White-Boxing RDMA with Packet-Granular Software Control

Chenxingyu Zhao and Jaehong Min, *University of Washington;*
Ming Liu, *University of Wisconsin-Madison;* Arvind Krishnamurthy, *University of Washington*

https://www.usenix.org/conference/nsdi25/presentation/zhao-chenxingyu

This paper is included in the
Proceedings of the 22nd USENIX Symposium on
Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

White-Boxing RDMA with Packet-Granular Software Control

Chenxingyu Zhao[1], Jaehong Min[1], Ming Liu[2], Arvind Krishnamurthy[1]
[1]University of Washington, [2]University of Wisconsin-Madison

Abstract

Driven by diverse workloads and deployments, numerous innovations emerge to customize RDMA transport, spanning congestion control, multi-tenant isolation, routing, and more. However, RDMA's hardware-offloading nature poses significant rigidity when landing these innovations. Prior workflows to deliver customizations have either waited for lengthy hardware iterations, developed bespoke hardware, or applied coarse-grained control over the black-box RDMA NIC. Despite considerable efforts, current customization workflows still lack flexibility, raw performance, and broad availability.

In this work, we advocate for *White-Boxing* RDMA, which provides control of the hardware transport to general-purpose software while preserving raw data path performance. To facilitate the white-boxing methodology, we design and implement *Software-Controlled RDMA (SCR)*, a framework enabling packet-granular software control over the hardware transport. To address challenges stemming from granular control over high-speed line rates, SCR employs effective control models, boosts the efficiency of subsystems within the framework, and leverages emerging hardware capabilities. We implement SCR on the latest Nvidia BlueField-3 equipped with Datapath Accelerators, delivering a spectrum of new customizations not present in legacy RDMA transport, such as Multi-Tenant Fair Scheduler, User-Defined Congestion Control, Receiver-Driven Flow Control, and Multi-Path Routing Selection. Furthermore, we demonstrate SCR's applicability for GPU-Direct and NVMe-oF RDMA with zero modifications to machine learning or storage code.

1 Introduction

Remote Direct Memory Access (RDMA) empowers data center workloads with high-performance networking thanks to hardware-offloaded transport. However, as deployments shift from single-tenant lossless fabric to multi-tenant lossy Ethernet and applications expand from HPC to storage and GPU communication, a one-size-fits-all approach to RDMA hardware transport is inadequate. Consequently, extensive innovations have emerged to customize RDMA transport. From the industry, Microsoft Hyperscale Deployment [1–3], Google Falcon [4], AWS SRD [5], Meta [6], Alibaba HPN [7], and the Ultra Ethernet Consortium (UEC) [8], among others, are driving RDMA transport customizations at scale. This is supplemented by academic contributions spanning topics such as congestion control [9–16], multi-tenant isolation [17–19], routing [20, 21], reliability [22–24], and more.

Although the innovations are in full swing, RDMA's hardware nature imposes significant rigidity on integrating these customizations. Fundamentally, ASIC hardware iterations are expensive both in terms of time and monetary cost. For instance, DCQCN congestion control [2] baked into ASICs by Microsoft and vendors of RDMA NIC (or RNIC) has persisted for nearly a decade. Despite subsequent congestion control innovations, the hardware rigidity has hindered the integration of new ideas. Some cloud providers have explored bespoke hardware or FPGA RNICs [25, 26], but these solutions still require substantial efforts and lack widespread availability compared to commercial RNICs. Alternatively, software-based approaches [18, 24, 27, 28] treat RNIC hardware transport as a black box, enforcing customizations through upper-layer mediators. However, these software overlay mediators are coarse-grained at the message/verb granularity (up to GB in size) instead of the transport-layer packet granularity (typically 1024B) and introduce additional processing overhead, leading to latency penalties and security concerns [17].

Inspired by the successful practice of white-boxing switches [29–34], we propose *white-boxing* RDMA, a new methodology to customize RDMA hardware transport. Historically, switch vendors tightly integrated control logic with hardware data planes, limiting flexibility for user-defined control. With white-box switches, control planes run in customizable software while preserving the data plane in high-performance hardware, sparking a series of SDN innovations over the last decades. Similarly, we advocate the same methodology for the current RNIC; we expect to take transport control from the hardware into software for flexible customization while keeping the data path in performant hardware.

Besides the need for customization, another key enabler for white-boxing RDMA is the advancements of SmartNICs [35–40]. SmartNICs provide two building blocks for white-boxing RDMA: computational power and control interfaces. Taking the latest Nvidia BlueField-3 (BF3) as one example, it interposes general-purpose Datapath Accerlators (DPA) in line with the data path, potentially supporting flexible control logic without incurring host-side CPU overhead. Moreover, DPA features heavy multi-threading capabilities to keep pace with line-rate processing. DPA also leverages low-profile RISC-V designs to achieve energy and cost efficiency. These advancements pave the way for white-boxing RDMA.

Realizing the vision of white-boxing RDMA poses two main challenges: flexibility and efficiency. In terms of flexibility, we aim to accommodate diverse control functionalities that rely on various control signals, versatile control laws, and control actions. As for efficiency, fine-grained control over RDMA transport encounters challenges in keeping pace with high-speed line rates. For instance, at 100 GbE with a

1500B MTU, achieving 8.3 Mpps generates packet-granular signals (such as ACK and TX) at the sub-μs level (120 ns!). Processing signals at this granularity demands high efficiency, as even single access to DDR memory (hundreds of ns) entails a relatively significant time cost. Therefore, boosting processing efficiency is essential.

In this work, we introduce a control framework called *Software-Controlled RDMA (SCR)*. Drawing from classical control theory [41–43], SCR centers on two fundamental questions: *What to Control* and *How to Control*. Addressing the first question, we propose the *Dequeue Rate* control model, which regulates the rate at which the RNIC fetches messages from memory regions associated with queue pairs. This control knob serves diverse purposes across local-host, fabric, and peer-receiver domains (detailed in §4.2). As for *How to Control*, the SCR framework comprises three subsystems: signal/event collector, event queues, and event processor. These subsystems are carefully designed to tackle the challenges of flexibility and efficiency. The event collector aims to gather comprehensive signals from various domains, while event queues enhance efficiency through coalescing signals. For event processors, we build a Deterministic Multi-Threading (DMT) system to achieve high-efficiency and general-purpose processing (detailed in §4.3).

We prototype SCR on the latest Nvidia BlueField-3 SmartNIC. Specifically, we leverage BF3 DPA's *central* architecture position: DPA can interact with RNIC hardware, communicate with the host CPU, and inject traffic into the fabric. Moreover, implementing SCR using the DPA minimizes the host CPU's computational burden and operates in a *non-intrusive* manner for host-side applications and libraries. We successfully deliver a spectrum of new functionalities not present in legacy RDMA transport: 1) Achieving fair multi-tenant QP scheduling (up to 1.78x improvement in fairness index compared to the default scheduler; up to 52.8% latency reduction); 2) Enabling user-defined congestion control (Swift [10]); 3) Enabling multi-path routing selection (MP-RDMA [20]; PLB [44]); 4) Enabling receiver-driven control (Homa [15]). Importantly, we demonstrate that storage scenarios using NVMe-oF RDMA and machine learning using GPU-Direct RDMA can benefit from these customizations without any code changes to applications and libraries.

2 Background and Motivation

2.1 Why Customize RDMA

We first comprehensively summarize prior efforts in customizing RDMA transport and then identify the implications (*I*) for an RDMA customization framework.

• Better Congestion Control: To achieve lossless fabric, RoCEv2 employs PFC, which, despite its effectiveness, faces issues like PFC storms, HoL blocking, and deadlocks [3,45,46]. Mitigating these drawbacks, numerous congestion control algorithms have been developed, including Microsoft's ECN-based DCQCN [2], Alibaba's INT-based HPCC [9], Google's RTT-based Timely [47] and Swift [10], and ACK-Based ACC [11], showcasing how to customize RDMA congestion control, especially for hyper-scale lossy Ethernet. *I1: New types of signals from the transport layer enable improved forms of congestion control laws.*

• Stronger Multi-Tenancy Performance Isolation: RDMA transport bypasses in-kernel multi-tenant isolation mechanisms, resulting in inadequate multi-tenancy isolation within cloud environments. Studies such as Husky [19,48], and Harmonic [17] have demonstrated a pressing need for improved performance isolation among multi-tenants sharing RNIC. Solutions like Flor [24], X-RDMA [49], and Justitia [18] introduce a software mediation layer between the tenant and the RNIC driver, aiming to improve multi-tenant isolation. *I2: New measurements reveal the need to control new domains like RNIC micro-architecture and multi-tenant host.*

• Receiver-driven Flow Control: The many-to-one traffic pattern, common in GPU collective communications [50] and storage IO [1], often leads to incast congestion at the responder. This challenge is intensified by RDMA's one-sided operations, which are agnostic to responder-side controls. RCC [51] proposes receiver-driven congestion control for RDMA, with EQDS [16], Homa [15], NDP [52], and pHost [53] advocating for a receiver-driven control loop. *I3: Emerging workload patterns motivate new types of flow controls, such as receiver-driven flow control.*

• Per-Packet Multi-Path Routing: RoCEv2's default single-path routing ensures in-order packet delivery but fails to utilize the fabric's multiple paths. MP-RDMA [20] introduces mechanisms for multi-path routing. AWS SRD [5] emphasizes multi-path routing benefits for RDMA in the production environment. And switches such as Nvidia Spectrum [54] and Broadcom Tomahawk [55] support adaptive routing (per-packet ECMP). *I4: New types of routing controls like multi-pathing are beneficial for RDMA transport.*

RDMA transport customization is vast in the literature, and we present additional studies in Appendix Table 6.

2.2 How Prior Works Deliver Customization

RNIC offloads control functions onto rigid ASIC hardware, making reprogramming most customizations difficult. We compare prior efforts to deliver customizations as follows:

1) One approach is collaborating with RNIC vendors to integrate customizations into next-gen ASICs, such as Microsoft/Mellanox incorporating DCQCN into ConnectX. However, this method relies on hardware generation cycles, which typically span years.

2) Developing a bespoke RNIC from scratch using FPGA or ASIC. FPGA-based solution [17, 20, 26, 56, 57], despite significant engineering efforts, still generally exhibit discrepancies compared to native commercial RNIC ASICs, which have evolved over almost a decade to become a commodity. Cloud providers like AWS SRD [5], Google 1RMA [25], and

Falcon [4] can tape out custom ASICs, yet requiring substantial costs and expertise, and they are primarily utilized for internal operations rather than as broad commodities.

3) Software overlay solutions [16, 18, 24, 28, 58] provide userspace libraries over RNIC drivers. However, host software is limited to operating at Verb-level (*i.e.,* message level) and still treats the hardware transport as a black box to work around. Userspace libraries are not transparent to the host software stack because they customize drivers or insert mediators between the RNIC and applications. This added software mediator inevitably incurs a latency penalty [18, 24, 28], consuming host CPU cycles, and could raise security concerns for cloud tenants [17].

4) Fully software emulation like SoftRoCE [59] lacks the crucial property of hardware-accelerated data paths.

2.3 Our Goals

Flexible and Packet-Granular Control: In delivering RDMA customization, software agility surpasses hardware rigidity. Thus, we aim to leverage flexible software to enable diverse control functionalities. We aim to enforce packet-granular control at the hardware transport, a lower layer than userspace message-granular coarse-grained control.

Raw High-Performance Data Plane: High performance is crucial for RDMA; otherwise, people could opt for fully programmable transport like in-kernel TCP/IP. Thus, we adhere to the principle of *Zero-Intrusion* (or *Zero-Touch*) for the hardware data plane to maintain its original high performance.

Deployment Ready: We utilize commodity hardware to provide ready-to-deploy solutions and adhere to the principle of *smart-edge-dumb-core*, placing complex functionalities on end-host devices while keeping in-network switches simple. Next, we explore the capabilities of commodity hardware.

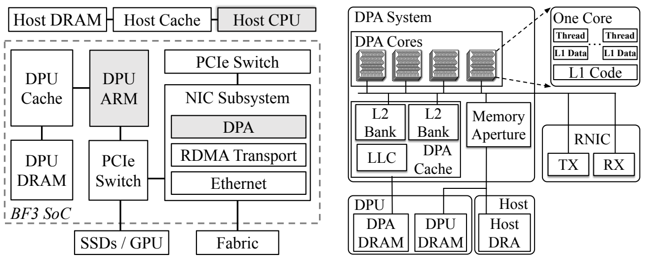
3 Characterizing Commodity Hardware

3.1 Requirements of Control System

Before characterizing existing hardware, we first examine the capabilities required for an ideal control system. Drawing from classic control theory's signal-driven feedback-loop [41–43], we identify three key components for an ideal control system: *signal collectors*, *processors*, and *action executor*. To achieve our design goals, these three components necessitate specific hardware capabilities. These requirements drive our following characterization.

3.2 Existing Hardware Model

As shown in Figure 1a, with the advancement of SmartNICs [35, 37, 38, 62], several processors within one server can be potentially utilized to implement the control system. Taking the latest BlueField-3 as an example, we have three options: Host CPU, DPU-embedded ARM CPU (off-path cores), or Datapath Accelerators (on-path cores). Next, we explore their potential capabilities and identify a suitable candidate.



(a) System Model (b) DPA Architecture

Figure 1: **System Model and DPA Architecture. Referring to Public Information Provided by the Vendor [60, 61].**

	Host CPU	**DPU ARM**	**DPA**
ISA	x86 Xeon	Arm v8.2+	RISC-V
Cores	64 Cores	16 Cores	16 Cores
Threads	128 Threads	16 Threads	256 Threads
Memory	384GB DDR4	32GB DDR5	Local 1GB DDR5 + Load/Store Access to Host/SoC DRAM
L1d Cache	1.5MB*32	64KB*16	1KB*256
L1i Cache	1MB*32	64KB*16	1KB*16
L2 Cache	40MB*32	0.5MB*16	1.5MB*1
L3 Cache	72MB*2	16MB*1	3MB*1
Network	General	General	Raw Ethernet
Interaction with RNIC	Verb-Granular	Verb-Granular	Packet-Granular
OS	Linux	Linux	Real-Time OS
Requirements Satisfied	**R1/R3/R4**	**R1*/R3**	**R1*/R2/R3/R4**

(* denotes partial satisfying)

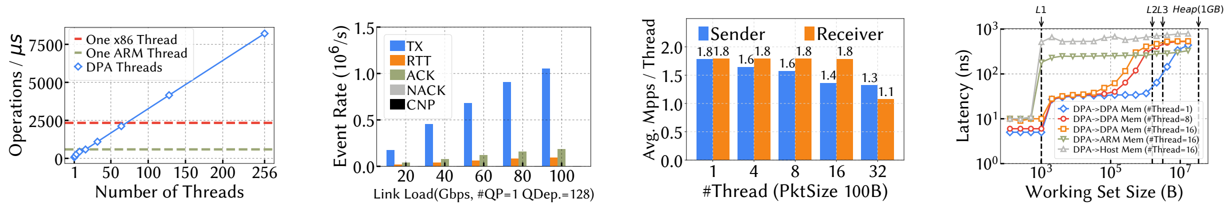
Table 1: **Comparing Heterogeneous Processors.**

Testbed: Our testbed (used throughout the paper) consists of two servers connected to a 100GbE switch. Each server has dual Intel Xeon Gold 6346 processors, 384GB DDR4, PCIe Gen4, and an Nvidia BlueField-3 DPU with 100GbE cables. Each server is equipped with one Nvidia A30 GPU (24GB HBM2, 933GB/s memory bandwidth) with driver-550 and CUDA v12.4. The host machines run Ubuntu 22.04 with kernel 6.5.0, DOCA SDK v2.6, and SPDK v24.05. We configure SR-IOV VFs for multi-tenant support.

3.3 Characterizing Datapath Accelerators and Others

As Table 1 shows, we characterize the Datapath Accelerator (DPA) and compare it with Host CPUs and DPU ARM cores. We focus on micro-benchmarking the DPA because extensive research has already been conducted on x86/ARM cores [39, 63–66]. Chen et al. [63] characterizes the DPA from the architectural perspective. We investigate DPA functionalities relevant to the control requirements mentioned, for example, the fine-grained interaction between RNIC and DPA (*e.g.,* Programmable Congestion Control), not covered in [63].

DPA Overview: DPA is a general-purpose on-datapath co-processor. Figure 1a illustrates the conceptual position of the DPA within the BF3 overall architecture, and Figure 1b provides a detailed view of the DPA's micro-architecture. As Table 1 lists, DPA comprises 256 hardware threads and possesses its own cache and memory system. DPA can access the host memory and DPU on-SoC memory with load/store



(a) Multi-Thread Execution (b) PCC Events (WRITE Verb) (c) L2 Send/Receive (d) Memory Access Latency

Figure 2: **Characterizing Hardware Capabilities of Datapath Accelerators (DPA).**

instructions (*i.e.,* via memory aperture [60]). DPA can operate the RNIC, which is co-packaged on the same PCIe card.

For R1, Multi-threading Performance: DPA features heavily multi-threaded computation with 256 hardware threads (even more than the host's beefy x86 CPU). In Figure 2a, we conduct General Matrix Multiplications (GEMM) to assess scalability while increasing the number of threads (Each thread computes the tiled outer product of GEMM [67]). Each operation represents one multiplication. The scalability trend closely approaches a linear relationship.

Notably, DPA cores in low-profile and low-cost RISC-V are much less powerful than x86/ARM cores. As shown in Figure 2a, it may require up to 16 DPA threads to achieve equivalent processing capability as one ARM thread. Therefore, tailoring the functionality over a single DPA thread, which involves a low amount of code and lightweight operations, becomes necessary. Although DPA offers 256 threads, given these low-code constraints, we believe that the wimpy cores partially satisfy the R1 requirement.

For R2, Interaction with RNIC: The RNIC offers the Programmable Congestion Control module [68] for the DPA, facilitating fine-grained interactions between the RNIC and DPA. These include the DPA receiving per-packet signals (TX/ACK/NACK/CNP, detailed definitions in §4.3.1), measuring RTT by injecting the RTT request packet into the RNIC, and the RNIC providing APIs for the DPA to set per-flow rates. In Figure 2b, we measure the event rates while varying the link load. It is evident that the DPA can receive packet-granular events. Note that NACK (packet loss) and CNP (congestion) events are rare; hence, the bars representing them are small.

For R3, Raw Ethernet Capability: Since we need to send and receive small-sized control signals over fabric, we test DPA's communication APIs for transmitting data. These APIs operate at the raw L2 Ethernet level, allowing for flexible customization of L3/L4 protocols such as TCP/IP or UDP/IP. Figure 2c illustrates the performance of L2 sending and receiving. For a single thread, it achieves packet rates of about 1.8 Mpps, meeting the requirements for μs-scale communication. As the number of threads increases, single-thread performance degrades due to contention, but the total Mpps increases. Since control signal packets are generally small, we focus on Mpps for small packets, such as 100B.

For R4, Memory Access Latency: Signal collectors interact with the host ideally via memory access. In Figure 2d, we measure the latency for DPA accessing the local cache, local memory, DPU on-SoC DRAM, and host-side DRAM. On workloads, we use the memory bandwidth test from *lmbench* [69], a classical micro-benchmark suite. We highlight several takeaways: DPA accessing L1 is significantly faster than accessing the DDR memory; DPA accessing local memory is faster than accessing the DPU memory and host memory; DPA L1 and L2 are precious and limited resources shared by multiple threads. Thus, during the design and implementation, we should keep the stateful memory (*e.g.,* per-flow states) compact and reduce memory access times.

We discuss the host CPU's and DPU ARM's limitations in meeting the above four requirements in Appendix §A.1. Compared to DPA, the host CPU and DPU ARM lack a control interface for requirement R2 and face performance issues for the other requirements.

Overall, we conclude that DPA is a suitable but not perfect candidate among three. We must address challenges like wimpy core constraints on low-code execution and limit states maintained to improve cache locality. Importantly, for generality, we loosely couple our design with any specific hardware. If DPA's APIs are exposed to host CPU and ARM cores, we can adapt our design to be compatible with them.

4 Software Control RDMA Framework Design

4.1 Overview

The key questions for Software-Controlled RDMA (SCR) are *What to Control* and *How to Control*. Firstly, we introduce a control model that forms the foundation of SCR to address the first question (§4.2). Next, addressing the second question relies on the event-driven rate computation framework (§4.3).

4.2 Dequeue Rate Control Model

QP Scheduler Model: We first examine RNIC's QP scheduler, which serves as the infrastructure for controlling network traffic at the sender side. Figure 3 illustrates the behavior of the RNIC QP scheduler, also functioning as the rate limiter: When tenants post WQEs to the SQ, the host driver notifies the RNIC via doorbell to update QP context (QPC), and the RNIC fetches several WQEs to aid in scheduling decisions. The scheduler leverages per-QP metadata, such as QPC and WQE, along with per-QP rates, to determine the execution order of WQEs. Typically, these per-QP rates are allocated by the congestion control module, such as DCQCN. For simplic-

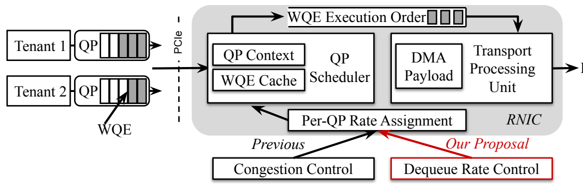


Figure 3: **RDMA NIC Queue Pair Scheduling Model.**

ity, we focus solely on the scheduler's input (metadata and rate assignment) and output (WQE execution order), while the internal structures of schedulers fall outside the scope of our work. Rich literature investigates the design of schedulers and rate limiters [70–72] (Tassel [70] specifically for RNIC). After the scheduler generates the WQE execution orders, the Processing Units of the Transport Logic (*e.g.,* RoCEv2) dequeue the WQE, DMA the payload from host memory, and complete packetization before sending it out over the wire.

	Standalone Metric	Competing with BW-Hungry	BW-Hungry QP (64KB Msg.)
Th.-Sensitive (4KB Msg.)	88.30 Gbps	33.25 Gbps	57.64 Gbps
Lat-Sensitive (256B Msg.)	2.44 us	5.31 us	90.57 Gbps

Table 2: **Limitations of DCQCN-Only Rate Assignment.**

Although the current RNIC QP scheduler delegates the per-QP rate assignment to fabric congestion control like DCQCN, unfortunately, relying solely on fabric congestion to control the sending rate has limitations. For instance, as shown in Table 2, consider a scenario where tenants compete with a bandwidth-hungry QP. We observe that the throughput-sensitive tenant experiences significant degradation and unfairness in throughput while the latency-sensitive QP suffers from increased latency. A desired behavior is that the throughput-sensitive QP can fairly share bandwidth with the bandwidth-hungry QP, while the latency-sensitive QP experiences no latency increase when competing with the bandwidth-hungry QP. It is worth noting that their aggregate bandwidth is at line rate, indicating no congestion in the fabric, so DCQCN does not react accordingly. Many prior works also report similar inadequacies in RDMA congestion control for multi-tenant scenarios [17, 18, 24, 28, 73].

Dequeue Rate: We observe that the input of the QP scheduler is a powerful control knob that can go beyond the congestion control's sending rate. As Figure 3 shows, we leverage this input knob as the per-QP *Dequeue Rate*, reflecting the rate at which the RNIC retrieves messages from memory regions. For instance, in WRITE verbs, the WQEs dequeued from the SQ indicate the message size per request, and then the dequeue rate is determined by the total message sizes over a given time period. The choice of the dequeue rate originates from its beneficial *multiplicity* as follows.

Multiplicity of Dequeue Rate: As Figure 4 shows, the dequeue rate influences multiple aspects of RDMA transport: 1) Primarily, it impacts the sending rate, as messages can only

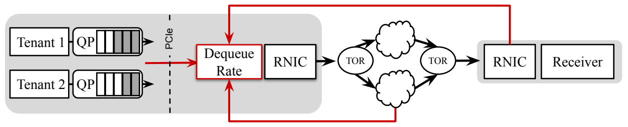


Figure 4: **Dequeue Rate Affects Multiple Domains.**

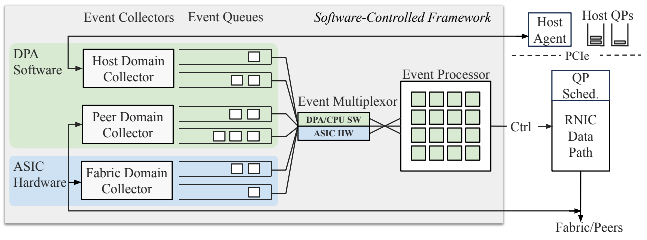


Figure 5: **Event-Driven Rate Computation Framework.**

be sent out by the RNIC hardware once dequeued from the host; 2) It affects PCIe utilization because a higher dequeue rate leads to increased PCIe utilization for DMA transfers; 3) The dequeue rate determines resource allocation within RNIC hardware, such as ASIC Processing Units, since dequeue operations initiate subsequent hardware processes like packetization; 4) The dequeue rate can respond to back pressure from the receiver side, as slowing down the sender's dequeue rate can alleviate pressure on the receiver side. In Appendix §A.2, we empirically evaluate how the dequeue rate influences various aspects mentioned above. Overall, the dequeue rate offers a comprehensive control, making it a suitable control knob for SCR.

4.3 Event-Driven Rate Computation Framework

With the dequeue rate control model, we next design an event-driven framework to collect multi-domain signals, compute the rate, and output the control actions. As Figure 5 shows the workflow, we next describe the subsystems of the framework.

4.3.1 Event Collectors Subsystem

Design Goals: As prior literature shows, the key enabler for flexible control is collecting accurate and diverse signals from multiple domains (*i.e.,* signal sources). Therefore, the primary objective of event collector systems is to be *all inclusive*, capable of providing a wide range of events. Furthermore, while we strive to collect more signal types, we prioritize *deployment-friendly* solutions that utilize existing features of NICs and switches rather than requiring specialized devices. Next, we outline how to achieve these design goals:

Multiple Domains: To control the dequeue rate, we gather signals from three primary domains: *Fabric Domain*, *Host Domain*, and *Peer Domain*. Table 3 lists the specific signals collected from each domain. These domains employ distinct mechanisms to gather events as follows:

Fabric Domain: We employ two types of collecting mechanisms: 1) In-Band Mechanisms: In-band packets carry signals such as CNP/ACK/NACK. Additionally, the TX/RX pipe of the RNIC can generate TX/RX events during network trans-

Domain	Signal	Definition
Fabric Domain	TX	Completion of transmitting a burst of packets, including timestamp and total byte count
	RX	Completion of receiving a burst of packets, including timestamp and total byte count
	CNP	Congestion Notification Packet (*e.g.,* ECN)
	ACK	Acknowledgements
	NACK	Negative Acknowledgements
	RTT	Measurements of Round Trip Time
Host Domain	App-Hints	Application-specific policies (*e.g.,* Fair bandwidth sharing among tenants)
	PCIe Util.	PCIe real-time bandwidth and latency
	RNIC Util.	RNIC hardware resource utilization
Peer Domain	Receiver Credits	Rate at which peer can handle incoming traffic
	Peer PCIe Util.	Peer node's PCIe bandwidth and latency
	Peer RNIC Util.	Peer RNIC's resource utilization

Table 3: **Signals From Multiple Domains. This list includes typical signals but can be expanded as needed.**

fer. RTT can also be obtained by timestamping the in-band packets. Some commodity RNICs, such as the BF3 used in this project, already support the collection of in-band signals such as CNP/ACK/NACK/TX/RTT. To our knowledge, the current BF3 does not support RX signal collection. As an alternative, we enable RX signal collection through host domain collectors, which will be discussed later. 2) Out-of-Band Mechanisms: Out-of-band packets probe network status in a telemetry manner. For probing packets, we implement the sender and receiver using the BF3 DPA, which supports communication over the fabric as shown in §3.3. We design a UDP-based protocol for transferring out-of-band messages over raw Ethernet APIs. Several reasons support the choice of UDP-based protocols: RoCEv2 is based on UDP, allowing one to track the header field of the RDMA in-band connection and reuse the meta info for associated out-of-band transfers. Additionally, UDP alleviates the burden of maintaining connections and handling reliability. An example of out-of-band probing is RTT measurements. The sender initiates out-of-band RTT request packets that are IP-routable. Upon the receiver NIC's ping-pong response to the RTT request packets, the sender can calculate the RTT. Unlike in-band RTT measurements with static per-ACK granularity, out-of-band RTT offers flexibility in the frequency of sending out RTT request packets. Also, out-of-band RTT carried as payloads is more flexible than in-band RTT, avoiding packet header format modifications.

Host Domain: We design several methods to collect host-domain signals: 1) For application hints, the event collector provides a shareable memory region that applications can access using load/store instructions, allowing them to write signals with sub-μs latency. Specifically, we implement the collector using two components: a lightweight agent on the host CPU and a process on the DPA. The host agent collects application hints and writes them to a shared memory region, from which the DPA process reads the signals. 2) For PCIe utilization measurement, we employ two methods: The first method involves the host-side agent utilizing tools like Intel PCM [74] to query real-time stats, followed by writing these stats to the event collector via the shared memory region. The second method entails implementing a proactive approach using PCIe endpoints like NIC to detect PCIe utilization. Specifically, the DPA process initiates a ping, to which the host agent responds with a pong, allowing the DPA to analyze PCIe latency. Importantly, both the host agent and the DPA process are configured per node rather than per flow, ensuring there are no scalability issues. This implementation is efficient because all flows within the same server typically share the same RNIC, host, and PCIe interconnect. 3) For RNIC utilization, we can utilize the RNIC hardware's query API to obtain real-time status. Also, we analyze the fabric domain signals to infer current RNIC utilization, such as the number of active connections.

Peer Domain: Signals of this domain must traverse the network fabric between the requester/responder. Similar to the fabric domain's out-of-band mechanism, we operate raw Ethernet packets to send/receive signals between local/peer nodes. We employ a UDP-based protocol to ensure the signal is routable while eliminating unnecessary heavy-weight mechanisms like connection setup and reliability. This approach achieves low latency and low processing overhead. Peers initially gather node information, including application hints, PCIe, and RNIC utilization, using the method outlined above for the host domain. These signals can serve as credits from the receiver to throttle the sender's dequeue rate. Once peers have gathered their information, they can transfer it using the lightweight UDP-based protocol. In our practice, using raw UDP with 128 Bytes payloads is sufficient to encapsulate essential information like flow ID and credits. We implement the sender and receiver using the BF3 DPA on both local and peer nodes. The sender injects packets into the network in several modes: periodically, sampling, and based on anomaly triggers. In the periodic approach, we set the packet interval to one RTT, balancing timely information delivery with moderate network usage. Considering that one DPA thread can handle 1.8+ Mpps, as detailed in §3.3, we can consolidate senders for multiple QPs into a single thread. With a typical RTT setting of 20 μs [75], a single thread can support up to 36 flows. In the sampling approach, the sender transmits out-of-band packets only after a specified number of in-band bytes have been sent. The anomaly-triggered method conserves network bandwidth by activating only when significant changes are detected, such as changes in the PCIe or port utilization.

4.3.2 Event Queues and Multiplexer Subsystem

Motivating Queuing and Coalescing: Event queues temporally store events[1] from the collectors before the multiplexer dequeues events and invokes the processor to handle them. The event queue subsystem is to facilitate *event coalescing*, a crucial mechanism for managing high-speed packet-granular signals. In Figure 6, we motivate the coalescing mechanism using the TX events example. Three factors contribute to

[1]Events encapsulate signals and their associated metadata such as flowID. We use the terms of events and signals interchangeably in the paper.

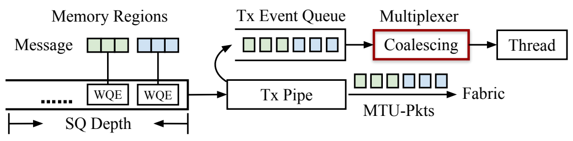


Figure 6: **Coalescing Mechanism (TX Events Example).**

the rapid accumulation of per-packet TX events: Firstly, in RDMA operations such as WRITE, each WQE can indicate a large message size (up to GB), while the wire has a typically small and fixed MTU size (*e.g.,* 1024B). Therefore, a message associated with a single WQE could be segmented into a large number of MTU-sized packets. Secondly, for SQ, users often configure a large queue depth to enhance utilization through batching, further speeding up the accumulation of per-packet signals. Thirdly, depending on the complexity of control laws, the processor may operate slowly. This speed may be outpaced by the event arrival rate, resulting in event queue buildup. If we process each per-packet event sequentially, the event queue will quickly fill up. Thus, coalescing multiple consecutive events into one event is a promising solution to improve processing efficiency. We micro-benchmark the effectiveness of coalescing in Appendix §A.3. Importantly, in addition to the TX signals from the fabric domain, signals from other domains can also benefit from coalescing, driven by similar motivations.

Design Goals: According to the above observations, the queuing system aims at 1) Supporting efficient and flexible coalescing strategies; 2) Applying to various types of signals. To achieve these two goals, we not only utilize existing functionalities in ASIC hardware but also introduce new functionalities in DPA software, as detailed below.

Hardware Coalescing/Multiplexing: As described in §4.3.1, event collectors for in-band signals, such as TX, ACK, and CNP events, are ASIC hardware modules positioned with the RNIC TX/RX pipelines. In addition to the event collectors, the current BF3 ASIC hardware also supports coalescing these in-band signals and subsequently invoking DPA threads for processing. However, to our knowledge, the current hardware coalescing and multiplexing approach has two main limitations: 1) Applicability: It is limited to in-band signals from the fabric domain, excluding other signal types from domains such as Host or Peer Domains, which cannot leverage these hardware circuits; 2) Flexibility: The hardware supports only fixed strategies for coalescing and multiplexing. Specifically, for coalescing, it only supports an *Accumulation* strategy, which accumulates values indicated by each event and outputs a single summary event. For multiplexing, it periodically invokes a thread based on hashing results of the flow ID associated with the events. Critically, it does not provide a programming interface for users to define more versatile coalescing and scheduling strategies. To address these shortcomings, we enhance the applicability and flexibility of the queuing system with software implementations using DPA.

Software Coalescing: As we previously described in §4.3.1, for out-of-band signals and host domain signals, we employ software-based collectors over the DPA cores (Figure 5 denoting DPA Software). These types of signals can not benefit from hardware coalescing. Consequently, the event coalescing and multiplexing for these out-of-band signals are software-based implementations over DPA. Specifically, we first allocate a queue for each type of event for each flow, simplifying the logic without considering cross-flow and cross-type dependencies. We adopt the single-producer and single-consumer model for these queues to ensure lock-free operation. These queues are allocated using the memory resources accessible to the DPA process. Unlike hardware coalescing, which is limited to the *Accumulation* strategy, software coalescing offers the flexibility to implement additional strategies through software modifications. In addition to basic strategies such as *Accumulation*, *Keep-Latest-Event*, and *Keep-Oldest-Event*, we support more complex strategies. For instance, when flow rates stabilize (*e.g.,* converge after a slow start), continuing to process certain fabric events like TX could be wasteful for processor cycles. Instead, we coalesce multiple TX events into one processing event, saving unnecessary processing cycles. Specifically, when the hardware multiplexer hands over TX events to the DPA software processing thread, we early terminate processing and enqueue events. Full processing executes only if we accumulate enough signals in the event queue. Such complex strategies are not supported by current hardware coalescing.

Software Multiplexing: The multiplexer is responsible for dequeuing events and invoking the DPA thread to process. Ideally, we aim to consolidate all functions—such as enqueuing, coalescing, dequeuing, and invoking—onto the DPA cores to free processing cycles of the host CPU or DPU ARM. However, to our knowledge, DPA operates a specialized real-time OS (RTOS) with limited capabilities for thread and process management, leading to two main issues:

1) The DPA process cannot create threads during runtime (*e.g.,* utilizing POSIX thread system calls). Consequently, we rely on the host CPU or DPU ARM to pre-launch a process with a pool of threads (we discuss the pool size parameter in §4.3.3). Within this pool, specific threads are dedicated for collecting events (*e.g.,* sending/receiving out-of-band packets), while others are dedicated to processing. Event queues are stored in process-global memory accessible to both collectors for enqueuing and processor threads for dequeuing.

2) DPA's inter-process communication (IPC) capabilities are limited. The current BF3 DPA allows only a specialized process (*i.e.,* doca_pcc) to set flow rates. Therefore, other processes handling host-domain and peer-domain events need to communicate with the doca_pcc process to adjust flow rates. However, the DPA lacks typical IPC mechanisms like Linux pipes or domain sockets. Instead, we use the host CPU or DPU ARM as an agent, facilitating memory sharing with processes handling host-domain and peer-domain events. Furthermore, the host CPU (or DPU ARM) utilizes a Mailbox Mechanism

[68] to activate doca_pcc for setting the flow rate. Importantly, most functionalities still operate on the DPA, with the host CPU merely handling lightweight operations to compensate for the current limitations of the DPA RTOS.

4.3.3 Event Processor Subsystem

Design Goals: The event processor subsystem is to execute user-defined control laws. The primary goal for the event processor is computational efficiency, given that the per-packet signal interval could be as short as a sub-μs. Inspired by classical OS and architecture techniques [76–78], we aim at building a *Deterministic Multi-Threading* (DMT) model. Next, we propose several mechanisms to realize such a model on a multi-core processor of the BF3 DPA.

Run-to-Completion and Shared-Nothing: To ensure deterministic execution, computation adheres to the run-to-completion manner without context switching or preemptive scheduling. Given the packet-granular signals at the sub-μs level, context switching becomes prohibitively expensive. By default, we handle event processing fairly for all event types without prioritization. Thus, when the multiplexer schedules an event, it waits until the target thread is idle before invoking it rather than preempting. Furthermore, processors operate based on the shared-nothing principle, avoiding inter-thread locking or synchronization. Specifically, the multiplexer maps each event to only one thread and assigns the same type of event from the same flow to the same thread. Each thread holds its partition of global stateful records, ensuring the thread remains independent of others.

Scalable Thread Pool: We allocate hardware threads into separate pools corresponding to different purposes, such as handling fabric events or peer-domain events. Under the run-to-completion model , where threads are not dynamically created during runtime, the pre-determined size of the thread pool is a critical parameter. Considering scalability, we carefully determine the appropriate thread pool sizes for two primary types of event processing:

1) For per-node processing, the number of processing threads does not need to scale with the number of QPs (flows). Some types of signals, such as those from host domains, can be processed per node without the need for per-flow processing. As described in §4.3.1, host domain signals like PCIe utilization are shareable to all QPs over the same RNIC, as they share the same PCIe interconnect. Consequently, there is no need to establish per-QP threads to process these signals. Instead, we configure a limited number of threads (generally fewer than 10) dedicated to each host signal such as PCIe Utilization, Application Hint, and RNIC utilization;

2) For per-flow processing, it intuitively seems that the number of processing threads should scale with the number of flows. However, we introduce an observation indicating that such scaling is *not* always necessary for our thread steering policy as follows:

Steering Policy: For signals requiring per-flow processing

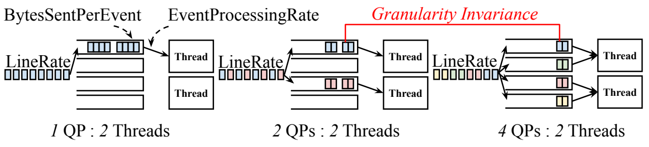


Figure 7: **Example of the Granularity Invariance Principle. While fixing two threads, the coalescence granularity remains invariant while increasing the number of flows from two to four.**

such as in-band fabric signals TX/ACK, out-of-band fabric signals RTT, and peer signals, we pre-determine the thread pool size before launching the DPA process. For a small number of QPs (*e.g.,* fewer than thread pool size), the subscription ratio of threads to QPs is fixed (*e.g.,* 1:1). For a large number of QPs exceeding the thread pool size, signal processing for multiple QPs is consolidated into a single thread.

Processing Fidelity: Surprisingly, consolidating signal processing for multiple QPs does not affect processing fidelity. To measure fidelity, we introduce the metric *coalescing granularity*, defined as the number of bytes sent out per event. Recalling §4.3.2, when the event queue builds up, multiple per-packet signals are coalesced into a single event. Consequently, a single event representing more bytes indicates more events awaiting processing. Coalescing granularity, therefore, serves as an effective metric for assessing the waiting time due to contention faced by the processing thread. A smaller granularity indicates finer control and improved fidelity. Next, we introduce a theoretical principle suggesting that consolidating the processing of multiple QPs into a single thread will not compromise fidelity:

Granularity Invariance Principle: For N flows saturating a fixed line rate ($LineRate$), when N exceeds the number of processing threads T, the coalescing granularity for each flow remains *invariant* even though increasing the number of flows (Figure 7 illustrates an example).

Proof: The principle holds because the line rate is invariant to the number of flows. Consider a processing system capable of handling packet-granular events without dropping events for any sent bytes. Let $BytesSentPerEvent_i$ be the number of bytes each flow i sends per event, and $EventRate_i$ the rate at which events are processed. Assuming a fixed number of processing threads T, uniform coalescing strategy for all flows ($BytesSentPerEvent_i = Bspe$ for all i), and equal sharing of processing capability among the flows:

$$\begin{aligned} LineRate &= \sum_{i=1}^{N} BytesSentPerEvent_i \times EventRate_i \\ &= Bspe \times \sum_{i=1}^{N} EventRate_i \end{aligned}$$

Since $\sum_{i=1}^{N} EventRate_i$ is a constant C (total event processing capacity shared among flows), we have: $LineRate = Bspe \times C$. Given C and $LineRate$ are invariant with respect to N (fixed T), $Bspe$ remains constant even as N increases. Hence, the coalescing granularity for each flow does not change.

The *Granularity Invariance Principle* provides scalability guarantees for per-flow processing utilizing the fixed number of threads, which applies to in-band fabric signals, out-of-band fabric signals, and peer signals. To achieve a reasonable coalescing granularity (*Bspe*), we typically launch 64 threads for processing in-band fabric signals. We empirically demonstrate its effectiveness and scalability in §6.1. For out-of-band fabric signals and peer signals, each flow's sender and receiver transmit out-of-band signals periodically (*e.g.,* per RTT), by sampling (*e.g.,* per BDP), or respond to anomaly triggers as discussed in §4.3.1. Compared with per-packet in-band signals, out-of-band signals are relatively coarse-grained, indicating more bytes sent per event. Thus, a smaller thread pool (16 in our setting) is sufficient for processing.

Strike Time-Space Trade-off: Improving both time/space efficiency is essential for DPA, which features a low-profile design with relatively wimpy cores and memory subsystems. Facing the fundamental time-space tradeoff, we employ distinct strategies for memory-intensive stateful operations and computation-intensive stateless operations. Details follow:

For stateful operations, we prioritize space efficiency by improving cache locality. Transport control algorithms often rely on stateful operations like membership queries, key-value queries, and counting. Stateful memory typically scales with the number of flows or the value range of flow information (*e.g.,* flow ID), but DPA memory resources are limited. To optimize stateful memory usage, we offer a computation library powered by compact data structures and algorithms. For membership queries, such as identifying the first packet of a flow, we utilize BloomFilter [79]. For key-value queries, such as recording flow statistics, we provide a HashMap. For counting tasks, we employ HyperLogLog [80] to estimate the cardinality and use Count-Min sketch [81] to estimate cumulative values.

For stateless operations, we prioritize time efficiency by lowering the instruction count. In addition to memory access costs, the instruction count directly translates into the execution time. For example, to improve time efficiency, we opt for fixed-point numbers (FXP-16) instead of floating-point numbers for arithmetic operations. Additionally, we employ bitwise operations for multiplication, square root calculations, and other arithmetic computations.

5 Spectrum of New Customizations

In this section, we demonstrate how to enable new functionality over SCR. To comprehensively cover a wide range of innovations, we categorize them into various system domains, as illustrated in Figure 8. For each domain, we explore candidate policies. Considering the rich customizations (Appendix Table 6), one policy never fits all, even for a single domain like congestion control. Thus, these policies showcase SCR's expressiveness (without claiming to be the best) and stimulate other innovations for diverse workloads and deployments.

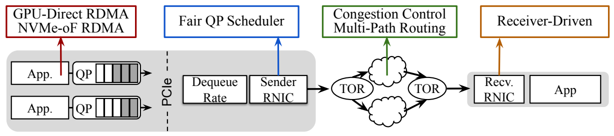


Figure 8: **New Customizations from Various Domains.**

5.1 Host Domain: Fair QP Scheduler

In this domain, we investigate the case of customizing the RNIC QP Scheduler to improve fairness and performance isolation across multiple tenants. Several prior works [17, 18, 24, 28, 73] investigate the importance of addressing multi-tenancy issues of QP schedulers.

Customized Policy: We specify one simple yet practical policy: Achieve fair bandwidth allocation for QPs with varying message sizes. SCR can support other policies, such as different performance metrics and different definitions of fairness. Here, we take a typical bandwidth fairness example. Before delving into the mechanisms, we first explain why the legacy RNIC is not fair for some scenarios. As described in §4.2, the QP scheduler dequeues WQEs from tenants' QPs, prompting RNIC to initiate DMA transfers to fetch messages across the PCIe. Consequently, the QP scheduler's decisions can significantly impact host-to-RNIC PCIe resources and RNIC hardware utilization. By default, RNIC adopts a greedy approach, prioritizing large message sizes to achieve high utilization, a phenomenon widely observed [18,24,56]. However, this greedy strategy may result in unfair resource allocation in multi-tenant environments with varying message sizes, potentially violating cloud providers' bandwidth commitments.

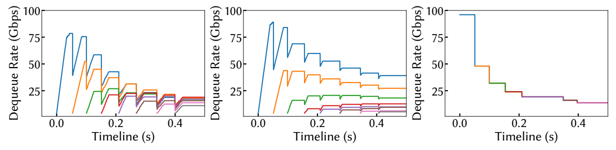
Control Laws: Firstly, we need to measure the real-time bandwidth for each flow, which is a metric to evaluate fairness. We utilize the TX events from SCR event colletor, which provide per-QP flow ID, event timestamps, and the amount of bytes sent out. For each flow, SCR event processors record the timestamp of the last observed TX event, calculate the time duration since then until the current event, and derive the real-time bandwidth utilization as the bytes sent out divided by the time duration. Note that such bandwidth calculation applies to events coalesced using the accumulation strategy. We utilize Exponentially Weighted Moving Average (EWMA) to smooth the real-time bandwidth. To enhance efficiency, BloomFilter performs the new flow identification, flow records are stored using HashMap, and floating-point calculations are approximated using fixed-point arithmetic.

Next, we present three algorithms to achieve fairness. Note that it is desirable to maintain high utilization while achieving fairness. SCR event processors support user-customized algorithms, here demonstrating three typical cases: 1) Default RNIC QP scheduler mechanism; 2) Static Allocation Algorithm: We statically control the dequeue rate for each flow as $\frac{Link\_Capacity}{Num\_Of\_Flow}$. 3) Water Filling (Progressive-Filling) Algorithm: Water Filling is a classical strategy to achieve max-min fairness in bandwidth allocation problems [82, 83]. Specifically, in our dequeue rate control model, we initially set the dequeue rates at starting values (discussed later) and

Case Studies	Packet-Granular Signals/Events	Benefits and Performance Gains
Fair QP Scheduler	TX Events	Improving Fairness with Varying Message Sizes; Isolating Performance between Lat./BW. Tenants
Congestion Control (§A.6)	Fabric RTT Events; Host/Peer Endpoint Latency	Enable RTT-based CC in addition to ECN-based CC
Multipath Routing (§A.7)	Fabric-Domain RTT Probing	Enable Sender-Side Multi-Path Monitoring and Selection
Receiver-Drivern Control (§A.8)	Peer Credits; Peer Congestion Notifications	Handle Incast; Enable NIC-Initiated ECN-like Backpressure
ML and Storage (§A.9)	The above signals are applicable.	Enhancing Fairness with Zero-Intrusion for GPU-Direct RDMA and NVMe-oF RDMA Applications and Libraries

Table 4: **Summary of Case Studies on Utilized Packet-Granular Signals and Achieved Benefits.**

then progressively increase rates. If a tenant QP can saturate the dequeue rate (*i.e.,* $BW\_EWMA \approx DequeueRate$) and the link capacity permits, we continue increasing it. However, if the tenant QP does not fully utilize the dequeue rate (*e.g.,* for non-bandwidth-hungry QPs), we stop further increases. Additionally, if a tenant's bandwidth degrades (*i.e.,* $BW\_EWMA \ll DequeueRate$), we decrease the dequeue rate and reclaim the credits. Event processors based on the DMT model ensure the rapid completion of the filling process and simultaneous progress among multiple flows.



(a) Scale down to half; Start from 0 (b) Scale down to $\frac{n-1}{n}$; Start from 0 (c) Scale down to $\frac{n-1}{n}$; Start from $\frac{1}{n}$

Figure 9: **Various Water Filling Strategies for Handling Flows Dynamicity (8 Bandwidth-Hungry QPs).**

To handle the dynamics of flows coming and going, we adjust the bandwidth allocations as follows: when a new flow joins, we scale the allocations for each existing flow i down to $\frac{n-1}{n} \times DequeueRate_i$ and start the new flow at $\frac{1}{n} \times LineRate$ (where n is the number of flows); when a flow terminates, we reclaim its allocated $DequeueRate$, making it available for other flows. In Figure 9, we visualize three Water Filling strategies with time-series dequeue rates for 8 flows, initiated one by one at 50 ms intervals (More experiment setup in §A.4). Our strategy (Figure 9c) demonstrates efficiency in both fairness and convergence time. Water Filling offers a broad design space, including strategies for handling the flows dynamicity and several tunable parameters, such as step size. In Appendix §A.4, we present more traces with varying parameters. In this work, we focus on the expressiveness of SCR (without claiming algorithms to be the best) and defer the algorithm optimality to future research.

5.2 More Case Studies

Table 4 summarizes key enablers and performance gains for more case studies. Due to space constraints, we provide their descriptions and experiments in the appendix (§A.6, §A.7, §A.8, and §A.9). Given the rich literature on RDMA, we provide a feasibility analysis for other works in §A.13.

6 Evaluation

Implementation: We implement SCR over the DPA in over 7.8K lines of C/C++ code. We describe specific SDK dependencies in A.10. We describe our testbed in §3.2.

6.1 Micro-Benchmarking SCR

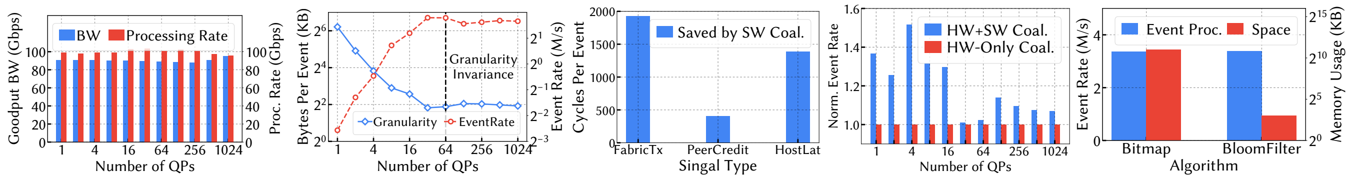
Setup: In Figure 10, we micro-benchmark the SCR framework using Perftest *ib_write_bw* [84] to generate traffic. We vary the number of QPs (flows) up to 1024 per node. The message size is set to 64KB with a queue depth of 4, allowing full 100Gbps line rate utilization. We launch up to 64 DPA threads. We primarily use the following metrics:

- *Event Rate:* Number of events handled (millions/second);
- *Coalescing Granularity:* Bytes sent per event;
- *Processing Rate:* Event Rate × Coalescing Granularity.

Processing Capacity: In Figure 10a, we demonstrate that the processing rate of the SCR can match the line rate traffic. In this experiment, SCR's processing task is to enforce fairness across multiple tenants. We utilize up to 1024 concurrent QPs and dump out the processing rate for the fabric TX signal, which is the most frequent signal among all signal types from all domains. The figure illustrates that SCR can handle these signals even during line-rate traffic. The processing rate exceeds the application goodput, as the bytes indicated by the TX event include both transport headers and payloads.

Scalability: In Figure 10b, we investigate the scalability using the same setup as the previous test. As we increase the number of QPs from 1 to 64, each flow exclusively utilizes one thread for event processing as designed in §4.3.3. With varying the number of flows and corresponding threads from 1 to 64, the line rate remains saturated and unchanged, reducing the processing load on individual threads. Thus, as the number of QPs increases to 64, we observe that the SCR can process events with the finest granularity and peak event rate. Beyond 64 QPs, we do not increase the thread count; instead, multiple flows are consolidated into a single thread. Excitingly, as analyzed by the *Granularity Invariance Principle* in §4.3.3, the granularity remains invariant even with up to 1024 QPs. This indicates that the SCR can handle more flows without additional threads, maintaining the same processing fidelity.

Software Coalescing Applicability: In Figure 10c, we show the applicability of software coalescing. Unlike hardware coalescing, which is limited to in-band signals from the fabric domain, software coalescing in SCR extends to signals across all domains. In this test, we apply software coalescing to fabric TX signals, peer credit signals, and host PCIe latency signals, showing the reduction in processing cycles per event. We use cycles as the metric—rather than execution times—because the DPA lacks a sub-μs timer (for reference, the DPA system clock runs at 505 MHz). Also, the figure highlights the flexibility of user-defined coalescing strategies. For TX events, we implement coalescing based on the cover-



(a) Proc. Matches LineRate (b) Granularity Invariance (c) SW Coal. Applicability (d) SW Coal. Efficiency (e) Space Efficiency

Figure 10: **Micro-Benchmarking Key Properties of the SCR Framework.**

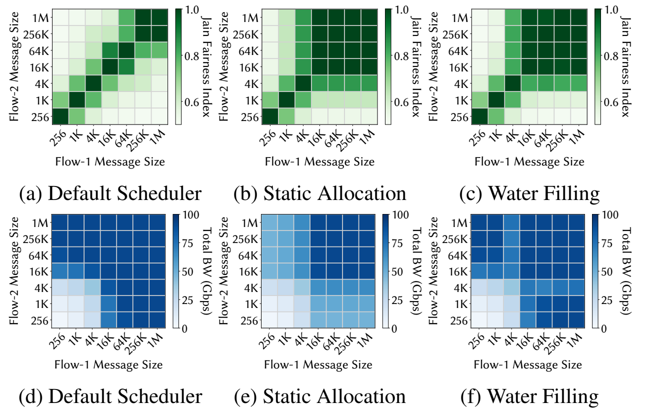
age status of flow rates (detailed in §4.3.2). For peer and host signals, we employ the *Keep-Latest-Event* strategy.

Software Coalescing Efficiency: In Figure 10d, we present the enhanced efficiency of software coalescing over typical hardware-only coalescing. As in previous experiments, we report the event rate for TX events with multi-tenant isolation processing task. To demonstrate enhanced efficiency, we normalized the combined hardware and software (HW+SW) coalescing against hardware-only (HW-Only) coalescing. The results show that software coalescing improves efficiency, particularly when the number of threads is limited, thereby increasing the load per thread. For instance, with only four threads, the HW+SW coalescing achieves a 51.3% higher event rate compared to the HW-Only coalescing.

Space Efficiency: In Figure 10e, we show the space efficiency for stateful operations, using the example of identifying the first signal of new flows—an essential operation for initiating processing. The Precise Bitmap Algorithm pre-allocates one bit for each potential flow ID. Upon signal arrival, it indexes the bit according to the flow ID to verify existence. With Flow IDs generated by BF3 hardware being 24 bits long, the Bitmap requires about 2 MB of memory (2^{24} bits). The Bloom Filter is an approximate algorithm for membership queries, with memory usage scaling only with the number of flows, not the value range. To achieve a false positive rate of 0.03 with three hash functions for up to 8192 flows, BloomFilter requires just 8 bits per flow (totaling 8KB). The Bloom Filter's space requirement is smaller than that of the Bitmap algorithm. Additionally, both methods can achieve similar event processing rates as they execute O(1) operations.

6.2 Achieving Fairness with the QP Scheduler

In Figure 11, we report the bandwidth as the utilization metric and Jain's Fairness Index [85] as the fairness metric. We present the evaluation results for the three aforementioned algorithms (§5.1). We set up two competing tenants using Perftest [84] WRITE verbs and vary the message size. The total link capacity is 100 Gbps. Figure 11a and 11d depict the performance of the default scheduler. It can only ensure fair allocation when both tenants use the same message size, as indicated by the diagonal line in Figure 11a. With the default scheduler, large messages can achieve high utilization, as shown in Figure 11d. Notably, for small messages (*e.g.,* 256 B), which are not bandwidth-hungry, they cannot saturate the link capacity. Figure 11b illustrates the fairness results when using the static allocation algorithm, which is the fairest

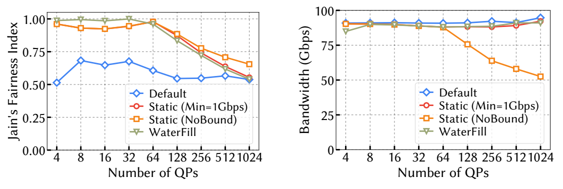


(a) Default Scheduler (b) Static Allocation (c) Water Filling
(d) Default Scheduler (e) Static Allocation (f) Water Filling

Figure 11: **Enhancing Fairness and Utilization in QP Scheduling. Deeper Color is Better for Both Metrics.**

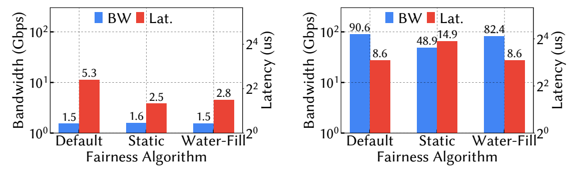
as it allocates exactly $\frac{1}{n}$ bandwidth to each tenant. However, static allocation can lead to bandwidth over-provisioning for smaller messages and starvation for larger messages. For instance, while bandwidth-hungry tenants with large message sizes (*e.g.,* 1 MB) can starve due to the 50 Gbps budget, small messages like 256B fail to fully utilize 50 Gbps. As a result, overall utilization suffers. The Water Filling algorithm strikes a balance between fairness and utilization by progressively filling the bandwidth demands for all tenants. As depicted in Figures 11c and 11f, the Water Filling algorithm can achieve comparable fairness to static allocation while enhancing utilization. Unlike static allocation, the Water Filling prevents over-provisioning for small messages and starvation for large messages. In Figure 11, the Water Filling algorithm improves the fairness index of up to 1.78x compared to the default scheduler (*e.g.,* 4KB-msg. QP *vs.* 1MB-msg. QP).

In Figure 12, we scale the number of QPs to evaluate fairness algorithms. Each server manages up to 1024 QPs, with each QP's message size uniformly sampled from the range [4KB, 1MB], resulting in varied message sizes across QPs. In this experiment, we use WRITE operations, while experiments for READ are in Appendix §A.5. The queue depth is 4 for all QPs. In Figure 12a, the Static and Water Filling allocations can improve fairness by up to 1.88x than the default scheduler (for #QPs=4, increasing from 0.51 to 0.96). Figure 12b shows that all three algorithms can almost saturate the line rate. Notably, when the number of flows exceeds 100, each flow is allocated less than 1 Gbps. Achieving accurate hardware rate limiting at the granularity of Mbps for RNICs is still challenging [86, 87]. In Figure 12b (Static-NoBound),



(a) Fairness Index (b) Bandwidth Utilization

Figure 12: **Fairness Across Multiple Tenants (WRITE).**



(a) Latency-Sensitive QP (b) Bandwidth-Hungry QP

Figure 13: **Performance Isolation (Lat QP *vs.* BW QP).**

setting the flow rate below 1 Gbps reduces total utilization and the inability to saturate the line rate, likely due to excessive arbitration within the rate limiter. Thus, to avoid low utilization, we set a minimum bandwidth bound of 1 Gbps per flow for both Static and Water Filling approaches. Setting the bandwidth roughly at 1 Gbps rather than a more precise Mbps level explains the decline in the fairness metric when the number of flows exceeds 100. Beyond 100 flows, the significance of achieving fairness becomes minimal since each flow is allocated only a small amount of bandwidth, which is fairly similar across flows.

6.3 Enhancing Tenants Isolation with the QP Scheduler

In Figure 13, we illustrate enhancing fairness and isolation can decrease latency for latency-sensitive tenants when competing with bandwidth-hungry tenants. We use Perftest WRITE [84] to set up two competing tenants: one bandwidth-hungry, utilizing large-message sizes (64 KB) and large batch sizes (128), and the other latency-sensitive, using small-message sizes (256B) and small batch sizes (4). Figure 13a displays the latency results for the latency-sensitive QP, while Figure 13b shows the bandwidth for the bandwidth-hungry QP. Under the default scheduler, the latency-sensitive tenant experiences significant latency spikes. With static allocation, where the bandwidth-hungry QP is limited to half the line rate, interference with the latency-sensitive tenant is reduced, but throughput suffers. Water Filling algorithms maintain high utilization while simultaneously reducing latency. In Figure 13, the Water Filling algorithm reduces latency by up to 52.8% compared to the default one while achieving high utilization. For the latency-sensitive tenant, all three algorithms deliver the same bandwidth. For the bandwidth-sensitive tenant, Water-Filling matches the default scheduler's latency. However, the Static algorithm increases latency due to restricting the bandwidth-hungry QP to only half the line rate.

Moreover, as depicted in Figure 13, we can draw a straightforward yet significant observation: SCR enables customization without compromising (even improving!) RDMA data plane performance, particularly critical latency metrics. This observation holds true across all case studies.

6.4 Discussion

More Cases and Guidelines: As referenced in Table 4, more cases are detailed in the appendix. We report some developer guidelines in A.11 and suggestions for vendors in A.12.

Relationship to DOCA PCC: We reference PCC [68] for APIs of setting rates for QPs and obtaining fabric-domain signals. In SCR, we propose the Dequeue Rate Control Model (§4.2 and §4.3.1), Software Coalescing and Multiplexing to address DPA RTOS limitations (§4.3.2), and strategies to efficiently utilize DPA (§4.3.3). We anticipate that these contributions will also benefit PCC.

Limitations of SCR: In §4.3.1, we describe the lack of RX signals. In §4.3.2, we discuss the inability to create threads during runtime and the IPC limitation. In Figure 12, achieving precise rate limiting at the Mbps granularity remains a challenge. In §A.7, we describe the limitation on how to specify routing paths. In Appendix A.11 and A.12, we report some latency issues such as setting rates. We expect future software and hardware advancements to address these issues.

Generality: Besides the BF3 DPA, we explore generalizing the SCR design to other platforms such as hardware devices for Google Falcon [4] and Broadcom's Inband Flow Analyzer (IFA) [88, 89]. Google Falcon, according to the spec [4], features a Rate Update Engine (RUE), which occupies a similar architectural position as the BF3 DPA. Thus, our dequeue rate control and coalescing/multiplexing strategies could be adapted for use with the RUE. Similarly, according to Broadcom IFA's draft, the Initiating Function Node can potentially fulfill a role akin to the BF3 DPA in SCR.

Resource Costs: We discuss computational and network resources costs of SCR. In practice with BF3, we use minimal computational resources from the host CPU or DPU ARM, typically for lightweight tasks such as launching DPA threads or responding to host probes; a single CPU core suffices in our experiments. Moreover, DPA, based on RISC-V, is designed for efficiency in power usage, chip area, and monetary cost [60, 90]. Besides BF3 DPU, even the low-power version of the BF3 (SuperNIC [91]) incorporates DPA. In terms of network resources, both out-of-band probing and in-band telemetry consume bandwidth. It is a common challenge [9, 92, 93].

7 Conclusion

In this work, we introduce Software-Controlled RDMA (SCR) to realize the vision of white-boxing RDMA. SCR applies the dequeue rate control model and facilitates an event-driven rate computation framework. We enable a spectrum of new functionalities not present in legacy transport. We envision SCR to help land past and future RDMA transport innovations.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd, Masoud Moshref Javadi. 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-2106199, CNS-2212192, CNS-2339755, and Cisco Research Grant.

References

[1] 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.

[2] 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.

[3] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery.

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

[5] 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.

[6] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 57–70, New York, NY, USA, 2024. Association for Computing Machinery.

[7] 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.

[8] Ultra Ethernet Consortium. https://ultraethernet.org/, 2024.

[9] 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.

[10] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. 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 514–528, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Yiran Zhang, Qingkai Meng, Chaolei Hu, and Fengyuan Ren. Revisiting congestion control for lossless ethernet. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 131–148, Santa Clara, CA, April 2024. USENIX Association.

[12] Yiran Zhang, Yifan Liu, Qingkai Meng, and Fengyuan Ren. Congestion detection in lossless networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 370–383, New York, NY, USA, 2021. Association for Computing Machinery.

[13] 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.

[14] Prateesh Goyal, Preey Shah, Naveen Kr Sharma, Mohammad Alizadeh, and Thomas E Anderson. Backpressure flow control. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1–3, 2019.

[15] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: a receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 221–235, New York, NY, USA, 2018. Association for Computing Machinery.

[16] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 761–777, 2022.

[17] 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.

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

[19] 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.

[20] 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, 2018.

[21] Sugi Lee, Yusung Kim, Honguk Woo, and Ikjun Yeom. Efficient user-level multi-path utilization in rdma networks. *IEEE Access*, 9:127619–127629, 2021.

[22] 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.

[23] Hao Wang, Han Tian, Jingrong Chen, Xinchen Wan, Jiacheng Xia, Gaoxiong Zeng, Wei Bai, Junchen Jiang, Yong Wang, and Kai Chen. Towards Domain-Specific network transport for distributed DNN training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, April 2024. USENIX Association.

[24] 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.

[25] 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.

[26] 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.

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

[28] Yanfang Le, Brent Stephens, Arjun Singhvi, Aditya Akella, and Michael M Swift. Rogue: Rdma over generic unconverged ethernet. In *Proceedings of the ACM symposium on cloud computing*, pages 225–236, 2018.

[29] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, page 1–12, New York, NY, USA, 2007. Association for Computing Machinery.

[30] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[31] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 15–28, New York, NY, USA, 2016. Association for Computing Machinery.

[32] Naveen Kr Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable calendar queues for high-speed packet scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 685–699, 2020.

[33] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[34] Everything You Should Know About White Box Switch. https://cloudswit.ch/blogs/what-is-white-box-switch/#what-is-white-box-switch, 2021.

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

[36] Broadcom Ethernet Network Adapters. https://www.broadcom.com/products/ethernet-connectivity/network-adapters, 2024.

[37] AMD Pensando. https://www.amd.com/en/accelerators/pensandoe, 2023.

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

[39] 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.

[40] Next-Generation Networking for the Next Wave of AI White Paper. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/next-generation-netw, 2024.

[41] Chris V Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1726–1734. IEEE, 2001.

[42] Steven H Low, Fernando Paganini, Jiantao Wang, Sachin Adlakha, and John C Doyle. Dynamics of tcp/red and a scalable control. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 239–248. IEEE, 2002.

[43] Laurent Massoulie. Stability of distributed congestion control with heterogeneous feedback delays. *IEEE Transactions on Automatic Control*, 47(6):895–902, 2002.

[44] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. Plb: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 207–218, New York, NY, USA, 2022. Association for Computing Machinery.

[45] 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.

[46] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, Santa Clara, CA, April 2024. USENIX Association.

[47] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 537–550, New York, NY, USA, 2015. Association for Computing Machinery.

[48] 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.

[49] Teng Ma, Tao Ma, Zhuo Song, Jingxuan Li, Huaixin Chang, Kang Chen, Hai Jiang, and Yongwei Wu. X-rdma: Effective rdma middleware in large-scale production environments. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–12. IEEE, 2019.

[50] NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl, 2024.

[51] Jiao Zhang, Jiaming Shi, Xiaolong Zhong, Zirui Wan, Yu Tian, Tian Pan, and Tao Huang. Receiver-driven rdma congestion control by differentiating congestion types in datacenter networks. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2021.

[52] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 29–42, New York, NY, USA, 2017. Association for Computing Machinery.

[53] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. phost: distributed near-optimal datacenter transport over commodity network fabric. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery.

[54] NVIDIA Spectrum-X Ethernet switch. https://www.nvidia.com/en-us/networking/spectrumx/, 2024.

[55] Broadcom Tomahawk Ethernet Switches. https://www.broadcom.com/products/ethernet-connectivity/switchin, 2024.

[56] Yunkun Liao, Jingya Wu, Wenyan Lu, Xiaowei Li, and Guihai Yan. Optimize the tx architecture of rdma nic for performance isolation in the cloud environment. In *Proceedings of the Great Lakes Symposium on VLSI 2023*, pages 29–35, 2023.

[57] 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.

[58] Xizheng Wang, Shuai Wang, and Dan Li. srdma: A general and low-overhead scheduler for rdma. In *Proceedings of the 7th Asia-Pacific Workshop on Networking*, pages 21–27, 2023.

[59] Soft-RoCE. https://enterprise-support.nvidia.com/s/article/howto-configure-soft-roce, 2024.

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

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

[62] Intel Infrastructure Processing Unit. https://www.intel.com/content/www/us/en/products/details/network-io/ipu/e2000-asic.html, 2023.

[63] 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.

[64] Benjamin Michalowicz, Kaushik Kandadi Suresh, Hari Subramoni, Dhabaleswar Panda, and Steve Poole. Dpu-bench: A microbenchmark suite to measure offload efficiency of smartnics. In *Practice and Experience in Advanced Research Computing*, pages 94–101, 2023.

[65] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 318–333. Association for Computing Machinery, New York, NY, USA, 2019.

[66] 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.

[67] Matrix Multiplication. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html, 2024.

[68] NVIDIA DOCA Programmable Congestion Control (PCC) API. https://docs.nvidia.com/doca/sdk/doca+pcc/index.html, 2024.

[69] Larry W McVoy, Carl Staelin, et al. Lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pages 279–294. San Diego, CA, USA, 1996.

[70] Zilong Wang, Xinchen Wan, Chaoliang Zeng, and Kai Chen. Accurate and scalable rate limiter for rdma nics. In *Proceedings of the 7th Asia-Pacific Workshop on Networking*, pages 15–20, 2023.

[71] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 404–417, New York, NY, USA, 2017. Association for Computing Machinery.

[72] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. SENIC: Scalable NIC for End-Host rate limiting. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 475–488, 2014.

[73] Yiwen Zhang, Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G Shin. Performance isolation anomalies in rdma. In *Proceedings of the Workshop on Kernel-Bypass Networks*, pages 43–48, 2017.

[74] Intel Performance Counter Monitor (Intel PCM). https://github.com/intel/pcm, 2024.

[75] Daniele De Sensi, Tiziano De Matteis, Konstantin Taranov, Salvatore Di Girolamo, Tobias Rahn, and Torsten Hoefler. Noise in the clouds: Influence of network performance variability on application scalability. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–27, 2022.

[76] Tom Bergan, Owen Anderson, Joseph Devietti, Luis Ceze, and Dan Grossman. Coredet: A compiler and runtime system for deterministic multithreaded execution. In *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*, pages 53–64, 2010.

[77] Heming Cui, Jingyue Wu, John Gallagher, Huayang Guo, and Junfeng Yang. Efficient deterministic multithreading through schedule relaxation. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 337–351, New York, NY, USA, 2011. Association for Computing Machinery.

[78] Tom Bergan, Nicholas Hunt, Luis Ceze, and Steven D Gribble. Deterministic process groups in dOS. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[79] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[80] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, 2007.

[81] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[82] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, 2018.

[83] Pooria Namyar, Behnaz Arzani, Srikanth Kandula, Santiago Segarra, Daniel Crankshaw, Umesh Krishnaswamy, Ramesh Govindan, and Himanshu Raj. Solving Max-Min fair resource allocations quickly on large graphs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, April 2024. USENIX Association.

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

[85] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 21:1, 1984.

[86] 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.

[87] Vishal Shrivastav. Fast, scalable, and programmable packet scheduler in hardware. In *Proceedings of the ACM special interest group on data communication*, pages 367–379, 2019.

[88] Network Performance Anomaly detection with In-band Flow Analyzer (IFA). https://www.broadcom.com/blog/network-performance-anomaly-detection-with-in-band-flow-analyzer, 2024.

[89] Jai Kumar, Surendra Anubolu, John Lemon, Rajeev Manur, Hugh Holbrook, Anoop Ghanwani, Dezhong Cai, Heidi Ou, Yizhou Li, and Xiaojun Wang. Inband Flow Analyzer. Internet-Draft draft-kumar-ippm-ifa-08, Internet Engineering Task Force, April 2024. Work in Progress.

[90] Mikhail Khalilov, Salvatore Di Girolamo, Marcin Chrapek, Rami Nudelman, Gil Bloch, and Torsten Hoefler. Network-offloaded bandwidth-optimal broadcast and allgather for distributed ai, 2024.

[91] What Is a SuperNIC? https://blogs.nvidia.com/blog/what-is-a-supernic/, 2024.

[92] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[93] Shuai Wang, Kaihui Gao, Kun Qian, Dan Li, Rui Miao, Bo Li, Yu Zhou, Ennan Zhai, Chen Sun, Jiaqi Gao, Dai Zhang, Binzhang Fu, Frank Kelly, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. Predictable vfabric on informative data plane. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 615–632, New York, NY, USA, 2022. Association for Computing Machinery.

[94] NVIDIA ConnectX NICs. https://www.nvidia.com/en-us/networking/ethernet-adapters/, 2024.

[95] Intel Ethernet Network Adapters. https://www.intel.com/content/www/us/en/support/articles/000031905/ethernet-products/700-series-controllers-up-to-40gbe.html, 2024.

[96] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. Bolt: Sub-RTT congestion control for Ultra-Low latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 219–236, 2023.

[97] RDMA Core Userspace Libraries and Daemons. https://www.mankier.com/3/mlx5dv_modify_qp_udp_sport, 2024.

[98] 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.

[99] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, and Hongqiang Harry Liu. From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 753–766, New York, NY, USA, 2022. Association for Computing Machinery.

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

[101] https://spdk.io/. https://spdk.io/, 2024.

[102] RDMA Core Userspace Libraries and Daemons. https://developer.nvidia.com/nccl, 2024.

[103] NVIDIA DOCA FlexIO SDK Programming Guide. https://docs.nvidia.com/doca/archive/doca-v1.5.0/flexio-sdk-programming-guide/index.html, 2024.

[104] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. Restructuring endpoint congestion control. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 30–43, New York, NY, USA, 2018. Association for Computing Machinery.

[105] Srinivas Narayana. Making decisions at data plane speeds. *SIGMETRICS Perform. Eval. Rev.*, 51(2):88–90, oct 2023.

[106] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. Elasticity detection: a building block for internet congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 158–176, New York, NY, USA, 2022. Association for Computing Machinery.

[107] Mellanox Neo-Host. https://support.mellanox.com/s/login/?ec=302&startURL=%2Fs%2Fproductdetails%2Fa2v50000000N2OlAAK%2Fmellanox-neohost, 2024.

[108] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: a microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 399–413, New York, NY, USA, 2019. Association for Computing Machinery.

[109] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 519–533, 2021.

[110] Athinagoras Skiadopoulos, Zhiqiang Xie, Mark Zhao, Qizhe Cai, Saksham Agarwal, Jacob Adelmann, David Ahern, Carlo Contavalli, Michael Goldflam, Vitaly Mayatskikh, Raghu Raja, Daniel Walton, Rachit Agarwal, Shrijeet Mukherjee, and Christos Kozyrakis. High-throughput and flexible host networking via control and data path physical separation. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

[111] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. CASSINI: Network-Aware job scheduling in machine learning clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, April 2024. USENIX Association.

[112] Sudarsanan Rajasekaran, Manya Ghobadi, Gautam Kumar, and Aditya Akella. Congestion control in machine learning clusters. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 235–242, 2022.

[113] 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.

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

[115] Yuzhen Su, Jiao Zhang, Zirui Wan, Pingping Lin, Yunpeng Zhang, Tian Pan, and Tao Huang. Hermes: An efficient building block for rdma incast in datacenters. In *2023 9th International Conference on Computer and Communications (ICCC)*, pages 2306–2311. IEEE, 2023.

[116] Jung-Hwan Cha, Shinhyeok Kang, Yewon Kang, Hansaem Seo, Jungeun Lee, Jongsung Kim, and Minsung Jang. Corn: Cloud-optimized rdma networking. In *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 52–59, 2023.

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

[118] Wenxue Cheng, Kun Qian, Wanchun Jiang, Tong Zhang, and Fengyuan Ren. Re-architecting congestion management in lossless ethernet. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 19–36, 2020.

[119] Haonan Qiu, Xiaoliang Wang, Tianchen Jin, Zhuzhong Qian, Baoliu Ye, Bin Tang, Wenzhong Li, and Sanglu Lu. Toward effective and fair rdma resource sharing. In *Proceedings of the 2nd Asia-Pacific Workshop on Networking*, pages 8–14, 2018.

[120] Dian Shen, Junzhou Luo, Fang Dong, Xiaolin Guo, Kai Wang, and John CS Lui. Distributed and optimal rdma resource scheduling in shared data center networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 606–615. IEEE, 2020.

[121] Yanfang Le, Mojtaba Malekpourshahraki, Brent Stephens, Aditya Akella, and Michael M Swift. On the impact of cluster configuration on roce application design. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking*, pages 64–70, 2019.

[122] Huichen Dai, Binzhang Fu, and Kun Tan. PFC-Free Low Delay Control Protocol. Internet-draft, Internet Engineering Task Force, July 2020.

[123] 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.

A Appendix

A.1 Limitations of CPU/DPU-ARM in Meeting Control System Requirements

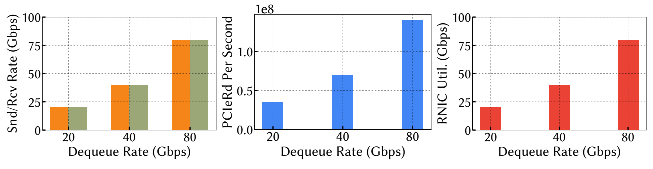
R1 Multi-threading Performance: For CPU and DPU ARM, while the host CPU has many cores, these cores should primarily be allocated to tenants for revenue-generating workloads. In contrast, the ARM CPU possesses only 16 cores without hyper-threading, partially satisfying the R1 requirement.

R2 Interaction with RNIC: For CPU and DPU ARM, several issues arise. Firstly, they currently lack APIs to acquire packet-granular signals, particularly in-band per-packet signals like ACK/CNP. Also, as far as we know, these cores lack APIs to enforce control actions, such as changing the per-flow rate during runtime. While the lack of APIs can be addressed in the short term, another long-term concern remains. As Figure 1a shows, DPA is co-packaged with the RNIC, with Network-on-Chip direct access to RNIC TX/RX pipes, resulting in lower access latency. However, host CPUs or ARM cores must cross the host PCIe or on-SoC PCIe to interact with RNICs. Furthermore, the host CPU and ARM cores access RNIC signals with the Performance Monitoring Unit (PMU) counters, which work in a sampling manner instead of timely monitoring.

R3 Raw Ethernet Capability: CPU and DPU ARM satisfy the requirements of the general network, including the in-kernel TCP/IP stack and RDMA transport. However, DPA is more tightly integrated with RNIC than CPU and ARM cores.

R4 Memory Access Latency: Note that it is trivial for the host CPU to monitor host-domain resource usage and record it with host memory, but this is not the case for the ARM CPU. Although the ARM CPU can access host memory using RDMA (μs scale), this incurs longer latency than the load/store accessing capabilities of the DPA (sub-μs scale).

A.2 Evaluation of Dequeue Rate's Impact Across Multiple Domains

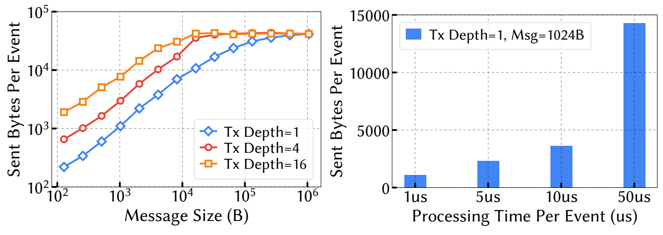


(a) Send/Receive BW (b) PCIe Utilization (c) RNIC Processing

Figure 14: **Dequeue Rate's Impact on RDMA Transport.**

In Figure 14, we explore the impact of dequeue rate on various domains: Fabric, Host, and Peer. Figure 14a illustrates how varying dequeue rates can regulate traffic sent into the fabric and received by peers. In Figure 14b, we monitor PCIe utilization using Intel PCM [74] tools across different dequeue rates. To understand how the dequeue rate influences RNIC hardware utilization, we track DMA jobs initiated by the RNIC to fetch messages from host memory. Specifically, we use Intel PCM to monitor the IIO (Integrated Input/Output) controller for In-Bound read activity, representing the number of bytes requested by RNIC for reading from main memory via DMA. Figure 14c demonstrates the effect of the dequeue rate on RNIC DMA job execution.

A.3 Micro Benchmark of Coalescing TX Events

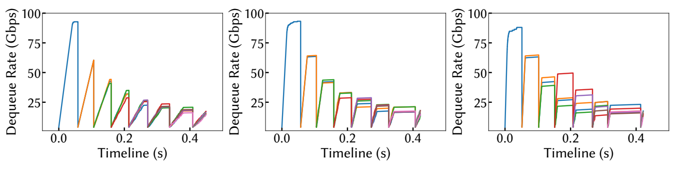


(a) Batching Triggers Coalescing (b) Slow-Proc. Triggers Coalescing

Figure 15: **Coalescing Experiments (TX Events Example).**

In Figure 15, we conduct a micro-benchmark on the coalescing of multiple TX events by accumulating the *sent_bytes* indicated by each TX event into a single sum value. Figure 15a demonstrates that larger message sizes per WQE and deeper SQ depths can trigger the coalescing of multiple TX events. It's worth noting the presence of a ceiling for *sent_bytes_per_event* due to the PCIe-related configuration on maximum batching size. Additionally, Figure 15b illustrates how slower event processing can also induce TX event coalescing. Overall, coalescing proves to be a promising mechanism for absorbing the high burstiness of packet-granular events.

A.4 Traces of Water Filling Procedure



(a) Step=0.02 GBps (b) Step=0.05 GBps (c) Step=0.1 GBps

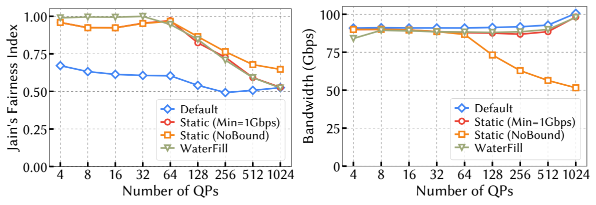
Figure 16: **Traces of Water Filling Procedure (#QPs=8). Scale down to 0; Start from 0.**

In Figure 16, we present time-series dequeue rates for eight flows during the Water Filling procedure. The minimum message QP utilizes 4KB, while subsequent QPs double this size successively, resulting in different message sizes per QP. All QPs have a queue depth of 4 and can saturate a 100 Gbps line rate individually. We initiate the dequeue rate at 0.5 Gbps, with step sizes varying from 0.03 Gbps, 0.05 Gbps to 0.1 Gbps. Flows are started one by one at 50 milliseconds intervals. To clearly demonstrate the impact of step size, we employ a strategy to handle flow dynamics: when a new flow joins, we simply reset all allocations and restart the Water Filling procedure. As indicated by the traces, smaller step sizes aid in converging to fairer rate allocations but require more time to achieve convergence. Table 5 shows the total time required for all flows to converge to a stable state when eight QPs begin filling concurrently. Note that to generate the trace for Figure 16, we need to output logs, which introduces some delay in the time series. For the measurements in Table 5, we disabled log printing to eliminate such delays.

Filling Step Size	Convergence Time
0.02 GBps	294 μs
0.05 GBps	122 μs
0.10 GBps	49 μs

Table 5: **Time Cost for All Flows to Converge to Stable Rates.**

A.5 Support RDMA READ Operation



(a) Fairness Index (b) Bandwidth Utilization

Figure 17: **Achieving Fairness Across Multiple Tenants (RDMA READ).**

In Figure 17, we demonstrate that SCR is applicable to various RDMA operations, including READ and WRITE. The experimental setup is identical to that shown in Figure 12, except that this experiment uses RDMA READ operations instead of the RDMA WRITE operations used previously. The results are similar: Static and Water-Filling allocations enhance fairness while maintaining high utilization and scalability, as analyzed in §6.2. SCR supports multiple RDMA operations, differing primarily in whether the requester's or responder's BF3 DPA takes effect. For WRITE operations, the Fair QP scheduler takes effect at the requester's BF3 DPA, as the requester sends out data and collects TX signals. Conversely, for READ operations, the Fair QP scheduler takes effect at the responder's BF3 DPA, since the responder sends out data and collects TX signals.

A.6 Fabric Domain: Congestion Control

In the fabric domain, we investigate congestion control (CC), a classical and everlasting problem. Most RNIC vendors [36, 94,95] baked DCQCN into the ASIC. With the introduction of

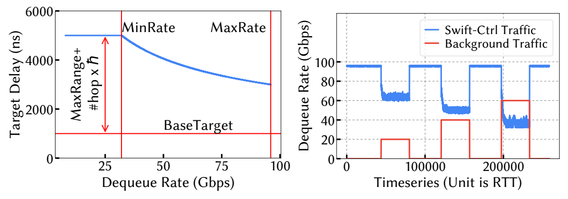
SCR, there is a promising opportunity for commercial RNICs to reap the benefits of other innovative CC [10, 11, 13, 47, 96].

Customized Policy: With ECN-based algorithms already demonstrated by DCQCN, we now explore implementing RTT-based CC using SCR. Delay-based CC is desired for several large-scale RDMA deployments [5, 46].

Swift Control Laws: We investigate Swift, a typical delay-based CC algorithm proposed by Google [10]. Swift's core logic is as follows: Additively increase the congestion window when RTT is less than the target delay; multiplicatively decrease the congestion window when RTT exceeds the target delay. With SCR, we examine RTT measurement and target delay computation, two building blocks for Swift's control.

Measuring RTT Delays: Swift control loops decompose the end-to-end delay into three components: Fabric Delay, Local NIC TX/RX Delay, and Remote NIC RX/TX Delay. Swift measures fabric delay by utilizing ACK packets to convey in-band timestamp information. As described in §4.3.1, SCR supports out-of-band per-flow RTT measurements, offering greater flexibility compared to in-band RTT measurements. This method allows for control over the frequency of sending RTT requests and eliminates the need to customize the ACK packet format. Pingmesh [92] shows such proactive RTT measurement is feasible in large-scale environments. For local NIC TX/RX delay indicating local host congestion, Swift originally utilizes timestamps from the local host and NIC to calculate per-packet delays, necessitating precise time synchronization between the NIC and host. Alternatively, SCR utilizes the host-domain signal of PCIe latency to reflect host congestion. For the remote NIC TX/RX delay, SCR utilizes signals from the peer domain, such as peer PCIe latency, to represent this delay.

Computing Target Delay: Swift encodes most of the complexity into the computation of the target delay, enabling it to scale to multiple competing flows and multiple-hop network typologies. The target delay calculations require floating-point and complex arithmetic operations like square root. To enhance efficiency, SCR employs fixed-point and bitwise operations. Given that Swift is a window-based algorithm and the current BF3 RNIC employs a rate-based mechanism, we utilize the Bandwidth-Delay Product (BDP) approach to translate the per-flow rate into the per-flow window.



(a) Target Delay Curve (b) Swift-Control Traffic

Figure 18: **Statistics Dump for Swift Congestion Control.**

Experiments of RTT-Based Congestion Control: As shown in Figure 18, we dump out the runtime statistics to verify the functionality of Swift. In Figure 18a, we present a trace demonstrating how the dequeue rate affects the target delay. According to our testbed, we configured Swift's parameters as follows: base target as 1 μs, #hop as 1, per-hop scaling factor as 2 μs, max scaling range as 2 μs, Min Rate as 32 Gbps, and Max Rate as 96 Gbps. We observe that our implementation accurately follows Swift's control law and reproduces the curve behavior as in the original design [10]. In Figure 18b, we generate background traffic varying from 20 Gbps, 40 Gbps to 60 Gbps and monitor the Swift-Controlled traffic. We can observe that SCR-enabled Swift can converge to the appropriate bandwidth to avoid fabric congestion.

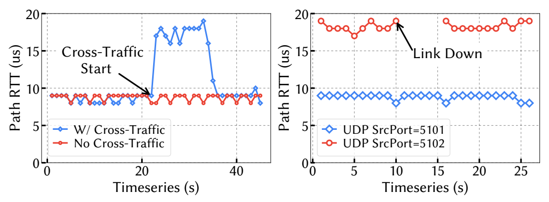
A.7 Fabric Domain: Multi-Path Routing Selection

We investigate the second case of the fabric domain, Multi-Path Routing, which is advocated by several industrial-level RDMA customizations such as Google PLB [44] in Falcon, AWS SRD [5], and UEC Transport [8].

Customized Policy: We monitor multi-path status and re-path according to congestion status and link failure at the sender side. We revisit one prior policy, MP-RDMA [20], which repurposes the *Src Port* field of the RoCEv2 transport to indicate the *Virtual Path ID*, enabling switch ECMP to execute multipathing accordingly. MP-RDMA relies on the in-band ACK clocking to detect congestion status for each *Virtual Path*. The end-host logic of MP-RDMA is prototyped via FPGAs, not commercial RNICs. Next, we examine the feasibility of multi-path routing using SCR.

Control Laws: SCR offers two key functionalities: Multipath Monitoring and Path Selection. To monitor the status of multiple paths, we employ out-of-band telemetry to probe status proactively. In SCR, the local collector pings the peer, which responds with a pong, allowing the local collector to obtain the RTT. By varying the header field of the request, we can enforce RTT requests through different paths in the fabric. For header field assignments, we adopt a random generation approach, similar to the practice used in MP-RDMA [20] and PLB [44]. After detecting the RTTs of multiple paths, we select the path with the lowest latency. For path selection enforcement, we use the source port field of RoCEv2 to indicate the path, as MP-RDMA advocates. Currently, it is feasible to specify the source port field of RoCEv2 for each QP using APIs like *mlx5dv_modify_qp_udp_sport()* [97]. However, this cannot be done at the granularity of individual packets. Thus, we rely on the BF3 hardware flow engines within the NIC-embedded switch to rewrite the packet header [60].

Experiments of Monitoring Multi-Path Status: As Figure 19 shows, we demonstrate two scenarios for monitoring multi-path RTT. In Figure 19a, we present the measurements that reveal the congestion status of network paths effectively. We introduce 100 Gbps background cross-traffic between the sender and receiver, observing an increase in RTT when cross-traffic begins and a decrease when it ceases. In Figure 19b, we



(a) Path RTT with Cross-Traffic (b) Multi-Path RTT

Figure 19: **Latency Monitoring Across Multiple Paths.**

present another scenario monitoring multiple path statuses by varying the source port to control routing paths. By manually adding a 10 μs latency for traffic with UDP SrcPort 5102, we compare it with traffic on UDP SrcPort 5101, revealing accurate latency representation for different paths. We also simulate a link failure scenario for the path of UDP SrcPort 5102. We configured a timeout value (100 μs) to detect the link failure, causing the loss of RTT probing packets. Figure 19b illustrates RTT probing's ability to detect link failures and recovery. This information aids in SCR selecting the optimal path for RDMA traffic, even at the per-packet level, via RoCEv2 header manipulation.

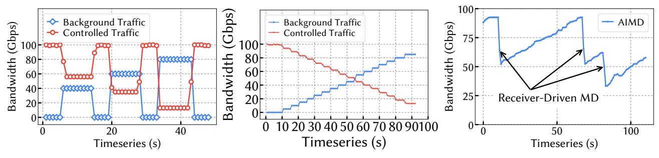
A.8 Peer Domain: Receiver-Driven Control Loop

In this domain, we investigate the receiver-driven control loop, which is not present for legacy RDMA transport. RDMA operations, especially one-sided READ/WRITE, are designed to be sender-driven, allowing transfers to bypass the involvement of the peer entirely. However, sender-side controls may struggle to handle traffic patterns that cause congestion at the receiver side, such as many-to-one incast. The receiver-driven paradigm can help handle the incast [15, 16, 52, 53].

Customized Policy: We refer to one typical receiver-driven protocol, Homa [15]. In Homa's control loop, the sender is only allowed to send bytes after receiving a *grant* from the receiver, enabling efficient handling of incast scenarios by allowing a few senders to transmit simultaneously. Next, we leverage SCR to transfer such receiver-initiated signals to benefit RDMA one-sided operations WRITE/READ.

Control Laws: SCR provides the peer-domain signals as described in §4.3.1. Peer-domain signals, transmitted out-of-band as packet payloads, offer flexibility for carrying information, distinct from modifying in-band packet headers. The in-band manner requires addressing size limitations and ensuring packet routability across the fabric and middlebox. To support Homa-like grant mechanisms, we utilize peer domain signals to convey the grant information. Originally, Homa utilized a window-based control mechanism where credits indicated the allowable byte counts. Given that the current BF3 supports only rate-based control, we have made two modifications: First, we use credits to represent the available bandwidth (in Gbps) for the sender. Second, the senders operate *optimistically*, continuing to transmit at the current rate unless new credits are received to adjust the traffic.

In addition to grant mechanisms, we also support a second receiver-driven mechanism where the receiver issues ECN-like backpressure notifications to the sender. Commodity RNICs currently lack mechanisms to mark ECN bits at the receiver NIC. This functionality is beneficial for related work [17, 98] because receivers can mark the ECN bits to backpressure the senders. While SCR doesn't directly mark ECN bits on in-band packets, we utilize out-of-band peer-domain signal packets to convey ECN-like information to backpressure senders.



(a) Recv.-Granted Credit (b) 16-to-1 Incast (c) Recv.-Init. ECN

Figure 20: **Receiver-Driven Control Mechanisms.**

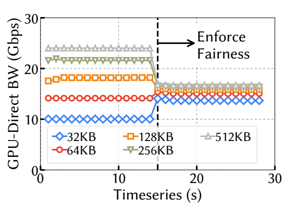
Experiments of Receiver-Driven Control: In Figure 20a, we illustrate the mechanism of receiver-granting credits to handle the 2-to-1 incast scenario. We use Perftest to configure two clients (one BF2 and one BF3) sending data to the same destination server (one BF3), with BF2-to-BF3 traffic acting as the background cross-traffic. The BF3-to-BF3 connection demonstrates the receiver-granting credits mechanism. Specifically, the receiver-side SCR monitors the real-time bandwidth utilization of the given port and calculates the credit as the remaining bandwidth headroom. These credits are then transferred to the sender as peer-domain signals, allowing the sender-side SCR to set the dequeue rate accordingly. From Figure 20a, we observe that the sender appropriately utilizes the remaining bandwidth. In Figure 20b, we dynamically scale the number of background senders up to 16, adding one sender every 5 seconds, with each sender generating traffic at 5 Gbps, achieving an incast ratio of up to 16-to-1. The figure shows that the BF3-to-BF3 connection, equipped with the receiver-granting credits mechanism, effectively captures the remaining bandwidth across all settings.

In Figure 20c, we demonstrate a scenario of receiver-initiated ECN-like backpressure flow control. We utilize Perftest [84] to establish a QP connection between the sender and receiver, generating WRITE traffic at the line rate (100 Gbps). The receiver can trigger receiver-driven backpressure signals while congestion is happening at its side. Upon receiving peer-domain signals, the sender multiplicatively decreases (MD) the dequeue rate and then performs additive increases (AI) without further backpressure signals. To ensure visibility, we use a small AI factor to extend the AI process. As shown in this experiment, while WRITE is inherently sender-driven, it currently can benefit from the receiver-driven control loop.

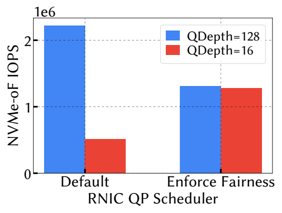
A.9 Application Domain: ML and Storage

We highlight the feasibility of harnessing SCR framework to NVMe-oF RDMA and GPU-Direct RDMA because storage [1, 99] and machine learning workloads [46, 100] take the significant portion of current data center traffic. They can seamlessly leverage our framework with minimal overhead, as SCR operates at the hardware transport level without changing the host software stack. Specifically, it leaves essential data path functionalities like peer-to-peer DMA transfer unchanged (*i.e.,* GPU-to-NIC and SSD-to-NIC). Also, SCR requires no alterations to the application code, host drivers like CUDA, and libraries like SPDK [101] and NCCL [102]. Moreover, SCR ensures optimal raw RDMA performance.

Customized Policy: We ensure application-driven bandwidth allocation for multi-tenant storage or ML workloads without changing any application code. Specifically, multiple NVMe-oF initiators on the same host share NVMe-oF RDMA bandwidth according to the policy, regardless of their individual IO depth settings. Similarly, multiple tenants on the same GPU share GPU-Direct bandwidth according to the policy, even though the size of messages sent from HBM varies among them. The control law is the same as in §5.1, but the policy in §5.1 manages traffic from main memory to PCIe endpoint (EP); this policy controls EP-to-EP traffic.



(a) GPU-Direct RDMA



(b) NVMe-oF RDMA

Figure 21: **Ensuring Fairness in GPU-Direct and NVMe-oF RDMA Traffic.**

Experiments of Achieving Fairness for ML/Storage Workloads: In Figure 21a, we demonstrate how SCR effectively manages GPU-to-GPU RDMA communication. Using Perftest, we set up five concurrent RDMA connections between two GPUs, utilizing GPUDirect RDMA to directly transfer messages between HBMs without involving the host DDR as an intermediary. Here, we demonstrate the impact of an application-driven QP scheduler. Initially, with five connections having different message sizes and using the default scheduler, bandwidth allocation favors large message tenants. When SCR is used to ensure even sharing, all tenants benefit from an equitable share of the bandwidth. Importantly, no modifications were made to any of the host software stack.

In Figure 21b, we showcase the efficacy of SCR for storage workloads utilizing NVMe-oF RDMA. At the NVMe-oF target side, we employ SPDK to configure one *nvmf_tgt* with NULL devices, rather than read SSDs, to maximize network traffic without encountering bottlenecks on the storage medium. On the initiator side, we set up two tenants. Both tenants issue 4KB write requests, while varying the IO queue depth for different tenants. It is evident that with the default QP scheduler, smaller IO depths struggle to obtain the bandwidth. However, when we employ SCR to enforce an application-specific policy of even sharing, these tenants can equitably share the bandwidth without compromising utilization. Importantly, no modifications were required for the SPDK NVMe-oF implementation and the applications.

A.10 SDK Dependency and Implementation

We implement SCR over the DPA in over 7.8K lines of C/C++ code, covering both DPA and host management functionalities. Fabric-domain signal collection and dequeue rate updates are facilitated using the DOCA PCC SDK [68]. For host-domain signal collection, we utilize the RPC and Command Queue mechanisms in the DOCA DPA SDK [60]. Peer-domain signals are managed using the FlexIO SDK [103] for out-of-band communications. Additionally, all analytic, approximation, and arithmetic libraries are implemented in C code, compatible with the DPA RISC-V toolchain.

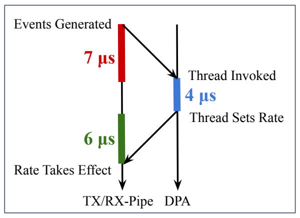
We operate most SCR functionalities on the DPA, only managing very lightweight tasks on the host x86 CPU and none on the DPU ARM cores. By default, we set up 25 DPA threads to handle signals from the fabric domain, including TX, ACK, CNP, NACK, and RTT. One DPA thread establishes communication with the host, and two manage telemetry tasks: one for ping and another for pong. Additionally, two threads exchange signals between peer domains—one for receiving and another for sending. The host CPU is required only to initialize the DPA process and relay the host domain signals to the DPA, for which a single x86 CPU core is sufficient.

A.11 Guidelines for Implementing Other Functionalities

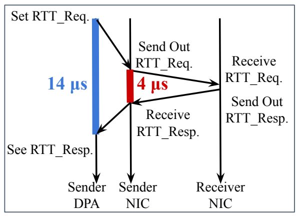
In addition to the aforementioned functionalities, SCR has the potential to support various other customizations. Here, we provide general implementation guidelines based on our analysis and experience.

Timescale of Control Feedback Matters. While SCR can produce per-packet signals at data plane speed, event processing might lag behind such high data rates. We identify three main components contributing to the overall latency of making control decisions: 1) Event triggering, 2) The computation procedure itself, and 3) The time it takes for control feedback to take effect.

We conducted a microbenchmark to quantify the delay breakdown, as shown in Figure 22. For in-band events like TX/ACK events (Figure 22a), event triggering, from the hardware TX/RX pipe generating the events to the DPA invoking the thread to process, could take up to 7 μs. This latency is due to event queuing, event multiplexer operation, and loading the program onto the DPA thread. To accelerate event triggering, we can keep the DPA thread busy waiting rather than relying



(a) In-Band Events



(b) Out-of-Band RTT Events

Figure 22: **Event Processing Latency Breakdown.**

on interrupt-like triggering, which can hide the latency. The second component of latency is the processing itself, which is heavily influenced by the complexity of the control laws. Here, we measure the Swift control law with some additional bookkeeping tasks, like maintaining per-flow dequeue rate, which takes about 4 μs. After making the decision for the new rate, the DPA thread writes new values to hardware registers. We measure how long it takes for the changes to take effect, which may require up to 6 μs to start affecting the data planes. This is longer than expected, and we anticipate that next-gen DPA hardware can optimize this rate-installing procedure. For out-of-band events like RTT (Figure 22b), besides the fabric delay, there are two components: 1) The latency between the DPA setting the request bit and the RNIC actually sending out the RTT request. 2) The delay between the RNIC receiving the RTT response and invoking the DPA thread to process. These two delays can take up to 10 μs, which can be optimized by the DPA thread busy waiting for the RTT response and vendors optimizing the delay between the request set and the request sent out. Overall, we observe that event processing could lag behind the data plane by a few microseconds even after optimization. Control lagging behind data is widely observed for high-speed line rates, not unique to our setting but also in traditional in-kernel congestion control for TCP/IP [104–106]. We propose two guidelines to address the lagging issues: 1) Congestion control algorithms would be better designed to work at the RTT granularity, as more fine-grained control could lead to inaccuracies. 2) If algorithms still aim for sub-RTT control, they should be designed to tolerate signaling coalescing, that is, folding consecutive signals into one event. This approach can help hide latency by combining multiple processing steps into one. Control laws should consider coalesced signals, similar to how Swift considers coalesced ACKs.

Keep Simple yet Effective Control Law. We evaluate simplicity from several aspects: 1) the computation of the control logic, 2) the memory access for per-flow states, and 3) the types of required signals. As we measured above, the latency for computing is positively proportional to the instruction counts of the control logic. Although SCR provides general-purpose computation to implement diverse logic, we still need to accelerate the control logic with approximation techniques to ensure faster reactions. Besides the instruction count, memory access is also expensive in terms of time cost. Reducing the memory footprint, such as fitting into the L1 cache, can reduce the time overhead associated with memory access. On the type of signals, we propose leveraging the signals supported by SCR, such as in-band and out-of-band signals. Although some complex signals like switch-aided in-network telemetry can enhance precision, they come with the cost of end-host processing cycles and switch customizations.

A.12 Suggestions for Next-Gen Datapath Accelerator

Based on our experience implementing SCR over DPA, we suggest improvements for the next-gen hardware in the following areas:

• Decrease event-triggering and rate-installing latency: As we described in the §A.11, the latency of event invocation and rate update is crucial for the timely response of the control loop. Next-gen DPA should optimize program loading time and reduce latency for user-invoked DPA threads via RPC mechanisms (*e.g.,* Mailbox RPC [68]).

• Expand the set of signals/events accessible from the RNIC. Diverse signals are essential to enable flexible control laws. For instance, at present, only in-band TX events are accessible. Rx events indicating the timestamp and bytes received would provide valuable additional signals. Currently, the RX pipe does not provide the API for the DPA to access INT fields. Also, expanding the set of NIC counters accessible for DPA would be beneficial. Currently, only TX/RX bytes counters are available. For instance, exposing utilization counters for RNIC micro-architecture resources accessible to DPA, as Neo-Host [107] enables, is promising.

• Expand the set of control actions. Currently, only the rate and RTT requests are considered control actions. As previously mentioned, incorporating additional control actions such as modifying the RoCEv2 header, like adjusting the source port or setting the ECN bits, would be beneficial.

• Streamline and improve the flexibility of DPA process management. The DOCA PCC process is currently the only one capable of adjusting the rate. However, at present, it cannot be easily extended to support other functionalities such as accessing host memory or operating raw Ethernet APIs. As a result, we must set up other DPA processes to access host memory and operate raw Ethernet APIs, which introduces some overhead in inter-process communication (IPC).

A.13 Extended Analysis for Prior Work

Table 6 is an extended list of related works, and we analyze the feasibility of supporting these customizations using SCR.

Prior Work	**Why Customization**	**Prior Customization**	**Feasibility and Limitations using SCR**
Microsoft: DCQCN [2]; Large-Scale Deployment [3]; Storage [1]	CC / FC	ASIC/ Tuning	DCQCN and PFC are SCR default strategies.
AWS: SRD [5]	CC / Re. / MP	Bespoke HW	SCR explores delay-based CC and multi-path routing similar to SRD.
Google: Falcon [4]; Swift [10]; PLB [44]; Carousel [71]; SNAP [108]	CC / Re. / MP	Bespoke HW	SCR showcases the implementation of Swift.
Ultra Ethernet Consortium [8]	CC / Re. / MP	Bespoke HW	Case studies of SCR follow design proposals by UEC.
Hyperscale Issues [45]	CC / FC / More	-	SCR demonstrates some solutions for issues reported.
Alibaba: Solar [99]; Pangu [109]	CC / Re. / MP	Bespoke HW	Solar utilizes UDP Src Port for multi-pathing and per-ACK CC requiring several path conditions (*e.g.,* RTT), which are supported by SCR.
ZeroNIC [110]	Data/Ctrl	FPGA	SCR supports data/ctrl path separation for RDMA.
MLT [23]	Re.	Userspace	SCR can monitor packet loss explored by MLT, but current BF3 has no APIs for SCR to program reliability logic.
ACC [11]	CC	Simulation	SCR supports per-ACK CC like ACC.
CASSINI [111, 112]	CC	Tuning	SCR supports encoding hints from ML workloads to the CC logic.
MegaScale [46]	CC	ASIC	MegaScale proposes ECN-RTT hybrid CC, which is supported by SCR.
Harmonic [17]	Iso. / FC	FPGA	SCR can support rate limiter logic required by Harmonic.
Host CC [98]	CC	In-Kernel	Host memory contention signal is one of SCR host-domain signals.
ConWeave [113]	MP	Switch	SCR focuses on endhost, which is orthogonal to switch innovations.
Hostping [114]	Iso.	SW	SCR applies the manner of Hostping to probe host PCIe utilization.
SRNIC [26]	Scalability	FPGA	SRNIC focus on RNIC connection scalability, which is orthogonal to the SCR scope.
Flor [24]; X-RDMA [49]	CC / Re.	Userspace	Flor enforces the customization at the message level above the transport layer while SCR directly customizes the transport. Flor enforces message-granular control while SCR enforces packet-granular control.
Hermes [115]	CC	Switch	SCR is compatible with Hermes mechanisms of incast control on switches.
CORN [116]	CC	Userspace	CORN enforces CC logic above RoCEv2 in userspace while SCR directly customizes RNIC CC.
Justitia [18]	ISo.	Userspace	Justitia investigates the RDMA multi-tenancy issues. SCR enforces multi-tenant isolation via customizing the QP scheduler. Justitia enforces message-granular control while SCR enforces packet-granular control.
EQDS [16]	CC / Re. / MP	Tunneling	SCR explores receiver-driven credit schemes similar to EQDS.
BFC [14]	FC	Switch	SCR can benefit from switch-side fine-grained flow control.
TCD [12]	CC	Switch	SCR can benefit from accurate congestion detection on the switch side.
Tonic [117]	CC / Re.	FPGA	Tonic explores FPGA-based programmable CC. SCR is based on general-purpose cores to re-program CC.
PCN [118]	CC	Simulation	SCR provides ECN signals which are required by PCN.
1RMA [25]	CC / Re.	Bespoke HW	1RMA requires bespoke HW.
HPCC [9]	CC	FPGA	Currently, BF3 does not provide INT signals for SCR. Alternatively, SCR utilizes out-of-band packets to carry in-network information.
IRN [22]	Re.	FPGA	Currently, retransmission logic is handled by the RNIC data path. NACK signals are available for SCR.
MP-RDMA [20]	MP	FPGA	SCR demonstrates multi-path selection at the sender side.
Tassel [70]	CC	FPGA	SCR can benefit from a better rate limiter.
TX-Arch [56]	Iso.	FPGA	TX-Arch investigates the RNIC multi-tenant contention. SCR enforces the fairness of the QP scheduler.
RCC [51]	CC	Simulation	SCR demonstrates receiver-driven control loop.
Avatar [119]	CC	Userspace	SCR can enforce more fair CC without additional userspace processing.
Shen et al., [120]	Iso.	Userspace	SCR enforces fairness for multi-tenant RDMA without additional userspace processing.
UL-MPRDMA [21]	MP	Userspace	SCR supports probing multi-path real-time conditions.
sRDMA [58]	Re.	Userspace	SCR supports customizing the QP scheduler without additional userspace overhead.
RoGUE [28]	CC / Re.	Userspace	SCR supports customizing CC.
Le et al., [121]	Iso.	Tunning	SCR supports tuning DCQCN.
LDCP [122]	CC	PCC	PCC is submodule of SCR.
RL-CC [123]	CC	PCC	PCC is submodule of SCR.
Restructuring CC [104, 105]	CC	In-Kernel	SCR explores restructuring CC in the context of RDMA transport.
pHost [53]; NDP [52]; Homa [15]	CC / FC	In-Kernel	SCR explores receiver-driven control loop.

(*CC* denotes Congestion Control. *FC* denotes Flow Control. *Iso.* denotes Isolation. *Re.* denotes Reliability. *MP* denotes Multi-Pathing.)

Table 6: **Prior Innovations in RDMA Transport Customization and Analyzing Feasibility with the SCR Framework.**