PACC: Proactive and Accurate Congestion Feedback for RDMA Congestion Control

Xiaolong Zhong[1], Jiao Zhang[12*], Yali Zhang[3], Zixuan Guan[3], Zirui Wan[1]
[1]State Key Laboratory of Networking and Switching Technology, BUPT, Beijing 100876, China
[2]Purple Mountain Laboratories, Nanjing 211111, China
[3]Huawei Technologies Co., Ltd, Beijing 100084, China
[1]{xlzhong, jiaozhang, wanzr}@bupt.edu.cn
[3]{zhangyali369, guanzixuan}@huawei.com

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796803

***Abstract*—The rapid upgrade of link speed and the prosperity of new applications in data center networks (DCNs) lead to a rigorous demand for ultra-low latency and high throughput. To mitigate the overhead of traditional software-based packet processing at end-hosts, RDMA (Remote Direct Memory Access) has been widely adopted in DCNs. Particularly, congestion control (CC) mechanisms designed for RDMA have attracted much attention to avoid performance deterioration when packets lose. However, through comprehensive analysis, we found that existing RDMA CC schemes have limitations of a sluggish response to congestion and unawareness of tiny microbursts due to the long end-to-end control loop. In this paper, we propose PACC, a switch-driven RDMA CC algorithm with easy deployability. PACC is driven by PI controller-based computation, threshold-based flow discrimination and weight-based allocation at the switch. It leverages real-time queue length to generate accurate congestion feedback proactively and piggybacks it to the corresponding source without modification to end-hosts. We theoretically analyze the stability and key parameter settings of PACC. Then, we conduct both micro-benchmark and large-scale simulations to evaluate the performance of PACC. The results show that PACC achieves fairness, fast reaction, high throughput, and 6∼69% lower FCT (Flow Completion Time) than DCQCN, TIMELY and HPCC.**

***Index Terms*—Data Center; RDMA; Congestion Control; Switch-driven**

I. Introduction

Over the last decade, the link speed in data centers has grown fast from 10Gbps, 100Gbps to upcoming 200Gbps. This leads to a trend to run various OLDI (OnLine Data-Intensive) applications, distributed computation services and machine learning training tasks on top of the data center fabrics [1]. Both throughput-sensitive and latency-sensitive applications relentlessly demand for ultra-low latency, high throughput and high CPU efficiency simultaneously. As a consequence, cloud providers and data center operators began introducing RDMA technology as the fundamental network component to alleviate the overhead of software-based processing at end-hosts [4].

RoCEv2 (RDMA over Converged Ethernet version 2) is a standard to deploy RDMA for Ethernet-based DCN [2]. Due to limited hardware resources in RNIC (RDMA NIC), RoCEv2 leverages a simple go-back retransmission mechanism to recover dropped packets. This simple packet recovery mechanism causes dramatic throughput degradation even when the packets loss ratio is low [4]. Thus, PFC (Priority Flow Control) [3] is employed in RoCEv2 to ensure the network lossless. However, as a hop-by-hop flow control mechanism working on the coarse-grained port/port+priority level, PFC may lead to potential problems. For instance, issues like Head-of-Line blocking, deadlock and PAUSE frame storm would probably exacerbate the performance of the whole domain. To essentially prevent the activation of PFC, enhanced CC mechanisms for RDMA-enabled DCNs have attracted much attention in recent years.

At present, there are mainly two types of RDMA CC schemes according to the entity playing the pivotal role, sender-driven [4]–[6] and switch-driven [9] solutions. Sender-driven solutions like DCQCN [4], TIMELY [5] and HPCC [6] passively adjust sending rate/window based on the information carried by end-to-end congestion signals. Specifically, the above three use ECN (Explicit Congestion Notification), RTT (Round Trip Time) and INT (In-band Network Telemetry), respectively. Among them, DCQCN is the most typical one and has already been integrated into RNICs as a default mechanism by vendors like Mellanox [7]. However, though widely used in production, DCQCN still encounters various issues relative to environment or operation. Specifically, the end-to-end design makes any unit along the path probably problematic, such as ECN-mark threshold tuning at switches, the adjustment of rate update period, etc. This would bring performance deterioration especially under bursty traffic. Actually, we found that all of these three sender-driven schemes suffer from similar performance problems under specific traffic patterns even though their designs have little in common (Section II).

We investigate that the problems of the above mechanisms roots in the long end-to-end control loop, which gives a remediation for existing network congestion after at least one RTT delay. Sluggish response to congestion leads to deep queues at the bottleneck switch, and subsequent backoffs based on heuristic mechanisms such as AIMD (Additive-Increase Multiplicative-Decrease) or MIMD (Multiplicative-Increase Multiplicative-Decrease) require multiple rounds for convergence after congestion occurs. Moreover, the long control loop becomes blind when it handles flows that finish within one RTT [8]. As a consequence, superfluous and obsolete signals further impede the follow-up transmissions. By reviewing the observations of existing RDMA CC algorithms, we believe that *a short control loop and effective discrimination of flows*

*Corresponding author: Jiao Zhang

is necessary for CC in RDMA-enabled data center networks. But where is the panacea?

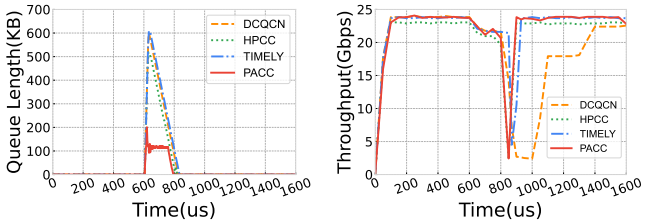
Looking back upon the development of transport protocols in data centers, we found that switch-driven solutions give a satisfactory answer. As the key component of network interconnections, the switch locates inside the fabric and is directly relative with in-network congestions. With the accurate and real-time metrics of the overall network and a flexible control loop range between single-hop and end-to-end delay, *it is possible and desirable to deal with congestion at the switch*. The newly proposed RoCC [9] uses the queue length at switches to explicitly calculate the fair rate of flows and encapsulate it in a packet to feed back to the corresponding senders. However, due to the modifications required on both switches and end-host NICs, the deployability of RoCC declines as most of the data center operators and cloud builders are reluctant to make a forklift upgrade of the whole network [11]. Besides, the tradeoff between computational accuracy and algorithm complexity may also downgrade its performance.

Enlightened by the above insights, in this paper, we propose a switch-driven RDMA CC mechanism with easy deployability and better performance, called PACC. PACC proactively generates accurate congestion feedback at switches and requires no modification to end-hosts. Specifically, PACC is compatible with DCQCN at end-hosts, while the switch takes in the real-time queue length for precise computation based on PI controller and returns back CNPs (Congestion Notification Packets) opportunely to certain sources. Besides, PACC provides good fairness among flows under various traffic patterns with a weight-based feedback allocation scheme.

To offer a reference for easy-to-use system parameter settings, we analyze the stability of PACC based on the control theory. We then evaluate it by conducting micro-benchmarks and large-scale simulations based on NS3 implementation. The results show that PACC can stabilize the queue length around the reference threshold quickly under different scenarios. Moreover, the throughput of both congested and non-congested flows can be guaranteed without sacrificing fairness. The results of large-scale simulations with realistic data center traffic workloads show that PACC achieves 6∼23%, 65∼69%, 9∼12% lower average FCT than DCQCN, TIMELY and HPCC. Besides, the FCT breakdown of PACC outperforms the other three schemes under different ranges of flow sizes.

In summary, our key contributions are:

- We analyze the limitations of state-of-the-art RDMA CC mechanisms and conclude that fast and accurate congestion feedback from the switch is necessary and promising for RDMA CC.
- We propose PACC, which leverages PI controller-based computation, threshold-based flow discrimination and weight-based allocation at switches to achieve proactive, accurate and fair feedback between various flows.
- We theoretically analyze the stability of PI controller-based algorithm used in PACC and give guidance for parameter settings.



(a) Queue length at bottleneck (b) Throughput at bottleneck

Fig. 1: Performance of RDMA congestion control schemes.

- We evaluate PACC with both micro-benchmarks and large-scale simulations compared with DCQCN, TIMELY and HPCC. The evaluation results show that PACC achieves rapid response, better fairness, high throughput, and 6∼69% lower average FCT compared with DCQCN, TIMELY and HPCC.

II. Motivation

As a switch-driven mechanism based on fast congestion feedback, PACC is inspired by the defects of the state-of-the-art RDMA CC algorithms coping with some specific traffic patterns and the paradigm shift from source-driven to core-driven [9]. Specifically, the former focuses on the pros and cons of the final performance of the algorithm, which is closely related to the industry background of rapid upgrades of data center infrastructure (link speed, network devices, etc.) in recent years. The latter concentrates on the design philosophy of the algorithm, aiming to make the algorithm more concise and efficient, and is promising in the context of the latest research results on programmable switches [10]. In general, PACC is motivated by threefold observations.

1) Slow Response to Congestion: The existing RDMA CC mechanisms [4]–[6] use advanced signals including ECN, RTT, and INT to detect congestion and then guide the sender to adjust its transmission behavior in different ways. DCQCN determines its rate adjustment mode based on the receiving event of CNP, while TIMELY and HPCC calculate the sending rate or window size according to the ACK packets returned by the receiver. However, since these signals work on an end-to-end time scale, the long control loop requires at least one RTT for the sender to perceive and react to congestion. In addition, the queue built on the bottleneck switch during the backhaul of congestion signals will also significantly increase network latency, which will further aggravate congestion.

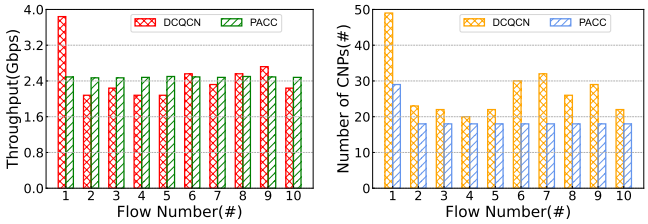
Experiment results: To illustrate the above issue, we conduct an NS3 simulation to compare the performance of the three algorithms with their default parameters in the burst flow scenario. The topology used is a typical FatTree [12], consisting of 16 Core switches, 20 Agg switches, and 20 ToR (Top of Rack) switches, with 16 servers connecting to each ToR switch via 25Gbps links. The links between switches are all 100Gbps with a 1μs propagation delay, which gives a 12μs maximum base RTT.

Two long-lived flows are generated from two servers under the same ToR switch at time 0, and their destinations are two different servers in another ToR switch. After the sending rate of the two converges, we randomly select 9 servers from 3 ToR switches and synchronously send a flow with 64KB data to the same receiver of one long flow at 600μs. Fig. 1(a) and Fig. 1(b) show the queue length and total throughput of the receiver-side ToR switch, respectively. The queue length peak at the bottleneck switch of all the three schemes is around 600KB, and keeps growing about 2-3 RTT (24-36μs) before it begins to drain out, which further slows down the recovery of throughput. Specifically, with a duration of bursts of about 230μs, HPCC and TIMELY take about 60μs (5$\times$ RTT) to saturate the remaining available bandwidth, while DCQCN experiences a persistent link under-utilization about 500μs. Since the burst traffic pattern is dominant in today's data center networks [13], *it is desirable for a CC scheme to shorten the control loop to eliminate congestion promptly*.

2) Inaccurate CNP Generation in DCQCN: In DCQCN, it depends on the receiver to decide whether to feed back CNP to the sender by identifying the ECN-bit in the packets coming from the switch. Therefore, the rate adjustment at the sender side is closely related to the CNP generation event. However, the switch marks passing packets indiscriminately once the queue length at the egress port exceeds the threshold. When multiple flows share a bottleneck link, especially when there are bursty short flows, the generation of CNP based on ECN marking has problems both when congestion exists and disappears. On the one hand, the port level marking makes the number of CNPs fed back to each sender vary greatly, resulting in unfairness between flows. On the other hand, due to the long feedback loop, excessive CNPs may continue to inhibit the subsequent rate increase at the sender, making it difficult for the residual flows to recover to the expected rate timely even after the congestion is relieved.

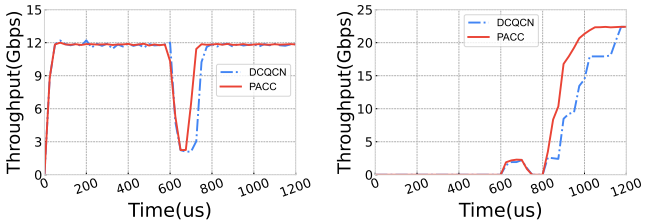
Experiment results: For DCQCN, we use the same settings as the preceding one to repeat the experiment and record flow-level events. Fig. 2(a) and Fig. 2(b) depict per-flow throughput and the number of CNPs received by each sender during congestion. We record the stable value of throughput and the counting of CNPs within 600-800μs. The unfairness of the number of CNPs between long-flow (#1) and short-flows (#2-10) leads to a significant difference in throughput, which is more than 60%. Although the initial settings of all short-flows are exactly the same, the difference of CNPs still results in the fluctuation of throughput between flows. Besides, redundant CNPs cause a considerable long gap between ideal and practical throughput after bursts finish. Intuitively, *an accurate CNP generation scheme is necessary* to fix the drawbacks above.

One might argue that the number of CNPs generated in DCQCN is related to the ECN marking threshold used by the switch so that appropriate parameter settings can solve this problem. However, the adjustment of the ECN threshold is not easy. Recent studies [14]–[16] have proposed solutions from multiple perspectives (machine learning, heuristic algorithms,



(a) Throughput per flow (b) Number of CNPs per flow

Fig. 2: Unfairness caused by inaccurate generation of CNPs in DCQCN.



(a) Throughput of 'long' flow (b) Throughput of 'short-long' flow

Fig. 3: Throughput suppression in congested and non-congested flows.

etc.), but most of them require additional deployment or collaboration with other information. On the contrary, taking the inherent attribute of the switch, queue length, as input, PACC could realize the conversion from queue length to the number of CNPs through PI controller and directly feeds back to the sender to guide accurate rate adjustment, which is fast and straightforward.

3) Unawareness of Tiny Microbursts: Recent study reveals that 60%-90% of production workload flows can be finished in less than one RTT [8], which is also evidenced in reports of request-reply style applications from Google and Facebook [17]. Although the duration is extremely short, the accumulation of deep queues at the switch caused by microbursts will still be regarded as a sign of congestion by the end-to-end CC mechanism. Consequently, the congestion signal from the last RTT will guide the behavior adjustment of all senders, even if it has no contribution to congestion. Correspondingly, the throughput of both the previous bottleneck link and non-congested new flows will be significantly suppressed, just as the congestion still exists.

Experiment results: We also conduct an NS3 simulation to illustrate this point using the same parameter settings in 1) but with different flow attributes. Specifically, we adjust the destinations of two long-lived flows to the same one. After the 30KB bursty flows finish, each sender of microbursts starts a long-lived flow with different receivers (under the same ToR switch of previous long flows) at 800us. Thus, there are two traffic patterns in all (two for 'long-lived' and nine for 'short-long') and we select one arbitrary flow from each to illustrate our observation. Fig. 3(a) and Fig. 3(b) show the throughput

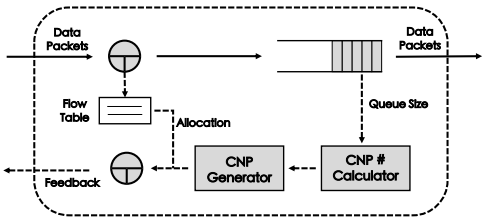


Fig. 4: Overview of PACC design at the switch.

TABLE I: The flow table maintained in PACC switches.

sip/dip	$txBytes$	N_{cong}	N_{cnp}
64bit	32bits	8bits	8bits

of flows with the patterns aforementioned, which reveals the long recovery time for both transmission behaviors ($\sim$150μs for 'long' flows and $\sim$350μs for 'short-long' flows).

The root cause of the phenomenon that the throughput of both congested and non-congested flows is suppressed is that the host is unaware of congestion inside the network. Since the source passively reacts to the congestion signal from the network, once the feedback delay of the congestion signal exceeds the maximum RTT of the network, the rate/window adjustment at the source end will become meaningless and harmful. Considering the switch is directly related to the bottleneck, we believe that *distinguishing and handling tiny microbursts (<1 RTT) at the switch* is of great benefit to the CC algorithm.

Based on the above analysis, we propose PACC, a switch-driven RDMA CC algorithm, which probes and feeds back the network congestion more precisely and timelier, and is compatible with all commercial RDMA NICs that support DCQCN.

III. Design

The key insight of PACC is to achieve prompt, accurate and fair congestion feedback for the congested flows. Nevertheless, there are still two questions to answer before implementation. First, *should we construct a brand-new rate adjustment scheme at the end-host?* Second, *how does the switch meet the goals above with limited resources?* In short, with inherited rate control from DCQCN, PACC combines PI controller-based, threshold-based and weight-based tricks to perform a lightweight but effective CC at the switch.

A. Framework of PACC

PACC mainly modifies switches which will be described in Section III-B in detail. The end-host NICs inherit from DCQCN without any modification for ease of deployment. The original DCQCN algorithm consists of three entities including Reaction Point (RP) at the sender, Congestion Point (CP) at the switch, and Notification Point (NP) at the receiver. Since PACC feeds back CNP directly from the switch to the source, the algorithm at NP can be removed if necessary, and the algorithm of RP remains unchanged. When receiving a CNP, RP updates its current rate (R_C), target rate (R_T) and rate reduction factor α as follows:

$$\begin{cases} R_T = R_C \\ R_C = R_C(1-\alpha/2) \\ \alpha = (1-g)\alpha + g \end{cases} \quad (1)$$

where g is a pre-configured constant between 0 and 1. See [4] for other details.

B. PI Controller-based CNP Generation

Fig. 4 presents an overview of the design of PACC at the switch, which includes three main components: the calculator for CNP number computation, the allocator and the generator for CNPs. Specifically, Algorithm 1 illustrates the overall process of CNP generation.

In order to achieve proactive and accurate fast congestion feedback, PACC maintains a flow table as shown in Table I for every switch port. Each entry includes information for flow record (sip/dip, $txBytes$) and indicators for congestion events (N_{cong}, N_{cnp}), where sip and dip is the source and destination IP address of a flow, respectively. The idea of identifying a flow by sip/dip pair instead of a five-tuple comes from the practical implementation of the RDMA NIC vendors, which is determined by limited hardware resources along with the programming style of RDMA applications [18]. The initialization and update of the flow table are realized by a background thread with a period of T (80μs by default). On receiving a packet, the switch will insert or update the flow table entry (Line 28-29) according to sip/dip. Once the queue length at the egress port exceeds the reference threshold (Q_{th}), the corresponding flow entry of the packet will update the congestion level represented by N_{cong} (Line 31). Furthermore, at the end of the period, all the entries in the flow table other than sip/dip will be set to 0, and the whole entry will be deleted if its $txBytes$ is less than B_{th} to save hardware resources (Line 33).

1) Computation of the Number of CNPs: Similar to the update of the flow table, the number of CNPs (N_{all}) is calculated periodically through a background thread. With a classical PI controller, PACC can find the accurate N_{all} shared by all the congested flows only with the knowledge of queue length at the egress port (Line 12) as follows:

$$N_{all}(t) = \beta_1 \times (Q(t) - Q_{th}) + \beta_2 \times (Q(t) - Q(t-T)) \quad (2)$$

To be specific, in implementation, equation (2) is transformed into the discrete form in Algorithm 1. The proportional term with deviation from the Q_{th} provides an instantaneous response to the error extent from the system steady-state, while the derivative term with the direction of queue change enables a rapid convergence to the equilibrium. Parameter β_1 and β_2 determine the weight of these two terms, resulting in different system stability ranges. Intuitively, PACC will uninterruptedly feed back CNPs to the sources until the current queue length (Q_{cur}) is stable at Q_{th}, which indicates all the congested flows have grabbed their fair share of the bottleneck. Besides, the

algorithm performs a boundary check on the calculated N_{all} to guarantee its validity (Line 14). And at the end of the computation, Q_{old} is set to Q_{cur}.

2) Distinguish and Handle Microbursts: To prevent possible under-throughput caused by micro bursty flows, PACC will not enter the CNP generation phase immediately after calculating N_{all}. It depends on the result of the comparison between the threshold (Q_{burst}) and the average queue length (Q_{avg}) to decide whether the calculated N_{all} will take effect. Q_{avg} is got by another background thread (Line 24) as follows:

$$Q_{avg}(t) = w \times Q(t) + (1 - w) \times Q_{avg}(t - T) \qquad (3)$$

where w is used to perform a weighted averaging on Q_{avg}.

If $Q_{avg} < Q_{burst}$, we assume all the flows are microbursts lasting less than one RTT and skip the follow-up procedures to ensure long-term high throughput. If $Q_{avg} > Q_{burst}$, PACC allocates and generates CNPs for congested flows according to the flow table (Line 17).

3) CNP Allocation Across Flows: The allocation of N_{all} among flows is based on the congestion level each flow experienced. The CNP number of each congested flow (N_{cnp}) is the weighted average of N_{all} based on the number of congested packets (N_{cong}) recorded in its flow table entry (Line 4). The floating-point result is rounded down to ensure the sending rate will not drop dramatically.

4) CNP Generation: The format of CNPs generated in PACC stays unchanged with DCQCN, but the encapsulation and management are controlled by the switch completely. To enable faster convergence and to avoid missing the CNP deadline, PACC feeds back CNPs with high priority. Besides, considering the conflict between practical implementation and original parameter settings in DCQCN, PACC sends one CNP to the corresponding sender every 4μs, which is the de facto standard adopted by most vendors. As a consequence, there is an upper bound of N_{cnp} for an arbitrary sender during T. In particular, PACC maintains a scheduler for each congested flow to realize pacing-like CNP generation and transmission with a fixed interval (Line 6). Actually, the scheduler covers CNP generation in the following two typical scenarios: one is the continuous congestion caused by long-lived flows, where $Q_{avg} > Q_{th}$; the other is the mix of long-lived flows and bursty flows (>1 RTT), where $Q_{burst} < Q_{avg} < Q_{th}$. Benefited from N_{cong} recorded in the flow table, PACC transforms the CNP generation event into a unified table lookup operation, which is general and effective.

Algorithm 1 PI Controller-based CNP Generation

INPUT: Queue_Length Q_{cur}
OUTPUT: Congestion_Nontification_Packet cnp

1: **function** CNP_ALLOCATION_AND_GENERATION(N_{all})
2: $PktAll \leftarrow Sum(N_{cong})$
3: **for** each sip/dip in $fTbl$ **do**
4: $fTbl[sip/dip].N_{cnp} \leftarrow \frac{N_{all} \times fTbl[sip/dip].N_{cong}}{PktAll}$
5: **if** $fTbl[sip/dip].N_{cnp} \neq 0$ **then**
6: SEND_CNP($sip/dip, fTbl[sip/dip].N_{cnp}$)
7: **end if**
8: **end for**
9: **end function**
10:
11: **procedure** CNP_NUMBER_COMPUTATION(Q_{cur})
12: $N_{all} \leftarrow \beta_1 \times (Q_{cur} - Q_{th}) + \beta_2 \times (Q_{cur} - Q_{old})$
13: **if** $N_{all} \leq 0$ **then**
14: $N_{all} \leftarrow 0$
15: **else**
16: **if** $Q_{avg} > Q_{burst}$ **then**
17: CNP_ALLOCATION_AND_GENERATION(N_{all})
18: **end if**
19: **end if**
20: $Q_{old} \leftarrow Q_{cur}$
21: **end procedure**
22:
23: **procedure** COMPUTE_AVERAGE_QUEUE_LENGTH(Q_{cur})
24: $Q_{avg} \leftarrow w \times Q_{cur} + (1 - w) \times Q_{avg}$
25: **end procedure**
26:
27: **procedure** RECEIVE_PACKET(Pkt, Q_{cur})
28: $sip/dip \leftarrow Pkt.SrcIP|Pkt.DstIP$
29: $fTbl[sip/dip].txBytes \leftarrow Add(Pkt.Size)$
30: **if** $Q_{cur} > Q_{th}$ **then**
31: $fTbl[sip/dip].N_{cong} \leftarrow Add(1)$
32: **end if**
33: CLEAR_FLOW_TABLE(T)
34: **end procedure**

C. Overhead and Implementation Choice

The key components of PACC at the switch include: (1) a flow table with flow information (table update and aging), (2) periodic calculation of Q_{avg} and N_{all} (timer-based background thread), (3) CNP allocation across flows (flow table lookup), and (4) CNP generation and transmission back to the sources (scheduler-based event).

Each entry in the flow table takes up 21 Bytes, which is lightweight and affordable for existing commodity switches. Since the computation of N_{avg} and N_{all} lies in different threads and is shared by all the flows, the computation overhead is negligible with multithreading processing. Besides, as the de facto framework for data plane programmability, P4 is supported by most major switch hardware vendors in their ASICs, the implementation of PACC can smoothly transition to programmable switches.

IV. THEORETICAL ANALYSIS

In this section, we analyze the stability of the PI controller-based CNP generation algorithm in PACC based on the control system theory [19].

A. Model Formulation

Considering N long-lived flows traversing a single bottleneck switch port with capacity C, the dynamics of the queue

at the switch could be formulated as follows:

$$\frac{dQ(t)}{dt} = \sum_{i=1}^{N} R_C^i(t-T) - C \tag{4}$$

where T and R_C^i denote the interval of the background thread and the sending rate in the last update period, respectively. Besides, after successive rate decrease controlled by accurate CNPs fed back from the switch, the sending rate of a flow is:

$$R_C^i(t+T) = R_C^i(t) \times (1 - \frac{\alpha^i(t)}{2})^{N_{cnp}^i(t-T)} \tag{5}$$

where $N_{cnp}^i(t)$ is the number of CNP allocated for flow i. Besides, when the system tends to equilibrium, the condition "α is quite small while N_{cnp} is particularly large" is true in most cases [20]. Thus $R_C^i(t+T)$ can be approximated as $R_C^i(t) \times (1 - \frac{\alpha^i(t)}{2} \times N_{cnp}^i(t-T))$, which further leads to the delay differential equation of R_C^i:

$$\frac{dR_C^i(t)}{dt} = -\frac{R_C^i(t)\alpha^i(t)}{2T} N_{cnp}^i(t-T) \tag{6}$$

As for the variation of $N_{all}(t)$ caused by PI adjustment in Algorithm 1, we take bilinear transformation [9] to convert it into a continuous one[1]:

$$\frac{dN_{cnp}^i(t)}{dt} = \frac{\beta_1}{T}[Q(t) - Q_{th}] + (\beta_2 + \frac{\beta_1}{2})\frac{dQ(t)}{dt} \tag{7}$$

where Q_{th} is the expected queue length at steady state.

Equation (4) indicates the queuing process at the switch, while (7) captures the evolution of the indirect control signal. By letting the LHS of equations (4), (6), (7) equal 0, with the assumption that all flows are synchronized and peak simultaneously (which is obvious), it is easily verified that N_{cnp}, α, Q, and R_C do not reach the steady-state until system satisfies[2]:

$$N_{cnp}^i* = 0, \quad \alpha^i = \alpha* \quad and \quad Q* = Q_{th} \tag{8}$$

$$R_C^i* = \frac{C}{N} \tag{9}$$

where symbol $*$ represents the value at the fixed points.

B. Derivation of the System Transfer Function

Referring to the linearization and Laplace transformation method used in [21]–[23], we give the derivation of the system function for PACC as follows:

To linearize the fluid model around the fixed points represented in equation (8) and (9), we firstly redefine the right-hand sides of (4), (6) and (7) by:

$$\begin{aligned} f(R_C) &\doteq N R_C(t-T) - C \\ g(R_C, \alpha, N_{cnp}) &\doteq -\frac{R_C(t)}{2T}\alpha(t) N_{cnp}(t-T) \\ h(Q, \dot{Q}) &\doteq \frac{\beta_1}{T}[Q(t) - Q_{th}] + (\beta_2 + \frac{\beta_1}{2})\frac{dQ(t)}{dt} \end{aligned} \tag{10}$$

where $\dot{Q}$ is the differential of Q.

Evaluating partials at the fixed point $(R_C*, \alpha*, N_{cnp}*, Q*)$ of (10) gives:

$$\frac{\partial f}{\partial R_C} = N \tag{11}$$

$$\begin{aligned} \frac{\partial g}{\partial R_C} &= -\frac{\alpha*}{2} N_{cnp}* = 0 \\ \frac{\partial g}{\partial \alpha} &= -\frac{R_C*}{2T} N_{cnp}* = 0 \\ \frac{\partial g}{\partial N_{cnp}} &= -\frac{R_C * \alpha*}{2T} \end{aligned} \tag{12}$$

$$\begin{aligned} \frac{\partial h}{\partial Q} &= \frac{\beta_1}{T} \\ \frac{\partial h}{\partial \dot{Q}} &= \beta_2 + \frac{\beta_1}{2} \end{aligned} \tag{13}$$

With the denotions $\delta Q \doteq Q - Q*$, $\delta R_C \doteq R_C - R_C*$, $\delta N_{cnp} \doteq N_{cnp} - N_{cnp}*$, we make linearization around the operation point and obtain:

$$\begin{aligned} \delta\dot{Q}(t) &= N\delta R_C(t-T) \\ \delta\dot{R_C}(t) &= -\frac{R_C * \alpha*}{2T}\delta N_{cnp}(t-T) \\ \delta\dot{N_{cnp}}(t) &= \frac{\beta_1}{T}\delta Q(t) + (\beta_2 + \frac{\beta_1}{2})\delta\dot{Q}(t) \end{aligned} \tag{14}$$

Performing the Laplace transform on differential equations in (14) we get:

$$\begin{aligned} Q(s) &= N\frac{e^{-sT}}{s} R_C(s) \\ R_C(s) &= -\frac{R_C * \alpha*}{2sT} N_{cnp}(s) \\ N_{cnp}(s) &= \frac{\frac{\beta_1}{T} + (\beta_2 + \frac{\beta_1}{2})s}{s} E(s) \end{aligned} \tag{15}$$

where $E(s) = Q(s) - Q_{th}$ is the error signal.

Combining functions in (15), we finally get the open-loop transfer function of the whole system as follows:

$$G(s) = -K\frac{1 + \frac{s}{z}}{s^3} e^{-2sT} \tag{16}$$

where $K = \frac{C\beta_1\alpha*}{2T^2}$ and $z = \frac{\beta_1}{(\beta_2 + \frac{\beta_1}{2})T}$.

C. Stability Analysis and Control Parameters

According to the Bode Stability Criteria [20], the system is stable only if the phase margin in the Bode diagram of the transfer function $G(s)$ is above 0. Fig. 5 shows the variation of margin phase relative to different (β_1, β_2) pairs, which illustrates that a closer pair of (β_1, β_2) provides a faster response to the control signal $Q(t)$, but may also lead to a reduction in stability as β_1 grows. In fact, there are still other ranges for (β_1, β_2) that can stabilize the system. However, based on the convenience of operation and the validity of N_{cnp}, we believe that the [0,1] interval is more practical. In general, we take $(0.05, 0.1)$ as the default adjustment factor of the PI controller in PACC.

[1]Since the allocation of CNPs between flows is proportional to N_{cnp}, the scaling operation from $N_{all}(t)$ to $N_{cnp}^i(t)$ has no effect to the stability.

[2]The derivation of stable value of α is given in appendix.

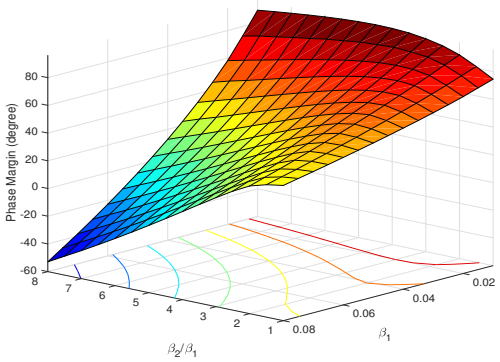


Fig. 5: Phase margin as a function of parameter β_1 and β_2.

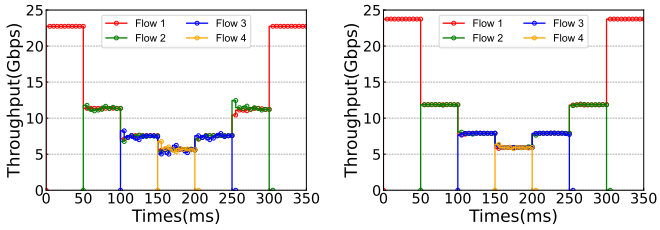
Besides, T and Q_{th} also affect the system gain (that is, K in equation (16)) and the equilibrium point, respectively. Weighing the response speed, queuing delay and link utilization, we set $T = 80\mu$s, and $Q_{th} = \frac{T}{2} \times C$ based on the bandwidth-delay product, where C is the egress link bandwidth. For Q_{burst}, in order to ensure it can effectively distinguish tiny microbursts, we consider the worst case that congestion occurs at the sender-side ToR switch and set $Q_{burst} = d \times C$, where d is the single-hop delay. The threshold for flow table aging (B_{th}) is set to the same as Q_{burst} for simplicity.

V. Evaluation

In this section, we implement PACC in NS3 based on the open-source in [6] and evaluate it using a combination of micro-benchmarks and large-scale simulations. Without extra instruction, the topology used is the same as Section II.

A. Evaluation Setup

Schemes compared: We compare PACC with DCQCN, TIMELY and HPCC as they are the main CC mechanisms used in RDMA-enabled data center networks. We use the open-source implementation of DCQCN, HPCC and TIMELY, where the last one is implemented based on its algorithm [6].
Parameter settings: For PACC, according to the analysis in Section IV-C, we set $Q_{th} = 125KB$, $Q_{burst} = B_{th} = 4KB$, $w = 0.9$ and $(\beta_1, \beta_2) = (0.05, 0.1)$, respectively. For DCQCN, TIMELY and HPCC, we use the parameters suggested in corresponding papers. Moreover, we also scale the ECN marking threshold proportional to the link bandwidth suggested in [6] for DCQCN.
Traffic workloads: In micro-benchmarks, we generate the same traffic as in Section II. And for large-scale simulations, we use traffic workloads derived from two publicly available datacenter traffic traces consisting of throughput-sensitive large flows ($WebSearch$ traffic [24]) and latency-sensitive small flows (FB_Hadoop traffic [17]) in our simulations. To mimic various scenarios in a real production data center, we use Poisson-based flow intervals and different link loads to construct multiple traffic patterns with random combinations of senders and receivers.



(a) HPCC (b) PACC

Fig. 6: PACC achieves fairness.

Performance metrics: The performance of PACC are evaluated using the following metrics, (1) bottleneck throughput, (2) in-network queue length, (3) average FCT and (4) tail FCT.

B. Micro-benchmarks

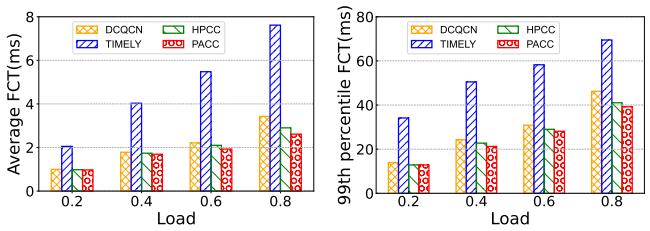
1) PACC has a fast reaction to the congestion: Fig. 1(a) and Fig. 1(b) depict the queue length and throughput at the bottleneck link with PACC. The queue stabilizes near the reference threshold ($125KB$) after an increase of about 15μs, and the peak decreases significantly compared with the other three solutions. Moreover, the shorter control loop of PACC advances the queue's reduction, thus giving a rapid convergence to the ideal throughput comparable with HPCC, which outperforms DCQCN and TIMELY.

2) PACC achieves fairness both in mixed and long flow scenarios: Fig. 2(a) and Fig. 2(b) show the results of PACC on per-flow throughput and N_{cnp} contrast with DCQCN. We note that PACC realizes a per-flow throughput close to the ideal fair share ($2.5Gbps$) during congestion. In addition, the proportion of the difference between throughput peaks in the total throughput decreases from 2.83% (0.07/24.72) to 1.21% (0.03/24.85). Due to the precise PI control of PACC based on Q_{th}, per-flow N_{cnp} is significantly reduced. This effectively prevents the continuous decrease of the sending rate and diminishes the overhead at end-hosts for processing superfluous CNPs in DCQCN.

We also conduct a micro-benchmark for the comparison of fairness between long flows using PACC and HPCC[1]. We use a dumbbell topology where five servers are connected to a switch via 25Gbps links, and randomly select four to send a long flow to the same receiver in turn with an interval of 50ms. From 200ms, we stop these senders one by one with the same interval. Fig. 6(a) and Fig. 6(b) show the throughput of flows in HPCC and PACC, respectively. With the rapid congestion feedback from the switch and the accurate allocation of N_{cnp}, PACC provides better fairness and can quickly throttle or recover the bottleneck bandwidth when a flow starts or ends. However, flows in HPCC can not grab the ideal fair share, the total throughput also suffers under-utilization on account of the 5% overhead on the INT header [6].

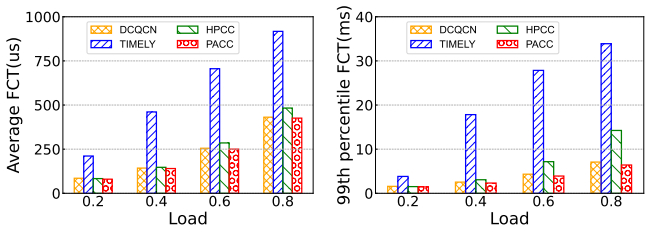
3) PACC handles well with tiny microbursts: Fig. 3(a) and Fig. 3(b) illustrate the variance of throughput under scenarios

[1]It can be verified in other literature [9] that HPCC outperforms TIMELY in fairness oftentimes, so we choose the better one for comparison.



(a) Average FCT (b) 99^{th} percentile FCT

Fig. 7: Average/99^{th} percentile FCT with $WebSearch$ traffic.



(a) Average FCT (b) 99^{th} percentile FCT

Fig. 8: Average/99^{th} percentile FCT with FB_Hadoop traffic.

in Section II-1) with PACC. Although the transmission time for tiny microbursts is less than one RTT ($\frac{30KB}{25Gbps} = 9.6\mu s$), with the accurate discrimination of congestion-contributive flows at the switch, PACC avoids the continuous low link utilization caused by the excessively long recovery phase under massive tiny microbursts. which is pervasive in applications such as machine learning training, key-value database access, etc [8].

C. Large-scale Simulations

1) Overall performance: Fig. 7 and Fig. 8 show the average and the 99^{th} percentile FCT of four schemes for $WebSearch$ traffic and FB_Hadoop traffic at different loads. We note that even under traffic with quite different patterns, PACC always outperforms other schemes both in the average and the 99^{th} percentile FCT. Specifically, in the $WebSearch$ traffic, PACC reduces the overall average FCT by up to 23%, 65% and 9% compared with DCQCN, TIMELY and HPCC, respectively, while the results change to 6%, 69% and 12% in FB_Hadoop traffic. The observations for the 99^{th} percentile FCT are similar to the average FCT under two workloads. Since PACC proactively feeds back accurate congestion signals from the switch rapidly for multiple types of flows once the queue length exceeds the threshold, the FCT of the flows contributed to the congestion will be reduced accordingly.

2) FCT breakdown based on flow size: By reviewing the results above, it is not hard to find that although TIMELY shows the same FCT trend under both workloads, DCQCN and HPCC has opposite results when traffic varies: HPCC gives a lower average and tail FCT under $WebSearch$ traffic but is inferior to DCQCN under FB_Hadoop traffic. To explore the fine-grained performance of four mechanisms, we break down FCT based on the flow size, as shown in Fig. 9 and Fig. 10.

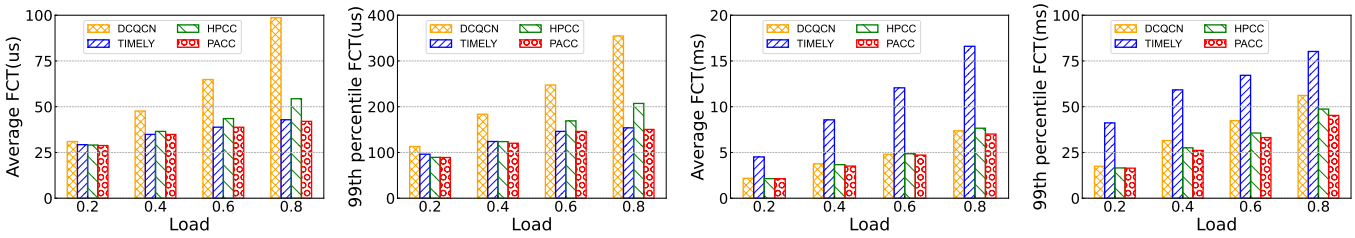
Unsurprisingly, PACC clearly surpasses DCQCN, TIMELY and HPCC for all the flow sizes either on the average or tail FCT regardless of traffic patterns. For DCQCN, with the increment of flow size, the gap between the long control loop and ideal transmission time shrinks, thus it gives a desirable result under long flows. Besides, we note that the performance of TIMELY degrades drastically as the flow size increases, which is consistent with its design based on the RTT variation. The micro views of FCT on small $(0 - 100KB)$ and large $(> 100KB)$ flows indicate that HPCC usually fails when flow size is extremely large. This is a consequence of the losing headroom bandwidth and INT information on data frames for large flows. However, with the help of PI controller-based feedback generation and flow discrimination at the switch, PACC is hardly affected by flow size. For short flows in $WebSearch$ traffic, PACC reduces the average and tail FCT of the best of the remaining three schemes by up to 3% and 8%, respectively. As for long flows, the improvement respectively are 6% and 7%.

VI. Related Work

CC is an enduring topic in data centers, and a plethora of novel proposals have emerged over the last decade. Here we briefly introduce some closely related work from three aspects.

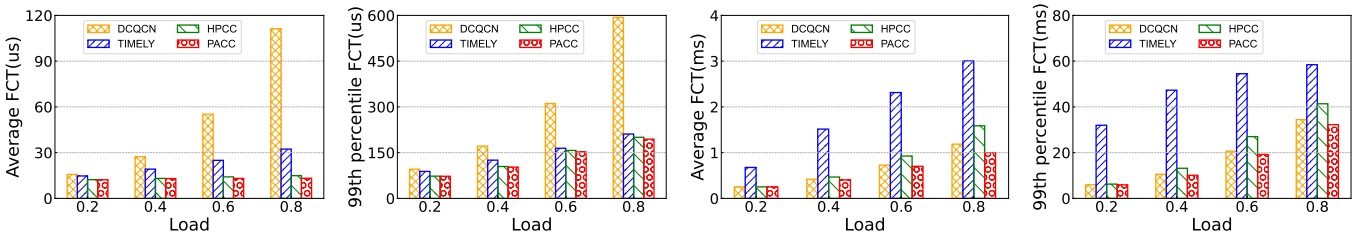
Reactive Congestion Control: The control entity of this type of CC is generally the sender side of the data transmission (i.e., ***sender-driven***). They passively adjust flow rate or window size according to various feedback signals from the congested path. Both DCTCP [24] and DCQCN [4] react to ECN marks. TIMELY [5] and Swift [25] are two general RTT-based CC mechanisms for DCN. HPCC [6] relies on accurate link load information offered by in-network telemetry (INT) to resize sending windows at sources. However, the intrinsic long end-to-end control loop in reactive CC algorithms may fail to cope with instantaneous abnormal traffic patterns and eventually lead to severe performance degradation [26].

Proactive Congestion Control: Proactive algorithms explicitly calculate the optimal bandwidth/token/credit allocation for each sender beforehand to meet the demanding application-level requirements. ***1) Switch-driven*** solutions measure in-network congestion accurately and directly send key information to the source. XCP [27] and RCP [28] adjust the window size information in the packet header and calculate each link's fair rate, respectively. TFC [29] uses a token-based bandwidth allocation scheme based on the number of active flows in each time interval. RoCC [9] uses a queue-length-based PI controller at the switch to proactively feed back accurately calculated fair rate to senders promptly. ***2) Receiver-driven*** solutions like ExpressPass [30] and NDP [31] are designed to cope with network congestions by combining RPS technology [32] at the switch and CC scheme at the receiver. pHost [33] and HOMA [34] aim to solve the last-hop congestion in DCNs, but their effectiveness largely depends on the assumption that most congestion occurs on the ToR downlinks. Though Aeolus [8] helps handle the "first-RTT" problem of receiver-driven



(a) 0-100KB average FCT (b) 0-100KB tail FCT (c) >100KB average FCT (d) >100KB tail FCT

Fig. 9: FCT breakdown with $WebSearch$ traffic.



(a) 0-100KB average FCT (b) 0-100KB tail FCT (c) >100KB average FCT (d) >100KB tail FCT

Fig. 10: FCT breakdown with FB_Hadoop traffic.

solutions with non-trivial scheduled-packet-first design, issues like core congestion remain pendent.

Flow Control Mechanism: Apart from PFC (buffer-based) for RoCE, CBFC (Credit-Based Flow Control) is widely used in the Infiniband fabrics [35]. The difference is that CBFC periodically generates flow control information (time-based), and the sender needs to determine the port status based on the feedback signal. GFC [36] controls the port rate with fine granularity to prevent diverse issues in PFC at scale. IRN [37] and MELO [38] intend to prevent PFC from triggering by reducing hardware-based selective packet retransmission. However, the above solutions do not change the blocking nature of PFC, and issues such as congestion proliferation and unfairness still need further study.

VII. Conclusion

This paper proposes PACC, a switch-driven, PI controller-based CC mechanism for RDMA-enabled DCNs. PACC is inspired by the defects of the state-of-the-art CC algorithms coping with some specific traffic patterns and the paradigm shift from source-driven to core-driven. With the combination design of precise computation, effective flow discrimination and fair allocation at the switch, PACC can promptly perceive in-network congestion and proactively piggybacks accurate congestion signals to the corresponding sender. The results of micro-benchmarks and large-scale simulations show that PACC achieves good fairness, high throughput and low FCT under multiple traffic patterns.

VIII. Acknowledgment

We would like to thank the anonymous reviewers for their constructive comments. This work was partly funded by the National Natural Science Foundation of China (NSFC) (Grant No. 61872401 and 62132022) and the Fok Ying Tung Education Foundation (Grant No. 171059).

Appendix A
Derivation of Stable Value of α

Referring to the variation of α in DCQCN, the increase of α is $g(1-\alpha)$ on receiving a CNP, and the decrease is $g\alpha$ every τ' without getting a CNP. When α converges to a stable value, the absolute change during a period of time P should equal 0. Assuming that a CNP arrives every t', by combining the two we have $\frac{P}{t'}g(1-\alpha) = g\alpha(\frac{P}{\tau'} - \frac{P}{t'})$. Solving this equation and we get $\alpha = \frac{\tau'}{t'}$.

Furthermore, if one rate increase phase covers l rate decrease events on average, then according to DCQCN we have:

$$T_i = l \times t' \tag{1}$$

where T_i is the timer for additive rate increase.

Besides, since the sending rate decreases by $\frac{\alpha}{2}R$ on receiving a CNP, the average reduction of rate during T_i is:

$$\Delta R = l \times \frac{\alpha}{2}R \tag{2}$$

Combining (1), (2) and α we obtain:

$$\alpha* = \sqrt{\frac{\tau'}{T_i} \times \frac{2\Delta R}{R}} \tag{3}$$

Statistically, the value of $\frac{\Delta R}{R}$ during T_i can be approximated as $\frac{(R_T - R_C)/2}{(R_T + R_C)/2}$. With the expected value of R_C (that is, $\frac{R_T}{2}$), we get $\alpha* = \sqrt{\frac{2\tau'}{3T_i}}$. We further take $\alpha* = 0.04714$ as default value using the typical $(1, 300)$ group of (τ', T_i).

REFERENCES

[1] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin and C. Yan, "Speeding up distributed request-response workflows," ACM SIGCOMM Computer Communication Review, 43(4), 2013, pp.219-230.

[2] ASSOCIATION, I. T. Supplement to InfiniBand TM Architecture Specification Volume 1 Release 1.2.1 Annex A17: RoCEv2 , 2014.

[3] 802.1Qbb - Priority-based Flow Control. http://www.ieee802.org/1/pages/802.1bb.html.

[4] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M.H. Yahia and M. Zhang, "Congestion Control for Large-Scale RDMA Deployments," SIGCOMM Comput. Commun. Rev., no. 4, 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787484

[5] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," ACM SIGCOMM Computer Communication Review, vol. 45, no. 4, 2015, pp. 537–550.

[6] Y. Li, R. Miao, H.H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly and M. Alizadeh, "HPCC: High Precision Congestion Control," in Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 44–58.

[7] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye and M. Lipshteyn, 2016, "RDMA over commodity ethernet at scale". in Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 202–215.

[8] S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, K. Chen, K. Tan and Y. 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, 2020, pp. 422-434.

[9] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster and T. Edsall, "RoCC: robust congestion control for RDMA," In Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies, 2020, pp. 17-30.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese and D. Walker, "P4: Programming protocol-independent packet processors," ACM SIGCOMM Computer Communication Review, 44(3), 2014, pp.87-95.

[11] W. Cheng, K. Qian, W. Jiang, T. Zhang and F. Ren, "Re-architecting congestion management in lossless Ethernet," in 17th USENIX Symposium on Networked Systems Design and Implementation, 2020, pp. 19-36.

[12] M. Al-Fares, A. Loukissas and A. Vahdat, "A scalable, commodity data center network architecture," ACM SIGCOMM computer communication review, vol. 38, no. 4, 2008, pp. 63–74.

[13] J. Woodruff, A.W. Moore and N. Zilberman, "Measuring burstiness in data center applications," in Proceedings of the 2019 Workshop on Buffer Sizing, 2019, pp. 1-6.

[14] W. Bai, L. Chen, K. Chen and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in 13th USENIX Symposium on Networked Systems Design and Implementation, 2016, pp. 537-549.

[15] J. Zhang, W. Bai and K. Chen, "Enabling ECN for datacenter networks with RTT variations," in Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, 2019, pp. 233-245.

[16] W. Bai, S. Hu, K. Chen, K. Tan and Y. Xiong, "One more config is enough: Saving (DC) TCP for high-speed extremely shallow-buffered datacenters," IEEE/ACM Transactions on Networking, 2021, pp.489-502.

[17] A. Roy, H. Zeng, J. Bagga, G. Porter and A.C. Snoeren, "Inside the social network's (datacenter) network," in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, 2015, pp. 123-137.

[18] ConnectX®-4 EN Adapter Card Single/Dual-port 100 Gigabit Ethernet Adapter. https://www.mellanox.com/products/ethernet-adapters/connectx-4-en.

[19] Gene Franklin, David Powell, and Abbas Emami-Naeini. 1995. Feedback Control of Dynamic Systems.

[20] Y. Zhu, M. Ghobadi, V. Misra and J. 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, 2016, pp. 313-327.

[21] M. Alizadeh, A. Kabbani, B. Atikoglu and B. Prabhakar, "Stability analysis of QCN: the averaging principle," in Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, 2011, pp. 49-60.

[22] R. Pan, P. Natarajan, C. Piglione, M.S. Prabhu, V. Subramanian, F. Baker and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in 2013 IEEE 14th international conference on high performance switching and routing (HPSR), 2013, pp. 148-155.

[23] C.V. Hollot, V. Misra, D. Towsley and W.B. Gong, "A control theoretic analysis of RED," in Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society, 2001, pp. 1510-1519.

[24] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in Proceedings of the ACM SIGCOMM 2010 Conference, 2010, pp. 63–74.

[25] G. Kumar, N. Dukkipati, K. Jang, H.M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld and M. Ryan., "Swift: Delay is simple and effective for congestion control in the datacenter," in Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, 2020, pp. 514–528.

[26] S. Liu, A. Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum and A. Sivaraman, "Breaking the transience-equilibrium nexus: A new approach to datacenter packet transport," in NSDI, 2021, pp. 47–63.

[27] D. Katabi, M. Handley and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, 2002, pp. 89-102.

[28] N. Dukkipati, "RCP: Congestion control to make flows complete quickly," Ph.D. dissertation, PhD Thesis, Department of Electrical Engineering, Stanford University, 2006.

[29] J. Zhang, F. Ren, R. Shu and P. Cheng, "TFC: Token flow control in data center networks," in Proceedings of the Eleventh European Conference on Computer Systems, 2016, pp. 1-14.

[30] I. Cho, K. Jang and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 239-252.

[31] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A.W. Moore, G. Antichi and M. 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, 2017, pp. 29-42.

[32] A. Dixit, P. Prakash, Y.C. Hu and R.R. Kompella, "On the impact of packet spraying in data center networks," in 2013 Proceedings IEEE INFOCOM, 2013, pp. 2130-2138.

[33] P.X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, 2015, pp. 1-12.

[34] B. Montazeri, Y. Li, M. Alizadeh, and J. 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, 2018, pp. 221–235.

[35] InfiniBand Trade Association. 2015. InfiniBand architecture specification: release 1.3. (2015). https://www.infinibandta.org/ibta-specifications-download/

[36] K. Qian, W. Cheng, T. Zhang and F. Ren, "Gentle flow control: avoiding deadlock in lossless networks," in Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 75-89.

[37] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy and S. Shenker, "Revisiting network support for RDMA," in Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 313-326.

[38] Y. Lu, G. Chen, Z. Ruan, W. Xiao, B. Li, J. Zhang, Y. Xiong, P. Cheng and E. Chen, "Memory efficient loss recovery for hardware-based transport in datacenter," in Proceedings of the First Asia-Pacific Workshop on Networking, 2017, pp. 22-28.