USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

CCC: Re-architecting Delay-based Congestion Control in Datacenter Networks

Wanchun Jiang, Haoyang Li, Kai Wang, Yujie Hu, and Xiao Han, *Central South University;* Danfeng Shan, *Xi'an Jiaotong University;* Fengyuan Ren, *Tsinghua University;* Jiawei Huang and Jianxin Wang, *Central South University*

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

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

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

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

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

CCC: Re-architecting Delay-based Congestion Control in Datacenter Networks

Wanchun Jiang[†], *Haoyang Li*[†], *Kai Wang*[†], *Yujie Hu*[†], *Xiao Han*[†], *Danfeng Shan*[‡,*],
Fengyuan Ren[§], *Jiawei Huang*[†], *Jianxin Wang*[†]
[†]*Central South University,* [‡]*Xi'an Jiaotong University,* [§]*Tsinghua University*

Abstract

In datacenter networks, delay-based congestion control (CC) algorithms are popular and protocols such as TIMELY and SWIFT are employed in production. However, even if delay signals are precisely measured, the congestion information deduction (CiD) may be incorrect under certain conditions, degrading the performance of CC. To address this problem, we re-architect the delay-based CC to make rate adjustments based not only on the congestion states but also the CiD trustworthiness. Based on this architecture, the CiD-sensitive congestion control (CCC) is developed. Specifically, CCC demarcates the regions, where the CiD from delay signals is untrustworthy, with the assistance of well-designed congestion criteria and inherent system inertia. When the CiD is untrustworthy, CCC constructs the probing rate adjustment rules for both the rapid response to congestion and the subsequent trustworthy CiD. Otherwise, CCC conducts the analytical rate adjustment to both reach the expected congestion state and keep the trustworthy CiD, fully leveraging the abundant information in delay signals. DPDK-based experiments and large-scale simulations confirm that CCC achieves similar good performance as INT-based CCs, including HPCC and PowerTCP, by using only delay signals, outperforming DCQCN, TIMELY, and SWIFT.

1 Introduction

Nowadays, the datacenter has become the infrastructure of cloud computing and artificial intelligence (AI). Meanwhile, applications produce complex traffic patterns and impose strict performance requirements on datacenter networks. Specifically, short flows, which take the majority in classic applications such as web search [11] and data mining [24], desire low latency. In contrast, long flows, which take the majority in emerging applications such as the training of AI models [57], require high throughput. Therefore, modern congestion control (CC) schemes should simultaneously achieve high throughput and low latency. Moreover, CC is also expected to avoid packet dropping or triggering PFC (Priority Flow Control) pauses [1].

Currently, many delay-based CCs have been developed for datacenter networks [32, 34, 44] due to the convenient deployment. They typically share the same architecture shown in Fig. 1, consisting of delay signal measurement, congestion information deduction (CiD), and rate adjustment rules. For example, TIMELY and SWIFT enable precise delay measurement and directly utilize the deduced congestion information to guide their rate adjustment. The advantage of delay signals is that fine-grained CiD can be made to reflect the congestion intensity, compared to the ECN (Explicit Congestion Notification) mechanism adopted by DCTCP [11] and DCQCN [67]. The dilemma is that the CiD may be wrong under several delay ambiguity conditions (DACs), including large queue length, link under-utilization, and dramatic queue variations, even if the delay signals are precisely measured, as revealed in §3.1. Without judging the CiD trustworthiness in the current architecture, existing delay-based CCs are hard to fully or correctly leverage the abundant information of delay signals.

In this work, we propose the CiD-sensitive architecture, which fully leverages the abundant information of delay signals and avoids the negative impacts of untrustworthy CiD. In brief, the CiD-sensitive architecture incorporates not only the congestion state, as in traditional architecture, but also the CiD trustworthiness to guide rate adjustments, as shown in Fig. 1. Meanwhile, the rate adjustment not only controls the congestion state but also enhances the CiD trustworthiness. In detail, when CiD is untrustworthy, the rate adjustment cannot rely on the explicit congestion intensity deduced from delay signals. In this case, the rough direction is solely used to avoid irrational responses to congestion and enhance the subsequent trustworthy CiD. On the contrary, trustworthy CiD directly provides accurate congestion intensity to guide rate adjustment. In this case, CCC can fully utilize the abundant information obtained from delay signals to reach the expected congestion state and keep CiD trustworthiness.

Based on the CiD-sensitive architecture, we design the CiD-

*Danfeng Shan is the corresponding author

sensitive congestion control (CCC) algorithm, which focuses on both the congestion state and the CiD trustworthiness. Specifically, CCC mainly addresses two key challenges: (1) the distinction of DACs, which corresponds to untrustworthy CiD, and (2) making rate adjustments to properly react to congestion and enhance CiD trustworthiness simultaneously. To address these challenges, CCC divides regions based on both congestion state and CiD trustworthiness, making rate adjustments for different purposes under different regions. In detail, CCC demarcates regions with the assistance of both congestion criteria and inherent system inertia. In each region, the direction of rate adjustments is determined by congestion state, while the rate adjustment rules are determined by CiD trustworthiness. Specifically, when CiD is trustworthy in DAC-tolerable regions, CCC conducts analytical rate adjustment rules for fast response to congestion and maintains the trustworthy CiD. Conversely, CCC conducts probing rate adjustment rules to drive itself into the DAC-tolerable regions for both the subsequent trustworthy CiD and good congestion state. During this process, the impact of untrustworthy CiD is either tolerated or eventually compensated during iterations. Combining these operations, CCC goes beyond merely focusing on congestion state and considers CiD trustworthiness by fully mining the abundant information of delay signals.

We implement CCC with DPDK [2] and conduct wide evaluations under complementary scenarios to show the good comprehensive performance of CCC across multiple conflicting indices. The first pair of conflicting indices is high throughput and low latency in equilibrium. CCC keeps high throughput. Meanwhile, the 99.9^{th} tail latency of CCC is smaller than DCQCN (62.1% $\sim$ 65.0%), TIMELY (6.8% $\sim$ 45.3%), and SWIFT (15.3% $\sim$ 17.5%). Second, CCC achieves good transience of rapid occupation of available bandwidth and suppression of congestion, achieving a favorable trade-off with equilibrium [40]. CCC occupies available bandwidth more than 3.5$\times$ faster than SWIFT, TIMELY, and DCQCN. Meanwhile, CCC suppresses congestion 304.6$\times$, 25.4$\times$, and 6.6$\times$ faster than DCQCN, TIMELY, and SWIFT, respectively. The third is fairness, which conflicts with the above two pairs of performance indices on efficiency [21, 33]. CCC remarkably reduces the time to achieve the fair bandwidth share, outperforming SWIFT and HPCC by an order of magnitude.

Benefiting from the above advantages, under realistic workloads of WebSearch [11], FB_Hadoop [48], CCC achieves low flow completion time (FCT) similar to HPCC and PowerTCP without relying on new devices and the INT feature. Besides, CCC reduces FCT by up to 47.6% compared to DCQCN and TIMELY, and shortens the FCT of long flows by 34.7% $\sim$ 48.8% compared to θ-PowerTCP. In addition, under the model training scenario, CCC triggers PFC pauses 85.7% $\sim$ 95.1% lower than DCQCN, TIMELY, and SWIFT.

In summary, the contributions of this paper are as follows.

- Revealing the issue of traditional architecture of delay-based CC that the untrustworthy CiD from delay signals, even if delay signals are precisely measured, would hurt the performance of delay-based CC.
- Proposing the CiD-sensitive architecture which adds the judgement of CiD trustworthiness before rate adjustments such that the abundant information of delay signals is fully leveraged.
- Designing CCC, which flexibly conducts either analytical or probing rules according to CiD trustworthiness, and implementing CCC on DPDK. Evaluations with comprehensive indices and realistic workloads show CCC performs as well as INT-based CCs with solely delay signals, outperforming existing delay-based CCs.

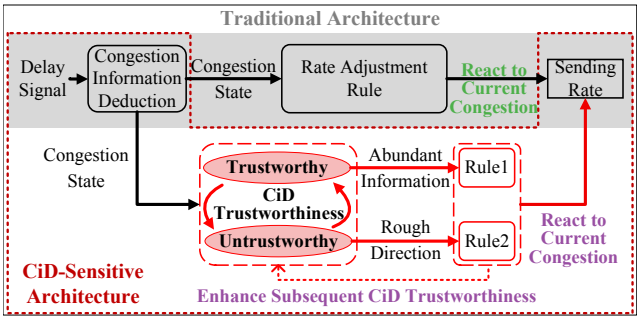


Figure 1: Architecture of Delay-based CCs.

2 Delay-based CC in Datacenter

Delay signals have played an important role in CC since the publication of Vegas [15]. Nowadays, delay signals are still the core of popular CCs such as Copa [13] and BBR [16]. In datacenters, the measurement of delay signals becomes precise [32, 44]. Meanwhile, delay signals are more fine-grained to reflect the intensity of congestion, compared to the ECN signals adopted by DCTCP and DCQCN [11, 67]. Consequently, delay-based CCs are good candidates in datacenters.

Although many delay-based CCs [9, 32, 34, 44] have been proposed for datacenters, they follow the same architecture shown in Fig. 1, consisting of delay signal measurement, CiD, and rate adjustment rules. For example, TIMELY [44] enables precise delay signal measurement via hardware timestamps. Its CiD focuses on both queue length and queue variation, obtained from the measured RTT and delay gradient. Such CiD directly provides congestion state for its probing rate adjustment rules, which heuristically adjust sending rate and iterate toward an expected point [36]. Moreover, inheriting the precise delay signal measurement, SWIFT [32] solely employs the CiD on queue length for the guidance of its probing rate adjustment rules. The delay gradient is eliminated to avoid multiple equilibrium points [68]. To address the doubt that delay signals would lag behind ECN in the face of large queuing delay [68], SWIFT sets a low target delay for its

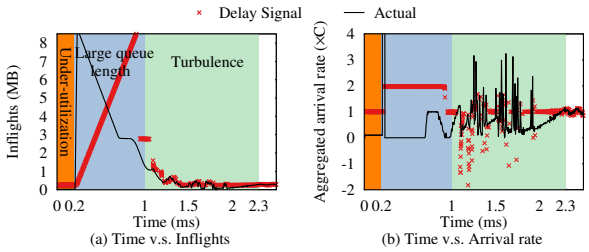


Figure 2: CiD from delay signals might be wrong.

probing rate adjustment rules. Furthermore, θ-PowerTCP [9] measures both RTT and delay gradient, employs CiD on both the inflights and the aggregated arrival rate, and then makes analytical rate adjustment, which precisely adjusts the sending rate to the expected point in one shot. In sum, traditionally, delay-based CCs make rate adjustments directly based on the congestion state provided by CiD for good performance.

3 Motivation

3.1 Congestion Information Deduction

Although the delay signal measurement has become precise and rate adjustment rules have been well refined, the performance of delay-based CCs may still degrade due to the wrong CiD under certain conditions. The following results illustrate when CiD becomes wrong and how it affects CCs.

In the test, one host serves as the receiver, while fifty nodes act as senders, with a link bandwidth of C. Different types of flows are initiated at different times. In detail, a flow with the initial sending rate $0.1C$ is active for $0 \sim 0.2ms$. Moreover, let 50 long flows are initialized with the sending rate C share the same bottleneck link and start concurrently at $0.2ms$. During $1 \sim 2.3ms$, multiple short flows randomly start and leave. θ-PowerTCP is employed because delay signals are used to deduce the congestion information of both inflights and aggregated arrival rates. The deduction results of the bottleneck link are shown in Fig. 2. For comparison, the actual values collected at bottleneck switches are exhibited by postponing one base RTT for the convenience of observation. The results are that the deduced inflights and aggregated arrival rates from delay signals severely deviate from the actual values under the following three conditions, which we refer to as delay ambiguity conditions (DACs).

***DAC 1*. The link is underutilized.** Under this condition, both the measured queuing delay and delay gradient equal 0. Therefore, the deduced inflights from the delay signals are the same as *BDP* (Bandwidth-Delay Product), and the aggregated arrival rate is deduced as the bottleneck bandwidth C, as shown in the orange area of Fig. 2(b). With the above information, θ-PowerTCP would wrongly recognize that it has converged and fail to occupy the available bandwidth during 0 to $0.2ms$. Similar issues are also observed in [9].

***DAC 2*. The queue length is large.** Under this condition, delay signals are significantly postponed. Specifically, as shown in Fig. 2(a), after initialization, the actual inflights instantaneously ramp up to the peak and then ramp down. In contrast, the deduced inflights from delay signals are severely postponed during $0.2ms \sim 1ms$, i.e., increase slowly in the blue area of Fig. 2, even the one base RTT offset between the measured and the actual signals has been manually aligned. The underlying reason is that the queuing delays of packets gradually rise. Correspondingly, the aggregated arrival rate is wrongly deduced during $0.2ms \sim 1ms$ as shown in Fig. 2(b), while the actual aggregated arrival rate increases hugely and then decreases rapidly to 0. Consequently, when the queue length is large, the wrong deductions from postponed delay signals will mislead the rate adjustment of θ-PowerTCP. Similar issues are also observed in [32, 68].

***DAC 3*. The bottleneck queue experiences turbulence.** Under this condition, the delay gradient might be wrongly measured due to skipped change points [52]. In the green area of Fig. 2(b), the bursty short flows frequently enter and leave from $1ms$ to $2.3ms$, and thus the bottleneck queue experiences turbulence, i.e., frequent drastic queue expansion and contraction. In this case, the deduced aggregated arrival rate severely deviates from the actual value. This is because the change points of the turbulent queue are usually skipped by senders, disturbing the capture of the continuous queue variations. Moreover, the aggregated arrival rate is wrongly deduced as negative under turbulence, due to the leapfrog change of the queuing delay between two consecutive packets. For example, suppose the k^{th} packet of a sender arrives at the bottleneck at t_0 and leaves after queuing delay Δt. Assume the $(k+1)^{th}$ packet of this sender arrives at $t_0+\Delta t$, and leaves at $t_0+\Delta t+\varepsilon$. Accordingly, the aggregated arrival rate is deduced as $(1+(\varepsilon-\Delta t)/\varepsilon)*C$ by this sender. In such a case, the aggregated arrival rate might be wrongly deduced as negative when Δt is large under turbulence. This phenomenon occurs even though the back-to-back packets corresponding to tiny ε are filtered out. The wrongly deduced aggregated arrival rate would also mislead the rate adjustment of θ-PowerTCP.

In reality, these DACs and other potential DACs are unavoidable due to the variable traffic. Accordingly, CiD is not always trustworthy. Without knowing the CiD trustworthiness, existing delay-based CCs fail to fully leverage the abundant information of delay signals. For example, TIMELY and SWIFT skeptically adopt delay signals by just leveraging CiD to determine the direction of their probing rate adjustments. Such a skeptical adoption may waste the abundant information of delay signals when the CiD is trustworthy, resulting in sub-optimal performance, e.g., slowly respond to network environment variations [36]. In contrast, θ-PowerTCP credulously adopts delay signals by permanently treating CiD as trustworthy in its analytical rate adjustment. Such a credulous adoption may guide the rate adjustments with the wrong CiD, resulting in performance degradation. This is why θ-

PowerTCP fails to maintain high performance under complex scenarios, as detailed in [9] and our results in Table 1 and Fig. 11. Therefore, delay-based CCs should judge the trustworthiness of CiD before utilizing it to guide rate adjustments, as shown in Fig. 1. Re-architecting delay-based CCs by adding the CiD trustworthiness judgement, CiD could be flexibly adopted such that the abundant information of delay signals is fully and correctly utilized.

3.2 CiD-sensitive Architecture

We propose the CiD-sensitive architecture for delay-based CCs, as shown in the red part of Fig. 1. Specifically, the rate adjustment is driven by both the congestion states and CiD trustworthiness. Meanwhile, instead of focusing solely on producing proper sending rates to react to current congestion, these rules also enhance the subsequent CiD trustworthiness. In detail, when CiD is trustworthy, rate adjustment rules should leverage the abundant congestion information to efficiently react to the current congestion and ensure the subsequent trustworthy CiD. Conversely, when CiD is untrustworthy, the rate adjustments can't rely on the explicit congestion intensity provided by CiD. Luckily, the rough direction provided by CiD is sufficient for reacting to congestion and guiding the system towards the state with trustworthy CiD. This is because the system is far from the desired destination, when the CiD is untrustworthy under DACs, including link under-utilization, large queues, or turbulence.

Although the above idea is reasonable, the following challenges remain in the CiD-sensitive architecture.

Distinction of DACs. It is the basis of identifying CiD trustworthiness. Although DAC 1 and DAC 2 can be distinguished by monitoring queue length, the distinction of DAC 3 is challenging. Firstly, the turbulence is hard to quantify. This is because turbulence mainly refers to how drastic and frequent the variations of the bottleneck queue are. However, degrees of both the drastic and frequent are hard to quantify. In detail, when the queue length is small, a small queue variation can be identified as turbulence. Conversely, such a small variation may not be considered turbulence when the queue length is relatively large. Secondly, the turbulence might be overlooked. This is because the change points in a turbulent queue are easily skipped with sparse measurement of delay signals. Ideally, if a flow could frequently receive delay signals, the change points would be fully captured. But in reality, the frequency of receiving delay signals is proportional to the allocated bandwidth, which is directly associated with the degree of fairness, bandwidth C, and the number of flows N sharing the bottleneck. Therefore, the frequency of receiving delay signals may be insufficient to distinguish DAC 3 with small C or large N. In total, the distinction of DAC 3 is challenging.

Making rate adjustments react to congestion and enhance CiD trustworthiness simultaneously. This is challenging because it intensifies the following trade-off issues of CC.

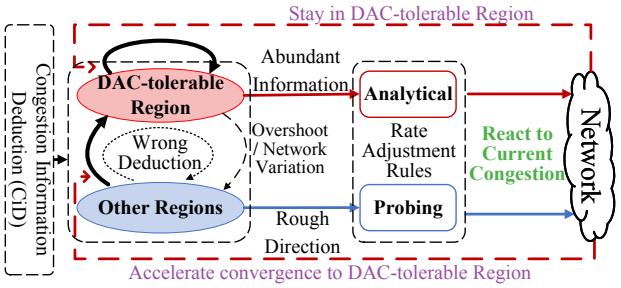


Figure 3: CiD-sensitive Congestion Control.

First, it's hard to make the trade-off between the transience and equilibrium without analytical rate adjustments as highlighted in [40]. This trade-off is intensified because the fast transience is important for the reaction to congestion, while the equilibrium is important to avoid DAC 3 for CiD trustworthiness. However, when the CiD is untrustworthy, analytical rate adjustments may fail as discussed in §3.1. Even when the CiD is trustworthy, the risk of pushing the system into a state with untrustworthy CiD would be increased due to the fast analytical rate adjustment. Second, it's hard for CC to make the trade-off between fairness and efficiency [21, 33]. This trade-off is also intensified as the fair bandwidth allocation becomes more important to avoid DAC 3 for trustworthy CiD.

4 Design

4.1 Basic Idea

Following the CiD-sensitive architecture, we develop the CiD-sensitive CC algorithm (CCC). As shown in Fig. 3, CCC judges CiD trustworthiness by identifying DAC-tolerable regions, besides judging the congestion state. Based on the CiD trustworthiness and the congestion state, different rate adjustment rules are applied in different regions. These rules also enhance subsequent CiD trustworthiness, instead of focusing solely on properly reacting to current congestion.

Specifically, to address the challenge of distinguishing DACs, CCC demarcates regions according to the impacts of DACs. In DAC-tolerable regions, the impact of DACs is tiny and can be tolerated. Thus, the CiD can be considered trustworthy, and the abundant information can be utilized for rate adjustment. Staying in DAC-tolerable regions is therefore a goal of CCC. In the other regions, the impact of DACs is relatively notable. Thus, the CiD is not entirely trustworthy, and only the rough direction from CiD will be utilized. Therefore, another goal of CCC is to quickly escape from such regions to DAC-tolerable regions. Meanwhile, the above regions are further divided into rate-increasing or rate-decreasing regions based on congestion criteria. Overall, instead of manually distinguishing all potential DACs, CCC identifies regions with high risks of DACs and employs rate adjustments that are insensitive to untrustworthy CiD in such regions.

For contribution to both congestion reaction and trustworthy CiD, i.e., relieving the two pairs of trade-off, CCC adopts either analytical or probing rate adjustment rules in different regions. On the one hand, the analytical rules aim to utilize the abundant information provided by trustworthy CiD for the proper reaction to congestion. Simultaneously, the analytical rules are elaborately designed to avoid extremely fast adjustments such that the system is forced to stay in DAC-tolerable regions to keep CiD trustworthy. On the other hand, the probing rules utilize the rough direction provided by the untrustworthy CiD to guide the congestion response. Meanwhile, such rules also facilitate transitions toward DAC-tolerable regions, where the rate adjustments are handed over to the analytical rules, as early as possible. In this way, a better trade-off between transience and equilibrium can be achieved. More specifically, during the transition process, the probing rules should avoid link under-utilization and pursue a low bottleneck queue. Meanwhile, the transition must remain smooth near region boundaries to prevent skipping over DAC-tolerable regions. At the same time, during this transition, the probing rules should ensure fairness to alleviate missing change points of delay, which causes untrustworthy CiD.

In sum, CCC employs different rate adjustment rules across different regions to cope with the impacts of DACs. In particular, the analytical rules and probing rules will cooperate with each other for both good congestion state and trustworthy CiD. In this way, the abundant information of delay signals can be fully utilized to improve network performance.

4.2 Overview

As shown in Fig. 4, the regions are divided with the help of both the congestion criteria $\delta(k)$ (see §4.3.2) and the inherent system inertia. Specifically, k denotes the sequence number of the acknowledgment (ACK) received by the sender, the horizontal axis $q(k)$ is the bottleneck queueing delay, and the vertical axis $q_v(k) \triangleq [R(k)-C]/C$ denotes the queuing delay variation. Here $R(k)$ is the aggregated arrival rate of flows sharing the bottleneck link. In Fig. 4, because $R(k) \geq 0$, the vertical axis $q_v(k) \geq -1$. Moreover, when $q_v(k) > 0$, $q(k)$ will spontaneously increase. The intrinsic tendency of the bottleneck link state to vary spontaneously is referred to as system inertia. Due to the system inertia, with the increase of time k, the system state $[q(k), q_v(k)]$ will move towards the right, as shown by the dashed green arrows in Fig. 4. Conversely, when $q_v(k) < 0$, $q(k)$ will spontaneously decrease and the system state will correspondingly move towards the left. Meanwhile, the congestion criterion $\delta(k)$ is used to judge congestion state, i.e., X^i regions for rate increasing and X^d for rate decreasing. According to the above system inertia and congestion criterion, two types of regions are divided. One is DAC-tolerable regions, i.e., A regions, where the analytical rules will be employed without considering the impact of *DACs* as discussed in §4.3.3. The other is H regions, where

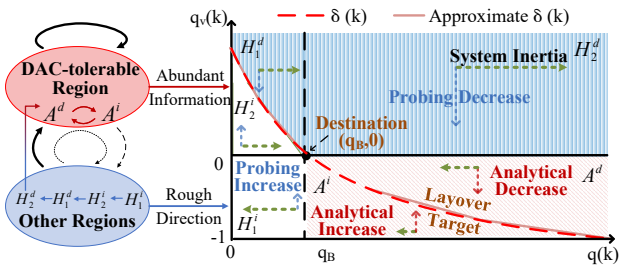


Figure 4: The region division of CCC. Dashed green arrows indicate the system inertia. Dotted blue arrows indicate the probing rate decrease executed in the regions H_1^d and H_2^d, as well as the probing rate increase executed in the regions H_1^i and H_2^i. Dotted red arrows indicate the analytical rate adjustment executed in the regions A^d and A^i.

the probing rules will be employed. In detail, as shown in Fig. 4, regions H_1^d and H_2^d, corresponding to drastic queue variations and large queuing, respectively, are probably affected by ***DAC 3*** and ***DAC 2***. Similarly, regions H_1^i and H_2^i, characterized by low queuing delay and small growth rate of the queuing, face the risk of link utilization loss (***DAC 1***). Moreover, because the queue variations tend to be relatively large in these two regions due to the small queue, H_1^i and H_2^i are also susceptible to ***DAC 3***. In other words, H regions have a large probability for the occurrence of the 3 representative DACs, which cover different bottleneck load states and the most common delay metric. Correspondingly, the demands for distinguishing DACs are replaced by positioning regions.

In different regions, different rate adjustment rules are specified for different purposes in CCC. On the one hand, as illustrated by the dotted red arrows in Fig. 4, the analytical rate adjustment rules lead the system to move along the layover targets, i.e., switching between A^d and A^i. Here the layover targets refer to the points in line $\delta(k) = 0$, which are the intermediate targets of each rate adjustment step in A regions. In this way, CCC would relieve the trade-off between transience and equilibrium, making itself stay in DAC-tolerable regions to maintain the trustworthy CiD while smoothly converging to the final destination $(q_B, 0)$. Here q_B is the expected queuing delay, which is set according to the delay requirement. On the other hand, as illustrated by the dotted blue arrows in Fig. 4, probing rate adjustment rules aggressively suppress congestion and occupy available bandwidth, as well as become moderate adaptively to avoid turbulence before approaching regions A^d and A^i (§4.3.4). In this way, the system state would be rapidly driven into A regions along the transition path $H_1^i \rightarrow H_2^i \rightarrow H_1^d \rightarrow H_2^d \rightarrow A^d$. Moreover, the independent reference rate $Irr(k)$ of each flow is invented for relieving the trade-off between efficiency and fairness during the converging process to A regions (§4.3.2 and §4.3.4).

In sum, CCC would drive the system state transit from H regions to A regions and finally reach the destination $(q_B, 0)$

as shown in Fig. 4, corresponding to both trustworthy CiD and good congestion state. Specifically, the rate adjustment rules can be mathematically demonstrated as follows.

$$r(k+1) \leftarrow r(k) + u(k)$$

$$u(k) = \begin{cases} \dfrac{0.5\log_2 \frac{q_B}{q(k)+Itv\cdot q_v(k)} - q_v(k)}{Itv} \cdot \dfrac{M\cdot Itv}{RTT_{min}}, & in\ A^i \\ S\cdot w_{ai}\cdot[v_{ai}\cdot f(k)\cdot \dfrac{Itv}{RTT_{min}}], & in\ H_1^i, H_2^i \\ r(k)\cdot w_{md}\cdot[f(k)\cdot \dfrac{Itv}{RTT_{min}}], & in\ A^d, H_1^d, H_2^d \end{cases} \quad (1)$$

Here $r(k)$ is the sending rate, which is updated on receiving the k^{th} *ACK*. Moreover, $u(k)$ stands for the rate adjustment step, calculated via either the analytical rate adjustment rules based on the sliding mode theory [61] or the probing rate adjustment rules in different regions, as shown in §4.3. Furthermore, M stands for the packet size, and q_B is the expected delay. In contrast, the basis S, the weights w_{ai}, w_{md}, and factors v_{ai}, $f(k)$, and $\frac{Itv}{RTT_{min}}$ are all adaptive updated coefficients (§4.3.4). In addition, the rate adjustment rules in the region A^d are simplified to probing rules instead of analytical rules to reduce the computation overhead as discussed in §4.3.3.

With equation (1), CCC would reach curve $\delta(k)=0$ and then converge to the destination $(q_B, 0)$ in zigzag along the layover targets in curve $\delta(k)=0$. Moreover, although equation (1) seems complex, it can be deployed with a small number of basic mathematical operations (§5). In other words, CCC has a low computation and storage overhead such that it can work well with a huge number of flows.

4.3 Details of CCC

4.3.1 Congestion Signals

When the sender receives the k^{th} *ACK*, the measured round-trip time is recorded as $RTT(k)$. The propagation delay RTT_{min} can be regarded as a known value [9, 36, 65]. Accordingly, the queuing delay $q(k)$ can be

$$q(k) = RTT(k) - RTT_{min}. \quad (2)$$

Moreover, CCC records the timestamp $t(k)$ when the k^{th} *ACK* is received and uses it to calculate the time interval Itv between the latest two *ACK*s, namely

$$Itv(k) = t(k) - t(k-1) \quad (3)$$

Accordingly, the delay gradient $q_v(k)$ could be deduced according to the variation of RTT and Itv as follows.

$$q_v(k) = \gamma\cdot q_v(k-1) + (1-\gamma)\cdot \frac{RTT(k)-RTT(k-1)}{Itv(k)} \quad (4)$$

Here the Exponentially Weighted Moving Average (EWMA) is employed to smooth the delay gradient, which can be configured similar to [44]. By the way, although the CiD might be untrustworthy due to DACs, the measurement of delay signals can be considered precise, referring to existing works [32, 44].

4.3.2 Congestion Criteria

The congestion criterion $\delta(k)$, shown below, serves as the boundary between regions in Fig. 4 and determines the destination point $[q(k), q_v(k)] = (q_B, 0)$.

$$\delta(k) = 0.5\log_2 \frac{q_B}{q(k)} - q_v(k) \quad (5)$$

Wherein, the base of the logarithmic function is set to 2 for the convenience of implementation as discussed in §5. In A regions, $\delta(k)=0$ performs as layover targets of analytical rate adjustment, and their destination $(q_B, 0)$ corresponds to full link utilization with low queuing. Under the analytical rate adjustment, the CC system would move along these layover targets to the destination point with the sliding mode motion [61] (§4.3.3). In H regions, $\delta(k)$ guides the direction of probing rate adjustment. When the queuing delay $q(k)$ is very small, the corresponding $q_v(k)$ on $\delta(k)=0$ is very large. Accordingly, the sending rate can be increased to the large $q_v(k)$ for rapid occupancy of available bandwidth. Similarly, when the queuing delay $q(k)$ is very large, the sending rate can be decreased to the small $q_v(k)$ for congestion suppression.

Moreover, as a complementary of $\delta(k)$, an independent reference rate $Irr(k)$ for each flow is set to accelerate the fairness convergence. Specifically, $Irr(k)$ is defined as

$$Irr(k) = max\{0, -\frac{R_l}{q_B}q(k) + R_l\} \quad (6)$$

where R_l is the line rate of NIC. Because flows sharing the same bottleneck link observe a common $q(k)$, $Irr(k)$ of each flow can be considered the same. $Irr(k) > r(k)$ suggests that the current sending rate is too low to be fair with others and vice versa. Note that $Irr(k)$ can not determine the direction of rate adjustment, which is only determined by $\delta(k)$. The specific operations bind with the fairness factor $f(k)$ in §4.3.4.

4.3.3 Analytical Rate Adjustment Rules

Analytical rate adjustment rules (lines 6 ∼ 7 of Algorithm 1) are conducted in the DAC-tolerable regions, i.e., A regions. On the one hand, CCC adjusts the sending rate to quickly converge to the destination for reacting to the current congestion. On the other hand, CCC ensures the system remains within A regions for enhancing the subsequent CiD trustworthiness.

In A^i region, the link is fully utilized, i.e., without *DAC 1*. At the same time, A^i region is bounded by the curve $\delta(k)=0$ and $q_v(k)=-1$, making $q(k)$ and $|q_v(k)|$ relatively small as shown in Fig. 4, i.e., the corresponding queue length and its variation are small. Therefore, the impact of *DAC 2* and *DAC 3* is limited. Moreover, the analytical rate adjustments drive the system into the sliding mode motion, tolerating the limited

impact of all *DACs* [61]. Under the sliding mode motion, the CC system converges to the destination $(q_B, 0)$ in zigzag along the layover targets $\delta(k)$. Specifically, the system state crosses the sliding mode curve $\delta(k)$ in each step. Mathematically, the rate adjustment should fulfill the formula.

$$\delta(k)\delta(k+1) \leq 0 \tag{7}$$

Accordingly, we develop the following analytical rules.

$$u(k) = \frac{0.5\log_2 \frac{q_B}{q(k)+Itv\cdot q_v(k)} - q_v(k)}{Itv} \cdot \frac{M \cdot Itv}{RTT_{min}},\ in\ A^i \tag{8}$$

As proved in Appendix A, such rules fulfill inequality (7) and ensure that the system crosses the curve $\delta(k)$ upwards.

In the A^d region, the system will spontaneously move leftwards to the curve $\delta(k)$ with the system inertia. Therefore, probing rate adjustment rules are inherited to provide a rough direction for the spontaneous crossing of $\delta(k)$. As discussed in §3.2, the impact of untrustworthy CiD by DACs could be resisted by the rough direction, i.e., A^d is DAC-tolerable. Moreover, the probing rate adjustment step conducted in A^d will decrease to 0 as q_B is approached. Therefore, the system inertia will take the majority, avoiding overshoot and unintended departure from DAC-tolerable regions.

Moreover, CCC adds a smooth factor Itv/RTT_{min} to ensure that the sliding mode motion occurs per RTT rather than per rate adjustment step. It helps to cope with the control delay, i.e., one RTT delay for the updated sending rate takes effect, according to the sliding mode theory [61].

In sum, with the rule (8) and the system inertia, the sufficient condition for sliding mode motion is met [61].

4.3.4 Probing Rate Adjustment Rules

The probing rate adjustment in H regions rapidly drives the system to enter A regions with good congestion reaction, as illustrated in lines 4, 5, and 8 $\sim$ 16 of Algorithm 1. It consists of the basis S, the weights w_{ai} and w_{md} of AIMD, and the factors v_{ai}, $f(k)$, and $\frac{Itv}{RTT_{min}}$. All factors are self-updating.

First, the adaptive basis S is utilized to indicate the magnitude order of rate adjustment granularity. To resist the impact of untrustworthy CiD, S should be roughly estimated. Specifically, S is only updated and used in regions H_1^i and H_2^i. CCC records the latest 5 sending rates if $|q_v(k)| < 0.05$, where the aggregate sending rate is close to C. The value of S is updated with the average value of the records when leaving A regions. In this way, S of each flow would converge to C/N when CCC is fair. Hence, S can reflect the network environment and serve as a proper rate adjustment granularity.

Second, the weights of AIMD are adaptively regulated to make the system fast and smoothly enter the A regions for analytical rate adjustment. Specifically, $w_{ai} = 0.5\frac{Irr(k)}{R_l}$ in H_1^i, and becomes $0.1\frac{Irr(k)}{R_l}$ in H_2^i because the points in H_2^i will be spontaneously pushed to H_1^d by system inertia. $Irr(k)$ helps mitigate the risks of link under-utilization by being inversely proportional to $q(k)$. Furthermore, w_{ai} is multiplied with the factor $max(0, 1-2\frac{r(k)}{R_l})$ in H_1^i to enhance fairness. In this way, flows with larger sending rates will ramp up slower, and stop ramping up when their sending rate exceeds $0.5R_l$ to wait for others. In regions H_1^d and H_2^d, $w_{md} = -0.25min(\frac{|q(k)-q_B|}{q_B}, 1)$. In this way, w_{md} decreases with the increasing intensity of congestion to fast drain the bottleneck queue. Moreover, CCC would enter A^d from H_2^d instead of skipping A^d, because w_{md} approaches 0 with $q(k)$ close to q_B. In total, the AIMD style rate adjustment smoothly guides the system moving along $H_1^i \to H_2^i \to H_1^d \to H_2^d \to H_1^i \to A^d$, because rate adjustment only affects the vertical movement and horizontal movement only follows the dashed green arrows in Fig. 4.

Moreover, several factors are designed to cope with DACs.
Adaptive accelerating factor v_{ai} helps to fast occupy available bandwidth when deductions are wrong in **DAC 1**. In detail, v_{ai} is updated as 2^n, where n is the number of consecutive $RTTs$ at the link under-utilization state. The link under-utilization state is judged by whether $q(k)$ is close to 0 within an error margin of 5% of RTT_{min}. The exponential growth of the rate facilitates rapid bandwidth occupancy.
Smoothing factor $\frac{Itv}{RTT_{min}}$ helps to resist the impact of postponed deductions in **DAC 2**. Specifically, the smoothing factor $\frac{Itv}{RTT_{min}}$ integrates to 1 over one RTT_{min}, ensuring that the rate update reaches the intended adjustment gradually within a single RTT_{min}, similar to [32]. In this way, the impact of a single postponed CiD on $u(k)$ is diluted. In other words, a postponed deduction would not sharply change the system state and its impact would be corrected during the iteration of probing rate adjustment. Meanwhile, CCC limits $\frac{Itv}{RTT_{min}}$ within the range of $(0, 1]$ such that $u(k)$ would not be amplified by a large Itv. Moreover, this factor also helps to resist the impact of aggregated or missing ACKs. Specifically, when ACKs are aggregated, the Itv between these ACKs would be small and not trigger excessive response. In contrast, when one ACK is lost, the Itv between its adjacent ACKs will expand and the corresponding $u(k)$ will be increased to compensate for the missing rate adjustment of the lost ACKs.
Fairness factor $f(k)$ helps the system converge to fairness. This ensures that the sampling frequency of each flow would not be too low and helps alleviate missing change points during turbulence, thus avoiding overlooked turbulence in **DAC 3** that could lead to incorrect deductions. Specifically, $f(k)$ is defined according to both $\delta(k)$ and $Irr(k)$.

$$f(k) = \begin{cases} 0, & [Irr(k) - r(k)] \cdot \delta(k) < 0 \\ 1, & [Irr(k) - r(k)] \cdot \delta(k) > 0 \end{cases} \tag{9}$$

In this way, when $\delta(k)$ suggests a rate increase, CCC stops rate adjustment of flows with sending rates larger than $Irr(k)$. Meanwhile, flows with sending rates smaller than $Irr(k)$ can continue to speed up. The opposite occurs when $\delta(k)$ suggests a rate decrease. Consequently, the gap among unfair flows is

Algorithm 1 CCC Algorithm ▷ 1÷, 17 × operations

function ONACKARRIVAL( )
  Update rtt, $q(k)$, $q_v(k)$, Itv, $\delta(k)$, and $Irr(k)$ ▷ 8 −, 6 ×, and 1 ÷ ops for $1/Itv$
  $Region \leftarrow$ REGIONJUDGEMENT($q(k)$, $q_v(k)$, $\delta(k)$)
  $f(k) \leftarrow \delta(k) \cdot (Irr(k) - r(k)) > 0?1 : 0$ ▷ *Update fairness factor.* 1 − and 1 × ops
  $S \leftarrow$ UPDATEBASIS($q_v(k)$, $r(k)$) ▷ 3 + ops
  if $Region == A^i$ **then** ▷ *Analytical rules*
    $u(k) \leftarrow \frac{0.5[log_2 q_B - log_2(q(k)+Itv \cdot q_v(k)] - q_v(k)}{RTT_{min}}$ ▷ 2 −, 1 +, and 4 × ops, pre-calculated $1/RTT_{min}$
  else if $Region == H_1^i, H_2^i$ **then** ▷ *Probing increase*
    $w_{ai} \leftarrow 0.5Irr(k)/R_l \cdot max(0, 1 - 2\frac{r(k)}{R_l})$ for H_1^i
    $w_{ai} \leftarrow 0.1Irr(k)$ for H_2^i ▷ 1 −, 5 × ops, $1/R_l$ is constant and pre-calculated
    Update v_{ai} ▷ At most double 1 time per RTT
    $u(k) \leftarrow \frac{Itv}{RTT_{min}} \cdot v_{ai} \cdot w_{ai} \cdot f(k) \cdot S$ ▷ 5 × ops
  else ▷ *Probing decrease*
    $w_{md} \leftarrow -0.25min(\frac{|q(k)-q_B|}{q_B}, 1)$ ▷ 1 − and 2 × ops, $1/q_B$ is pre-calculated constant
    $u(k) \leftarrow \frac{Itv}{RTT_{min}} \cdot w_{md} \cdot f(k) \cdot r(k)$ ▷ 4 × ops
  $r(k+1) \leftarrow r(k) + u(k)$ ▷ 1 + operation

effectively reduced while the system keeps approaching $\delta(k)$.

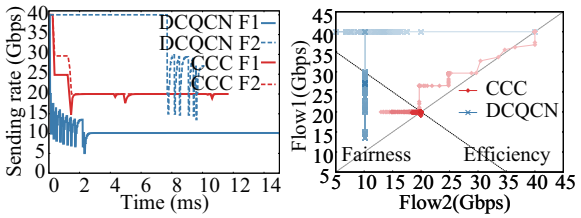
5 Implementation

We implement CCC using DPDK. Specifically, we just need to develop a CCC sender, including modules of delay parsing, rate calculation, and packet sending. First, the delay parsing module is similar to TIMELY [44] and SWIFT [32], where NIC timestamps are required. In detail, when a packet is posted to the NIC, the sender will record a timestamp on its header. Afterward, when the sender receives an ACK, it will calculate the end-to-end delay using the current timestamp and the recorded timestamp in the header. With the measured delay, the delay gradient can be calculated by the difference between two adjacent delay signals. Second, the rate calculation module is built according to the logic of CCC. In particular, $\delta(k)$ is approximated to reduce the computation overhead in the implementation of CCC. As shown in Fig. 4, $\delta(k)$ is replaced by segmented functions $\hat{\delta}(k)$ as follows. Let $i = \lfloor \log_2 q(k) \rfloor$, equation (5) can be approximated by

$$\hat{\delta}(k) = 0.5(\log_2 q_B - i - \frac{q(k) - 2^i}{2^i}) - q_v(k) \qquad (10)$$

Wherein the $\log_2 q_B$ is a preconfigured constant and i can be calculated by bit-shift operation. Equation (8) could also be simplified similarly. In total, the design of logarithmic operations in CCC facilitates optimization of subsequent computations. Moreover, the division operation of $1/Itv$ is calculated only once and reused at each time of receiving ACK, while $1/RTT_{min}$ is a pre-calculated constant. In this way, all CCC calculations on each ACK arrival can be completed in a limited number of operations. The numbers of operations are $\#_{\div} = 1$, $\#_{\times} \leq 17$, $\#_{-,+} \leq 11$, as summarized in Algorithm 1. Meanwhile, the necessary storage of the system states and intermediate variables in primitive types costs less than 1000bits per flow. Moreover, the storage overhead can be further reduced to 250bits via value-range-based compression, which is acceptable even in the face of a large number of flows. Third, a typical DPDK packet-sending module is built. Packets are sent with intervals calculated according to the output sending rate $r(k)$. Meanwhile, the sending window $r(k) \cdot RTT_{min}$ also limits the number of inflights similar to [36].

In sum, CCC has low computational and storage overhead without special demands on new devices or features.



(a) Evolution of Rate (b) Fairness v.s. efficiency. The darker the color, the later the time.

Figure 5: Experimental results on the testbed.

6 Evaluation

The performance of CCC is evaluated through small-scale testbed experiments and large-scale NS3 [3–5] simulations.

6.1 Setup

The experimental testbed is composed of 3 hosts with NIC of Mellanox ConnectX-5 and a Tofino switch. Wherein, 2 hosts are senders and the other is the receiver. The network topology in NS3 simulation is a Fat-tree [10] with 250 hosts or a simple many-to-one topology. Similar to [9, 49], the link bandwidth between arbitrary two switches is 100Gbps, and the link bandwidth between a host and a switch is 25Gbps. Moreover, the maximum RTT is 20μs for both Fat-tree and many-to-one topologies, and the buffer size of switches is set to 4MB. Realistic workloads, including WebSearch [11], FB_Hadoop [48], and AlexNet [31], and sets of synthetic workloads are used. PFC is enabled during the tests with realistic workloads to evaluate the performance of CCC in the real world.

In addition, CCC sets the queuing delay of destination $q_B = 5\mu s$ and the EWMA weighting factor $\gamma = 0.2$ by default.

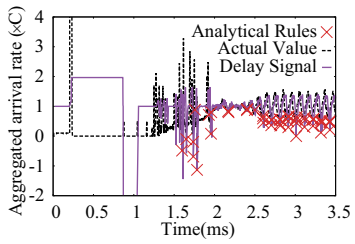


Figure 6: The timing of CCC conducting analytical rate adjustment.

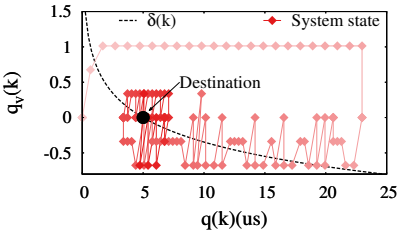


Figure 7: The evolution of system state $[q(k), q_v(k)]$ in CCC.

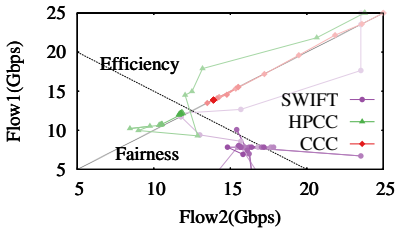
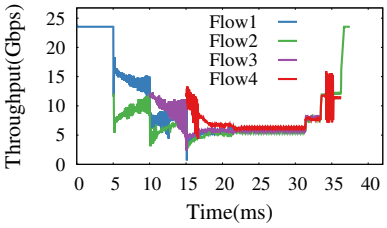
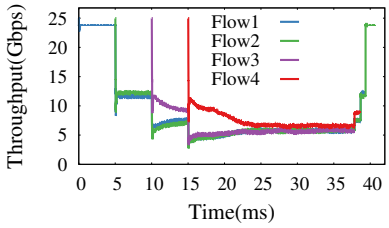


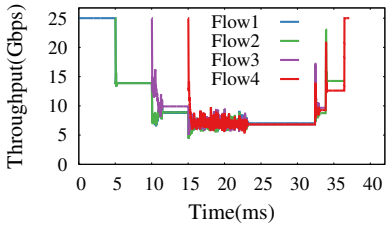
Figure 8: Sending rates of the first two flows.



(a) SWIFT



(b) HPCC



(c) CCC

Figure 9: Fairness with dynamically entering flows.

6.2 Basic Properties

First, we verify the implementation of CCC and its basic properties via experiments on the testbed.

CCC is efficient and fair. Upon experiments in the real world, the queuing delay of the destination is set to $20\mu s$ to counter unpredictable external interference. Let each sender run a long flow concurrently aiming at the same 40Gbps port on the receiver. As shown in Fig. 5(a), CCC rapidly achieves fair bandwidth sharing while maintaining full link utilization. In contrast, though DCQCN achieves full link utilization, it fails to converge to fairness during the test. In other words, CCC has better fairness-efficiency trade-off than DCQCN. This is confirmed by the fairness-efficiency trajectories of CCC and DCQCN shown in Fig. 5(b), where the darker color indicates the later time. CCC quickly converges to the intersection point of the fairness line and the efficiency line within several RTTs. It indicates that CCC ensures fairness between flows while keeping efficient data transmission.

CCC conducts analytical rate adjustment when the CiD is trustworthy. We repeat the simulation in §3.1 using CCC to illustrate the different behavior of CCC under DACs or not. Specifically, the timings of conducting analytical rate adjustments are illustrated. As shown in Fig. 6, CCC occasionally conducts analytical rate adjustment after the huge bottleneck queue is drained out at $1ms$ and before the system leaves from the turbulence at $2.3ms$. This is due to the temporary trustworthy CiDs in the gap between short flows randomly entering. After $2.3ms$, DACs vanish and the deduced values from delay signals are close to the actual value as shown in Fig. 6. Therefore, after $2.3ms$, CiDs are trustworthy and CCC continuously conducts analytical rate adjustments.

CCC quickly converges to destination along the layover targets in $\delta(k) = 0$. We simultaneously initiate two flows targeting a common receiver and record the system state $[q(k), q_v(k)]$ every 1 μs during the convergence to the destination. The evolution of the system state is shown in Fig. 7, where the darker color means the later time and the black point denotes the destination $(q_B, 0)$. Specifically, corresponding to the diagrammatic sketch Fig. 4, Fig. 7 presents the process of CCC transitioning from H_2^i region to H_1^d region, then to H_2^d region, and finally entering into A regions. The process from H to A regions is brief, corresponding to the sparse points before reaching curve $\delta(k) = 0$. This is because the probing rate adjustment of CCC not only responds to congestion but also enables a fast and smooth transition into A regions. Moreover, in A regions, CCC switches between region A^d and region A^i, i.e., performs sliding mode motion along the layover target curve, and converges toward the destination along this curve.

CCC converges to fairness rapidly. We initialize 4 flows with an interval of 5ms in a 4-to-1 topology. The sending rate trajectories of the first two flows are shown in Fig. 8. Specifically, CCC could fast achieve fairness and keep approaching the line of efficiency along the fairness line. It benefits from the rate adjustment rules which take both congestion criteria of aggregated $\delta(k)$ and independent $Irr(k)$ as reference. In contrast, HPCC suffers from slower convergence to fairness, as [27] proposed. SWIFT cannot even converge to fairness within 5 ms, as shown in Fig. 9.

CCC adaptively regulates the basis and the accelerating

	Average link utilization(E_B)		99.9^{th} tail latency(E_L)		Bandwidth acquisition(T_B)	Congestion suppression(T_{L1})	Packet loss ratio(T_{L2})
	① ***S1***	② ***S2***	③ ***S1***	④ ***S2***	⑤ ***S1***	⑥ ***S2***	⑦ ***S2***
DCQCN	51.1%	99.7%	77.4μs	148.8μs	null	414.3ms	**0.41%**
TIMELY	80.9%	92.9%	53.6μs	55.8μs	2760μs	34.6ms	15.13%
SWIFT	95.4%	**99.9%**	32.6μs	61.4μs	980μs	9.3ms	1.58%
HPCC	91.0%	99.6%	**23.5**μs	**35.2**μs	200μs	3.3ms	0.42%
PowerTCP	90.3%	99.7%	**23.5**μs	57.2μs	**100**μs	5.4ms	0.59%
θ-PowerTCP	86.9%	98.8%	25.6μs	70.9μs	1180μs	10.6ms	3.54%
CCC	**95.8%**	99.6%	29.3μs	52.0μs	280μs	**1.4ms**	0.42%

Table 1: CCC performs well across comprehensive indices.

factor in probing rate adjustment according to network conditions. The basis S and accelerating factor v_{ai} under the 4-to-1 fairness scenario are shown in Fig. 10. Specifically, the basis S can roughly reflect the values of C/N after N changes at $5ms$, $10ms$, and so on. With the help of adaptive S, the probing rate adjustment can respond to congestion smoothly and effectively. Although flow 3 gets a larger S during $10 \sim 15ms$, the S of all the flows still well reflects the magnitude of C/N. Moreover, when the bottleneck link is under-utilized due to the leaving of flows around $35ms$, the accelerating factor v_{ai} grows exponentially such that CCC can occupy available bandwidth quickly.

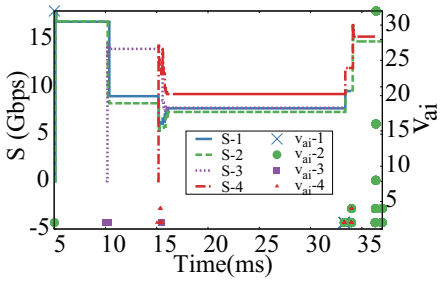


Figure 10: Basis S and accelerating factor v_{ai}.

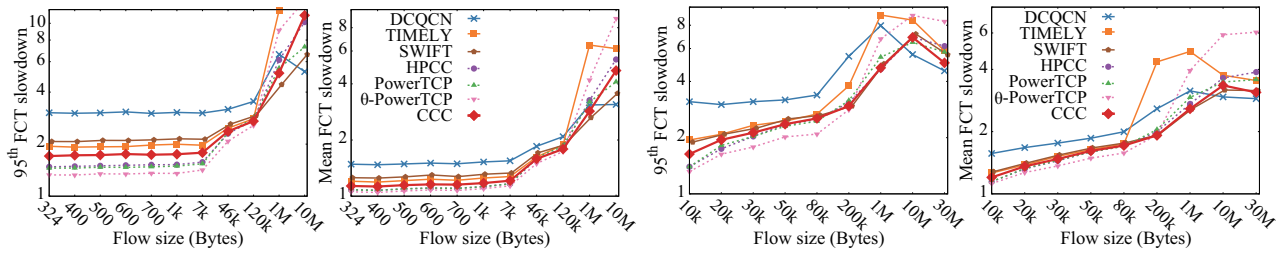
6.3 Comprehensive evaluations

In this section, comprehensive evaluations are constructed to reflect holistic performance under broader scenarios. Specifically, the conflicting performance indices are employed to reflect the primary trade-offs focused by CCs [9, 36, 40, 56]. As shown in Table 1, the conflict between high throughput and low latency is reflected by both the performance indices $\boldsymbol{E_L}$ and $\boldsymbol{E_B}$. Specifically, $\boldsymbol{E_L}$ refers to the 99.9^{th} tail of the RTTs collected after the CC system reaches its equilibrium points, while $\boldsymbol{E_B}$ refers to the average link utilization. Meanwhile, the conflict between equilibrium and transience [40] is reflected by the equilibrium performance indices, including $\boldsymbol{E_B}$ and $\boldsymbol{E_L}$, and the transience performance indices, including $\boldsymbol{T_B}$, $\boldsymbol{T}_{L1}$ and $\boldsymbol{T}_{L2}$. Here $\boldsymbol{T_B}$ refers to the time spent to acquire the available bandwidth until more than 95% of the bottleneck link bandwidth is utilized. $\boldsymbol{T}_{L1}$ refers to the time taken for CC to reduce the delay below $2 \cdot RTT_{min}$, and $\boldsymbol{T}_{L2}$ refers to the packet loss ratio. Note that T_B, T_{L1}, and T_{L2} are complementary on reflecting the transience performance.

Moreover, two complementary scenarios, which contain extreme scenarios and corner cases, are customized under the 250-host Fat-tree topology. The first one (***S1***) is that 5 flows are started concurrently to a common receiver. Wherein, 4 short flows finish at 3ms, and the rest flow is a long flow of size 12.5MB. The second one (***S2***) is that 100 long-lived flows are started concurrently to a common receiver. In this way, heavy concurrency brings overloaded traffic, which is beneficial to high throughput but challenging for low latency. Conversely, light concurrency is beneficial to low latency but challenging for high throughput.

As shown in Table 1, $\boldsymbol{E_B}$ and $\boldsymbol{E_L}$ are measured under the both scenarios ***S1*** and ***S2***. Moreover, $\boldsymbol{T_B}$ is measured only under the scenario ***S1*** to show how quickly CCs can acquire the released bandwidth after a bundle of flows finishes. $\boldsymbol{T}_{L1}$ and $\boldsymbol{T}_{L2}$ are measured only under the scenario ***S2*** to show how CCs handle the huge congestion in the face of the large burst. The results in Table 1 indicate that CCC achieves superior comprehensive performance. Details are as follows.

CCC simultaneously maintains high throughput and low latency without using INT. In the scenario ***S1***, most CCs have a relatively low tail latency (③) but only SWIFT, HPCC, and PowerTCP maintain high average link utilization (①). In contrast, in the scenario ***S2***, CCs except TIMELY achieve nearly full link utilization (②) while only HPCC maintains low tail latency (④). Although θ-PowerTCP conducts analytical rate adjustment, the untrustworthy CiD will mislead it, resulting in sub-optimal performance in ① and ④. In contrast, CCC copes with the impact of untrustworthy CiD via nimbly executing probing and analytical rate adjustments. In



(a) Hadoop workload of Facebook

(b) Web search workload of Microsoft

Figure 11: FCT slowdown, i.e. actual FCT/ideal FCT, over realistic workloads.

this way, CCC can inherit the advantages of the analytical rate adjustment. Consequently, CCC maintains both high link utilization and low tail latency across scenarios ***S*1** and ***S*2**.

CCC rapidly occupies available bandwidth and suppresses congestion. As shown in column ⑤, after 4 short flows leave, PowerTCP occupies spare bandwidth the fastest, and HPCC is the next. Meanwhile, ECN-based and delay-based CCs suffer from slow bandwidth occupancy. DCQCN flow even did not occupy 95% of the available bandwidth until it was completed in scenario ***S*1**, thus its $\boldsymbol{T_B}$ is not recorded. Although θ-PowerTCP is an analytical CC similar to PowerTCP, it suffers from slow bandwidth occupancy too. It is blamed on the untrustworthy CiD in the face of link under-utilization, i.e. **DAC 1**, as discussed in §3.1. In contrast, benefiting from trustworthy CiD as well as the adaptive basis S and accelerating factor v_{ai}, CCC can rapidly occupy spare bandwidth similar to HPCC and PowerTCP, which is 9.9× , 3.5×, and 4.2× faster than TIMELY, SWIFT, and θ-PowerTCP as shown in column ⑤. Moreover, as shown in column ⑥, analytical CCs, i.e., HPCC and PowerTCP, suppress congestion significantly faster than others. However, blamed on the combinations of the per-ACK and per-RTT reactions aimed at avoiding the overreaction [36], the control loop is prolonged, resulting in late responses to congestion [12]. In contrast, CCC uses analytical rules with layover targets to avoid overreaction and thus could conduct per-ACK reactions. Correspondingly, CCC has a smaller control loop, making it respond to congestion in time. Therefore, CCC achieves lower packet loss (⑦) and faster suppresses congestion (⑥) than DCQCN (304.6×), TIMELY (25.4×), SWIFT (6.6×), HPCC (2.5×), PowerTCP (4.0×), θ-PowerTCP (7.8×).

Furthermore, Appendix B provides examples to show how CCC achieves these advantages in detail and how to weigh these advantages via tuning q_B.

6.4 Realistic Workloads

CCC benefits the classic datacenter workloads. The Fat-tree topology which consists of 250 hosts is employed for the large-scale simulation. We test CCC and other CCs using the realistic workloads of FB_Hadoop [48] and Web-Search [11] with 30% load. The results are shown in Fig. 11(a) and (b) respectively, where dashed lines denote INT-based CCs and their variants. Briefly, CCC achieves similar FCT to HPCC and PowerTCP, except for a higher (16.4% in average) 95^{th} tail FCT of short flows (<10KB [9]). Meanwhile, CCC achieves 41.0%∼43.6% and 9.4%∼12.5% lower 95^{th} tail FCT slowdown of short flows compared to DCQCN and TIMELY over FB_Hadoop, as well as 47.6% and 15.9% respectively over Websearch. Moreover, θ-PowerTCP, another delay-based analytical CC, has a similar 95^{th} tail FCT slowdown to HPCC and PowerTCP. However, it fails in the FCT slowdown of long flows (>1MB [9]). This is because θ-PowerTCP conducts analytical rate adjustment even if the CiD is untrustworthy. CCC outperforms θ-PowerTCP with 31.9%∼47.4% and 34.7%∼48.8% lower mean FCT slowdown of long flows over FB_Hadoop and Websearch, respectively. In addition, although SWIFT performs well over Websearch, it suffers from high FCT of short flows over FB_Hadoop.

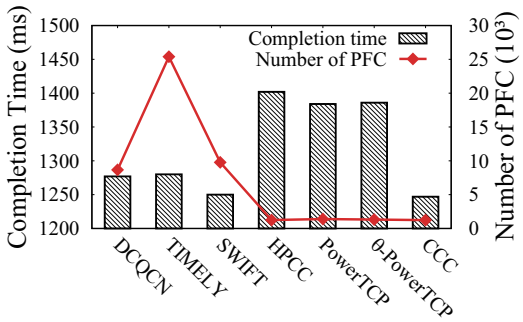


Figure 12: Model training workload of AlexNet.

CCC benefits the model training. We also test these CCs over the workloads of training with AlexNet [31]. We simulate the backpropagation stage in distributed data-parallel training [35] using parameter servers. A Leaf-spine topology with 128 hosts, where 4 of them are parameter servers and the rest are workers, is built. The workloads of model training are challenging due to the mixture of short flows and long flows, as well as the frequent incast from workers

to parameter servers when a new tier of training begins. If a CC cannot handle the incast well, the PFC might cause congestion spreading to postpone the whole iteration or even deadlock. As shown in Fig. 12, CCC triggers the fewest PFC pauses as INT-based CCs, which is 85.7% $\sim$ 95.1% lower than productional CCs, and achieves the lowest completion time for once model training iteration. In other words, CCC can accelerate communication during model training. In contrast, INT-based CCs also maintains high throughput, but their completion times are postponed due to the extra bandwidth overhead for the INT header. Moreover, although ECN-based and the traditional delay-based CCs get low completion times, they cannot handle incast well, triggering PFC frequently.

7 Related Work

Delay-based CCs. According to recent delay-based CCs, e.g., TIMELY [44], SWIFT [32, 51], PrioPlus [66], and MCC [37], delay signals can be precisely measured in datacenters. Benefiting from the fine-grained congestion intensity deduced from delay signals, these CCs achieve good performance. Our work re-architects delay-based CC by adding the CiD trustworthiness judgment. In this way, the abundant information of delay signals can be fully mined to guide rate adjustments.

Delay signals are also widely used in CCs such as Vegas [15], BBR [16], Copa [13], ICC [28], and Themis [39] for the Internet. They follow the traditional architecture and may be enhanced by referring to the CiD-sensitive architecture. Moreover, the CiD-sensitive architecture may also be enhanced with other congestion signals (see Appendix C).

CCs with probing rate adjustments. Probing rate adjustments are usually performed with linear or proportional steps based on the gap between the explicit targets and the current congestion state. Such a gap can be justified according to the existence of ECN, as DCTCP [11], D2TCP [55], and DCQCN [67] do. Delay signals can reflect the gap fine-grained, providing congestion intensity for CCs, e.g., TIMELY [44], DX [34], SWIFT [32, 51], and PrioPlus [66]. However, existing probing rate adjustments only aim to react to the current congestion without considering the trustworthiness of CiD. In contrast, probing rate adjustments of CCC also enhance the subsequent CiD trustworthiness, enabling subsequent analytical rate adjustments for better performance.

CCs with analytical rate adjustments. HPCC [36], PowerTCP [9], Poseidon [56], and EagerCC [42] calculate and adjust proper sending rates in one shot based on the abundant information of INT. In particular, Poseidon pushes each flow to its layover targets in one shot, then iteratively converges to the destination. Similarly, CCC adjusts rates with layover targets. However, instead of setting layover targets for each flow separately for fairness, CCC associates layover targets with the bottleneck state, guaranteeing efficiency at first.

In addition, several works argue for analytically calculating the proper sending rate at the switch and feeding it back to senders, e.g., XCP [29] and RCP [20]. Similarly, based on the accurate information about the released bandwidth, Bolt switches analytically calculate the number of tokens for adjusting the sending rate [12]. These works require modifying switches for computation. In contrast, CCC is economical as it is a pure end-host solution without modification on switches.

Other schemes. Receiver-driven CCs, including ExpressPass [18], pHost [22], NDP [25], Homa [45], Aeolus [26], Fork [41], and SIRD [46] firstly send requests to the scheduler or receiver, and then send packets based on the allocation from the scheduler or receiver. CiD also performs an important role in guiding the allocation and scheduling in these schemes. Therefore, the idea of taking the CiD trustworthiness into account might also be applied to receiver-driven CCs.

Moreover, learning-based methods are used to design CC algorithms [54] or optimize the configuration of CC [30, 60] in datacenters, and are also widely used in the Internet [6, 19, 38, 43, 58, 59, 62, 63]. Their rate adjustment processes are hard for humans to reason about. In contrast, CCC is interpretable with foreseeable low overhead and deployment cost.

Besides, lots of other flow control, congestion detection, flow scheduling, and buffer management schemes are proposed recently, including GFC [47], BFC [23], BackDraft [50], PCN [17], TCD [64], PPT [53], ABM [7], and Reverie [8]. These works are complementary to CCC.

8 Conclusion

This paper shows that CiD is untrustworthy under DACs and is adopted skeptically or credulously in existing delay-based CC algorithms. To further mine the potential of delay-based CC, we suggest the CiD-sensitive architecture, where the rate adjustment is made based on both the congestion state and the CiD trustworthiness, and aims at reacting to the current congestion as well as the subsequent CiD trustworthiness. In this way, the abundant information of delay signals can be fully utilized. Based on the architecture, we propose CCC, which demarcates regions according to both the congestion criterion and the system inertia, and executes either analytical or probing rate adjustment rules in different regions. Extensive evaluations confirm that CCC performs as well as INT-based CCs with solely delay signals.

Acknowledgements

We appreciate the shepherd Shuang Chen and all the anonymous reviewers for their constructive feedback. This work is partially supported by the National Natural Science Foundation of China under Grant No. 62472451, 62132007, 62372363, U24A20245, and 62132022, the Science and Technology Innovation Program of Hunan Province under Grant No. 2023RC3047, and the High Performance Computing Center of Central South University.

References

[1] Ieee 802.1 qbb - priority-based flow control. https://1.ieee802.org/dcb/802-1qbb/.

[2] Dpdk. 2024. https://www.dpdk.org/.

[3] Hpcc open source code. 2024. https://github.com/alibaba-edu/High-Precision-Congestion-Control.

[4] Network simulator 3. 2024. https://www.nsnam.org/.

[5] Powertcp open source code. 2024. https://github.com/inet-tub/ns3-datacenter.

[6] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the ACM SIGCOMM 2020 Conference*, pages 632–647, 2020.

[7] Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. Abm: Active buffer management in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 36–52, 2022.

[8] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. Reverie: Low pass filter-based switch buffer sharing for datacenters with rdma and tcp traffic. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), Santa Clara, CA*, 2024.

[9] Vamsi Addanki, Oliver Michel, and Stefan Schmid. Powertcp: Pushing the performance limits of datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 51–70, 2022.

[10] 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.

[11] 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*, pages 63–74, 2010.

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

[13] V. Arun and H. Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[14] Wei Bai, Kai Chen, Li Chen, Changhoon Kim, and Haitao Wu. Enabling ecn over generic packet scheduling. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, page 191–204, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the ACM SIGCOMM 1994 Conference*, pages 24–35, 1994.

[16] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

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

[18] 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*, pages 239–252, 2017.

[19] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. Pcc: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.

[20] Nandita Dukkipati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Citeseer, 2008.

[21] Bryce L Ferguson and Jason R Marden. The impact of fairness on performance in congestion networks. In *2021 American control conference (ACC)*, pages 4521–4526. IEEE, 2021.

[22] 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*, pages 1–12, 2015.

[23] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E Anderson. Backpressure flow control. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 779–805, 2022.

[24] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[25] 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*, pages 29–42, 2017.

[26] 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*, pages 422–434, 2020.

[27] Wanchun Jiang, Yujie Hu, Haoyang Li, Kai Wang, Jiawei Huang, and Jianxin Wang. Analysis and improvement of powertcp. In *IEEE/ACM International Symposium on Quality of Service (IWQoS 2024)*, 2024.

[28] Wanchun Jiang, Haoyang Li, Jia Wu, Kai Wang, Fengyuan Ren, and Jianxin Wang. Introspective congestion control for consistent high performance. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 428–445, 2025.

[29] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, 2002.

[30] Shiva Ketabi, Hongkai Chen, Haiwei Dong, and Yashar Ganjali. A deep reinforcement learning framework for optimizing congestion control in data centers. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, 2023.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[32] 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 *Proceedings of the ACM SIGCOMM 2020 Conference*, pages 514–528, 2020.

[33] Adrian Lahanas and Vassilis Tsaoussidis. Exploiting the efficiency and fairness potential of aimd-based congestion avoidance and control. *Computer Networks*, 43(2):227–245, 2003.

[34] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. Accurate latency-based congestion feedback for datacenters. In *12nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 403–415, 2015.

[35] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[36] 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 SIGCOMM 2019 Conference*, 2019.

[37] Yuxuan Li, Zhenghang Ren, Wenxue Li, Xiangzhou Liu, and Kai Chen. Congestion control for ai workloads with message-level signaling. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*, APNET '25, page 59–65, New York, NY, USA, 2025. Association for Computing Machinery.

[38] Xudong Liao, Han Tian, Chaoliang Zeng, Xinchen Wan, and Kai Chen. Astraea: Towards fair and efficient learning-based congestion control. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 99–114, New York, NY, USA, 2024. Association for Computing Machinery.

[39] Feida Liu, Yifan Wang, Jiaqi Zheng, Boxi Liu, and Guihai Chen. Themis: Toward stable near-zero queuing delay in congestion control for low-latency interactive video streaming. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, page 12131–12139, New York, NY, USA, 2025. Association for Computing Machinery.

[40] 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, 2021.

[41] Yuan Liu, Wenxin Li, Yulong Li, Lide Suo, Xuan Gao, Xin Xie, Sheng Chen, Ziqi Fan, Wenyu Qu, and Guyue Liu. Fork: A dual congestion control loop for small and large flows in datacenters. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 446–459, New York, NY, USA, 2025. Association for Computing Machinery.

[42] Yuan Lu, Guoyuan Yuan, Yang Bai, Dezun Dong, and Renjie Zhou. Eagercc: An ultra-low latency congestion control mechanism in datacenter networks. *Computer Networks*, 236:110009, 2023.

[43] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. Multi-objective congestion control. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 218–235, 2022.

[44] 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. 2015.

[45] 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*, pages 221–235, 2018.

[46] Konstantinos Prasopoulos, Ryan Kosta, Edouard Bugnion, and Marios Kogias. SIRD: A Sender-Informed, Receiver-Driven datacenter transport protocol. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 451–471, Philadelphia, PA, April 2025. USENIX Association.

[47] 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*, pages 75–89. 2019.

[48] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *ACM SIGCOMM Computer Communication Review*, 45(4):123–137, aug 2015.

[49] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, et al. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the ACM SIGCOMM 2020 Conference*, pages 735–749, 2020.

[50] Alireza Sanaee, Farbod Shahinfar, Zerina Kapetanovic, Gianni Antichi, Bodhi Priyantha, Brent E Stephens, Srinivas Narayana, Deepak Vasisht, Luyang Li, Anand Sarwate, et al. Backdraft: a lossless virtual switch that prevents the slow receiver problem. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1375–1392, 2022.

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

[52] Ashutosh Srivastava, Fraida Fund, and Shivendra S. Panwar. Coexistence of delay-based tcp congestion control: Challenges and opportunities. In *2022 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pages 43–48, 2022.

[53] Lide Suo, Yiren Pang, Wenxin Li, Renjie Pei, Keqiu Li, Xiulong Liu, Xin He, Yitao Hu, and Guyue Liu. Ppt: A pragmatic transport for datacenters. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 954–969, New York, NY, USA, 2024. Association for Computing Machinery.

[54] Chen Tessler, Yuval Shpigelman, Gal Dalal, Amit Mandelbaum, Doron Haritan Kazakov, Benjamin Fuhrer, Gal Chechik, and Shie Mannor. Reinforcement learning for datacenter congestion control. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):43–46, 2022.

[55] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.

[56] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, TS Eugene Ng, Neal Cardwell, and Nandita Dukkipati. Poseidon: Efficient, robust, and practical

datacenter cc via deployable int. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 255–274, 2023.

[57] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. Topoopt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, 2023.

[58] Keith Winstein and Hari Balakrishnan. Tcp ex machina: computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference*, 2013.

[59] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. Genet: Automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 397–413, New York, NY, USA, 2022. Association for Computing Machinery.

[60] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. Acc: Automatic ecn tuning for high-speed datacenter networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 384–397, 2021.

[61] S. V. Yemel'yanov. Automatic control systems with variable structure. 1970.

[62] Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. ACM SIGCOMM '23, page 255–274, New York, NY, USA, 2023. Association for Computing Machinery.

[63] Junxue Zhang, Chaoliang Zeng, Hong Zhang, Shuihai Hu, and Kai Chen. Liteflow: Towards high-performance adaptive neural networks for kernel datapath. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 414–427, New York, NY, USA, 2022. Association for Computing Machinery.

[64] Yiran Zhang, Yifan Liu, Qingkai Meng, and Fengyuan Ren. Congestion detection in lossless networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 370–383, 2021.

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

[66] Zhaochen Zhang, Feiyang Xue, Keqiang He, Zhimeng Yin, Gianni Antichi, Jiaqi Gao, Yizhi Wang, Rui Ning, Haixin Nan, Xu Zhang, Peirui Cao, Xiaoliang Wang, Wanchun Dou, Guihai Chen, and Chen Tian. Enabling virtual priority in data center congestion control. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 396–412, New York, NY, USA, 2025. Association for Computing Machinery.

[67] 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. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, 2015.

[68] 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.

A Proof of the upwards crossing of curve $\delta(k) = 0$

Proof. Because $\delta(k)$ is always larger than 0 in A^i, inequality (7) could be simplified as follows referring to (5).

$$\delta(k+1) = 0.5\log_2 \frac{q_B}{q(k+1)} - q_v(k+1) \leq 0 \qquad (11)$$

Because $q_v(k) \triangleq [R(k) - C]/C$, we have

$$q_v(k+1) - q_v(k) = \Delta R/C \qquad (12)$$

where $\Delta R \triangleq R(k+1) - R(k)$. ΔR should be the total amount of rate adjustment during the time interval between the k^{th} and the $(k+1)^{th}$ ACK of the current flow. During this interval, all flows sharing the bottleneck would receive $Itv \cdot C/M$ ACKs on average. Thus, the sending rate of all flows will be adjusted for $Itv \cdot C/M$ times. Moreover, observing a common system state $[q(k), q_v(k)]$, $u(k)$ of these flows calculated by (8) using $[q(k), q_v(k)]$ can be approximately considered as the same values. Hence, $\Delta R = u(k) \cdot Itv \cdot C/M$ and thus

$$q_v(k+1) = q_v(k) + Itv \cdot u(k)/M \qquad (13)$$

Substituting (13) into (11), we have

$$u(k) \geq \frac{0.5\log_2 \frac{q_B}{q(k+1)} - q_v(k)}{Itv} \cdot M \qquad (14)$$

Here $q(k+1)$ can be considered equaling to $q(k) + Itv \cdot q_v(k+1)$ because $q_v(k+1)$ is the queue variation during Itv. Substituting $u(k) > 0$ into (13) in region A^i, we have $q(k+1) > q(k) + Itv \cdot q_v(k)$. Therefore, if $u(k)$ satisfies equation (8), the inequalities (14) and (7) hold, i.e., the curve $\delta(k)$ would be crossed upwards. □

B Performance details of CCC

How CCC achieves the advantages in Table 1? To understand how CCC achieves the advantages in Table 1, a composite scenario is constructed. Specifically, a long flow is started at 0*ms* as the background flow, and 7 concurrent flows enter at 1.5*ms* over an 8-to-1 topology. As shown in Fig. 13, when concurrent flows enter, CCC cuts down the queue length as fast as PowerTCP, and faster than SWIFT. Besides, when these flows leave around 3*ms*, CCC and PowerTCP can occupy the available bandwidth immediately, while SWIFT spends about 1*ms* on that. These are the processes of CCC getting fast congestion suppression in column ⑥ and rapid bandwidth occupancy in column ⑤. In addition, CCC maintains both high throughput and low queue length after reaching the equilibrium, which corresponds to its good performance in columns ①-④.

How q_B acts as a knob to weigh tail latency and link utilization? To show how q_B acts as a knob to weigh tail latency and link utilization as well as provide a reference for q_B configuration, we vary q_B from 1μs to 30μs under the scenario ***S1*** in §6.3. As shown in Fig. 14, with the increase of q_B, the 99.9^{th} tail latency keeps increasing as q_B determines the expected queuing delay. In contrast, the link utilization rises at the early stage and tends to be smooth finally. The reason is that a larger q_B leads to a higher bottleneck queue length, which provides sufficient buffering to prevent utilization loss, but it does not further improve utilization beyond saturation.

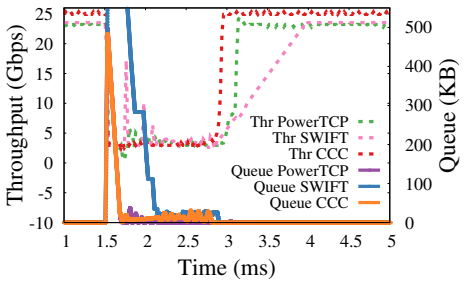


Figure 13: Short flows dynamically enter and leave.

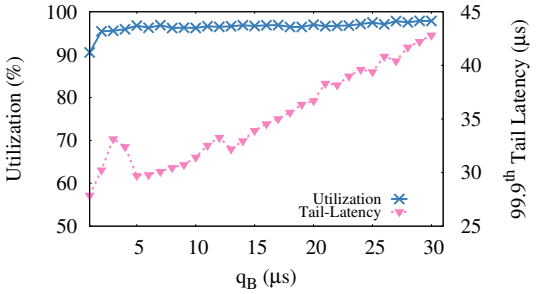


Figure 14: The tail latency and link utilization of CCC with different q_B.

Multi-bottleneck scenarios. End-to-end delay may fail in the face of multiple bottlenecks. CCC flows that pass by multiple bottlenecks might overreact to the congestion, similar to the other delay-based CC schemes [9,32,34,44]. This issue can be mitigated with per-hop delay, as suggested in [56]. However, INT is indispensable for obtaining per-hop delay in [56], which contradicts the original goal of CCC in terms of achieving high performance without introducing INT. Instead, new DACs under the multi-bottleneck scenarios might be explored to enhance CCC in the future.

C CiD trustworthiness of Other Congestion Signals

In addition to delay signals, other congestion signals may also suffer from untrustworthy CiD caused by the ambiguity conditions. According to the exploration of 3 respective ambiguity

conditions for delay signals, i.e., 3 DACs, the root causes can be basically divided into three categories:

Generation. The root cause of DAC 1 corresponds to the generation of delay signals. That is, when the bottleneck link is under-utilized, the delay signals cannot be generated, and the CiD is blind. In other words, an ambiguity condition exists when the congestion signal cannot be generated. Similarly, ECN signals may also suffer from this ambiguity condition once the bottleneck queue length is below the ECN marking threshold, even though the bottleneck link is full-utilized or congested.

Observation. The root cause of DAC 2 is that the delay signals observe the congestion a packet has experienced practically, which has already determined at its enqueue time. Therefore, delay signals will overlook the queued packets that arrive after the packet that carried them, leading to the lag of CiD, especially in the face of a large queue [32]. Things get different with ECN signals. ECN signals utilized by DCQCN [67] and DCTCP [11] are marked at the dequeue time, which can observe the congestion more timely. In other words, dequeue ECN signals may not suffer from this ambiguity condition. However, some enqueue ECN signals and dequeue ECN marked by sojourn time, like TCN [14], may also suffer from this ambiguity condition.

Sampling. The root cause of DAC 3 is that delay signals are sampled at discrete time intervals, leading to the loss of information between samples, especially in the face of turbulence. Similarly, ECN signals marked at discrete packets may also suffer from this ambiguity condition once the congestion varies drastically between packets. Single-bit ECN signals may be less sensitive to turbulence than delay signals, because they only reflect whether the congestion exceeds a threshold. However, multi-bit ECN signals, e.g., ECN signals utilized in [65], may also suffer from this ambiguity condition.

In a word, the ambiguity conditions are related to the properties of congestion signals, including their generation, observation, sampling, and maybe more aspects. Therefore, different types of congestion signals with different properties might be complementary to each other for better resisting the untrustworthy CiD caused by the ambiguity conditions. For example, dequeue ECN signals can be used to assist delay signals in the face of DAC 2. In other words, the CiD-sensitive architecture might be further enhanced by combining different types of congestion signals in the future.