. Additionally, these capabilities are neither demanding nor vendor-specific; similar mechanisms appear across many platforms [26–30]. We choose DPAs primarily for their broad accessibility.

In this work, we design **Software-Interposed Datapath (SID)**, an efficient, flexible, and low-cost solution for rack-scale interconnects. The key idea of SID is to exploit WLP via the on-NIC interposition processors. It allows applications to transparently use a single QP without breaking FIFO ordering semantics and without perceiving underlying parallel processing. This property is attractive, but it also raises non-trivial challenges: How can we preserve ordering semantics and efficiently dispatch batches of requests to the parallel processing threads on the interposition processor?

Elastic QP (E-QP) (§5) is the core data structure we design to address the above challenges. The rationale of E-QP for achieving WLP is as follows: As defined by OpenSHMEM routines (§2), although requests are submitted in FIFO order, the bulk of requests between two synchronization points (*e.g.,* `barriers`) typically do not require strict execution order. This allows the datapath to execute them in parallel (*i.e.,* without ordering constraints) and combine many requests into one, as long as completion signals are committed in the original submission order. This follows the same spirit as CPU's out-of-order instruction execution with in-order commits [24, 31]. As a result, applications perceive execution as sequential while transparently benefiting from the performance gains of parallel processing. Besides the ordering challenge, E-QP improves dispatching efficiency and minimizes the coordination overhead of threads. Beyond the TX side, E-QP is also applicable to the RX side.

FlexOp Engine (§5.5) is another component of SID to enhance flexibility. SID is built atop multi-core interposition processors. With software programmability, the FlexOp Engine exposes a comprehensive set of operations beyond basic data transfer, including support for synchronization, atomic operations, collective communication, time-sensitive operations, and more. To ensure cost efficiency, we perform processing via low-profile interposition processors while remaining compatible with legacy PCIe and Ethernet.

We prototype SID on both NVIDIA ConnectX-8 (CX8) and BlueField-3 (BF3) using their DPAs (§6) and evaluate it on hardware testbeds (§7). We compare SID against a high-performance RDMA-based OpenSHMEM implementation [5, 32]. In end-to-end evaluations of basic operations (`PUT`/`GET`), SID achieves 56.50 Mpps with our Elastic QP, 11.03x higher than the RDMA-based solution, which requires eight host x86 CPU cores and eight QPs to match that message rate. For application workloads such as YCSB [33], sparse matrix multiplication (SpMM), and several HPC/ML traces [34, 35], SID achieves up to 12.2x speedup over the RDMA baseline. We also present micro-benchmarks for key SID techniques. SID fully supports both CPU and GPU Direct operations. In terms of flexibility, the FlexOp Engine supports a rich and extensible operation set, compatible with both the OpenSHMEM model for ML/HPC workloads [12, 32] and the Publish–Subscribe Message Queue model for cloud service workloads [1, 36]. More experiments are in §7.

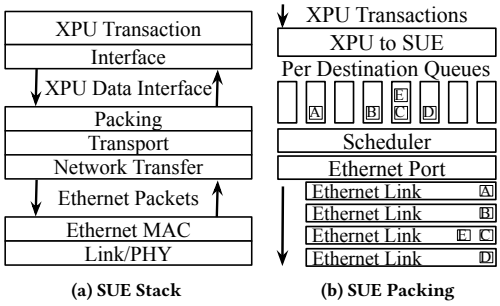
We make the following contributions:

- We characterize RDMA-based solutions for rack-scale fine-grained transfers and identify several issues (§3).
- We design Software-Interposed Datapath (SID) that exploits WQE-Level Parallelism (§4) through Elastic QP (§5) and integrates a FlexOp Engine.
- We demonstrate the performance and flexibility of SID on CX8/BF3 DPAs (§6) using ML/HPC workloads, complying with OpenSHMEM and Message-Queue models (§7).

2 Background

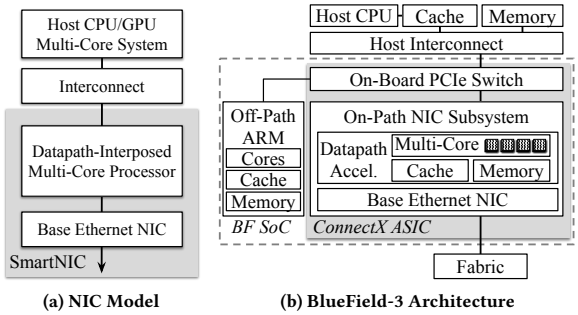
2.1 Rack-Scale Interconnect

Rack Scale: Similar to the scale definitions from Open Compute Project (OCP) Ethernet for Scale-Up Networking (ESUN) [13, 37], NVIDIA NVL-72 [6, 38], UEC [14, 39], and more [40–42], we consider a rack as a frame containing tens of servers, each equipped with multiple processing elements (PE), such as CPUs or GPUs, and a matching number of NICs. We assume that there is a sufficient switching fabric within the rack to connect all NICs, forming a low-diameter network fabric (typically one switch hop) with non-oversubscribed (1:1) bisection bandwidth.



(a) SUE Stack (b) SUE Packing

Figure 1: Overview of Scale Up Ethernet (SUE). Figures are adapted from the public SUE specification by OCP [13, 37].



(a) NIC Model (b) BlueField-3 Architecture

Figure 2: SmartNIC Model and BlueField-3 Example. BF-3 diagram refers to vendor information [25, 51].

Communication Semantics: Rack-scale interconnects typically follow two communication semantics: the network semantic or the memory semantic, as described below.

Network Semantic: Each PE manages its own memory space and communicates with others via explicit network operations, such as `send/receive`. The Message Passing Interface (MPI) is a representative example of this model. Due to the intrinsic overheads of transfer initiation, collective scheduling, and transport-layer processing, network semantics typically favor coarse-grained, bulk data transfers (*e.g.,* 64KB messages) to achieve high efficiency.

Memory Semantic: All PEs share a global memory space and access it using load/store-fashion operations, which implicitly handle communication through shared memory abstractions. Similar to how local memory is accessed at cache-line granularity, memory-semantic operations are typically fine-grained, often at the cache-line or smaller size. A classic example is Distributed Shared Memory (DSM). However, DSM introduces fundamental challenges, most notably, cache coherence, which have been extensively studied [43–47] but remain difficult to scale efficiently. To improve efficiency and scalability, models such as Partitioned Global Address Space (PGAS) [48–50] preserve the memory-semantic interface while explicitly partitioning the global memory into segments associated with each PE, without enforcing rack-scale coherence guarantees. PGAS offers a more practical and scalable design for communication-intensive systems.

Fine-Grained OpenSHMEM-Based Library: In this work, we primarily focus on memory-semantic models, particularly PGAS-based models, as they support fine-grained access. In practice, several PGAS libraries are widely adopted; for example, NVSHMEM [12], which adheres to the OpenSHMEM [5] standard specification. OpenSHMEM defines multiple classes of API routines, such as remote memory access, collective operations, and memory-ordering primitives. Importantly, it separates remote memory access operations (*e.g.,* `shmem_put`) from ordering routines (*e.g.,* `shmem_fence`), exposing ordering control as explicit APIs.

2.2 Ethernet for Scale-Up Networking (ESUN)

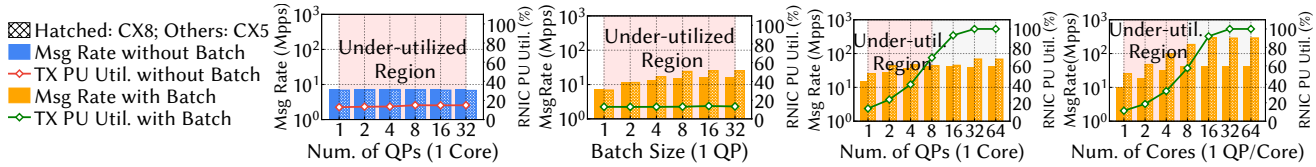
We introduce OCP ESUN as one representative rack-scale interconnect solution [13, 37]. Notably, Scale-Up Ethernet (SUE) [37] serves as the transport layer for ESUN. As Figure 1a shows, the key principle of the SUE stack is to convert XPU (*e.g.,* GPU/CPU) transactions (*e.g.,* load/store-style memory access requests) into Ethernet packets. Within the stack, we highlight an important *Packing* module, as illustrated in Figure 1b. As noted in the SUE specification (Chapter 6.4.5 in [37]), the XPU-to-SUE interface (intra-node interconnect) operates at a higher bandwidth than the Ethernet link (inter-node interconnect), causing transactions to accumulate in per-destination queues within SUE. Figure 1b illustrates this *transaction queuing* behavior, which enables an important optimization opportunity for our design (details in §4).

Relationship Between ESUN and Our SID: First, we adhere to the SUE deployment model (Figure 1a), which is driven by an open OCP community with broad industry participation. We then highlight several distinct contributions of SID: (1) SID introduces a mechanism for elastically harnessing request-level parallelism (§5), which is not covered by the ESUN/SUE specifications; (2) SID provides flexibility in XPU-to-NIC interactions, transport protocols (with SUE compliance), and operation sets (including and beyond those defined by SUE); (3) Importantly, while the ESUN/SUE specifications primarily focus on abstractions and protocol descriptions, we deliver a real hardware prototype.

2.3 On-NIC Datapath Programmability

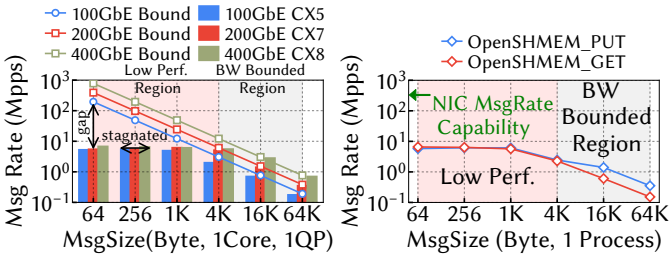
Emerging NICs, including both ASIC-based NICs (*e.g.,* NVIDIA ConnectX-8) and SoC-based SmartNICs (*e.g.,* BlueField-3), incorporate programmable units that allow customization and acceleration of datapath functionality. In Figure 2a, we present a high-level abstract model for such NICs. Specifically, the embedded multi-core sits at a unique *midpoint* serving as a bridge between intra- and inter-server interconnects. We refer to this kind of processor as an on-NIC *interposition processor*.

In Figure 2b, we exemplify the SmartNIC abstract model using BlueField-3 (BF3). BF3 consists of two main subsystems: an on-path NIC subsystem and an off-path ARM subsystem. In this work, we primarily focus on the on-path NIC subsystem, which is more tightly integrated with the base NIC than the ARM subsystem. Notably, such on-path subsystems also exist in the latest RNICs like NVIDIA ConnectX-8 [52]. The NIC subsystem embeds a Datapath Accelerator (DPA), which features 256 RISC-V hardware threads, along with DPA-accessible cache and DRAM memory. DPA is a



(a) More QPs (No Batch, MMIO Bound) (b) More Batching (c) More QPs (Batch) (d) More Cores (Batch)

Figure 3: Journey Toward Maxing Out RDMA NIC Message Rate (64B Msg).



(a) Message Rate Stagnation (b) OpenSHMEM Message Rate

Figure 4: Low Small-Message Rate of RDMA Datapath.

promising fit for our I/O scenario, which features high parallelism for low instruction-per-cycle (IPC) operations such as load/store. Further hardware details are provided in §6.

3 Motivation

In this section, we examine RDMA (RoCEv2), a widely adopted interconnect that leverages hardware-offloaded transport. While RoCEv2 supports fine-grained transfers over Ethernet (*e.g.,* one-sided `READ` and `WRITE`), our characterization reveals inefficiencies and inflexibilities under fine-grained access patterns that motivate our design.

3.1 Achilles' Heel of RDMA Small-Message Rate

Micro-benchmark Setup: In Figures 3 and 4, we present test results using ConnectX-8 (CX8), BlueField-3 (copackaging CX7), and ConnectX-5 (CX5) NICs over 400/200/100 GbE Ethernet cables (testbed details in §7.1). Unless otherwise specified, we test with common `ib_write_bw` operations (64B message size) of *perftest* [53], which reports message rate in Mpps. Besides message rate, we monitor utilization of the TX Processing Unit (PU) inside the RNIC ASIC. To monitor PU utilization, we use *Neo-Host* [54] for CX5 and *Ngauge* [55] for BF3/CX8 (detailed method in Appendix §A.1).

Single QP stagnates despite bandwidth increase. As shown in Figure 4a, large messages (*e.g.,* 64KB) can saturate the link bandwidth, but small messages fall far below the theoretical line-rate limits (*i.e.,* bandwidth divided by message size). Notably, even when upgrading from 100GbE CX5 to 200GbE CX7 and 400GbE CX8, the small-message rate stagnates. This limitation directly impacts the application-level IOPS of memory-semantic load/store operations as shown in Figure 4b, which we measure with the OpenSHMEM `PUT/GET`. Given that the processing rate of a single QP has stagnated, we next investigate how to improve small-message performance.

Journey to Max Out Message Rate (Figure 3): Firstly, as shown in Figure 3a, the single-QP message rate is bounded by the core's MMIO-triggered doorbell. For a single core, increasing the number of QPs does not help, since the MMIO bottleneck remains. Next, as Figure 3b shows, we enable batching, which is a common optimization [56] to improve RDMA performance, as it amortizes doorbell notification overhead and improves DMA efficiency when the NIC fetches WQEs. Batching can help, but beyond a certain threshold, the message rate no longer improves. Therefore, after enabling batching, we increase the number of QPs, as shown in Figure 3c. This approach is effective because it better activates multiple PUs in the RNIC. Once all PUs are utilized, the rate stagnates. Finally, to further improve performance, as shown in Figure 3d, we increase the number of host cores that can concurrently trigger MMIO doorbells.

Overhead of Maximizing Message Rate with QP-Level Parallelism (QLP): As the above journey shows, current RDMA NICs can be driven toward high throughput by engaging multiple QPs across multiple host cores. However, engaging multiple QPs increases computational overhead, such as WQE submission and CQE polling. Another key issue in engaging multiple cores and QPs for applications is that a single QP naturally guarantees FIFO ordering, whereas multi-QP execution requires dispatching requests across QPs and tracking their completions. Enforcing in-order execution without breaking FIFO semantics becomes a heavily *stateful* operation. In GPU-Direct RDMA Async (*i.e.,* GPU Kernel-Initiated), the burden falls on GPU threads to handle such stateful dispatching and tracking across multiple QPs.

Motivation for WQE-Level Parallelism (WLP): The above analysis shows that while QLP can drive RDMA NICs toward high throughput, it does so by pushing stateful parallelism management and ordering complexity to the host. This motivates intra-QP WLP, which transparently preserves a stateless single-QP FIFO abstraction while *elastically* exploiting NIC parallelism to boost small-message rates. As with instruction-level and thread-level parallelism in CPUs, WLP does not replace QLP; the two are complementary and can be jointly exploited.

3.2 Inflexible Hardware-Baked Interconnects

Besides performance limitations, the hardware-fixed datapath of RDMA lacks flexibility in several aspects:

Fixed Optimizations for Last-Mile PCIe Interactions: Although RDMA offloads most processing (*e.g.,* RoCEv2 transport) onto hardware, the *last-mile* PCIe interactions, such as MMIO Doorbells, DMA WQE fetching, and CQE notifications, remain critical

for the overall performance. As shown in Figure 3, batching improves performance by reducing MMIO frequency and increasing DMA WQE efficiency. For RNICs, such optimizations are rigid and difficult to reprogram, due to the fixed interactions between the host driver and the hardware-baked ASIC logic.

Fixed Transport Protocol: RNICs bake network transport into ASIC hardware, leaving little flexibility to customize protocols such as reliable transfers, flow control, and more. However, as described in §2.1, rack-scale topologies are relatively simple, often involving just a single switch hop. As a result, network transports like RoCEv2, which are designed to support both rack-scale and cross-rack communication, are relatively heavyweight. These protocols could be simplified in the rack-scale setting to improve efficiency.

Fixed Operation Set: RNICs typically support a fixed set of verbs, such as one-sided `READ/WRITE` and two-sided `SEND/RECV`, with functionality baked into the RNIC ASIC. In rack-scale communication, both MPI and PGAS models often rely on compound operations, such as collectives (*e.g.,* `alltoall scatter`). If the verb set does not support such compound operations, applications must consume host PE cores to implement the operation logic. Given the rapid evolution of workloads, supporting an extensible, and even programmable, operation set is increasingly valuable.

4 Design Overview

We aim to design an efficient, flexible, and low-cost *Software-Interposed Datapath (SID)* for rack-scale interconnects. We first present several key principles that underpin our design.

4.1 Principles

• $\mathcal{P}1$ **Exploiting Intra-QP WQE-Level[1] Parallelism:** First, adhering to the OpenSHMEM specification (§2.1), we define the request ordering model used by SID, as follows:

> ***SID Ordering Model***
>
> **`Request Submission:`** Requests are submitted in *FIFO* order.
>
> **`Execution:`** Requests are permitted to execute *out of order* between consecutive memory-ordering primitives (*e.g.,* `shmem_fence`).
>
> **`Completion Visibility:`** Request completions are exposed strictly in *FIFO* order, consistent with the submission order. A request's completion is not visible until all earlier submitted requests have completed.

Even under the strict FIFO submission ordering described above, rack-scale communication with memory semantics exposes a high degree of request-level parallelism. The following factors lead to the RMA (Remote Memory Access) request FIFO queue buildup:

1) *Memory Bandwidth Exceeds Network Bandwidth*: The bandwidth of rack-scale interconnects (*e.g.,* Ethernet links) is typically lower than that of local memory (*e.g.,* DDR). As a result, when remote memory is accessed at a rate similar to local memory, it quickly exceeds the network's capacity, resulting in a significant queue buildup. The ESUN/SUE specification reports a similar observation, as described in §2.2 and illustrated in Figure 1b.

2) *Fine-Grained Requests and Episodic Patterns*: Memory-semantic models like OpenSHMEM support fine-grained operations at the granularity of specific data types (*e.g.,* `shmem_int_get()`), as shown

```
shmem_barrier_all(); // Episode begins
for (int dst = 0; dst < comm_size; dst++) {
  int send_off = send_pos[dst];// Local offset for PE dst
  static int recv_off;
  // Get remote offset where dst expects data from me
  shmem_int_get(&recv_off, &recv_pos[my_rank], 1, dst);
  // Put to dst's receive buffer at recv_off
  shmem_int_put(recv_buf + recv_off, send_buf + send_off,
        send_length[dst], dst);
}
shmem_barrier_all(); // Episode ends
```

Code 1: Pseudocode of OpenSHMEM Alltoallv.

in Codeblock 1. Moreover, OpenSHMEM communication often relies on synchronization primitives (*e.g.,* `barrier()`) to enforce completion, forming natural execution episodes — bulks of requests submitted between two synchronization points. Together, these fine-grained and episodic patterns lead to significant queue buildup in request FIFO queues. Notably, as defined in the SID ordering model, within an episode, request execution typically has no ordering constraints for executing `PUT/GET` operations before the `barrier()`. Here, we define the number of requests without ordering constraints within an episode as the *Episode Length*. A larger episode length indicates better applicability of the SID ordering model to a given workload. In Appendix A.2, we analyze the episode lengths of the workloads used in our evaluation to demonstrate the broad applicability of SID.

3) *Collective Operations*: As shown in Codeblock 1, operations like `All-to-All` (variable length) involve every PE in the system. These collective patterns further increase the depth of the request queue due to the large number of simultaneous point-to-point communication requests.

The buildup of FIFO request queues presents a valuable opportunity to exploit request parallelism at the intra-QP level. In §5, we describe how our design leverages this property to boost small-message performance.

• $\mathcal{P}2$ **Exploiting On-NIC Interposition Processor:** We identify the NIC as a critical chokepoint bridging intra-server interconnects with inter-server networks. Moreover, as mentioned in §2.3, modern NICs incorporate programmable units that can access host-side memory, perform processing, and attach to sufficiently high line-rate Ethernet MACs (*e.g.,* 400/800 GbE) for small messages. Such an interposition processor lays the foundation for elastically exploiting parallelism across intra-host PCIe, on-NIC processing units, and inter-host fabrics. Beyond performance gains, the interposition processor enables flexible customization of PCIe interactions, network protocols, and compound collective operation sets.

4.2 SID Architecture Overview

Figure 5 illustrates the architecture of SID, where host PEs (*e.g.,* CPU/GPU) interact with a userspace library that provides Elastic QPs composed of Submission Queues (SQ) and Completion Queues (CQ), all residing in host memory. SID has two key modules on interposition processors, as follows:

Elastic QP (§5.1-§5.4): To enable intra-QP WLP, Elastic QP utilizes a dispatcher and a pool of processing threads of interposition processors. On the sender side (TX), processing threads encapsulate memory semantic operations (*e.g.,* `PUT/GET`) into network packets, which are then transmitted over Ethernet. On the receiver side

[1] In this work, we use the terms "WQE" and "Request" interchangeably; thus, WQE-level parallelism is equivalent to request-level parallelism.

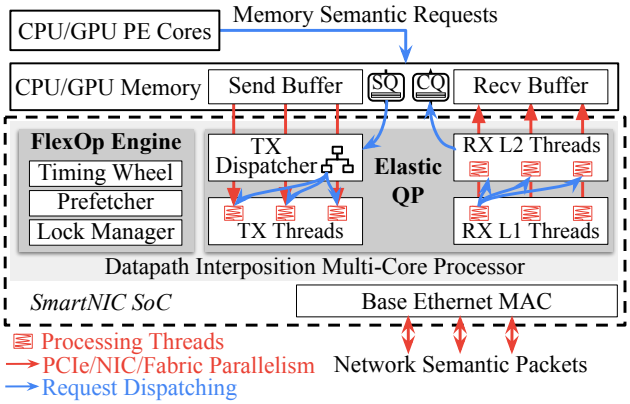


Figure 5: Architecture of Software-Interposed Datapath.

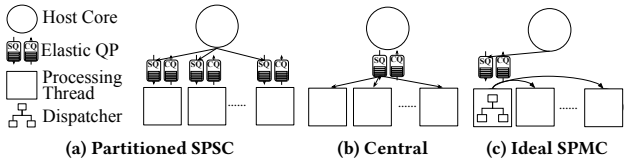


Figure 6: Request Dispatching Models.

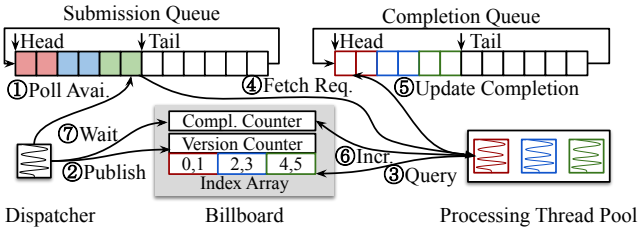


Figure 7: Request Dispatching for Elastic QP. Both SQ and CQ are ring buffers residing in host memory, where head and tail pointers indicate available data and slots. The dispatcher accesses these pointers to detect new requests in the SQ (①). When requests are available, the dispatcher determines the number of pending requests and active processing threads and assigns request slots accordingly. To coordinate this assignment, the dispatcher maintains a global data structure called the *Billboard*, which is visible to all processing threads. The Billboard includes a version counter, a completion counter, and an index array (one cell per thread) that records the assigned slot ranges. After assigning slots, the dispatcher updates the index array with each thread's assigned request range and increments the version counter to publish the new assignments (②). Processing threads check the version counter for updates (③). When a new version is detected, each thread reads its assigned range from the index array, fetches the corresponding requests from the SQ (④), processes them, and writes completions to the CQ accordingly (⑤). Each thread then increments the completion counter (⑥). Once all completions are acknowledged (⑦), the dispatcher updates the tail pointer of the CQ, allowing the host to perceive the completed operations.

(RX), the peer node's threads extract the request and perform the corresponding memory-semantic operations.

FlexOp Engine (§5.5): This module provides auxiliary services to Elastic QP, enabling a flexible operation set. FlexOp supports framework capabilities that accommodate different communication models (*e.g.,* PGAS OpenSHMEM model).

5 Software-Interposed Datapath Design

5.1 TX-Side Elastic QP (E-QP)

We first model the *request dispatching* behavior of E-QP, as shown in Figure 6, and then present the data structure design.

Request Dispatching Model: Efficient request dispatching is non-trivial due to these requirements:

• *Single Producer Multi Consumer (SPMC):* To simplify logic on the host PE side, we dedicate E-QPs for each PE to accept requests, allowing us to treat each E-QP as having a single producer. On the consumer side, while enabling the request dispatching mechanism, we allow multiple processing threads to consume requests from one E-QP concurrently (*i.e.,* multi-consumer).

• *Ordered Intra-QP Parallelism:* As defined in the SID ordering model (§4.1), while the producer submits requests to the SQ in FIFO order, completions must be placed into the CQ in the same order so that the host core perceives them as FIFO-consistent with its submissions. Notably, episodic request patterns allow for out-of-order execution, as long as completion signals are committed to the CQ in order.

• *Stateless Single-QP Host:* The dispatcher must track each thread state and ensure ordered parallelism in a stateful manner. To free the host PE cycles (*e.g.,* CPU/GPU), we offload the dispatchers onto interposition processors, relieving the host from tracking thread states and executing such stateful dispatching logic. The host PEs can then transparently retain a single QP while benefiting from parallel execution through elastic scaling to multiple threads.

• *Lock Free:* As the dispatcher lies on the critical data path, lock-based synchronization across many processing threads incurs significant overhead. Additionally, first-own-first-served lock mechanisms make it difficult to ensure fair request distribution among processing threads or to customize other allocation policies.

• *Copy Free:* For long outstanding request queues, if the dispatcher first fetches requests and then copies them to processing threads, this $O(N)$ copying overhead (N is the number of requests) can turn the dispatcher into a bottleneck. Thus, processing threads need to access requests directly, without intermediate copying by the dispatcher.

Strawman solutions: In Figure 6, we illustrate several strawman and ideal models for request dispatching. Figure 6a shows a Partitioned SPSC model using multiple concurrent SPSC QPs, with each QP corresponding to a single processing thread (*i.e.,* QP-level parallelism). While it satisfies most requirements, it requires the host to dispatch requests across multiple SQs and track multiple CQs, violating the goal of a stateless single-QP host. Figure 6b uses a centralized SPMC queue shared by all processing threads. This

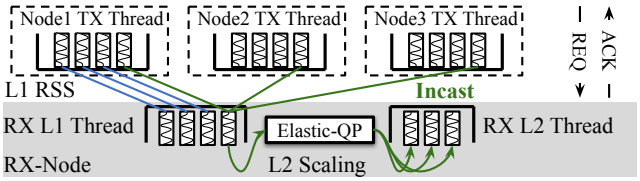


Figure 8: RX-Side Two-Level Scaling.

keeps the host stateless but requires strong and expensive synchronization (*e.g.,* locks) among consumers. Figure 6c presents our ideal model, where the single producer handles a single E-QP with no state, and an offloaded dispatcher thread on the interposition processor performs lock-free, copy-free request dispatch to multiple processing threads. To realize the ideal model, we next design the E-QP data structure.

Request Dispatching for E-QPs enables the ideal dispatching model, as illustrated in Figure 7. The core idea is to use dispatcher threads to monitor request availability and assign SQ and CQ slots to processing threads. Figure 7 presents the data structure and workflow of how E-QP works. E-QP meets all previously defined requirements for request dispatching and addresses the challenges of engaging multiple processing threads. We empirically show that E-QP outperforms other strawman solutions in §7.3.

5.2 RX-Side Elastic QP

Next, we present the RX-side mechanism to improve efficiency. Unlike the TX side, where requests originate from software, the RX side handles incoming packets directly from the NIC hardware, requiring a different approach. On the RX side, we introduce a mechanism called *two-level scaling*.

First-Level Receiver-Side Scaling (RSS): The first-level scaling is to dispatch incoming packets from the wire to receiver processing threads using hardware-based Receiver-Side Scaling (RSS), a widely supported feature in modern NICs. As shown in Figure 8, SID employs two-sided Ethernet `SEND/RECV` operations to transfer requests and ACKs (transport details discussed later in §5.4). We dedicate two pools of processing threads over the interposition processor for TX threads and RX L1 threads. Notably, we configure an equal number of RX L1 processing threads and TX processing threads to simplify RSS configuration.

Second-Level Request Dispatching: While first-level RSS works well for one-to-one transfers, the incast pattern poses additional challenges. To handle many-to-one traffic, we introduce a second thread pool of RX L2 processing threads to support RX-side request dispatching. As shown in Figure 8, under incast traffic, a single RX L1 thread may receive packets from multiple nodes, each containing multiple memory-semantic requests to be processed. To enable parallel processing, the L1 thread (*i.e.,* root thread) then dispatches these packets to multiple L2 processing threads. This dispatching follows the single-producer, multiple-consumer model and must also avoid packet copying during L1-to-L2 handoff. Therefore, the Elastic QP is well-suited for this scenario. Besides the TX side, here is a second scenario for E-QP: a receiver thread stores a batch of packet descriptors in an SQ and then dispatches them to a pool of threads for processing. After processing, the L2 thread delegates the task of sending ACKs back to the original node.

Extension to Multiple E-QP Instances: E-QP primarily focuses on intra-QP parallelism (WLP), but can be naturally extended to support multiple E-QPs (QLP). For this scenario, we extend the design described above. On the interposition processor, we can have multiple dispatcher threads, with each managing one or more E-QPs. To coordinate access to the shared pool of processing threads and the global Billboard, we utilize a centralized, grant-based coordination mechanism: each dispatcher must request a grant before accessing the shared thread pool and release it upon completion, allowing other dispatchers to proceed. The overhead of granting access is minimal, as each dispatcher can process and publish many requests within a single grant period. Notably, we organize all processing threads into a unified pool in a TX/RX-agnostic manner, allowing a specific E-QP to access a larger pool of threads during the grant period.

5.3 Flexible Last-Mile PCIe Operations

In the E-QP workflow (Figure 7), several cross-PCIe operations occur: ① detecting SQ availability, ④ fetching requests, and ⑤ updating completion notifications. As shown by micro-benchmarking in §3.1, these last-mile PCIe (*e.g.,* MMIO) operations impact overall performance. SID supports flexible customization to optimize these PCIe interactions:

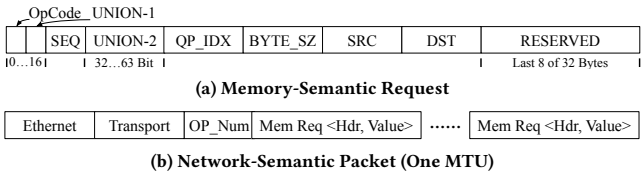
Batching Availability Notification: For ① detecting SQ availability, there are two mechanisms: the host can notify the NIC via MMIO doorbells, or the NIC can proactively poll for availability. In both cases, processing threads are notified only when a batch of requests is ready, amortizing the MMIO overhead across the batch.

Efficient WQE Fetching: For ④ fetching requests, processing threads by default use DMA. Notably, when the number of processing threads is large (*e.g.,* 128), we switch to MMIO fetching, since many threads concurrently fetching their own small request ranges can be more efficient than DMA. DMA incurs a setup time on the order of μs. Another DMA optimization is *request shrinking*, which applies when all requests in a batch share the same type and opcode. We retain the complete request fields only for the first request, while subsequent requests are shrunk to exclude the opcode. This reduces the total amount of data transferred via DMA.

Pipelining WQE Fetching and Processing: To overlap ④ request fetching with processing execution, we employ a pipelined design, in which processing threads execute the current batch while the next batch of requests is fetched in parallel. Once the current batch completes, processing threads can immediately begin processing the next batch.

Inlining Small Payload: During ④ request fetching, if the message payload is sufficiently small (*e.g.,* 8B), we inline the payload with the WQE request, eliminating the additional operations required to fetch the payload separately.

Importantly, the mechanisms above highlight the runtime programmability of SID rather than the novelty of individual optimizations. Existing RDMA NICs offer limited runtime programmability for such optimizations, whereas SID can potentially adopt a rich set of techniques from the host–NIC interaction literature [57–63].



(a) Memory-Semantic Request

(b) Network-Semantic Packet (One MTU)

Figure 9: Format Definitions for Memory-Semantic Requests and Network Packets. 9a shows the format of a memory-semantic request, where an OpCode denotes the operation type (*e.g.,* PUT/GET), along with metadata fields such as the QP index, some union structures (U1/U2) for optional parameters (*e.g.,* multicast group size or time delay), source/destination memory addresses, and reserved bytes for software-defined extensibility. 9b shows the network packet format.

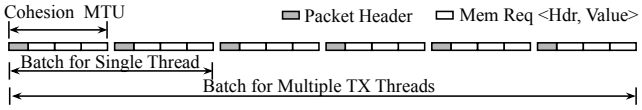


Figure 10: Sub-MTU Request Cohesion.

5.4 Flexible Transport Protocol

Request Format: In SID, the bit-level formats of both memory requests and network packets (above L2 Ethernet layer) are software-defined and flexible, as they are processed by a general-purpose interposition processor. We present the detailed request and packet formats in Figure 9. At the link layer, our design is based on Ethernet. At the transport layer, software processing enables flexibility in transport protocol choice. By default, we use UDP/IP, similar to RoCEv2, but the design can potentially support emerging transports such as SUE [37] and UEC [14]. Next, we describe how to encapsulate packet payloads.

Sub-MTU Request Cohesion: To utilize long outstanding request queues, each TX processing thread attempts to pack multiple fine-grained memory-semantic requests for the same point-to-point connection into a single MTU-sized network packet, a technique we refer to as *request cohesion* or association (illustrated in Figure 10). In addition to cohesion MTUs, we apply batching techniques to both single- and multi-TX processing threads. To reduce the overhead of frequent doorbell MMIO rings, we batch multiple cohesion MTUs and trigger the doorbell only once per batch. Furthermore, we dispatch the tasks of assembling cohesion MTUs and transmitting packets to multiple parallel TX processing threads via E-QP.

Reliability and Flow Control: As described in §2.1, rack-scale interconnects typically feature simpler topologies and sufficient link connectivity, enabling the use of lightweight transport protocols. We adopt a connectionless UDP transport with run-to-completion execution, where the TX thread, after sending REQ packets, waits for ACK from the RX or retries upon timeout. The timeout value is configurable; in our prototype, we simply calibrate it based on the measured RTT. Such run-to-completion execution and timeout-based retransmission guarantee reliable delivery. For flow control, we apply the responder-driven credit-based flow control. Specifically, ACK packets carry a credit field that informs the requester whether to shrink the pool of TX processing threads. If a TX thread receives a credit bit indicating throttling, it is excluded from the next transmission round. After a period, the system attempts to resume the thread to expand the pool adaptively. For packet-loss scenarios, credits are held until ACKs are received, preventing over-injection. For multipathing scenarios, we maintain credits globally rather than on a per-path basis; per-path feedback can be integrated through programmable customization. Notably, our system flexibly customizes protocols, such as flow control, thanks to its software-defined approach over interposition cores.

5.5 FlexOp Engine

Building on the above performance mechanisms, we next turn to software-interposed flexibility. SID supports a versatile FlexOp Engine targeting two models: OpenSHMEM for ML/HPC and Message Queues for cloud services. Table 1 summarizes the operation sets for the two models.

Framework General Capability: SID adopts a *vertical integration* approach spanning the communication software (*e.g.,* OpenSHMEM library), on-NIC interposition processor, and the base Ethernet NIC. This approach enables flexible extension of operation sets to adapt to application workloads, built upon the following key building blocks: 1) Software-defined request format; 2) E-QP for parallel dispatch of compound operations; 3) Programmable computation on TX/RX sides; and 4) Device-Initiated Ethernet SEND/RECV capability. Next, we exemplify with two use cases.

Target-1: PGAS OpenSHMEM Model. The OpenSHMEM specification [5] defines multiple routine types (Table 1). After supporting the data-transfer routines PUT/GET (§5), we now extend to additional types.

Atomic Operations: SID supports atomic operations such as `fetch-and-add (FADD)` and `compare-and-swap (CAS)`. Atomic operations require the responder to guarantee atomicity on host memory. Because responder logic runs on the NIC, atomicity must hold across PCIe transactions and concurrent on-NIC cores. The FlexOp Engine provides a lock service to ensure exclusive cross-PCIe access: The responder threads acquire the lock before executing atomic operations.

Synchronization Operations: With atomic operations, SID supports synchronization primitives such as `barrier()`. Efficient barrier implementations have been extensively studied [64–66]. We take a classic algorithm: each PE performs a FADD to increment a shared counter on a root PE. Once the counter equals the number of PEs, the last-updating PE broadcasts a completion to all PEs.

Collective Operations: With the framework building blocks, SID supports generic collective operations as Table 1 shows. To exemplify, we introduce an advanced operation, `All-to-All Scatter` and `Gather` [67], which even extends beyond the basic OpenSHMEM specification. `All-to-All Scatter` and `Gather` generalizes `AllGather` by allowing each process to send *distinct* data to every other process: the jth block sent by process i is received by process j and stored in the ith block of its `recvbuf`. SID supports `All-to-All Scatter` and `Gather`, allowing the DPA to delegate intra-host data gathering, thereby reducing GPU computation overhead.

Operation Type	Comm. Model	Operations	Key Mechanisms
Data Transfer	Both	`PUT/GET`	Elastic QPs primarily handle them.
Atomic	OpenSHMEM	`Fetch-and-Add/Compare-and-Swap`	FlexOp Engine provides memory access locks.
Synchronization	OpenSHMEM	`Barrier/Fence/Wait`	Barrier using FADD to update shared root counter.
Collective	OpenSHMEM	`Broadcast/Multicast/Reduction/Scatter/Gather`	E-QP and Programmability on TX/RX sides.
Time-Sensitive	Pub/Sub	`Delayed_Op/Periodic_Op`	A service based on Hierarchical Timing Wheel.
Delegation	Pub/Sub	`Pub_Prepush/Delegated_Poll`	Pre-push messages to SmartNIC and offload subscriber.

Table 1: Flexible Operation Set Enabled by the FlexOp Engine.

Target-2: Publish-Subscribe Message-Queue Model. In addition to primary ML/HPC workloads, SID also targets cloud service workloads. The FlexOp Engine provides operations required by the common Publish–Subscribe (Pub/Sub) Message Queue model. Due to space limitations, we describe the support for this case in Appendix §A.3.

6 Implementation

We prototype SID on both NVIDIA ConnectX-8 (CX8) and BlueField-3 (BF3) NICs. Specifically, we implement SID on the Datapath Accelerator embedded on the NICs, as follows:

Datapath Accelerator (DPA) Characterization: Some prior work [68–71] presents comprehensive characterizations of DPA. Among them, *White-Boxing RDMA* [68] offloads transport control (*e.g.,* congestion control) to the DPA software, while the datapath remains in RDMA ASIC hardware. SID further leverages the DPA for *both* control and datapath processing, to fully unlock software flexibility and elasticity. Due to space limitations, the DPA hardware specification is in Appendix §A.4.

Memory Aperture: A key mechanism we utilize is the memory aperture [72], which enables the DPA to access host-side memory using load/store instructions. This mechanism is compatible with legacy PCIe. We measure the latency of DPA accessing host-side DDR memory to be around 900ns. Notably, this access is asymmetric: while the DPA can access host DDR memory, the host cannot directly access DPA-side DDR via load/store instructions.

GPU Support: The Memory Aperture mechanism also applies to GPU memory. Specifically, we allocate a GPU memory buffer using `cudaMalloc`, register it with `ibv_reg_mr`, and pass the resulting handler to the DPA, enabling the DPA to perform load/store accesses to the GPU buffer.

Ethernet Capability: Both CX8 and BF3 (packaging a CX7 RNIC) expose raw Ethernet capabilities to the DPA. Specifically, the DPA can use Unreliable Datagram (UD) QPs to issue `SEND_EN` and `RECV_EN` requests. Importantly, these requests bypass the RoCEv2 hardware module, meaning the transport logic is defined by the DPA, unlike RDMA-hardware-fixed RoCEv2 transport.

Programmability Toolchain: The DPA supports general-purpose C programming, based on a restricted subset of the C11 standard (*e.g.,* no floating-point support).

Thread Pool Management: The DPA runs a lightweight real-time operating system (RTOS) rather than a full-fledged Linux kernel. Fortunately, SID defines fixed thread roles, such as TX dispatchers and processing, and RX L1/L2 processing, allowing it to operate effectively within the RTOS.

Lightweight Jobs on Host: The core functionalities run on the DPA, while the host is responsible for loading the program onto the DPA and registering memory regions for QPs to accept tenant requests. Then the DPA fetches requests from QPs via the memory aperture mechanism, performs request cohesion in the DPA-local cache, and uses Ethernet to transmit the data.

Userspace Library: We implement SID in about 7K lines of C code. The integration of SID follows a workflow similar to other OpenSHMEM libraries, such as NVSHMEM.

Based on our implementation experience, we provide several suggestions for DPA vendors in Appendix §A.7.

7 Evaluation

7.1 Setup

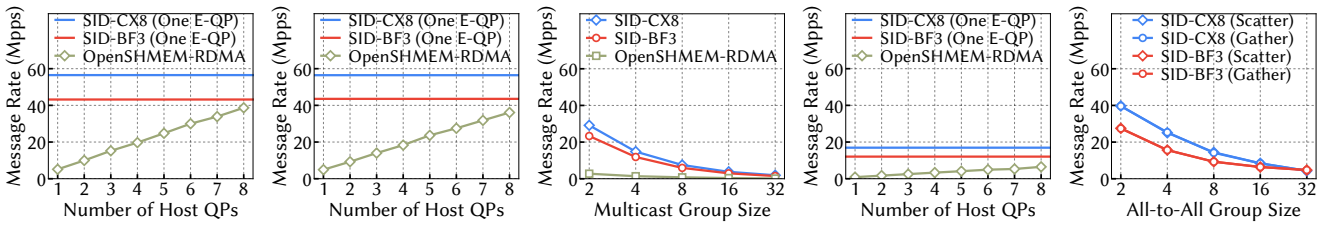
Testbed: We have two testbeds: one using ConnectX-8 NICs and the other using BlueField-3 NICs. The CX8 testbed consists of two servers connected back-to-back via a 400GbE cable. Each server is equipped with dual Intel Xeon Gold 6438M CPUs, 512GB DDR5 memory, PCIe Gen5, and an NVIDIA ConnectX-8 NIC (2×400GbE, C8180). The BF3 testbed consists of two servers connected back-to-back via a 100 GbE cable. Each server has dual Intel Xeon Gold 6346 processors, 384GB DDR4, PCIe Gen4, and a BlueField-3 DPU (B3220) with 100GbE cables. BF3 integrates a CX7 RNIC. The host machines run Ubuntu 22.04 with kernel 5.15, DOCA SDK v3.1. For the characterization shown in §3.1, we install CX5 NICs in the CX8 testbed and query PU utilization using Neo-Host [54] for CX5 and Ngauge [55] for CX8; CX5 is not used in the following evaluations.

Baseline: We compare SID against an RDMA-based OpenSHMEM implementation (abbreviated as *OpenSHMEM-RDMA* in experiment figures). Specifically, we use CX8 NICs and the high-performance UCX library [4, 32], developed by NVIDIA and an active open-source community. UCX (Unified Communication-X) provides production-grade RDMA optimizations and supports the OpenSHMEM library.

Parameter Setting: For SID, we configure 128 DPA threads (8 Cores) for TX processing and RX L1 processing to support the request dispatching and 32 threads for TX dispatchers and RX L2 processing to enable two-level scaling. The Ethernet MTU size is set to 1500B, supporting up to 64 cohesion requests per MTU. We use `shmem_double_get/put` (8B) operations, allowing up to 4K outstanding requests between two `barrier` function calls. Accordingly, we configure each SID memory-semantic request to access 8B of data. We report *millions of memory operations per second* (Mpps).

7.2 End-to-End Performance

Performance of Critical Operations: In Figure 11, we evaluate the small-message rate for common operations.



(a) PUT (b) GET (c) Multicast (d) Fetch-and-Add (e) Scatter/Gather

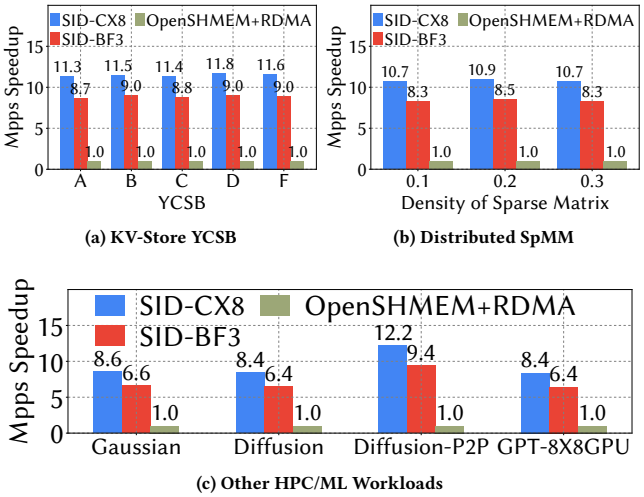
Figure 11: Message Rates of SID Critical Operations.

In Figure 11a, we present message rates for OpenSHMEM PUT operations. With our E-QP, SID achieves 56.50 Mpps on CX8 and 43.19 Mpps on BF3, 11.03x higher than the RDMA-based OpenSHMEM on CX8. SID performance is boosted thanks to the E-QP harnessing intra-QP WLP and pushing parallelism exploitation onto the NIC. In comparison, the RDMA-based solution requires eight host x86 CPU cores and eight QPs to reach similar performance, which is substantial for the host, particularly considering that RDMA has offloaded transport. Notably, E-QP uses only half of the available DPA resources, and micro-benchmarks (§7.3) indicate headroom for scaling. Additionally, SID, leveraging low-cost RISC-V DPA cores, not only reduces host x86 CPU usage but also operates on basic Ethernet MACs rather than relying on RDMA ASIC modules like the RoCEv2 module.

In Figure 11b, we present message rates for OpenSHMEM GET operations. With one E-QP, SID achieves 56.45 Mpps on CX8 and 43.54 Mpps on BF3, 11.49x higher than OpenSHMEM-RDMA on CX8. Similar to the PUT case, RDMA requires eight host CPU cores driving eight QPs to reach a comparable total message rate. Notably, E-QP uses only one host core, which keeps the host stateless and avoids stateful management across multiple QPs and cores.

In Figure 11c, we present the results for `Multicast` operations. We fix a single host CPU core to perform one-to-many multicast and count each `Multicast` request as a single operation, regardless of the `Multicast` group size. In SID, a `Multicast` request is split and dispatched to multiple thread pools, with each pool of threads responsible for sending to a specific responder. We also enable batching for `Multicast` requests, allowing each thread pool to process a batch of `Multicast` requests for better efficiency. As the group size increases, the message rate decreases proportionally since the thread pool assigned to each responder is reduced. Nevertheless, SID consistently outperforms the RDMA-based solution across all group sizes (*e.g.,* 9.74x higher at size 8).

In Figure 11d, we present the results for `Fetch-and-Add` atomic operations. Since it requires a lock mechanism on the responder side to ensure atomicity, its performance is lower than that of PUT and GET for both RDMA and SID. To mitigate this overhead, the FlexOp Engine provides fine-grained lock services, maintaining by default $O(N)$ locks where N is the number of RX L1 processing threads. Each lock corresponds to a specific memory region, helping reduce lock contention. As a result, even under several concurrent connections, SID maintains high performance—for example, achieving 2.62x higher throughput than OpenSHMEM-RDMA with 8 concurrent QPs.



(a) KV-Store YCSB (b) Distributed SpMM

(c) Other HPC/ML Workloads

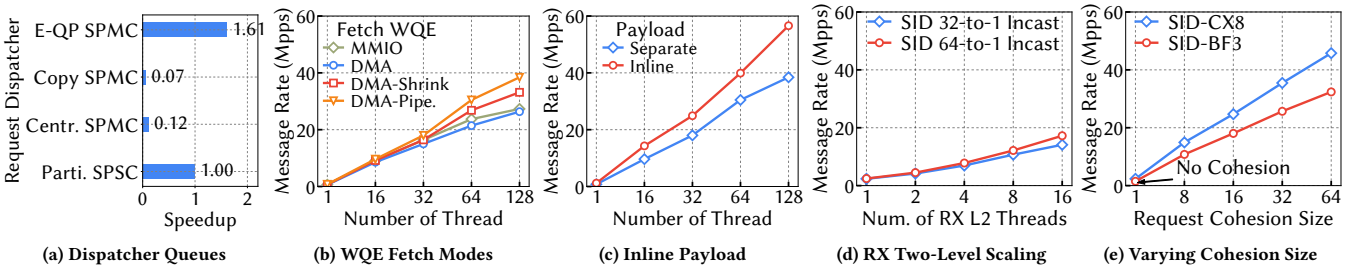
Figure 12: Message Rate for Application Workloads.

In Figure 11e, we present the results for All-to-All *Scatter* and *Gather* collective operations. Here, without loss of generality, we simulate a node with 8 PEs (*e.g.,* GPUs), each issuing 8 messages to be gathered or scattered to a peer node. The resulting 64 requests can then be packed into a single MTU using our sub-MTU cohesion. Results show that SID can deliver high message rates for these advanced collective operations. Note that OpenSHMEM provides no native support for this operation type, so no baseline is shown.

Performance of Application Workloads: In Figure 12, we evaluate the message rate of SID using PUT/GET traces from several real applications: the YCSB key-value store benchmark, a distributed sparse matrix multiplication (SpMM) workload, and several advanced HPC/ML applications.

Figure 12a shows results for YCSB workloads. YCSB [33] is a widely used benchmark suite for evaluating NoSQL datastores through a mix of write/read operations, which we map to PUT/GET for accessing remote memory. We use one QP for this test. We evaluate five YCSB workloads, excluding YCSB-E as its `Scan` operation is not in the OpenSHMEM specification. Given the varying Mpps across workloads, we normalize SID performance to OpenSHMEM-RDMA and report the speedup. SID outperforms RDMA across all workloads (*e.g.,* achieving 11.3x higher rate for YCSB-A).

Figure 12b shows the results for SpMM workloads. SpMM is an important building block for ML and has been extensively studied [73–75]. In this work, we focus on evaluating data transfer efficiency, orthogonal to algorithmic optimizations. Therefore, we



(a) Dispatcher Queues **(b) WQE Fetch Modes** **(c) Inline Payload** **(d) RX Two-Level Scaling** **(e) Varying Cohesion Size**

Figure 13: Micro-Benchmarking SID Key Techniques. 13a–13d present only CX8 results for clarity and higher performance.

adopt a simple distributed SpMM algorithm to stress the communication layer. Specifically, we consider matrix multiplication $A \times B = C$ ($A, B, C \in \mathbb{R}^{n \times n}$) where the matrices A, B, C are distributed across different servers. During computation, a server holding A fetches non-zero elements $B_{[i,j]}$ from a remote server on demand and writes the resulting $C_{[i,j]}$ back to another remote server. We use density (the probability of a non-zero element) to quantify matrix sparsity. As Figure 12b shows, SID outperforms OpenSHMEM-RDMA. Moreover, performance gains increase with matrix density, since denser matrices require more on-demand fetches from the peer node.

Figure 12c shows the results for several advanced HPC/ML workloads. For HPC workloads, we use traces from the *Tartan* benchmark suite [34], an open-source suite released by PNNL to benchmark scale-up and scale-out interconnects for multi-GPU systems. Specifically, we use two application traces from the suite: Gaussian Mixture [76] and Diffusion [77]. For Diffusion, we use the original collective-based version and also design a variant that leverages point-to-point (p2p) PUT/GET operations. For ML workloads, we generate communication traces using *SimAI*, a simulator for large-scale AI training open-sourced by Alibaba [35]. Specifically, we simulate a topology of eight nodes, each equipped with eight GPUs, to train the GPT-13B model. The results show SID delivers higher Mpps than RDMA-based OpenSHMEM.

7.3 Micro-benchmarking Key Techniques of SID

In Figure 13, we present several micro-benchmarks to evaluate SID's key techniques described in §5.

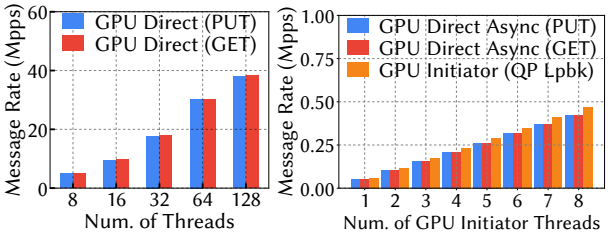
E-QP Request Dispatching: In Figure 13a, we evaluate the effectiveness of E-QP, the key enabler of the request dispatching in SID. We compare E-QP against several strawman solutions described in §5.1, using the setting of one E-QP with 128 processing threads. We normalize the message rate of each solution to the baseline of Partitioned SPSC, which uses N concurrent QPs. The Centralized SPMC solution uses locks to coordinate all threads for dequeuing, leading to significant overhead due to lock contention. The Copy SPMC solution requires the dispatcher to copy requests to other processing threads, which introduces high overhead from copying. These results highlight the importance of design requirements such as lock-free and copy-free dispatching, as discussed in §5.1. E-QP achieves the highest performance with a single SPMC queue and avoids the host-side overhead of operating N QPs as in the Partitioned SPSC design.

WQE Fetching: In Figure 13b, we evaluate the impact of the request dispatching by varying the number of processing threads serving a single E-QP for PUT operations. As the number of processing threads increases, the message rate improves significantly — for example, increasing from 1 to 128 threads results in a 42.70x higher rate. This result highlights the effectiveness of E-QP in exploiting WQE-level parallelism via utilizing multiple lightweight processing threads to boost performance. Furthermore, we demonstrate several optimizations mentioned in §5.3. We support processing threads that fetch memory-semantic requests using either DMA or MMIO. When the number of threads is large, concurrent MMIO becomes efficient due to its lower setup overhead compared to DMA jobs. Moreover, DMA shrinking is an efficient optimization that reduces the amount of data transferred via DMA, which is particularly beneficial when more threads are active and processing larger batches of requests. Furthermore, pipelining DMA fetching and thread processing improves efficiency.

Payload Inlining: In Figure 13c, we evaluate the impact of inlining small payloads as described in §5.3. For both inlined and separated payloads, the same DMA mechanism is used to fetch WQEs. Inlining the payload improves efficiency by avoiding extra DMA operations for payload fetching.

RX Two-Level Scaling: In Figure 13d, we evaluate the effectiveness of RX two-level scaling in the incast scenario. To handle incast, we scale the processing capacity of a single L1 thread by dispatching requests to multiple L2 processing threads. In Figure 13d, we present the message rates of one L1 thread for 32-to-1 and 64-to-1 incast scenarios. The results show that increasing the number of L2 processing threads significantly improves the message rate that a single L1 thread can sustain, effectively handling the incast challenge. Note that we use two CX8 NICs and partition the requester-side NIC resources to emulate 32/64 senders.

Request Cohesion: In Figure 13e, we evaluate the impact of request cohesion by varying the number of packed requests within a single MTU packet, using one E-QP to perform PUT operations. As the number of packed requests increases, SID achieves significantly higher message rates because of improved transfer efficiency. For example, increasing the number from 1 to 64 on CX8 improves efficiency by 19.64x. We cap the cohesion size at 64 requests, corresponding to the common 1500B MTU; enabling jumbo frames (*e.g.,* 9KB) allows packing more requests into a single packet. Notably, this experiment uses 128 DPA threads to enable E-QP request dispatching because request cohesion alone is insufficient. As Figure 13b shows, even with request cohesion enabled (cohesion size 64), E-QP cannot achieve high performance with only a limited number of DPA threads (*e.g.,* one thread).



(a) Mode-1: GPU Direct (b) Mode-2: GPU Direct Async

Figure 14: GPU Initiated Operations over SID.

7.4 Support for GPU-Direct Operations

SID fully supports GPU-Direct transfers, even for entry-level GPUs. We set up one NVIDIA RTX 4000 Ada GPU in the responder server of two machines in our CX8 testbed (§7.1) with CUDA v13.0. As Figure 14 shows, similar to GPU-Direct RDMA, SID supports two modes: GPU-Direct SID (data on GPU, initiators on CPU) and GPU-Direct Async SID (both data and initiators on GPU, *i.e.,* IBGDA [12]). Here, initiator control includes operations such as submitting requests and ringing the doorbell. For Mode-1 GPU-Direct, we use one E-QP while varying the number of processing threads. The results show that SID achieves high message rates in GPU-Direct operations. For Mode-2 GPU-Direct Async, we vary the number of GPU initiator threads, each of which operates a dedicated E-QP. Notably, GPU-Direct mode outperforms Async mode because a single CPU core is much more efficient than a single GPU thread at feeding requests. Such GPU-bound constraints are common to both RDMA and our solution. To further validate this bound, we set up a "QP Loopback (Lpbk)" test (Figure 14b), which disables the cross-host transport phase and enables only the request-feeding phase. The results show that the message rate is limited by the request feeding rather than by SID transfer.

7.5 Generality for Large-Size or Low-Latency Tasks

Although this work focuses on fine-grained access (§4.1), with throughput as the primary metric, SID also demonstrates promising generality for other message types, as E-QP enables higher operation rates and achieving high bandwidth is inherently easier with larger messages. We present the results for large messages in Appendix §A.5. For low-latency tasks, despite the overhead of software interposition, SID achieves latency comparable to the ASIC-based RDMA solution. In Appendix §A.5, we present latency results under both idle and loaded settings and describe several latency-reduction optimizations used in our implementation.

7.6 Flexibility of the FlexOp Engine

To further showcase the flexibility, we present operations for the Pub/Sub Message Queue model in Appendix §A.6.

8 Related Work

RDMA NIC and Hardware Transport: Several prior works reveal the RNIC micro-architecture [18, 56, 68, 78–88] or typical NIC architecture [27, 58, 63, 89–93], which builds the knowledge base for SID to program RNIC and Host–NIC interactions, while our E-QP exploits WLP to boost datapath performance, a capability not present in prior approaches. Prior work focuses on improving QP scalability (*e.g.,* DCT [94], SRC [95], and more [80, 96]), while E-QP focuses on improving performance via WLP. Several works explore in-fabric multi-path RDMA or out-of-order delivery (*e.g.,* DDP [97], ConWeave [98], MP-RDMA [99], and Eunomia [100]), whereas E-QP primarily focuses on end-host NIC processing parallelism, which is orthogonal to in-fabric ones. Rich literature [7, 98, 101–115] optimizes transport, but SID targets rack-scale settings with simpler topologies, enabling a lightweight and programmable transport design that can adopt innovations from these approaches. Prior works discuss limitations in RDMA semantics [16, 84, 116–119]; SID offers a flexible operation set.

RDMA-Based Remote Memory: RDMA enables memory disaggregation [43, 120–128]. Unlike disaggregated memory settings, where nodes are asymmetric (*i.e.,* separated computation and memory nodes), SID targets the PGAS model, in which all nodes are symmetric and possess both computation and memory resources. Some may point out similarities to the DSM [44–46] or other emerging interconnects [42, 129–131], but SID follows the OpenSHMEM advocacy [5], dropping rack-scale cache coherence to prioritize efficiency and compatibility with legacy PCIe/Ethernet.

SmartNIC: While SmartNICs often offload upper-layer processing (ULP) or network functionalities, including support for cloud multi-tenancy [132–139], security [140, 141], filesystem [142], storage [143–150], distributed systems [17, 151–154], and more [155, 156], SID uses SmartNIC programmability to enhance the low-level datapath, potentially benefiting ULP. We draw on prior SmartNIC designs [92, 116, 157–159], but unlike the bespoke NIC, SID utilizes a commodity interposition processor.

9 Conclusion

In this work, we propose Software-Interposed Datapath (SID), an efficient, software-flexible, and low-cost solution for rack-scale interconnects. Elastic QP boosts small-message efficiency via exploiting intra-QP WQE-level Parallelism with on-NIC interposition processors. FlexOp Engine supports a comprehensive operation set compatible with the OpenSHMEM model for ML/HPC workloads and the Message Queue model for cloud workloads. Built on cost-effective hardware, SID achieves high performance and is open to evolving protocols and extending operations.

This work does not raise any ethical issues.

Acknowledgments

We would like to thank the anonymous shepherd and reviewers for their feedback. This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This work was supported in part by NSF grants CNS-2212193, CNS-2213387, CNS-2504469, CNS-2106199, CNS-2212192, and CAREER-2339755.

References

[1] Apache Kafka. https://kafka.apache.org/, 2025.

[2] Xuya Jia, Zhiyi Yao, Chao Peng, Zihao Zhao, Bin Lei, Edison Liu, Xiang Li, Zekun He, Yachen Wang, Xianneng Zou, et al. Turbo: Efficient communication framework for large-scale data processing cluster. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 540–553, 2024.

[3] Konstantin Taranov, Steve Byan, Virendra Marathe, and Torsten Hoefler. Kafkadirect: Zero-copy data access for apache kafka over rdma networks. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2191–2204, 2022.

[4] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. Ucx: an open source framework for hpc network apis and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE, 2015.

[5] OpenSHMEM. http://openshmem.org/site/, 2025.

[6] NVIDIA GB200 NVL72. https://www.nvidia.com/en-us/data-center/gb200-nvl72/, 2025.

[7] Building Meta's GenAI Infrastructure. https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/, 2024.

[8] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[9] Chenchen Shou, Guyue Liu, Hao Nie, Huaiyu Meng, Yu Zhou, Yimin Jiang, Wenqing Lv, Yelong Xu, Yuanwei Lu, Zhang Chen, et al. Infinitehbd: Building datacenter-scale high-bandwidth domain for llm with optical circuit switching transceivers. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 1–23, 2025.

[10] Raj Joshi, Saksham Agarwal, ChonLam Lao, and Minlan Yu. Your network doesn't end at the nic: A case for unifying the inter-host and intra-host networks in (ai) datacenters. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*, pages 280–288, 2025.

[11] Open MPI Documentation: OpenSHMEM Programming Model. https://docs.open-mpi.org/en/v5.0.x/man-openshmem/man3/OpenSHMEM.3.html, 2025.

[12] NVIDIA NVSHMEM. https://docs.nvidia.com/nvshmem/index.html, 2025.

[13] Open Compute Project (OCP). Ethernet Scale-up Networking. https://www.opencompute.org/wiki/Networking/ESUN, 2026.

[14] Ultra Ethernet Consortium. https://ultraethernet.org/, 2025.

[15] Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE). https://www.infinibandta.org/, 2014.

[16] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan RK Ports. Prism: Rethinking the rdma interface for distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 228–242, 2021.

[17] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. Characterizing off-path SmartNIC for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 987–1004, Boston, MA, July 2023. USENIX Association.

[18] 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.

[19] Google Falcon. https://github.com/opencomputeproject/OCP-NET-Falcon, 2024.

[20] Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan M. G. Wassel, Naveen Kr. Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, Sarin Thomas, Inho Cho, Hyojeong Lee Seibert, Baijun Wu, Rui Yang, Yuliang Li, Kai Huang, Qianwen Yin, Abhishek Agarwal, Srinivas Vaduvatha, Weihuang Wang, Masoud Moshref, Tao Ji, David Wetherall, and Amin Vahdat. Falcon: A reliable, low latency hardware transport. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 248–263, New York, NY, USA, 2025. Association for Computing Machinery.

[21] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A cloud-optimized transport protocol for elastic and scalable hpc. *IEEE Micro*, 40(6):67–73, 2020.

[22] Min Si, Pavan Balaji, Yongzhou Chen, Ching-Hsiang Chu, Adi Gangidi, Saif Hasan, Subodh Iyengar, Dan Johnson, Bingzhe Liu, Regina Ren, Deep Shah, Ashmitha Jeevaraj Shetty, Greg Steinbrecher, Yulun Wang, Bruce Wu, Xinfeng Xie, Jingyi Yang, Mingran Yang, Kenny Yu, Minlan Yu, Cen Zhao, Wes Bland, Denis Boyda, Suman Gumudavelli, Prashanth Kannan, Cristian Lumezanu, Rui Miao, Zhe Qu, Venkat Ramesh, Maxim Samoylov, Jan Seidel, Srikanth Sundaresan, Feng Tian, Qiye Tan, Shuqiang Zhang, Yimeng Zhao, Shengbao Zheng, Art Zhu, and Hongyi Zeng. Collective communication for 100k+ gpus, 2026.

[23] Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, and Dennis Cai. Alibaba stellar: A new generation rdma network for cloud ai. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 453–466, New York, NY, USA, 2025. Association for Computing Machinery.

[24] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[25] NVIDIA BLUEFIELD-3 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf, 2024.

[26] Eric Quinnell. 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)*, pages 1–23. IEEE Computer Society, 2024.

[27] Henry N Schuh, Arvind Krishnamurthy, David Culler, Henry M Levy, Luigi Rizzo, Samira Khan, and Brent E Stephens. Cc-nic: a cache-coherent interface to the nic. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 52–68, 2024.

[28] Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E Grant, and Ron Brightwell. spin: High-performance streaming processing in the network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2017.

[29] Yifan Yuan, Jinghan Huang, Yan Sun, Tianchen Wang, Jacob Nelson, Dan RK Ports, Yipeng Wang, Ren Wang, Charlie Tai, and Nam Sung Kim. Rambda: Rdma-driven acceleration framework for memory-intensive μs-scale datacenter applications. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 499–515. IEEE, 2023.

[30] AMD Pensando. https://www.amd.com/en/products/data-processing-units/pensando.html, 2025.

[31] Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N Patt. Runahead execution: An alternative to very large instruction windows for out-of-order processors. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pages 129–140. IEEE, 2003.

[32] The Unified Communication X Library. http://www.openucx.org.

[33] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[34] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. Tartan: evaluating modern gpu interconnect via a multi-gpu benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 191–202. IEEE, 2018.

[35] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, et al. SimAI: Unifying architecture design and performance tuning for Large-Scale large language model training with scalability and precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 541–558, 2025.

[36] Eashan Gupta, Prateesh Goyal, Ilias Marinos, Chenxingyu Zhao, Radhika Mittal, and Ranveer Chandra. Dbo: Fairness for cloud-hosted financial exchanges. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 550–563, New York, NY, USA, 2023. Association for Computing Machinery.

[37] Open Compute Project (OCP). Scale Up Ethernet (SUE). https://www.opencompute.org/documents/ocp-sue-spec-final-pdf-1, 2026.

[38] MGX Accelerated Computing Rack and Trays Specification. https://www.opencompute.org/documents/mgx-accelerated-computing-rack-and-trays-specification-101024-pdf, 2025.

[39] Torsten Hoefler, Karen Schramm, Eric Spada, Keith Underwood, Cedell Alexander, Bob Alverson, Paul Bottorff, Adrian Caulfield, Mark Handley, Cathy Huang, Costin Raiciu, Abdul Kabbani, Eugene Opsasnick, Rong Pan, Adee Ran, and Rip Sohan. Ultra ethernet's design principles and architectural innovations, 2025.

[40] UALink Consortium. https://ualinkconsortium.org/, 2025.

[41] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 255–270, 2019.

[42] Yuxin Ren, Mingrui Liu, Hongbo Li, Chang Liao, Xiaojia Huang, Jianhua Zhang, Hanjun Guo, Yubo Liu, and Ning Jia. Towards rack-as-a-computer in memory interconnect era with coordinated operating system sharing. In *Proceedings of the 17th ACM Workshop on Hot Topics in Storage and File Systems*, pages 77–85, 2025.

[43] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, 2014.

[44] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-Tolerant software distributed shared memory. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 291–305, 2015.

[45] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient distributed memory management with rdma and caching. *Proceedings of the*

VLDB Endowment, 11(11):1604–1617, 2018.

[46] Qing Wang, Youyou Lu, Erci Xu, Junru Li, Youmin Chen, and Jiwu Shu. Concordia: Distributed shared memory with In-Network cache coherence. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 277–292, 2021.

[47] Yuxin Ren, Gabriel Parmer, and Dejan Milojicic. Ch'i: scaling microkernel capabilities in cache-incoherent systems. In *2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*, pages 12–21. IEEE, 2020.

[48] D.E. Culler, A. Dusseau, S.C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel programming in split-c. In *Supercomputing '93:Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 262–273, 1993.

[49] Barbara Chapman, Tony Curtis, Swaroop Pophale, Stephen Poole, Jeff Kuehn, Chuck Koelbel, and Lauren Smith. Introducing openshmem: Shmem for the pgas community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*, pages 1–3, 2010.

[50] Mattias De Wael, Stefan Marr, Bruno De Fraine, Tom Van Cutsem, and Wolfgang De Meuter. Partitioned global address space languages. *ACM Comput. Surv.*, 47(4), May 2015.

[51] Idan Burstein. Nvidia data center processing unit (dpu) architecture. In *2021 IEEE Hot Chips 33 Symposium (HCS)*, pages 1–20. IEEE, 2021.

[52] Idan Burstein. Nvidia connectx-8 supernic: A programmable roce architecture for ai data centers. In *2025 IEEE Hot Chips 37 Symposium (HCS)*. IEEE, 2025.

[53] Linux-rdma perftest. https://github.com/linux-rdma/perftest, 2024.

[54] Mellanox Neo-Host. https://support.mellanox.com/s/login/?ec=302&startURL=%2Fs%2Fproductdetails%2Fa2v50000000N2OlAAK%2Fmellanox-neohost, 2025.

[55] DOCA Ngauge and DOCA Telemetry Diag. https://docs.nvidia.com/doca/sdk/doca+telemetry+diag/index.html, 2025.

[56] 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.

[57] Hugo Sadok, Nirav Atre, Zhipeng Zhao, Daniel S Berger, James C Hoe, Aurojit Panda, Justine Sherry, and Ren Wang. Ensō: A streaming interface for NIC-Application communication. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 1005–1025, 2023.

[58] Saksham Agarwal, Arvind Krishnamurthy, and Rachit Agarwal. Host congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 275–287, New York, NY, USA, 2023. Association for Computing Machinery.

[59] Alireza Farshin, Tom Barbette, Amir Roozbeh, Gerald Q. Maguire Jr., and Dejan Kostić. Packetmill: toward per-core 100-gbps networking. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery.

[60] Nikita Tyunyayev, Clément Delzotti, Haggai Eran, and Tom Barbette. Asni: Redefining the interface between smartnics and applications. *Proc. ACM Netw.*, 3(CoNEXT2), June 2025.

[61] Bowen Liu, Xinyang Huang, Qijing Li, Zhuobin Huang, Yijun Sun, Wenxue Li, Junxue Zhang, Ping Yin, and Kai Chen. Ceio: A cache-efficient network i/o architecture for nic-cpu data paths. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 381–394, 2025.

[62] Qijing Li, Xinyang Huang, Bowen Liu, Pengbo Li, Junxue Zhang, and Kai Chen. Cache-aware i/o rate control for rdma. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, APNET '25, page 9–16, New York, NY, USA, 2025. Association for Computing Machinery.

[63] Seyyidahmed Lahmer, Nikita Tyunyayev, and Tom Barbette. Opendesc: From static nic descriptors to evolvable metadata interfaces. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*, HotNets '25, page 271–279, New York, NY, USA, 2025. Association for Computing Machinery.

[64] Debra Hensgen, Raphael Finkel, and Udi Manber. Two algorithms for barrier synchronization. *International Journal of Parallel Programming*, 17:1–17, 1988.

[65] John M Mellor-Crummey and Michael L Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 9(1):21–65, 1991.

[66] Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. *The art of multiprocessor programming*. Newnes, 2020.

[67] Marc Snir. *MPI–the Complete Reference: the MPI core, Section 4.9*, volume 1. MIT press, 1998.

[68] Chenxingyu Zhao, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. Whiteboxing rdma with packet-granular software control. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, April 2025.

[69] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Ming Liu, and Zeke Wang. Demystifying datapath accelerator enhanced off-path smartnic. *arXiv preprint arXiv:2402.03041*, 2024.

[70] Mikhail Khalilov, Salvatore Di Girolamo, Marcin Chrapek, Rami Nudelman, Gil Bloch, and Torsten Hoefler. Network-offloaded bandwidth-optimal broadcast and allgather for distributed ai. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17. IEEE, 2024.

[71] Mikhail Khalilov, Siyuan Shen, Marcin Chrapek, Tiancheng Chen, Kenji Nakano, Nicola Mazzoletti, Peter-Jan Gootzen, Salvatore Di Girolamo, Rami Nudelman, Gil Bloch, et al. Sdr-rdma: Software-defined reliability architecture for planetary scale rdma communication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1223–1239, 2025.

[72] NVIDIA DPA Subsystem Programming Guide . https://docs.nvidia.com/doca/sdk/doca+dpa/index.html, 2024.

[73] Zhenhao He, Dario Korolija, Yu Zhu, Benjamin Ramhorst, Tristan Laan, Lucian Petrica, Michaela Blott, and Gustavo Alonso. ACCL+: an FPGA-Based collective engine for distributed applications. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 211–231, 2024.

[74] Martin D Schatz, Robert A Van de Geijn, and Jack Poulson. Parallel matrix multiplication: A systematic journey. *SIAM Journal on Scientific Computing*, 38(6):C748–C781, 2016.

[75] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70, 2020.

[76] Expectation Maximization with a Gaussian Mixture Model using CUDA. https://github.com/corv/cuda-gmm-multigpu, 2025.

[77] MultiGPU AdvectionDiffusion. https://github.com/wme7/MultiGPU_AdvectionDiffusion, 2025.

[78] 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, 2022.

[79] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R Lebeck, and Danyang Zhuo. Understanding RDMA microarchitecture resources for performance isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 31–48, 2023.

[80] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. SRNIC: A scalable architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1–14, 2023.

[81] Zilong Wang, Xinchen Wan, Luyang Li, Yijun Sun, Peng Xie, Xin Wei, Qingsong Ning, Junxue Zhang, and Kai Chen. Fast, scalable, and accurate rate limiter for rdma nics. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 568–580, New York, NY, USA, 2024. Association for Computing Machinery.

[82] Stanko Novakovic, Yizhou Shan, Aasheesh Kolli, Michael Cui, Yiying Zhang, Haggai Eran, Boris Pismenny, Liran Liss, Michael Wei, Dan Tsafrir, et al. Storm: a fast transactional dataplane for remote data structures. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 97–108, 2019.

[83] Jiaqi Lou, Xinhao Kong, Jinghan Huang, Wei Bai, Nam Sung Kim, and Danyang Zhuo. Harmonic: Hardware-assisted RDMA performance isolation for public clouds. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, April 2024. USENIX Association.

[84] 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, 2022.

[85] Zhuolong Yu, Bowen Su, Wei Bai, Shachar Raindel, Vladimir Braverman, and Xin Jin. Understanding the micro-behaviors of hardware offloaded network stacks with lumina. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 1074–1087, 2023.

[86] 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, 2019.

[87] Yibo Huang, Yiming Qiu, Yunming Xiao, Archit Bhatnagar, Sylvia Ratnasamy, and Ang Chen. Exposing rdma nic resources for software-defined scheduling. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, pages 1–8, 2025.

[88] Zhenghang Ren, Yuxuan Li, Zilong Wang, Xinyang Huang, Wenxue Li, Kaiqiang Xu, Xudong Liao, Yijun Sun, Bowen Liu, Han Tian, Junxue Zhang, Mingfei Wang, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. Enabling efficient gpu communication over multiple nics with fuselink. In *Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation*, OSDI '25, USA, 2025. USENIX Association.

[89] Boris Pismenny, Adam Morrison, and Dan Tsafrir. ShRing: Networking with shared receive rings. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 949–968, Boston, MA, July 2023. USENIX Association.

[90] Boris Pismenny, Adam Morrison, and Dan Tsafrir. Disentangling the dual role of NIC receive rings. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 651–669, 2025.

[91] Boris Pismenny, Liran Liss, Adam Morrison, and Dan Tsafrir. The benefits of general-purpose on-nic memory. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1130–1147, 2022.

[92] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. Enabling programmable transport protocols in High-Speed NICs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 93–109, Santa Clara, CA, February 2020. USENIX Association.

[93] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, and Amin Vahdat. Understanding host interconnect congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, HotNets '22, page 198–204, New York, NY, USA, 2022. Association for Computing Machinery.

[94] Dynamically Connected (DC) QPs. https://docs.nvidia.com/networking/display/rdmacore50/dynamically+connected+(dc)+qps, 2025.

[95] Yiren Zhao, Ran Shu, and Yongqiang Xiong. Src: A scalable reliable connection for rdma with decoupled qps and connections. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, pages 44–50, 2025.

[96] 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.

[97] Out-of-order Data Placement. https://docs.nvidia.com/doca/sdk/out-of-order-data-placement/index.html, 2025.

[98] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. Network load balancing with in-network reordering support for rdma. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 816–831, New York, NY, USA, 2023. Association for Computing Machinery.

[99] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. Multi-Path transport for RDMA in datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 357–371, Renton, WA, April 2018. USENIX Association.

[100] Sana Mahmood, Jinqi Lu, and Soudeh Ghorbani. Orderly management of packets in rdma by eunomia. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, pages 17–23, 2025.

[101] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 313–326, New York, NY, USA, 2018. Association for Computing Machinery.

[102] 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. Association for Computing Machinery, New York, NY, USA, 2019.

[103] 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.

[104] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 691–706, New York, NY, USA, 2024. Association for Computing Machinery.

[105] Qiang Li, Yixiao Gao, Xiaoliang Wang, Haonan Qiu, Yanfang Le, Derui Liu, Qiao Xiang, Fei Feng, Peng Zhang, Bo Li, et al. Flor: An open high performance RDMA framework over heterogeneous RNICs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 931–948, 2023.

[106] Torsten Hoefler, Duncan Roweth, Keith Underwood, Bob Alverson, Mark Griswold, Vahid Tabatabaee, Mohan Kalkunte, Surendra Anubolu, Siyuan Shen, Abdul Kabbani, et al. Datacenter ethernet and rdma: Issues at hyperscale. *arXiv preprint arXiv:2302.03337*, 2023.

[107] Sumit Kumar Monga, Sanidhya Kashyap, and Changwoo Min. Birds of a feather flock together: Scaling rdma rpcs with flock. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 212–227, 2021.

[108] Wenxue Li, Xiangzhou Liu, Yunxuan Zhang, Zihao Wang, Wei Gu, Tao Qian, Gaoxiong Zeng, Shoushou Ren, Xinyang Huang, Zhenghang Ren, Bowen Liu, Junxue Zhang, Kai Chen, and Bingyang Liu. Revisiting rdma reliability for lossy fabrics. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 85–98, New York, NY, USA, 2025. Association for Computing Machinery.

[109] Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. dcpim: Near-optimal proactive datacenter transport. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 53–65, 2022.

[110] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. Poseidon: Efficient, robust, and practical datacenter CC via deployable INT. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 255–274, Boston, MA, April 2023. USENIX Association.

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

[112] Benjamin Fuhrer, Yuval Shpigelman, Chen Tessler, Shie Mannor, Gal Chechik, Eitan Zahavi, and Gal Dalal. Implementing reinforcement learning datacenter congestion control in nvidia nics. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 331–343. IEEE, 2023.

[113] Soudeh Ghorbani, Yimeng Zhao, Srikanth Sundaresan, Ying Zhang, Yijing Zeng, Abhigyan Sharma, Prashanth Kannan, and Cristian Lumezanu. Congestion patterns in a large-scale rdma datacenter. In *Proceedings of the 2025 ACM Internet Measurement Conference*, IMC '25, page 944–951, New York, NY, USA, 2025. Association for Computing Machinery.

[114] Vamsi Addanki, Oliver Michel, and Stefan Schmid. PowerTCP: Pushing the performance limits of datacenter networks. In *19th USENIX symposium on networked systems design and implementation (NSDI 22)*, pages 51–70, 2022.

[115] Tommaso Bonato, Sepehr Abdous, Abdul Kabbani, Ahmad Ghalayini, Nadeen Gebara, Terry Lam, Anup Agarwal, Tiancheng Chen, Zhuolong Yu, Konstantin Taranov, et al. Uno: A one-stop solution for inter-and intra-data center congestion control and reliable connectivity. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1195–1210, 2025.

[116] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 1rma: Re-envisioning remote memory access for multi-tenant datacenters. In *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 708–721, New York, NY, USA, 2020. Association for Computing Machinery.

[117] Teng Ma, Kang Chen, Shaonan Ma, Zhuo Song, and Yongwei Wu. Thinking more about rdma memory semantics. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 456–467, 2021.

[118] 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.

[119] Haifeng Sun, Yixuan Tan, Yongtong Wu, Jiaqi Zhu, Qun Huang, Xin Yao, and Gong Zhang. Rb2: Narrow the gap between rdma abstraction and performance via a middle layer. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 1071–1080, 2024.

[120] Xinyi Chen, Liangcheng Yu, Vincent Liu, and Qizhen Zhang. Cowbird: Freeing cpus to compute by offloading the disaggregation of memory. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1060–1073, New York, NY, USA, 2023. Association for Computing Machinery.

[121] Zixuan Wang, Xingda Wei, Jinyu Gu, Hongrui Xie, Rong Chen, and Haibo Chen. ODRP:on-demand remote paging with programmable RDMA. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 1101–1115, 2025.

[122] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[123] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, 2017.

[124] Marcos K Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote memory in the age of fast networks. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 121–127, 2017.

[125] Quanxi Li, Hong Huang, Ying Liu, Yanwen Xia, Jie Zhang, Mosong Zhou, Xiaobing Feng, Huimin Cui, Quan Chen, Yizhou Shan, et al. Beehive: A scalable disaggregated memory runtime exploiting asynchrony of multithreaded programs. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 167–187, 2025.

[126] Lei Chen, Shi Liu, Chenxi Wang, Haoran Ma, Yifan Qiao, Zhe Wang, Chenggang Wu, Youyou Lu, Xiaobing Feng, Huimin Cui, et al. A tale of two paths: Toward

a hybrid data plane for efficient Far-Memory applications. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 77–95, 2024.

[127] Weigao Su and Vishal Shrivastav. Edm: An ultra-low latency ethernet fabric for memory disaggregation. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 377–394, 2025.

[128] Anil Yelam, Stewart Grant, Enze Liu, Radhika Niranjan Mysore, Marcos K Aguilera, Amy Ousterhout, and Alex C Snoeren. Limited access: The truth behind far memory. In *Proceedings of the 4th Workshop on Resource Disaggregation and Serverless*, pages 37–43, 2023.

[129] Yanpeng Yu, Nicolai Oswald, and Anurag Khandelwal. Cord: Low-latency, bandwidth-efficient and scalable release consistency via directory ordering. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, page 1311–1326, New York, NY, USA, 2025. Association for Computing Machinery.

[130] Mingxing Zhang, Teng Ma, Jinqi Hua, Zheng Liu, Kang Chen, Ning Ding, Fan Du, Jinlei Jiang, Tao Ma, and Yongwei Wu. Partial failure resilient memory management system for (cxl-based) distributed shared memory. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 658–674, 2023.

[131] Xu Zhang, Ke Liu, Yuan Hui, Xiaolong Zheng, Yisong Chang, Yizhou Shan, Guanghui Zhang, Ke Zhang, Yungang Bao, Mingyu Chen, et al. DRack: A CXL-Disaggregated rack architecture to boost Inter-Rack communication. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pages 1261–1279, 2025.

[132] Annus Zulfiqar, Ali Imran, Venkat Kunaparaju, Ben Pfaff, Gianni Antichi, and Muhammad Shahbaz. Gigaflow: Pipeline-aware sub-traversal caching for modern smartnics. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 467–481, 2025.

[133] Heng Yu, Jian Wang, Jian Zhao, Kai Ren, Guozhi Lin, Baozeng Zhang, Yunpeng Guan, Xin Li, Hao Yin, Jiajun Liang, Liang Wang, Chao Pei, Yachen Wang, Xin Jin, Jilong Wang, and Congcong Miao. Fornax: A hardware-centric session management in large public cloud network. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 663–676, New York, NY, USA, 2025. Association for Computing Machinery.

[134] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Songyuan Sui, Khalid Manaa, Omer Shabtai, Yonatan Piasetzky, Matty Kadosh, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. Unleashing smartnic packet processing performance in p4. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1028–1042, New York, NY, USA, 2023. Association for Computing Machinery.

[135] Shaofeng Wu, Qiang Su, Zhixiong Niu, and Hong Xu. Performance prediction of on-nic network functions with multi-resource contention and traffic awareness. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 828–842, 2025.

[136] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated smartnic offloading insights for network functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 772–787, 2021.

[137] Georgios P Katsikas, Tom Barbette, Marco Chiesa, Dejan Kostić, and Gerald Q Maguire Jr. What you need to know about (smart) network interface cards. In *International Conference on Passive and Active Network Measurement*, pages 319–336. Springer, 2021.

[138] Qiang Su, Shaofeng Wu, Zhixiong Niu, Ran Shu, Peng Cheng, Yongqiang Xiong, Zaoxing Liu, and Hong Xu. Meili: Enabling smartnic as a service in the cloud. *arXiv preprint arXiv:2312.11871*, 2023.

[139] Shixiong Qi, Songyu Zhang, K. K. Ramakrishnan, Diman Zad Tootaghaj, Hardik Soni, and Puneet Sharma. Palladium: A dpu-enabled multi-tenant serverless cloud over zero-copy multi-node rdma fabrics. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 1257–1259, New York, NY, USA, 2025. Association for Computing Machinery.

[140] Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafrir. Autonomous nic offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 18–35, 2021.

[141] Peng Jiang, Hanlin Jiang, Ruizhe Huang, Hanwen Lei, Zhineng Zhong, Shaokun Zhang, Yuxin Ren, Ning Jia, Xinwei Hu, Yao Guo, et al. Dpuaudit: Dpu-assisted pull-based architecture for near-zero cost system auditing. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 29–43. IEEE, 2025.

[142] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 756–771, 2021.

[143] Taehyun Kim, Deondre Martin Ng, Junzhi Gong, Youngjin Kwon, Minlan Yu, and KyoungSoo Park. Rearchitecting the TCP stack for I/O-Offloaded content delivery. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 275–292, 2023.

[144] Junyi Shu, Kun Qian, Ennan Zhai, Xuanzhe Liu, and Xin Jin. Burstable cloud block storage with data processing units. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 783–799, 2024.

[145] Qizhen Zhang, Philip A. Bernstein, Badrish Chandramouli, Jiasheng Hu, and Yiming Zheng. Dds: Dpu-optimized disaggregated storage. *Proc. VLDB Endow.*, 17(11):3304–3317, July 2024.

[146] Jiasheng Hu, Chihan Cui, Anna Li, Raahil Vora, Yuanfan Chen, Philip A Bernstein, Jialin Li, and Qizhen Zhang. dpbento: Benchmarking dpus for data processing. *arXiv preprint arXiv:2504.05536*, 2025.

[147] Yu Zhu, Aditya Dhakal, Pedro Bruel, Gourav Rattihalli, Yunming Xiao, Johann Lombardi, and Dejan Milojicic. An rdma-first object storage system with smartnic offload. In *Proceedings of the SC'25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1650–1657, 2025.

[148] Zerui Guo, Hua Zhang, Chenxingyu Zhao, Yuebin Bai, Michael Swift, and Ming Liu. Leed: A low-power, fast persistent key-value store on smartnic jbofs. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 1012–1027, 2023.

[149] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 106–122, New York, NY, USA, 2021. Association for Computing Machinery.

[150] Chenxingyu Zhao, Tapan Chugh, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. Dremel: Adaptive configuration tuning of rocksdb kv-store. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(2), June 2022.

[151] Henry N Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. Xenic: Smartnic-accelerated distributed transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 740–755, 2021.

[152] Yunming Xiao, Diman Zad Tootaghaj, Aditya Dhakal, Lianjie Cao, Puneet Sharma, and Aleksandar Kuzmanovic. Conspirator:SmartNIC-Aided control plane for distributed ML workloads. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 767–784, 2024.

[153] Tong Xing, Hesam Tajbakhsh, Israat Haque, Michio Honda, and Antonio Barbalace. Towards portable end-to-end network performance characterization of smartnics. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '22, page 46–52, New York, NY, USA, 2022. Association for Computing Machinery.

[154] Zihao Chang, Jiaqi Zhu, Haifeng Sun, Yunlong Xie, Kan Shi, Ninghui Sun, Yungang Bao, and Sa Wang. Poby:SmartNIC-accelerated image provisioning for coldstart in clouds. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pages 19–37, 2025.

[155] AWS Nitro System. https://aws.amazon.com/ec2/nitro/, 2023.

[156] Zerui Guo, Jiaxin Lin, Yuebin Bai, Daehyeok Kim, Michael Swift, Aditya Akella, and Ming Liu. Lognic: A high-level performance model for smartnics. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 916–929, 2023.

[157] Xing Li, Xiaochong Jiang, Ye Yang, Lilong Chen, Yi Wang, Chao Wang, Chao Xu, Yilong Lv, Bowen Yang, Taotao Wu, Haifeng Gao, Zikang Chen, Yisong Qiao, Hongwei Ding, Yijian Dong, Hang Yang, Jianming Song, Jianyuan Lu, Pengyu Zhang, Chengkun Wei, Zihui Zhang, Wenzhi Chen, Qinming He, and Shunmin Zhu. Triton: A flexible hardware offloading architecture for accelerating apsara vswitch in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 750–763, New York, NY, USA, 2024. Association for Computing Machinery.

[158] Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, Muhammad Shahbaz, Changhoon Kim, and Nick McKeown. The nanopu: A nanosecond network stack for datacenters. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 239–256, 2021.

[159] Jiaxin Lin, Zhiyuan Guo, Mihir Shah, Tao Ji, Yiying Zhang, Daehyeok Kim, and Aditya Akella. Enabling portable and High-Performance SmartNIC programs with alkali. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 107–126, Philadelphia, PA, April 2025. USENIX Association.

[160] David M Bloom. Birthday problem. *American Mathematical Monthly*, 80(10):1141–1142, 1973.

[161] Xuhao Luo, Shreesha G Bhat, Jiyu Hu, Ramnatthan Alagappan, and Aishwarya Ganesan. Lazylog: A new shared log abstraction for low-latency applications. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 296–312, 2024.

[162] Aishwarya Ganesan, Ramnatthan Alagappan, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Exploiting nil-externality for fast replicated storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 440–456, 2021.

[163] Amazon Simple Queue Service. https://aws.amazon.com/sqs/, 2025.

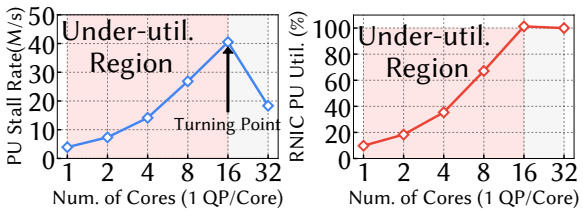
[164] Beihao Zhou, Samer Al-Kiswany, and Mina Tahmasbi Arashloo. Toward ebpf-accelerated pub-sub systems. In *Proceedings of the 3rd Workshop on eBPF and*

Kernel Extensions, pages 38–44, 2025.
[165] Junzhi Gong, Yuliang Li, Devdeep Ray, KK Yap, and Nandita Dukkipati. Octopus: A fair packet delivery service, 2024.
[166] George Varghese and Tony Lauck. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 25–38, 1987.

A Appendix

Appendices are supporting material that has not been peer-reviewed.

A.1 RNIC PU Utilization Measurements



(a) TX PU Stall Rate **(b) TX PU Utilization**

Figure 15: Converting RNIC PU Stall Rate into Utilization.

In this section, we describe our method for monitoring RNIC Processing Unit (PU) utilization. To access hardware performance counters, we use NVIDIA *Neo-Host* tool [54] for ConnectX-5 and *Ngauge* for BlueField-3 and ConnectX-8. To the best of our knowledge, neither tool provides direct counters for PU utilization. Instead, we rely on the counter "TX Descriptor Handling Stopped due to Work Done" to infer PU utilization. This counter indicates that the RNIC TX-side PU is stalled because it is waiting for new requests. As shown in Figure 15a, we monitor the PU stall rate (*i.e.*, the rate of "TX Descriptor Handling Stopped due to Work Done"). We observe that when increasing from 1 to 16 cores and more QPs, the PU stall rate rises. We attribute this to more QPs activating more PUs on the RNIC, with additional active PUs concurrently serving more QPs, thereby amplifying the stall rate. Beyond a turning point, the stall rate decreases, which we interpret as the state where all PUs are active and peak utilization is reached. Using this peak utilization as a reference, we convert PU stall rate into PU utilization. Before the turning point, utilization is computed as the stall rate normalized by the peak stall rate; after the turning point, we assume full utilization. The resulting PU utilization is shown in Figure 15b. We apply this conversion method to the results in Figure 3. Notably, DOCA *Ngauge* does not expose the counter "TX Descriptor Handling Stopped due to Work Done" on BF3/CX8. Therefore, we report PU utilization only for CX5, obtained via *Neo-Host*.

A.2 Episode-Length Analysis for Workload Applicability

Workload	Description	$\mathbb{E}$[Episode Length]
YCSB-Uniform [33]	records=10M	3,963
YCSB-Zipfian [33]	skewness=0.6	850
SpMM [73–75]	$\mathbb{R}^{1024\times1024}$, sparsity-ratio=0.1	10K
Tartan-Diffusion [34]	$N_x = N_y = 1024$, RADIUS=4	32K
SimAI-Training [35]	comm-size=64MB, fwd-pass-comm	O(128K)

Table 2: Average Episode Length for Workloads in Evaluation.

As described in §4.1, we define the number of requests without ordering constraints within an episode as the *Episode Length*. A

larger episode length indicates better applicability of the SID ordering model (§4.1) to a given workload. Table 2 presents the average episode length for the workloads used in our evaluation (§7.2). For YCSB [33], we approximately estimate the average episode length by reducing it to the Birthday Problem [160]: estimating the average length of an operation segment with no repeated records. For other workloads, we extract remote-memory-access snippets from source code and estimate the average episode length under common parameters. The results show that these workloads typically have large episode lengths, making the SID ordering model applicable.

A.3 Target-2: Publish-Subscribe Message-Queue Model

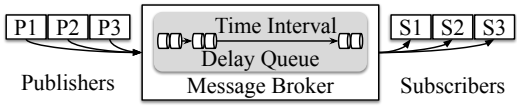


Figure 16: Publish-Subscribe Message Queue Model.

In addition to primary ML/HPC workloads, SID also targets cloud service workloads. The FlexOp Engine provides operations required by the common Publish–Subscribe (Pub/Sub) Message Queue model, as illustrated in Figure 16. The Pub/Sub model typically involves three types of roles: publishers that produce messages, brokers that store messages in queues, and subscribers that consume messages. This model is widely adopted in cloud workloads including logging systems [1, 161–163], notification services [3, 164], and trading systems [36]. Besides data transfer, SID extends the operation set to support Pub/Sub operations as follows:

Time-Sensitive Operations: Time services are widely used in Pub/Sub message queues. For example, publishers may require scheduled message transfers, or brokers in trading systems may perform precise timestamp calibration for accurate message ordering [36, 165]. To support such cases, the FlexOp Engine provides a timing system based on a hierarchical timing wheel [166]. Specifically, a DPA thread can submit a request to the timing system via an SQ, along with a specified delay. Once the delay elapses, the timing service enqueues the request into the CQ so the DPA thread can fetch and continue to process it. Via this mechanism, SID supports time-sensitive operations such as `Delayed_PUT`.

Delegation Operations: In the Pub/Sub model, publishers and subscribers can execute asynchronously. As a result, a publisher may produce messages that remain unconsumed for some time, or a subscriber may need to periodically poll the broker to check for new messages, incurring CPU overhead for continuous polling. To address these cases, the FlexOp Engine provides *delegation* operations. One such operation is `Delegated_Poll`, which allows polling to be offloaded from the subscriber to the SmartNIC, eliminating CPU overhead. Combined with the timing service, `Delegated_Poll` can also be configured to execute periodically at specified time intervals. Another operation is `Pub_Prepush`, which pre-pushes messages to the subscriber's SmartNIC. When the subscriber later needs to consume the message, it can fetch it locally, avoiding remote broker access and improving performance.

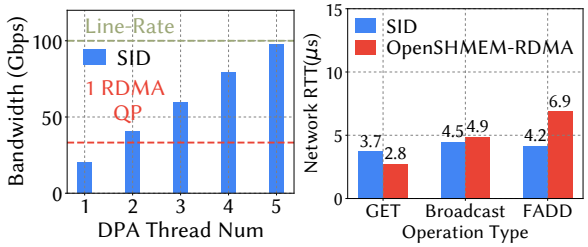
A.4 DPA Hardware Specification

	Host CPU	Datapath Accelerator
ISA	x86 Xeon	RISC-V
Cores	64 Cores	16 Cores
Threads	128 Threads	256 Threads
Memory	384GB DDR4	Local 1GB DDR5 + Load/Store Access to Host DDR
L1d Cache	1.5MB*32	1KB*256
L1i Cache	1MB*32	1KB*16
L2 Cache	40MB*32	1.5MB*1
L3 Cache	72MB*2	3MB*1
Network	General (*e.g.*, RDMA)	Ethernet SEND/RECV
OS	Linux	Real-Time OS

Table 3: Specifications of Host CPU and CX8/BF3 DPA.

In Table 3, we compare the DPA with a host CPU. The DPA adopts a low-cost design based on the RISC-V ISA and includes only modest amounts of SRAM cache.

A.5 Generality for Large-Size or Low-Latency Tasks



(a) Large Message **(b) Latency**

Figure 17: Large-Message and Low-Latency Operations.

Operation	P25 (μs)	P50	P99
OpenSHMEM-RDMA-PUT	4.11	5.50	7.68
SID-PUT	5.82	6.61	9.06
OpenSHMEM-RDMA-GET	4.43	5.68	8.79
SID-GET	5.76	6.56	9.18

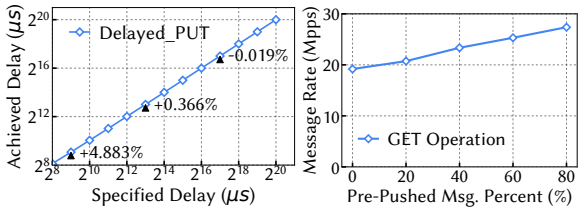
Table 4: Latency under heavy load with 64 connections.

Although we focus on fine-grained access, SID also demonstrates promising generality for other message types. Since attaining high bandwidth is inherently easier with large messages (*e.g.*, 1500B in Figure 17a), SID further enhances performance by substantially improving message rate. Furthermore, for larger messages, SID attains high bandwidth with significantly fewer processing threads than required for fine-grained transfers.

Figure 17b and Table 4 present latency results. In these tests, we issue only one request per batch to measure latency while avoiding queuing delay. The results show SID achieves comparable latency

to RDMA-based OpenSHMEM. Table 4 presents the latency results under heavy load with 64 concurrent connections. Despite the overhead of software interposition, SID achieves latency comparable to the ASIC-based RDMA solution. We reduce latency jitter through several optimizations: 1) utilizing the deterministic RTOS on the DPA (§6); 2) reducing the working-set size to fit within the limited DPA SRAM cache (§A.4); and 3) optimizing WQE fetching, as described in §5.3.

A.6 More Experiments on FlexOp Engine



(a) Time-Sensitive **(b) Pub_Prepush**

Figure 18: Time and Cache Ops of the FlexOp Engine.

In Figure 18, we present operations for the Pub/Sub Message Queue model to further showcase the flexibility.

In Figure 18a, we present results for time-sensitive operations. Specifically, we evaluate the achieved delay using `Delayed_PUT` operations under various delay durations specified by host users. Here, "achieved delay" refers to the time between when the host submits a request to the SQ and when it receives a completion signal from the CQ. The FlexOp Engine supports setting delay ranging from sub-millisecond to minutes. The results show that FlexOp can execute delayed operations with high precision. In contrast, many existing message queue services, such as AWS Simple Queue Service [163], only support second-level delay. In comparison, FlexOp achieves sub-millisecond delay accuracy. For example, the error is merely around 4% for a specified delay as low as 512 μs, given that the timing wheel ticks every 10 μs.

In Figure 18b, we evaluate the delegated operation `Pub_Prepush`. As described in §5.5, `Pub_Prepush` allows subscribers to fetch messages locally from their on-NIC interposition processor without accessing the remote broker. As the results show, increasing the percentage of pre-pushed messages enables subscribers to fetch a higher ratio of messages locally, thereby improving the overall message rate.

A.7 Suggestions for DPA Vendor

Based on our development experience with implementing SID on DPA, we suggest the following improvements:

- Enabling interoperability between FlexIO and DOCA_DPA: We note that there are currently two concurrent SDKs for programming the DPA: FlexIO and DOCA_DPA. We developed SID on FlexIO, but certain functionalities available in DOCA_DPA are also valuable—for example, enabling DMA issued by the DPA and registering callbacks for doorbells. From our understanding, interoperability between FlexIO and DOCA_DPA is currently difficult, as each SDK maintains its own process handler.
- Enhancing Window Register Mechanisms: We observe limitations in the number of windows that can be active simultaneously. In addition, windows registered for host DDR and GPU memory exhibit unnecessary ordering constraints.
- Easing Memory Address Management: Current memory address handling is confusing. For example, when the DPA accesses the host, the host address must be translated into a DPA address. In contrast, when the DPA initializes RDMA with data in host memory, `swqe->mem_ptr_send_data.addr` must instead be filled with the host address.
- Adding Performance Counters for RNIC PU Utilization: As noted in §A.1, neither *Neo-Host* nor the latest DOCA Telemetry (*Ngauge*) offer direct counters for PU utilization.
- Support for RDMA RC in FlexIO: The current FlexIO SDK supports programming raw Ethernet SEND/RECV, but it does not easily allow the use of the Reliable Connection (RC) transport mode.