



Cache-Aware I/O Rate Control for RDMA

Qijing Li

Hong Kong University of Science and
Technology
Hong Kong, China
qlicw@connect.ust.hk

Xinyang Huang

Hong Kong University of Science and
Technology
Hong Kong, China
xhuangci@connect.ust.hk

Bowen Liu

Hong Kong University of Science and
Technology
Hong Kong, China
liubw@ust.hk

Pengbo Li

Hong Kong University of Science and
Technology
Hong Kong, China
pliht@connect.ust.hk

Junxue Zhang

Hong Kong University of Science and
Technology
Hong Kong, China
jzhangcs@connect.ust.hk

Kai Chen

Hong Kong University of Science and
Technology
Hong Kong, China
kaichen@cse.ust.hk

Abstract

Remote Direct Memory Access (RDMA) has become a cornerstone technology in modern datacenter networks due to its high throughput and extremely low latency. However, recent works have revealed that congestion arises in the "last mile" of the RDMA I/O path—between DRAM and CPU registers—due to inefficiencies in the memory hierarchy, where severe cache misses and memory bandwidth contention degrade performance. We identify the root cause of this I/O congestion as the speed mismatch between network ingress and CPU processing, which leads to data accumulation and, eventually, last-level cache overflow. To address this, we propose CARC, a credit-based rate control mechanism that dynamically aligns network ingress speed with CPU processing speed. Our preliminary evaluation on eRPC over RDMA, a widely used RPC framework, demonstrates that CARC effectively mitigates I/O congestion, reducing flow completion time by up to 1.40× and improving throughput by up to 1.35× compared to prior work.

CCS Concepts

• **Software and its engineering** → **Operating systems**; • **Networks** → **Transport protocols**; **Data center networks**.

Keywords

I/O Congestion Control, Datacenter Networks, RDMA

ACM Reference Format:

Qijing Li, Xinyang Huang, Bowen Liu, Pengbo Li, Junxue Zhang, and Kai Chen. 2025. Cache-Aware I/O Rate Control for RDMA. In *9th Asia-Pacific Workshop on Networking (APNET 2025), August 07–08, 2025, Shang Hai, China*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3735358.3735376>

1 Introduction

Datacenter networks are undergoing a revolution in the post-Moore era [12, 17]. While link throughput continues to scale to 100/200/400Gbps, CPU processing power is growing at a significantly slower

pace, leading to a bottleneck transition from in-network processing (i.e., between switches and NICs) to the host I/O path (i.e., between NICs and CPUs) [7, 8, 13, 20, 37, 45]. To address this imbalance, researchers have explored various approaches, including NIC offloading [6, 31, 32, 34, 40] and efficient packet processing techniques [5, 18, 25, 38, 39], to alleviate the computational burden on CPUs.

Among these techniques, RDMA has emerged as the most widely adopted solution for distributed systems, offering a fully offloaded network stack and a high-performance I/O library [11, 24, 26, 27, 35, 49]. In the RDMA I/O path, received network packets are directly transferred to the application's memory space, bypassing the CPU. Thus, RDMA is commonly believed to ensure line-rate performance. However, recent studies [14, 15, 43, 46, 47, 51] reveal that severe congestion occurs in the "last mile" of the RDMA I/O path—between the DRAM and CPU registers—due to the inefficient memory hierarchy where severe cache misses and memory bandwidth contention occur. This leads to significant performance degradation, a phenomenon referred to as I/O congestion.

HostCC [1] is the first approach to mitigate this issue by extending congestion control protocols with an intra-host congestion signal—the Integrated I/O (IIO) buffer occupancy. Specifically, when the memory controller becomes congested, it backpressures the IIO buffer, leading to high occupancy. By monitoring IIO occupancy, HostCC detects I/O congestion and propagates congestion signals to the network, reducing queueing delays in the "last mile" and lowering tail latency. However, as discussed in §2, HostCC fails to eliminate I/O congestion entirely, as inefficiencies in the memory hierarchy persist. This is evidenced by a high cache miss rate, indicating that memory access remains a critical bottleneck.

In this paper, we ask: *What is the root cause of I/O congestion, and how can it be addressed?* To answer this, we analyze the micro-behaviors of the memory hierarchy during the entire I/O congestion process, and identify that the speed mismatch between network ingress and CPU processing is the root cause of I/O congestion. This mismatch causes I/O data to be frequently flushed to DRAM before the CPU can process it, leading to inevitable cache misses and reducing memory hierarchy efficiency. Furthermore, our observations reveal that narrowing this speed gap slows down the flush process, and timely speed coordination ensures that the unprocessed data never exceeds the Last-Level Cache (LLC), providing an opportunity to eliminate I/O congestion.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

APNET 2025, Shang Hai, China

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1401-6/25/08

<https://doi.org/10.1145/3735358.3735376>

Inspired by this observation, we propose CARC¹, which leverages a credit-based rate control mechanism to dynamically align network ingress speed with CPU processing speed in the same rhythm of change. To account for control inaccuracies caused by variable round-trip times (RTT) and unpredictable application memory accesses, we further introduce loosened credits to provide additional tolerance. Our preliminary evaluation on eRPC over RDMA, a widely used RPC framework, demonstrates that CARC effectively reduces cache misses and mitigates I/O congestion, leading to up to 1.40× lower flow completion time (FCT) and 1.35× higher throughput compared to HostCC. We summarize our contributions are:

- We model the root causes of I/O congestion and validate our analysis through comprehensive experiments (§3).
- We design and implement CARC, a rate control system that coordinates the receiver’s network ingress speed with CPU processing speed to address I/O congestion (§4).
- Microbenchmark experiments demonstrate that CARC effectively reduces I/O congestion while introducing negligible computational overhead (§5).

2 Background and Motivation

We begin with a brief introduction to the RDMA I/O path, focusing on why I/O congestion happens and its impact on performance degradation. Then, we revisit the state-of-the-art solution, HostCC [1], to highlight its limitations and explore potential optimization opportunities.

RDMA I/O Path. Remote Direct Memory Access (RDMA) is a technology that enables direct memory access initiated by a remote endpoint over a network. The foundation of RDMA is the RDMA Network Interface Card (RNIC), which integrates a full network stack and manages local registered memory to handle remote memory access requests (i.e., RDMA verbs) without involving the host CPU. By bypassing the CPU, RDMA can achieve line-rate throughput and ultra-low latency communication, making it a cornerstone technology in modern datacenter applications [11, 16, 24, 25, 27, 35, 48–50]. As illustrated in Figure 1, an RDMA packet received by the RNIC undergoes the following steps in the I/O path:

- ① The RNIC receives packets from the network, processes the transport protocol to ensure reliable and ordered delivery, re-assembles them into application data, and stores them in the RX buffer.
- ② The RNIC fetches a Work Queue Element (WQE) from the WQE cache or parses the incoming packet header to obtain the metadata. This metadata describes the RDMA operation, including the verb type (e.g., read, write, send, receive), target memory address, and data length. WQEs are pre-initialized during RDMA connection setup.
- ③ Based on the metadata, the RNIC initiates local Direct Memory Access (DMA) operations through its DMA engine, encapsulating them as PCIe transactions. The application data is then written to the destination memory address and temporarily buffered

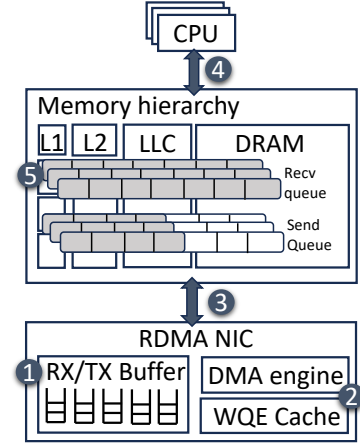


Figure 1: The RDMA I/O path in the receiver with DDIO acceleration.

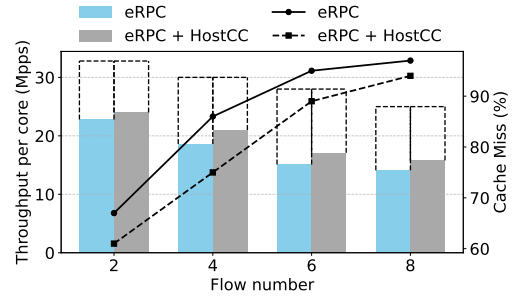


Figure 2: Limitations of HostCC under varying connection numbers, where the flow size is 7.9MB.

in the Last-Level Cache (LLC) via Data Direct I/O (DDIO) acceleration² [10, 29].

- ④ The application asynchronously retrieves data from the LLC and processes it based on user-defined logic. The completion of data arrival is signaled via Completion Queue Elements (CQEs), which the RNIC generates upon completing the DMA operation.
- ⑤ After processing the received data, the application replenishes WQEs in the Receive Queue (RQ) if needed.

I/O Congestion. Recent studies have shown that the above I/O path may suffer from severe cache evictions due to the limited cache capacity [1, 14, 15, 33, 43, 46, 47, 51], leading to low efficiency in the memory hierarchy and queueing in RQs. Specifically, data stored in the RQ fills up LLC in the order of arrival (Step 3). Once all cache lines are occupied, newly received data will evict the earliest cached entries to make room. More critically, cache evictions can occur across different RQs when the memory addresses of buffers in separate queues map to the same cache set. These evictions introduce significant performance degradation:

- *Heavier CPU Burden.* When an application accesses evicted data from DRAM (e.g., for RPC processing), cache misses occur, leading to additional memory access overhead. This increases CPU cycles

²DDIO, first introduced in Intel Xeon E5 processors, enables network data to be written directly to the LLC, bypassing the longer-latency DRAM access. Similar optimizations have been adopted in AMD architectures [4] and further explored in research [19, 44].

¹CARC is short for Cache-Aware Rate Control.

and hinders the expected packet processing rate. Furthermore, since the application is responsible for replenishing WQEs in the RQ (Step 5), a slower replenishment rate delays subsequent DMA operations, ultimately causing backpressure on network ingress (e.g., through Priority Flow Control (PFC) [22]).

- **Excessive Memory Bandwidth Consumption:** Evicted data must be reloaded from DRAM, effectively doubling memory bandwidth consumption. Once memory bandwidth is saturated, the processing rate of DMA operations degrades, further exacerbating backpressure on network ingress.

In this paper, we refer to such processing rate degradation caused by cache evictions as *I/O Congestion*, and we will deeply analyze this phenomenon in the following sections.

HostCC and Its Limitations. To address I/O congestion, HostCC [1] extends traditional congestion control protocols, such as DCTCP [3], by incorporating I/O congestion signals. Specifically, HostCC monitors the occupancy of the Integrated I/O (IIO) buffer, which reflects the volume of in-flight data in the I/O path. When the buffer occupancy exceeds a predefined threshold, HostCC generates a congestion signal and propagates it via the network protocol, throttling the network ingress rate to mitigate CPU workload and memory bandwidth consumption.

However, HostCC operates in a reactive manner, meaning it intervenes only after I/O congestion has already occurred. This results in an inherent delay, during which the IIO buffer accumulates excessive data before the rate control mechanism takes effect. Consequently, cache misses become inevitable as illustrated in our experiments of deploying HostCC to eRPC [25] (Figure 2). Experiments show that as the degree of I/O congestion increases—e.g., by increasing the number of concurrent RDMA connections—the per core throughputs of eRPC and HostCC-enhanced configuration (HostCC+eRPC) both suffer from 1.62× and 1.52× performance degradation, respectively. The performance improvement provided by HostCC is limited and diminishes as congestion intensifies, eventually yielding no speedup under severe congestion.

These findings highlight the need for a deeper investigation into cache behavior in the RDMA I/O path, which motivates us to model I/O congestion more precisely and explore *proactive* optimization strategies to address it.

3 Modeling I/O Congestion

In this section, we dive into the root reasons of cache misses in network I/O with modeling (§3.1), and conduct experiments to validate our analysis (§3.2).

3.1 Producer-Consumer Speed Mismatch

Dive into Cache Behavior. Figure 3 illustrates a simplified case of cache replacement in network I/O, where the receiver reuses 8 buffers in RQ to receive messages sequentially and our cache policy is basic LRU. Assume DDIO cache³ (marked as dark) can accommodate 2 buffers, while non-DDIO cache can accommodate 3 buffers for brevity. In step 1, incast messages are DMAed to pre-posted buffers, during which DDIO cache is flushed and finally holds

³DDIO cache is part of LLC, typically two ways of LLC.

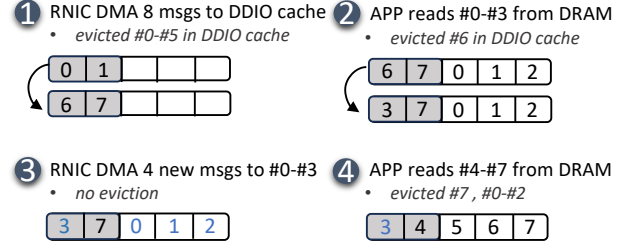


Figure 3: A simplified cache replacement in I/O path.

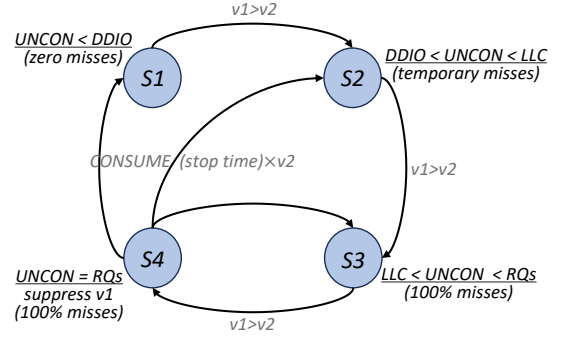


Figure 4: State transition diagram of I/O congestion.

buffers #6-#7. In step 2, application begins processing messages and cache misses happen. Meanwhile, the retrieval of evicted data would cause another eviction to part of useful data (i.e., #6). In step 3, newly incast messages (marked as blue) are written to the posted buffers (i.e., #0-#3) without eviction since buffers are in LLC. In step 4, application continues reading remaining messages of last round (i.e., #4-#7), which triggers cache misses and flushes the LLC again.

In this procedure, the reason of cache misses is the eviction of useful data (i.e., the data has not been processed by application). The types of data evictions are as follows:

- **RNIC eviction:** When RNIC receives a message while the destination address is not in LLC and the DDIO cache is full, old data will be evicted from DDIO cache to make space for new payload. Step 1 is an example of this case.
- **CPU eviction:** When CPU reads a message while the message buffer is not in LLC, CPU will evict data and fetch needed buffer from DRAM. Step 2 and 4 are examples of this case.

Root Reason: Speed Mismatch. The I/O path can be viewed as a producer-consumer model. We observe that both types of evictions are attributed to speed mismatch between network ingress (denoted as producer) v_1 and CPU processing (denoted as consumer) v_2 . Mismatch of speeds leads to *unconsumed data accumulation*. In step 1, when $(v_1 - v_2) \times t \geq DDIO_SIZE$, unconsumed data exceeds DDIO cache, thus evictions happen and useful data are flushed to DRAM. Such mismatch and flushes continues until RQs are filled up (i.e., all available WQEs have been exhausted, $(v_1 - v_2) \times t = RQs_SIZE$ ⁴), then the producer and consumer speeds will achieve a balance (Step 3) as the producer speed is bound by available WQEs. However, such match will not mitigate cache misses, as the queued

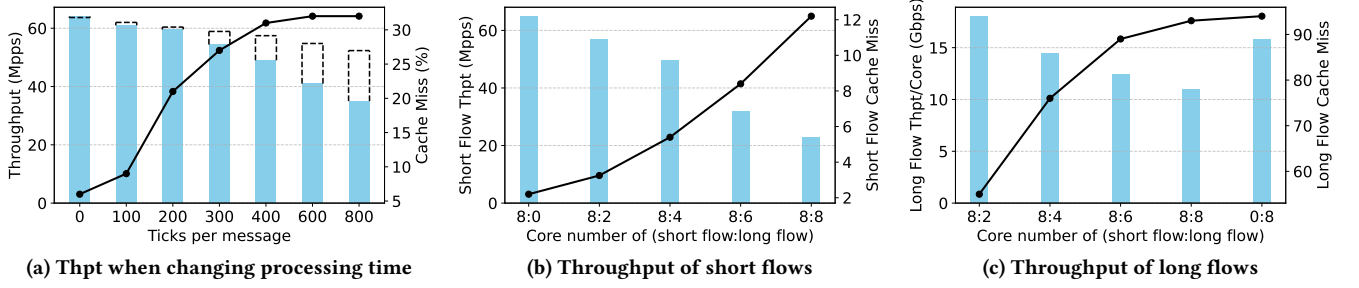


Figure 5: eRPC throughput (thpt) and cache misses rate when changing CPU processing rate or network ingress rate. Bar is the thpt and line is the cache misses rate. Dotted bars in (a) denotes theoretical thpt degradation.

unconsumed data size (i.e., RQs_SIZE) is larger than LLC size⁴ and will not reduce. The read of head unconsumed data always lead to eviction of tail unconsumed data (Step 2 and 4).

We represent the data accumulation degree with four states in Figure 4. S1 means the unconsumed data size is less than DDIO cache size, where no cache misses happen; S2 represents the unconsumed data size is larger than DDIO size but less than LLC size, in which case, though cache misses happen due to RNIC eviction in the beginning, after all buffers are read from LLC, cache misses stop; S3 and S4 means the unconsumed data size is larger than LLC size but limited by RQs' size⁴, cache misses is 100% as the example in Figure 3 shows. In S4 state, the speed match is achieved by network rate control (e.g., PFC, DCQCN, Receive not Ready(RnR) in RoCE), which is delayed and conservative as they rely on signal or timeout setting [22, 54]. Thus, producer's average speed is far below consumer speed as there exists a latency between buffer post and new message arrives, which give opportunities for CPU to consume massive messages before new messages arrive. In this case, S4 can transfer back to S1, S2 or S3, decided by v_2 and stop time of sender (i.e., timeout value or network RTT). Noticed that, though above analysis is based on usages of WQEs (i.e., using double-sided verbs), it can be seamlessly extended to single-sided verbs by redefining the RQs' size as the total buffer space registered by receivers.

Modeling I/O Congestion. The state transition diagram helps to bridge the producer-consumer speeds (v_1 and v_2) and performance metric—cache miss rate. Specifically, cache miss rate is decided by the mismatch gap ($v_1 - v_2$). A smaller gap leads to more time spent in earlier states (S1 and S2), resulting in fewer cache misses and higher performance in RDMA I/O—**A shorter duration in S1 and S2 indicates more severe I/O congestion.** Thus, the mismatch gap is a good indicator to model the degree of I/O congestion. The reason we do not use cache miss rate as an indicator is that such metric is too sensitive to the cache policy, pre-fetching, application access pattern, etc., which makes it difficult to reflect the root reason of I/O congestion.

Note that modern CPUs adopt complex cache policies, including optimized eviction algorithms [23, 42] and prefetching techniques. However, our experiments in §3.2 and §5 show that these optimizations gain minimal improvement on addressing I/O congestion.

⁴In our context, RQs' size means the total buffer space pointed by WQEs of all RQs. For example, if 16 RQs each have 4096 entries, and the buffer size is fixed at 1500 bytes, the total RQs' size is calculated as: $RQs' size = 16 \times 4096 \times 1500B = 94MB$.

3.2 Validation of I/O Congestion Model

In this section, we validate our I/O congestion model by controlling the speed mismatch gap and observing the cache miss rate and throughput variations. To be more realistic, we deploy eRPC [25] over RDMA, a popular high-performance RPC framework, to generate end-to-end workloads.

Experiment Setup: We deploy eRPC with hostCC on two servers with 200Gbps network configurations, each equipped with two Intel Xeon Silver 4309Y CPUs, one NVIDIA ConnectX-7 200GbE NIC with PCIe 4.0×16 interconnection, and 512GB DDR4 3200MT/s DIMMs connected to 8 memory channels.

Scenario 1: Reduce V2. In this case, we change the processing time for each received message (i.e., ticks per message), which is a common scenario in datacenter as different application behaviors have different processing times. When the processing time increases, the consumer speed v_2 defined in §3 decreases. We evaluate the end-to-end throughput for basic echo operations using 16 cores within a NUMA, where one server sends 256B requests to another, which then reads payload, generates 256B responses back. Results are shown in Figure 5(a)⁵, where throughput decreases by up to 45% and read cache misses increases to over 30% with increasing processing time. The results are consistent with our analysis in §3. When processing is fast ($v_1 \leq v_2$), receiver remains in S1 with nearly zero cache misses. When processing time increases ($v_1 > v_2$), receiver fluctuates between four states and the time portions spent in different states are decided by mismatch degree of v_1 and v_2 . With smaller v_2 , more time is spent on posterior states, where cache misses rate is high.

Scenario 2: Increase V1. In this scenario, we change the network ingress rate of receiver, which is a common scenario in datacenter as different flow sizes leads to different network ingress rates (i.e., long flows have higher throughput (Gbps)). When the network ingress rate increases, the producer speed v_1 defined in §3 increases. We locate two types of flows on one sever, and the flow sizes are 7.9MB, 72B respectively, equipped with same 32B responses. We use 8 cores to process messages of short flows, and assign different number of cores (0-8 cores) to long flows, through which we can increase v_1 of receiver via increasing core number of long flows. The results of short flows' total throughput (Mpps) and long flows' per core throughput (Gbps), are shown in Figure 5(b) and Figure 5(c) respectively. In Figure 5(b), with the increase of core number of long

⁵Ticks per message=0 means that application just reads payload once, then sends response.

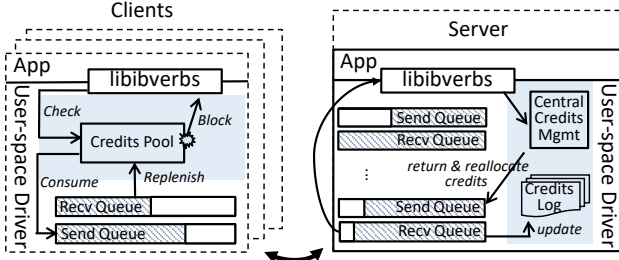


Figure 6: The architecture of CARC.

flows, the throughput of short flows decreases by up to 60% with surging read cache misses increases. This is because the ingress data of long flows evict short flows' data in LLC, which become more severe with the increase of core number of long flows⁶. From Figure 5(c), we can see that the long flows' cache miss rates also increase to more than 90% when introducing more cores, which bounds the performance of long flows that per core throughput decreases by up to 60%. Results of Figure 5(b)(c) are also in line with our analysis in §3. With increasing v_1 , S1 can transfer to S4 quickly, which leads the receiver keep staying in high cache misses state.

To summarize, the results in Figure 5 are consistent with our analysis in §3.1 that the speed mismatch of producer and consumer leads to states transitions and cache misses (i.e., From S1 to S4). With more severe mismatch, cache misses become more serious, resulting in greater performance degradation (i.e., Degeneration to S4 is more frequent and faster).

Our Insight. Narrowing the gap between producer and consumer speeds can slow down the state transfer. If we can identify the current status of system and timely apply rate control (e.g., when detecting status approaches S3, reduce v_1 until it less than v_2), it is possible to avoid S3 and S4, thus eliminating I/O congestion.

4 CARC Design

In this paper, we propose CARC, aligning the producer's speed with the consumer's rhythm as soon as possible to avoid I/O congestion. We achieve this goal by adopting credit-based rate control for speed coordination, with loosened credit adjustment to tolerate coordination error due to RTT latency and I/O-unrelated memory access. Figure 6 illustrates the overall workflow of CARC.

Credit-Based Rate Control. Typically, the consumer rate changes frequently and is hard to predict, for example, in RPC, the processing time of different requests varies significantly. Therefore, we adopt a credit-based rate control to force the network ingress rate (producer speed) to match receiver's processing rate (consumer speed), whatever the consumer rate changes to.

Specifically, CARC maintains a credit pool for each RDMA connection, where each credit represents a cache line in receiver's LLC. When the sender sends an RDMA request, CARC first checks the available credits and only posts the request to RNIC if credits

⁶We also conducted experiment to show that the performance degradation of short flows is not caused by HoI problem as there is no queuing in NIC and using priority flow control mechanism [22] has no improvement.

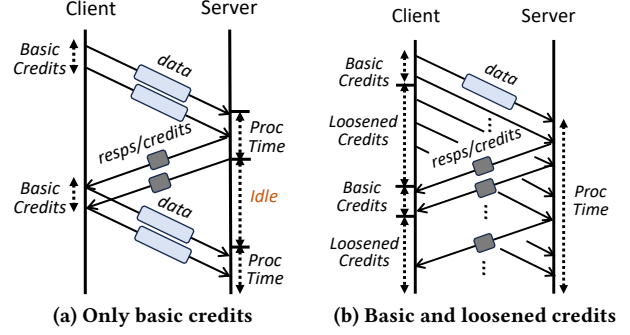


Figure 7: Comparison between a system with only basic credits and a system with loosened credits. Using only basic credits may lead to server idling when credits are exhausted, whereas loosened credits provide precise supplementary credits to prevent idling.

are enough. Otherwise, request is blocked until credits are replenished. When the receiver finishes processing the data, it triggers CARC and returns corresponding credits. Noticed that, in order to reduce synchronization overhead, when application's communication mode is request-response, CARC will not explicitly return credits as receiving response represents its request's credits are returned. Only when application's communication mode is one-way, CARC explicitly return credits to sender when finishing processing data. Since the LLC on the receiver side is shared by all connections, each sender's credits are determined by the LLC size and number of connections on the receiver side. We can calculate the number of credits by Formula 1:

$$C = \frac{s_{llc}}{s_{line} \times n}, \quad (1)$$

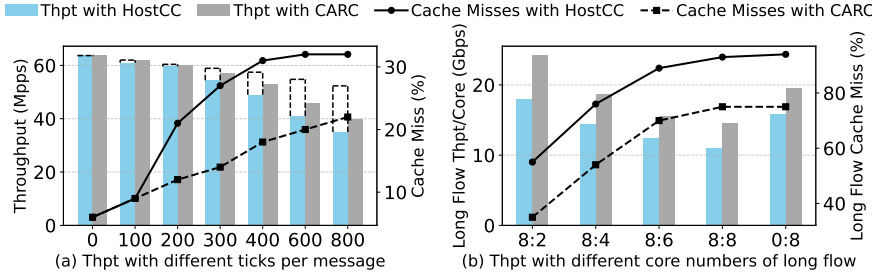
where we denote cache line size as s_{line} , LLC size as s_{llc} and connection number as n . We define credits in the granularity of cache line to accommodate packets with different sizes while considering cache access granularity. Through this mechanism, unconsumed data cannot exceed LLC size and receiver can stay in state S1 and S2 with zero cache miss.

Loosened Credits. Such above case is built on the assumption that the sender can receive the returned credits with *zero latency*, which is not practical due to network latency. Specifically, when receiver finishes processing existing data and returns credits, it needs to wait for near RTT time for new data if client's credits are exhausted before, which leads to receiver's CPU idle, as shown in Figure 7(a).

It is also worth noting that applications may have other memory accesses unrelated to I/O (e.g., looking up tables in key-value stores), which incurs unexpected I/O data eviction in S2 and may degrades receiver to enter S3 or S4. To address above problems, we introduce loosened value δ to original credits, formulated in Formula 2:

$$C_{loosened} = \frac{s_{llc}}{s_{line} \times n} + \delta. \quad (2)$$

When CARC observes that the receiver's CPUs are idle (i.e., it is in state S1 with zero unconsumed data, waiting for messages arrival), we increase δ to enlarge the sender's credits and increase v_1 . In this way, we enable the sender to have in-flight messages outside LLC, providing a supplement to receiver to overlap RTT latency



Flows Ratio	with HostCC		with CARC	
	P50	P99	P50	P99
8:2	3883	4025	2786 (↓1.4×)	2926 (↓1.4×)
8:4	4765	4865	3625 (↓1.3×)	3753 (↓1.3×)
8:6	5403	5556	4129 (↓1.3×)	4286 (↓1.3×)
8:8	5983	6275	4341 (↓1.4×)	5109 (↓1.2×)

Table 1: FCT (us) of long flows in eRPC with HostCC or CARC. Flows Ratio refers to the ratio of CPU cores allocated to short flows and long flows. The symbol ↓ indicates the speedup factor.

Figure 8: Throughput and cache misses rate of eRPC deployed with hostCC and CARC when changing message processing speed or network ingress speed.

of returning credits. Specifically, as shown in Figure 7(b), client is allocated with extra loosened credits, whose value is decided by receiver data processing rate multiplying RTT time. Then, during the interval of RTT time, data volume consumed by receiver is equal to the loosened data volume, followed by receiving new round of data using basic credits. Thus, total credits are always enough for receiver to avoid CPU idle. Loosened credits value needs to be updated frequently to reflect the processing rate changes. To mitigate the synchronization overhead of server sending back updated credits to clients, CARC calculated updated value in clients through measuring latency between request and response, which is consisted of two parts: RTT and request processing time.

When CARC detects receiver’s status approaches S3 though with bounded credits allocation, we infer that there are other I/O-unrelated memory accesses occurring in the server. To address this, we set a negative δ to limit the credits allocated to senders, thereby adapting to the unknown intensive memory accesses. This reduces v_1 , curbing I/O ingress and rolling back the system status to S2.

5 Evaluation

We use the same experiment setup in §3 to evaluate the effectiveness of CARC, and we report our results in Figure 8. We observe the following results:

- In Figure 8(a), we vary the processing speed of receiver. Results show that our credit-based rate control can adapt to the change of consumer speed, and reduce the cache misses by up to 42% compared to hostCC. Meanwhile, the throughput is increased and is closer to the theoretical limit.
- In Figure 8(b), we vary the ingress rate of receiver through involving more long flows. Results of long flows show that after deploying CARC, throughput is increased by up to 1.35× compared to baseline and cache misses are reduced by 20%. FCT of long flows are presented in Table 1, which shows that CARC can reduce FCT by up to 1.4×. Cache misses cannot be reduced to zero because application (i.e., eRPC) maintains another set of message buffers to assemble large messages before processing, thus incurring memory copy and dense memory access.

6 Discussion

Integrated with local I/O control. Beyond hostCC, several other approaches can alleviate I/O congestion to some extent. For instance,

prior works such as [9, 21, 41] propose the use of shared ring buffers between queues to reduce memory footprint. Additionally, [2, 52] allocate more cache space for DDIO by adjusting LLC allocation strategies or utilizing higher-level caches. We clarify that CARC operates orthogonally to these approaches and serves a distinct role in I/O optimization. Specifically, CARC and hostCC provide end-to-end adaptation as enhancements to transport protocols, while these new I/O architectures offer intra-host mechanisms to mitigate memory pressure. These two approaches are complementary and can be integrated to achieve enhanced performance.

Integrated with network congestion control. Network congestion controls (NCC) adapted by RDMA [28, 30, 36, 53, 54] primarily focus on the network congestion signals, such as queue occupancy and packet loss, to regulate the sending rate. However, such signals are unsuitable for I/O congestion control, as they do not accurately capture the congestion of the host memory system. CARC is designed to complement existing NCCs by specifically targeting host I/O congestion. We currently implement CARC as an independent module in RDMA userspace driver to provide flexibility and extensively support for various RDMA transport protocols. While promising, integrating CARC with existing NCCs presents significant challenges. Such integration requires hardware modifications, as offloading transport layers is increasingly prevalent in the RDMA community. Moreover, careful consideration of interactions between the two congestion control systems is essential, along with meticulous tuning of hyperparameters to prevent oscillatory behavior. We leave the exploration of integration to future work.

7 Conclusion

RDMA-based systems experience severe I/O congestion. In this paper, we model I/O congestion as a speed mismatch between network ingress and CPU processing. To address this, we propose CARC, which coordinates these speeds within the same rhythm of change. Preliminary evaluations demonstrate that CARC effectively mitigates I/O congestion and enhances the performance of RDMA-based systems.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work is supported in part by the Hong Kong RGC TRS T41-603/20R, the GRF 16213621, and the ITC ACCESS. Kai Chen is the corresponding author.

References

- [1] Saksham Agarwal, Arvind Krishnamurthy, and Rachit Agarwal. 2023. Host Congestion Control. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 275–287.
- [2] Mohammad Alian, Siddharth Agarwal, Jongmin Shin, Neel Patel, Yifan Yuan, Daehoon Kim, Ren Wang, and Nam Sung Kim. 2022. IDIO: Network-driven, inbound network data orchestration on server processors. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 480–493.
- [3] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*. 63–74.
- [4] Luna Backes and Daniel A Jiménez. 2019. The impact of cache inclusion policies on cache management techniques. In *Proceedings of the International Symposium on Memory Systems*. 428–438.
- [5] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: a protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 49–65.
- [6] Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. 2022. hXDP: Efficient software packet processing on FPGA NICs. *Commun. ACM* 65, 8 (2022), 92–100.
- [7] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 65–77.
- [8] Qizhe Cai, Midhul Vuppalapati, Jaehyun Hwang, Christos Kozyrakis, and Rachit Agarwal. 2022. Towards μ s tail latency and terabit ethernet: disaggregating the host network stack. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 767–779.
- [9] Youmin Chen, Youyou Lu, and Jiwu Shu. 2019. Scalable RDMA RPC on reliable connection with efficient resource sharing. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–14.
- [10] Intel Corporation. Intel data direct i/o technology (intel DDIO): A primer. 2012. <https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/data-direct-i-o-technology-brief.pdf>. Accessed 2025-01-01.
- [11] Aleksandar Dragojević, Dushyant Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 401–414.
- [12] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*. 365–376.
- [13] Alireza Farshin, Tom Barbette, Amir Roozbeh, Gerald Q Maguire Jr, and Dejan Kostić. 2021. PacketMill: toward per-Core 100-Gbps networking. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 1–17.
- [14] Alireza Farshin, Amir Roozbeh, Gerald Q Maguire Jr, and Dejan Kostić. 2019. Make the most out of last level cache in intel processors. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–17.
- [15] Alireza Farshin, Amir Roozbeh, Gerald Q Maguire Jr, and Dejan Kostić. 2020. Reexamining Direct Cache Access to Optimize I/O Intensive Applications for Multi-hundred-gigabit Networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 673–689.
- [16] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 202–215.
- [17] John L Hennessy and David A Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2019), 48–60.
- [18] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. 2018. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies*. 54–66.
- [19] Ram Huggahalli, Ravi Iyer, and Scott Tetrick. 2005. Direct cache access for high bandwidth network I/O. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 50–59.
- [20] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. 2020. TCP≈RDMA: Remote Storage Access with i10. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 127–140.
- [21] InfiniBand Trade Association (IBTA). 2025. What is InfiniBand. <https://www.infinibandta.org/ibta-specification/>. Accessed 2025-01-01.
- [22] IEEE. 2020. 802.1Qbb, Priority-based Flow Control. <https://1.ieee802.org/dcb/802-1qbb/>. Accessed 2025-01-01.
- [23] Aamer Jaleel, Kevin B Theobald, Simon C Steely Jr, and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). *ACM SIGARCH computer architecture news* 38, 3 (2010), 60–71.
- [24] Kim Jongyul, Jang Insu, Reda Waleed, Im Jaeseong, Canini Marco, Kostic Dejan, Kwon Youngjin, Peter Simon, and Witchel Emmett. 2021. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*. 756–771.
- [25] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPC can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 1–16.
- [26] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2014. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 295–306.
- [27] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. FaSST: fast, Scalable and Simple Distributed Transactions with Two-Sided RDMA Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 185–201.
- [28] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. 2020. 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*. 514–528.
- [29] Sheng Li and et al. 2016. Full-stack architecting to achieve a billion-requests-per-second throughput on a single key-value store server platform. *ACM Transactions on Computer Systems (TOCS)* 34, 2 (2016), 1–30.
- [30] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *Proceedings of the ACM special interest group on data communication*. 44–58.
- [31] Jiaxin Lin, Adney Cardoza, Tarannum Khan, Yeonju Ro, Brent E Stephens, Hassan Wassel, and Aditya Akella. 2023. {RingLeader}: Efficiently Offloading {Intra-Server} Orchestration to {NICs}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1293–1308.
- [32] Jiaxin Lin, Kiran Patel, Brent E Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. {PANIC}: A {High-Performance} programmable {NIC} for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 243–259.
- [33] Xiaofeng Lin, Yu Chen, Xiaodong Li, Junjie Mao, Jiaquan He, Wei Xu, and Yuanchun Shi. 2016. Scalable kernel tcp design and implementation for short-lived connections. *ACM SIGARCH Computer Architecture News* 44, 2 (2016), 339–352.
- [34] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*. 318–333.
- [35] Fangming Lu, Xingda Wei, Zhuobin Huang, Rong Chen, Minyu Wu, and Haibo Chen. 2024. Serialization/Deserialization-free State Transfer in Serverless Workflows. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 132–147.
- [36] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 537–550.
- [37] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W Moore. 2018. Understanding PCIe performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 327–341.
- [38] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarmo Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. 2015. The design and implementation of open {vSwitch}. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 117–130.
- [39] Solal Pirelli and George Candea. 2020. A Simpler and Faster NIC Driver Model for Network Functions. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 225–241.
- [40] Boris Pismenny, Haggai Eran, Aviad Yehezkel, Liran Liss, Adam Morrison, and Dan Tsafir. 2021. Autonomous NIC offloads. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 18–35.
- [41] Boris Pismenny, Adam Morrison, and Dan Tsafir. 2023. ShRing: Networking with Shared Receive Rings. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 949–968.
- [42] Moinuddin K Qureshi, Aamer Jaleel, Yale N Patt, Simon C Steely, and Joel Emer. 2007. Adaptive insertion policies for high performance caching. *ACM SIGARCH Computer Architecture News* 35, 2 (2007), 381–391.
- [43] Hugo Sadok, Nirav Atre, Zhipeng Zhao, Daniel S Berger, James C Hoe, Aurojit Panda, Justine Sherry, and Ren Wang. 2023. Ensō: A Streaming Interface for NIC-Application Communication. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 1005–1025.

- [44] Dan Tang, Yungang Bao, Weiwu Hu, and Mingyu Chen. 2010. DMA cache: Using on-chip storage to architecturally separate I/O data from CPU data for improving I/O performance. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 1–12.
- [45] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. 2018. {ResQ}: Enabling {SLOs} in Network Function Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 283–297.
- [46] Midhul Vuppapapati, Saksham Agarwal, Henry Schuh, Baris Kasikci, Arvind Krishnamurthy, and Rachit Agarwal. 2024. Understanding the host network. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 581–594.
- [47] Minhu Wang, Mingwei Xu, and Jianping Wu. 2022. Understanding I/O direct cache access performance for end host networking. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 1 (2022), 1–37.
- [48] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchun Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. 2023. SRNIC: A scalable architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1–14.
- [49] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. 2015. Fast in-memory transaction processing using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 87–104.
- [50] Li Wenxue, Zhang Junyi, Liu Yufei, Zeng Gaoxiong, Wang Zilong, Zeng Chaoliang, Zhou Pengpeng, Qiaoling Wang, and Kai Chen. 2024. Cepheus: Accelerating Datacenter Applications with High-Performance RoCE-Capable Multicast. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2024, Edinburgh, United Kingdom, March 2–6, 2024*. IEEE, 908–921.
- [51] Yifan Yuan, Mohammad Alian, Yipeng Wang, Ren Wang, Ilia Kurakin, Charlie Tai, and Nam Sung Kim. 2021. Don't forget the I/O when allocating your LLC. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 112–125.
- [52] Yifan Yuan, Mohammad Alian, Yipeng Wang, Ren Wang, Ilia Kurakin, Charlie Tai, and Nam Sung Kim. 2021. Don't forget the I/O when allocating your LLC. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 112–125.
- [53] Yiran Zhang, Qingkai Meng, Chaolei Hu, and Fengyuan Ren. 2024. Revisiting congestion control for lossless ethernet. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 131–148.
- [54] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.