Revisiting Congestion Control for Lossless Ethernet

Yiran Zhang, *Tsinghua University and Beijing University of Posts and Telecommunications;*
Qingkai Meng, *Tsinghua University and Beihang University;*
Chaolei Hu and Fengyuan Ren, *Tsinghua University*

https://www.usenix.org/conference/nsdi24/presentation/zhang-yiran

This paper is included in the
Proceedings of the 21st USENIX Symposium on
Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Revisiting Congestion Control for Lossless Ethernet

Yiran Zhang[1,2], Qingkai Meng[1,3], Chaolei Hu[1], Fengyuan Ren[1]
[1]*Tsinghua University,* [2]*Beijing University of Posts and Telecommunication,* [3]*Beihang University*

Abstract

Congestion control is a key enabler for lossless Ethernet at scale. In this paper, we revisit this classic topic from a new perspective, i.e., understanding and exploiting the intrinsic properties of the underlying lossless network. We experimentally and analytically find that the intrinsic properties of lossless networks, such as *packet conservation*, can indeed provide valuable implications in estimating pipe capacity and the precise number of excessive packets. Besides, we derive principles on how to treat congested flows and victim flows individually to handle HoL blocking efficiently. Then, we propose ACK-driven congestion control (ACC) for lossless Ethernet, which simply resorts to the knowledge of ACK time series to exert a temporary halt to exactly drain out excessive packets of congested flows and then match its rate to pipe capacity. Testbed and large-scale simulations demonstrate that ACC ameliorates fundamental issues in lossless Ethernet (e.g., congestion spreading, HoL blocking, and deadlock) and achieves excellent low latency and high throughput performance. For instance, compared with existing schemes, ACC improves the average and 99th percentile FCT performance of small flows by 1.3~3.3× and 1.4~11.5×, respectively.

1 Introduction

As the adoption of RDMA continues to grow in Ethernet-based data centers, there is a wave of the deployment of lossless networks. A lossless network can harness the full potential of RDMA and benefit application latency performance that used to be affected by packet loss [6, 15, 17].

Lossless Ethernet employs hop-by-hop flow control, i.e., Priority-based Flow Control (PFC) [1], to ensure that packets are not dropped due to buffer overflow. However, when persistent congestion occurs, PFC may impose a backpressure effect on upstream ports, and the cascade reaction can even spread to remote switches. Flows not destined to the congestion point are also paused, which is well-known as the Head-of-Line (HoL) blocking issue [42, 49]. The frequent trigger of PFC may also be along with deadlock and unfairness issues [21, 25, 26, 34, 42, 49]. The root cause of the above side effects is that PFC itself can not allocate appropriate bandwidth for each flow, so network congestion can not be eliminated but only spreads. As a result, end-to-end congestion control becomes a key enabler for high-performance lossless Ethernet at scale [21, 49].

Many efforts have been devoted to developing congestion control to facilitate the deployment of lossless Ethernet. DCQCN [49] employs widely-used ECN in switches to detect congestion and heuristically throttles congested flows. Nevertheless, ECN solely based on queue length may not provide the correct congestion indicator once PFC takes effect [45]. HPCC [29] relies on high-precision INT [2] to guide congestion control but incurs high per-packet overhead and sacrifices throughput performance. TIMELY [33] advocates RTT-based congestion control. Still, the RTT samples may mislead congestion judgment when RTT suddenly increases owing to HoL blocking. Recently, TCD [45] makes a new attempt to accurately detect congested flows but still follows the heuristic rate control rules as traditional congestion controls, which makes it essentially hard to achieve low latency and high throughput simultaneously.

Reflecting on these efforts, we notice that instead of designing congestion control for lossless Ethernet from scratch, almost all existing works weave unfitted pieces into their schemes thus resulting in sub-optimal performance. In particular, congestion detection is the foundation while rate control is indeed the core of achieving appropriate bandwidth allocation for each flow. Both of them should consider essential features of lossless Ethernet and work harmoniously together to realize high-performance congestion control. Therefore, we ask this question: *can we take a step back and rethink congestion control for lossless Ethernet by taking full advantage of its intrinsic properties?*

The crux to understanding the intrinsic properties of lossless Ethernet is to recognize the profound impact of hop-by-hop flow control (i.e., PFC). For instance, the ON-OFF regulation of hop-by-hop flow control introduces a new ON-

OFF style port state in switches, changing the foundation of congestion detection. More importantly, no packet dropping at intermediate switches in turn enables an end-to-end lossless path, which alters the view of the network pipe and opens new opportunities for flow-level bandwidth allocation. In fact, previous efforts unconsciously neglect one intrinsic yet powerful property, i.e., **packet conservation**: packets are not lost in the lossless network[1]. All injected packets are eventually delivered to receivers and acknowledged by ACKs, before which they fly in the network pipe or queue at switches.

In this paper, based on the understanding of above intrinsic properties and in-depth experimental investigations, we derive two key principles to build a desirable congestion control for lossless Ethernet: (1) Since packet conservation property implies that the number of ACKs equals to the number of packets injected into the network, the ACK-driven paradigm should gain renewed emphasis in lossless Ethernet to infer network pipe capacity and the exact number of excessive packets (§ 3.2). (2) To handle HoL blocking efficiently, congested flows should be suppressed sufficiently to eliminate accumulated buffers as soon as possible; victim flows should adapt to the severity of congestion to balance HoL blocking alleviation and throughput performance (§ 3.3).

Armed with the above principles, we propose a new congestion control for lossless Ethernet called ACC (ACK-Driven Congestion Control). ACC utilizes TCD [45] to accurately detect congested flows and victim flows. For congested flows, ACC gracefully employs a two-step strategy: senders first wait for the excessive packets to drain out via a temporary halt and then match the rate to network pipe capacity. In detail, by correlating *ACK sequences* across multiple periods and *ACK arrival rate*, ACC senders figure out the number of excessive packets and the exact time to be drained at switches. For victim flows, ACC senders collect the duration pattern of *ACK markings* to perceive the severity of congestion and only adjust the rate when congestion spreading lasts long.

Our key contributions are summarized as follows:

• Understanding the implications of intrinsic properties of lossless Ethernet (e.g., packet conservation) on congestion control and deriving basic principles for handling congested flows and victim flows. Our perspective strikes out a new path of exploiting intrinsic properties to guide the precise rate control for lossless networks.

• Developing a new congestion control scheme called ACC for lossless Ethernet. As ACK can correspond to each injected packet precisely, ACC utilizes ACK time series to derive exact backlogged packets in switches and network pipe capacity. ACC can quickly converge to the proper rate in one RTT and empty accumulated queues rapidly.

[1]Note that in lossy networks where packets can be dropped due to buffer overflow, the number of packets injected into the network is not equal to the number of ejected packets (i.e., acknowledged by receivers). So packet conservation property does not hold in lossy networks. In addition, we do not consider packet loss due to packet corruption or device failure in this paper.

• Implementing ACC in SoftRoCE [36] and evaluating ACC via extensive experiments. Results show that ACC well alleviates congestion spreading and HoL blocking issues and can achieve low latency and high throughput. For instance, compared with state-of-the-art schemes, ACC improves the average and 99th percentile FCT performance of small flows by 1.3~3.3× and 1.4~11.5×, respectively.

2 Design Space

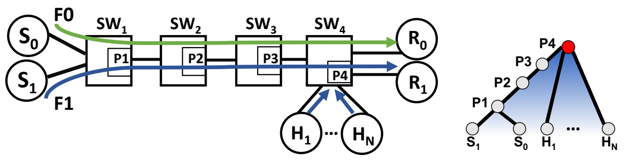
2.1 Desirable Properties

We begin with reexamining the desirable properties of congestion control for lossless Ethernet. Specifically, we target the scenario of a single RDMA domain deployed with a single CC scheme, which is common in Ethernet-based datacenters.

(1) **Fast convergence.** The primary goal of congestion control is allocating the proper rate for each flow thus the aggregation rate converges to the bottleneck capacity. However, in lossless Ethernet, fast convergence is particularly crucial to the following issues:

• *Restricting the spreading of congestion and alleviating HoL blocking.* Without fast convergence, a congestion point is likely to spread into a congestion tree with cascading accumulated queues along branches. Take Figure 1(a) as an example. After congestion happens at port P4, P4 becomes the root of the congestion tree. Once the PFC pause propagates to upstream switches, the paths of congested flows (e.g., <S_1-R_1>) become the main branch of the congestion tree. When other flows (e.g., F0) destined for non-congestion points pass through the main branch, the backpressure of PFC may further induce the secondary branch. Thus, HoL blocking occurs. Flows not destined for congestion points innocently suffer from PFC pause. The accordingly secondary branch in Figure 1(b) is <P1-S_0>. When the congestion tree grows, the input and output of the switches along the branches are prone to be modulated into an ON-OFF pattern by PFC (i.e., alternating between sending and pausing), thus disturbing congestion detection and subsequent rate regulation. If multiple congestion points are in the same congestion tree (i.e., a larger congestion tree covers a smaller congestion tree), congestion point in the larger congestion tree will be first eliminated then the smaller congestion tree.

• *Lowering the risk of deadlock.* Deadlock is a silent killer in the lossless network [21, 25, 26, 38]. The typical causes of deadlock in today's data centers are as follows: Initially, owing to reasons such as routing misconfiguration, switch firmware failure, or link failure, a Cyclic Buffer Dependency (CBD) emerges in the topology. Then if congestion happens and the congestion control fails to converge quickly, the frequent trigger of PFC may be induced. Eventually, PFC pause propagates to the whole cycle and all switches wait for its upstream to send PFC resume, thus a deadlock occurs. From another perspective, even if CBD exists, the fast convergence of congestion control can make it resilient against deadlock.



(a) Topology and flows ($N > 1$) (b) Congestion tree

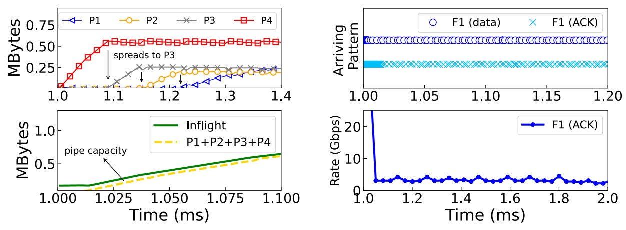
Figure 1: Typical scenario.

(2) **Low latency and high throughput.** Today's applications increasingly pursue stringent latency and throughput performance [23,27]. In lossless Ethernet, offloading network stacks into hardware (i.e., RDMA) helps sustain the critical latency and bandwidth requirements at hosts but leads to the network becoming the bottleneck. On the one hand, for increasingly dominant storage workloads with NVMe-over-Fabrics technology [10,27], the overall completion latency for a single storage operation is determined by the latency of the slowest network operation. Each storage operation involves messages with a size of only several KB and may require microsecond-scale latency [3, 19, 47, 48]. On the other hand, data processing applications [16,44] and AI applications increasingly involve communicating among high computation speed devices at scale [29, 39, 51]. These applications periodically transfer a large volume of data and the average transfer time of each round directly impacts the overall processing/training time and cost, which imposes high requirements on throughput performance. To sum up, congestion control should be able to achieve low latency for short flows and predictable high throughput for longer flows.

2.2 Ternary Flow States in Lossless Ethernet

The foundation of congestion control in lossless Ethernet is understanding the network state space under the influence of PFC. We notice that a recent work TCD [45] develops a ternary congestion signal tailored to lossless networks, which can distinguish between switch ports that are roots of congestion trees (i.e., congestion ports) and switch ports on the branches that are only affected by PFC (i.e., undetermined ports). TCD terms the new state as "undetermined" because the real states of these ports are masked due to intermittent ON-OFF sending pattern when PFC triggers. TCD-enabled switches mark packets passing through a congested port with CE and mark packets passing through an undetermined port with UE, which indicates "congestion encountered" and "undetermined encountered", respectively. Besides, UE can only be marked when the packet is not marked with CE. As a result, TCD can notify end hosts of ***ternary*** flow states: congested flows that pass through congestion ports, undetermined flows [2] that *only* pass through undetermined ports, and uncongested flows. As shown in Figure 1, port P4 is the congestion port, while all other nodes of the congestion tree are undetermined ports. F1 is the congested flow, while F0 is the victim flow.

[2]We interchangeably use "victim flows" and "undetermined flows" throughout the paper.



(a) F1: inflight packets and packets backlogged in switch buffers (b) Packets arriving pattern

Figure 2: [$N = 20$] F1: packets backlogged in switches and data/ACK arriving pattern.

However, although TCD can be useful for perceiving network states, it remains an essential issue on how to deal with these refined flows to fulfill the desirable properties for congestion control. Concretely, for congested flows that are actual contributors to congestion, congestion control should regulate its rate to converge quickly to the proper value and stay near it. By doing so, emerging congestion can be eliminated rapidly and prevents the potential risk of congestion spreading. Still, chances are that congestion spreads (e.g., caused by bursty traffic or first RTT traffic) and thus may accompany the emergence of HoL blocking and victim flows.

3 Principles

In this section, we aim to present design principles to answer key questions for congestion control in lossless Ethernet:

(1) How to adjust the rate of congested flows to achieve fast convergence? (§ 3.2)

(2) How to treat congested flows and victim flows individually to handle HoL blocking? (§ 3.3)

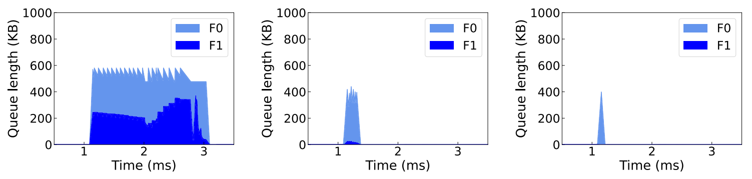
3.1 Experiment Setup

To provide vivid illustration, we conduct detailed investigations via ns-3 simulations. The network topology is shown in Figure 1(a). The link capacity is 100Gbps with 2 μs propagation delay. F0 and F1 are long-lived flows. $H_1 \sim H_N$ send concurrent 64KB bursts lasting for about 2ms and can hardly be regulated by end-to-end congestion control as the size is smaller than bandwidth-delay-product (BDP). For rate decrease algorithms at hosts, we adopt the widely-used DCQCN [49]. The PFC threshold is 512KB.

The switches support TCD so hosts are aware of ternary states of flows. Specifically, assume F0 and F1 start simultaneously and achieve their fair bandwidth allocation, then bursts start. After congestion occurs at port P4, F0 becomes the victim flow in the branch <P4-S_1>. We introduce different congestion degrees by changing the value of N, which indicates the number of concurrent senders. A larger N induces more severe congestion at port P4, resulting in a deeper congestion tree.

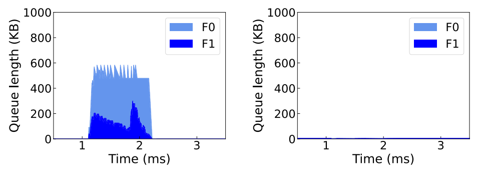
3.2 The Power of ACK-Driven

To explore the potential principle to throttle congested flows, we first take a close look by combining the in-network per-



(a) F1:DCQCN (b) F1 stops 20μs and sends at PC_{rate} (c) F1 stops 180μs and sends at PC_{rate}

Figure 3: [$N = 20$] Queue occupancy at Port P3. PC_{rate}: the rate corresponding to the pipe capacity of the flow.



(a) F1: DCQCN (b) F1 stops 90μs and sends at PC_{rate}

Figure 4: [$N = 10$] Queue occupancy at Port P3.

spective (e.g., congested packets backlogged in switches) and the end-host perspective (e.g., inflight bytes seen by senders).

Figure 2(a) shows the evolution of F1 packets backlogged in each switch port (the upper subfigure) and the inflight bytes maintained at the sender H_1 (the lower subfigure) under Setting I ($N = 20$). The inflight bytes are derived from the difference between the next sending byte sequence and the unacknowledged byte sequence. After congestion occurs at 1ms, F1 packets are only queued at P4 till 1.1ms. As PFC takes effect and congestion gradually spreads to upstream switches, the number of backlogged bytes at ports P3, P2, and P1 rise one by one. For congested flow F1, the maximum amount of traffic that can be delivered per RTT is $C \times RTT/(N+1)$, also called as the network pipe capacity of this flow (around 0.012MB in this scenario). As shown in Figure 2(a), we notice that the number of *inflight bytes* closely tracks the sum of *total backlogged packets* (i.e., packets queued in P4, P3, P2, and P1) and the *network pipe capacity* of F1.

From the end-host perspective, Figure 2(b) depicts the packets arriving pattern of congested flow F1 at R_1 and its ACKs at S_1. After congestion occurs at 1ms, the arriving pattern of F1 data packets is continuous for a long time period at first because excessive packets would eventually go through the continuously-ON congested port P4. The arriving pattern of ACKs corresponds well to data packets. We also record the arriving rate of F1 ACKs. For congested flow F1, the arriving rate of ACKs (around 4.7Gbps) is exactly the available bandwidth of F1, which can also deduce the capacity of network pipe (i.e., available bandwidth $\times$ RTT). In summary, we have ***Observation 1****: For congested flows: (1) the number of excessive packets backlogged in switches is exactly the difference between inflight packets and the network pipe capacity; and (2) the ACK arrival rate can imply the available bandwidth.*

Indeed, the above observations corroborate an intrinsic property of lossless Ethernet: **packet conservation**, i.e., sent packets are never lost. Different from traditional lossy networks where packets may be dropped due to buffer overflow, in lossless Ethernet, excessive packets will backlog in switch buffers along the branch after filling the pipe. Thus all of them are eventually delivered and acknowledged by ACKs.

The implications behind packet conservation property lie in that: the precise number of excessive packets and the network pipe capacity of congested flow can both be inferred from the ACK time series. Thus, the *ACK-driven* paradigm really comes into its own in a lossless network with packet conservation property. To this end, we derive **Principle 1: In lossless Ethernet, the ACK-driven paradigm is a powerful knob to infer the proper throttled rate and the precise number of excessive packets for congested flows.**

3.3 Handling HoL Blocking

Once PFC takes effect, victim flows may emerge and suffer from HoL blocking. The following question is how to treat congested flows and victim flows individually to handle HoL blocking. Ideally, HoL blocking should be eliminated as soon as possible without incurring unnecessary performance loss.

Next, we focus on the effect of rate control strategies of congested flows on HoL blocking alleviation and victim flows under different degrees of congestion (e.g., $N = 20$ and $N = 10$). We enable TCD in switches thus flows with packets marked as UE are identified as victim flows. The default rate control strategy is that only congested flows will be throttled once detected, and victim flows will adjust the sending rate the same as uncongested flows according to DCQCN.

Figure 3(a) reports the queue occupancy of individual flows at the switch port P3 when $N = 20$ with DCQCN. After bursts start at 1ms, congestion emerges at port P4. Then congestion spreads to upstream switches and packets accumulate at ports P3 and P2 (not shown in the figure). The HoL blocking process lasts for about 2ms. We notice that the queue accumulation of F0 accompanies with the queue buildup of F1, which confirms that HoL blocking originates from queue buildup of congested flows along the branch.

Further, to understand how congested flows with precise rate adjustment may impact HoL blocking, we let congested flow F1 first empty the accumulated packets along switches by stopping for a while (after congestion is detected) and then send at the rate corresponding to the network pipe capacity (i.e., PC_{rate}). As shown in Figure 3(b), congestion spreading at P3 is largely alleviated. The queue occupancy of F1 is quickly suppressed and the total blocking time of F0 is reduced. We also estimate an ideal stopping time (i.e., 180us) and Figure 3(c) depicts the queue occupancy. With sufficiently long time to empty the accumulated packets along switches, the duration of HoL blocking is much shorter than in Figure 3(b). However, F0 is still blocked at port P3 and has queue buildup. This is because although congested flow F1 are stopped precisely, pause frames may still be triggered due to large ingress queue previously accumulated at the downstream port [3].

When $N = 10$, we see a more desired impact on alleviating HoL blocking. As shown in Figure 4(a), the congestion degree is smaller when $N = 10$, where the duration time of HoL blocking is short. With DCQCN, the queue occupancy of F0 is even much larger than F1 at port P3. However, as depicted in Figure 4(b), once congested flow F1 stops for a sufficient time to empty the queue and then sends at the rate corresponding to the network pipe capacity, HoL blocking can even be eliminated. With fewer accumulated packets and a faster draining rate at the downstream congested port, the corresponding ingress queue falls below the PFC threshold faster. In this scenario, victim flows can safely pass without causing pause frames and HoL blocking. In summary, we have ***Observation 2***: *Stopping congested flows sufficiently long can eliminate associated buffers as soon as possible. However, under different congestion situations, HoL blocking may still occur and victim flows have the risk of inducing further congestion spreading.*

The above observations tell that although stopping congested flows does not necessarily stop pause frames from affecting victim flows, it does help get rid of such blocking states as soon as the accumulated buffer of congested flow is sufficiently drained. During this process, victim flows may still emerge and have the risk of congestion spreading. Thus, we advocate treating victim flows dynamically to adapt to different congestion situations, rather than treat them identically. We derive **Principle 2: Stopping congested flows sufficiently long is the foremost means to suppressing HoL blocking. While congested flows are stopping, victim flows should balance HoL blocking alleviation and throughput by adapting to the severity of congestion.**

Incorporating **Principle 1** and **Principle 2**, we come to the following strategies for congested flows and victim flows in lossless Ethernet: Congested flows should first stop to wait for accumulated congested queues to drain out, and then send at the rate of network pipe capacity. While victim flows should sacrifice throughput as little as possible and benefit HoL blocking alleviation. Such collaborative strategies can rapidly suppress congestion spreading without link underutilization, and benefit latency and throughput performance.

4 ACK-Driven Congestion Control

Following the above principles, we present the design of ACK-driven congestion control (ACC). The switch supports TCD and marks ternary congestion notification in data packets. A CE marked packet indicates a congested flow that passes through a congestion port. A UE marked packet indicates that the flow only passes through ports affected by PFC. The flow is uncongested if neither CE nor UE is marked (denoted by NO). The receiver copies the TCD marking to the corresponding ACK and sends it back to the sender.

[3]PFC is triggered based on the ingress queue length. In today's commodity shared-buffer switches, the ingress queue length is a counter which is updated when packets enqueue the ingress and dequeue the egress [21].

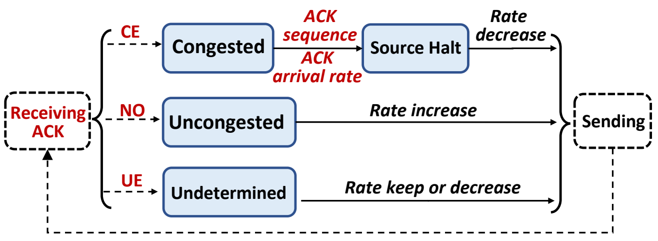


Figure 5: ACC state machine.

At its core, ACC conducts ACK-driven rate adjustment including enforcing a source halt according to ACK sequences (§ 4.2), and referring to ACK arrival rate to guide rate decrease for congested flows (§ 4.3). The source halt state involves a halt time which guarantees draining out accumulated packets in the network while avoiding link under-utilization. For victim flows, ACC adaptively adjusts the rate according to the feature of duration time of ACK markings (§ 4.4).

4.1 State Machine Overview

The ACC sender makes rate adjustment decisions every period T (e.g., base RTT, see § 5). At the end of each period T, the sender identifies the current state of the flow according to TCD marking and conducts corresponding rate adjustment.

Figure 5 illustrates the state machine of ACK-driven rate adjustment at ACC senders. TCD marking is aggregated in each T with the priority order CE > UE > NO. If there is ACK with CE received, the flow may experience congestion. The ACC sender records the number of CE marked ACKs of each flow. If the marking fraction of CE is above a threshold (i.e., 90%) during a period T, it is considered a steadily congested flow. If there is no ACK with CE received but with UE marking received, the flow is regarded as a victim flow that experiences an undetermined state. Only when neither UE nor CE marking is received, the flow is uncongested. Specifically, when the flow is identified as congested, it will first enter into a source halt state where the connection ceases transmission and the ACK arrival rate is recorded for later sending. After leaving the source halt state, congested flows are throttled at the proper rate derived from the ACK arrival rate. For uncongested flows, the sender directly increases the sending rate. For victim flows, the sender adjusts the sending rate according to the duration pattern of UE markings.

4.2 Halting Congested Flows

For congested flows, ACC introduces a source halt state which actively stops the transmission of congested flows before sending at the appropriate rate (§ 3.2 **Principle 2**). The crux of the source halt is the halt time. An appropriate halt time should ensure that accumulated packets are drained without under-utilizing the link. Indeed, a proper number of packets should fill the network pipe without causing queue buildup in the switch buffer. Specifically, ACC senders calculate the

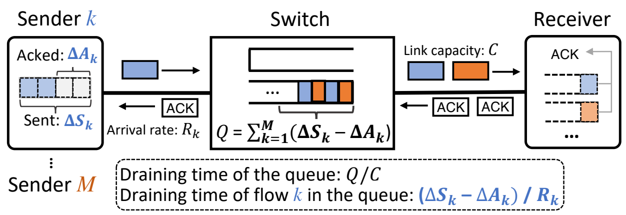

Sender k
Acked: ΔA_k
Sent: ΔS_k
Sender M
Switch
$Q = \sum_{k=1}^{M}(\Delta S_k - \Delta A_k)$
ACK
Arrival rate: R_k
Link capacity: C
Receiver
ACK
Draining time of the queue: Q/C
Draining time of flow k in the queue: $(\Delta S_k - \Delta A_k) / R_k$

Figure 6: The rationale of calculating t_{Halt}.

duration of the source halt state according to the time series of ACK sequences (§ 3.2 **Principle 1**).

Concretely, for each flow, the sender maintains a value called $t_{NextPkt}$ to represent when the next packet is transmitted. Initially, $t_{NextPkt}$ is set to 0 and is assigned by the congestion control algorithm. With ACC, $t_{NextPkt}$ is updated as follows when the sender enters into the source halt state:

$$t_{NextPkt} \leftarrow t_{NextPkt} + t_{Halt} \quad (1)$$

where t_{Halt} is calculated based on the estimated draining time. After t_{Halt}, a sender considers that the accumulated queues have drained out. Figure 6 illustrates the rationale under the typical incast scenario with M senders. Assume for flow k, ΔS_k is the total sent out bytes in one period T (i.e., base RTT). ΔA_k is the total acknowledged bytes in the next period. Ideally, all packets sent in one period should be acknowledged in the next period when there is no congestion (i.e., $\Delta A_k = \Delta S_k$). Once congestion happens, the excessive packets build up the queue. The total excessive sent packets are $\sum_{k=1}^{M}(\Delta S_k - \Delta A_k)$. Then the draining time of the bottleneck queue is $\sum_{k=1}^{M}(\Delta S_k - \Delta A_k)/C$, where C is link capacity. Since flow k can only pass through the bottleneck with the rate R_k, the draining time of total packets belonging to flow k in the bottleneck queue can be calculated as $(\Delta S_k - \Delta A_k)/R_k$, which is the expected t_{Halt} of flow k. For clarity, in the following, we use ΔS and ΔA to denote ΔS_k and ΔA_k for flow k, respectively.

Each flow maintains a sequence space to calculate ΔS and ΔA. Assume *snd_una* denotes the first sequence that has been sent but not acknowledged, and *snd_nxt* denotes the next sequence to be sent. *snd_una* is updated every time an ACK is received. When a packet is sent, *snd_nxt* is also updated. Then ΔS is the difference between *snd_nxt* at the end of the current period and *snd_nxt* at the beginning of the current period. ΔA is the difference between *snd_una* at the end of the next period and *snd_una* at the beginning of the next period. When severe congestion causes no ACK to arrive during a period, ΔS should accumulate until an ACK is received.

In detail, ACC senders manipulate ΔS and ΔA among several periods to calculate t_{Halt} and exert source halt as shown in Figure 7: (1) Period T_{i-1} in normal transmission; (2) Period T_i starts receiving ACKs of packets sent in T_{i-1} and sends at the same rate as T_{i-1}; (3) The first period T_{i+1} enters the source halt state and stops transmission; (4) Period T_{i+n} leaves the source halt state and starts transmission. Assume the sender starts a flow and then experiences congestion during T_{i-1}. The available bandwidth is derived from ACK arrival rate, and

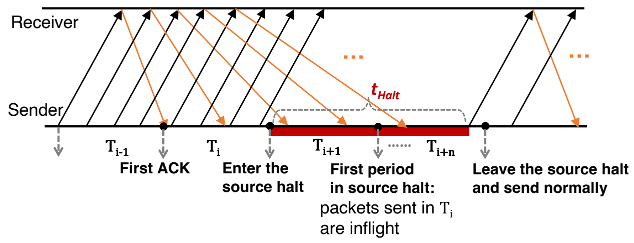

Receiver
Sender
...
...
t_{Halt}
T_{i-1}
First ACK
T_i
Enter the source halt
T_{i+1}
First period in source halt: packets sent in T_i are inflight
T_{i+n}
Leave the source halt and send normally

Figure 7: The timeline of entering and exiting source halt.

T_{i-1} period directly uses ACK sequences received in T_i period to obtain the excessive packets in T_{i-1}. Indeed only T_i period will use the excessive rate to infer the excessive packets. The detailed process of exerting source halt is as follows:

• At the end of T_i, the sender identifies the flow as congested and prepares to exert source halt. t_{Halt} is calculated by $(\Delta S - \Delta A)/R$ and the flow halts until over period T_{i+1}. ΔS is obtained during T_{i-1}, while ΔA and R is obtained in T_i. Note that the flow still sends packets at the previous rate during T_i, where excessive packets sent in T_i may also lead to queue buildup.

• At the end of T_{i+1}, the sender already enters the source halt state and stops transmission. The sender should estimate excessive packets sent in T_i and extend the insufficient halt time calculated at the end of T_i. We notice that the excessive packets sent in T_i can be inferred according to how much the sending rate exceeds the available bandwidth, without the need to wait for the arrival of ACKs. Concretely, based on the sending rate R_s in the period before source halt (i.e., sending rate of T_{i-1}) and the current available bandwidth R (i.e., the ACK arrival rate in T_i), the number of excessive sent packets r in T_i equals to ΔS subtracting the estimated pipe capacity (line 7 in Algorithm 1). Finally, t_{Halt} is extended by t_{extend} (line 8-9 in Algorithm 1).

• During T_{i+n}, the sender leaves the source halt state and starts transmission. The sender should only consider ACKs of new packets sent after leaving the source halt state for following ACK-driven rate adjustment. To this end, the sender records *snd_nxt* before entering into the source halt state. For each received ACK, only when the current *snd_una* is greater than the recorded *snd_nxt*, the ACK is considered eligible (line 2 in Algorithm 1).

4.3 Throttling Congested Flows

For congested flows, after leaving the source halt state, the sender utilizes the ACK arrival rate to guide rate decrease (§ 3.2 **Principle 1**). ACK arrival rate (line 3 in Algorithm 1) can reflect the data receiving rate at the receiver [4]. The aggregate receiving rate is the capacity of the bottleneck link. To reduce the disturbance due to congestion in the reverse path, ACKs are sent with higher priority than data packets to avoid a significant queueing delay. The effectiveness of this method has been confirmed in [33]. Besides, the arrival

[4]Currently, we only consider per-packet ACKs for accurately estimating excessive packets and network pipe capacity. However, if there is ACK coalescing, the precise number of excessive packets can still be inferred from cumulative ACK sequences.

Variables	Description
snd_una	The current unacknowledged byte
snd_nxt_start	The next sending byte when the sender is allowed to send, initialized as 0
halt	Whether the sender has halted
R_s	The current sending rate
RI	The rate increment value
$t_{NextPkt}$	The next sending time
N_u	The number of consecutive uncongested periods
N_v	The number of consecutive periods receiving UE
P_{thresh}	The threshold of N_v to throttle victim flows

Table 1: Variables of ACC sender algorithm.

rate of ACKs may be so small that just one ACK arrives in several periods. To address this corner case, the sender also records the inter-arrival time of ACKs. When the inter-arrival time is larger than T, the sender estimates the arrival rate by replacing T with the inter-arrival time.

4.4 Adapting Undetermined Flows

If ACKs marked with UE are received in a period T, the corresponding flow is an undetermined flow (i.e., victim flow). ACC treats victim flows adaptively according to the severity of congestion (§ 3.3 **Principle 2**). Indeed, the duration of receiving UE marks can indicate the severity of congestion. If congestion is not severe, ACC senders will receive few UE marks (e.g., only within one period) as congestion may not spread or can be quickly suppressed due to source halt of congested flows. However, if congestion is severe, congestion spreading may last long and ACC senders are likely to observe UE marks stretching across multiple periods. Thus, after identifying victim flows, ACC senders continuously observe the arriving pattern of UE marks to adjust the sending rate.

Concretely, ACC senders first keep the rate and then reduce the sending rate according to the number of consecutive periods with UE marks. As shown in Algorithm 1 (lines 23-24), if the number of consecutive periods with UE marks exceeds a period threshold P_{thresh}, victim flows will decrease the current rate by half to reduce the injection rate and prevent further congestion spreading. In this way, victim flows avoids blindly decreasing the rate with the side effect of losing throughput, but help alleviate HoL blocking when congestion spreading lasts long.

4.5 Rate Increase for Uncongested Flows

If neither CE nor UE marked ACKs are received during period T, the flow is uncongested and should attempt to increase the sending rate. The principles are as follows:

(1) The increase step of uncongested flows should consider the current sending rate. A flow with a small sending rate should increase more, while a flow with a large sending rate should increase less.

(2) The rate increase process should be gradual at first to avoid causing congestion immediately while aggressive after available bandwidth is considered adequate.

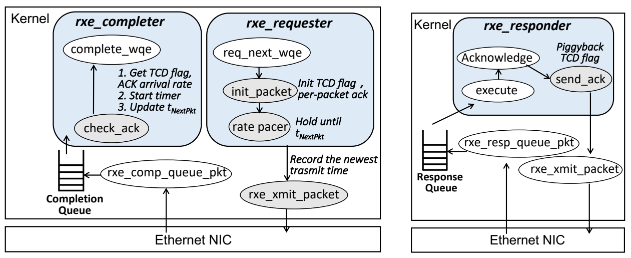
Algorithm 1 ACC sender algorithm.

```
1:  function CONGESTED($recvNum, snd\_nxt\_start, snd\_una$)
2:      if $halt == false$ and $snd\_una > snd\_nxt\_start$ then
3:          $R = recvNum * MTU/T$; //calculating ACK arrival rate
4:          $R_s = R$;
5:          $SourceHalt(R, \Delta S, \Delta A)$; //calculating the halt time
6:      else if $Halt == true$ then
7:          $r = \Delta S - \Delta S * R/R_s$;
8:          $t_{extend} = r/R$;
9:          $t_{Halt} \leftarrow t_{Halt} + t_{extend}$;
10:         $t_{NextPkt} \leftarrow t_{NextPkt} + t_{extend}$;
11:     end if
12: end function
13: function UNCONGESTED
14:     if $N_u < 2$ then
15:         $RI = RI_{Low}$;
16:     else
17:         $RI = RI_{High}$;
18:     end if
19:     $R_s = R_s + \frac{(lineRate - R_s)}{lineRate} * RI$;//increasing the sending rate
20:     $N_u = N_u + 1$;
21: end function
22: function UNDETERMINED
23:     if $N_v > P_{thresh}$ then
24:         $R_s = R_s/2$;
25:     end if
26: end function
27: function SOURCEHALT($R, \Delta S, \Delta A$)
28:     if $\Delta S > \Delta A$ then
29:         $t_{Halt} = (\Delta S - \Delta A)/R$;
30:         $t_{NextPkt} \leftarrow t_{NextPkt} + t_{Halt}$;
31:         $halt = true$;
32:         $snd\_nxt\_start = snd\_nxt$;
33:     end if
34: end function
```

ACC senders make rate adjustments following the law in lines 13-21 in Algorithm 1. Generally, the sender increases the sending rate by adding an amount proportional to the difference between $lineRate$ and the current sending rate. Note that $lineRate$ means the maximum rate of NIC which may vary if NIC speed changes. There are two stages for the rate increase. RI is the maximum increase amount per period with different values for two stages and is proportional to $lineRate$. For the early stage, RI is set as a small value RI_{Low}. After two consecutive increase periods, the maximum increase amount per period RI is set as a large value RI_{High}. RI_{Low} and RI_{High} are both proportional to RI. In this way, uncongested flows increase gradually at first, then aggressively with a large increase step and finally increase less per period as the sending rate gets closer to the line rate.

4.6 Theoretical Analysis

We build a fluid model of ACC and analyze its performance, including convergence and fairness. The main conclusions are summarized in the following propositions, and the detailed proof is listed in Appendix A.



(a) Sending data and receiving ACKs (b) Receiving data and sending ACKs

Figure 8: ACC implementation in SoftRoCE.

Proposition 1: When the aggregate sending rate is larger than the bottleneck link bandwidth, ACC can converge the aggregate sending rate towards the bottleneck link bandwidth within one control period T.

Proposition 2: ACC can always guarantee fair bandwidth allocation regardless of the initial sending rate of the flow, i.e.,

$$S_i \to \frac{C}{N}$$

S_i is the sending rate of flow i, C is the bottleneck link bandwidth, N is the number of flows sharing the bottleneck link.

5 Implementation

In this section, we describe the Linux implementation of ACC using SoftRoCE, which is a software implementation of RDMA and fulfills the RoCE NIC function on Ethernet NICs. Figure 8 illustrates the kernel SoftRoCE architecture with our ACC extensions shaded in gray. ACC involves modifications in three main modules, as described below.

`rxe_completer` This module is responsible for processing received ACKs at the sender. We modify the `check_ack` function to implement the ACC algorithm and maintain the state machine of ACC. For the first ACK of each flow, we start a timer for periodically checking ACKs in ACC. The ACC algorithm determines the sending rate of each flow according to current state (congested, undetermined or uncongested) and converts the rate to pacing delay among packets. Based on the latest sending time and pacing delay, the function can calculate $t_{NextPkt}$. $t_{NextPkt}$ is also updated once the flow enters the source halt state (Equation 1).

`rxe_requester` This module is responsible for sending data at the sender. Since the original SoftRoCE does not support congestion control, we add a rate pacer to controlling the transmission time of packets according to $t_{NextPkt}$. The rate pacer continues comparing current time with $t_{NextPkt}$, and source halt can be enforced by holding back the next packet until $t_{NextPkt}$.

`rxe_responder` This module receives data packets and triggers the generation of ACKs. For ACC, after processing the data packets and prepare to generate ACK, it will piggyback the TCD marking to ACK packets.

Discussion on NIC implementation. In RDMA NICs, the QP Context (QPC) maintains for a QP all its contexts, including the DMA states and connection states (e.g., expected packet sequence numbers and sending rate). For ACC, the additional QPC size required for each QP is less than 40B (including ternary states and variables in Table 1), so the total required size for 1K QPs can be less than 40KB, which is acceptable for on-chip memory consumption. Besides, current commercial RDMA NICs already support around 1K rate limiters based on timers [43].

6 Evaluation

6.1 Evaluation Setup

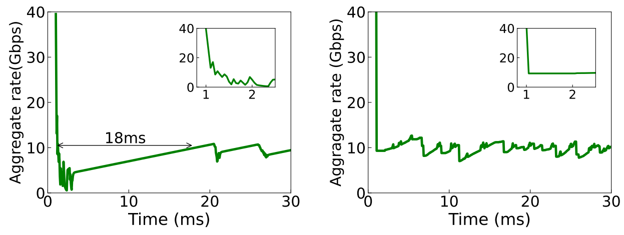
Testbed. We build a testbed consisting of 5 servers connected to one switch. Each server is equipped with AMD Ryzen 9 3950X CPU@3.5GHz, 64GB RAM, an Intel 82599ES 10GbE NIC and runs Ubuntu 20.04 with Linux kernel 5.4.127. All hosts are connected via a 32×100Gbps Tofino switch. We use the default shared buffer setting in the Tofino switch, and also implement ACC and DCQCN [49] in SoftRoCE [36].

Schemes compared. We focus on the comparison among ACC, ECN-based DCQCN [49], RTT-based TIMELY [33] and INT-based HPCC [29]. We use the open-source code of DCQCN, TIMELY and HPCC provided in the HPCC simulator [28] and implement ACC (including the support of TCD in switches). HPCC and DCQCN are state-of-the-art and state-of-the-practice in data centers, respectively. Note that HPCC requires INT support from switches to obtain precise link load. In large-scale simulations, we also compare ACC with DCQCN+TCD and TIMELY+TCD [45], which are congestion control schemes enhanced by accurate congestion detection in TCD. TCD preliminarily proposes to reduce the rate of congested flows aggressively and adjust victim flows gently via heuristically modifying the parameters of existing algorithms.

Network topology. We adopt a fat-tree [5] topology in large-scale simulations. There are 320 servers in 20 racks, 20 aggregation switches and 16 core switches. Each server has a 100 Gbps link connected to ToR switches. All links between core, aggregation and ToR switches are 400 Gbps. Each link has $1\mu s$ delay. The switch buffer size is set to 32MB according to real device configurations. PFC is enabled by default and X_{OFF} is set to 512KB.

Workloads. We generate flows according to *Web Search* [6] and *Cache Follower* [40] workloads. The overall load is 80%. We also use *Hadoop* [40] workload (load 50%) adding incast traffic (load 20%). For each incast, 64 randomly selected senders send 20KB to one receiver. These workloads contain typical traffic patterns in data centers with most traffic constituted by few but large flows [6, 11]. *Cache Follower* and *Web Search* workloads contain more heavy flows. *Hadoop* workload is the lightest, around 70% flows smaller than 10KB.

Parameter setting. The core parameters of ACC include the period T, RI (i.e., RI_{Low} and RI_{High}) and P_{thresh}. Since ACC relies on the difference between sent bytes and acknowledged bytes in a period T to figure out excessive data, T must be



(a) DCQCN (b) ACC

Figure 9: Testbed experiment.

the base RTT to ensure that ACKs can arrive during the next period when there is no congestion. RI_{Low} and RI_{High} are the maximum increase amount in each period. As the aggressive source halt state would directly cease transmission, RI_{Low} and RI_{High} can be set large to rapidly occupy available bandwidth. We recommend setting RI_{Low} as 1/250 of the line rate while $RI_{High} = 10RI_{Low}$. For example, RI_{Low} is 400Mbps and RI_{High} is 4Gbps when link capacity is 100Gbps. We recommend setting the queue length threshold in TCD as a small value, i.e., 2MTU. A small but non-zero threshold ensures that congestion can be sensed at the onset while not too sensitive to transient jitter. By default, we set $RI_{High} = 4Gbps$, $RI_{Low} = 400Mbps$ and $P_{thresh} = 1$. The experiments also indicate that our default parameters are proper (Appendix B).

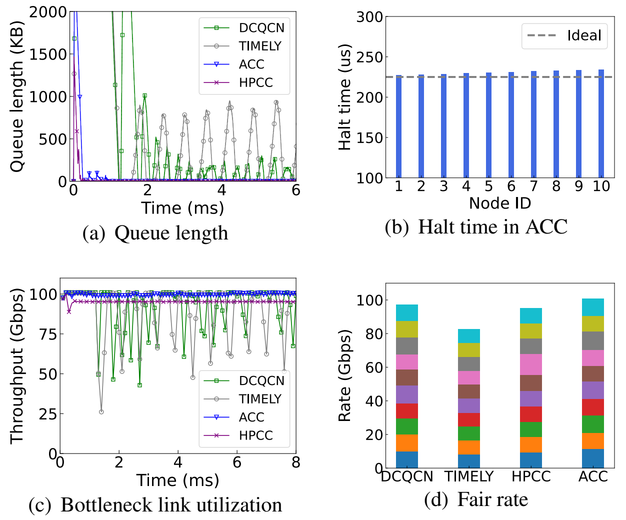
For congestion detection at switches, TCD relies on $max(T_{on})$ to determine the maximum duration of a continuous ON period to detect the ternary state transitions. According to the equation in [45], $max(T_{on})$ is set to 24μs in our large-scale simulations. For HPCC, we use the default parameter with $\eta = 0.95$ and $maxStage = 5$. For DCQCN, we set $K_{min} = 5KB$ and $K_{max} = 200KB$ following the parameter setting in [31,49]. For TIMELY, $\alpha = 0.875$ and $\beta = 0.8$, as suggested in [50].

6.2 Testbed

We implement ACC and DCQCN in SoftRoCE referring to [49]. In our testbed, RTT is around 20μs. Since the period of generating CNPs in DCQCN is 50μs, we set the period T in ACC also to 50μs. The congestion detection results of TCD are equal with ECN in this incast scenario. We configure SoftRoCE such that ACK is generated for every packet[5].

We let each of the four servers start a long flow and record the aggregated sending rates calculated by the congestion control algorithm. Note that in ACC, the sender maintains a sending rate driven by ACK arrival rate and the rate increase algorithm. As shown in Figure 9(a) and Figure 9(b), ACC outperforms DCQCN in terms of fast convergence and stability. After the congestion occurs, DCQCN gradually reduces the sending rate. Due to the heuristic rate regulation, it takes about 0.5ms (~25RTT) to reduce to 10Gbps, and then the aggregation sending rate is reduced to 1Gbps. Finally, it takes around 18ms to converge to the fair share. By contrast, ACC can directly adjust the rate to the proper rate (i.e., 10Gbps) after one period. The subgraph in each figure shows the aggregate sending rate just after the congestion occurs. The rate determined by the ACC algorithm is maintained as 10Gbps under the guidance of the ACK arrival rate. The actual sending is halted because the flows firstly enter the source halt.

[5]We set MTU to 4000B due to the performance issue of SoftRoCE.



(a) Queue length (b) Halt time in ACC

(c) Bottleneck link utilization (d) Fair rate

Figure 10: Convergence, link utilization and fairness.

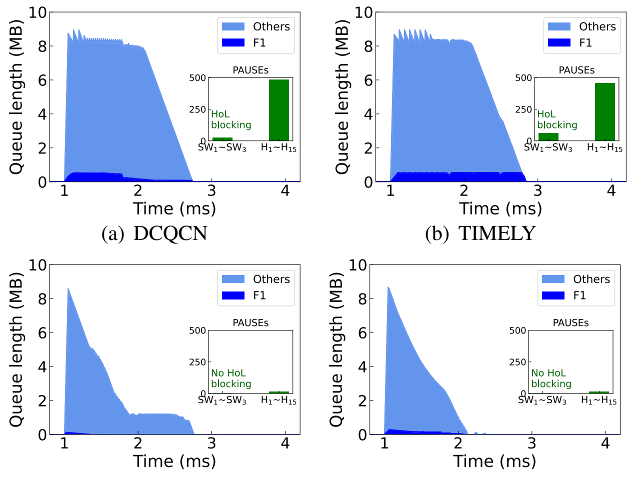
6.3 Microbenchmarks

We first conduct fine-grained simulations to evaluate the basic performance of ACC (e.g., convergence, link utilization and fairness). Then we focus on its ability in dealing with typical issues in lossless Ethernet such as congestion spreading, HoL blocking and deadlock.

ACC can quickly eliminate congestion, maintain near full link utilization and attain fair rate allocation: We let ten source nodes send long-lived traffic to a single destination node through a switch. All links are 100Gbps. Figure 10(a) shows the bottleneck queue length evolution. ACC quickly suppresses the deep queue. Thus congestion is eliminated rapidly. This benefits from the source halt state based on precise excessive packet information in ACC. As depicted in Figure 10(b), the actual halt time of each flow after entering the source halt state is close to the ideal halt time (i.e., the maximum bottleneck queue length divided by link bandwidth), but DCQCN and TIMELY fail to rapidly eliminate the congestion and maintain a low-standing queue.

As for link utilization, both DCQCN and TIMELY incur unstable utilization and underutilize the link due to step-by-step rate adjustment rules. Since HPCC trades high throughput for low latency, it maintains a 95% link utilization. Overall, ACC can achieve steady and near-full link utilization.

As illustrated in Figure 10(d), each flow attains the expected average fair rate (i.e., 10Gbps) with ACC. DCQCN, TIMELY, and HPCC also achieves reasonable fairness. The aggregate flow rate attained in DCQCN, TIMELY, and HPCC are lower than expected. The reasons are that HPCC reserves 5% link bandwidth headroom and the heuristic rate adjustment in



(a) DCQCN (b) TIMELY

(c) HPCC (d) ACC

Figure 11: Buffer occupancy at the congested port P4/SW$_4$ and the number of received PAUSEs.

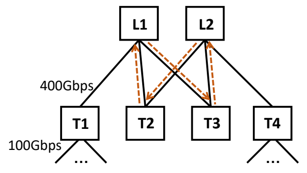


Figure 12: Topology

Scheme	Fraction
DCQCN	6%
TIMELY	74%
HPCC	0%
ACC	0%

Table 2: Fraction of deadlock

DCQCN and TIMELY.

ACC can effectively suppress congestion spreading and alleviate HoL blocking under challenging bursty traffic: We adopt the topology in Figure 1 to further evaluate the ability to deal with congestion spreading and HoL blocking under bursty traffic in simulation (*N*=15). Note that uncongested flow F0 shares links with congested flow F1 in SW$_1$$\sim$SW$_3$. Figure 11 depicts the individual buffer occupancy of F1 and burst at the congested port P4, as well as the number of received PAUSEs in switches and hosts. DCQCN and TIMELY both receive PAUSEs at switches and hosts, indicating that congestion spreading is severe and the congestion tree grows many branches. The backlogged packets of F1 can not be drained rapidly due to sluggish rate decrease in DCQCN and TIMELY. HoL blocking (i.e., F0 is blocked) also occurs because PAUSEs are received at SW$_1$$\sim$SW$_3$ where uncongested flow F0 sharing ports with F1.

With ACC and also HPCC, congestion spreading is suppressed rapidly with many fewer PAUSEs received at both switches and hosts. ACC and HPCC receive no PAUSEs in SW$_1$$\sim$SW$_3$, indicating that there is no HoL blocking and F0 avoids becoming a victim. Only a few PAUSEs are received due to uncontrollable bursts from H$_1$$\sim$H$_{15}$. Besides, HPCC drastically drains the queue at first, then suffers a standing queue for a long period, i.e., from 2ms to 2.7ms, as illustrated in Figure 11(c). This is because the total sending rate has matched the bottleneck bandwidth while there are still backlogged packets. As shown in Figure 11(d), ACC drains backlogged packets at the congested port at the maximum rate, with 2ms to empty the long-standing queue. This is because ACC senders timely enter the source halt state and refer to the precise information of excessive packets.

ACC is resilient against deadlock: To validate the significance of congestion control in preventing deadlock in lossless Ethernet, we adopt a typical topology as illustrated in Figure 12, which is a common unit in Clos topology. To emulate the typical scenarios where deadlock is prone to appear, we artificially let link L0-T3 and link L1-T0 fail to form a CBD. There are eight servers under each ToR switch. We simulate ACC, DCQCN, TIMELY, and HPCC with *Web Search* workload [6]. The load is 60%. We run simulations 50 times with different traffic traces, and every simulation lasts for 100ms. Table 2 shows the fraction of deadlock runs. We identify a deadlock by finding whether there are flows with infinite completion times. Among 50 simulations, there is no deadlock in ACC and HPCC. However, DCQCN and TIMELY encounter deadlocks 3 and 37 times, respectively. With fast convergence, ACC is resilient against deadlock.

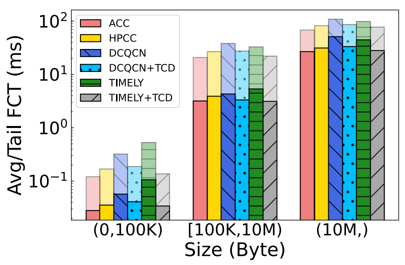
6.4 Large-Scale Simulations

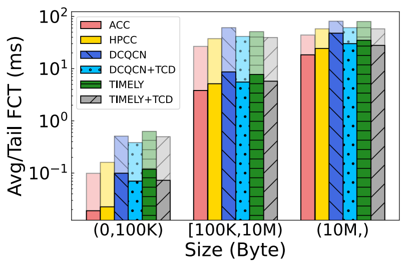
We conduct large-scale simulations to evaluate the overall FCT performance of ACC. For each workload, we classify flows into small (<100KB), medium ($\geq$100KB and $<$ 10MB), and large (>10MB) flows. Note that in *Hadoop* workload, the size of large flows is larger than 1MB.

ACC achieves low latency for small and medium flows: Figure 13(a) exhibits the average FCT and 99th percentile FCT under *Web Search* workload. On the whole, DCQCN and TIMELY have undesirable performance. For DCQCN, the average FCT for small flows is 3.3$\times$ of ACC, and the 99th percentile FCT is 11.5$\times$ of ACC. This is because ACC can quickly converge to the appropriate rate through the ACK arrival rate. Besides, the FCT performance of ACC is better than HPCC. For small flows, ACC reduces the average FCT by 29% compared to HPCC, and the 99th percentile FCT drops by 40%. The performance gain on small flows comes from the source halt that directly stops the transmission of the congested flow so that the queues can be rapidly drained.

Under *Cache Follower* workload, ACC achieves better FCT performance than HPCC, with lower FCT even for small and medium flows. For medium flows, ACC reduces the 99th percentile FCT by 41% compared with HPCC. For DCQCN, the average FCT and 99th percentile FCT are 2.4$\times$ and 1.7$\times$ larger than ACC, respectively.

Under *Hadoop* workload, the FCT performance of ACC is close to HPCC. As shown in Figure 13(c), ACC achieves comparable FCT performance with HPCC for small and medium flows. The flow sizes in *Hadoop* workload are smaller, with 55% of flows being less than 1KB. Such flow size distribution induces less persistent congestion than the other two workloads. As a result, the queuing delay dominates the FCT for


(a) *Web Search* 80% load


(b) *Cache Follower* 80% load

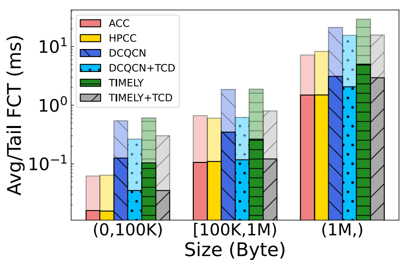
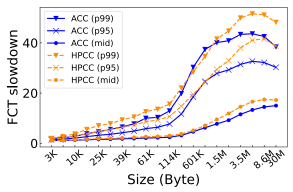

(c) *Hadoop* (50% load)+*incast* (20% load)

Figure 13: Average and tail (99th percentile) FCT performance under various workloads.


(a) *Web Search* 80% load

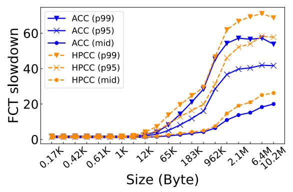
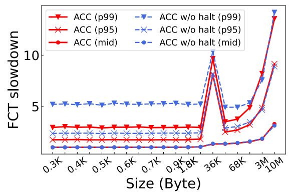

(b) *Cache Follower* 80% load

Figure 14: FCT slowdown performance


(a) FCT slowdown

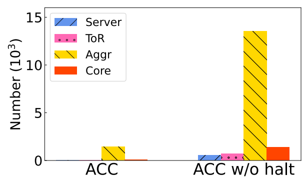
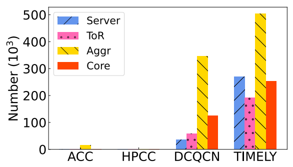

(b) PFC PUASEs

Figure 16: Impact of source halt


(a) *Web Search* 80% load

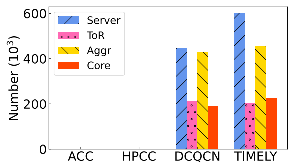

(b) *Cache Follower* 80% load

Figure 15: Number of PFC PAUSEs

small flows. As HPCC sacrifices link utilization leading to queue underflow, thus benefiting FCT performance for small and medium flows.

ACC does not sacrifice throughput of large flows: Figure 14(a) and Figure 14(b) depict the detailed FCT slowdown under *Cache Follower* and *Web Search* workload. For numerous small and medium flows, ACC achieves comparable or even better 99th percentile FCT slowdown than HPCC under both workloads while always attaining better performance than HPCC for relatively large flows. For example, for flows larger than 30MB/10MB, the 99th percentile FCT slowdown with ACC is 26% and 27% lower than HPCC, respectively. In Figure 13(b), ACC also outperforms HPCC by 30% for the 99th percentile FCT of large flows. ACC does not sacrifice the throughput of large flows because it precisely drains out excessive packets and does not underflow the queues to waste link bandwidth.

ACC outperforms DCQCN+TCD and TIMELY+TCD enhanced with heuristic rules: Figure 13 also demonstrates that with accurate congestion detection and heuristic rules on adjusting the rate of congested/victim flows, both DCQCN+TCD and TIMELY+TCD improve the average and tail FCT performance compared with DCQCN and TIMELY. However, since DCQCN+TCD and TIMELY+TCD still heuristically decrease the rate of congested flows following the original paradigm, they can not drain congested queues rapidly and allocate the proper rate precisely under various workloads. Thus, DCQCN+TCD and TIMELY+TCD may suffer from slow convergence and long-standing queues. For instance, under *Cache Follower* workload, ACC improves tail FCT performance of small flows by 3.9× and 5.1× compared with DCQCN+TCD and TIMELY+TCD, respectively.

ACC greatly reduces PFC PAUSE generation: The number of received PAUSEs in different layers under *Web Search* and *Cache Follower* workload is drawn in Figure 15. The results show that ACC almost has no trigger of PFC messages because of fast convergence and the ability to maintain low-standing queues. Thus there is little HoL blocking. In *Web Search* workload, for DCQCN and TIMELY, most PAUSEs are received in the aggregation layer because the challenging incast pattern happens in the ToR layer. In *Cache Follower* workload, most PAUSEs are received at the aggregation layer and servers, indicating that congestion spreading is severe hence many innocent senders are affected.

Source halt benefits low latency for small flows and reduces the risk of PFC PAUSEs: Aiming at understanding the significance of source halt, we take a closer look at the comparison between ACC and ACC w/o halt. ACC w/o halt removes the source halt state and directly decreases the sending rate of congested flows according to the ACK arrival rate. Figure 16(a) manifests that ACC w/o halt can achieve similar performance for medium and large flows, while small flows suffer from larger FCT. Small flows experience undesirable queueing delays as congested queues can not be rapidly drained out. ACC w/o halt can still fulfill fast convergence owing to the guidance of ACK arrival rate. Besides, as illustrated in Figure 16(b), since there are more long-standing queues, the risk of triggering PFC PAUSEs increases. ACC significantly reduces the PFC PAUSEs at each layer compared with ACC w/o halt. The congestion spreading is also restricted effectively, with many fewer PAUSEs received at the servers, ToR, and the core layer.

7 Related Work

Lossless Ethernet vs. Lossy Ethernet. There has been an ongoing discussion about lossless and lossy Ethernet in RDMA-based datacenter. Lossless Ethernet can benefit application performance but comes with side effects of deploying PFC (i.e., HoL blocking, congestion spreading, deadlock, etc). However, ACC reveals that besides its side effects, PFC also gives a unique packet conservation property which can facilitate precise congestion control and in turn mitigate the side effects of PFC. In contrast, although lossy Ethernet can essentially avoid the side effects of PFC, it may suffer from latency degradation due to packet losses and imposes higher implementation requirements on RDMA NICs (e.g., more complex loss recovery and reorder logic). Thus, existing efforts can be mainly classified into two lines: *(i) Developing congestion control (or flow control) to minimize the side effects of PFC*: For example, CC schemes including DCQCN [49], PCN [13], TCD/TCD-MQ [45, 46], ACC, etc, and new link-layer flow control like deadlock-free GFC [38]. *(ii) Developing advanced loss recovery to improve RDMA performance*: For example, IRN [34] first introduces selective retransmission into RDMA NICs. TLT [30] proposes to reduce timeouts by protecting important packets. SLR [32] utilizes switches to send loss notifications to request fast retransmissions and is compatible with default Go-back-N retransmission.

Congestion control in lossless interconnects. Besides lossless Ethernet, there are other lossless interconnects, e.g., InfiniBand [9] and Fibre Channel [8]. For example, InfiniBand deploys credit-based flow control to guarantee no packet loss, which also suffers from HoL blocking, congestion spreading, and deadlock problems [9, 22, 41]. Despite the fact that ACC is designed for lossless Ethernet employing PFC, we think our key insights (e.g., utilizing the ACK-driven paradigm to infer excessive packets and throttled rate) are also suitable for congestion control in other lossless interconnects with packet conservation property.

Receiver-driven and other advanced congestion controls. Recently several receiver-driven congestion controls have been proposed, such as ExpressPass [14], pHost [18], NDP [23], and Homa [35]. The core of receiver-driven congestion control is "request and allocation" style transport [24]. These receiver-driven schemes are first proposed in lossy Ethernet, where new flows blindly transmit unscheduled packets in the first RTT. Hence congestion may also occur with the risk of packet loss. Besides, NDP and ExpressPass utilize CP [12] or explicitly drop credits to enforce a bounded queue. PCN [13] is a receiver-driven congestion control scheme designed in lossless Ethernet while only utilizing the receiving rate at the receiver to guide rate decrease, thus accumulated queues may still be eliminated slowly and HoL blocking can not be suppressed rapidly. There are also other advanced congestion controls designed for datacenters. For instance, PowerTCP [4] proposes to fulfill fine-grained congestion control by adapting to *Power* (i.e., bandwidth-window product) based on INT. Bolt [7] leverages sub-RTT signals provided by programmable switches to enable reacting to congestion faster than the RTT control loop, which realizes low queueing and benefits minimal packet loss under bursty workloads. Besides, Fasspass [37] first proposes to delegate all control (i.e., when each packet should be transmitted and what path it should follow) to a centralized arbiter.

ACC stands out as it fully builds on the simple yet powerful packet conservation law in lossless Ethernet while achieving low latency and high throughput simultaneously.

Source halt. On-Ramp [31] proposes to pause the source actively to deal with transient and equilibrium tension in traditional lossy Ethernet. The source PAUSE technique in On-Ramp is similar to the source halt state in ACC. However, On-Ramp relies on clock synchronization [20] to obtain accurate one-way delay measurement and roughly estimates the pause time without knowing network pipe capacity. The source halt state in ACC precisely drains out accumulated packets of congested flows while only relying on the ACK time series. To the best of our knowledge, none of the existing congestion control schemes in lossless Ethernet proposes to accurately and rapidly drain out backlogged packets.

8 Conclusion

In this paper, we revisit congestion control for lossless Ethernet from the perspective of unlocking its intrinsic properties, e.g., *packet conservation*. We propose a new congestion control called ACC, which essentially treats the ACKs as accurate "tokens" of the network pipe to guide the traffic injection rate and timing, i.e., waiting for backlogged packets to drain out and then sending with the rate corresponding to network pipe capacity. Extensive experiments show that ACC alleviates thorny issues in lossless Ethernet, such as congestion spreading, HoL blocking, and deadlock. Besides, ACC achieves low latency and high throughput simultaneously under various workloads and even outperforms state-of-the-art congestion control schemes. We believe that the philosophy of exploiting intrinsic properties opens new avenues for rethinking congestion control and other traffic management mechanisms in lossless Ethernet.

Acknowledgements

We thank the anonymous reviewers and our shepherd Naveen Kr. Sharma, for their valuable comments and helpful suggestions. This work is supported in part by the National Key Research and Development Program of China (No. 2022YFB2901404), and by National Natural Science Foundation of China (NSFC) under Grant No. 62132007, No. 62221003 and No. 62302055, and by the China Post-Doctoral Science Foundation under Grant No. 2023M730172, as well as gifts from Huawei. Corresponding author: Qingkai Meng and Fengyuan Ren.

References

[1] Ieee 802.1 qbb - priority-based flow control. http://www.ieee802.org/1/pages/802.1bb.html, 2010.

[2] In-band network telemetry (int) dataplane specification. https://p4.org/p4-spec/docs/INT_v2_1.pdf, 2020.

[3] Nvm express over fabrics revision 1.1a. https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1a-2021.07.12-Ratified.pdf, 2021.

[4] 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, Renton, WA, April 2022. USENIX Association.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 63–74, New York, NY, USA, 2008. Association for Computing Machinery.

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery.

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

[8] Fibre Channel Industry Association. Fibre channel. https://fibrechannel.org/, 2022.

[9] InfiniBand Trade Association. Infiniband architecture specification volume 2 release 1.3.1. https://cw.infinibandta.org/document/dl/8125, 2016.

[10] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. Attack of the killer microseconds. *Commun. ACM*, 60(4):48–54, mar 2017.

[11] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, page 267–280, New York, NY, USA, 2010. Association for Computing Machinery.

[12] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. Catch the whole lot in an action: Rapid precise packet loss notification in data center. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 17–28, 2014.

[13] 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.

[14] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 239–252, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[16] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, jan 2008.

[17] Nandita Dukkipati, Neal Cardwell, Yuchung Cheng, and Matt Mathis. Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses. Internet-Draft draft-dukkipati-tcpm-tcp-loss-probe-01, Internet Engineering Task Force, February 2013. Work in Progress.

[18] 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.

[19] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 519–533. USENIX Association, April 2021.

[20] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 81–94, Renton, WA, April 2018. USENIX Association.

[21] 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.

[22] M. Gusat, D. Craddock, W. Denzel, T. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato. Congestion control in infiniband networks. In *13th Symposium on High Performance Interconnects (HOTI'05)*, pages 158–159, 2005.

[23] 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.

[24] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. Aeolus: A building block for proactive transport in 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 422–434, New York, NY, USA, 2020. Association for Computing Machinery.

[25] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Deadlocks in datacenter networks: Why do they form, and how to avoid them. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, page 92–98, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Tagger: Practical pfc deadlock prevention in data center networks. *IEEE/ACM Trans. Netw.*, 27(2):889–902, apr 2019.

[27] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *ACM SIGCOMM*, pages 514–528, 2020.

[28] Yuliang Li. HPCC NS-3 simulator. https://github.com/alibaba-edu/High-Precision-Congestion-Control, 2019.

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

[30] Hwijoon Lim, Wei Bai, Yibo Zhu, Youngmok Jung, and Dongsu Han. Towards timeout-less transport in commodity datacenter networks. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 33–48, New York, NY, USA, 2021. Association for Computing Machinery.

[31] Shiyu Liu, Ahmad Ghalayini, Mohammad Alizadeh, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. Breaking the Transience-Equilibrium nexus: A new approach to datacenter packet transport. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 47–63. USENIX Association, April 2021.

[32] Qingkai Meng, Yiran Zhang, Shan Zhang, Zhiyuan Wang, Tong Zhang, Hongbin Luo, and Fengyuan Ren. Switch-assistant loss recovery for rdma transport control. *IEEE/ACM Transactions on Networking*, pages 1–16, 2023.

[33] 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. *SIGCOMM Comput. Commun. Rev.*, 45(4):537–550, aug 2015.

[34] 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.

[35] 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.

[36] NVIDIA. How To Configure Soft-RoCE. https://enterprise-support.nvidia.com/s/article/howto-configure-soft-roce, 2023.

[37] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: a centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 307–318, New York, NY, USA, 2014. Association for Computing Machinery.

[38] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. Gentle flow control: Avoiding deadlock in lossless networks. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 75–89, New York, NY, USA, 2019. Association for Computing Machinery.

[39] 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.

[40] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, aug 2015.

[41] J.R. Santos, Y. Turner, and G. Janakiraman. End-to-end congestion control for infiniband. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 2, pages 1123–1133 vol.2, 2003.

[42] Brent Stephens, Alan L. Cox, Ankit Singla, John Carter, Colin Dixon, and Wesley Felter. Practical dcb for improved data center networks. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1824–1832, 2014.

[43] 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*, APNET '23, page 15–20, New York, NY, USA, 2023. Association for Computing Machinery.

[44] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, oct 2016.

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

[46] Yiran Zhang, Qingkai Meng, Yifan Liu, and Fengyuan Ren. Revisiting congestion detection in lossless networks. *IEEE/ACM Transactions on Networking*, 31(5):2361–2375, 2023.

[47] Yiwen Zhang, Gautam Kumar, Nandita Dukkipati, Xian Wu, Priyaranjan Jha, Mosharaf Chowdhury, and Amin Vahdat. Aequitas: admission control for performance-critical rpcs in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 1–18, 2022.

[48] Yiying Zhang and Steven Swanson. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2015.

[49] 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.

[50] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. Ecn or delay: Lessons learnt from analysis of dcqcn and timely. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, page 313–327, New York, NY, USA, 2016. Association for Computing Machinery.

[51] Siyuan Zhuang, Zhuohan Li, Danyang Zhuo, Stephanie Wang, Eric Liang, Robert Nishihara, Philipp Moritz, and Ion Stoica. Hoplite: Efficient and fault-tolerant collective communication for task-based distributed systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 641–656, New York, NY, USA, 2021. Association for Computing Machinery.

Parameter	Definition
C	Bottleneck link bandwidth
N	The number of flows
T	Control period
k	The kth period
RI	Rate increment
$S_i(k)$	Sending rate of flow i at kth period
$R_i(k)$	Actual sending rate of flow i considering the source halt state at kth period
$Q(k)$	Bottleneck queue length at kth period
$\varphi_i(k)$	Allocated bandwidth fraction of flow i at kth period, $\varphi_i(k) = \frac{S_i(k)}{\sum S_i(k)}$

Table 3: Parameter list

APPEDNDIX

A. Theoretical Analysis

In this section, we analyze the convergence and fairness of ACC in theory. The key parameters are listed in Table 3. We assume that the receiver will generate ACK for each received packet and the delay of generating ACK is negligible.

Convergence: Without loss of generality, assuming that the switch queue corresponding to the bottleneck link is initially empty, the sending rate of N flows is arbitrary. There are following two cases:

(1) $\sum S_i(0) > C$: If the aggregate sending rate of the initial N flows exceeds the bottleneck link bandwidth, congestion occurs and queues start to accumulate. All congested flows will enter the source halt state:

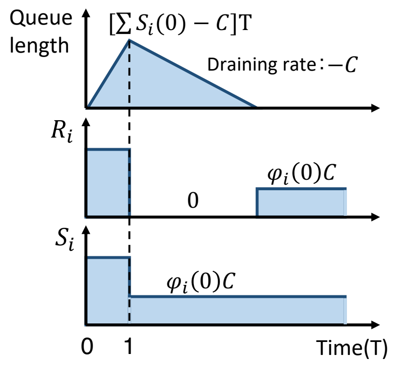
$$\begin{cases} Q(1) = (\sum S_i(0) - C)T \\ R_i(1) = 0 \\ S_i(1) = \varphi_i(0)C \end{cases} \tag{2}$$

As shown in Figure 17(a), after each flow enters the source halt state, the actual sending rate $R_i(1) = 0$. Flow i stops transmission to wait for the queue to be emptied. During this period, the sending rate $S_i(1)$ derived from ACK arrival rate is $\varphi_i(0)C$, and is recorded for following transmission. The congested queue drains at the rate C, and needs $\lceil \frac{\sum S_i(0)-C}{C} \rceil$ periods to approach 0:

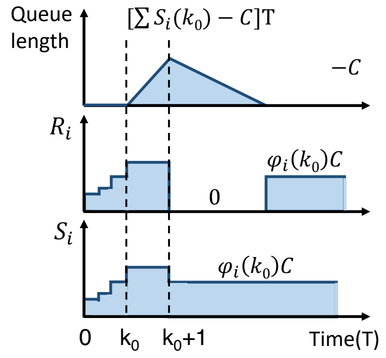
$$\begin{cases} Q(k) = (\sum S_i(0) - C)T - (k-1)CT \\ R_i(k) = \varphi_i(0)C \\ S_i(k) = \varphi_i(0)C \end{cases} \tag{3}$$

Thus, we derive $\sum S_i(k) = \sum \varphi_i(0)C = \sum \frac{S_i(0)}{\sum S_i(0)} C = C$, indicating that the aggregate sending rate $\sum S_i(k)$ converges to C within one period. After leaving the source halt state, $R_i(k) = S_i(k)$ and flow i starts normal transmission.

(2) $\sum S_i(0) \le C$: If the initial aggregate sending rate of N flows is less than or equal to the bottleneck link bandwidth, all flows will try to increase the sending rate. Eventually, the aggregate sending rate of N flows will exceed the bottleneck bandwidth after k_0 periods, then the evolution behavior of the congested queue and aggregate sending rate of N flows is consistent with those in the previous case. Figure 17(b) shows the evolution of bottleneck queue length, R_i and S_i.



(a) $\sum S_i(0) > C$



(b) $\sum S_i(0) \le C$

Figure 17: Convergence of ACC

To sum up, when the aggregate sending rate is larger than the bottleneck link bandwidth, ACC can converge to the bottleneck link bandwidth within one control period T and eliminate queue accumulation at the fastest speed of the bottleneck link bandwidth.

Fairness: Assuming that the above convergence process ends at the beginning of the k_1 period, when the switch queue has been emptied, and the sending rate of the flow i starts to increase from $\varphi_i(k_0)C$. Let $r = \frac{RI}{C}$, we have

$$\begin{cases} Q(k_1) = 0 \\ S_i(k_1) = (1-r)\varphi_i(k_0)C + rC \\ R_i(k_1) = S_i(k_1) \end{cases} \tag{4}$$

We have

$$\varphi_i(k_1) = \frac{S_i(k_1)}{\sum S_i(k_1)} = \frac{(1-r)\varphi_i(k_0) + r}{1 - r + rN} \tag{5}$$

Since $N > 1$, it is obvious that $\sum S_i(k_1) = (1 + rN - r)C > C$. The link is congested, and the flow i will slow down in the next period and enter the source halt state:

$$\begin{cases} Q(k_1+1) = (rN - r)CT \\ S_i(k_1+1) = \varphi_i(k_1)C \\ R_i(k_1+1) = 0 \end{cases} \tag{6}$$

The switch queue will be drained at a rate of C, during which the sending rate S_i is kept unchanged, we have

$$\begin{cases} Q(k_1+m) = (rN - r - m + 1)CT \\ S_i(k_1+m) = \varphi_i(k_1)C \\ R_i(k_1+m) = 0 \end{cases} \tag{7}$$

Let $M = \lceil rN - r + 1 \rceil$. Finally, at the beginning of the $k_1 + M$ period, the queue length is 0, and the flow i will start to increase the sending rate again. We have

$$\begin{cases} Q(k_1+M) = 0 \\ S_i(k_1+M) = (1-r)\varphi_i(k_1)C + rC \\ R_i(k_1+M) = S_i(k_1+M) \end{cases} \tag{8}$$

We have

$$\varphi_i(k_1+M) = \frac{S_i(k_1+M)}{\sum S_i(k_1+M)} = \frac{(1-r)\varphi_i(k_1)+r}{1-r+rN} \tag{9}$$

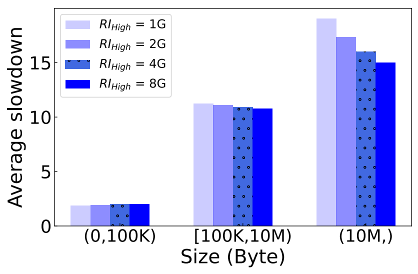
Comparing Equation (5) and Equation (9), it can be seen that ACC will repeat the process of rate increase for one period and rate decrease for $M-1$ periods. At the same time, according to Equation (5) and Equation (9), we can calculate $\varphi_i(k_1 + kM)$ as follows:

$$\begin{aligned} \varphi_i(k_1+kM) &= \frac{(1-r)\varphi_i(k_1+(k-1)M)+r}{1-r+rN} \\ &= \frac{1-r}{1-r+rN}\varphi_i(k_1+(k-1)M) + \frac{r}{1-r+rN} \\ &\dots \\ &= (\frac{1-r}{1-r+rN})^{k+1}\varphi_i(k_0) + \\ &\sum_{j=0}^{k}(\frac{1-r}{1-r+rN})^{j}(\frac{r}{1-r+rN}) \end{aligned} \tag{10}$$

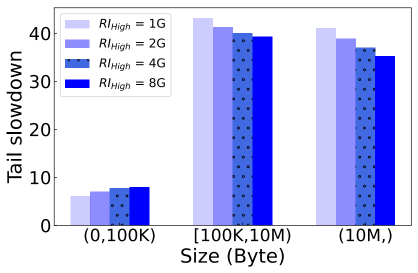
When $k \to \infty$, as $\frac{1-r}{1-r+rN} < 1$, so $(\frac{1-r}{1-r+rN})^{k+1}$ approaches zero. We have:

$$\varphi_i(k_1+kM) \to \frac{\frac{r}{1-r+rN}}{1-(\frac{1-r}{1-r+rN})} = \frac{1}{N} \tag{11}$$

Equation (11) indicates that ACC can achieve fair rate allocation.



(a) Average slowdown



(b) The 99th percentile slowdown

Figure 18: Parameter sensitivity of *RI*, 80% load (*WebSearch*)

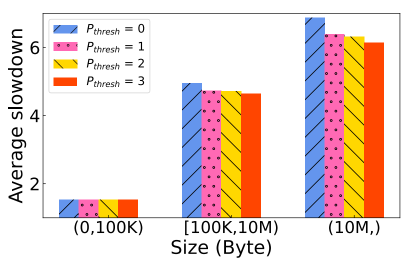
B. Supplementary of Evaluations

To better understand the parameter sensitivity and link utilization performance of ACC, we supplement experiment results in this section. In §B.1, we present the parameter sensitivity of *RI* and P_{thresh}. After that, we present the simulation results under typical bursty traffic (§B.2).

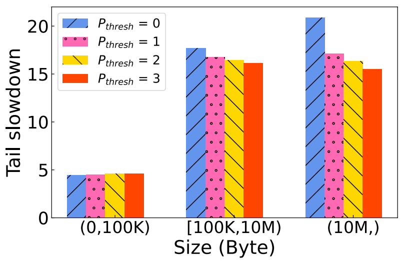
B.1 Parameter Sensitivity of *RI* and P_{thresh}

Parameter *RI*: In ACC, the key parameter in the rate increase process is *RI*. A larger *RI* indicates a more aggressive rate increase. To explore the performance sensitivity of ACC on *RI*, we use the same *Web Search* workload and topology as in Figure 14(a) but vary *RI* between 1G and 8G. Figure 18 shows the average FCT slowdown of small, medium, and large flows. On the whole, a larger *RI* yields higher throughput for large flows but may introduce larger FCT for small and medium flows. Due to the source halt state and ACK arrival rate guidance in ACC, the sending rate can quickly drop once the aggressive rate increase induces congestion. As a result, the average FCT slowdown of small and medium flows is not sensitive to *RI*. The overall results indicate that our recommended value of *RI* is reasonable.

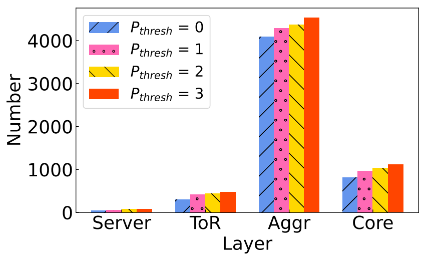
Parameter P_{thresh}: In ACC, P_{thresh} controls the rate regulation of victim flows. Victim flows will keep its rate until P_{thresh} consecutive periods. A smaller P_{thresh} will enforce the victim flow to throttle its rate earlier to help mitigate the congestion spreading. To explore the performance sensitivity of ACC on P_{thresh} especially when HoL blocking happen frequently, we run a challenging workload consisting of foreground traffic



(a) Average slowdown

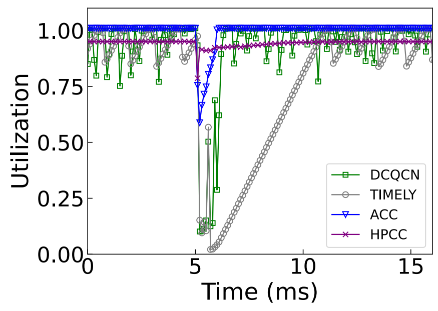


(b) The 99th percentile slowdown

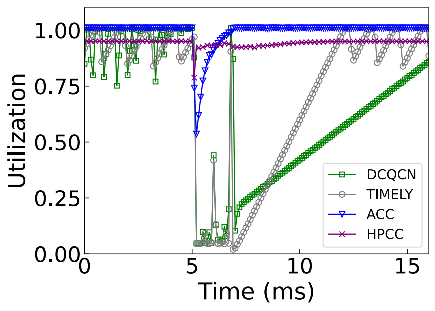


(c) The number of PFC PAUSEs

Figure 19: Parameter sensitivity of P_{thresh}, 80% load (*WebSearch* + incast)



(a) $N = 10$



(b) $N = 20$

Figure 20: Utilization of link <SW$_3$-SW$_4$>. Bursts start at time 5ms. Topology is Figure 1(a).

and background traffic: 60% *Web Search* workload as the foreground traffic with 20% load of 64-1 incast workload as the background traffic. For each incast, 64 randomly selected senders send 50KB to one receiver. We vary P_{thresh} from 0 to 3. Note that $P_{thresh} = 0$ represents that victim flows will always reduce half of the sending rate per T.

Figure 19(a) and Figure 19(b) show the average and 99th percentile FCT slowdown of the foreground *WebSearch* workload. As P_{thresh} increases, medium flows and large flows have better FCT slowdown performance. For example, for large flows, the 99th percentile FCT slowdown with $P_{thresh} = 0$ is 1.22× of when $P_{thresh} = 1$. For medium and large victim flows, by keeping the sending rate until P_{thresh} periods, they lose less throughput with a larger P_{thresh}. While the FCT slowdown performance of small flows with size less than 10KB are mainly determined by queueing delay. As a result, the average and tail FCT slowdown of small flows are not sensitive to P_{thresh}.

However, as shown in Figure 19(c), a larger P_{thresh} may trigger more PFC PAUSEs at each layer, indicating heavier HoL blocking and more risks of PFC. This is because with a large P_{thresh}, victim flows react very late to congestion spreading, at which HoL blocking already occurs for a long time and propagations to multiple switches. Indeed, throttling the rate of victim flows early can aggressively reduce the injected traffic around the blocked switch ports, thus speeding up the alleviation of HoL blocking. On the whole, the results indicate that our recommended value of P_{thresh} (i.e., 1) is proper, which can balance the FCT performance and alleviation of HoL blocking.

B.2 Link Utilization

We adopt the same topology and traffic pattern as in Figure 1(a) to evaluate the link utilization performance especially under bursty traffic. Note that uncongested flow F0 shares links with congested flow F1 in SW$_1$∼SW$_3$. At time 5ms, concurrent bursts with size of 64KB start. Then the congested flow F1 should be throttled while uncongested flow F0 should be unaffected and occupy available bandwidth. We use $N = 10$ and $N = 20$ to introduce different degree of burstiness.

As illustrated in Figure 20(a), after time 5ms, both DCQCN and TIMELY incur low link utilization lasting for around 1.5ms and 6ms, respectively. Figure 20(b) shows that when $N = 20$, DCQCN and TIMELY get worse burst tolerance performance, with around 7ms and 12ms to recover to full link utilization. Due to step-by-step rate adjustment rules of DCQCN and TIMELY, they can not eliminate congestion at SW$_4$ quickly and congestion spreads to upstream SW$_3$, thus link <SW$_3$-SW$_4$> is blocked. The low link utilization is also attributed to the uncongested flow F0 being throttled mistakenly due to misleading queue-based (or delay-based) congestion signals. ACC incurs not full link utilization lasting for about 1.5ms ($N = 10$) and 2ms ($N = 20$), which is the time required for uncongested flow F0 to increase the sending rate. HPCC can ramp up quickly after bursts start but only maintains a 95% link utilization in steady state.