An Anatomy of Token-Based Congestion Control

Kexin Liu, Chang Liu, Qingyue Wang, Zhiqiang Li, Lu Lu, *Member, IEEE*, Xiaoliang Wang, *Member, IEEE*, Fu Xiao, *Senior Member, IEEE*, Ying Zhang, *Senior Member, IEEE*, Wanchun Dou, Guihai Chen, *Fellow, IEEE*, and Chen Tian, *Senior Member, IEEE*

***Abstract*— Congestion control protocols are crucial for optimizing the performance of datacenter network applications. Although reactive congestion control (RCC) protocols are commonly used in commercial datacenters, researchers have been exploring token-based proactive congestion control (TCC) protocols to further enhance network performance. Despite the development of numerous TCC variants, there has not been a thorough examination of the design space of TCC protocols until now. This paper aims to address this gap by introducing a framework for understanding the design choices within the TCC approach for TCC protocols. By analyzing various design aspects of TCC approaches, we create a novel TCC protocol called ToCC. At the central of ToCC design is that it leverages congestion control mechanisms over tokens. To implement ToCC, we tackle several challenges and integrate it into NP-based smart NICs. Comparing ToCC with state-of-the-art TCC and RCC protocols through extensive large-scale simulations and testbed evaluations, we find that ToCC consistently achieves low latency across different scenarios. Moreover, ToCC significantly reduces buffer occupancy by 4.8 times compared to existing methods, and during incast scenarios, it decreases flow completion time by up to 90%.**

***Index Terms*— Datacenter networks, token-based proactive congestion control, NP-based smart NICs.**

I. Introduction

THE performance of applications in datacenter networks heavily relies on the effectiveness of congestion control protocols [1], [2], [3], [4]. These protocols can be broadly classified into two categories: *reactive* and *proactive*. Reactive congestion control (RCC) protocols are commonly implemented in commercial datacenters. In RCC protocols, senders utilize the network bandwidth by transmitting data at line rate. When congestion occurs and buffers start to fill up, congestion signals are either sent back to or detected by the senders [5], [6], [7], [8], [9]. Upon receiving these signals, senders reduce their transmission rate and attempt to recover later. RCC protocols have inherent limitations in handling bursty flows [10]. To address these limitations, a promising branch of protocols called token-based proactive congestion control (TCC) has emerged as a solution. TCC protocols aim to overcome the challenges posed by bursty flows and improve overall performance in datacenter networks.

Token-based proactive congestion control protocols, such as ExpressPass [10], [11], Homa [12], [13], NDP [14], pHost [15], and Aeolus [16], leveraging the transmission of *token* packets from receivers to allocate bandwidth for future data transmissions. Actual packet transmission occurs only after senders receive the bandwidth allocation signal. These TCC protocols feature diverse design choices. For example, ExpressPass solely transmits scheduled packets and eliminates incast issues by sacrificing the initial round-trip time (RTT). In contrast, protocols like Homa, NDP, and Aeolus send *unscheduled* packets before token packets arrive, preventing waste of the first RTT. However, these methods still struggle with completely alleviating burstiness in incast traffic scenarios.

To propose an enhanced TCC protocol, we should thoroughly investigate the fundamental design space of current TCC protocols. We seek to assess various design choices under different scenarios and offer insights on when and why specific design aspects lead to better performance. By examining these design choices in detail, we can gain a more profound understanding of their effects and optimize TCC protocols for improved network functionality.

Firstly, we introduce an analytical framework for TCC called the "Anatomy Framework." This framework methodically breaks down the design space into multiple aspects, sets up relevant metrics, and thoroughly examines existing literature on TCC. The term "anatomy" is used metaphorically, similar to how medical science analyzes a subject's intricate details. Creating such a framework presents several challenges:

- Each design dimension should be defined relatively independently and orthogonally for a comprehensive analysis.
- Determining the optimal granularity of design choices is difficult due to potential correlations between certain design aspects; hence, careful consideration is necessary to distinguish separate choices within each dimension.

Received 17 December 2023; revised 30 July 2024; accepted 6 October 2024; approved by IEEE TRANSACTIONS ON NETWORKING Editor S. Kompella. Date of publication 1 January 2025; date of current version 18 April 2025. This work was supported in part by the Nanjing University-China Mobile Communications Group Co., Ltd., Joint Institute; in part by the Key Project of Jiangsu Province fundamental Research Program under Grant BK20243053; in part by the National Natural Science Foundation of China under Grant 62325205 and Grant 62172204; in part by the Fundamental Research Funds for the Central Universities; in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization; and in part by the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program. *(Corresponding authors: Xiaoliang Wang; Lu Lu.)*

Kexin Liu, Chang Liu, Qingyue Wang, Xiaoliang Wang, Wanchun Dou, Guihai Chen, and Chen Tian are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: waxili@nju.edu.cn).

Zhiqiang Li and Lu Lu are with China Mobile Research Institute, Beijing 100053, China (e-mail: lulu@chinamobile.com).

Fu Xiao is with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China.

Ying Zhang is with Meta, Menlo Park, CA 94025 USA.

Digital Object Identifier 10.1109/TNET.2024.3491763

- Isolating the effects of individual design choices is crucial for accurately assessing their advantages and disadvantages. Analyzing each choice in isolation offers a more detailed understanding of their implications.

After addressing the aforementioned challenges, we develop the Anatomy Framework, which enables the systematic disassembly of TCC into five fundamental components (*i.e.*, transmission of unscheduled packets, token generation, in-network rate-limiters, and total volume control of token).

Based on the insights derived from the Anatomy Framework, we propose a novel token-based proactive congestion control protocol called Token-oriented Congestion Control (ToCC). At the central of ToCC design is that it leverages congestion control mechanisms over tokens. This unique approach enables more efficient congestion management compared to that of the congestion control over data. We thoroughly describe multiple scenarios where ToCC can effectively apply token-based congestion control, subsequently enhancing its performance in real-world deployment settings. Besides, ToCC takes on several key components from the anatomy framework of TCC and offers customization capabilities tailored to various scenarios (*e.g.*, incast and lightly-loaded networks). Our design is not a merely assemblage of successful design choices from existing protocols. It requires careful consideration since not all design choices are compatible (*e.g.*, rate-based token generation and total volume control of token) or work well in combination (*e.g.*, no unscheduled phase and data-driven token generation); thus ToCC leverages adaptations and compatibility assessments for smooth integration.

We implement ToCC on NP-based smart NICs by addressing various challenges, such as unforeseen performance deterioration caused by the limitation on the packet processing rate of programmable NICs, implementation complexity, and restricted hardware capabilities. Overcoming these hurdles demanded finding a balance between practicality and high performance. To our knowledge, this represents the first time TCC has been implemented on NP-based hardware smart NICs.

We conduct a thorough evaluation of ToCC through both NS3 simulations and testbed experiments. ToCC converges rapidly, showcasing its capability to promptly adjust to diverse network conditions. Moreover, ToCC demonstrates resilience to parameter settings and places fewer demands on congestion control algorithms in terms of convergence speed and congestion detection. This makes ToCC more appropriate for large-scale implementation in production environments. Additionally, the evaluation reveals that ToCC demonstrates enhanced flexibility in handling bursty traffic scenarios such as incast. It decreases flow completion time by 90% and buffer occupancy by 4.8 times while maintaining stable and high throughput. These findings emphasize the strong performance of ToCC in enhancing network efficiency and overall application experience.

II. Background

Reactive congestion control (RCC). As the name of RCC indicates, an RCC sender transmits a portion of packets (usually at line rate) when a new flow arrives and then adjusts the transmission rate according to the congestion signals.

Built upon DCTCP [17] and QCN [18], [19], DCQCN [5], [20] uses Explicit Congestion Notification (ECN) to detect in-network congestion. TIMELY [8] uses Round-Trip Time (RTT) variations as congestion signals. Swift [9] measures RTT in a more precise way, *i.e.*, it decomposes end-to-end RTT to separate fabric delay from host delay. HPCC [6] leverages in-band network telemetry (INT) to obtain precise link load information, which makes the rate adjustment more accurate. PowerTCP [7] captures both absolute network state and variations through INT to obtain the bottleneck link state. Bolt [21] reduces the congestion detection loop by sending back notifications at switches directly to the senders.

However, with the rapid growth of link bandwidth (*i.e.*, from 10 Gbps to 100 Gbps), the bandwidth-delay product (BDP) is increased accordingly. It results in datacenter traffic becoming more bursty [16]. A larger fraction of flows can be sent out even before RCC takes effect. Moreover, the switch buffer does not catch up with the increasing links, making it harder for switches to absorb bursty traffic [22], [23].

Token-based Proactive congestion control (TCC). A TCC receiver transmits *token* packets to allocate bandwidth for flows (*e.g.*, *credit* in ExpressPass, and *PULL* in NDP). When a token packet is received, the sender can transmit a corresponding full-length *scheduled* data packet.

In ExpressPass [10], [11], all data are scheduled. When a new flow arrives, a control packet is sent from the sender to notify the receiver. A receiver then transmits *token* to compete for bandwidth before arriving at senders. The bandwidth that tokens can consume is limited to around 5% of the link bandwidth to guarantee that the bandwidth consumed by scheduled packets does not exceed the network bottleneck. The rate is calculated according to the size of a token and a full-length data packet (*i.e.*, $\frac{84}{1538+84} \approx 5\%$). Zero packet loss and low buffer occupancy are achieved, at the cost of wasting the first Round-Trip Time (RTT) waiting for tokens.

To make full use of the first RTT, other state-of-art TCCs (*i.e.*, Homa, NDP, and pHost) transmit a portion of *unscheduled* packets before the bandwidth allocation packets (*i.e.*, tokens) are received. Homa [12] mainly aims to optimize RPC-like scenarios [24], [25]. It leverages the in-network priority queue and Shortest-Remaining-Time-First (SRTF) strategy on the host to optimize the flows' FCT. NDP [14] uses *cut-payload* method to cut the payloads of data packets when the switch queue length exceeds a small threshold and the packet header is transmitted to the receiver to exactly notify the packet loss. Homa and NDP assume that in-network congestion is rare. Thus, tokens do not compete for bandwidth at the network bottleneck. Instead, they act as scheduling signals from receivers to senders. Aeolus [16] is a patch to TCC, and it aims to solve the first-RTT problem. It drops unscheduled packets selectively when the queue length exceeds a small threshold, *i.e.*, 8 KB, by leveraging the Active Queue Management (AQM) feature of commodity datacenter switches. This ensures that scheduled packets are not dropped due to buffer overflow. Especially when Aeolus is patched to ExpressPass,

TABLE I
DESIGN OPTIONS OF TCC

	Unscheduled Phase (P1)			Scheduled Phase					Others
	w/		w/o	Token Generation (P2)		Rate-limiter (P3)	Total Volume Control (P4)	Token CC (P5)	
	Passive Drop	Active Drop		Rate-based	Data-driven				
ExpressPass			✓	✓		✓		✓	
Homa	✓				✓		✓		Priority
NDP		✓			✓		✓		Cut Payload
Aeolus		✓		-	-	-	-	-	
ToCC	✓*				✓*	✓	✓	✓	

unscheduled packets are generated. *In this paper, we use EP+Aeolus to denote ExpressPass patched with Aeolus.*

III. Analysis Methodology

Our goal is to analyze the relative merits of the design choices made by various TCCs. We do this by firstly developing a framework of TCC (§ III-A). Then, we use reasoning and conduct variable-controlling simulations across multiple topologies and traffic patterns (§ III-B).

A. TCC *Characterization Framework*

We should isolate the effects of each individual TCC design choices as far as possible to provide convenience for further controlled experiments. At the same time, we should be careful that some choices are coupled. By considering these targets and constraints, we develop a characterization framework of TCC. It divides the components of TCC into two phases, *i.e.*, unscheduled and scheduled phases, which can be further anatomized into five key design points (P1-P5) as shown in Table I.

Unscheduled Phase. There are three choices when handling the unscheduled phase (P1). When a new flow arrives, the sender can wait for bandwidth allocation before sending data or initialize a portion of *unscheduled packets* without waiting for bandwidth allocation. After that, the switch can transmit unscheduled packets normally or drop unscheduled packets actively when the queue length exceeds a small threshold to avoid disturbing the scheduled phase (§ IV-A).

Scheduled Phase. The transmission of scheduled packets is triggered by token packets. When a token packet is received by the sender, a corresponding scheduled packet is transmitted. The design choices of token packets can be anatomized into the way to generate token (P2), the rate-limiter on tokens (P3), the total volume control of tokens (P4), and the congestion control on tokens (P5).

Token packets can be generated based on a given rate (*i.e.*, rate-based) or upon receiving a data packet (*i.e.*, data-driven) (§ IV-B). Rate-limiters on tokens can ensure that tokens arriving at senders do not trigger scheduled packets exceeding bottleneck bandwidth (§ IV-C). Total volume control of tokens ensures the number of tokens generated exactly matches the scheduled packets to be transmitted (§ IV-D). The congestion control (CC) algorithms on token packets adjust the transmission rate of tokens according to the token queue length (§ IV-F). When using rate-based token generation, configuring rate-limiters and a small token queue on switches is a necessity to drop excessive tokens to reduce bandwidth waste.

B. Evaluation Harness

Performance isolation. For each TCC protocol, we isolate the effects of its multiple design points on performance as far as possible by controlling the variable when setting up simulation scenarios and using contrastive analysis. For instance, when analyzing three design choices of transmitting unscheduled packets, Memcached traffic made up of tiny flows are used. Hence, only the unscheduled phases are involved, and the interference of design choices on scheduled phases can be avoided. When analyzing the effectiveness of different ways to generate tokens, workloads with relatively large flows (*e.g.*, Cache Follower and Web Search) are used. And the tail latency performance is compared since it is dictated by the scheduling phase.

Topologies and Workloads. We construct simulations under different scenarios to make our findings more comprehensive. Four topologies(§ VI) and four realistic traffic widely used in previous works are used to conduct a wide range of evaluations [12], [16], [17], [26]. As shown in Figure 1, the workloads cover a wide range of flow sizes. Memcached consists of tiny flows, where more than 90% fraction of flows is smaller than one full-length packet. Web Server and Cache Follower are mostly mixed of small ($<$ 100 KB) and medium-sized (100 KB-1 MB) flows. Web Search is mixed of medium and large ($>$1 MB) flows. Performance under different loads (*e.g.*, 0.2-0.8) is conducted (Appendix B), and we choose a load of 0.8 to best reflect the performance variations of different TCCs.

Buffer Model. For fairness comparison, the shared buffer model is used in our evaluation (except for NDP and Aeolus since they set a small drop threshold on the data queue to avoid a large queuing delay). For Homa, a relatively large buffer size (*e.g.*, 35 MB) is used to reduce its packet loss. For RCC protocols, PFC [27] is used to ensure packet loss does not occur. And we also discuss the limitations of TCCs by depicting how TCCs performs when facing packet drop (§ VI).

Parameter Settings. For different protocols, authors' recommended parameters are used. And to ensure fairness of the comparison, we use the same load balance protocols (*e.g.*, per-flow ECMP) in our evaluations.

Metrics. We choose multiple performance metrics to best reflect the effects of the validating design point. Four major performance metrics are presented: (i) average /99-tail Flow Completion Time (FCT), (ii) average/99-tail slowdown, where slowdown denotes the ratio of the actual FCT divided by the best possible time required to complete the flow on an unloaded network, (iii) maximum buffer occupancy, and (iv)

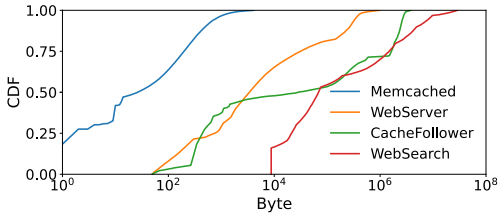


Fig. 1. Flow size distributions of typical workloads.

real-time throughput. When analyzing the design choices, implementation flexibility is also considered.

IV. TCC Anatomy

In this section, we anatomize different design choices of TCC approaches according to TCC framework (§ III-A). Table I summarizes different design choices of them. And then, we present our findings on different design choices by following the guidance of the analysis methodology (§ III-B).

A. Transmission of Unscheduled Packets

As shown in Table I (P1), there are typically three categories of choices regarding the unscheduled packet transmission.

Absence of the unscheduled packet. ExpressPass does not transmit unscheduled packets, and all its packets are scheduled by token packets sent from receivers.

Make full use of the first-RTT. Instead of wasting the first RTT, Homa/NDP transmits a portion (*e.g.*, one BDP) of unscheduled packets when a new flow arrives.

Back-off to send unscheduled packets without disturbing scheduled packets. Aeolus can be patched to TCCs. When a new flow arrives, it transmits a portion of unscheduled packets at the line rate. When the queue length of the switch's output port exceeds a small threshold, *e.g.*, 8KB, unscheduled packets are dropped selectively. Hence, scheduled packets are not likely to be interfered by unscheduled packets. This ensures that scheduled packets are not dropped due to buffer overflow.

Pros and Cons discussion. ExpressPass achieves an ultra-low buffer occupancy and handles incast scenarios well by scheduling every packet. However, it sacrifices the utilization of new flows' first RTT. For tiny flows whose size is smaller than one full-length packet, their FCTs could be tripled. Figure 2(a) shows the slowdown performance under the Memcached workload. ExpressPass does not perform well compared with those that transmit unscheduled packets (*e.g.*, Homa and NDP).

It is not always a good choice to fully use the unscheduled phase. Figure 2(c) shows that under Cache Follower workloads, the buffer occupancy of Homa can reach 20 MB. A large buffer occupancy faces the risk of packet drop. Homa leverages in-network priority queues to split flows according to their size. The benefit of priority scheduling and queuing gradually vanish with a large buffer occupancy. When in-network priority queues are not efficient, or the flow size is unknown, the performance of small flows can be further hurt by a large buffer occupancy.

When small flows dominate the traffic, it could be in vain to transmit unscheduled packets and selectively drop it afterwards since it induces a large drop ratio. It can be seen from Figure 2(a) that, compared with ExpressPass, the performance of EP+Aeolus even downgrades. It is also unfriendly for NICs by forcing them to handle packet-reordering caused by packet loss and retransmitting a large number of data packets, which complicates the implementation logic and wastes the packet per second (pps) on NICs.

Along with the high link bandwidth (*e.g.*, 100 Gbps / 400 Gbps), BDP is much larger than before. Nevertheless, the average flow size does not change obviously. On the one hand, there can be more flows whose size can be fit within one BDP, making the flows' first-RTT time more critical. Figure 2(b) shows that under 100 Gbps links, the performance gap between ExpressPass and Homa/NDP becomes more significant than that under 10 Gbps links. EP+Aeolus achieves better performance than ExpressPass, which contradicts the trends in 10 Gbps links. These results indicate that it is more costly to give up the unscheduled phase. On the other hand, there can be more unscheduled packets using bandwidth without allocation, which should be handled more carefully to avoid buffer overflow. As shown in Figure 2(d), under 100 Gbps links, Homa occupies a large buffer.

Finding 1. *The unscheduled phase should be configurable according to the traffic and link bandwidth characteristics of datacenter (§ V-A).*

B. Token Generation

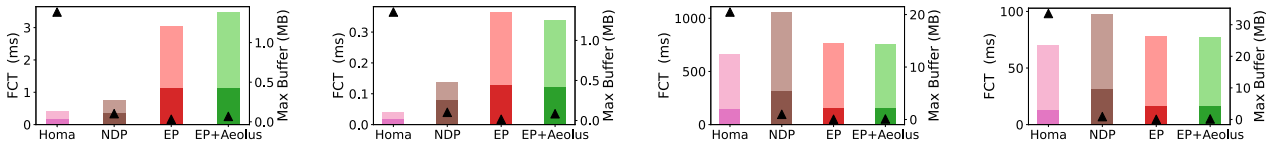
As shown in Table I (P2), the transmission of tokens can be based on an adjusted rate (*i.e.*, rate-based) or triggered by data packets (*i.e.*, data-driven).

Rate-based token generation. ExpressPass leverages a rate-based way to generate tokens. When a sender notifies the receiver that a new flow arrives, the receiver starts to send tokens for the flow based on a rate and stops sending tokens when all data of the flow are received (or when a dedicated *FIN* packet arrives). Since when tokens are generated in a rate-based manner, a large number of unnecessary tokens can be generated. If these excessive tokens arrive at the sender, it indicates that useful tokens are delayed, which can result in serious bandwidth waste. Hence, the rate-based token generation is coupled with in-network rate-limiters and a small token queue length to alleviate bandwidth waste. Tokens compete for 5% bandwidth at the network bottleneck. Then, tokens are dropped when the token queue length exceeds a given small threshold (*i.e.*, eight-packet length).

Data-driven token generation. Homa leverages a data-driven way to generate token packets, *i.e.*, only when a data packet is received can a corresponding token be transmitted to the sender. In NDP, when the receiver receives a data packet or a packet header notifying the data drop, the receiver adds a *PULL* packet to its pull queue. The pull packet is then sent to the corresponding sender to trigger a corresponding data packet.

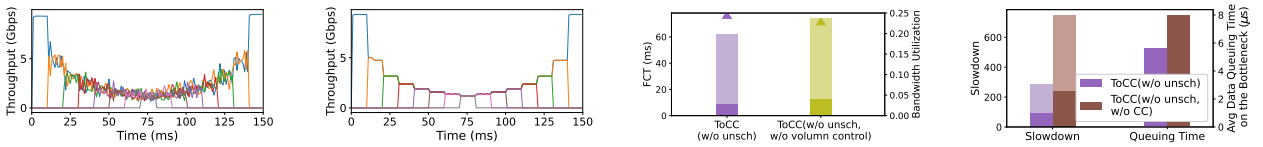
Pros and cons discussion. Rate-based token generation induces token loss, which leads to four problems.

(i) Tokens for a flow can be dropped repeatedly. It can cause a flow to wait for tokens for a long time, which especially prolongs the FCTs of small flows. Figure 2(a) shows the performance under Memcached. NDP and EP+Aeolus both



(a) Memcached under 10 Gbps. (b) Memcached under 100 Gbps. (c) Cache Follower under 10 Gbps. (d) Cache Follower under 100 Gbps.

Fig. 2. The investigation of the key design options of TCC. The deep/light color for each bar represents the average/99th-tail value, respectively. The left/right y-axis of (a)-(d) denotes the value of FCTs and the maximum buffer occupancy, respectively. The triangles of (a)-(d) denote the maximum buffer occupancy.



(a) Fairness of ExpressPass. (b) Fairness of Homa. (c) Volume control improves tail FCTs (Web Server under 10 Gbps). (d) Token CC is necessary (Web Server under 10 Gbps).

Fig. 3. The investigation of the key design options of TCC. The deep/light color for each bar of (c) and (d) represents the average/99th-tail value, respectively. The left/right y-axis of (c) denotes the value of FCTs and the bandwidth utilization, respectively. The triangles of (c) denote the bandwidth utilization.

leverage the active drop of unscheduled packets, while NDP shows a better performance. For that, the rate-based token mechanism of ExpressPass causes redundant drops of tokens.

(ii) Because of token loss, the receiver of ExpressPass can not stop sending tokens until all data packets or the stop-sending notification is received. It results in the last-RTT waste of tokens, *i.e.*, at least one RTT worth of tokens for each flow cannot trigger data packets. It wastes the bandwidth on both receiver and sender sides. Excessive tokens waste the bandwidth reserved for tokens. Moreover, when useless tokens arrive at senders, it indicates that other useful tokens that should have triggered data packets are dropped. It could downgrade the network throughput and prolong the tail latency of flows. Figure 2(c) compares the performance of different TCCs under Cache Follower. The tail latency of flows in ExpressPass is prolonged. Benefiting from the data-driven mechanism, Homa achieves better tail latency.

(iii) The random drop of tokens causes unfairness issues, especially synchronized flows. Although ExpressPass claims that rate-based token generation can achieve fairness performance by uniform random dropping of tokens at switches, we found that unfairness could happen when flows are synchronized. To test the fairness of two token generation methods, simulations are conducted in the case where eight flows of the same size pass through the same bottleneck one by one and leave afterwards. Figure 3(a) and 3(b) show the result. For ExpressPass, a relatively large jitter on throughput is observed, mainly caused by token loss. Homa achieves better fairness (Note that in this scenario, flows have the same priority in Homa; hence in-network priority does not interfere the fairness.)

(iv) Rate-based token generation consumes the pps of NICs, which puts much pressure on NICs.

The main drawback of data-driven token generation mainly lies in that there should be a token recovery mechanism to handle rare token loss, *i.e.*, token packet corruption or configuration error on switches. It can be overcome by leveraging the packet loss recovery mechanism (§ V-B).

Finding 2. *Data-driven token generation achieves better tail latency and fairness performance.*

C. In-Network Rate-Limiter

TCCs can choose whether to leverage in-network rate-limiters on tokens, as depicted in Table I.

Absence of rate-limiters. In Homa and NDP, no in-network congestion is assumed, so no rate-limiter is used. Upon a token is transmitted by the receiver, it passes through the network without bandwidth competition.

Presence of rate-limiters. In ExpressPass, NICs of both switches and end-hosts maintain rate-limiters on the token queue. When a token passes through the network bottleneck, it should compete for the rate-limited bandwidth with other tokens. Only tokens win from bandwidth competition, *i.e.*, those arriving at the sender can successfully trigger data transmission. It ensures that the bandwidth consumed by scheduled packets does not exceed the network bottleneck.

Pros and cons discussion. TCCs are particularly attractive as they enable fast convergence and low buffering by letting senders and receivers accurately estimate the available bandwidth for a given flow. However, receivers transmit tokens in a distributed way, based on their own link bandwidth. The total bandwidth allocated by tokens could exceed the network bottleneck, which can in turn result in buffer built-up [16]. This is the reason why some TCCs (*e.g.*, Homa and NDP) fall short in presence of bottlenecks in the network core, *e.g.*, due to oversubscription or load imbalance. As shown in Figure 2(c) and 2(d), under Cache Follower workloads which consists of the scheduling phase, the buffer occupancy of Homa is extremely large (*e.g.*, reach up to 20 MB-30 MB). And the performance of NDP is downgraded because it induces a massive number of packet loss and its retransmission mechanism does not consider the existence of in-network congestion, finally resulting in a severe packet drop ratio (around 30%).

In-network rate limiters on tokens provide *centralized* coordination of bandwidth allocation for different flows implicitly. By rate limiting the transmission of tokens as a whole, it regulates the number of tokens arriving at senders in each time unit accurately. Then senders in turn transmit data that exactly match the bottleneck link bandwidth.

Generally, rate-limiter is supported in commercial data-center switches [28], [29], [30]. However, there are some concerns about using a rate-limiter. (i) Rate-limiter induces

token built-up. Fortunately, the size of a token packet is small, which induces a relatively small buffer occupancy. And congestion control on tokens can be leveraged to mitigate the problem (§ IV-F). (ii) Whenever the bandwidth reserved for tokens is used, scheduled packets utilize the left 95% link bandwidth at most. There are corner cases where 5% bandwidth is wasted when only one-sided traffic exists (§ VI-D). Generally, unscheduled packets could fill in the bandwidth gap. (iii) A dedicated queue for a token should be used to leverage rate limiting. Each output port in a commodity switch supports multiple priority levels (typically eight) [12], [27]. Hence, a dedicated queue for tokens is reasonable.

Finding 3. *Token rate-limiters on switches provide centralized bandwidth allocation control. It ensures that the bandwidth allocated by tokens exactly matches the bottleneck bandwidth.*

D. Total Volume Control of Token

The total volume control of tokens denotes that receivers track the number of tokens sent and stop sending tokens when the number equals the volume needed for a flow. It is coupled with the token generation manner.

Absence of token total volume control. Without total volume control, receivers do not stop sending tokens until all data are received. The rate-based token generation (*e.g.*, ExpressPass) is not compatible with the total volume control because whether tokens arrive at senders is unpredictable after competing for bandwidth in networks (§ IV-B).

Presence of token total volume control. When tokens are generated in a data-driven manner, it can be chosen whether to control the total volume of tokens or not. Homa and NDP both track the number of tokens needed according to the traffic left to be scheduled.

Pros and cons discussion. Figure 3(c) compares the performance of ToCC (soon be presented in § V) and ToCC (w/o total volume control). When leveraging total volume control, the tail latency of ToCC can be greatly improved.

Findings 4. *Total volume control of tokens is vital to avoid the last-RTT token waste which affects the tail latency of flows.*

E. Other In-Network Support

Besides the main design points discussed above, different TCC has specific mechanisms to optimize its performance (§ II). Homa leverages SRPT (shortest remaining processing time) scheduling strategies on senders, prioritizing smaller flows over larger ones. In addition, eight in-network priority queues are leveraged to assign priorities to data according to their flows' sizes. NDP trims the payload of a dropped packet and uses the header to precisely inform the receiver about the drop. By limiting the length of packet queues to a small value, NDP constrains one-way delay and provides deterministic packet loss under ideal scenarios (*e.g.*, congestion only occurs at the edge of the network, and control packets do not drop). Since these optimize on performance are specific and somewhat far from the core design of TCCs, in this paper, we do not dig into these design points.

F. Token Congestion Control

ExpressPass adapts the tokens' sending rate to reduce the token loss ratio. For TCCs generating tokens based on a data-driven manner (*e.g.*, Homa and NDP), token congestion control (CC) is neglected. This section explores when TCCs can benefit from token CC, even without token loss.

Token incast problem. Since receivers schedule the transmission of tokens in a distributed fashion, a receiver could not know whether other receivers are allocating bandwidth for the same sender. When multiple receivers send tokens to the same sender simultaneously, *token incast* could occur. It results in some senders receiving bandwidth allocation which exceeds their forwarding capacity while other senders lack tokens for transmission. Hence, unlike the data incast occurring in RCC protocols which induces a large buffer occupancy, token incast brings under-utilization.

Data queue can still build up. End-to-end bandwidth allocation by tokens together with in-network *rate-limiters* ensure that the bandwidth used by scheduled packets would not exceed the network bottleneck. However, there are still scenarios where data queue can build up.

(i) Tokens could have different RTT (*i.e.*, topology consisting of paths with different base RTTs or encountering token queuing at in-network points). When these tokens arrive at different senders simultaneously, they trigger the transmission of scheduled packets transmission concurrently, which could result in the collision of scheduled packets [10].

(ii) In addition, tokens may not always be used in time. For instance, the sender host is busy sending other data/control packets, or the packet processing of hosts is relatively slow. Then there is a time shift between bandwidth allocation and its usage, resulting in the bandwidth allocation becoming stale. It can also cause the transmission collision.

(iii) When unscheduled packets exist, they consume bandwidth without allocation.

Since state-of-the-art TCCs discuss little on token CC, this paper uses ToCC for illustration (soon be introduced in § V). ToCC adjusts the sending window of tokens according to the congestion state in networks. Figure 3(d) validates that token CC helps to relieve the transmission collision brought by different RTTs of tokens. By leveraging token CC, the data queuing time is reduced, especially benefiting small flows. Hence, their slowdown is significantly reduced.

Figure 7(c) and Figure 7(d) (will be depicted later in § VI-B) validate that token congestion control helps reduce the buffer occupancy brought by unscheduled packets. At first, the transmission rate of unscheduled packets exceeds the network bottleneck and occupies the buffer. With token CC, the sending window of tokens is reduced when the congestion signal arrives at receivers. Hence, the transmission rate of scheduled packets is reduced correspondingly.

Finding 5. *Token CC is necessary to further reduce the data queue length and provide high bandwidth utilization.*

V. ToCC Design and Implementation

Through the anatomization of different design choices of TCCs (§ IV), we find that state-of-the-art approaches do not fully benefit from the design space of TCC. We propose

Fig. 4. ToCC Architecture.

ToCC based on these findings. It targets to provide robust performance across different scenarios.

The last row of Table I depicts the design chosen by ToCC. It is not a trivial composition of the design choices of existing approaches. Firstly, each design choice has its pros and cons. We should select the most appropriate and effective one. Moreover, some should be adjusted carefully to be compatible with each other (*e.g.*, no unscheduled phase and data-driven token generation); while some can be incompatible and should not be used jointly (*e.g.*, rate-based token generation and total volume control of token). ToCC composes these design choices into three major components: sender logic, receiver logic, and switch configuration, as depicted in Figure 4.

A. Sender

Configurable unscheduled phase. The unscheduled phase of ToCC is configurable. The sender can initialize a portion of unscheduled packets or wait for the bandwidth allocation from the receiver. When there is no unscheduled phase, the sender transmits a Request-to-Send (RTS) to notify the receiver. We give our guidance on settings, *i.e.*, when the application is composed of tiny flows (*e.g.*, Memcached), one-BDP unscheduled phase is necessary; when the application is mixed with large flows (*e.g.*, Web Search), giving up the unscheduled phase is a good choice to reduce the buffer occupancy and queuing delay. Besides, the unscheduled phase is more critical to a high link bandwidth (*e.g.*, 100 Gbps) with tiny flows.

Scheduled phase. As for the scheduled phase, the sender transmits a corresponding full-length scheduled packet when receiving a token packet. Besides, the sender is responsible to carry back congestion signals to the receiver to adjust the transmission rate of token packets.

B. Receiver

Data-driven token generation. The receiver of ToCC generates token packets in a data-driven manner and follows the total volume control of tokens to avoid bandwidth waste. Since there can be no unscheduled packets to trigger token generation, the token generation mechanism should be adjusted. The receiver should initialize a portion of token packets (*i.e.*, one BDP) when receiving RTS packets from the sender. Besides, a sending window is set to bound the number of inflight token packets. Instead of requiring flow size as a-priori, ToCC's sender checks whether the unsent data is larger than one BDP when transmitting a data packet and carries the flag in the data packet header. Only when a data packet requiring additional tokens is received, a token is transmitted. Particularly, when applications generate packets in bursts, packets of a flow can arrive intermittently. ToCC treats these intermittent packets as new flows.

Configurable congestion control (CC). CC on token packets is leveraged to reduce the buffer occupancy further. ToCC is flexible to leverage the CC algorithms of RCCs by transferring the rate adjustment on data to tokens. Note that when CC on tokens is leveraged, tokens are generated according to the adjusted sending window rather than a data-driven method with a static sending window.

Currently, we integrate the CC algorithm of DCQCN to ToCC for comparison. Token packets are marked with ECN according to their queue length. Especially, the marking threshold of tokens is reduced by the proportion of a full-length data size *vs.* a token size. When the sender receives token packets marked with ECN, the sender carries congestion signals in data packets instead of using dedicated congestion signals (*e.g.*, CNP). This is because dedicated signals consume unallocated bandwidth that could induce buffer built-up (Appendix A). When the congestion signals arrive at the receiver, the sending window of tokens is reduced according to the CC algorithms. Furthermore, to reduce the buffer occupancy brought by unscheduled packets, ToCC also marks unscheduled packets when their queue length exceeds the ECN marking threshold. When the congestion signals carried in unscheduled packets arrive at the receiver, the sending window of tokens is also adjusted. Note that we do not claim that the CC algorithm used in DCQCN is the most suitable one for ToCC. We will integrate CC algorithms of other RCCs in the future.

Packet loss handling. Packet loss is expected to be rare since ToCC achieves a low-level buffer occupancy. To handle rare packet loss caused by packet corruption, ToCC's receivers maintain Packet Sequence Numbers (PSNs) and timeout to detect and recover from packet loss. When token loss is detected, the receiver retransmits the corresponding number of tokens. Token loss is transparent to ToCC's sender. When data loss is detected, the sender is notified by tokens carrying corresponding PSNs of lost data packets.

C. Switch

In-network rate-limiters. Switches are configured with rate-limiters to make tokens compete for bandwidth at the network bottleneck. The value of rate-limiters is calculated according to the guidance that the bandwidth used by tokens and data packets equals the link speed. Hence, the rate-limiters are set to $TokenSize/(DataSize + TokenSize)$. Rather than dropping token packets when their arriving rate exceeds the limited rate, ToCC lets tokens queue at the network bottleneck. It is reasonable since tokens generated in a data-driven manner do not overwhelm the switch, and the token queue can be further reduced by token CC.

D. Implementation

We implement ToCC in our NP-based smart NICs, as shown in Figure 5. It consists of the data and token transmission components. The transmission of token packets is adjusted by token CC, and the transmission of scheduled packets is controlled by the receiving rate of tokens. Token and data packets are assigned a dedicated queue. The implementation challenges and details are discussed below.

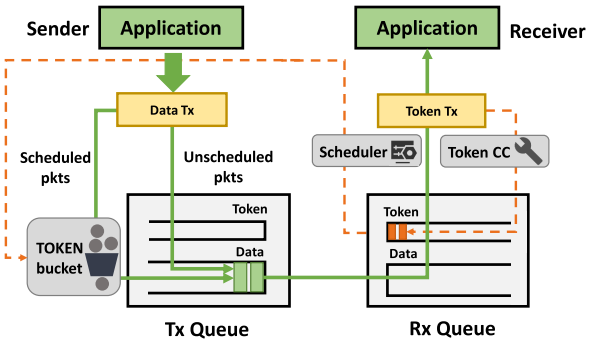


Fig. 5. ToCC implementation.

Limitations on programmable NICs' sending/receiving ability can cause throughput downgrade. Bandwidth under-utilization is observed when experiments are conducted on TCC, especially when running HPC-like workloads consisting of tiny messages. For that programmable NIC is inadequate to provide a line-rate bandwidth when handling a large number of token packets whose size is relatively small. In our testbed environment, the minimum size of a RoCE frame is 94 bytes. A RoCE frame is made up of: 14B (Eth) + 20B (IP) + 8B (UDP) + 12B (BTH) + 4B (ICRC) + 4B (VLAN) + 24B (inter-frame interval and FCS) = 86B. Besides, 8B (payload) is added for subsequent features. A tiny packet consumes the same pps as a full-length packet while consuming a smaller bandwidth. In addition, TCC's end-host is responsible to transmit and receive token packets (a minimum RoCE frame), which also consumes its sending/receiving ability. To decrease the pressure put on NICs' pps, we reduce the bandwidth consumed by tokens by inducing *allocation ratio* K, *i.e.*, the total data bytes that can be transmitted when a sender receives a token. Generally, we set K to larger than 2 KB.

When only one TOKEN bucket is available. In a typical programmable NIC, a flow has only one TOKEN bucket to schedule its transmission. For that the term TOKEN here differs from the 'token packet' in TCC. To avoid ambiguity, capital letters are used for the term TOKEN.

In TCC, token and data packets have different priorities. Therefore, they cannot be scheduled by one TOKEN bucket as a whole. To solve this problem, we use a timer to schedule the transmission of token packets instead. The defect of this alternative is that the precision of the timer is limited.

Handling tokens not used in time. When the tokens received by senders are not used in time, the bandwidth allocation is stale (§ IV-F). The data packets transmitted by using stale bandwidth allocation could cause transmission collision with other data packets, which may lead to buffer built-up. Therefore, it should be careful to use stale tokens. Typically, a relatively large timeout value (*e.g.*, 200μs) is set to limit the maximum duration a token can stay on the sender. In addition, a real-time BDP value is calculated by using the transmission rate of token packets. Tokens that trigger data packets exceeding one BDP are discarded.

Takeaway 1. *Rate-limiter configuration.* To improve the network goodput and solve the problem brought by hardware limitations, one token allocates bandwidth for K bytes. Typically, K ranges from 2 KB to 8 KB. It is acceptable to compromise precision on bandwidth allocation to an extent.

Takeaway 2. *Requirements for our next version of programmable NICs.* To improve the performance of ToCC, it is also important to upgrade NICs' capabilities. Rather than using a timer to imitate the TOKEN bucket, we consider implementing two buckets for scheduling in our next NIC versions.

VI. Evaluation

In this section, we first compare ToCC with state-of-the-art TCCs such as Homa, NDP, ExpressPass and EP+Aeolus to validate that ToCC is robust across different scenarios (§ VI-A). Then, ToCC is chosen as a representative of TCC to compare with RCCs. We provide an analysis of ToCC's key advantages over RCCs (§ VI-B). Then, ToCC is compared with RCCs such as DCQCN, TIMELY, and HPCC through large-scale simulations (§ VI-C) and testbed evaluations (§ VI-D). The authors' contributed code and recommended parameter settings are used [31], [32], [33], [34], [35]. There are two versions of ToCC according to whether configured with unscheduled packets or not. ToCC (w/o unsch) denotes ToCC configuring no unscheduled stage. *In our description, We may use* ToCC *to stand for two versions of* ToCC.

The key results are summarized here.

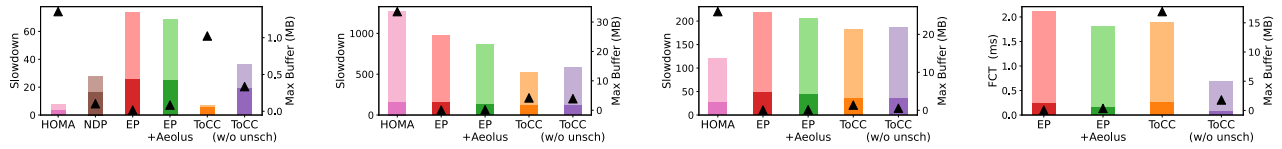
- Key advantages of ToCC: it converges quickly without relying on CC algorithms; it is relatively insensitive to parameter settings; it achieves better performance when leveraging the same CC algorithms as RCCs.
- ToCC is flexible for higher bandwidth links, and ToCC $w/o unsch) is robust for incast scenarios, benefiting from its freedom on unscheduled stages.
- ToCC benefits small flows by achieving a small queue length as well as benefiting large flows by obtaining stable high throughput.
- ToCC achieves a relatively small deviation of FCTs.

Workloads. Poisson arrival flows with a load of 0.8 are generated (§ III-B). Performance under load 0.2 to 0.7 are left in Appendix B. In incast scenarios, we generate non-incast flows following a Poisson arrival process with a load of 0.8 and periodic incast flows with a load of 0.2 (the average load of the incast destination host). The size of incast flows is 30 MTU-40 MTU by default.

Topologies. The default one is a 2-level leaf-spine network that contains 4 core switches, 10 ToRs, and 160 hosts (similar to that in [12]). Each ToR connects to hosts/cores via 100 Gbps links, *i.e.*, the oversubscription ratio of the network is 4:1. The base RTT and Bandwidth-Delay-Product (BDP) are 5.3μs and 65.8 KB, respectively. Links are decreased to 10 Gbps to dig into the performance variation among different bandwidth links. The links connecting ToR and core switches are increased to 400 Gbps/40 Gbps to construct non-blocking topologies. In incast scenarios, non-blocking topologies are used to focus on the congestion caused by incast flows.

A. Comparison Among TCCs

ToCC is compared with state-of-the-art TCCs under Poisson arrival and incast scenarios, as shown in Figure 6.



(a) Memcached. (b) Cache Follower. (c) Web Search. (d) Web Server mixed with incast.

Fig. 6. Comparing with TCCs under 100 Gbps. The triangles denote the max buffer occupancy. The deep/light color represents the average/99th-tail value for each bar.

For that § 2 already discussed their FCT performance of them, in this section, we mainly use slowdown metric for illustration.

Realistic workloads composed of Poisson flows. We evaluated TCCs in different workloads with Possion arrival flows, as shown in Figure 6(a)-6(c). Homa benefits from its SRTF scheduling policy. Hence it achieves good performance for Memcached which is mostly composed of tiny flows, as demonstrated in Figure 6(a). However, the benefits of Homa's priority scheduling diminish with an oversubscribed topology. Homa faces packet loss and timeout retransmission under workloads composed of larger flows. Hence, its performance downgrades under Cache Follower.

For NDP, it actively drops packets when the queue length exceeds a small threshold. The header of packets is sent to the receiver to notify the packet loss. Then PULL packets are sent at line rate for retransmission. It handles congestion that occurs at the last hop well since the transmission rate of PULL packets exactly matches the line rate of the last hop. However, NDP falls short in the presence of bottlenecks in the network core (as opposed to congestion happening at the edge), *e.g.*, due to oversubscription of the topology or poor load balancing. Since receivers do not usually have visibility on this, this may result in higher-than-required PULL generation and result in more severe network congestion. Under Memcached, flows benefit from a small queuing delay. However, under other workloads, the massive drop of data packets prolongs the slowdown of flows to a large extent. To present the performance of different algorithms more clearly, we put the results of NDP in Appendix B.

For ExpressPass, it schedules every data packet to avoid network congestion. Therefore, it achieves a relatively small buffer among different scenarios, at the cost of prolonging the slowdown of flows, especially under Memcached.

ToCC achieves a relatively small buffer occupancy, reducing the average and tail latency across different workloads. To improve the performance of small flows, utilizing the first RTT and achieving low buffer occupancy are both important. Under Memcached, when ToCC is initialized with unscheduled packets, ToCC benefits from utilizing the first RTT. Under Cache Follower and Web Search, ToCC and Homa both transmit the same portion of unscheduled packets, while ToCC reduces the buffer occupancy by 3.2×. It indicates that ToCC's design choices of scheduled phase (*e.g.*, rate-limiters and token congestion control) are efficient. ToCC (w/o unsch) further reduces the buffer occupancy, thus achieving a smaller slowdown compared with ToCC. It means that under stressful workloads where traffic is mixed with large flows, ToCC (w/o unsch)'s strategy that giving up the first-RTT to get a lower queue length works well.

Incast scenarios. Figure 6(d) demonstrates the performance of different TCCs when Web Server flows are mixed with incast flows. ToCC utilizes the design space of TCC. It leverages the freedom of TCC to configure unscheduled stage. Under bursty scenarios, ToCC can choose to schedule every data packet. In scenarios where non-incast flows are mixed with incast flows, as shown in Figure 6(d), ToCC (w/o unsch) obtains a low buffer occupancy. Benefiting from data-driven token generation, ToCC reduces the average and tail FCTs compared with ExpressPass and EP+Aeolus.

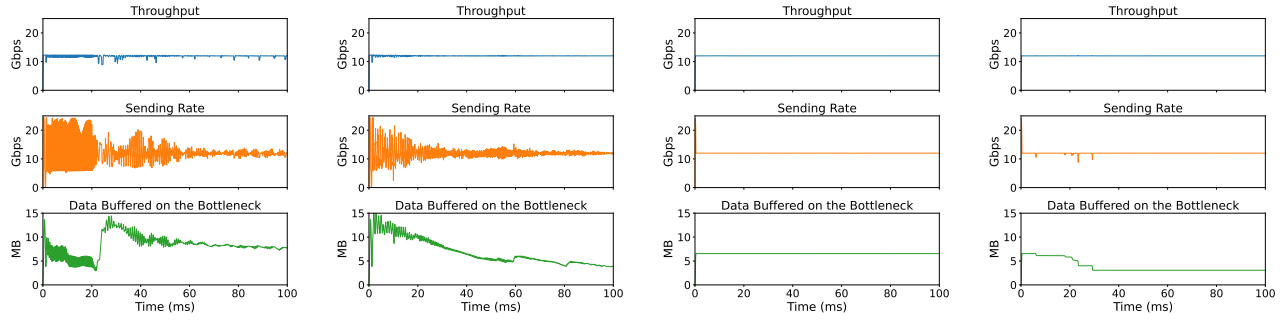
B. Key Advantage Over RCCs

When ToCC leverages the same CC algorithm as DCQCN, ToCC can achieve better performance. This is rooted in the token-based bandwidth allocation manner of the TCC approach.

ToCC converges quickly. TCC leverages token packets to allocate bandwidth for each *scheduled* packet precisely. Therefore, the bandwidth used by scheduled packets can utilize just the right amount of the bottleneck bandwidth even before tokens converge to a stable rate. To validate the benefit brought by token, a simulation is conducted under all-to-all scenarios, where eight flows pass through one 100 Gbps bottleneck link simultaneously. Figures 7(a) and 7(c) depict the results of DCQCN and ToCC (w/o token CC), respectively. For DCQCN, it endures a long process before the CC algorithm converges. Both the sending rate and buffer occupancy are unstable. The sending rate varies between 5 Gbps to 25 Gbps, and the convergence target is 12.5 Gbps. The maximum buffer occupancy can reach up to 14.3 MB. And the throughput is also unstable because of the jittering sending rate. For ToCC, the sending rate on each host is stable, benefiting from the token bandwidth allocation mechanism. Therefore, the network throughput is high and stable. Meanwhile, the buffer occupancy is smaller than 6.5 MB and unscheduled packets take up the buffer. Because no CC algorithm is used, the queue length does not decrease until flows finish.

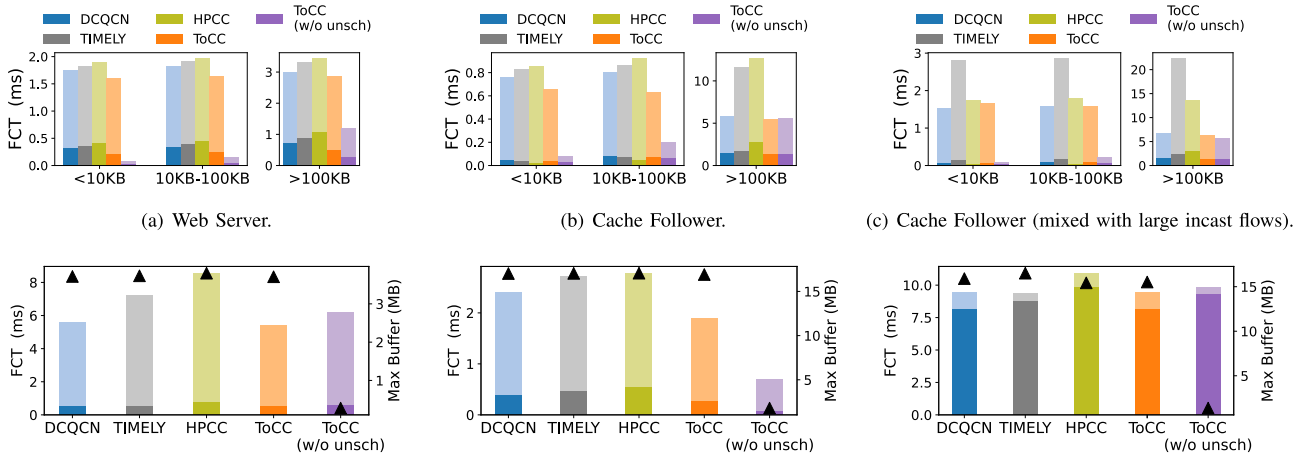
Parameters in DCQCN are hard to tune, while ToCC is relatively insensitive to the parameters. CC parameter tuning is always complicated and time-consuming [36], [37]. In DCQCN, there are dozens of parameters. Although there is a set of recommended parameter settings [5], [35], they do not work well across variable conditions. It consumes much time and labor work to validate an appropriate set of parameter settings. Figure 7(a) and 7(b) compare the performance of DCQCN under two sets of parameter settings.[1] When the

[1]The first one has a more aggressively addictive rate-increase value with a larger time period between rate-increase events, while the second one is contrary. In the first set, rpg_time_reset = 900 μs, rpg_ai_rate = 50 Mb/s, rpg_hai_rate = 100 Mb/s. In the second set, rpg_time_reset = 300 μs, rpg_ai_rate = 5 Mb/s, rpg_hai_rate = 50 Mb/s.



(a) DCQCN using the first para. set. (b) DCQCN using the second para. set. (c) ToCC w/o token CC. (d) ToCC w/ token CC using the first para. set.

Fig. 7. Performance of DCQCN and ToCC under all-to-all traffic. Two set of parameters are used, the first one is recommended parameters in a customized storage [6], the second one is recommended parameters provided by Mellanox [35].



(a) Web Server. (b) Cache Follower. (c) Cache Follower (mixed with large incast flows).

(d) Web Server (10 Gbps). (e) Web Server. (f) Incast flows mixed with Cache Follower.

Fig. 8. Comparing with RCCs under incast-mix.

second parameter set is used, the converging process is faster than the first one, *i.e.*, the sending rate converges before 40 ms. Bandwidth is more utilized, *i.e.*, throughput downgrade only happens at first. However, its buffer occupancy has a larger variation, *i.e.*, from 5 MB to 15 MB.

ToCC is relatively insensitive to different parameters, which is preferred to industries. Figure 7(d) depicts the performance of ToCC, which leverages the same congestion control algorithms used in DCQCN. The performance difference of ToCC is negligible between the two sets of parameters. Thus this paper provides the second one in Appendix B.

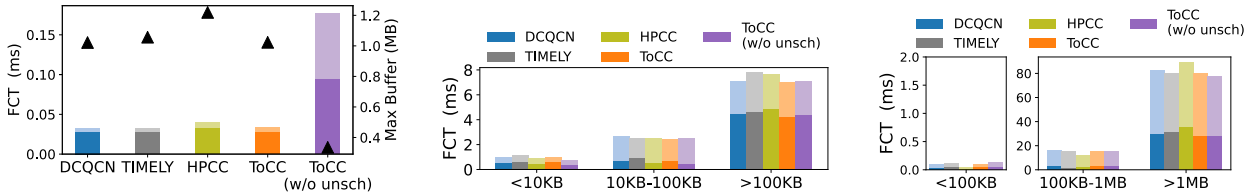
ToCC has fewer requirements on CC algorithms compared to RCCs. More accurate congestion signals bring better network performance. Nevertheless, it has been found that ToCC does not require accurate congestion signals or fine-tuned CC algorithms on tokens to achieve fast convergence and good stability. Firstly, scheduled packets are transmitted according to the bandwidth allocation even before the CC algorithm converges. Secondly, tokens instead of data are queuing when the network bottleneck exists. Token queuing costs less than data queuing. For that, the token queue is much smaller than the data queue and will not be directly transferred to the data queue. Figure 7(a) and Figure 7(d) show that TCC can already achieve better performance than DCQCN by leveraging the same CC algorithm as DCQCN.

C. Performance Comparison With RCCs

ToCC is compared with RCCs under Poisson arrival and incast scenarios. Flows are classified into tiny (<10 KB), small (10 KB-100 KB), medium (100 KB-1 MB), and large (>1 MB).

Incast scenarios. Figure 8 evaluates the performance of ToCC and RCCs under incast scenarios. Results show that ToCC tends to achieve better performance than RCCs. This is because ToCC utilizes the design space of TCC and benefits from the configurable design of the unscheduled phase. In incast scenarios, unscheduled packets of incast flows contribute to network congestion and dominate the buffer occupancy. Fortunately, ToCC can choose to schedule every data packet to avoid incast occurring. Hence, ToCC (w/o unsch) obtains low average and tail latency of Poisson flows. In addition, even with unscheduled phase, ToCC can reduce FCTs compared to RCC approaches, for that ToCC benefits from the stable bandwidth allocation provided by tokens.

Among RCCs, DCQCN can achieve better performance than TIMELY and HPCC. There are two main factors to the performance downgrade of non-incast flows. The first factor is the bursty incast flows. With RCCs, non-incast flows sharing the same output queue with incast flows endure a long queuing delay. Further triggered PFCs can spread the incast congestion to the whole network. The second factor is the reaction of



(a) Memcached. (b) Web Server. (c) Web Search.

Fig. 9. Comparing with RCCs under Poisson arrival scenarios (100 Gbps).

the CC. After congestion signals arrive at senders, the transmission rate of non-incast flows is reduced. Already injected unscheduled packets of incast flows can not be controlled by the transmission adjustment, only making the transmission rate of scheduled packets be reduced. Since HPCC and TIMELY are more sensitive to network congestion, they are more likely to overreact to bursty congestion, *i.e.*, the transmission rate of non-incast flows is reduced aggressively. This especially hurts large flows. As shown in Figure 8(a) and 8(b), with HPCC and TIMELY, flows larger than 100 KB are prolonged to a larger extent than small flows compared to DCQCN and ToCC. Figure 8(c) and 8(f) demonstrate the performance of non-incast flows and incast flows when increasing the size of incast flows to 90-120 MTU. The non-incast flows of ToCC achieve good performance. Meanwhile, the performance of incast flows is not compromised. The performance of TIMELY downgrades, for that TIMELY suffers from the infinite fixed points problem.

When link bandwidth varies. With the growth of link bandwidth, the control of the unscheduled phase becomes more necessary. Under 100 Gbps link bandwidth, flow with the same size could result in transient congestion, making the network become more bursty. Hence, ToCC (w/o unsch) improves the performance of non-incast flows more significantly, as shown in Figure 8(d) and 8(e).

Realistic workloads composed of Poisson flows. Figure 9(a) depicts the performance of Memcached. As discussed in § IV-A, the transmission of unscheduled packets plays an important part in reducing FCTs of small flows. Correspondingly, CC is a less critical factor since it can only control scheduled packets. Therefore, RCCs and ToCC all achieve good performance. Among them, the performance of HPCC is a little downgraded for that it leverages the INT header which brings network overhead and a larger queue length.

Figure 9(b) -9(c) demonstrate the performance under two other workloads. HPCC benefits from its precise in-network congestion signals brought by INT. Therefore, it achieves a small buffer occupancy and reduces the FCTs of small and medium flows. HPCC faces the overreaction to transient congested traffic. Hence, its tail latency is prolonged. ToCC benefits from its token-based bandwidth allocation. It obtains fast convergence and a relatively small buffer occupancy. Besides, ToCC leverages token CC to further reduce the buffer occupancy. Hence ToCC reduces the FCTs of small and medium flows compared to other RCCs except for HPCC. Besides, ToCC achieves a lower tail latency for large flows, indicating a stable high throughput.

D. Testbed Results

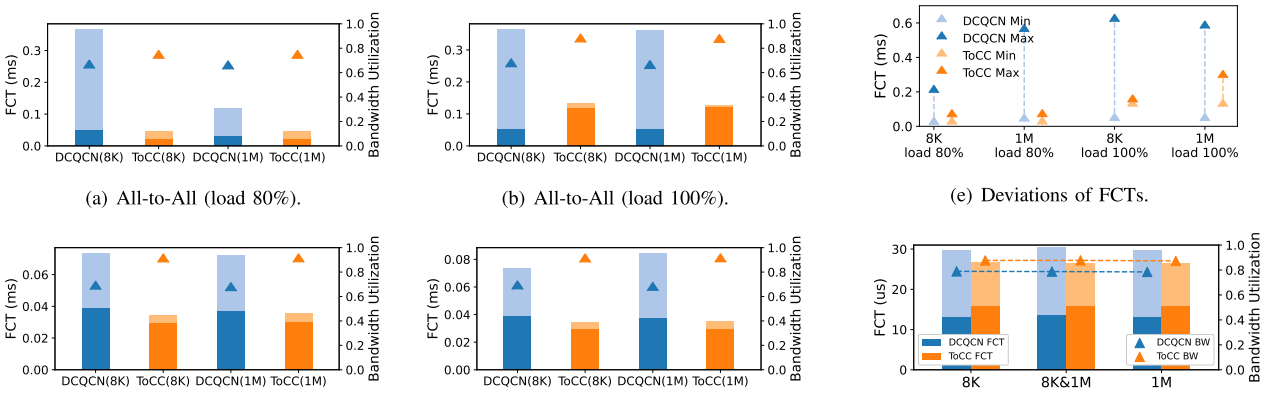
Topology. A 2-tier clos network containing one core switch, two ToRs, and 32 hosts (*i.e.*, 16 hosts per rack) is used. Each host is connected to a ToR via a 25 Gbps link. Each ToR is connected to the core switch via a 200 Gbps link. Each server is equipped with a 25 Gbps NP-based smart NIC.

Workloads. Perftest [38], a standard network benchmarking tool for OpenFabrics Enterprise Distribution (OFED) on the RDMA, is used to generate distributed storage-like traffic. *ib_write_bw* and *ib_write_lat* are exploited to generate background bandwidth-sensitive and latency-sensitive flows [39]. The size of flows is either 8K or 1M. In all-to-all and incast scenarios, each server connects to other servers via 8 and 32 Queue-Pairs (QP), respectively.

All-to-All scenarios (load 80%). Figure 10(a) depicts the throughput of bandwidth-sensitive background flows and the FCTs of latency-sensitive flows under 80% load. Performance under different size background flows shows the same trends. ToCC achieves a 12.1% higher average throughput than DCQCN, benefiting from the precise bandwidth allocation to ensure a stable network throughput. ToCC reduces the average and tail latency of latency-sensitive flows by 52.4% and 7.8× compared with DCQCN, respectively. In ToCC, the bandwidth used by background flows is relatively stable. Therefore the latency-sensitive flows can pass through the network quickly. Figure 10(e) demonstrates that ToCC's FCTs are similar, with a relatively small deviation between FCTs. This property is suitable for coflow applications.

All-to-All scenarios (load 100%). Figure 10(b) shows the performance under highly stressful workloads, *i.e.*, the transmission rate of the bandwidth-sensitive load is not limited on end-hosts. Bandwidth-sensitive flows can utilize 88% bandwidth, while DCQCN only utilizes 63%. It is because DCQCN's large buffer occupancy triggers PFC frames. HOL blocking brought by PFC influences its network throughput. ToCC's average FCTs on latency-sensitive flows are larger compared with DCQCN. However, it is somehow unfair because DCQCN's background flows are significantly under-utilized, which can benefit latency-sensitive flows. Nevertheless, ToCC reduces the tail latency.

All-to-Many scenarios. Two kinds of all-to-many traffic are conducted, *i.e.*, random intra-rack servers or inter-rack servers. Similarly, the network throughput is highly utilized by ToCC. The network is less congested than in all-to-all scenarios since the number of active QPs is decreased. Besides, flows spend less time waiting for scheduling at sender end-hosts. Hence, the FCT for flows is reduced compared to that in all-to-all (load 100%) scenarios. Nevertheless, ToCC achieves



(a) All-to-All (load 80%). (b) All-to-All (load 100%). (e) Deviations of FCTs.

(c) All-to-Many (load 100%, intra-rack). (d) All-to-Many (load 100%, inter-rack). (f) 31:1 Incast scenarios.

Fig. 10. Performance of DCQCN and ToCC in testbed. The triangles denote the bandwidth utilization. The deep/light color represents the average/99.9th-tail value for each bar, respectively.

better performance on average FCTs, and the tail latency is significantly reduced compared with DCQCN.

Pure incast scenarios. Figure 10(f) demonstrates the results under pure 31:1 incast. The throughput of ToCC is stable and fully utilized across different background flow size. The under-utilization of DCQCN is not caused by PFC, since for pure incast, PFC does not influence the throughput. Instead, DCQCN waits for CC algorithms to converge. In addition, ToCC's rate-limiter determines the maximum utilization for data. In one-sided traffic scenarios like pure incast, the bandwidth reserved by tokens could be wasted (§ IV-C).

VII. Discussions

Related works. FlexPass [11] tries to solve the co-existence of TCC and RCC traffic. On-Ramp [40] is a patch to current RCCs. It divides the network into transient and equilibrium states. It handles the transient state of the network by leveraging accurate measurements of one-way delay, leaving the equilibrium state for RCCs. dcPIM [41] tries to provide a full match between senders and receivers, guaranteeing near-optimal utilization with a constant number of rounds. dcPIM assumes that the flow size is known in advance. And it wastes several RTTs to match before transmission hence it hurts the performance of middle-sized flows (since in dcPIM, small flows can be transmitted without waiting for matching). Apart from CC protocols, per-hop flow control protocols [22], [27] can react to congestion in a quicker way. Hence, it reduces the buffer occupancy significantly and ensures zero packet loss. However, flow control protocols tame the transmission of flows in a per-queue manner, which might involve HOL-blocking and congestion spreading.

Cooperating with priority queues. TCC can leverage priority queues to support SLO (service level objective). While Homa assigns priorities to data packets directly, there are other approaches [42] that choose to assign priorities to tokens instead, where data packets use one queue, and tokens use the remaining queues according to the priority of the corresponding data. The main benefit brought by assigning priorities to tokens is that it can avoid blocking data packets in a low-priority queue, which could result in packet loss. Besides, in the presence of the rate-limiters on tokens, it avoids flows with a high priority waiting for tokens for a long time.

Transmission coordination between unscheduled and scheduled packets. One may wonder whether the rate-limiter of tokens should be reconfigured to different values when unscheduled packets exist in the network. However, reconfiguring the rate-limiter in running time is not trivial. Fortunately, congestion control on tokens can be aware of the existing congestion in networks (§ V-B). Besides, there is work [43] already noticing the over-injection of scheduled packets with a fixed rate-limiter setting. It coordinates the transmission between unscheduled and scheduled packets via back-pressuring the network when unscheduled packets carrying congestion signals are received.

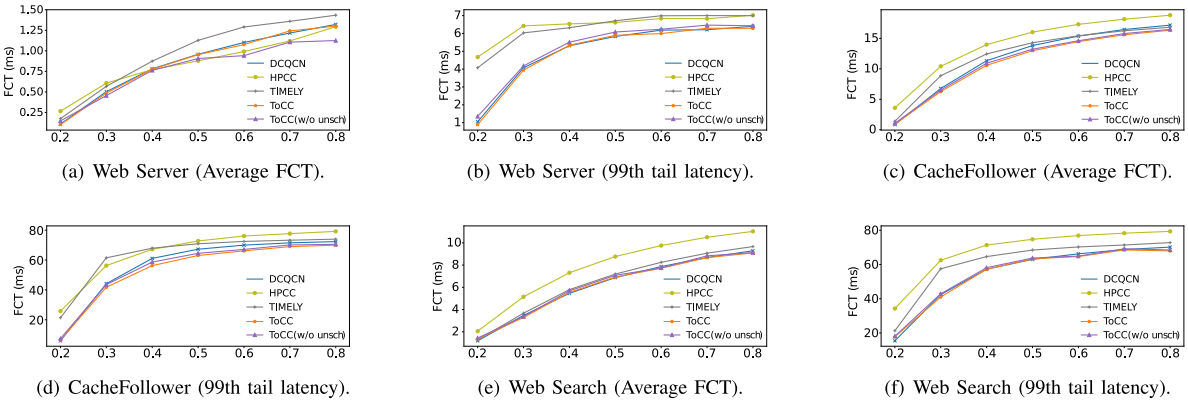
VIII. Conclusion

This paper explores the design space of token-based proactive congestion control (TCC) protocols by systematically analyzing the design choices of state-of-the-art approaches. Then we present ToCC, a token-oriented CC utilizing the design space of TCC. It provides configurable design components (*e.g.*, unscheduled packets and CC algorithms) and guidance to configure them. Evaluations show that ToCC converges quickly and provides a stable high throughput. We envision that ToCC could motivate further exploration of TCC.

Appendix

A. Detailed Design

When integrating DCQCN congestion control algorithm to ToCC, we first use dedicated CNP packets to carry back ECN-marking signal of token queue. Figure 12 shows the buffer occupancy at the network bottleneck. Instantaneous buffer burst occurs several times. Owning to the fact that the bandwidth used by CNP packets is unexpected and unallocated. The perfect allocation of token/data packets does not aware of bandwidth used by CNP, which can result in the total bandwidth usage exceeds the network bottleneck. To this end, ToCC leverages data packets to carry back congestion status of token queue. An alternative is that configuring the rate-limiter to a smaller value, allowing the bandwidth consumed by token and data packets slightly beneath the network bottleneck.



(a) Web Server (Average FCT). (b) Web Server (99th tail latency). (c) CacheFollower (Average FCT).

(d) CacheFollower (99th tail latency). (e) Web Search (Average FCT). (f) Web Search (99th tail latency).

Fig. 11. Performance of ToCC and RCCs across different load.

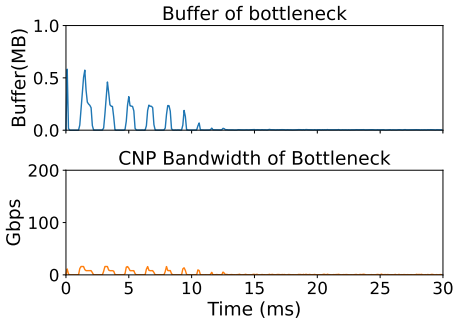
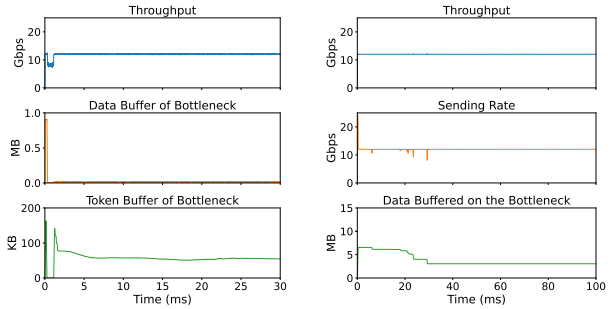


Fig. 12. Detailed design choice: leveraging CNP to carry congestion signals.



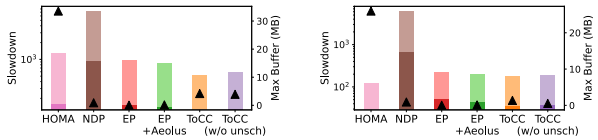
(a) Parameter Selection. When a low ECN-marking threshold is used. (b) ToCC w/ token CC using the second para. set.

Fig. 13. ToCC with different para. set.

B. Supplemental Evaluation

Different parameters for congestion control. Figure 13(b) shows the performance of ToCC when using the second parameter settings. The performance is relatively the same as ToCC using the first parameter settings, as shown in Figure 7(d).

Parameter selection. When using the recommended ECN marking threshold (*i.e.*, K_{min} = 22KB, K_{max} = 85KB, P_{max} = 0.2 for 100G links), ToCC achieves high throughput and relatively small buffer occupancy (§ VI). Figure 13(a) shows the result of ToCC when ECN marking threshold is set to half of the recommended value. A period of time (*i.e.*, hundreds of microseconds) of an empty token queue results in an empty data queue. Hence, the network throughput downgrades for a short period of time. Given that token queuing has a relatively small impact on network performance



(a) Cache Follower. (b) Web Search.

Fig. 14. Comparing with TCCs under 100 Gbps.

(§ IV-F), we recommend that the ECN marking threshold of token queue is set to a conservative value.

NDP performance under Poisson arrival flows. Figure 14 shows the performance of NDP under Cache Follower and Web Search. As discussed in § VI-A, NDP suffers from a large drop ratio by setting an aggressive threshold to cut packets' payload under oversubscribed networks. Hence, the FCT of flows is severely prolonged.

When network loads vary. We evaluate the performance of different protocols under a wide range of network loads from 20% to 80%, as shown in Figure 11. Figure 11(a) shows the result of the average FCT of Web Server workload. As the load increases, the gap of average FCTs between different protocols becomes larger. And ToCC (w/o unsch) achieves a stable good performance across the different load. It is worth noticing that when load increases, ToCC (w/o unsch) performs better than ToCC. This is because under a high load, giving up the unscheduled phase provides a lower queue length as well as a lower queuing delay. Figure 11(b) shows the result of tail latency. At a light load (*e.g.*, 20%), the tail latency of HPCC and TIMELY is prolonged compared to other approaches. And the gap shrinks as the network load increases. It indicates that the aggressive rate adjustment is sensitive to micro-bursts, hurting the throughput especially under light load scenarios.

Different from Web Server workloads, for Cache Follower and Web Search workloads, the performance of ToCC and ToCC (w/o unsch) are much more closer. This is because the average size of Cache Follower and Web Search are larger, composing less fraction of unscheduled packets. HPCC prolongs the average FCT and tail latency, for that it is sensitive to transient congestion which could lead to the throughput downgrade.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM Conf. SIGCOMM*, 2015, pp. 123–137.

[2] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, 2010, pp. 267–280.

[3] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. ACM IMC*, 2009, pp. 202–208.

[4] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proc. ACM SIGMETRICS*, 2012, pp. 53–64.

[5] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," in *Proc. ACM SIGCOMM*, 2015, pp. 1–14.

[6] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 44–58.

[7] V. Addanki, O. Michel, and S. Schmid, "PowerTCP: Pushing the performance limits of datacenter networks," in *Proc. NSDI*, 2022, pp. 1–21.

[8] R. Mittal et al., "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM SIGCOMM*, 2015, pp. 1–14.

[9] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. ACM SIGCOMM*, 2020, pp. 514–528.

[10] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM*, 2017, pp. 239–252.

[11] H. Lim, J. Kim, I. Cho, K. Jang, W. Bai, and D. Han, "FlexPass: A case for flexible credit-based transport for datacenter networks," in *Proc. 18th Eur. Conf. Comput. Syst.*, May 2023, pp. 606–622.

[12] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. ACM SIGCOMM*, 2018, pp. 221–235.

[13] J. Ousterhout, "A Linux kernel implementation of the homa transport protocol," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 1–16.

[14] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. ACM SIGCOMM*, 2017, pp. 29–42.

[15] 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 *Proc. ACM CoNEXT*, 2015, pp. 1–12.

[16] S. Hu et al., "Aeolus: A building block for proactive transport in datacenters," in *Proc. ACM SIGCOMM*, 2020, pp. 422–434.

[17] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2011, pp. 63–74.

[18] *Congestion notification*, IEEE Standard 802.11Qau, 2010.

[19] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proc. ACM CoNEXT*, 2012, pp. 25–36.

[20] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "ECN or delay: Lessons learnt from analysis of DCQCN and TIMELY," in *Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2016, pp. 313–327.

[21] S. Arslan, Y. Li, G. Kumar, and N. Dukkipati, "Bolt: Sub-RTT congestion control for ultra-low latency," in *Proc. NSDI*, 2023, pp. 1–19.

[22] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proc. NSDI*, 2022, pp. 1–3.

[23] D. Gibson et al., "Aquila: A unified, low-latency fabric for datacenter networks," in *Proc. NSDI*, 2022, pp. 1–19.

[24] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proc. NSDI*, 2019, pp. 1–17.

[25] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," in *Proc. ACM SIGCOMM*, 2012, pp. 139–150.

[26] M. Alizadeh et al., "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM*, vol. 43, 2013, pp. 435–446.

[27] IEEE. (2011). *802.11Qbb. Priority Based Flow Control*. [Online]. Available: https://1.ieee802.org/dcb/802-1qbb/

[28] *Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 7.X*, Cisco, San Jose, CA, USA, 2021.

[29] Broadcom. (2018). *BCM53154*. [Online]. Available: https://docs.broadcom.com/doc/53154-PB100

[30] Juniper. (2021). *Configuring Rate Limits on Enhanced Queuing DPCs*. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/cos/topics/concept/cos-configuring-rate-limits-on-enhanced-queuing-dpcs.html

[31] Alibaba. (2019). *HPCC Simulator*. [Online]. Available: https://github.com/alibaba-edu/High-Precision-Congestion-Control

[32] KAIST. (2017). *Expresspass Simulator*. [Online]. Available: https://github.com/kaist-ina/ns2-xpass

[33] Stanford. (2018). *Homa Simulator*. [Online]. Available: https://github.com/PlatformLab/HomaSimulation

[34] U. C. London. (2017). *NDP Simulator*. [Online]. Available: https://github.com/nets-cs-pub-ro/NDP/wiki/NDP-Simulator

[35] Mellanox. (2020). *DCQCN Parameters*. [Online]. Available: https://community.mellanox.com/s/article/dcqcn-parameters

[36] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, "ACC: Automatic ECN tuning for high-speed datacenter networks," in *Proc. SIGCOMM*, 2021, pp. 384–397.

[37] V. Arun, M. T. Arashloo, A. Saeed, M. Alizadeh, and H. Balakrishnan, "Toward formally verifying congestion control behavior," in *Proc. SIGCOMM*, 2021, pp. 1–16.

[38] Perftest. (2021). *Open Fabrics Enterprise Distribution (OFED) Performance Tests README*. [Online]. Available: https://github.com/linux-rdma/perftest

[39] Mellanox. (2019). *Perftest Package*. [Online]. Available: https://community.mellanox.com/s/article/perftest-package

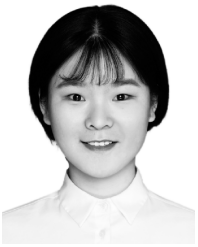
[40] S. Liu, A. Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum, and A. Sivaraman, "Breaking the transience-equilibrium nexus: A new approach to datacenter packet transport," in *Proc. NSDI*, 2021, pp. 1–18.

[41] Q. Cai, M. T. Arashloo, and R. Agarwal, "dcPIM: Near-optimal proactive datacenter transport," in *Proc. ACM SIGCOMM Conf.*, 2022, pp. 53–65.

[42] K. Liu et al., "Exploring token-oriented in-network prioritization in datacenter networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 5, pp. 1223–1238, May 2020.

[43] K. Liu et al., "PayDebt: Reduce buffer occupancy under bursty traffic on large clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4707–4722, Dec. 2022.

Kexin Liu received the B.S. degree from the Department of Software Engineering, Sun Yat-sen University, China, in 2017, and the Ph.D. degree in computer science and technology from Nanjing University, China, in 2023. She is currently a Senior Research Engineer with Huawei. Her research interests include data center networks, network architecture, and networks for AI.

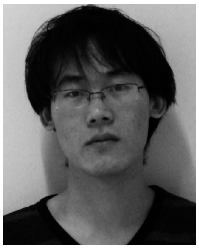


Chang Liu received the B.S. degree from the School of Computer Science and Engineering, Northeastern University, China, in 2021. She is currently pursuing the master's degree with the Department of Computer Science and Technology, Nanjing University, China. Her research interests include programmable switches and data center networks.

Qingyue Wang received the B.S. degree from the Department of Computer Science and Technology, Wuhan University, China, in 2019, and the M.S. degree from the Department of Computer Science and Technology, Nanjing University, China, in 2022. Her research interests include data center networks.

Zhiqiang Li is currently a Senior Engineer with China Mobile Research Institute and the Technical Manager of the Future Network Innovation Laboratory. His research interests include the technology of computing, network convergence, and next-generation IP networks.

Lu Lu (Member, IEEE) is currently the Deputy Director of the Department of Basic Network Technology, China Mobile Research Institute (CMRI), the Leader of the Core Network Group of CCSA TC5, and the Vice Chair of ITU-T SG13. Her research interests include 5G/6G network architecture.



Xiaoliang Wang (Member, IEEE) is currently an Assistant Professor with the Department of Computer Science and Technology, Nanjing University, China, where he is currently an Associate Professor. He has published more than 30 technical papers at premium international journals and conferences, including IEEE TRANSACTIONS ON INFORMATION THEORY, IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE INFOCOM, ACM SIGCOMM, USENIX, NSDI, FAST, and OSDI. His research interests include networking systems and mobile computing.

Fu Xiao (Senior Member, IEEE) received the Ph.D. degree in computer science and technology from Nanjing University of Science and Technology, Nanjing, China, in 2007. He is currently a Professor and the Ph.D. Supervisor with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing. His research papers have been published in many prestigious conferences and journals, such as IEEE INFOCOM, IEEE ICC, IEEE IPCCC, IEEE/ACM ToN, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON MOBILE COMPUTING, ACM TECS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. His research interests include the Internet of Things and mobile computing. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

Ying Zhang (Senior Member, IEEE) is currently the Software Engineering Manager with Meta. She works on large-scale network management problems. She has more than 30 granted U.S./international patents and 50 peer-reviewed publications with about 1500 citations. Her research interests include software-defined networks, network function virtualization, network monitoring, Internet routing, and network security. She was named by Swedish Media as the Mobile Network 10 Brightest Researcher. She was awarded as the Rising Star in the networking and communications area.

Wanchun Dou received the Ph.D. degree in mechanical and electronic engineering from Nanjing University of Science and Technology, China, in 2001. From April 2005 to June 2005 and from November 2008 to February 2009, he visited the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, as a Visiting Scholar. He is currently a Full Professor with the State Key Laboratory for Novel Software Technology, Nanjing University. Up to now, he has chaired three National Natural Science Foundation of China projects and published more than 60 research papers in international journals and international conferences. His research interests include workflow, cloud computing, and service computing.

Guihai Chen (Fellow, IEEE) received the B.S. degree in computer software from Nanjing University in 1984, the M.E. degree in computer applications from Southeast University in 1987, and the Ph.D. degree in computer science from The University of Hong Kong in 1997. He had been invited as a Visiting Professor with the Kyushu Institute of Technology, Japan; The University of Queensland, Australia; and Wayne State University, USA. He is currently a Distinguished Professor with Nanjing University. He has published more than 350 peer-reviewed papers, and more than 200 of them are in well-archived international journals such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, ACM/IEEE TON, and ACM TOSN, and also in well-known conference proceedings, such as HPCA, MOBIHOC, INFOCOM, ICNP, ICDCS, CoNext, and AAAI. His research interests include parallel computing, wireless networks, data centers, peer-to-peer computing, high-performance computer architecture, and data engineering. He has won nine paper awards, including the ICNP 2015 Best Paper Award and the DASFAA 2017 Best Paper Award.

Chen Tian (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He is currently a Professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. Previously, he was an Associate Professor with the School of Electronics Information and Communications, Huazhong University of Science and Technology, China. From 2012 to 2013, he was a Post-Doctoral Researcher with the Department of Computer Science, Yale University. His research Interests include data center networks, network function virtualization, distributed systems, internet streaming, and urban computing.