One More Config Is Enough: Saving (DC)TCP for High-Speed Extremely Shallow-Buffered Datacenters

Wei Bai, Shuihai Hu, Kai Chen, *Senior Member, IEEE*, Kun Tan, and Yongqiang Xiong, *Member, IEEE*

***Abstract*— The link speed in production datacenters is growing fast, from 1 Gbps to 40 Gbps or even 100 Gbps. However, the buffer size of commodity switches increases slowly, e.g., from 4 MB at 1 Gbps to 16 MB at 100 Gbps, thus significantly outpaced by the link speed. In such *extremely shallow-buffered* networks, today's TCP/ECN solutions, such as DCTCP, suffer from either excessive packet losses or significant throughput degradation. Motivated by this, we introduce BCC,[1] a simple yet effective solution that requires only one more ECN configuration (i.e., shared buffer ECN/RED) at commodity switches. BCC operates upon real-time global shared buffer utilization. When available buffer space suffices, BCC delivers both high throughput and low packet loss rate as prior work; When it gets insufficient, BCC automatically triggers the shared buffer ECN to prevent packet loss at the cost of sacrificing a small amount of throughput. BCC is readily deployable with existing commodity switches. We validate BCC's efficacy in a 100G testbed and evaluate its performance using extensive simulations. Our results show that BCC maintains low packet loss rate persistently while only slightly degrading throughput when the buffer becomes insufficient. For example, compared to current practice, BCC achieves up to 94.4% lower 99th percentile flow completion time (FCT) for small flows while only degrading average FCT for large flows by up to 3%.**

***Index Terms*— Datacenter networks, congestion control, ECN.**

I. Introduction

DATACENTER applications generate a mix of workloads with both latency-sensitive small messages (*e.g.*, web search) and throughput-sensitive bulk transfers (*e.g.*, data replication). Hence, datacenter network (DCN) transport should provide low latency and high throughput simultaneously to meet the requirements of applications.

TCP is the dominant transport protocol in today's production datacenters. However it was a challenge for TCP to achieve good performance on both metrics that are essentially at odds, especially under the shared *shallow-buffered* switches in production DCNs. This challenge has been identified almost 10 years ago by Microsoft researchers in their production DCNs. To address it, they proposed DCTCP [11] which leverages ECN [47] to strike the tradeoff between high throughput and low latency, and showed that a properly configured per-port ECN/RED marking scheme could well utilize the shallow buffer to achieve both high throughput and low latency, while still reserving certain headroom for absorbing micro-bursts [11].

Since then, TCP/ECN variants become flourishing [11], [16], [19], [42], [51], [55], [58], [61] and are widely adopted in industry. For example, DCTCP has been integrated into various OS kernels [4], [5] and deployed in DCNs of Microsoft, Google [49] and Morgan Stanley [38].

However, in this paper, we show that this seemingly solved problem resurges and the solution is now being re-challenged, due to the recent industrial trend. The link speed of production DCNs is growing fast from 1Gbps to 40Gbps or 100Gbps, whereas the buffer size of commodity switches increases slowly (*e.g.*, from 4MB at 1Gbps to 16MB at 100Gbps), significantly outpaced by the link speed. Consequently, the buffer size per port per Gbps drops from 85KB to 5.12KB, leading to an extremely shallow-buffered DCN environment (§II-D).

We find it is hard for prior TCP/ECN solutions to remain effective when buffer is extremely shallow (§III). On the one hand, if we configure a standard ECN marking threshold as prior work [11], it will cause packet losses even before ECN starts to react when more ports are active. On the other hand, if we configure a lower conservative ECN threshold, it will waste bandwidth and degrade throughput when few ports are active because ECN over-reacts. Our results (§III-C) show that such dilemma could lead to either 0.34% packet loss rate (thus over 50X higher FCT for short flows) or 7.8% FCT slowdown for large flows.

Our key contribution of this work is to uncover the above problem, demonstrate its consequences, and introduce an effective and readily deployable solution called BCC.[2]

Manuscript received January 22, 2020; revised July 27, 2020; accepted September 23, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Schapira. This work was supported in part by the Hong Kong Research Grants Council (RGC) under Grant GRF-16215119 and in part by the Huawei Research Grant. *(Corresponding author: Kai Chen.)*

Wei Bai was with the iSING Laboratory, The Hong Kong University of Science and Technology, Hong Kong. He is now with Microsoft Research Lab, Redmond, WA 98052 USA (e-mail: baiwei0427@gmail.com).

Shuihai Hu was with the iSING Laboratory, The Hong Kong University of Science and Technology, Hong Kong. He is now with Clustar Technology, Shenzhen 518052, China (e-mail: shuihai@clustar.ai).

Kai Chen is with the iSING Laboratory, Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong (e-mail: kaichen@cse.ust.hk).

Kun Tan is with Huawei Technology, Shenzhen 518129, China (e-mail: kun.tan@huawei.com).

Yongqiang Xiong is with Microsoft Research Asia, Beijing 100080, China (e-mail: yongqiang.xiong@microsoft.com).

Digital Object Identifier 10.1109/TNET.2020.3032999

[1] **BCC** : **B**uffer-aware Active Queue Management (AQM) scheme for **C**ongestion **C**ontrol in extremely shallow-buffered datacenters.

[2] An earlier version has been published in IEEE INFOCOM 2020 [20].

BCC is surprisingly simple, *one more ECN config is enough*! At its core, BCC inherits the success of per-port ECN/RED from DCTCP [11], and further enables shared buffer ECN/RED to cope with the extremely shallow buffer scenario (§IV). The shared buffer ECN/RED follows the original RED algorithm [26] but tracks the occupancy of the global shared buffer to mark packets. We note that while this function is available on chips [3], [9], [19], it was less understood previously and its utility has not been fully exploited in literature.

BCC's shared buffer ECN/RED and per-port ECN/RED work complementarily to each other (§IV-B). When fewer ports are active, the available buffer space suffices, per-port ECN/RED will take effect first to strike the balance of high throughput and low latency as DCTCP [11]. As more and more ports become active, the buffer space becomes insufficient, shared buffer ECN/RED will be triggered first to prevent packet loss—BCC trades little throughput for latency when achieving both is impossible. Furthermore, when applying shared buffer ECN/RED, BCC leverages probabilistic marking (instead of DCTCP's single cut-off marking) to desynchronize flows across ports, so as to avoid global synchronization and further throughput loss.

We have implemented BCC in a small testbed with a 100G Arista 7060CX-32S-F switch connecting 6 servers (§V-A). We show that BCC can be enabled by just one more command at the switch (Figure 6). We note that our implementation mainly validates that BCC is readily-deployable and functional as expected, however, a non-trivial BCC deployment at scale is beyond the scope of this paper. Our goal here is to show the hardware feasibility of BCC, leaving large-scale deployment as future work.

We further evaluate the performance of BCC using large-scale ns-2 simulations with realistic production datacenter workloads (§V-B–§V-C). Our key findings include:

- At low loads (few ports active), BCC fully utilizes the link capacity. Compared to a conservative ECN configuration, BCC achieves up to 19.3% lower average FCT for large flows (§V-B);
- At high loads (more ports active), BCC keeps low packet loss rate while only sacrificing little throughput. Compared to a standard ECN configuration, BCC achieves up to 94.4% lower 99th percentile FCT for small flows while only degrading average FCT for large flows by up to 3% (§V-B);
- BCC can effectively mitigate the global synchronization problem with probabilistic shared buffer ECN marking, and its performance is robust to various parameter settings (§V-C).

The rest of the paper is organized as follows. We introduce the extremely shallow buffer problem and its impacts in §II and §III. We then present the BCC design and evaluation in §IV and §V. We finally discuss related work in §VI and conclude the paper in §VII.

To make our work reproducible, we have made our code available at https://github.com/baiwei0427/buffer-management/.

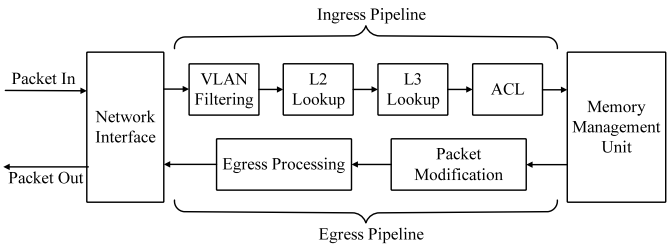


Fig. 1. Packet forwarding pipeline of Broadcom Trident II switching chip.

II. Buffer Is Becoming Extremely Shallow

In this section, we first understand the buffering logic of existing switching chips. Then, we quantify the buffer requirements of TCP[3] at high-speed. Finally, we show that the buffer space becomes increasingly insufficient as link speed increases.

A. Understanding the Switch Buffering

Figure 1 demonstrates a typical packet forwarding pipeline inside the commodity datacenter switching chip. Among the stages, we focus on Memory Management Unit (MMU), which performs packet buffering and scheduling functions.

Basically, the MMU allocates on-chip buffer memory to incoming packets. The buffer memory is divided into several memory pools, which can be generally classified into the following two categories:

- **Private pools**: dedicated buffers that have been reserved to switch egress queues.
- **Shared pool**: buffers that are shared by many (all the) switch egress queues.

When a packet arrives, the MMU uses the following two steps to decide how to buffer it (or drop it):

- **Enqueue into the private pool:** The MMU first checks the private pool of the destination egress queue. If there is enough buffer space, the packet will be enqueued into the private pool. Otherwise, the MMU will move to step 2 for further checking.
- **Enqueue into the shared pool:** If the private buffer has been used up, the MMU will dynamically allocate buffer from the shared pool. If there is still no enough buffer space, the packet will get dropped.

As shown above, packets only get dropped by MMU if neither the private pool nor the shared pool has enough space. Moreover, the MMU only drops new arriving packets. Packets in the pool cannot be pushed out.

B. Dynamic Buffer Allocation

Today's commodity switching chips typically use Dynamic Threshold (DT) algorithm [24] for dynamic buffer allocation. The shared buffer allocated to a queue is controlled by a parameter α. At time t, the MMU will compute a threshold

[3]In this paper, by TCP we refer to various TCP-variants, such as DCTCP [11] and ECN* [55], etc., that are designed for datacenters.

$T(t)$ to limit the queue length. $T(t)$ is actually a function of the unused shared buffer size and α as follows:

$$T(t) = \alpha \times (B - \sum_{i=1}^{N} Q_i(t)) \quad (1)$$

where B is shared buffer pool size and $Q_i(t)$ is length if queue i at time t. A packet arriving in queue i at time t will get dropped if $Q_i(t) \geq T(t)$. As analyzed in [24], if there are M active queues, each queue can eventually get $\alpha \times B/(1 + M \times \alpha)$ buffer space. Obviously, the more active queues we have, the smaller buffer space each queue can get from the shared pool. Moreover, a large α can help a queue to get more buffer space. But a too large α can cause short-term imbalanced buffer allocation. For example, if α is infinitely large, the shared buffer pool will be allocated in a first in first serve manner without any fairness guarantee. In switching chips, α values are typically powers of two for implementation simplicity (*e.g.*, 1/128 to 8 in most Broadcom chips).

C. Buffer Requirement of TCP at High-Speed

TCP is the dominant transport protocol in today's production datacenters [11], [29], [49]. The switch buffer has a great impact on TCP's performance, especially for TCP variants[4] that are designed for datacenter environments [11], [51], [55]. Moderate buffer occupancies are necessary for high throughput [15]. In addition, we also need certain buffer headroom to absorb transient busrts (*e.g.*, incast [11], [17], [36], [52], [54]). Therefore, insufficient switch buffers cause 1) *low throughput*, thus slowing bulk transfers; and 2) *excessive packet losses*, thus resulting in large tail completion times for small messages. The tail completion time really matters because the performance of many responsive large-scale applications seriously degrades when even a small fraction of messages are late [25].

Unlike Internet, typically there are only a small number of concurrent large flows in production DCNs [11]. In such scenarios, to achieve the desired performance, TCP requires at least $C \times RTT \times \lambda$ buffer space per port, where C is the link capacity, RTT is the average round-trip time and λ is a characteristic constant of the congestion control algorithm (*e.g.*, 1 for regular ECN-enabled TCP).

In recent years, the link speed in datacenters has increased greatly, from 1Gbps to 40Gbps and now to 100Gbps. However, the base latency in datacenters does not change much as it is mainly determined by processing overhead from various sources (*e.g.*, kernel stack, NIC driver, and middlebox). Hence, the buffer demand of TCP almost increases in proportion to the link speed in datacenters.

Testbed measurement: We measure the buffer demand of conventional TCP stacks in our testbed. Three servers (Mellanox ConnectX-4 100Gbps NIC, Linux kernel 3.10.0) are connected to an Arista 7060CX-32S-F switch. To reduce system overhead, various optimization techniques, *e.g.*, TCP segmentation offload (TSO) and generic receive offload (GRO), are enabled. The base round-trip time in our testbed is $\sim$30μs.

[4]In this paper, by TCP we generally refer to various TCP-variants, such as DCTCP [11] and ECN* [55], etc., that are designed for datacenters.

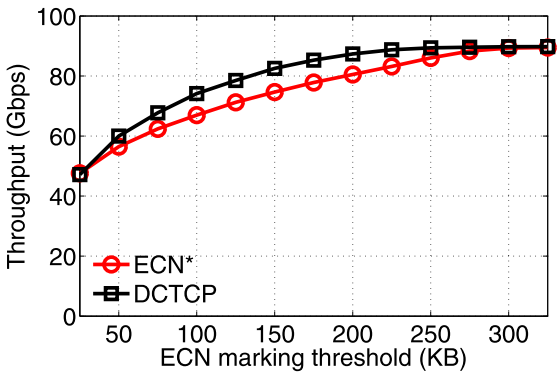


Fig. 2. [Testbed] Aggregate TCP throughput with different ECN/RED marking thresholds.

We consider two datacenter TCP variants: DCTCP [11] and ECN* [55] (regular ECN-enabled TCP which simply cuts window by half in the presence of an ECN mark). We use open source DCTCP implementation from Linux 3.18 kernel. We generate 16 TCP long-lived flows using `iperf` from two senders to a receiver. We vary the ECN/RED marking threshold[5] at the switch and measure the aggregate throughput at the receiver side. For a TCP variant, its basic buffer requirement approximately equals to the minimum ECN marking threshold that achieves 100% link utilization.

Figure 2 shows aggregate throughput results with different ECN marking thresholds. As expected, ECN* starts to achieve 100% throughput on 325KB which is close to the bandwidth-delay product (BDP) in our testbed. Our measurement also shows that DCTCP performs similar as ECN* in practice. The minimum ECN marking threshold that DCTCP requires for 100% throughput is 250KB. The reader may wonder why our experiment observation of DCTCP seems inconsistent with theory results in [12] (0.17$\times$BDP buffering is enough for 100% throughput). We believe this is mainly due to packet bursts that are caused by various interactions between the OS and the NIC (*e.g.*, TSO, GRO). Hence, a much larger ECN marking threshold is required to absorb bursts. Such complex burst behaviors are difficult to capture by ideal fluid model in [12], thus resulting in the theory-practice gap.[6] We also conduct the above experiment using Windows Server 2012 R2 and observe that DCTCP requires $\sim$60-70% BDP buffering for 100% throughput in practice.

Production datacenters: Compared to our simple small-scale testbed, production datacenters are more challenging and have larger base latency. At the end host, packets may experience high kernel processing delay when servers are busy doing CPU-intensive computations [29]. In virtualized environments, hypervisors introduce extra processing overhead. In the network, packets experience innegligible processing delay when going through various middleboxes [27], [43]. Long-distance cables and multiple switch hops also bring several-microsecond delay. Above factors greatly increase the actual latency in production environments. In [30], the authors

[5]We set the maximum and minimum queue length thresholds of RED [26] to the same value as previous work [11], [55] suggests.

[6]Such performance-theory gap has also been identified by previous work [55] and even DCTCP paper itself [11].

TABLE I
BUFFER AND CAPACITY INFORMATION OF COMMODITY DATACENTER SWITCHING CHIPS. NOTE THAT BROADCOM TOMAHAWK HAS 4 SWITCH CORES, EACH WITH ITS OWN MMU AND 4MB BUFFER [1], [2]

ASIC	Broadcom 56538	Broadcom Trident+	Broadcom Trident II	Broadcom Tomahawk	Barefoot Tofino
Capacity (ports × BW)	48 p × 1 Gbps	48 p × 10 Gbps	32 p × 40 Gbps	32 p × 100 Gbps	64 p × 100 Gbps
Total buffer	4MB	9MB	12MB	16MB (4 MMUs)	22MB
Buffer per port	85KB	192KB	384KB	512KB	344KB
Buffer per port per Gbps	85KB	19.2KB	9.6KB	5.12KB	3.44KB

show that even the 50th percentile inter-pod latency can exceed 200μs. Such latency eventually transfers to a large buffer demand. Consider a 100Gbps network with 80μs base RTT, the per-port buffer requirement of ECN* can easily reach 1MB ($100Gbps \times 80\mu s$).

D. Buffer Becomes Increasingly Insufficient

However, the buffer size of commodity switching chips in datacenters does not increase as expected. We list buffer and capacity information of some widely used commodity switching chips in Table I. The capacity significantly outpaces the buffer size, resulting in decreasing buffer per port per Gbps (from 85KB to 5.12KB). The reasons of shallow switch buffers are at least two-fold.

- The memory used in switch buffers is high-speed SRAM (Static Random Access Memory). Compared to DRAM (Dynamic Random Access Memory), SRAM is more expensive as it requires more transistors.
- The area increases with the memory size. When the area becomes large, the read/write latency will increase, making the memory access speed hard to match the link speed.

Therefore, most commodity switches in DCNs are shallow buffered. We envision that such trend will hold for future 200/400Gbps switching chips.

III. PROBLEMS CAUSED BY EXTREMELY SHALLOW BUFFER

In this section, we demonstrate the performance impairments of existing TCP/ECN solutions in extremely shallow-buffered high-speed DCNs. Existing solutions use switch buffers either 1) too aggressively, thus causing excessive packet losses at high loads (§III-A) or 2) too conservatively, thus seriously degrading throughput at low loads (§III-B).

A. Standard ECN Configuration Causes Excessive Packet Losses

Most datacenter TCP variants [11], [19], [51], [55], [61] leverage ECN to achieve high throughput with certain buffering. Network operators need to configure a moderate ECN marking threshold (*e.g.*, $C \times RTT \times \lambda$) for them to achieve 100% throughput. To the best of our knowledge, this is current operation practice in many production datacenters. However, the standard ECN configuration is likely to overfill extremely shallow buffers when many ports are active simultaneously, thus causing excessive packet losses and degrading tail performance of small flows.

We take Broadcom Tomahawk with 16MB buffer and 32 100Gbps ports as an example. If TCP desires 1MB ($100Gbps \times 80\mu s$) marking threshold per port, the switch buffer will be overfilled when more than half of total ports ($\geq$16) are congested. What is worse, Tomahawk has 4 switch cores to achieve desired performance at the high-speed. Each core has its own MMU and 4MB shared buffer [1], [2]. Dynamic buffer sharing only happens within the single core. Therefore, the buffer of a Broadcom Tomahawk chip will be overfilled when more than 4 ports attached to a single core are congested simultaneously.

B. Conservative ECN Configuration Degrades Throughput

Realizing the above limitation, a straight forward solution is to configure a lower ECN marking threshold (*e.g.*, smaller than average per-port switch buffer), thus leaving buffer headroom to reduce packet losses. However, this conservative ECN configuration causes unnecessary bandwidth wastage when few ports are busy. For example, when only one port is active, this method still throttles TCP throughput despite the sufficient buffer space available.

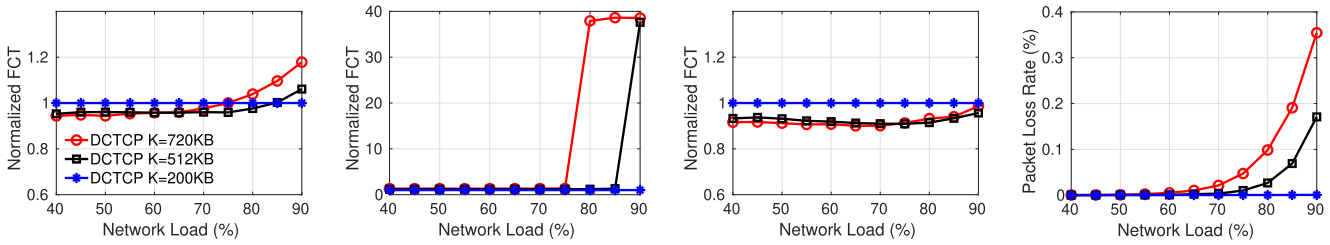
C. Simulation Validation

Since we do not have enough servers to saturate the switch buffer in the testbed, we use ns-2 simulations to quantify the impact of above performance impairments. We consider two schemes: standard ECN configuration (§III-A) and conservative ECN configuration (§III-B). Our simulation settings are shown as follows.

Topology: There are 32 servers connected to a switch using 100Gbps links. The base latency is 80μs. Hence, the BDP is $80\mu s \times 100Gbps = 1MB$. The jumbo frame (9KB MTU) is enabled.

Buffer: To emulate Broadcom Tomahawk chip, we attach every 8 switch ports to a 3MB shared buffer pool. Each switch port also has 128KB static reserved buffer. All switch ports have the same $\alpha = 4$ for dynamic buffer sharing. At the end host, we allocate 10MB static buffer to each NIC.

Schemes compared: We enable DCTCP at the end host and set RTOmin to 5ms. Based on our testbed measurement in §II-C, 720KB marking threshold (0.72BDP) should be enough for DCTCP to achieve full link utilization. Hence, we consider two marking thresholds: 720KB (standard) and 200KB (conservative).



(a) Overall: Avg FCT (b) ≤100KB: 99th percentile FCT (c) ≥10MB: Avg FCT (d) Packet Loss Rate

Fig. 3. [Simulation] FCT and packet loss rate results for the web search workload with the single switch topology. Note that we normalize FCT results to values achieved by DCTCP K=200KB.

Workloads: Among 32 servers, 24 server send traffic to the rest 8 servers. Note that the 8 ports connected to receivers are attached to the same shared buffer pool. We generate flows according to the web search distribution (Figure 8). We vary the network load (average utilization of links to receivers) from 40% to 90%.

Performance metrics: We use FCT as the primary metric and also measure packet loss rate for analysis. For clear comparison, we normalize all FCT results to values achieve by 200KB threshold.

Results analysis: Figure 3 shows FCT and packet loss rates. We make the following two observations.

At low loads (few ports active), the 720KB threshold (standard) obviously outperforms the 200KB threshold (conservative). For example, at 40% load, compared to the 200KB threshold, the 720KB threshold achieves $\sim$ 6% lower overall average FCT and $\sim$ 8.7% lower average FCT for large flows. Moreover, both solutions achieve near zero packet loss rate when the load is smaller than 60%. In such scenarios, the switch has sufficient buffer. But the conservative ECN configuration still leaves much buffer space unused, thus seriously degrading throughput. This confirms our analysis in §III-B.

At high loads (more ports active), the 720KB threshold causes excessive packet losses, thus seriously degrading the 99th percentile FCT for small flows. For example, as shown in Figure 3(d), at 90% load, 0.34% packets are dropped with the 720KB threshold, which is much higher than the 0.1% network service level agreement (SLA) threshold used in Microsoft datacenters [30]. Such high packet loss rate causes numerous packet retransmissions and 10,390 TCP timeouts (100,000 flows). As a result, at 90% load, the 99th percentile FCT for small flows with the 720KB threshold is $\sim$37.6 (5399μs to 140μs) higher than that with the 200KB threshold. This indicates that the standard ECN configuration causes excessive packet losses when many ports are congested simultaneously, confirming our analysis in §III-A.

IV. Solution

A. Design Goals

Good performance: We seek to achieve both high throughput and low packet loss rate simultaneously. However, as shown in §III, the scarce buffer resource in switching chips may make it difficult to guarantee both metrics in all scenarios. Like prior work [31], [40], when a conflict arises between the two metrics, we prefer low packet loss rate at the cost of sacrificing little throughput. This is because the bandwidth is generally plentiful in datacenters, while a small increase in packet loss rate (*e.g.*, $\geq 0.1\%$) can seriously degrade the performance of user-facing applications and in turn, operator revenue [40].

Deployability: Our solution should work with existing commodity switches and be backward compatible with legacy TCP/IP network stacks. Modifying switch hardware is costly as a new switch ASIC often takes years to design and implement. We note that some prior DCN transport designs [13], [23], [28], [31], [45], [46], [60] may work with extremely shallow buffers. However, as we will discuss in §VI, these solutions require non-trivial modifications to switch hardware and network stacks or make unrealistic assumptions. They fail to achieve our goal.

B. BCC *Mechanism*

Overview: BCC keeps conventional TCP/ECN algorithm at the end host, and its core is to perform global *buffer-aware* ECN marking at the switch. When the shared buffer utilization is low, BCC marks packets with standard ECN configuration to achieve both high throughput and low packet loss rate. When the utilization goes high, BCC marks packets more aggressively to prevent packet loss with minimal throughput loss. BCC realizes this with just one more ECN configuration in existing commodity switches.

Details: We now describe the mechanism of BCC in detail. We model the switch as a shared-buffer output-queued switch. Variables and parameters used in the model are listed in Table II and III. We start from the simplest assumption that each switch port only contains a single egress queue[7] and no buffer is reserved for each queue. Hence, all buffers are dynamically allocated from a single shared buffer pool. The switch has B shared buffer space and N egress queues in total. Any TCP/ECN variant (e.g., [11], [42], [51], [55]) can be enabled at the end host. The standard ECN setting is configured on each port/queue for high throughput.

We assume that our TCP variant (with standard ECN configuration) requires at least B_R buffer space per queue to achieve both high throughput and low packet loss rate. We simply treat B_R as a known constant here and show

[7]In §$IV-B$ and §$IV-C$, we use queue and port interchangeably. We further discuss the impact of multiple queues per port in §$IV-D$.

TABLE II
SHARED BUFFER MODEL PARAMETERS

Parameter	Description
B	Switch shared buffer size
N	Total number of switch egress queues
C	Capacity of the switch queue
RTT	Base round-trip time
α	Parameter for shared buffer allocation
B_R	Minimum per-queue required buffer for high throughput and low packet loss rate
K_{min}	Minimum marking threshold for shared buffer ECN/RED
K_{max}	Maximum marking threshold for shared buffer ECN/RED
P_{max}	Maximum marking probability for shared buffer ECN/RED
h	See Equation 3

TABLE III
SHARED BUFFER MODEL VARIABLES

Variable	Description
t	Time
$Q_i(t)$	Length of switch queue i at time t
$T(t)$	Queue length control threshold at time t

how to determine B_R later in $\S IV-C$. When $T(t) > B_R$, it means that the switch has sufficient buffer space to achieve both goals simultaneously. Hence, BCC just marks packets with the standard ECN configuration without degrading throughput.

When $T(t) \leq B_R$, it indicates that the shared buffer pool is highly utilized by many active ports. In such scenarios, only relying on standard ECN configuration may cause excessive packet losses as analyzed in $\S III-A$. Hence, BCC throttles the shared buffer occupancy to avoid excessive packet losses. By Equation 1 and $T(t) \leq B_R$, we derive that

$$\sum_{i=1}^{N} Q_i(t) \geq B - B_R/\alpha \tag{2}$$

Here $\sum_{i=1}^{N} Q_i(t)$ is the occupancy of the shared buffer pool at time t, and B, B_R and α are all known parameters. This implies that, to prevent excessive packet losses, BCC should throttle the shared buffer occupancy from exceeding a static threshold $B - B_R/\alpha$.

To realize this, we leverage the shared buffer ECN/RED functionality which has been supported in commodity switching chips [3], [9], [19]. Shared buffer ECN/RED follows the original RED algorithm [26] but tracks the occupancy of a shared buffer pool to mark packets. Note that all transmitted packets in the shared buffer pool can get marked regardless of their ingress/egress ports and queues. Therefore, shared buffer ECN/RED can effectively control shared buffer occupancies. Moreover, shared buffer ECN/RED can be used in combination with other ECN/AQM configurations at the switch. When multiple ECN configurations enabled, a packet gets marked if anyone decides to mark it first. Hence, BCC works as follows:

- When few ports are active, the available buffer is abundant and per-port standard ECN configuration will take effect first to strike the balance of high throughput and low latency as prior work [11]. Both high throughput and low packet loss rate can be achieved.
- When more and more ports become active, the buffer space becomes insufficient. Shared buffer ECN/RED will be automatically triggered first to prevent packet loss at the cost of degrading throughput slightly.

Furthermore, BCC performs a RED-like probabilistic marking over shared buffer ECN by setting minimum and maximum thresholds (K_{min} and K_{max} in Table II) to different values. This is because, if we use a single cut-off threshold like DCTCP [11], all flows across ports sharing the same buffer pool are likely to reduce their window at the same time. This causes global synchronization problem and a further loss of throughput [26]. We confirm such problem in our simulations ($\S$V-C.1). With probabilistic marking, we effectively desynchronize flows' window reduction activities and improve throughput.

C. BCC *Parameters*

We now derive BCC parameters. First, we determine B_R, the minimum per-queue (port) buffer size for both high throughput and low packet loss rate. With B_R fixed, we then decide marking thresholds and probability of shared buffer ECN/RED. Note that in this section we give several useful rules-of-thumb to set parameters while leaving optimal parameter settings for future work.

Determine B_R: Statistics has shown that there is typically a small number of concurrent large flows to the same receiver in production datacenters [11]. Hence, we start from a simple scenario where several synchronized long-lived flows share a bottleneck link. In such scenario, we need $C \times RTT \times \lambda$ per port buffering to achieve 100% throughput, where λ is a characteristic constant of the congestion control algorithm.

However, the lag in ECN control loop imposes extra buffer requirement to avoid packet losses. When a packet gets ECN marked at switch egress,[8] the sender will reduce its window after one RTT. During this RTT interval, extra buffer space is required to absorb the queue increase. We assume that the receiver acknowledges every MTU-sized data packet. We consider the most challenging TCP slow start phase. As an ACK packet can trigger two MTU-sized data packets, the aggregate sending rate reaches $2C$ and the switch queue gradient is C. Therefore we need $C \times RTT$ extra buffer to avoid packet losses and $C \times RTT \times (1 + \lambda)$ buffer in total to achieve both goals.

We next consider a realistic scenario that a mix of small and large flows arrive and leave dynamically. In this scenario, it is less likely that all active flows enter slow start phase simultaneously, thus reducing the switch queue gradient. But the arrivals and departures of flows also affect the switch

[8]Modern shared buffer switches mark packets using RED at egress side [63].

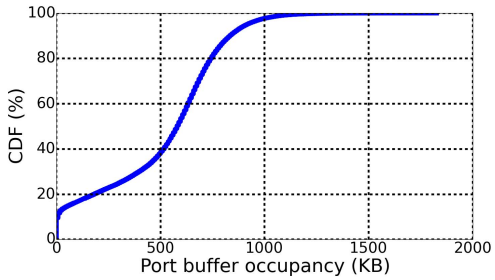


Fig. 4. [Simulation] CDF of buffer occupancies of the congested port at 90% load.

queue gradient, which is hard to model. Hence, we run a ns-2 simulation instead. In this simulation, 31 senders generate traffic to the same receiver according to the web search workload. The average link utilization is 90%. We increase the shared buffer size to 10MB, which eliminates packet loss in the network. The other settings are same as those in $\S III-C$. We configure the per-port ECN/RED marking threshold to 720KB ($C \times RTT \times \lambda$ where C is 100Gbps, RTT is 80μs and λ is 0.72). Hence, $C \times RTT \times (1+\lambda)$ is equal to 1720KB. Figure 4 plots the CDF of buffer occupancies of the congested port. Around 25% occupancies are larger than 720KB, suggesting that $C \times RTT \times \lambda$ is not enough. The 99.99th percentile buffer occupancy is 1609KB, which is smaller than $C \times RTT \times (1+\lambda)$. Hence, we envision that $C \times RTT \times (1+\lambda)$ also works well for mixes of small and large flows.

In summary, we recommend setting B_R to $C \times RTT \times (1+\lambda)$. As C and λ are both known and RTT can be measured [30], [55] in production datacenters, operators can easily compute the value of B_R.

Determine parameters for shared buffer ECN/RED: We leverage shared buffer ECN/RED to prevent the shared buffer occupancy from exceeding $B-B_R/\alpha$. To achieve fast reaction to bursty traffic, we mark packets based on the instantaneous buffer occupancy. Shared buffer ECN/RED has 3 parameters to configure: minimum threshold K_{min}, maximum threshold K_{max} and maximum probability P_{max}. When the buffer occupancy is: 1) below K_{min}, no packet is marked; 2) between K_{min} and K_{max}, packets are marked according to a probability; 3) exceeds K_{max}, all packets get marked.

As analyzed before, if we set $K_{min} = K_{max}$, flows across ports sharing a buffer pool are likely to get synchronized, resulting throughput loss. Therefore, we decide to perform a probabilistic marking by setting $K_{min} < K_{max} = B - B_R/\alpha$. The key here is to control the range between K_{min} and K_{max}. A too small $K_{max} - K_{min}$ will make buffer occupancy regularly ramp up beyond K_{max}, still causing global synchronization and even packet losses. As original RED work [26] suggests, $K_{max} - K_{min}$ should be made sufficiently large (*e.g.*, larger than typical increase in the shared buffer occupancy during a RTT) to avoid global synchronization. Hence, the choice of $K_{max} - K_{min}$ depends on both the number of ports N and link capacity C. In BCC, given $K_{max} = B - B_R/\alpha$, we set K_{min} as follows:

$$K_{min} = B - B_R/\alpha - C \times N \times h \quad (3)$$

where h is a parameter to control $K_{max} - K_{min}$, the range performing probabilistic marking. In our evaluation, we set h to 8μs. For the maximum marking probability P_{max}, we set it to 10% following the guideline in [26]. In $\S V-C.2$, we find that BCC is robust for a range of h and P_{max}.

D. Discussion

In this section, we first discuss several factors that may affect the design of BCC. Then we discuss BCC's compatibility with delay-based transports.

Impact of multiple MMUs: Some switching chips (*e.g.*, Broadcom Tomahawk) have multiple MMUs and dynamic buffer allocation only happens within the single MMU. BCC also takes effect in such architecture as the shared buffer ECN/RED operates in a per-MMU manner. Each MMU has its own shared buffer ECN/RED and only marks its own packets based on its own shared buffer occupancy (see validation in $\S V-A$).

Impact of different α values: Each switch egress port has multiple queues for traffic isolation and scheduling. To provide differentiated network services, operators may configure different α values for different queues. For example, operators may give higher α values to queues carrying traffic of more important services, *e.g.*, real-time workloads.

When there are different α values, we can choose the minimum value α_{min} among them and update shared buffer ECN/RED parameters as follows: $K_{max} = B - B_R/\alpha_{min}$, $K_{min} = B - B_R/\alpha_{min} - C \times N \times h$.

Impact of static reserved buffers: In $\S IV-B$, we simply assume that all buffers are dynamically allocated. In practice, each egress queue has a small amount of static reserved buffer by default. Operators can also configure different reserved buffer sizes for different queues based on their importances.

When both static reserved buffers and dynamic shared buffers exist, the MMU first tries to use static reserved buffers. Therefore, we should reduce B_R, the minimum per-queue shared buffer size for both high throughput and low packet loss rate, to incorporate the static reserved buffer into BCC. Let S_{min} denote the minimum static buffer size reserved for a single queue. Our recommended value for B_R should become $C \times RTT \times (1+\lambda) - S_{min}$

V. Evaluation

In this section, we evaluate BCC using both small-scale testbed experiments and large-scale ns-2 [8] simulations. We highlight the results as follows:

- Our testbed validations ($\S V-A$) show that BCC is easy to configure at switches and performs as expected.
- Our ns-2 simulations ($\S V-B$) demonstrate BCC's superior performance in large-scale networks. At low loads, BCC fully utilizes the link capacity and achieves up to 19.3% lower average FCT for large flows compared to a conservative ECN configuration. At high loads, compared to a standard ECN configuration, BCC achieves up to 94.4% lower 99th percentile FCT for small flows while only degrading large flow FCT by up to 3%.

- Our targeted simulations ($\S V-C$) demonstrate that BCC can mitigate the global synchronization problem with probabilistic marking and is robust to various parameter settings.

A. Testbed Validations

Due to the scale of our testbed, we only use testbed micro-benchmarks to (1) show that BCC is readily-deployable at commodity switches and (2) validate its functionalities in two scenarios: single MMU and multiple MMUs. Our goal here is to show the hardware feasibility of BCC, however, a non-trivial BCC deployment at scale is beyond the scope of this paper and left as future work.

Testbed setup: Our testbed consists of 6 servers connected to a Arista 7060CX-32S-F 100Gbps switch. The 6 servers and 6 switch ports are denoted as S1-S6 and P1-P6, respectively. Pi is the port connected to Si. The base round-trip time is $\sim$30μs. We use S5 and S6 as receivers. S1 and S2 send traffic to S5. S3 and S4 send traffic to S6. Hence, the ports P5 and P6 are congested.

Our Arista 7060CX-32S-F switch is built on the top of Broadcom Tomhawk [6] chip. It has 4 MMUs, each with 4MB packet buffer. The dynamic buffer allocation only happens within the single MMU. In each MMU, $\sim$1MB buffer space has been reserved and only 3MB buffer space can be dynamically allocated.

Each server is a Dell PowerEdge R730 with 2 16-core Intel Xeon E5-2698 2.3GHz CPUs, 256GB RAM and 1 Mellanox ConnectX-4 NIC. All servers run Linux 3.10.0 kernel. Various system optimizations, *e.g.*, TSO and GRO, are enabled. We use ECN* [55] (regular ECN-enabled TCP) in testbed experiments as it is more sensitive to the ECN marking threshold (Figure 4 in [55] and Figure 2). To fully utilize the link capacity, we configure the per-queue ECN/RED marking threshold to 325KB according to Figure 2.

Validating BCC in a single MMU: In this experiment, two congested ports P5 and P6 are attached to the same MMU. S1 and S2 start 16 long-lived TCP flows using `iperf` to S5. S3 and S4 generate 100Gbps UDP traffic to S6 using a high performance packet generator. We configure α to 1 for dynamic buffer allocation.

Figure 5 gives queue length sample variations of two congested ports. Due to the impact of ECN marks, queue length of P5 remains very low. By contrast, UDP traffic to S6 does not react to ECN marks (and drops) at all, thus building up large queues in P6.[9] In theory, P6 can get almost half of the total shared buffer space according to dynamic threshold algorithm [24], which is $\sim$1.5MB. Our testbed results closely match the theory results.

Now, we enable shared buffer ECN/RED using a single command shown in Figure 6. For simplicity, both minimum and maximum thresholds of shared buffer ECN/RED are set to the same value in our testbed experiments.

Figure 7 shows aggregate TCP goodput with different marking thresholds. As analyzed above, UDP traffic to S6 does not react to ECN marks and drops at all. Thus, switch queues in P6 keep oscillating around 1.5MB, regardless of the shared buffer ECN/RED settings. When the shared buffer ECN/RED marking threshold is set to 500KB, all TCP packets to S5 get ECN marked, resulting in $\sim$8Gbps TCP goodput. After we increase the threshold above 1.5MB, the TCP goodput keeps increasing and reaches 90Gbps with 1750KB threshold.

The above experiment (1) shows that BCC is extremely easy to deploy at commodity switches and (2) validates BCC's functionality in a single MMU.

Validating BCC in multiple MMUs: In this experiment, two congested ports P5 and P6 are attached to different MMUs. S1 and S2 start 16 long-lived TCP flows to S5. S3 and S4 start 16 long-lived TCP flows to S6.

As shown in Table IV, without shared buffer ECN/RED, both S5 and S6 can receive $\sim$90Gbps goodput as expected. After we set shared buffer ECN threshold to 350KB, the goodput results remain unchanged. The is because shared buffer ECN/RED *operates in a per-MMU manner:* each MMU has its own shared buffer ECN/RED and only marks its own packets based on its own shared buffer occupancy. In a MMU, when

[9]Our switch performs ECN/RED marking (or dropping non-ECT packets) at the egress side rather than ingress side. Therefore, it cannot prevent queue build-ups caused by UDP traffic.

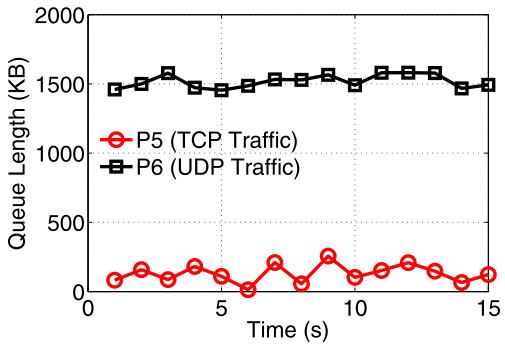


Fig. 5. [Testbed] Queue length samples of port 5 and 6. Note that shared buffer ECN/RED is disabled. Traffic to P5 is TCP while traffic to P6 is UDP.

```
switch(config)# qos random-detect ecn global-buffer
minimum-threshold 500 kbytes maximum-threshold
500 kbytes
```

Fig. 6. Command to enable shared buffer ECN/RED on Arista EOS [9]. Both minimum and maximum thresholds are set to 500KB.

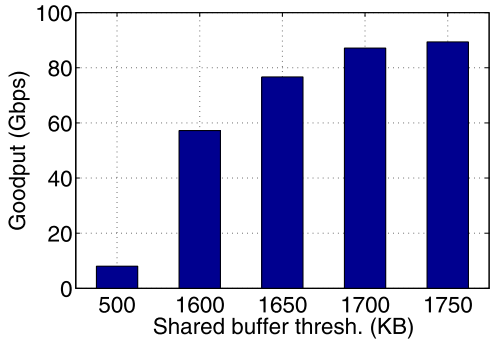


Fig. 7. [Testbed] Aggregate goodput of TCP traffic to S5 with different shared buffer ECN thresholds.

TABLE IV

[TESTBED] TCP GOODPUT RESULTS WITH DIFFERENT SHARED BUFFER ECN THRESHOLDS. NOTE THAT PACKETS TO S5 AND S6 ARE STORED IN DIFFERENT MMUS

Shared buffer thresh. (KB)	N/A	350	150
Goodput of S5 (Gbps)	91.02	90.94	78.43
Goodput of S6 (Gbps)	89.42	89.39	77.89

there is only a single congest port and the per-port (queue) threshold is smaller than the shared buffer threshold, per-port (queue) ECN marking will be triggered earlier than shared buffer ECN. As a result, shared buffer ECN/RED does not take any effect. But if we reduce the shared buffer threshold to 150KB, shared buffer ECN/RED starts to take effect, reducing TCP goodput to ∼78Gbps.

This experiment validates BCC's functionality in multiple MMUs.

B. Large Scale Simulations

In this section, we use ns-2 [8] to evaluate BCC's performance in large-scale DCNs with realistic workloads.

Topology: We simulate a 128-host leaf-spine topology with 8 leaf (ToR) switches and 8 spine (Core) switches. Each leaf switch has 16 100Gbps down links to hosts and 8 100Gbps up links to spines. Hence, we have a 2:1 oversubscription, which is common in production datacenters. We employ Equal-Cost Multi-Path routing (ECMP) for load balancing. The base fabric RTT across the spine is ∼80μs of which 72μs is spent at the end host. The BDP is 1MB. The jumbo frame is enabled.

Buffer: Our leaf and spine switches have 24 and 8 ports, respectively. To emulate Broadcom Tomahawk chip, we attaches every 8 switch ports to a 3MB shared buffer pool. Hence, the leaf switch has 3 shared buffer pools while the spine switch only has one. At the leaf switch, 8 up ports, which are connected to spines, are attached to a shared buffer pool while the rest 16 down ports are attached to the other 2 shared buffer pools. We set α to 4 for all switch ports. In addition, each switch port has 128KB static reserved buffer. At the end host, we allocate 10MB static buffer for each NIC.

Schemes compared: We use DCTCP [11] as the transport protocol. We set RTOmin to 5ms and initial window to 20 packets (180KB). Note that 5ms is the minimum effective RTOmin for many Linux kernel versions [38]. We exclude PFC due to its large buffer reservation requirement. We compare the following four schemes:

- **DCTCP K=720KB**: This is a standard ECN configuration (current practice). We configure the per-port (queue) ECN/RED marking threshold to 720KB (0.72BDP based on measurement in §$II-C$) to achieve 100% throughput.
- **DCTCP K=512KB**: We configure the per-port (queue) ECN/RED marking threshold to 512KB, which equals to the average per-port switch buffer size.
- **DCTCP K=200KB**: This is a very conservative ECN configuration. We configure the per-port (queue) ECN/RED marking threshold to 200KB, which is much smaller than average per-port switch buffer size (512KB), to reduce packet losses.

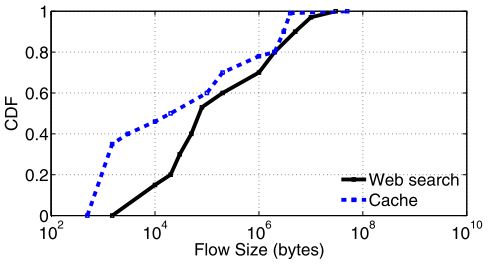


Fig. 8. Empirical traffic distributions.

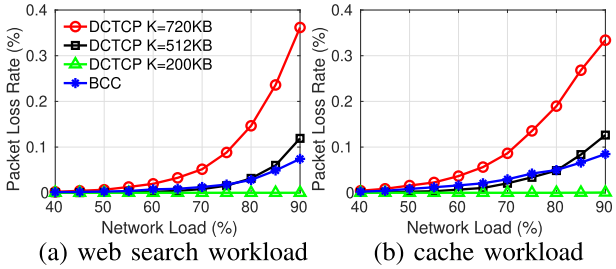


Fig. 9. [Simulation] Packet loss rate results.

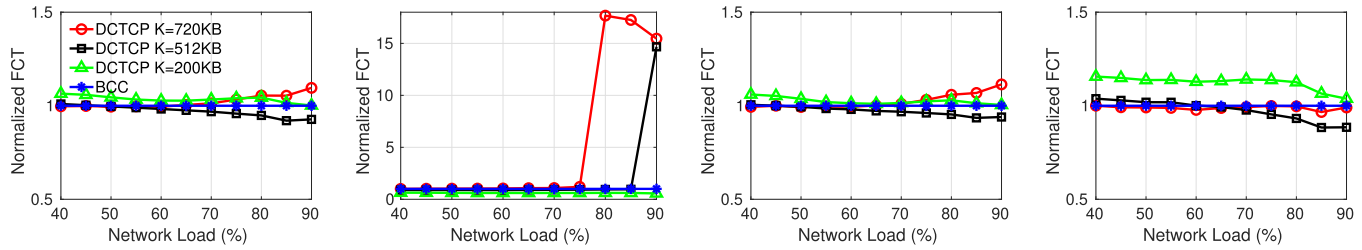
- **BCC** : BCC requires two ECN configurations at the switch. We set per-port (queue) ECN/RED marking threshold to 720KB like the standard ECN configuration. Since λ is 0.72 for DCTCP and the per-port static reserved buffer size S_{min} is 128KB, $B_R = C \times RTT \times (1+\lambda) - S_{min} \approx$ 1.6MB. Therefore, $K_{max} = B - B_R/\alpha \approx$ 2.6MB. Since each buffer pool is shared by $N = 8$ ports and the recommendation value for h is 8μs, $K_{min} = K_{max} - C \times N \times h \approx$ 1.8MB. P_{max} is set to 10%. We further evaluate BCC's sensitivity to h and P_{max} in §$V-C$.2.

Workloads: We conduct our simulations using realistic workloads based on empirically observed traffic patterns in production datacenters. We consider the two flow distributions in Figure 8: web search [11] and cache [48]. Both distributions are heavy-tailed. We generate flows according to a Poisson process and choose the source and destination for each flow uniformly at random. We vary the flow arrival rate to achieve a desired levels of load in the fabric. Each simulation lasts for 100,000 flows.

Performance metrics: We use FCT as the primary performance metric and also measure packet loss rate in certain simulations for analysis. In addition to overall results, we also breakdown FCT results across small (0,100KB], medium (100KB,10MB] and large (10MB,∞) flows. Since the request completion time of many large-scale responsive applications depends on the slowest flow, we consider the 99th percentile FCT for small flows. For the rest flows, we consider their average FCT. For clear comparison, we normalize all FCT results to values achieve by BCC by default.

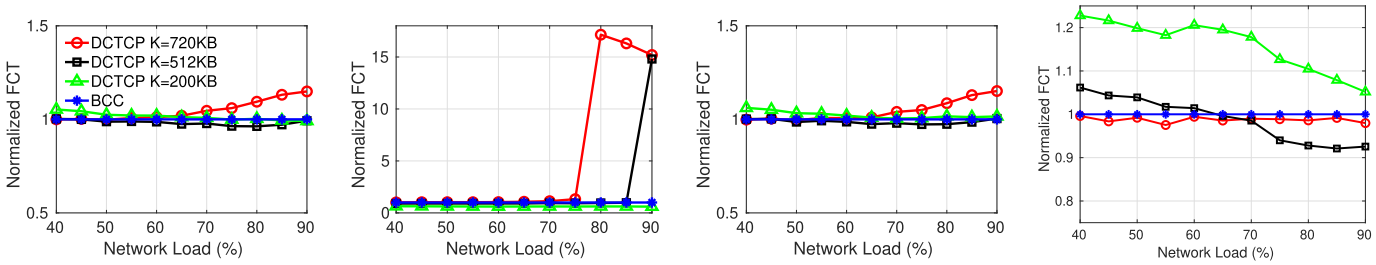
Results analysis: Figure 10 and 11 give FCT results for two workloads. We also plot packet loss results in Figure 9.

At low loads, BCC performs similarly as K=720KB while generally outperforming K=200KB and K=512KB.



(a) Overall: Avg (b) (0,100KB]: 99th percentile (c) (100KB,10MB]: Avg (d) (10MB,∞): Avg

Fig. 10. [Simulation] FCT results for the web search workload. Results are normalized to values of BCC.



(a) Overall: Avg (b) (0,100KB]: 99th percentile (c) (100KB,10MB]: Avg (d) (10MB,∞): Avg

Fig. 11. [Simulation] FCT results for the cache workload. Results are normalized to values of BCC.

K=200KB only shows some performance advantage (∼100μs) on small flows, due to its lower switch queueing.

As shown in Figure 9, K=720KB achieves very low packet loss rate when the load is smaller than 55%. It indicates that the switch has enough buffer space to achieve both high throughput and low packet loss rate at low loads. However, K=200KB still marks packets conservatively regardless of the sufficient buffer resource, thus causing much unnecessary bandwidth wastage. For K=512KB, while it is better than K=200KB in terms of bandwidth utilization, such an ECN marking threshold setting is still a bit conservative compared to K=720KB, thus leading to certain performance degradation. In such scenarios, due to the low shared buffer occupancy, BCC marks packets like standard per-port ECN configuration without triggering shared buffer ECN. Therefore, BCC can fully utilize the link capacity. Compared to K=200KB, BCC achieves up to ∼13.5% (6362μs to 5503μs) and ∼19.3% (9205μs to 7424μs) lower average FCT for large flows, in the web search and cache workloads, respectively.

At high loads, BCC performs similarly as K=200KB while generally outperforming K=720KB and K=512KB in terms of small flows. However, for large flows, BCC's performance is not as good as K=720KB and K=512KB, but still much better than K=200KB. This is a design tradeoff made by BCC

For small flows, BCC achieves up to 94.4% (5174μs to 291μs) and 94.2% (5135μs to 296μs) lower 99th FCT compared to K=720KB, in the web search and cache workloads, respectively. This is because K=720KB causes excessive packet losses due to the exorbitant shared buffer utilization. As shown in Figure 9, the packet loss rate with K=720KB approaches 0.1% (SLA threshold used in Microsoft [30]) at 75% load and exceeds 0.3% at 90% load. This results in frequent TCP timeouts, which seriously increases FCT by at least 5ms. For example, at 90% load, K=720KB leads to 11,434 timeouts for web search workload and 5,935 timeouts for cache workload (100,000 flows in total). By contrast, BCC can greatly reduce packet losses even though it cannot achieve near lossless performance as K=200KB. At 90% load, the packet loss rate with BCC is lower than 0.08% for both workloads. Hence, BCC only causes 2,432 timeouts for web search workload and 1,278 timeouts for cache workload. Furthermore, we find that the packet loss rate of K=512KB is slightly worse than BCC at high load (e.g., 90%), but less severe than that of K=720KB. As a result, while the performance of K=512KB for small flows at high load is worse compared to BCC due to packet loss, it is better than K=720KB.

For large flows, BCC's performance is not as good as K=720KB and K=512KB. For example, BCC's performance is within ∼0.4-2.8% of the K=720KB for the web search workload and within ∼0.3-3.0% for the cache workload. This is the performance tradeoff made intentionally by BCC to maintain good performance on small flows as shown above. In the meanwhile, since the packet loss rate of K=512KB is less severe than that of K=720KB at high load, it achieves better throughput for large flows than K=720KB (and BCC as well). On the other hand, since K=200KB is still very conservative, its performance is constrained, and therefore it increases FCT by over ∼9% compared to K=720KB for both workloads.

In summary, BCC can operate effectively based on the real-time shared buffer utilization, thus keeping good performance in various scenarios.

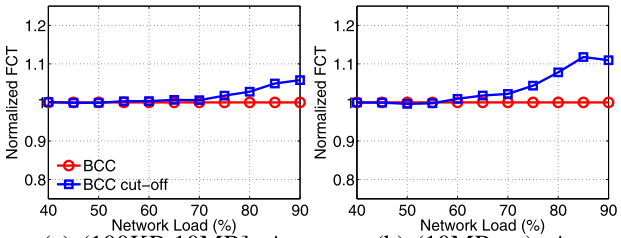


Fig. 12. [Simulation] FCT results for the web search workload. Note that results are normalized to values achieved by BCC.

C. BCC *Deep Dive*

In this section, we dig deeper into BCC's design using a series of targeted simulations. Due to space limitation, we only show results with the web search workload, and similar trends are observed on the cache workload as well.

1) Impact of Probabilistic Marking: In $\S IV-C$, we recommend probabilistic ECN marking with $K_{min} < K_{max}$. Another choice is cut-off ECN marking with $K_{min} = K_{max}$. Though cut-off ECN marking simplifies the switch configuration, it causes global synchronization problem, resulting in throughput loss. Here, we compare BCC with BCC using cut-off ECN marking (BCC cut-off) to explore the impact of probabilistic marking.

For original BCC, we recommend setting K_{min} and K_{max} to ∼1.8MB and ∼2.6MB, respectively. To achieve a fair comparison, we need to determine a threshold in [1.8MB, 2.6MB] for BCC cut-off that can achieve comparable packet loss rates as BCC. Through simulations, we find that 2.2MB is a good choice. For example, at 90% load, the packet loss rates with BCC and BCC cut-off (2.2MB threshold) are 0.0687% and 0.0699%, respectively. Therefore, we set the marking threshold of BCC cut-off to 2.2MB.

Figure 12 plots the FCT results of BCC and BCC cut-off with the web search workload. Note that results are normalized to values achieved by BCC for clear comparison. Since BCC and BCC cut-off almost achieve the same 99th percentile FCTs (within 3μs) for small flows at all loads, we omit the figure for small flows due to space limitation. As shown in Figure 12, with the increase of the network load, the performance gap between two schemes becomes larger and larger. Compared to BCC cut-off, BCC achieves up to ∼5.5% and ∼10.5% lower average FCT for medium and large flows, respectively. This shows that BCC cut-off causes global synchronization problem among different switch ports, resulting in loss of throughput. By contrast, BCC leverages probabilistic marking to mitigate this problem, thus delivering better throughput.

2) Impact of Parameters: BCC has two tunable parameters: h and P_{max}. As shown in Equation 3, h determines the buffer occupancy range for probabilistic marking. P_{max} determines the marking probability in this range. The larger h and P_{max} we have, the more likely a packet gets ECN marked.

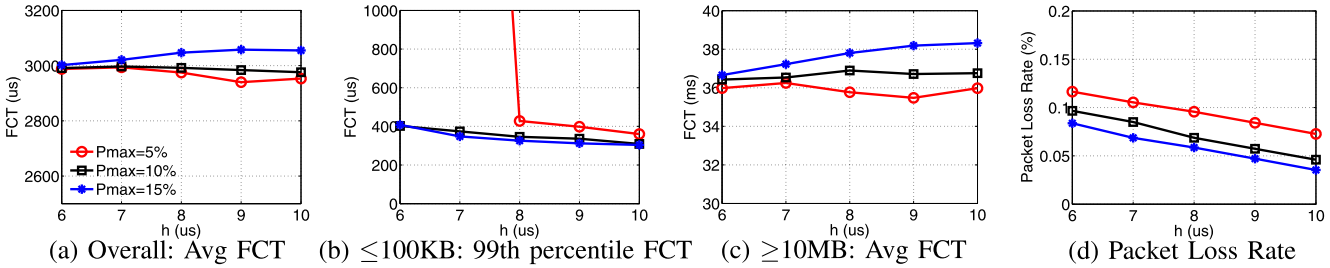
To explore BCC's sensitivity to h and P_{max}, we repeat the web search workload at 90% load for different h and P_{max} settings. h is varied from 6μs to 10μs. We consider 3 P_{max} values: 5%, 10% and 15%. Figure 13 gives FCT and packet loss rate results. In the interest of space, we omit FCT for medium flows. As expected, larger h and P_{max} can help reduce packet losses at the cost of slightly larger FCT for large flows. Generally speaking, BCC is robust to parameter settings. Overall average FCTs with different settings are within the range of [2940μs, 3058μs]. Average FCTs for large flows with different settings are within the range of [35.5ms, 38.3ms]. All settings also achieve good performance for small flows except for two settings: $h = 6\mu$s, $P_{max} = 5\%$ and $h = 7\mu$s, $P_{max} = 5\%$. As shown in Figure 13(d), packet loss rates with these two settings slightly exceed 0.1% (SLA threshold used in Microsoft Datacenters [30]). Though such results are much better than that achieved by DCTCP K=720KB (∼0.33%), more than 1% small flows still suffer from at least one TCP timeout, resulting in large 99th percentile FCTs. This suggests that we should not set too small h and P_{max} for BCC.

VI. Related Work

Bufferless DCN transport: Many DCN transport designs [13], [23], [28], [31], [33], [41], [45], [46], [60] can cope with shallow switch buffers. However, they are hard to deploy in production DCNs due to their non-trivial modifications to switch hardware or network stacks. For example, pHost [28], ExpressPass [23] and Homa [41] require clean-slate network stacks. HULL [13] and TFC [60] adopt TCP at the end host, but require non-trivial modifications to switch hardware. Fastpass [46] and Flowtune [45] require changes to network stacks and leverage a centralized scheduler, which suffers from failures and scalability issues. NDP [31] modifies both switch hardware and network stacks. Furthermore, some of them make unrealistic assumptions to the underlying network. For example, pHost [28] and NDP [31] assume that the congestion-free network core, which does not hold for many production DCNs. ExpressPass [23] requires path symmetry, which incurs increased configuration complexity with ECMP. In contrast to all these efforts, BCC is easy to deploy as it only requires one more ECN configuration at commodity switches.

PFC: PFC (Priority-based Flow Control) [7] has been enabled in many production datacenters to enable lossless networks for RoCE (RDMA over Converged Ethernet) deployments [29], [50]. It can also be used with TCP. With standard ECN configuration plus PFC, it *seems* that we can achieve high throughput when few ports are active and zero packet loss even when many ports are active.

However, PFC needs to reserve buffer for each priority class as headroom to prevent packet drops [29], [62]. The size of the headroom is determined by the MTU size, the PFC reaction time of the egress port, and most importantly, the propagation delay between the sender and the receiver. Deploying PFC in production DCNs requires large headroom size, which may not be affordable for shallow buffered switches. For example, Microsoft DCN operators can reserve headroom for at most 2 priority classes. This conflicts with the current practice where multiple queues are used for



(a) Overall: Avg FCT (b) ≤100KB: 99th percentile FCT (c) ≥10MB: Avg FCT (d) Packet Loss Rate

Fig. 13. [Simulation] FCT and packet loss rate results for the web search workload at 90% load using a variety of h and P_{max} settings.

QoS [14], [18], [19]. Furthermore, it has also been reported that PFC can introduce various performance and management problems such as head-of-line blocking [62], pause frame storm [29] and deadlock [34], [35], thus making networks difficult to understand and manage.

BCC can be used in conjunction with PFC. When few ports are active, both shared buffer ECN/RED and PFC are not triggered. When many ports are active, shared buffer ECN/RED will be triggered first to reduce shared buffer occupancy. If shared buffer ECN/RED still cannot effectively throttle queue build-ups (*e.g.*, when many small flows arrive simultaneously), PFC will be triggered as the last defense to eliminate packet losses.

Other work: PERC [37] targets at fast convergence in high-speed DCNs. DIBS [56] achieves a near lossless network by detouring packets. AuTO [22] optimizes DCN performance via deep reinforcement learning. Some efforts [44], [52], [57] have been made to reduce the impact of packet losses in DCNs. In addition, there are a large body of work on load balancing [10], [21], [59] and flow scheduling [14], [18], [32], [39], [53]. They are all complementary to our work.

VII. Conclusion

In production DCNs, the increase of link speed significantly outpaces the increase of switch buffer size, resulting in an extremely shallow-buffered environment. Consequently, prior TCP/ECN solutions suffer from severe performance degradation. To address this problem, we have introduced BCC, a simple yet effective solution with only one more shared buffer ECN/RED configuration at commodity switches. BCC operates upon real-time shared buffer utilization. It maintains low packet loss rate persistently while only slightly degrading throughput when the buffer becomes insufficient. We validated BCC's efficacy in a 100G testbed and demonstrated its superior performance using extensive simulations.

References

[1] *Information on Datacenter Switching Chip I*. [Online]. Available: https://people.ucsc.edu/~warner/Bufs/7060CX.html

[2] *Information on Datacenter Switching Chip II*. [Online]. Available: https://people.ucsc.edu/~warner/Bufs/tomahawk

[3] *Cisco Nexus 3000 Series Nx-Os Qos Configuration Guide, Release 7.x*. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus3000/sw/qos/7x/b_3k_QoS_Config_7x/b_3k_QoS_Config_7x_chapter_010.html

[4] *Dctcp in Linux Kernel 3.18*. [Online]. Available: http://kernelnewbies.org/Linux_3.18

[5] *Dctcp in Windows Server 2012*. [Online]. Available: http://technet.microsoft.com/en-us/library/hh997028.aspx

[6] *High-Density 25/100 Gigabit Ethernet Strataxgs Tomahawk Ethernet Switch Series*. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm56960

[7] *IEEE DCB. 802.1Qbb—Priority-Based Flow Control*. [Online]. Available: http://www.ieee802.org/1/pages/802.1bb.html.

[8] *The Network Simulator NS-2*. [Online]. Available: http://www.isi.edu/nsnam/ns/

[9] *User Manual of Arista EOS Version 4.15.0F*. [Online]. Available: https://www.arista.com/assets/data/docs/Manuals/EOS-4.15.0F-Manual.pdf

[10] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 503–514.

[11] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf. SIGCOMM (SIGCOMM)*, 2010, pp. 63–74.

[12] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: Stability, convergence, and fairness," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*, 2011, pp. 73–84.

[13] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. 9th USENIX Symp. Netw. Syst. Design Implement.*, 2012, pp. 253–266.

[14] M. Alizadeh *et al.*, "PFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf. SIGCOMM (SIGCOMM)*, 2013, pp. 435–446.

[15] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2004, pp. 281–292.

[16] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu, "Enabling ECN over generic packet scheduling," in *Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2016, pp. 191–204.

[17] W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, "PAC: Taming TCP incast congestion using proactive ACK control," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 385–396.

[18] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement.*, 2015, pp. 455–468.

[19] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement.*, 2016, pp. 537–549.

[20] 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," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 2007–2016.

[21] J. Cao *et al.*, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2013, pp. 49–60.

[22] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 191–205.

[23] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 239–252.

[24] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 130–140, Apr. 1998.

[25] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[26] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[27] R. Gandhi *et al.*, "Duet: Cloud scale load balancing with hardware and software," in *Proc. ACM Conf. SIGCOMM (SIGCOMM)*, 2014, pp. 27–38.

[28] 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. 11th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2015, pp. 1–12.

[29] C. Guo *et al.*, "RDMA over commodity Ethernet at scale," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 202–215.

[30] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 139–152.

[31] M. Handley *et al.*, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 29–42.

[32] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2012, pp. 127–138.

[33] S. Hu *et al.*, "Aeolus: A building block for proactive transport in datacenters," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 422–434.

[34] S. Hu *et al.*, "Deadlocks in datacenter networks: Why do they form, and how to avoid them," in *Proc. 15th ACM Workshop Hot Topics Netw. (HotNets)*, 2016, pp. 92–98

[35] S. Hu *et al.*, "Tagger: Practical PFC deadlock prevention in data center networks," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2017, pp. 451–463.

[36] C. Jiang, D. Li, and M. Xu, "LTTP: An LT-code based transport protocol for many-to-one communication in data centers," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 52–64, Jan. 2014.

[37] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, "High speed networks need proactive congestion control," in *Proc. 14th ACM Workshop Hot Topics Netw. (HotNets)*, 2015, pp. 1–7.

[38] G. Judd, "Attaining the promise and avoiding the pitfalls of TCP in the datacenter," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement.*, 2015, pp. 145–157.

[39] Z. Li *et al.*, "Rate-aware flow scheduling for commodity data center networks," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[40] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 537–550.

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

[42] A. Munir *et al.*, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2157–2165.

[43] P. Patel *et al.*, "Ananta: Cloud scale load balancing," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 207–218.

[44] P. Cheng *et al.*, "Catch the whole lot in an action: Rapid precise packet loss notification in data center," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement.*, 2014, pp. 17–28.

[45] J. Perry, H. Balakrishnan, and D. Shah, "Flowtune: Flowlet control for datacenter networks," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement.*, 2017, pp. 421–435.

[46] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized 'zero-queue' datacenter network," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 307–318.

[47] K. Ramakrishnan *et al.*, *The Addition of Explicit Congestion Notification (ECN) to IP*, document RFC 3168, 2001.

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

[49] A. Singh *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," in *Proc. SIGCOMM*, 2015, pp. 1–15.

[50] C. Tian *et al.*, "P-PFC: Reducing tail latency with predictive PFC in lossless data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1447–1459, Jun. 2020.

[51] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2012, pp. 115–126.

[52] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)*, 2009, pp. 303–314.

[53] S. Wang, D. Li, and J. Geng, "Geryon: Accelerating distributed CNN training by network-level flow scheduling," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 1678–1687.

[54] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," in *Proc. 6th Int. Conf. CoNEXT*, 2010, pp. 1–12.

[55] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 25–36.

[56] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "DIBS: Just-in-time congestion mitigation for data centers," in *Proc. 9th Eur. Conf. Comput. Syst. (EuroSys)*, 2014, pp. 1–14.

[57] D. Zats *et al.*, "FastLane: Making short flows shorter with agile drop notification," in *Proc. 6th ACM Symp. Cloud Comput. (SoCC)*, 2015, pp. 84–96.

[58] G. Zeng *et al.*, "Congestion control for cross-datacenter network," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–12.

[59] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 253–256.

[60] J. Zhang, F. Ren, R. Shu, and P. Cheng, "TFC: Token flow control in data center networks," in *Proc. 11th Eur. Conf. Comput. Syst. (EuroSys)*, 2016, pp. 1–14.

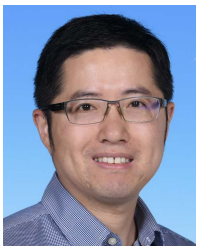
[61] J. Zhang, W. Bai, and K. Chen, "Enabling ECN for datacenter networks with RTT variations," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2019, pp. 233–245.

[62] Y. Zhu *et al.*, "Congestion control for large-scale RDMA deployments," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 523–536.

[63] 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.

Wei Bai received the B.E. degree in information security from Shanghai Jiao Tong University in 2013, and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2017. He is currently a Research Software Development Engineer at Microsoft Research Lab, Redmond. He is broadly interested in computer networking with a special focus on data center networking. His research work has been published in many top conferences and journals, such as SIGCOMM, NSDI, CoNEXT, and ToN. His current research mainly focuses on network infrastructure to support large-scale RDMA deployments.

Shuihai Hu received the B.S. degree in computer science from USTC in 2013, and the Ph.D. degree in computer science from HKUST in 2019. He is currently the Chief Scientist at Clustar. His current research interests include datacenter networks and machine learning systems (with a special focus on federated machine learning).



Kai Chen (Senior Member, IEEE) received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2012. He is currently an Associate Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong. His research interests include data center networking, machine learning systems, and privacy-preserving AI infrastructure.

Kun Tan is currently the Director of the Distributed and Parallel Software Laboratory, 2012 Labs, Huawei. Before joining Huawei in 2016, he was a Senior Researcher and Research Manager at Microsoft Research Asia, Beijing. His research interests include networked systems, cloud networking, and mobile computing. He received the Best Paper Award at NSDI 2009 and USENIX Test-of-Time Award in 2019.

Yongqiang Xiong (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 1996, 1998, and 2001, respectively. He is currently with the Networking Researching Group, Microsoft Research Asia, as a Principal Researcher and Research Manager. His research interests include system and networking and network security. He has published over 80 articles, and served as a TPC member or reviewer for the international key conferences and leading journals in the areas of system and networking.