Enabling ECN over Generic Packet Scheduling

Wei Bai[1] Kai Chen[1] Li Chen[1] Changhoon Kim[2] Haitao Wu[3]

[1]SING Group, Hong Kong University of Science and Technology
[2]Barefoot Networks [3]Microsoft

ABSTRACT

Explicit Congestion Notification (ECN) is crucial for production datacenters, but current queue-length based ECN/RED implementation does not work with generic packet schedulers, leading to either degraded network performance or violated scheduling policies. In this paper, we first dive into this issue and reveal that the invalidity of ECN/RED lies in the difficulty of measuring changing queue capacities under various schedulers and traffic dynamics. Then we present Time-based Congestion Notification (TCN), a simple yet effective ECN solution, by combining two successful ideas: the sojourn time from CoDel [23] and the instantaneous marking from DCTCP [6]. Using packet sojourn-time, as opposed to queue-length, as the congestion signal, TCN eliminates the need of measuring dynamic queue capacities, making it suitable for arbitrary schedulers with traffic dynamics. By performing stateless instantaneous ECN marking rather than complex stateful dropping, TCN is designed to be inexpensive to implement on commodity switching chips. Through extensive testbed experiments and large-scale simulations, we show TCN can strictly preserve scheduling policies while providing desirable network performance. For example, TCN significantly reduces the average and 99th percentile completion times for small flows by up to 82.8% and 95.3% compared to current practice in a testbed experiment with production workload.

1. INTRODUCTION

Datacenters must simultaneously deliver high throughput and low latency to applications they host—some such as web search [6] and distributed memory caches [24] require low latency for short messages, while others like cloud storage and data-parallel computation [15] desire high throughput for large data transfers. To accommodate these two requirements, ECN is adopted by recent datacenter transport designs [6, 8, 22, 32, 35, 37], and they have shown that a well-tuned ECN marking threshold can deliver both high throughput and low latency [6, 35]. For this reason, ECN-based transports, like DCTCP [6] and DCQCN [37], have been widely implemented in various OS kernels [1, 2] and NICs [37] and deployed in production datacenters [19, 28, 37].

Basically, ECN requires both ECN-enabled transport protocols at end hosts and ECN-capable switches in the network. At the switch, ECN is typically combined with an active queue management (AQM) scheme. Current switching chips typically adopt random early detection (RED) to enable ECN [5]. In ECN/RED, an ECN mark on a packet simply indicates that the average or instantaneous buildup of the queue corresponding to the packet exceeds a static ECN marking threshold.

Meanwhile, existing commodity switching chips also support multiple queues (typically 4-8 [9, 10]) per egress port. To improve network performance and provide QoS, the current practice is to use these multiple queues to classify traffic (*e.g.*, interactive applications vs data backup) and enforce various scheduling algorithms (*e.g.*, Weighted Fair Queueing (WFQ) or Strict Priority (SP)) among different queues/classes.

However, current ECN/RED marking scheme does not support generic packet schedulers [11, 21]. The crux is that RED uses queue-length as the congestion signal, and compares it against a pre-configured ECN marking threshold to mark packets. The ideal queue-length marking threshold depends on the queue capacity (the effective instantaneous drain rate of a queue). When multiple queues share a port, the individual queue capacities can vary significantly subject to both scheduling disciplines and traffic dynamics. This entails changing ECN marking thresholds in response to dynamic queue capacities. But today's commodity switches only support static thresholds, leading to either degraded performance or violated scheduling policies (§3.2).

Recently, MQ-ECN [11] makes a first effort to enable dynamic ECN/RED for multi-queue. It leverages special properties of round-robin schedulers to calculate dynamic ECN marking threshold for each queue. While it works for round-based schedulers like Weighted Round Robin (WRR) and Deficit Weighted Round Robin (DWRR), this solution does not generalize to other schedulers such as WFQ and SP that do not possess the concept of "round". Furthermore, with the advent of programmable schedulers such as PIFO [30] and UPS [20], we are faced with more sophisticated packet scheduling algorithms that MQ-ECN does not apply.

CoNEXT '16, December 12-15, 2016, Irvine, CA, USA
 ISBN 978-1-4503-4292-6/16/12…$15.00
DOI: http://dx.doi.org/10.1145/2999572.2999575

In this paper, we move one step further and ask a fundamental question: *can we enable ECN over arbitrary packet schedulers in datacenter networks?*

Towards answering this question, we first revisit ECN/RED (§3). In order to enable ECN/RED for generic packet schedulers, the key is to seek a generic solution to estimate dynamic queue capacities. However, through extensive experiments, we find that it is very hard to accurately measure dynamic queue capacities in practice. The reason is that it is extremely difficult to determine an ideal measurement window for generic packet schedulers with traffic dynamics. A smaller measurement window (*i.e.*, sampling too frequently) degrades accuracy, while a larger one (*i.e.*, sampling insufficiently) fails to timely capture fine-grained queue capacity changes (§3.3).

Based on above observation, we exploit latency experienced by a packet, instead of queue-length, as the congestion signal and present Time-based Congestion Notification (TCN), a simple yet effective sojourn-time based solution (§4). TCN synthesizes two successful ideas: the sojourn time from CoDel [23] and the instantaneous marking from DCTCP [6]:

- By using the sojourn time as the congestion signal, the marking threshold of TCN is *independent* of the changing queue capacities, making it particularly suitable for arbitrary packet schedulers (§4.1).
- With *instantaneous* ECN marking, TCN operates in a stateless fashion. This makes TCN much simpler than CoDel [23] which tracks the minimum sojourn time over a complex varying time window to drop (or mark) packets. Therefore, we expect TCN will be simpler to implement in hardware (§4.2). Moreover, compared to CoDel, TCN achieves faster reaction to bursty datacenter traffic (§6.1).

We have implemented a TCN software prototype as a Linux `qdisc` kernel module which supports a variety of packet scheduling algorithms (§5), and built a small-scale testbed with 9 servers connected to such a server-emulated TCN software switch. We use this testbed to evaluate basic properties of TCN (§6.1). Our experiment results demonstrate that TCN can strictly preserve different scheduling policies while delivering high throughput and low latency simultaneously. For example, in a realistic datacenter web search workload [6], we find that TCN significantly reduces the average and 99th percentile completion times for small flows by up to 82.8% and 95.3% respectively while delivering similar performance for large flows, compared to current operation practice (§6.1.3).

To complement small-scale testbed experiments, we also conduct large-scale ns-2 [4] simulations with 4 realistic datacenter workloads (§6.2). Our simulation results confirm that TCN maintains its superior performance in large-scale multi-hop datacenter topologies, For example, TCN achieves up to 94.3% lower 99th percentile completion time for small flows compared to current practice (§6.2.1). We further use a series of targeted simulations to show that TCN is robust to different network settings, such as the number of queues and transport protocol (§6.2.2).

To make our work easy to reproduce, we make our experiment and simulation code available online at http://sing.cse.ust.hk/projects/TCN.

2. BACKGROUND

2.1 ECN/RED

In current switching chips, ECN is typically combined with RED [16] that uses *average* buffer occupancy[1] as the congestion signal. RED has (at least) 3 parameters to configure: low buffer occupancy threshold K_{min}, high buffer occupancy threshold K_{max}, and maximum probability P_{max}.

In production datacenters, network operators usually make two changes to simplify ECN/RED configuration [6, 35]. First, they use *instantaneous* buffer occupancy rather than average buffer occupancy to faster react to traffic bursts. Second, they set both K_{min} and K_{max} to K. An arriving packet gets ECN marked only when the instantaneous buffer occupancy is larger than K. In this paper, we also adopt such simplified ECN/RED scheme.

Obviously, the choice of marking threshold K greatly affects network performance. Unlike Internet, the number of concurrent large flows is very low in datacenter environments (*e.g.*, 2 in the 75th percentile) [6]. Given such low degree statistical multiplexing, we consider an ideal model where several synchronized long-lived flows with identical round-trip times RTT share a bottleneck link with capacity C. As previous works [6, 11, 35] have shown, to fully utilize link capacity while delivering low latency, the ECN marking threshold K should be set as follows:

$$K = C \times RTT \times \lambda \tag{1}$$

where λ is a parameter determined by congestion control algorithms (*e.g.*, $\lambda = 1$ for ECN* [35][2]). In this paper, we call such $C \times RTT \times \lambda$ as the *standard marking threshold for ECN/RED*.

Prior experience has shown that computing the standard marking threshold in production datacenters is feasible [35]. This is because, unlike Internet, round-trip times in datacenter networks are relatively stable and can be well estimated through large-scale measurements [18, 35]. For example, in a Microsoft production datacenter, the 23%, 74% and 90% percentile inter-rack RTT are around $200us$, $300us$ and $400us$, respectively [35]. Moreover, since operators have full control of end-host network stacks, λ can be easily obtained. Hence, it is fairly easy to compute such a static standard marking threshold in production datacenters.

2.2 Packet Scheduling

Current commodity switches usually provide some fixed-function packet scheduling algorithms (*e.g.*, SP and DWRR) across several queues (*e.g.*, 4-8) per egress port. Packet schedulers are used by operators to improve network performance. Here, we give two examples.

- **Inter-Service Traffic Isolation:** Modern production datacenters host a variety of services (applications) with diverse network requirements. For example, a real-time online service desires ultra-low latency for short messages while a

[1] We use buffer occupancy and queue length interchangeably here.

[2] ECN* is regular ECN-enabled TCP which simply cuts the window by half in the presence of an ECN mark.

data backup service requires high throughput for large flows. Current practice is to use queues to isolate traffic from different services [9, 11] and configure DWRR/WRR among the queues. In this way, we can reserve a minimum guarantee bandwidth for each service and mitigate inter-service network interference to some extent.

- **Traffic Prioritization:** A service may have a small amount of important traffic, such as SDN control traffic, BGP control traffic, RDMA Congestion Notification Packets (CNP) [37], ACK packets [21], latency-sensitive query messages. The completion times of this traffic greatly affect service operation and user-perceived performance. Hence, network operators usually reserve some (typically one) strict higher priority queues to prioritize the delivery of such important traffic [11]. The rest of queues in the lowest priority can still be used for inter-service isolation.

Further, very recently, Sivaraman et al. [30] have proposed a programmable packet scheduler allowing operators to implement custom scheduling algorithms for specific application-level objectives. Mittal et al. [20] have shown that the classical Least Slack Time First (LSTF) scheduling could be used to emulate different scheduling algorithms in practice. Given above efforts in programmable schedulers, we envision that more and more scheduling algorithms will be used in the future.

2.3 Combining ECN & Packet Scheduling

In some production datacenters, ECN and packet scheduling are used simultaneously, and operators usually configure them in a hierarchical style: they employ various packet scheduling algorithms among different queues while leveraging ECN to achieve high throughput and low latency in the same queue. Given that more and more packet scheduling algorithms will be used in the future, it is important to find an ECN marking scheme that can efficiently support any packet scheduler.

3. ECN/RED DEEP DIVE

In this section, we first present the ideal ECN/RED solution for generic packet scheduling. Then, we show that current practice is far from the ideal. Finally, we demonstrate that the ideal ECN/RED for generic schedulers is intrinsically hard to realize, motivating our design of TCN.

3.1 Ideal ECN/RED for Generic Schedulers

We consider a switch egress port with multiple queues. We consider a per-queue ECN/RED marking model where each queue only tracks its own buffer occupancy and performs ECN marking independently to other queues. The ECN/RED marking threshold of queue i is denoted as K_i. Based on Equation 1, K_i should be set as follows:

$$K_i = C_i \times RTT \times \lambda \quad (2)$$

where C_i is the capacity (effective instantaneous drain rate) of queue i, which is determined by traffic demands and the underlying packet scheduler. Thus, given a packet scheduler, the per-queue marking threshold K_i should be dynamically adjusted according to traffic dynamics.

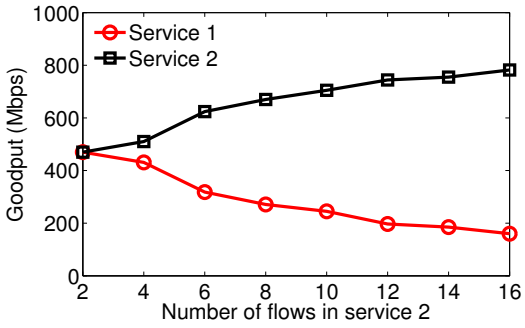


Figure 1: [Testbed] Aggregate goodputs achieved by per-port ECN/RED. Note that service 1 always has 1 flow.

3.2 Current Practice

Today's switching chips provide multiple ECN/RED configuration schemes, such as per-queue, per-port and per-service-pool[3] ECN/RED [11]. The key difference among them is that they use buffer occupancies of different egress entities when making marking decisions. However, all these ECN/RED schemes are essentially *static*: operators can only configure static per-queue/port/service-pool marking thresholds. Hence, compared to the ideal ECN/RED solution that dynamically adjusts the per-queue marking threshold, existing ECN/RED implementation unavoidably introduces some performance impairments. We use the following two examples to illustrate them.

3.2.1 *Per-queue ECN/RED*

Network operators usually configure per-queue ECN/RED for its ideal isolation among different queues [6, 19]. However, how to set fixed per-queue marking threshold is a challenge. Considering that the queue capacity C_i is always no larger than the link capacity C, in current practice, many operators chose to configure the standard marking threshold $C \times RTT \times \lambda$ for each queue [11]. Such configuration can achieve high throughput while strictly preserving any scheduling algorithm. However, when many queues are busy simultaneously, C_i will be much smaller than C. Under such circumstances, the standard marking threshold $C \times RTT \times \lambda$ will result in excess queue buildups, thus degrading packet latency and burst tolerance.

Remark 1: Per-queue ECN/RED with the standard threshold can seriously degrade packet latency and burst tolerance, especially when many queues are busy simultaneously.

3.2.2 *Per-port/service-pool ECN/RED*

Though the per-queue capacity C_i is dynamically changing, the aggregate capacity of all the queues belonging to a switch port keeps the same. Hence, the reader may wonder whether per-port ECN/RED can meet our requirements.

By setting the per-port marking threshold to the standard threshold $C \times RTT \times \lambda$, per-port ECN/RED can achieve high throughput and low latency simultaneously. However, it fails to preserve scheduling policies. A packet, which should not be marked according to Equation 2, may get ECN marked due

[3]A service pool is a shared buffer region.

to the buffer occupancies of the other queues belonging to the same port. Such impact will become more serious if we enable per-service-pool ECN/RED because queues from different ports can interfere with each other.

To confirm such impairment, we conduct a simple testbed experiment. We connect 3 servers to a Pica8 P-3295 GbE switch. We enable DCTCP [6] on the servers. At the switch, we configure per-port ECN/RED and DWRR with 2 equal-quantum queues. We set the per-port marking threshold to 30KB as the DCTCP paper recommends [6]. We start several TCP long-lived flows from two senders to the same receiver. We classify flows into two services based on their senders. Both services have their dedicated queues at the switch.

In this experiment, service 2 has a varying number of TCP flows from 2 to 16 while service 1 always has 1 flow. Ideally, based at the switch scheduling policy, two services should always fairly share the link capacity equally. Figure 1 shows the final goodput sharing results. The aggregate goodput of service 2 becomes higher and higher as we increase its number of flows. For example, service 2 can achieve 670Mbps goodput with 8 flows and 782Mbps goodput with 16 flows. This suggests that, under per-port ECN/RED, traffic from service 1 gets excess ECN marked due to the impact of traffic from service 2, thus violating the scheduling policy.

Remark 2: Per-port/service-pool ECN/RED can seriously violate scheduling policies.

3.3 Fundamental Difficulty for Ideal ECN/RED

Given existing ECN/RED configuration schemes do not satisfy our requirements, we need to design a new AQM to enforce the ideal ECN/RED solution (Equation 2). To perform the ideal ECN/RED marking, the key is to find a general solution to estimate the queue capacity C_i for any packet scheduling algorithm under traffic dynamics.

We observe that queue capacity is equal to actual queue departure rate when the queue remains non-empty. Hence, when a queue always has data in the buffer, we can use the queue departure rate as the estimate for queue capacity. MQ-ECN [11] leverages the special properties of round-robin schedulers (*e.g.*, WRR and DWRR) to realize this. In round-robin schedulers, a non-empty queue i can transmit $quantum_i$ worth of bits at most in each round. Therefore, MQ-ECN uses $quantum_i$ over the round time T_{round}[4] to estimate the capacity of queue i. Since MQ-ECN relies on the special properties of round-robin schedulers, it does not generalize to other schedulers such as WFQ and PIFO [30] that do not have the round concept. To this end, we employ the departure rate measurement technique in Algorithm 1. Note that this is the best solution we learned in PIE [25].

Basically, we start a measurement cycle *only* when the queue length is over a threshold, dq_thresh. This ensures there are sufficient data in the buffer. Then, we use dq_count to track the number of bytes departed in this measurement cycle. Once dq_count crosses dq_thresh, we stop the current cycle and obtain a sample dq_rate. Note that this method ensures that the queue buffer is non-empty during the measurement cycle. After that, we exponentially average dq_rate to calculate a smooth departure rate, avg_rate, which is the estimate for the queue capacity. Finally, we use $avg\_rate \times RTT \times \lambda$ (Equation 2) to calculate the ECN marking threshold.

In the algorithm, the parameter dq_thresh is key, which determines the measurement frequency. Conventional wisdom recommends 10KB [25]. However, we find there is an intrinsic tradeoff in the choice of dq_thresh. In datacenter environment with dynamic traffic, it is unlikely to choose a *right* value for generic packet schedulers. A smaller dq_thresh value (results in too frequent measurement) can degrade measurement accuracy; whereas a larger dq_thresh value (leads to insufficient measurement) cannot accurately capture the dynamic changes of queue capacity. We believe this tradeoff applies to any general departure rate estimation.

To demonstrate such a tradeoff, we conduct a simple ns-2 [4] simulation. We simulate a 10Gbps topology with 11 servers connected to a switch. The base RTT is $100us$. We employ ECN* [35] as the transport protocol. Hence, the standard threshold is 125KB ($100us \times 10Gbps$) in our setup. We configure DWRR with two queues at the switch. Both queues have a quantum of 18KB. We evaluate three ECN/RED schemes: the ideal ECN/RED solution with dq_thresh of 40KB and 10KB and MQ-ECN [11], the state-of-the-art dynamic ECN/RED solution for round robin schedulers. We use 0.875 as the averaging parameter. Among 11 servers, 10 act as the senders and the rest one is the receiver. We first start 8 TCP flows from 8 senders, which are mapped to queue 1. At time 0.01s, we start another 2 TCP flows from the rest 2 senders, which are mapped to queue 2. Hence, the capacity of queue 1 should be decreased to 5Gbps after 0.01s.

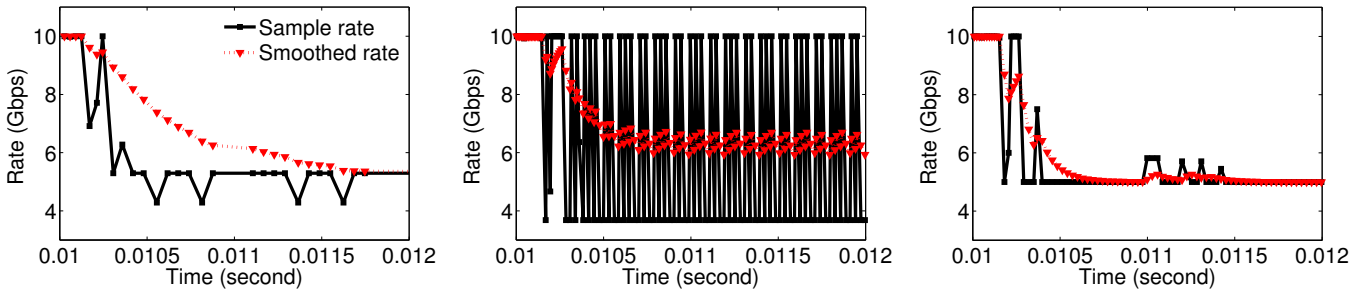
Parameter	Description
dq_thresh	queue length threshold
$is\_measure$	whether in a measurement cycle
dq_count	number of bytes depart in the measurement
dq_start	start time of measurement cycle
$dq\_pktsize$	size of current dequeue packet
dq_rate	sample departure rate
avg_rate	smoothed departure rate

Table 1: Parameters used in Algorithm 1

Algorithm 1 Departure rate (queue capacity) measurement:

Upon packet departure;
1. Decide to be in a measurement cycle:
if $qlen \geq dq\_thresh$ *and* $!is\_measure$ **then**
  $dq\_count = 0$;
  $dq\_start = now$;
  $is\_measure = true$;
2. During the measurement cycle:
if $is\_measure$ **then**
  $dq\_count = dq\_count + dq\_pktsize$;
  if $dq\_count \geq dq\_thresh$ **then**
    $dq\_rate = dq\_count/(now - dq\_start)$;
    $avg\_rate = \epsilon \times avg\_rate + (1 - \epsilon) \times dq\_rate$;
    $is\_measure = false$;

[4] T_{round} is the total time to serve all the queues once.



(a) dq_thresh = 40KB (b) dq_thresh = 10KB (c) MQ-ECN

Figure 2: [Simulation] The capacity of queue 1 estimated by different schemes. Note that rates in (a) and (b) are estimated by Algorithm 1 with different dq_thresh values. In (c), MQ-ECN only works for round-robin schedulers, not others.

Figure 2 plots the capacity of queue 1 estimated by different schemes. Note that we use the smoothed rates in the figures to calculate ECN marking thresholds. We make the following three key observations.

- First, with dq_thresh of 40KB, the ideal ECN/RED solution can only obtain 29 sample rates in 2ms period. Consequently, it takes more than 2ms for the smoothed capacity to converge to 5Gbps, which is too long for highly dynamic datacenter workloads. Moreover, a large dq_thresh may increase packet latency as the ECN marking scheme starts to react only when it sees a large queue buildup.
- Second, with dq_thresh of 10KB (recommended by [25]), the sample rates of the ideal ECN/RED solution drastically oscillate between 3.7Gbps and 10Gbps. This is because dq_thresh (10KB) is smaller than the queue quantum (18KB), we obtain either a very high sample rate (10Gbps) if the measurement cycle is inside the same round or a very low sample rate (3.7Gbps) if the measurement cycle spans across several rounds. Such highly oscillated samples not only result in the oscillation of the smoothed rate, but also lead to a wrong queue capacity estimate above 6Gbps, deviating from 5Gbps by over 20%. Thus, using it for ECN marking threshold calculation will introduce excess latency.
- Third, MQ-ECN [11] can obtain a large number of accurate samples by explicitly using the round time, T_{round}, as the measurement window. Within T_{round}, all the queues exactly get served once and queue i can transmit $quantum_i$ worth of bits at most according to the scheduling policy. Hence, using $quantum_i/T_{round}$ as the estimate, MQ-ECN's smoothed rate quickly converges to 5Gbps within $600us$. Despite its good performance, MQ-ECN can only handle round-robin schedulers. In case of WFQ, SP or other advanced programmable schedulers, there is no way to have such a round, and MQ-ECN does not apply.

Due to the above tradeoff, network operators need to cautiously tune a suitable rate measurement frequency. However, this frequency is related to both underlying packet schedulers and network traffic situations, instead of a static universal value. Such effort is a huge burden for network operators, which is almost impossible to achieve in practice, even for a given scheduler, *e.g.*, WFQ, under highly dynamic traffic.

Remark 3: Network operators need to cautiously tune the rate measurement frequency for the ideal ECN/RED solution. This greatly increases their burden and is hard to accomplish.

4. TCN

The goal of TCN is to enable ECN over generic packet schedulers, while maintaining good network performance. The main problem with ECN/RED, as introduced above, is that its use of queue-length as the congestion signal is conflated with the queue capacity, which is very hard to measure in highly dynamic datacenter networks. Motivated by this, TCN exploits packet sojourn-time, instead of queue-length, as the congestion signal. By using the sojourn time as the congestion signal, the marking threshold of TCN is *independent* of the changing queue capacities, making it suitable for arbitrary packet schedulers.

4.1 Mechanism

We use Q_i to denote the length of queue i. According to the ideal ECN/RED solution (Equation 2), the switch should mark arriving packets when Q_i is larger than $C_i \times RTT \times \lambda$. Thus, an arriving packet should get ECN marked when $\frac{Q_i}{C_i}$ is larger than $RTT \times \lambda$.

$\frac{Q_i}{C_i}$ is exactly the sojourn time (the amount of time a packet spends in the switch queue) for packets in queue i. We consider that a packet pkt enqueued to queue i sees Q_i worth of bits behind it. Before pkt is transmitted, queue i keeps non-empty. Hence, C_i is equal to the departure rate during this period. Therefore, pkt spends $\frac{Q_i}{C_i}$ time in queue i.

Inspired by above analysis, TCN employs sojourn time as the congestion signal. Sojourn time can be directly measured in today's switching chips (more details in §4.2). A departing packet gets ECN marked when its sojourn time is larger than the threshold T. Obviously, T can be given as:

$$T = RTT \times \lambda \tag{3}$$

In this paper, we call $RTT \times \lambda$ as the *standard marking threshold for* TCN. Compared to previous ECN/RED solution, TCN mainly has the following three benefits.

Maintain good network performance: TCN starts to notify the sources of congestion when sojourn times of packets exceed $RTT \times \lambda$. The choice of $RTT \times \lambda$ ensures that TCN can fully

utilize link capacity while delivering low latency. By contrast, per-queue ECN/RED with standard threshold suffers from poor packet latency and burst tolerance (**Remark 1**).

Support arbitrary schedulers: TCN employs the same sojourn time marking threshold for all the queues. Sojourn time directly reflects congestion states regardless of traffic demands and packet schedulers. Hence, TCN allows any scheduling policy. By contrast, MQ-ECN [11] can only handle round-robin schedulers while per-port ECN/RED can violate scheduling policies (**Remark 2**)

Easy to configure: To enable TCN, network operators only need to configure a static sojourn time threshold: $RTT \times \lambda$. As §2.1 has shown, RTT and λ can be well estimated and easily known in production datacenters. Hence, it is easy for operators to configure TCN. By contrast, it is hard, almost impossible, for operators to choose a proper measurement frequency for the ideal ECN/RED solution for generic schedulers and various traffic patterns (**Remark 3**).

4.2 Hardware Implementation Feasibility

TCN is simple and inexpensive to implement in hardware. In this section, we briefly analyze the feasibility of TCN and present a few observations; we leave a prototype implementation of a TCN-capable hardware as future work. We implement a software TCN prototype in §5, and believe realizing TCN on a switch equipped with programmable data plane technologies [12, 13] is relatively simple and cheap.

Calculating packet sojourn time: To track the packet-level sojourn time, the switching chip simply needs to attach one more metadata to each packet at the enqueue time: enqueue-time timestamp. Because draining a smallest (64B) frame takes around 13ns at 40Gbps or 5ns at 100Gbps, and a typical RTT in a datacenter is at most a few hundred us, a 2B-long timestamp with a resolution of 4 or 8ns would be sufficient ($4ns * 2^{16} \approx 262us, 8ns * 2^{16} \approx 524us$). When the packet is drained from the queue, the scheduler or the egress pipeline logic can simply take the current timestamp (*i.e.*, the dequeue-time timestamp) from the chip-local clock and perform an integer subtraction operation (*e.g.*, an unsigned subtraction with two 17b or 18b operands to cope with the case when a timestamp value wraps). Performing such an operation is trivial in hardware. The overhead of attaching a 2B-long metadata to every buffered packet introduces only a marginal complexity for two reasons. First, 2B is already a tiny fraction of an average packet size. Second, today's switching chips already attach some tens to a lower hundred bytes of metadata to each packet for statistics monitoring, forwarding, hints for scheduling, (*e.g.*, broadcast domain ID, VRF ID). Note that the latest P4 behavior model has already exposed the sojourn time as the metadata [3].

Making marking decision: TCN makes marking decisions in a completely stateless fashion. The switching chip simply needs to compare the sojourn time (a local variable) with a static threshold (a constant) to make the marking decision for a packet. Hence, the chip does not need to keep any state across packets or queues. This is in contrast to CoDel [23], which requires four state variables per queue and modifies these states in a sophisticated fashion (described below §4.3).

Marking, as opposed to dropping: To obtain the sojourn time, TCN needs to be implemented on the dequeue side of a switching chip. Since TCN only marks packets instead of dropping packets, it does not cause any bubble (*i.e.*, idle time) on the output link, which otherwise, can seriously reduce the effective throughput. By contrast, CoDel [23] requires a special speed-up or prefetching logic on the dequeue side to avoid introducing bubbles when it drops packets that are just dequeued. This speed-up logic is particularly complicated and expensive to build in high-speed switching silicon, which must ensure a full line rate (a few Tbps today) under any circumstances, irrespective of dropping dequeued packets or not.

4.3 Deeper Understanding of TCN

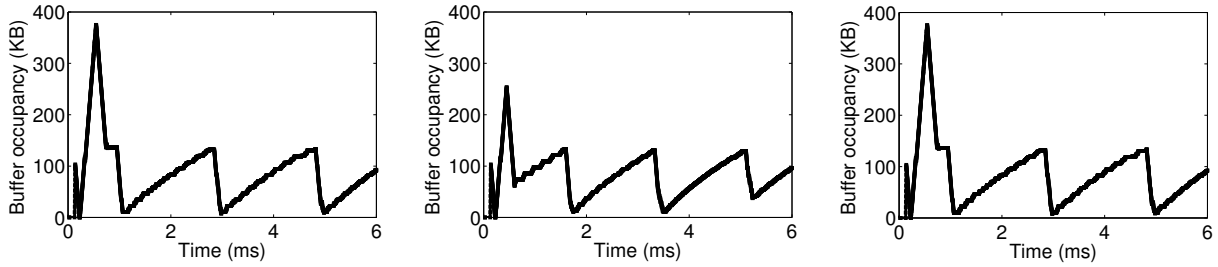
TCN vs CoDel [23]: TCN draws inspiration from CoDel [23], an AQM solution to address the bufferbloat problem on Internet. Similar to TCN, CoDel also uses sojourn time as the congestion signal. However, CoDel tracks the *minimum* sojourn time in a varying time window $interval$ to identify persistent packet delay (or bad queues). If the *minimum* sojourn time of recent $interval$ is larger than a threshold, a packet gets dropped (or ECN marked) at dequeue. Despite its potential in theory, CoDel is expensive and complex to implement in hardware as introduced in §4.2 and [25, 29].

Unlike CoDel, TCN marks packets simply based on the *instantaneous* sojourn time of each departing packet. It is important to note that such simplicity of TCN is mainly driven by the homogeneous network environments of datacenters. In datacenters, round-trip times do not change drastically and the statistical multiplexing degree of large flows is known to be very low (§2.1). Further, we have full control of end hosts. Thus, it is feasible to compute a static threshold $RTT \times \lambda$ in datacenters for instantly identifying potential excess delay and marking packets in a completely stateless fashion.

By contrast, on Internet, it is hard to figure out such a $RTT \times \lambda$ due to the significant variations of round-trip times and unknown statistical multiplexing degree. As a result, CoDel needs to conservatively identify the persistent delay by using the minimum sojourn time over a time interval against an empirical threshold.

Compared to CoDel, TCN has following two advantages:

- **Cheaper to implement in hardware:** TCN is stateless as it does not modify state in the data plane. Therefore, we expect TCN is easier to implement in hardware. By contrast, CoDel creates and modifies states in a sophisticated fashion, which increases hardware implementation complexity. For example, Sivaraman et al. [29] found that CoDel could not be implemented on their targets as it required a complex square root operation to update time window. Moreover, due to lack of control of end hosts, CoDel should support dequeue dropping for non-ECT (ECN-Capable Transport) Internet traffic, which is also expensive to implement in hardware [25].
- **Faster reaction to bursty traffic:** TCN delivers faster congestion notification since it makes marking decisions instantly rather than after a time window. So TCN can better handle bursty datacenter traffic (*e.g.*, incast [33, 34]). Our evaluation results (§6.1) confirm this.



(a) Enqueue ECN/RED (b) Dequeue ECN/RED (c) TCN

Figure 3: [Simulation] Switch buffer occupancies achieved by enqueue ECN/RED, dequeue ECN/RED and TCN.

TCN vs dequeue ECN/RED marking: Wu et al. [35] have proposed dequeue ECN/RED marking. When a packet is dequeued, it compares current queue length against the marking threshold to make the marking decision. Since dequeue ECN/RED is still based on queue length, it has all the drawbacks discussed in §3. However, compared to traditional enqueue ECN/RED marking, dequeue ECN/RED marking can deliver the congestion information earlier when queues are building up, thus providing better burst tolerance.

Similar to dequeue ECN/RED marking, TCN also marks packets at dequeue side as it relies on the sojourn time. Hence, the reader may wonder whether TCN can also accelerate the delivery of congestion information like dequeue ECN/RED. Our answer is **no**. This is because TCN leverages congestion states of *current* dequeued packets to make marking decisions, while dequeue ECN/RED indeed uses congestion states of *future* dequeued packets. Hence, when the switch buffer occupancy keeps increasing, dequeue ECN/RED marking reacts earlier than TCN as it predicts that future dequeued packets will experience congestion.

To confirm this, we simulate a 10Gbps network with 9 servers connected to a switch using ns-2 [4]. The base RTT is $100us$. We employ ECN* [35] (regular ECN-enabled TCP) as the transport protocol. We start 8 synchronized long-lived TCP flows from 8 senders to the same receiver. All the flows are classified to the same switch queue. We evaluate three ECN marking schemes: dequeue ECN/RED, enqueue ECN/RED and TCN. The marking thresholds for both ECN/RED schemes are 125KB ($100us \times 10Gbps$) in our setup. The marking threshold for TCN is $100us$.

Figure 3 shows the switch buffer occupancies versus time achieved by the three schemes. At the beginning, there is a peak buffer occupancy. This is because TCP window grows exponentially during the slow start phase before ECN takes effect. The peak value is around 375KB (3× Bandwidth Delay Product (BDP)) for both TCN and enqueue ECN/RED. In theory, enqueue ECN/RED and TCN should make the same marking decisions in this simulation. This is because when the queue capacity is fixed, we can directly convert a buffer occupancy to a corresponding sojourn time. By contrast, dequeue ECN/RED achieves a smaller peak value, which is around 250KB (2× BDP). This is because dequeue ECN/RED reacts earlier than the other two schemes, thus reducing peak buffer occupancy. After getting ECN marked, TCP flows enter congestion avoidance phase. Therefore, all the three marking schemes deliver very similar buffer occupancy variations from 0 to 125KB after the peak.

RED-like, probabilistic TCN marking: Raw ECN/RED has two marking thresholds and a maximum probability to perform a probabilistic marking (§2.1). In the previous analysis, TCN assumes only one marking threshold by using the same value for both thresholds. However, some ECN-based transports, like DCQCN [37], do require RED-like probabilistic marking to alleviate the unfairness problem.

TCN can be easily extended to perform such probabilistic marking. Similar to RED, TCN can also have two sojourn time thresholds T_{min}, T_{max} and a maximum marking probability P_{max}. If the sojourn time is $< T_{min}$, the packet does not get ECN marked. If the sojourn time is $> T_{max}$, the packet gets ECN marked. Otherwise, the packet is marked with a probability in the range of $(0, P_{max})$.

Discussion: We note that delay-based transports [14, 31] have long been studied in Internet. But they are not widely used in datacenters because in-network delays are comparable to sources of noise in the system. Recently, TIMELY [21] leverages special NIC hardware functions (*e.g.*, high-quality timestamping, hardware-generated ACKs, *etc.*) to better measure in-network delays in datacenters and uses delay gradients for congestion control. Compared to TIMELY, TCN is actually a sojourn-time based AQM to enable ECN at each hop. One goal of TCN is to make ECN-based transports, like DCTCP [6] or DCQCN [37] that are already adopted in production datacenters, work under multi-queue scenarios. It would be interesting to compare the performance of TCN-empowered DCQCN with TIMELY under multi-queue scenarios, which is part of our future work.

5. SOFTWARE PROTOTYPE

We use a server with multiple Gigabit Network Interface Cards (NICs) to emulate the switch. As a software prototype, we implement TCN, CoDel, MQ-ECN and per-queue/port ECN/RED and in a Linux queueing discipline (`qdisc`) kernel module running at the server-emulated switch. Consequently, packets are completely processed in kernel space without introducing data copy or context switch overhead between user and kernel space.

In summary, our `qdisc` kernel module has five components: packet classifier, enqueue ECN marking, packet scheduler, rate limiter and dequeue ECN marking (based on packet processing order). We now describe them in detail.

Packet Classifier: The `qdisc` kernel module maintains multiple FIFO transmission queues for each NIC. When the kernel module receives a packet from IP layer, it classifies the packet based on the Differentiated Services Code Point (DSCP) field. Then this packet is enqueued to the corresponding queue.

Enqueue ECN Marking: Upon an arrival packet is enqueued, we start to perform enqueue ECN marking. Per-queue/port ECN/RED directly compare buffer occupancy against the static threshold to make the marking decision. MQ-ECN needs to calculate the dynamic threshold of the corresponding queue and then make the marking decision. By contrast, TCN and CoDel use `tstamp` field of `sk_buff` to record enqueue timestamp here, leaving the marking decision on the dequeue side.

Packet Schedulers: We implement 4 common packet scheduling algorithms: WFQ, DWRR, SP/WFQ, and SP/DWRR.

- **WFQ:** To implement WFQ, we maintain a *virtual time* for the head packet of each queue. The WFQ scheduler chooses the head packet with the smallest virtual time to transmit.
- **DWRR:** The DWRR scheduler maintains a linked list to store all the active queues. When a packet arrives at an empty queue, this queue is inserted into the tail of the linked list. The DWRR scheduler always serves the head queue of the linked list. If a queue just finishes its service but still has packets, it is attached to the tail of the linked list again. Furthermore, to implement MQ-ECN, we maintain a timestamp for each queue to track round time.
- **SP/WFQ and SP/DWRR:** Both schedulers reserve several strict higher priority queues while leaving all the WFQ/DWRR queues on the lowest priority. They schedule packets as follows: they first try to schedule packets from strict higher priority queues based on priorities; only when all the higher priority queues are empty, they start to schedule packets on the lowest priority according to WFQ/DWRR.

Rate Limiter: A packet dequeued by `qdisc` will further go through NIC driver and NIC hardware before it is pushed to the wire. If we dequeue packets from `qdisc` without any rate control, many packets can still get queued on NIC driver TX ring buffer and NIC hardware. Consequently, the buffer occupancy monitored by `qdisc` can be much smaller than the real value of a switch port, thus degrading the accuracy of ECN marking and packet drop decisions. Such issue was also identified by many previous works [11, 36].

To address above issue, we use a Token-Bucket rate limiter to shape outgoing traffic from `qdisc` at 99.5% of the NIC capacity (995Mbps). To minimize traffic burstiness, we use a small bucket size of ∼1.67MTU (2.5KB) which can achieve 99% link utilization. In this way, we can minimize undesirable buffering in other places and make the buffer occupancy of `qdisc` better reflect the real buffer occupancy of a switch port.

Dequeue ECN Marking: Eventually, TCN and CoDel mark dequeued packets when they are about to be delivered to NIC driver. Both schemes uses current system time minus the enqueued timestamp to obtain the sojourn time and makes the marking decision. Our CoDel implementation closely tracks the Linux source code.

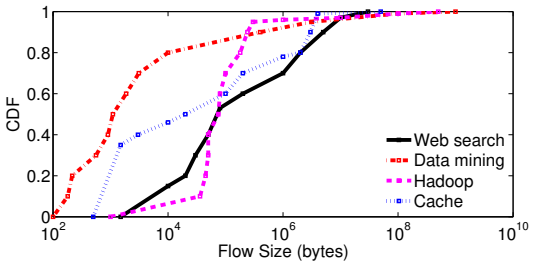


Figure 4: Traffic distributions for evaluation.

6. EVALUATION

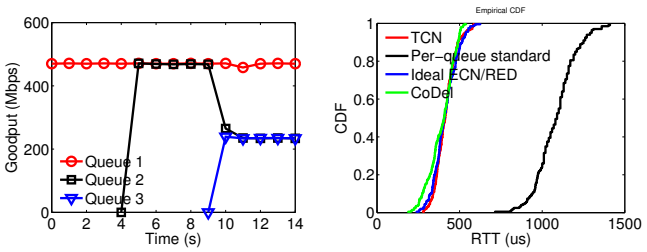
In this section, we use a combination of testbed experiments and ns-2 simulations to answer the following three questions:

- **How does TCN perform in practice?** In the static flow experiment (§6.1.1), we find that TCN can achieve high throughput and low latency while strictly preserving the scheduling policies. With realistic workloads (§6.1.2 and §6.1.3), we find that TCN can maintain its good performance with various scheduling policies (including WFQ and SP/WFQ that MQ-ECN cannot support). For example, compared to per-queue ECN with standard threshold, TCN (with SP/DWRR) reduces the average and 99th percentile completion times for small flows by up to 82.8% and 95.3%, respectively. Compared to CoDel, TCN (with SP/WFQ) achieves up to 84% lower 99th percentile completion time for small flows.
- **Does TCN perform well in large datacenters?** In large-scale ns-2 simulations (§6.2.1), we find that TCN generally achieves the best performance in large multi-switch topologies. For example, compared to per-queue ECN with standard threshold, TCN achieves up to 94.3% lower 99th percentile completion time for small flows.
- **How robust is TCN to network settings?** Using a series of targeted simulations, we show that TCN is robust to transport protocol and the number of queues (§6.2.2) .

Schemes compared: We mainly evaluate the performance of four ECN solutions: TCN, CoDel [23], MQ-ECN [11] and per-queue ECN/RED with standard threshold (current practice §3.2). In the simple static flow experiment (§6.1.1), we also consider the ideal ECN/RED solution (Equation 2) by assuming the prior knowledge of queue capacities. We exclude per-port ECN/RED as it can violate scheduling policies.

Among above solutions, MQ-ECN and per-queue ECN/RED with standard threshold are queue-length based solutions. Instead using the static standard threshold, MQ-ECN dynamically adjusts the queue-length threshold based on the estimated queue capacity. For MQ-ECN, we set β to 0.75 and T_{idle} to the transmission time of a MTU as the paper [11] suggests. Note that MQ-ECN can only support pure round-robin schedulers.

CoDel and TCN are sojourn-time based solutions and mark packets on the dequeue side. To achieve a fair comparison, we



(a) Aggregate goodputs (b) RTT

Figure 5: [Testbed] (a) Aggregate goodputs of three queues achieved by TCN. (b) RTT distributions of queue 3.

configure CoDel to only mark packets (rather than drop) in our evaluation.

Benchmark traffic: We use four traffic distributions based on measurement from production datacenters (Figure 4): a web search workload [6], a data mining workload [17], a Hadoop workload [27] and a cache workload [27]. In general, all the workloads are heavy-tailed. Among them, the web search workload is less skewed: $\sim$60% of its all bytes are from flows smaller than 10MB. Hence, the web search workload is more difficult to handle as it is likely that multiple flows are concurrently active on the same link. Therefore, we focus on the more challenging web search workload in testbed experiments while using all the four in large-scale simulations.

Performance metric: We use flow completion time (FCT) as the performance metric. We consider average FCT across all flows, small flows and large flows. To evaluate tail latency, we also give the 99th percentile FCT of small flows. For clear comparison, we normalize final FCT results to the values achieved by TCN.

6.1 Testbed Experiments

Testbed setup: We built a small-scale testbed that consists of 9 servers connected to a 9-port server-emulated switch. Each server is a Dell PowerEdge R320 with a 4-core Intel E5-1410 2.8GHz CPU, 8G memory, a 500GB hard disk. All the servers run Linux kernel 3.18.11. We enable DCTCP as the transport protocol and set TCP RTOmin to 10ms as many proposals suggest [6, 33]. The server-emulated switch has 10 Broadcom BCM5719 NetXtreme Gigabit NICs in total where one is reserved for control access. To better emulate real switches, we disable NIC offloading techniques at the switch to avoid large segments. Each switch port has a 96KB buffer which is completely shared by all the queues in a first-in-first-serve basis. Given the base RTT is around $250us$, the standard ECN marking threshold is 32KB for ECN/RED and $256us$ for TCN. For Codel, we experimentally determine its best setting ($target = 51.2us, interval = 1024us$) since the recommendation setting ($5ms$ and $100ms$) in [23] is for Internet.

6.1.1 Static Flow Experiment

We begin with a simple experiment to show that TCN can achieve good network performance while preserving the scheduling policies. At the switch, we configure SP/WFQ with three queues: queue 1 has a strict higher priority while queue 2 and 3 have equal weights in the lowest priority. We use three servers as the sender and one as the receiver. We first start a 500Mbps TCP flow from sender 1, then start a TCP flow from sender 2, and finally, start 4 TCP flows from sender 3. The packets from sender 1, 2 and 3 are classified to queue 1, 2 and 3, respectively. Based on the SP/WFQ scheduling policies, the final aggregate throughput of queue 1, 2 and 3 should be 500Mbps, 250Mbps, and 250Mbps, respectively.

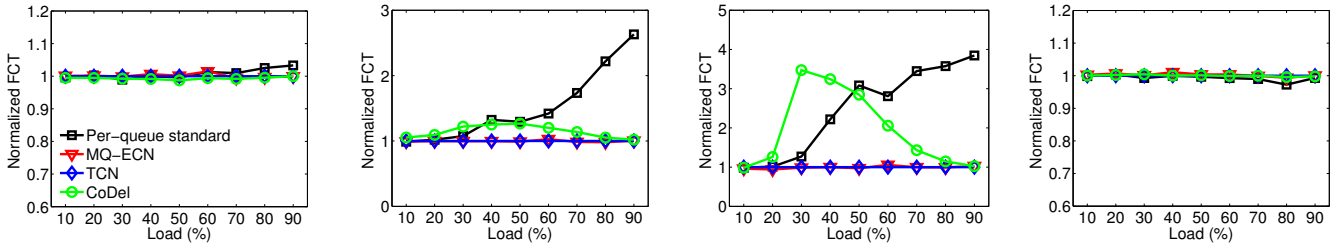
Figure 5(a) gives per-queue aggregate goodputs versus time achieved by TCN. During the experiment, queue 1 maintains its $\sim$470Mbps goodput (goodput is slightly smaller than throughput due to the IP/TCP header overhead), matching its strict higher priority at the switch. When queue 3 becomes active, queue 2 and queue 3 fairly share the remaining capacity regardless of the numbers of flows. We also use ns-2 simulation to reproduce above experiment and find that TCN can achieve similar convergence time as per-queue ECN with standard threshold. This suggests that TCN can strictly preserve the scheduling policies (a combination of WFQ and SP) while achieving high throughput.

We also measure RTT of queue 3 using `ping` packets. Figure 5(b) gives RTT distributions achieved by TCN, per-queue ECN with standard threshold, the ideal ECN/RED (equation 2) and CoDel. Note that for the ideal ECN/RED, the marking thresholds of queue 2 and 3 are both 8KB ($250Mbps \div 1000Mbps \times 32KB$). TCN provides similar latency as the ideal ECN/RED and CoDel while significantly outperforming per-queue ECN/RED with standard threshold. Compared to per-queue ECN/RED with standard threshold, TCN achieves 61.7% (1084us to 415us) and 58.4% (1400us to 582us) lower RTT in average and the 99th percentile. If we exclude the base RTT (250us), TCN indeed achieves more than 4$\times$ improvement in queueing delay. This suggests that TCN can also deliver low latency.

6.1.2 Inter-Service Traffic Isolation

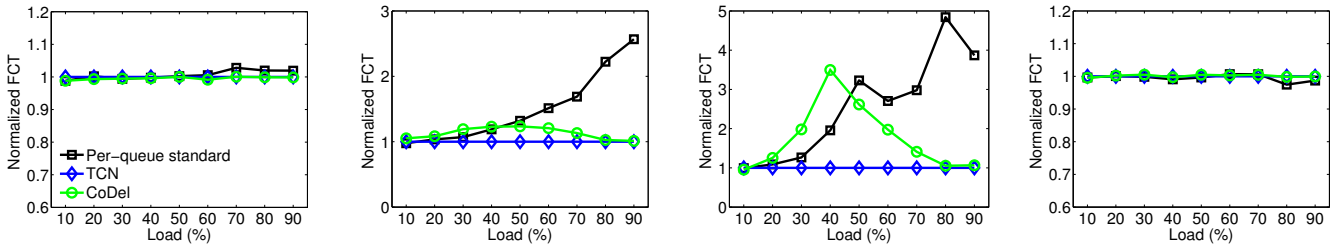
In this experiment, we develop a client/server application to generate realistic traffic according to the web search workload (Figure 4) and measure the FCT on the application layer. The client application, running on 1 server, initially opens 5 persistent TCP connections to each of the rest 8 servers. During the experiment, the client application generates requests based on a Poisson process through available connections (or create new connections if no available connection) to the other servers to fetch data. The server applications, running on the rest 8 servers, respond with requested data. Hence, a TCP connection can carry multiple flows (messages). At the switch, we configure DWRR and WFQ with 4 equal weight/quantum (1.5KB) queues. To emulate inter-service traffic isolation, the server application uses `setsockopt` to set DSCP values for outgoing traffic, thus mapping different flows to different queues. Note that a flow is randomly mapped to one of the 4 service queues. We vary the network load from 10% to 90% and run 5000 flows for each setting.

Figure 6 and 7 show the FCT results across all flows (a), small flows (0,100KB] (b,c), and large flows (10MB,∞) (d) respectively. We omit the results for medium flows due to space



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

Figure 6: [Testbed] Inter-service traffic isolation with DWRR (4 queues) and DCTCP: FCT statistics.



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

Figure 7: [Testbed] Inter-service traffic isolation with WFQ (4 queues) and DCTCP: FCT statistics.

limitation. Note that we exclude MQ-ECN from Figure 7 as it does not support WFQ. Based on the results, we make the following three observations:

Overall: Generally, all the schemes achieve similar overall average FCTs. This is expected. In a long-tailed traffic distribution, the majority of bytes are from a small number of throughput-sensitive large flows. Therefore, the overall average FCT is mainly determined by throughput. Since all the schemes can fully the link capacity, they achieve similar overall average FCT results.

Despite this, we observe that TCN, CoDel and MQ-ECN (for DWRR only) slightly outperform per-queue ECN with standard threshold at high loads ($\geq 70\%$). For example, compared to per-queue ECN with standard threshold, TCN reduces the overall average FCT by up to 2.5% and 2.7% for DWRR and WFQ, respectively. This is because, when networks run at high loads, per-queue ECN with standard threshold suffers from frequent packet drops and TCP timeouts as multiple queues are likely to be active simultaneously. By contrast, TCN can still keep low buffer occupancies to minimize such impact.

Small flows: For small flows, TCN performs similarly as MQ-ECN for DWRR while maintaining good performance for WFQ. Compared to per-queue ECN with standard threshold, the average FCT for the small flows with TCN is up to 61.4% (9679us to 3733us) lower for DWRR and 61.1% (9529us to 3711us) lower for WFQ. As expected, TCN achieves larger performance improvement at the 99th percentile: up to 73.3% (DWRR) and 79.3% (WFQ) lower 99th percentile FCT compared to per-queue ECN with standard threshold. Compared to CoDel, TCN achieves up to 71.2% (11370us to 3273us) and 71.4% (12323us to 3525us) lower 99th percentile FCT for DWRR and WFQ, respectively. The reason is that CoDel introduces many packet drops and TCP timeouts ($\geq$10ms) as it cannot quickly react to bursty traffic. All above suggest that TCN can provide low latency and good burst tolerance, thus minimizing the FCT for small flows.

Large flows: TCN also maintains good performance for large flows. Compared to per-queue ECN with standard threshold, the average FCT for large flows with TCN is within 2.8% for DWRR and 2.6% for WFQ. This shows that TCN can achieve high throughput in highly dynamic workloads by using $RTT \times \lambda$ as the sojourn time marking threshold.

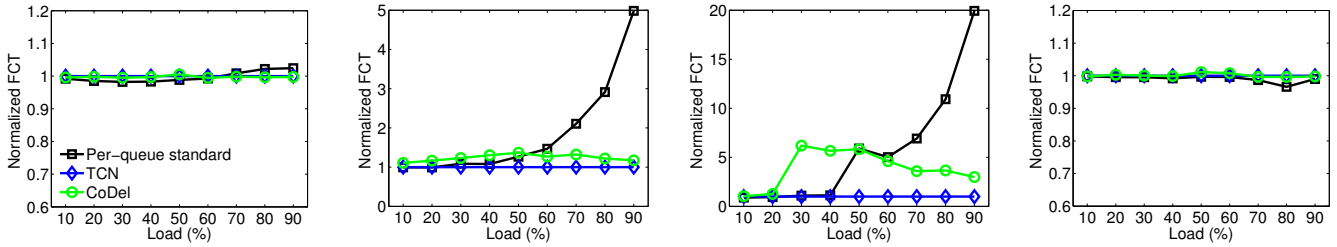
6.1.3 *Traffic Prioritization*

In this experiment, we use the same setting as §6.1.2 except for two changes. First, at the switch, we configure SP/DWRR and SP/WFQ by adding a new strict higher priority queue to prioritize the delivery of latency-sensitive small flows. Second, at end host, we install a Linux `Netfilter` kernel module to perform a two-priority PIAS [10] flow scheduling. For each flow (message), the first 100K bytes are classified into the shared strict higher priority queue while the rest bytes are classified into their dedicated service queues in the low priority[5]. In this way, small flows are prioritized over large flows in general.

Figure 8 and 9 show the FCT results with SP/DWRR and SP/WFQ. Here we exclude MQ-ECN [11] as it does not support SP in general. Again, we breakdown FCT across different flow sizes, and make the following two observations:

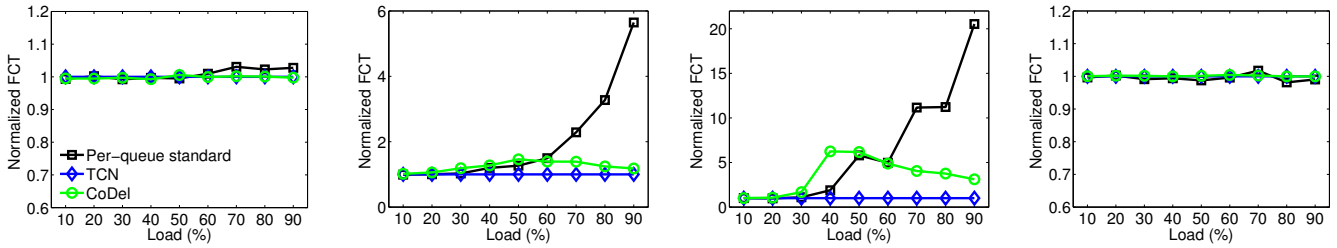
Overall and large flows: In general, TCN and CoDel achieve

[5]Our kernel module can identify flow (message) boundaries over a persistent connection by tracking `tcp_sendmsg` called times. When the time gap between two consecutive calls over the same connection is larger than a threshold (500us in our experiments) and TCP send buffer is empty, we regard this as the beginning of a new flow.



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

Figure 8: [Testbed] Traffic prioritization with SP (1 queue) / DWRR (4 queues) and DCTCP: FCT statistics.



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

Figure 9: [Testbed] Traffic prioritization with SP (1 queue) / WFQ (4 queues) and DCTCP: FCT statistics.

similar performance as per-queue ECN with standard threshold while slightly outperforming it at high loads. TCN achieves up to 3.7% and 3% lower overall average FCT, for SP/DWRR and SP/WFQ, respectively. This is mainly because TCN can greatly reduce packet drops and TCP timeouts at high loads by maintaining low buffer occupancies.

TCN also maintains good performance for large flows. Compared to per-queue ECN with standard threshold, TCN's performance is within 2.1% for SP/DWRR and within 1.9% for SP/WFQ. This suggests that TCN can efficiently utilize the link capacity in highly dynamic datacenter workloads.

Small flows: Compared to the previous service isolation experiment (§6.1.2), all ECN marking schemes indeed achieve much lower FCT for small flows since small flows are all finished in the high priority queue[6]. For example, at 90% load, TCN with PIAS (SP/DWRR) achieves 71.3% ($3733us$ to $1073us$) lower average FCT for small flows compared to TCN without PIAS.

Even with flow scheduling, TCN still significantly outperforms per-queue ECN with standard threshold and CoDel. For example, TCN (SP/DWRR) reduces the FCT for small flows by up to 82.8% ($6222us$ to $1073us$) and 95.3% ($82658us$ to $3860us$) in average and the 99th percentile, respectively. Given all small flows are finished in the high priority queue, why can TCN still gain such a large performance improvement? This is because packets in the high priority queue can still get dropped under the buffer pressure from other low priority queues according to shared buffer policy. TCN can greatly mitigate this impact by maintaining low buffer occupancies. By contrast, per-queue ECN with standard threshold causes frequent packet drops and TCP timeouts ($\geq$10ms) at high loads, leading the tail FCT to as high as tens of milliseconds. Compared to CoDel, TCN (SP/WFQ) achieves up to 84% lower 99th percentile FCT for small flows. This further confirms that TCN can quickly react to traffic burstiness with instantaneous marking.

[6]Due to space limitation, we omit raw FCT results here.

6.2 Large-Scale NS-2 Simulations

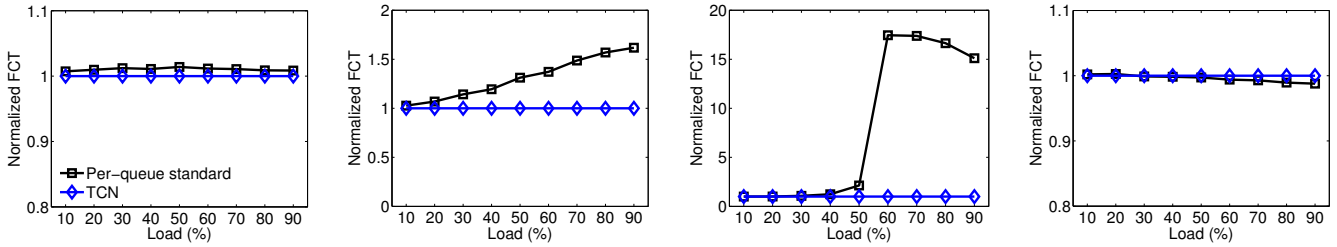
In this section, we use ns-2 [4] simulator to evaluate the performance of TCN in large-scale multi-hop datacenter networks. Similar to the testbed experiments, we have evaluated both inter-service traffic isolation and traffic prioritization in simulations. In the interest of space, we only show results for the more complex case of traffic prioritization here.

Topology: We simulate a 144-host leaf-spine topology with 12 leaf (ToR) switches and 12 spine (Core) switches. Each leaf switch has 12 10Gbps links to hosts and 12 10Gbps links to spines, forming a non-blocking fabric. We employ ECMP for load balancing. The base RTT across the spine (4 hops) is $85.2us$ which $80us$ is spent at end hosts.

Transport: We use DCTCP [6] by default. The initial window is 16 packets. We set both minimum value and the initial value of TCP RTO to 5ms. Note that initial TCP RTO is 3 seconds by default which is too large for datacenter networks.

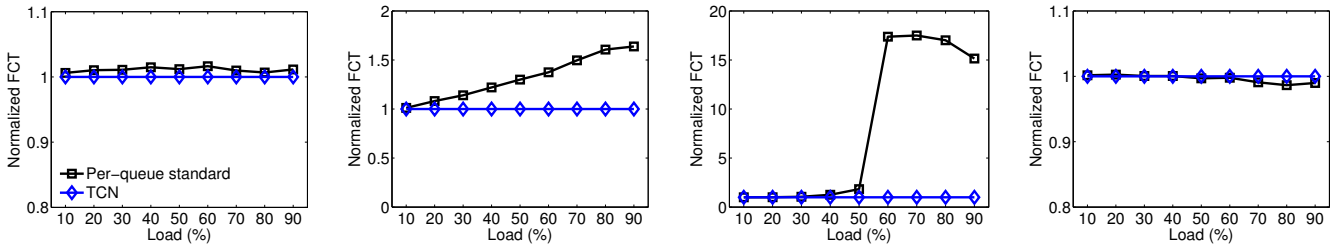
Switch: Each switch port has a 300KB buffer which is completely shared by 8 queues in a first-in-first-serve basis. The standard marking threshold is 65 packets for ECN/RED and $78us$ for TCN. We configure SP/WFQ and SP/DWRR where one queue has a strict higher priority over the rest 7 equal weight/quantum (1.5KB) WFQ/DWRR queues.

Workloads: Since we have 144 hosts, there are 144×143 communication pairs in total. We evenly classify these pairs into 7 services. Different services use different traffic distributions in Figure 4. Each service has its own dedicated service queue while sharing the same high priority queue at the switch. Same



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

Figure 10: [Simulation] Traffic prioritization with SP (1 queue) / DWRR (7 queues) and DCTCP: FCT statistics.



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

Figure 11: [Simulation] Traffic prioritization with SP (1 queue) / WFQ (7 queues) and DCTCP: FCT statistics.

as §6.1.3, we perform a two-priority PIAS [10] flow scheduling. The first 100K bytes of a flow are classified to the shared high priority queue while the rest bytes are classified to their dedicated service queues in the low priority. All simulations last for 50000 flows.

6.2.1 *Overall Performance*

Figure 10 and 11 show the FCT results across different flow sizes. We make the following two observations.

Overall and large flows: TCN slightly outperforms per-queue ECN with standard threshold across all flows in all scenarios. It achieves ∼0.72-1.38% and ∼0.61-1.62% lower overall average FCT for SP/DWRR and SP/WFQ, respectively. TCN also achieves very good performance for large flows. TCN's performance is within 1.2% of per-queue ECN with standard threshold for SP/DWRR and within 1.37% for SP/WFQ. This suggests that TCN can achieve high throughput in large-scale multi-hop datacenter networks.

Small flows: Compared to per-queue ECN with standard threshold, the average FCT for the small flows with TCN is up to 38.2% lower for SP/DWRR and up to 38.8% lower for SP/WFQ. As expected, TCN achieves larger improvements at the 99th percentile: it reduces the 99th percentile FCT by up to 94.3% for both schedulers. With PIAS, the performance for small flows is mainly determined by the burst tolerance of the ECN marking scheme. For example, at 90% load, per-queue ECN with standard threshold (with SP/DWRR) causes 589 TCP timeouts for small flows, resulting in a very large 99th percentile FCT (5390us). This indicates that more than 1% small flows experience at least one TCP timeout. By contrast, TCN only experiences 46 TCP timeouts for small flows in the same scenario, thus achieving a very small 99th percentile FCT (357us).

6.2.2 TCN *Deep Dive*

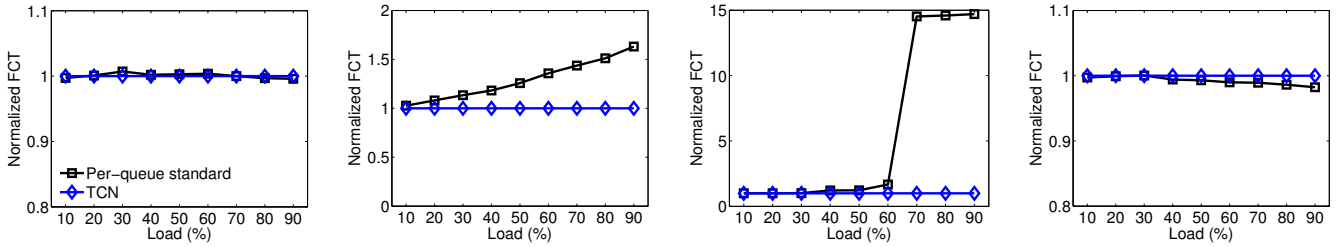
In this section, we evaluate TCN's robustness to network settings through a series of targeted simulations.

Impact of transport protocol: In addition to DCTCP [6], there are many other ECN-based transport protocols, such as DCQCN [37] and ECN* [35]. Among them, ECN* may be the most challenging one as it simply cuts window by half in the presence of ECN without any smoothing. Consequently, ECN* is more sensitive to premature ECN markings than DCTCP. A lower ECN marking threshold can severely degrade the throughput of ECN*. For example, with zero buffering, DCTCP can still maintain 94% throughput in theory while ECN* can only deliver 75% throughput [7].

We run simulations using ECN* instead of DCTCP. We set the standard ECN marking threshold to 84 packets for ECN/RED and 101us for TCN. Figure 12 gives the FCT with ECN*. We find that, compared to per-queue ECN with standard threshold, TCN achieves similar performance for large flows while showing large improvements for small flows. Even with very challenging ECN*, TCN's performance is still within 1.8% of per-queue ECN with standard threshold for large flows. This indicates that TCN is robust to transport protocol.

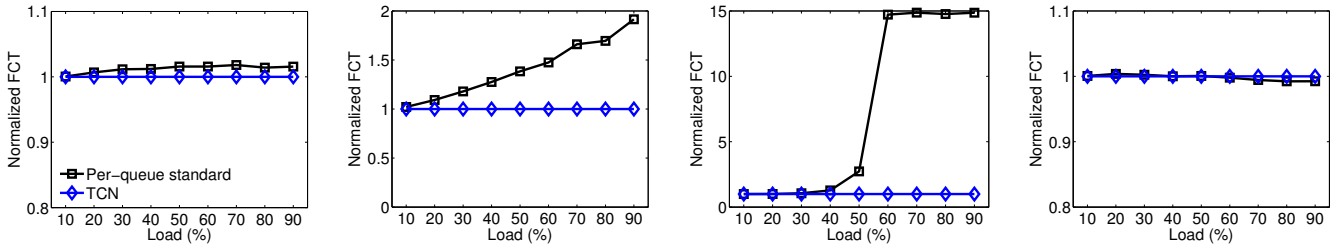
Impact of the number of queues: In the future, switching chips may provide more and more queues. To evaluate TCN's robustness to the number of queues, we increase the number of switch queues to 32 which 31 queues with equal quantum run in the low priority. To create more challenges, we still employ ECN* as the transport protocol.

Figure 13 gives the FCT results with 32 queues and ECN*. In general, TCN still outperforms per-queue ECN with standard threshold. By comparing Figure 13 and 12, we observe that TCN shows larger performance improvements over per-queue



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

Figure 12: [Simulation] Traffic prioritization with SP (1 queue) / DWRR (7 queues) and ECN*: FCT statistics.



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

Figure 13: [Simulation] Traffic prioritization with SP (1 queue) / DWRR (31 queues) and ECN*: FCT statistics.

ECN with standard threshold for small flows with 32 queues. For example, at 90% load, TCN shows 38.7% and 47.8% lower average FCT for small flows with 8 queues and 32 queues, respectively. This is because per-queue ECN with standard threshold causes more packet drops and TCP timeouts with more queues. At 90% load, per-queue ECN with standard threshold causes 2469 and 4478 TCP timeouts with 8 queues and 32 queues, respectively. By contrast, TCN does not have such performance degradation. This suggests that TCN is robust to the number of queues.

7. RELATED WORK

Generally, ECN-related works in datacenters are vast [6, 11, 26, 32, 35, 37]. But the closest related one to TCN is MQ-ECN [11], which also tries to enable ECN for multiple queues. However, MQ-ECN only works for round-robin schedulers, whereas TCN enables ECN for arbitrary packet schedulers. More fundamentally, MQ-ECN is still designed along the line of ECN/RED principle that uses queue-length as congestion signal, thus sharing all the problems we discussed in §3. By contrast, TCN completely dismisses queue-length based ECN/RED after identifying its limitations, and instead uses packet sojourn-time as congestion signal to activate ECN.

CoDel [23] is the state-of-the-art sojourn-time based AQM for Internet. Compared to CoDel, TCN can better handle bursty datacenter workloads and be easier to implement in hardware due to its stateless fashion.

Recently, Mittal et al. [20] have proposed an edge-based AQM solution, in which they use one-way cumulative queuing delay as the congestion signal and only mark ECN at the network edge. In comparison, TCN relies on packet sojourn time at each hop.

8. CONCLUSION

With the goal of supporting ECN over generic packet schedulers, this paper has delivered two points: 1) queue-length based ECN/RED is fundamentally hard to achieve it; 2) sojourn-time based TCN is a simple yet elegant solution to it. We have spent extensive experimental efforts demonstrating not only the validity of TCN, but also the invalidity of ECN/RED.

Acknowledgements

This work is supported in part by the Hong Kong RGC ECS-26200014, GRF-16203715, GRF-613113, CRF-C703615G, and the China 973 Program No.2014CB340303. Wei is also supported by a Microsoft Research Asia PhD Fellowship. We thank our shepherd, Colin Dixon, and the anonymous reviewers for their valuable feedback.

9. REFERENCES

[1] "DCTCP in Linux kernel 3.18," http://kernelnewbies.org/Linux_3.18.

[2] "DCTCP in Windows Server 2012," http://technet.microsoft.com/en-us/library/hh997028.aspx.

[3] "P4 Behavior Model," https://github.com/p4lang/behavioral-model/blob/master/targets/simple_switch/tests/testdata/queueing.p4.

[4] "The Network Simulator NS-2," http://www.isi.edu/nsnam/ns/.

[5] "WRED Explicit Congestion Notification on Cisco switches," http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_conavd/configuration/xe-3s/asr1000/qos-conavd-xe-3s-asr1000-book/qos-conavd-wred-ecn.pdf.

[6] 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*.

[7] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: stability, convergence, and fairness," in *SIGMETRICS 2011*.

[8] 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*.

[9] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM 2013*.

[10] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-Agnostic Flow Scheduling for Commodity Data Centers," in *NSDI 2015*.

[11] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in Multi-Service Multi-Queue Data Centers," in *NSDI 2016*.

[12] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, 2014.

[13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN," in *SIGCOMM 2013*.

[14] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE J.Sel. A. Commun.*, pp. 1465–1480.

[15] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, pp. 107–113, 2008.

[16] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, pp. 397–413.

[17] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *SIGCOMM 2009*.

[18] 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*.

[19] G. Judd, "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter," in *NSDI 2015*.

[20] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal Packet Scheduling," in *NSDI 2016*.

[21] 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*.

[22] 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*.

[23] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[24] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," in *NSDI 2013*.

[25] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *HPSR 2013*.

[26] K. Ramakrishnan, S. Floyd, D. Black *et al.*, "RFC 3168: The addition of explicit congestion notification (ECN) to IP," 2001.

[27] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in *SIGCOMM 2015*.

[28] 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*.

[29] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet Transactions: High-level Programming for Line-Rate Switches," in *SIGCOMM 2016*.

[30] A. Sivaraman, S. Subramanian, A. Agrawal, S. Chole, S.-T. Chuang, T. Edsall, M. Alizadeh, S. Katti, N. McKeown, and H. Balakrishnan, "Programmable Packet Scheduling," in *SIGCOMM 2016*.

[31] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *INFOCOM 2006*.

[32] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *SIGCOMM 2012*.

[33] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *SIGCOMM 2009*.

[34] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in data center networks," in *CoNEXT 2010*.

[35] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *CoNEXT 2012*.

[36] 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*.

[37] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion Control for Large-Scale RDMA Deployments," in *SIGCOMM 2015*.