One More Config is Enough:
Saving (DC)TCP for High-speed *Extremely* Shallow-buffered Datacenters

Wei Bai[1*], Shuihai Hu[1*], Kai Chen[1], Kun Tan[2], Yongqiang Xiong[3]
[1]Sing Lab@HKUST, [2]Huawei, [3]Microsoft Research

***Abstract*—The link speed in production datacenters is growing fast, from 1Gbps to 40Gbps or even 100Gbps. However, the buffer size of commodity switches increases slowly, e.g., from 4MB at 1Gbps to 16MB at 100Gbps, 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 loss or substantial throughput degradation.**

To this end, we present BCC[1], a simple yet effective solution that requires just one more ECN config (i.e., shared buffer ECN/RED) over prior solutions. BCC operates based on real-time global shared buffer utilization. When available buffer space suffices, BCC delivers both high throughput and low packet loss rate as prior work; Once it gets insufficient, BCC automatically triggers the shared buffer ECN to prevent packet loss at the cost of sacrificing little throughput. BCC is readily deployable with existing commodity switches. We validate BCC's hardware feasibility in a small 100G testbed and evaluate its performance using large-scale simulations. Our results show that BCC maintains low packet loss rate while slightly degrading throughput when the available 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 degrading average FCT for large flows by up to 3%.

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 requirement of applications.

TCP is the dominant transport protocol in production DCNs. However, it was a challenge for traditional TCP to achieve good performance on both metrics that are essentially at odds, especially with shallow-buffered switches. This challenge was identified almost 10 years ago by Microsoft researchers. To address it, they proposed DCTCP [9] which leverages ECN [33] to strike the tradeoff between throughput and latency, and showed that a properly configured per-port ECN/RED marking scheme can well utilize the shallow buffer to achieve both high throughput and low latency, while still reserving certain headroom for absorbing micro-bursts. Since then, TCP/ECN variants[2] become popular [9, 28, 36, 39] and are widely adopted in industry. For example, DCTCP has been integrated into various OS kernels [4, 5] and deployed in many production DCNs [24, 35].

However, in this paper, we call the community's attention that this seemingly solved problem resurges and the solution is now being re-challenged, due to the recent industry trend—To accommodate more traffic, 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 dramatically from 85KB to 3.44KB, leading to an *extremely shallow-buffered DCN* (§II-C).

We find it is hard for prior TCP/ECN solutions (including DCTCP) to remain effective when buffer is extremely shallow (§III). On the one hand, if we use a standard ECN marking threshold as DCTCP [9], it will cause packet loss even before ECN starts to react when many 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, since ECN will over-react. Our results 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 (§III-C).

The contribution of this work is to uncover the above problem, demonstrate its consequences, and introduce an effective and readily deployable solution called BCC to address it.

At its core, BCC inherits the success of per-port ECN/RED from DCTCP [9], and further enables shared buffer ECN/RED to cope with the extremely shallow buffer scenario (§IV). BCC is indeed simple: one more ECN config (*i.e.*, shared buffer ECN/RED) is enough! Such shared buffer ECN/RED follows the vanilla RED algorithm [17] but tracks the occupancy of the global shared buffer pool to mark packets. It is worthwhile to note that while this function is available on chips [3, 8, 13], 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 few 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 [9]. As more and more ports become active, the available buffer 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

*Wei is now at Microsoft Research, and Shuihai is now at CLUSTAR Technology Ltd. This work was done when they were both at HKUST SING Lab.

[1]**B**uffer-aware Active Queue Management (AQM) for **C**ongestion **C**ontrol.

[2]In this paper, by TCP we generally refer to various TCP-variants, such as DCTCP [9] and ECN* [39], etc., that are designed for datacenters.

shared buffer ECN/RED, BCC leverages probabilistic marking (instead of DCTCP's single cut-off marking) to desynchronize flows across ports, in order to avoid global synchronization and further throughput loss.

We evaluate the performance of BCC using ns-2 simulations over a 128-node leaf-spine topology with two realistic workloads derived from production DCNs (§V-A). We find:

- At low loads (few ports active), BCC fully utilizes the link capacity and achieves high throughput. For example, compared to a conservative ECN configuration, BCC achieves up to 19.3% lower average FCT for large flows;
- 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%;

We further implement BCC in a small testbed with a 100G Arista 7060CX-32S-F switch connecting 6 servers (§V-B). We show BCC can be enabled by one more command at the switch (Figure 8). 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 aim is to show the hardware feasibility of BCC, leaving large-scale deployment as future work.

In what follows, we will 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.

II. Buffer is becoming extremely shallow

A. Understanding the switch buffering

On the switching chip, the Memory Management Unit (MMU) allocates on-chip buffer memory to incoming packets. The buffer memory is divided into several memory pools, which can be classified into the following two categories:

- **Private pools**: dedicated buffers that have been reserved to switch egress queues.
- **Shared pools**: 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:** The MMU uses Dynamic Threshold (DT) algorithm [15] to allocate buffer space in the shared pool (more details in §IV-B). If there is 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. Buffer requirement of TCP at high-speed

TCP is the dominant transport in today's datacenters [9, 34, 35]. The switch buffer has a great impact on datacenter TCP variants [9, 36, 39]. Moderate buffer occupancy is necessary for high throughput [12]. Meanwhile, we also need certain buffer headroom to absorb transient busrts (*e.g.*, incast [9, 37, 38]). Insufficient switch buffer would cause: 1) *low throughput*, thus slowing bulk transfers; and 2) *packet losses*, thus resulting in long tail FCT for short flows. The tail FCT matters because the performance of many interactive applications seriously degrades even if a small fraction of messages are late [16].

Unlike Internet, the number of concurrent large flows in DCN is typically small [9]. In such case, to achieve full throughput, TCP requires at least $C \times RTT \times \lambda$ buffer space per port, where C is the link capacity, RTT is the round-trip time, and λ is a characteristic constant depending on the congestion control algorithm (*e.g.*, $\lambda = 1$ for ECN* [39]). In recent years, the link speed in DCNs has increased significantly, from 1Gbps to 40/100Gbps. However, the base RTT does not change much as it is mainly determined by processing overhead from multiple sources. Thus, the buffer demand of TCP increases nearly in proportion to the link speed.

Testbed measurement: We measure the buffer demand of conventional TCP in our testbed. Three servers (Mellanox CX-4 100Gbps NIC, Linux kernel 3.10.0) are connected to an Arista switch. To reduce system overhead, various optimization techniques, *e.g.*, TCP segmentation offload (TSO) and generic receive offload (GRO), are enabled. The base RTT in our testbed is $\sim 30\mu$s. We consider DCTCP [9] and ECN* [39] (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. To fully utilize the link capacity, we generate 16 TCP long-lived flows using `iperf` from two senders to a receiver. We vary the ECN/RED marking threshold[3] 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 threshold achieving 100% utilization.

Figure 1 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 [10] (0.17×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

[3]We set the maximum and minimum queue length thresholds of RED [17] to the same value as previous work [9, 39] suggests.

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

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].

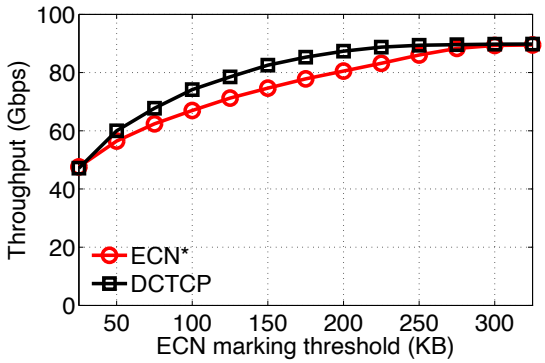


Fig. 1: [Testbed] Aggregate TCP throughput with different ECN/RED marking thresholds

in [10], thus resulting in the theory-practice gap [9, 39]. We also conduct the above experiment using Windows Server 2012 R2 and observe that DCTCP requires ∼60-70% BDP buffering for 100% throughput in practice.

Production DCNs: Compared to the small testbed, production DCNs are more complex and have larger base RTTs. At the end host, packets may experience high kernel processing delay when CPUs are busy [20]. In virtualized environments, hypervisors introduce extra processing overhead. In the network, packets experience non-negligible processing delay when going through various middleboxes [18, 29]. Long-distance cables and multiple switch hops also bring several-microsecond delay. All these factors greatly increase the actual latency in production environments. A measurement study [21] shows that production DCNs have up to 100s of μs latency. Such latency transfers to a large buffer demand. Consider a 100Gbps network with 100μs base RTT, the per-port buffer requirement of ECN* reaches 1.25MB ($100Gbps \times 100\mu s$).

C. Buffer becomes increasingly insufficient

However, the buffer size of commodity switches does not increase as expected. We list buffer and capacity information of some widely used commodity switching chips in Table I. The link capacity significantly outpaces the buffer size, resulting in decreasing buffer per port per Gbps (from 85KB to 3.44KB). The reasons behind are at least two-fold.

- The memory used in switch buffers is high-speed SRAM. Compared to DRAM, 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 high-speed extremely shallow-buffered DCNs.

A. Standard ECN configuration causes excessive packet losses

Most datacenter TCP variants [9, 36, 39] leverage ECN to achieve high throughput with certain buffering. In current practice, DCN operators configure a moderate ECN marking threshold (*i.e.*, $C \times RTT \times \lambda$) for them to achieve 100% throughput, which we called *standard ECN configuration* in this paper. However, such standard ECN configuration is likely to overfill extremely shallow buffers when multiple ports are active, causing packet losses and long tail latency.

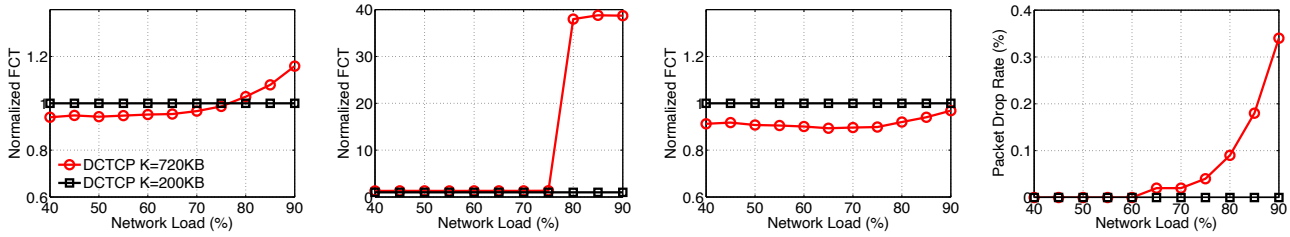
Take Broadcom Tomahawk with 16MB buffer and 32 100Gbps ports as an example. If TCP desires 1MB marking threshold per port, the switch buffer will be overfilled when more than half of total ports ($\geq$16) are active. What's worse, Tomahawk has 4 switch cores to achieve desired performance at the high-speed. 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.

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, such *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 Demonstration

Since we do not have enough servers to saturate the switch buffer in the testbed, we use ns-2 [7] simulations to quantify the impact of the above two performance impairments.



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

Fig. 2: [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.

Topology: 32 servers are 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, we attach every 8 switch ports to a 3MB shared buffer pool. Each port also has 128KB private buffer. All switch ports have the same $\alpha = 4$ for dynamic buffer sharing (see §IV-B for introduction of α).

Workloads: Among 32 servers, 24 servers 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 [9]. We vary the network utilization from 40% to 90%.

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

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

Results analysis: Figure 2 shows FCT and packet loss rate. 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 validates our analysis in §III-B.

At high loads (more ports active), the 720KB threshold causes excessive packet losses, thus degrading the 99th percentile FCT for small flows. For example, as shown in Figure 2(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 [21]. 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$51.2 (5380μs to 103μs) higher than that with the 200KB threshold. This confirms 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 [22, 26], when contradiction 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 DCNs, 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 [26].

Deployability: Our solution should work with existing commodity switches and be backward compatible with legacy TCP/IP stacks. Modifying switch hardware is costly as a new ASIC often takes years to design and implement. We note that some prior DCN transport designs [11, 14, 19, 22, 31, 32, 42] may also work with extremely shallow buffers. But, as we will discuss in §VI, these solutions require non-trivial modifications to switch hardware and network stacks or make unrealistic assumptions, making them hard to deploy in practice.

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 has a single egress queue and no buffer

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 II: Shared buffer model parameters

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

TABLE III: Shared buffer model variables

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., [9, 28, 36, 39]) can be enabled at the end. The standard ECN setting is configured on each port/queue for high throughput.

Today's commodity switching chips typically use Dynamic Threshold (DT) algorithm [15] 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 $T(t)$ to limit the queue length. $T(t)$ is actually a function of the unused shared buffer size and α as follows:

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

A packet arriving in queue i at time t will get dropped if $Q_i(t) \geq T(t)$. As analyzed in [15], 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. Typically, α values are set powers of 2 for implementation simplicity (*e.g.*, 1/128 to 8 in Tomahawk).

Assume that our TCP variant (with standard ECN) 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 how to determine B_R later in §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 §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

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

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 exploit the shared buffer ECN/RED which is already available in commodity switches [3, 8, 13]. Shared buffer ECN/RED follows the original RED [17] 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, it 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 [9]. Both high throughput and low packet loss rate can be achieved.
- When more and more ports become active, the available buffer 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 max and min 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 [9], all flows across ports sharing the same buffer pool are likely to reduce their windows at the same time. This causes global synchronization problem and affects throughput [17]. 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 high throughput and low packet loss rate. With B_R fixed, we then decide marking thresholds and probability of shared buffer ECN/RED. In this section, we give several useful rules-of-thumbs to set parameters while leaving optimal settings as future work.

Determine B_R: Given the number of concurrent large flows in DCN is typically small [9], 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 full throughput.

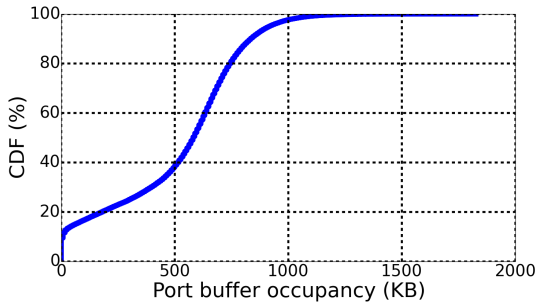


Fig. 3: [Simulation] CDF of buffer occupancies of the congested port at 90% load.

However, the lag in ECN control loop imposes extra buffer requirement to avoid packet losses. When a packet gets ECN marked at switch [43], the sender will reduce its window after one RTT. During this interval, extra buffer space is required to absorb more packets. 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. Thus, we need $C \times RTT$ extra buffer to avoid packet loss 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 case, it is less likely that all active flows enter slow start phase simultaneously, thus reducing the switch queue gradient. However, the arrivals and departures of flows also affect the switch 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 [9]. The average link utilization is 90%. We increase the shared buffer to 10MB, which eliminates packet loss in the network. The other settings are same as those in §III-C. We configure the per-port ECN/RED marking threshold to 720KB ($C \times RTT \times \lambda$). Thus, $C \times RTT \times (1+\lambda) = 1720$KB. Figure 3 plots the CDF of buffer occupancies of the congested port. Around 25% of them are larger than 720KB, suggesting $C \times RTT \times \lambda$ is not enough. The 99.99th percentile buffer occupancy is 1609KB $< C \times RTT \times (1+\lambda)$. Hence, we envision that $C \times RTT \times (1+\lambda)$ works well for mixed 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 [21, 39] 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: min threshold K_{min}, max threshold K_{max} and max 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 probabilistically; 3) above 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 [17] 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, 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 $K_{max} - K_{min}$. In our evaluation, we set $h = 8\mu$s. For the maximum marking probability P_{max}, we set it to 10% as [17] suggests.

V. Evaluation

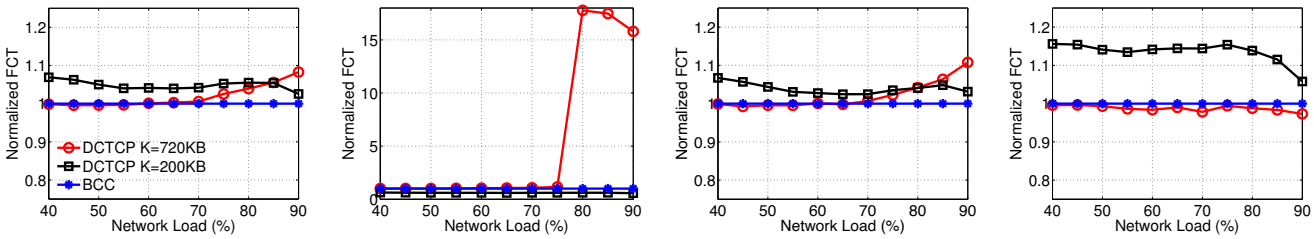
In this section, we evaluate BCC using both large-scale ns-2 [7] simulations and small-scale testbed experiments. We highlight some of the results as follows:

- Our simulations (§V-A) with realistic workloads 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 testbed implementation (§V-B) validates that BCC is easy to configure at switches and functional as expected.

A. Large-scale Simulations

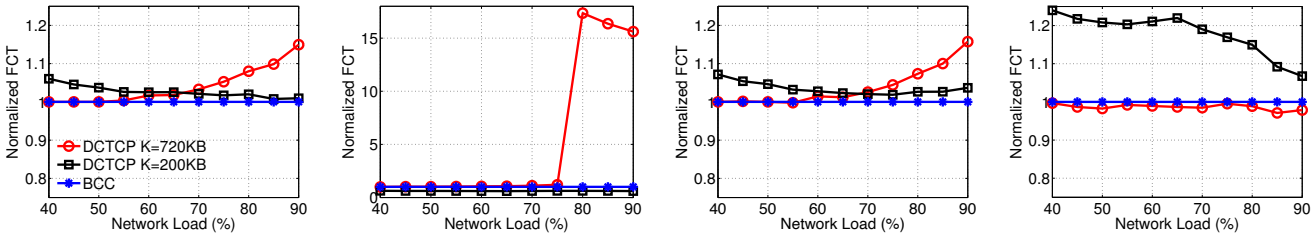
Topology: We simulate a 128-host leaf-spine topology with 8 leaf (ToR) switches and 8 spine (Core) switches. Each leaf switch has 16 100Gbps down links to hosts and 8 100Gbps up links to spines. Hence, we have a 2:1 oversubscription, which is common in production datacenters. We employ Equal-Cost Multi-Path routing (ECMP) for load balancing. The base fabric RTT across the spine is $\sim$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



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

Fig. 4: [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

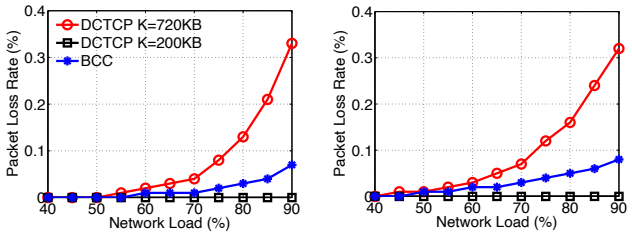
Fig. 5: [Simulation] FCT results for the cache workload. Results are normalized to values of BCC.

addition, each switch port has 128KB static reserved buffer. At the end host, we allocate 10MB static buffer for each NIC.

Workloads: We generate realistic workloads according to two flow size distributions in production: web search [9] from Microsoft and cache [34] from Facebook. Both distributions are heavy-tailed. We generate flows according to a Poisson process and choose the source/destination for each flow uniformly at random. We vary the flow arrival rate to achieve desired load in the fabric. Each simulation lasts for 100,000 flows.

Schemes compared: We use DCTCP [9] 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 [25]. We exclude PFC due to its large buffer reservation requirement. We compare the following three schemes:

- **DCTCP K=720KB**: This is the standard ECN configuration (current practice). We configure the per-port (queue) ECN/RED marking threshold to 720KB (0.72BDP based on measurement in §II-B) to achieve 100% throughput.
- **DCTCP K=200KB**: This is the conservative ECN configuration. We configure the per-port (queue) ECN/RED marking threshold to 200KB to reduce packet losses.
- **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, $B_R = C \times RTT \times (1+\lambda) - S_{min}$ ≈1.6MB. Therefore, $K_{max} = B - B_R/\alpha$ ≈2.6MB. Since each buffer pool is shared by $N = 8$ ports and the recommendation value for h is 8μs, $K_{min} = K_{max} - C \times N \times h$ ≈1.8MB. P_{max} is set to 10%.



(a) web search workload (b) cache workload

Fig. 6: [Simulation] Packet loss rate results.

Performance metrics: We use FCT as the primary metric and also measure packet loss rate in certain simulations for analysis. Besides overall, we 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 DCTCP K=720KB by default.

Results analysis: Figure 4 and 5 give FCT results for two workloads. We also plot packet loss results in Figure 6. We make the following observations.

At low loads, BCC performs similarly as K=720KB while outperforming K=200KB. K=200KB only shows some performance advantage (∼100μs) on small flows.

As shown in Figure 6, K=720KB keeps 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 aggressively regardless of the sufficient buffer space, thus causing much unnecessary bandwidth wastage. In such scenarios, due to the low shared buffer occupancy, BCC marks packets just as the standard per-port ECN configuration without triggering the 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 generally outperforms the other two schemes. Specifically, BCC delivers the best performance for medium flows while approximating the best scheme for small flows (*i.e.*, K=200KB) and large flows (*i.e.*, K=720KB) respectively.

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 6, the packet loss rate with K=720KB approaches 0.1% (SLA threshold used in Microsoft [21]) 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.

For large flows, BCC achieves comparable performance as K=720KB. 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 suggests that the performance trade-off made by BCC does not impact too much on large flows. We think that the lower packet loss rate with BCC can make up for throughput loss to some degree. By contrast, K=200KB is still so conservative that it increases FCT by at least ∼9% compared to K=720KB for both workloads.

B. Small-scale Testbed Validation

Due to scale limitation, we only use testbed micro-benchmarks to 1) validate that BCC is readily-deployable at commodity switches, and 2) validate its functionalities in two scenarios: single MMU and multiple MMUs.

Testbed setup: Our testbed consists of 6 servers connected to an 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. 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.

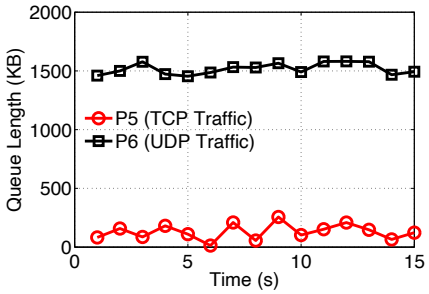


Fig. 7: [Testbed] Queue length samples of port 5 and 6. Note that shared buffer ECN/RED is disabled. Traffic to P5 is TCP while traffic to P6 is UDP.

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

Fig. 8: Command to enable shared buffer ECN/RED on Arista EOS [8]. Both minimum and maximum thresholds are set to 500KB.

Our Arista switch is built with Broadcom Tomahawk [6] chip. It has 4 MMUs, each with 4MB buffer. The dynamic buffer allocation only happens within the single MMU. In each MMU, ∼1MB buffer has been reserved and 3MB buffer can be dynamically allocated. Each server has a Mellanox CX-4 NIC and runs Linux 3.10.0 kernel. Various system optimizations, *e.g.*, TSO and GRO, are enabled. We use ECN* [39] (regular ECN-enabled TCP) in experiments as it is more sensitive to different ECN marking settings. To fully utilize the link capacity, we configure the per-queue ECN/RED marking threshold to 325KB according to Figure 1.

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 7 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[4]. In theory, P6 can get almost half of the total shared buffer space according to dynamic threshold algorithm [15], which is ∼1.5MB. Our testbed results closely match the theory results.

Now, we enable shared buffer ECN/RED using a single command shown in Figure 8. For simplicity, both minimum

[4]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.

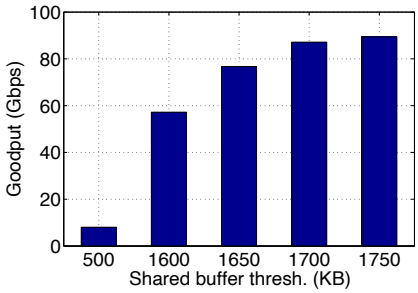


Fig. 9: [Testbed] Aggregate goodput of TCP traffic to S5 with different shared buffer ECN thresholds.

and maximum thresholds of shared buffer ECN/RED are set to the same value in our testbed experiments.

Figure 9 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 ∼8Gbps TCP goodput. After we increase the threshold above 1.5MB, the TCP goodput keeps increasing and reaches 90Gbps with 1750KB threshold.

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

Validating BCC in 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.

In this experiment, we validate BCC in multiple MMUs. For this purpose, 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 ∼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 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.

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

TABLE IV: [Testbed] TCP goodput results with different shared buffer ECN thresholds. Note that packets to S5 and S6 are stored in different MMUs.

VI. Related Work

The literature on DCN transport is vast. Here, we only cover some closely related work that we have not discussed elsewhere in the paper.

Bufferless DCN transport: Many DCN transport designs [11, 14, 19, 22, 27, 31, 32, 42] can cope with extremely 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 [19], ExpressPass [14] and Homa [27] require clean-slate network stacks. HULL [11] and TFC [42] adopt TCP at the end host, but require non-trivial modifications to switch hardware. Fastpass [32] and Flowtune [31] require changes to network stacks and leverage a centralized scheduler, which suffers from failures and scalability issues. NDP [22] modifies both switch hardware and network stacks. Furthermore, some of them make unrealistic assumptions to the underlying network. For example, pHost [19] and NDP [22] assume that the congestion-free network core, which does not hold for many production DCNs. ExpressPass [14] 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.

Other work: PERC [23] targets at fast convergence in high-speed DCNs. DIBS [40] achieves a near lossless network by detouring packets. Some efforts [30, 37, 41] have been made to reduce the impact of packet losses in DCNs. 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 config at commodity switches. BCC operates based on 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 feasibility in a 100G testbed and demonstrated its superior performance over current practice using extensive simulations.

Acknowledgement: This work is supported by the Hong Kong RGC 16215119, and a research funding from Huawei.

REFERENCES

[1] https://people.ucsc.edu/~warner/Bufs/7060CX.html.
[2] https://people.ucsc.edu/~warner/Bufs/tomahawk.
[3] "Cisco Nexus 3000 Series NX-OS QoS Configuration Guide, Release 7.x," 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," http://kernelnewbies.org/Linux_3.18.
[5] "DCTCP in Windows Server 2012," http://technet.microsoft.com/en-us/library/hh997028.aspx.
[6] "High-Density 25/100 Gigabit Ethernet StrataXGS Tomahawk Ethernet Switch Series," https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm56960.
[7] "The Network Simulator NS-2," http://www.isi.edu/nsnam/ns/.
[8] "User Manual of Arista EOS version 4.15.0F," https://www.arista.com/assets/data/docs/Manuals/EOS-4.15.0F-Manual.pdf.
[9] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM 2010*.
[10] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: stability, convergence, and fairness," in *SIGMETRICS 2011*.
[11] 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 *NSDI 2012*.
[12] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *SIGCOMM 2004*.
[13] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in Multi-Service Multi-Queue Data Centers," in *NSDI 2016*.
[14] I. Cho, K. Jang, and D. Han, "Credit-Scheduled Delay-Bounded Congestion Control for Datacenters," in *SIGCOMM 2017*.
[15] 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. [Online]: http://dx.doi.org/10.1109/90.664262.
[16] J. Dean and L. A. Barroso, "The Tail at Scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013. [Online]: http://doi.acm.org/10.1145/2408776.2408794.
[17] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, pp. 397–413.
[18] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud Scale Load Balancing with Hardware and Software," in *SIGCOMM 2014*.
[19] 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 *CoNEXT 2015*.
[20] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over Commodity Ethernet at Scale," in *SIGCOMM 2016*.
[21] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis," in *SIGCOMM 2015*.
[22] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *SIGCOMM 2017*.
[23] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, "High Speed Networks Need Proactive Congestion Control," in *HotNets 2015*.
[24] G. Judd, "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter," in *NSDI 2015*.
[25] G. Judd, "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter," in *NSDI 2015*.
[26] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *SIGCOMM 2015*.
[27] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities," in *SIGCOMM 2018*.
[28] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *INFOCOM 2013*.
[29] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri, "Ananta: Cloud Scale Load Balancing," in *SIGCOMM 2013*.
[30] R. S. C. L. Peng Cheng, Fengyuan Ren, "Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Centers," in *NSDI 2014*.
[31] J. Perry, H. Balakrishnan, and D. Shah, "Flowtune: Flowlet Control for Datacenter Networks," in *NSDI 2017*.
[32] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A Centralized "Zero-queue" Datacenter Network," in *SIGCOMM 2014*.
[33] K. Ramakrishnan, S. Floyd, D. Black *et al.*, "RFC 3168: The addition of explicit congestion notification (ECN) to IP," 2001.
[34] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in *SIGCOMM 2015*.
[35] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," in *SIGCOMM 2015*.
[36] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *SIGCOMM 2012*.
[37] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *SIGCOMM 2009*.
[38] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in data center networks," in *CoNEXT 2010*.
[39] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *CoNEXT 2012*.
[40] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "DIBS: Just-in-time Congestion Mitigation for Data Centers," in *EuroSys 2014*.
[41] D. Zats, A. P. Iyer, G. Ananthanarayanan, R. Agarwal, R. Katz, I. Stoica, and A. Vahdat, "FastLane: Making Short Flows Shorter with Agile Drop Notification," in *SOCC 2015*.
[42] J. Zhang, F. Ren, R. Shu, and P. Cheng, "TFC: token flow control in data center networks," in *EuroSys 2016*.
[43] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY," in *CoNEXT 2016*.